An Industry Experience on Dynamic Handling of Multiple Database Types for Data Integration Tool

Badrul Affandy Bin Ahmad Latfi, Anbarasan Kodandaraman, Lee Chee Kiam, and Thong Tong Khin
Software Development Lab
MIMOS Bhd
Kuala Lumpur, Malaysia
e-mails: {badrul.affandy, anbarasan.raman, ck.lee, thong.tkin}@mimos.my

Abstract— **This paper presents the industry experience of implementing a Structured Query Language (SQL) Query Builder component and its interface supporting multiple database types in web-based Data Integration Tool (DIT), which performs Extract-Transform-Load (ETL) and Data Cleansing. Due to the dynamic nature of database connectivity for different types of source and target tables, the usage of direct SQL statement is preferred over Object Relationship Management (ORM). The common problem while handling multiple different databases is that it requires database specific queries for SQL functions, like create table, insert data or query specific range of rows for each type of database. Hence, we developed the SQL Query Builder (QB), which generates specific SQL statements for specific syntax. By standardizing the usage of delimited identifier with American National Standards Institute (ANSI) mode, and schema hierarchy, the QB's common interface will enable to dynamically handle multiple types of databases. With the QB, we created ETL and data cleansing applications compatible with multiple database types.**

Keywords-Data integration; SQL; multiple database type.

I. INTRODUCTION

A metadata-intensive application, such as data integration that performs Extract-Transform-Load (ETL) and data cleansing works on low level Application Programming Interface (API) to produce actions on source and target database. Logical data mapping needs to be done at early stage of ETL to ensure ETL produce quality data [1]. ETL operation could be done through effective usage of SQL to map and transform data [2]. Manual tables mapping with the help of user interface is a slow process [3]; thus, we use the automated mapping of source and target tables with additional supports of multiple database types. The data integration tool that we developed is based on server centric solution, which can be deployed on powerful servers.

We developed a SQL Query Builder component, also referred as Query Builder (QB) in this document, for supporting multiple database connections, providing services to source and target database, and the system resource itself. QB has three benefits: (1) creates SQL based on source/target tables, (2) eases of applying system identification to manage source and target tables, and (3) enforces delimited identifiers that are generic for all database types. The output of QB is SQL statement produced specifically for source or target table based on database type. QB is the main component of Data Integration Tool (DIT). Since the DIT is a web-based solution, whereby table entities classes are loaded during system startup, it is essential for the DIT to execute SQL query to multiple database types dynamically, which cannot be performed by traditional solution, such as Object Relational Management (ORM).

Section II elaborates problems encountered and solutions. Section III covers the QB architecture and design. Section IV presents the results, and finally, Section V concludes QB benefit and future enhancement.

II. PROBLEMS ENCOUNTERED AND SOLUTIONS

A. *Database Mapping*

Systems or platforms that require accessing multiple database types had been presences many years back. These systems connect database and access its tables through Java connector with configured object mappings via ORM programming tool. ORM is a powerful tool that allows Java objects interacts with relational database. The object/relational metadata are configured prior to Java object instantiations. The object instantiation for all mappings normally happens during application startup, after that application could perform standard operations like create, update, delete and read. If the XML configuration of the ORM is not correct, exceptions occur and the application loading is aborted. Open-source tools, such as Hibernate [4], MyBatis [5], provide object relational mapping with database tables.

For ETL, the source and target databases could be of different database types. The source and target databases are configured via a configuration screen. User may add or remove the configuration as required. User may link up source and target databases to perform ETL process. Once tables are connected, ETL processes extract, transform and then load data into data warehouse. The data warehouse is the primary environment of ETL process. ORM is not fitting the requirement due to the tool need to dynamically connect to multiple data sources (with any number of columns of different type) and data structure during runtime. We use database specific SQL over ORM. Each ETL process would require a set of dynamic SQL scripts to be generated during runtime for its operations (table creation, alteration and selection).

