SCHEMA INDEPENDENT APPLICATION SERVER DEVELOPMENT PARADIGM

A. Afaq, G. Graham*, L. Lueking, S. Veseli*, V. Sekhri
Fermi National Accelerator Laboratory
Batavia, IL 60510, USA

ABSTRACT

This paper presents a paradigm for rapid development of an application server in a schema independent fashion. The central idea is to represent tables as objects and use generic templates and algorithms to generate and execute queries spanning multiple tables, at runtime. As an example, Dataset Bookkeeping Service [1] for CMS Experiment [2] is presented.

INTRODUCTION

The idea of using Objects for database manipulation has several different philosophies and implementations, from Object Database Management Systems [3] (ODBMS) to Object-Relational Mapping. Every approach has its own limitations [4] [5] [6] and in general, such developments are engineering intensive and time consuming. Most of the effort is focused at identifying interesting relationships and composing compound queries to retrieve needed information with minimal database queries. Generally, these queries end up being hard-coded and any maintenance or schema change needs manual code changes.

We present here a paradigm for rapid development of an application server in a schema independent fashion. The emphasis is on generating Database and Query Objects directly from a Data Definition Language (DDL) [7] file and using them in association with a Query Object Layer to perform database operations. Any schema change/maintenance will only require re-running the code generation and not any type of manual changes.

The major components of such a system are:

- Code Generator, A set of tools for generating C++ Database/Query objects.
- Set of Database/Query Objects (DO/QO) are C++ class representation of individual tables and joins. The join conditions are deduced from Foreign Key relations. Developers may provide additional join conditions, if required.
- A Query Object Layer (QOL) provides the framework for creating runtime Objects for database interaction.
- Algorithms that generate Queries/SQL at runtime for any type of DO/QO during a QOL invocation.
- Business Logic Layer (BLL) makes it easy to rapidly write the Business Logic.

CODE GENERATOR

- The Code Generator parses DDL and extracts details of all tables, their keys and relationships.
- Each table in DDL is translated into a Python object of type rowRepresentation.
- A set of tables can be seen collectively as a “Query Object View” (QOV). The user describes which tables need to be included in the QOV. Describing a QOV is a very simple configuration step which can be performed in a few lines of Python code.
- The Code Generator figures out the details of how to collectively look at the QOV and generates appropriate multiRowrepresentation objects in Python.
- C++ classes are written out for each row/multiRowrepresentation object.
- The Code Generator is developed in Python.

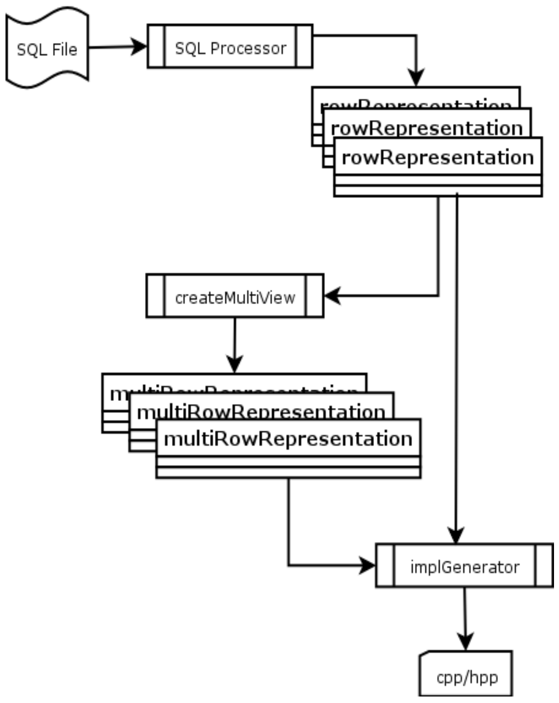


Figure 1: Structure of Code Generator

QUERY OBJECT LAYER

The Query Object layer hides the details of the query generation and the database schema from the user of this layer, as well as from the developer. The Objects that hold the database schema information (rows and tables) are generated by the code generator. The developer does not need to know about the schema as such. This layer provides simple insert and select interfaces on the C++ objects (generated table and rows). The interface

* Greg Graham and Sinisia Veselli recently left FNAL.

implementation is templated and can be used for any generated row or table objects. The template interface gives a great flexibility to incorporate any changes in the schema, since the objects can be regenerated by the code generator and the interface implementations remains the same.

The Organization of this layer is shown in the Figure 2.