B. *Data Integration Tools*

There are a few open source data integration tools available in the IT industry, such as Pentaho Data Integration (Kettle) [6] and Talend Open Studio for Data Integration (TOS) [7]. These tools run job flow on client desktop or on remote server through deployment. The job flow typically bounds specific database tables, so that each ETL job is unique. In our application, an ETL process runs on generic ETL job. A new ETL process with database configurations could be added without application redeployment. Kettle and TOS jobs perform stream-based process, which is per record basis. Our application could not make use these open source tool platforms because it is required to runs multi-threaded jobs with bulk data stored in multiple database types. It is also required to connect with Graphic Processing Unit (GPU) system for data cleansing operation [8]. Our application also has other proprietary data cleansing features, which runs in web-based application.

C. *Database syntax*

There are many commercial RDBMS in the market where we use three types, namely, Oracle, MySQL and PostgreSQL. We use SQL language to connect with these database types be it at data processing level or user interface level. Problems encountered by the development team of DIT are listed below. Other problems, such as built-in function and dialect, are not considered as they are out of scope.

- **Quoted Identifier**. Many sources recommended avoiding quoted identifier, as it destroys portability of the code and invites poorly constructed names [9]. However, this may not be true if we were to extract data from multiple database types. Using quoted identifier with ANSI [10] setting would improve code maintainability across different database vendors [11][12][13].
- **Schema hierarchy**. The schema name is always supplied in order to avoid confusion especially in PostgreSQL. This problem is re-solved using quoted identifier for schema, so that it works across multiple database types.
- **Record number and limit**. Record limit built-in function normally receive two arguments, namely, start number and number of records. According to MYSQL manual, the LIMIT command in MYSQL limits the return row [14]. For PostgreSQL, the offset value specifies the number of row to skip. For Oracle, it has top N-query processing with ROWNUM [15]. The query has inline view of the target table with full query. The returned records are bigger than start number but less than record number. The next batch of records will be received by incrementing the start and record numbers.
- **Upsert**. Upsert is a function to update and insert data (if data not found in target table). There is no standard command of upsert among multiple database types. Therefore, we develop small SQL functions to build an upsert command for each type of database.
- **Type casting** for Numeric Type. Usually, the data types are same across database types. But, in PostgreSQL, NUMERIC type can save float or integer, this data type has to be considered carefully during data conversion, otherwise SQL exception will occur. Casting numeric data type to float re-solves the issue.
- **Auto increment**. Creation of auto increment constraint in Oracle is not straight forward as in MySQL (AUTO INCREMENT) and PostgreSQL (BIGSERIAL). We have to create a new sequence with id and other details like start and increment value. A trigger with unique id is also created and the "NEXTVAL" of sequence is called on the column where the value has to be incremented.
- **Connection pooling on dynamic connections**. In an ETL application, it is essential to connect to multiple databases and tables in a dynamic way. Since connection pooling is used, pooled connections have to be reused for connecting to the same data source. As we are using DBCP of Apache [16], the pooled connections for the dynamically created BasicDataSource have to be reused to avoid connection leak or to open too many connections. Hence, we need to pool the Apache's DBCP connections into a map. We can make sure that only one connection pool is created for one data source and is pooled in our map. When requested for an existing connection, the map holding the object of the current data source is returned.
- **Auto generates column value in Oracle.** There might be several instances where the auto generated id upon inserting a row into table is required. The prepared statement or statement has the facility to retrieve the generated value. After setting the values, the statement is executed and the result set hold the generated values. For Oracle, since the auto generated values are achieved by using sequence and trigger, the column which holds the auto-generated key is not considered as generated column and hence, it is not returned. It holds the ROWID of the inserted row. This ROWID is used to retrieve the generated key by a separate call, as shown in Figure 1.

```
StringBuilder que = new StringBuild-
er();
tempStat= conn.createStatement();
que.append("SELECT generatedColumn-
Name FROM "+ tablename+("
    WHERE rowid ='"+rowValue+"'");
```

Figure 1. Auto generate key for Oracle

Figure 1 illustrates way of creating a SQL query string to get the auto generate key for Oracle. Executing the SQL query string with Oracle database generated a result set and from the result set, the generated key value is obtained.

III. ARCHITECTURE

A. *Component architecture*

The component architecture for DIT is displayed in Figure 2.

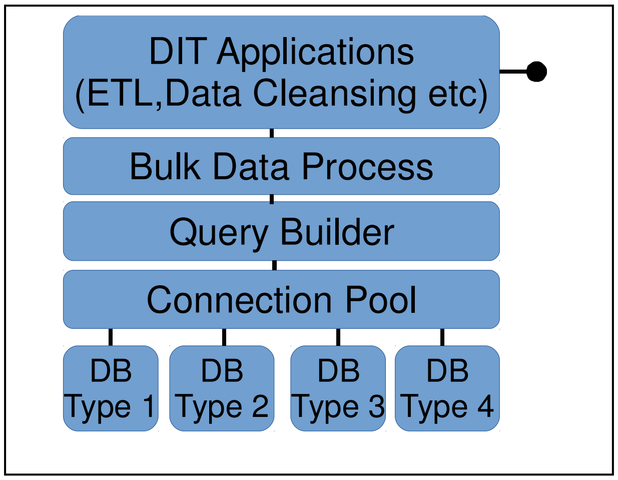

Figure 2. DIT Component Architecture

The application establishes connection with source and target tables through connection pool. For source and target tables, the database connection pool is used for each type of database. The lookup cache for connection pool will provide available connection to the connecting database type. The QB formats SQL syntax according to connecting database types. The business logic in the form of general query arguments is passed to QB to form a valid SQL syntax. The QB component output are read data list and new database structure, such as system columns, system tables and systems data. The bulk data process is a component to process data through multithreaded process whereby its SQLs are also supplied by QB. The applications are also connected to external processes via an interface.

B. *Query Builder interface*

The source and target tables operations rely on the generated SQL statement. Figure 3 shows the QB's interface for Create table and Select query statement.

```
//create a table
ICreate ic= new ICreate(databaseType);
ic.table('edu.address_xd').columns(('NA
ME','string',20),('ADDRESS','string',20
0));
String query= ic.buildQuery();

//select
ISelect is= new ISelect(databaseType);
is.table('edu.address','*').limit(1000,
2000).where('_group_id','001');
String query= is.buildQuery();
```

Figure 3. Query Builder interface

For creating a table, the respective data type semantic (e.g., STRING, INTEGER) are translated into correct syntax (e.g., VARCHAR2 or VARCHAR, NUMBER or INT). Similarly, we have separate interfaces to generate queries for Insert, Update, Delete and Alter with their respective inputs. These are the interfaces for consuming QB component services. The QB's role is to produce the following outputs in relation to source or target tables (based on database type), namely, (1) read statement for paging, (2) create new system tables, (3) create new index and trigger, (4) append new columns, and (5) populate system values.

IV. RESULTS

By implementing the above design and architecture, we are able to create an ETL and data cleansing applications that compatible with MySQL, PostgreSQL and Oracle RDBMS. All SQL issues specific to databases are handled in QB. It is easy to use for user without in depth knowledge of SQL. The DIT's user only needs to configure the function and the target database's table; QB will generate all the SQL statement specified to the target database type. The QB project is loosely coupled it can be easily scaled to more databases without any code change in the main project.

Some of the problems below and their solutions are handled by the QB: (1) the quoted identifier is implemented in QB for all databases without requiring database administrator to configure any database system settings. (2) The schema hierarchy is implemented in QB for all database types as it is mandatory for specific database, i.e., PostgreSQL. (3) QB also handles specific queries that use SQL built-in functions. Specific queries are built when database type is known and its respective built-in functions are included in the queries. (4) The UPSERT command also varies for different database types, which is handled in QB. (5) The auto increment is a standard feature in MySQL and PostgreSQL, but not in Oracle; this is handled in QB by issuing set of dependent queries for different database types.