- **RowClass** is the generated C++ object (class) corresponding to a row in a specific table. The RowClass objects inherits from a parent class RowInterface and implements the interface for simple setValue and getValue calls to set or get the values to and from the RowClass.
- **SchemaNConstraintsClass** holds the entire schema including the foreign keys, unique keys and other references in a table. This class is generated, so any changes in the schema will get reflected in this class. It inherits from a parent class BaseSchemaNConstraintsClass, and implements the interface for simple getSchema, getPrimaryKeys, etc calls to get the schema information about the table.
- **SQLClass** can generate the SQL statements for any table from its schema. Its implementation is independent of the changes in the schema.
- **TableTemplateClass** is a templated container that can contain a vector of RowClass. It inherits from TableInterface class, and implements the interface for insert and select calls. It instantiates objects of SchemaNContraintsClass, RowClass and SQLClass to contain the Row data, the schema for the table and the mechanisms to generate the SQL queries.

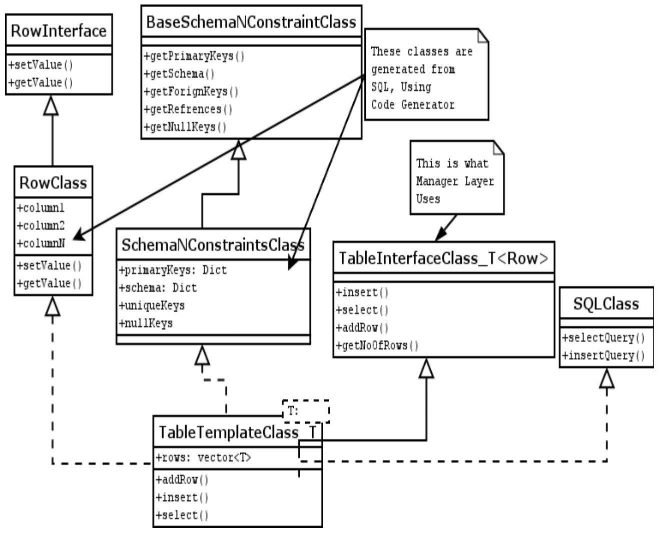


Figure 2: QOL Class Diagram.

DATABASE ALGORITHMS

Database algorithms are a key part of the Query Object Layer. Algorithms are empirically evolved to handle any kind of Insert/Select operation for a Database (Single Table) or Query (Multi Table) Object.

These algorithms generate the required SQL, execute them and return the results to the calling layer by storing them as Database/Query Objects. All data exchanges between QOL/Algorithms and user layers happen through Database/Query Objects. After a select operation the calling layer receives a set of Dataset/Query Objects. The calling layer must provide a set of DO/QO for the insert operation. Figure 3 shows an example flowchart for smart insert operation.

DATASET BOOKKEEPING SERVICE PROTOTYPE AS AN EXAMPLE

The Dataset Bookkeeping Service is a tool being developed at Compact Muon Solenoid Experiment (CMS) to define, discover and track datasets for processing event data as a key component of Data Management System.

The user interaction with the application server is either through a Python Web Services Interface [8] or through a direct Python Interface. In both cases, the user provides a set of data structures (Client Data Structures in Python) to write into or to read from the database, and invokes an API call. The Python interface (using SWIG [9]) translates client data structures into appropriate QOL Table/Multi-Table Objects in C++ and then the Business Logic Layer performs the desired operations.

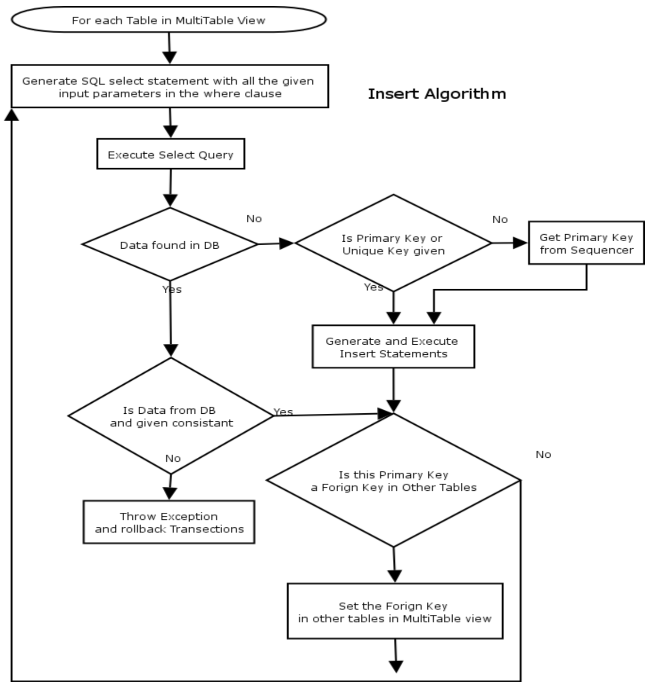
SAM Web Services [10] shows very good performance of Web Services under various load conditions and with different usage patterns.