For application wide problems, the issues and solutions are also being identified: (1) PostgreSQL database has a specific data type called NUMERIC, which can hold decimal and integer, hence, decimal and integer data should be type casted to float type. (2) While using connection pooling on multiple databases, pooled connection instances should be created only once and maintained in a map. This has to be reused to avoid connection leak. (3) Specific JDBC driver like Oracle JDBC doesn't return the auto generated column values by using the function "getGeneratedKeys()". This problem can be overcome by reading the ROWID column to get the generated value.

The DIT has been successfully deployed in User Integration Test (UAT) environment and currently supporting ORACLE, MySQL and PostgreSQL databases.

V. Conclusion And Future Work

This industry experience gives a glimpse of the best practices and the issues faced while using multiple type of database dynamically in a DIT web-based application. The major problems are namely, dynamic database mapping of source and target database, maintaining database connections in application and validation of query syntax for each database types. These problems are difficult for ORM framework as discussed in this paper. Hence, the usage of direct SQL statement is preferred.

The QB is able to generate specific SQL statement for multiple database types. This is an independent component which can be scaled to more databases and plugged to any applications.

A lesson learned during working with the DIT application is that a complex housekeeping is needed when user cancels a job. User could terminate a running job, but as a result, specific SQL commands need to be executed; for example, to drop supporting tables and remove system columns of the target table.

The DIT can work with other RDBMS in future with minimum changes, and the changes are structured and only happen in QB. In future, QB will support version specific SQL queries. Moreover, this service can be extended as web service, which can be consumed by external parties.

References

[1] R. Kimball and J. Caserta, The Data Warehouse ETL Toolkit, Indianapolis: Wiley Publishing, 2004.

[2] B. Walek and C. Klimes, "Expert System for data migration between different database management systems," Advances in Data Networks, Communications, Computers and Materials, 2012, pp. 167-172.

[3] N. Vijayendra and M. Lu, "A Web-based ETL Tool for Data Integration," in The 6th International Conference on Human System Interaction (HSI), IEEE Explore, Sopot, Poland, 2013, pp. 434-438.

[4] RedHat, "Hibernate ORM," [Online]. Available: http://hibernate.org/orm/. [retrieved: May, 2015].

[5] Clinton. Begin, "MyBatis," [Online]. Available: http://blog.mybatis.org/. [retrieved: May, 2015].

[6] Pentaho, "Data Integration - Kettle" [Online]. Available: http://community.pentaho.com/projects/data-integration/ [retrieved: May, 2015].

[7] Talend, "Talend Product - Data Integration" [Online]. Available: https://www.talend.com/products/data-integration [retrieved: May, 2015].

[8] E. K. Karuppiah, Y. K. Kok and K. Singh, "A Middleware Framework for Programmable Multi-GPU-Based Big Data Applications" Springer Link, 2015, pp. 187-206.

[9] J. Celko, "Joe Celko's SQL Programming Style," Morgan Kaufmann, 2005.

[10] ANSI X3.135-1992, American National Standard for Information Systems — Database Language — SQL, November, 1992.

[11] PostgreSQL, "Lexical Structure," [Online]. Available: http://www.postgresql.org/docs/9.3/static/sql-syntax-lexical.html. [retrieved: May, 2015].

[12] Oracle, "Schema Object Names," [Online]. Available: http://dev.mysql.com/doc/refman/5.6/en/identifiers.html. [retrieved: May, 2015].

[13] Oracle, "Schema Object Names and Qualifiers," [Online]. Available: http://docs.oracle.com/cd/B28359_01/server.111/b28286/sql_elements008.htm#SQLRF51109. [retrieved: May, 2015].

[14] Oracle, "MySQL SELECT," [Online]. Available: http://dev.mysql.com/doc/refman/5.0/en/select.html. [retrieved: May, 2015].

[15] Oracle, "Ask Tom ROWNUM," [Online]. Available: http://www.oracle.com/technetwork/issue-archive/2006/06-sep/o56asktom-086197.html. [retrieved: May, 2015].

[16] Apache, "The DBCP Component," Apache, 12 July 2014. [Online]. Available: http://commons.apache.org/proper/commons-dbcp/ [retrieved: May, 2015].