The various layers in the DBS prototype are (shown in Figure 4):

- **DB Layer**: Handles all the low level database transactions using unixODBC [11].
- **Query Object Layer**: Creates C++ objects corresponding to Tables and Rows in the schema and the views configured by the user. It provides a simple insert and select interface.
- **Business Logic Layer**: This layer provides implementations corresponding to API calls. These implementations are called Managers which instantiate the objects from QOL and make use of the interface provided to retrieve or insert data. Any logic about validation or cross referencing between various table and multi tables are done in this layer.
- **Interface Layer**: This layer exposes the DBS defined API to any user of the layer. The API calls are translated into invoking Managers in the Business

Logic Layer onto which the specified task are delegated.

- **Python Interface**: Translates the Client data structures into C++ Tables and Rows object (using SWIG) and invokes the API from the Interface layer.



Insert Algorithm
For each Table in MultiTable View
Generate SQL select statement with all the given input parameters in the where clause
Execute Select Query
Data found in DB
No
Yes
Is Primary Key or Unique Key given
No
Yes
Get Primary Key from Sequencer
Generate and Execute Insert Statements
Is Data from DB and given consistant
Yes
No
Throw Exception and rollback Transections
Is this Primary Key a Forign Key in Other Tables
No
Set the Forign Key in other tables in MultiTable view

Figure 3: Insert Algorithm.

These layers can collectively be used as a plug-in inside any standalone application or server. In DBS Web Services case we have a SOAP Server that uses this plug-in to communicate with the remote clients.

The whole development cycle requires only:

- Describing QOV and running the Code Generator to produce Database and Query Objects.
- Writing appropriate Client Data Structures and their translation into QOL Objects.
- Writing Business Logic, which is mainly the instantiation of appropriate QOL Database/Query Objects and invoking API calls.

The rest of tasks to interact with the database, query generation, execution and etc. are all handled by the QOL and BLL.

CONCLUSION

Test results from the test system built using above mentioned development process are very promising, following are major observations,

- There are fewer chances of making any Query mistakes. We have several examples where a failure in QOL actually pointed towards a mistake in the DDL and Schema.
- The turn around time to completely incorporate all the schema changes is short.
- The chances of executing an incorrect query are minimal as query generation is done automatically.
- The user interface deals in terms of views (of single and multi tables) and rows which is much simpler than dealing with queries.
- The development cycle is simple and short since most of the code is generated by the code generator.
- The plug-in has a simple interface and can be easily used in any application.

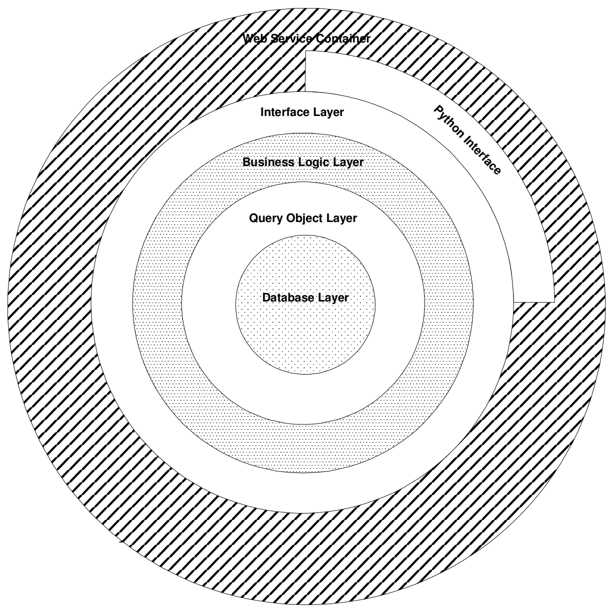


Figure 4: DBS Server Layers

REFERENCES

[1] Distributed Data Management in CMS, A. Fanfani, CHEP06, Mumbai 2006
[2] http://en.wikipedia.org/wiki/Compact_Muon_Solenoid
[3] Introduction to Object-Oriented Databases. Kim, Won, Massachusetts: The MIT Press, 1990
[4] Object-Oriented Database Systems: Strengths And Weaknesses. Kim, Won, Journal of Object-Oriented Programming Focus On ODBMS, (1992) pg. 27
[5] A Comparison of Object-Oriented Database Management Systems for Engineering Applications. Ahmed, Shamim, et al., Massachusetts Institute of Technology, Research Report R91-12, 1991
[6] Hitting the Relational Wall, Loomis, Mary E.S., Journal of Object-Oriented Programming, January 1994, pp. 56- 59+
[7] http://databases.about.com/od/sql/a/sqlfundamentals_2.htm
[8] http://sourceforge.net/projects/pywebsvcs
[9] http://www.swig.org
[10] SAMGrid WEB services, S. Veseli, FNAL, CHEP06, Mumbai 2006
[11] http://www.unixodbc.org