Extended Object Languages for the Extolware Persistence Framework

Lim Tong Ming
Monash University Malaysia
Selangor, Malaysia

Lee Sai Peck
University of Malaya
Kuala Lumpur, Malaysia

lim.tong.ming@infotech.monash.edu.my

saipeck@um.edu.my

Abstract

Users interact with a database system through a set of database languages and this makes designing database languages a very challenging task to a computer software engineer. A set of well-defined database languages must be easy to learn, easy to understand and powerful enough to capture semantic of a problem domain. This paper discusses design issues of a proposed database language, namely Extended Object Language or EOL for short, for an Extolware Persistent Object framework (Lim & Lee, 1997, 1998, 1999, 2001, 2002a, 2002b, 2002c) that provide wrapping services for relational database systems and multidimensional database systems (DataPro, 1996; IBM Corp., 2001; Informix Software Inc., 2001a, 2001b). This research examines SQL3 (Fortier, 1999) and ODL/OQL (Cattell & Barry, 1999) with an overview of their language constructs and operators that support object-oriented requirements as stated in Object Data Management Group (ODMG) object model. Next, a discussion on the Extended Object Language (EOL) and its language constructs are examined. This is followed by a close examination of new database operators and constructs introduced into EOL. A design overview and evaluation of these database languages are examined. A summary on these languages is presented at the end of the paper with conclusion and further research plans.

Keywords: SQL3, ODL, OQL, EOL

Introduction

Structured Query Language or SEQUEL (later rename to SQL) has been evolving since early 1970 after Relational Data Model was first proposed by E.F. Codd (Date, 2000) and released by IBM as System R commercially to the market. The impact was great then with its ease of use and simple to understand concept. Since then, SQL2 was continuously enhanced and evolved with many new features such as new data types, new predicates, enhanced security and semantic, and active database. As such, the idea of having a query language must incorporate three requirements, that is simple to use, simple to understand and powerful enough to capture difficult and complex domain semantics. The SQL standardisation committees, ANSI (X3H2) and ISO (ISO/IEC JTC1/SC21/WG3), have been adding and enhancing the SQL standard so that new requirements could be handled by SQL efficiently, nevertheless, one must also not forget these three design ingredients that make SQL the de factor standard database language. Many questions such as "Could we simplify SQL3?" by David Beech (2001) and "Do we really need SQL3?" by John

E. Seytabla (2001) and other database language issues concerning language complexity and computational completeness are always discussed and raised in journals and articles, hence, designing database languages are no small task at all.

ODMG, on the other hand, proposes Object Definition Language and Object Query Language standard, to get most of the leading object database vendors to agree and endorse a set of guidelines in order to consolidate skill, expertise and technology so that applications developed on Object Database Management System (ODMS) can easily be migrated with minimum changes to another ODMS (Yugopuspito & Araki, 2000) environment. SQL standard was used by ODMG as a reference in the process of drafting ODL and OQL standard. The intension is not to produce a computationally complete database language but to create a standard, which is easy to migrate from SQL to OQL in very short period. This will help popularising ODMS (Yugopuspito & Araki, 2000) in a much faster pace. Even some aspects of Object Database technology are not quite as mature as Relational Database technology; the need of such technology is definitely undeniable. Hence, many debates and proposals are put forward to relational and object people such as integrating SQL and OQL and "merging" relational with object database technology (Pujianto Yugopuspito & Araki, 2000).

C. J. Date, M. Nelson, Francois Bancilhon and Richard Mark (1995) in "Objects and SQL: Strange Relations?" argue facts such as possibilities of bringing SQL and OQL as the standard of database language, keeping relational database technology unchanged and using object database technology on a specific domain. But the reality is the market demand should be the bottom line that drives and dictates what is required and what is not, not the technology.

The next three sections present an overview on Extended Object Language or EOL, SQL3 and ODL/OQL on their database constructs, operators and features. The fourth section summaries these languages' features from three aspects, i.e. ease of use, ease of understanding and powerful constructs and operators. Object-oriented features such as simple object, complex object, association, collection and inheritance are criteria used in conjunction with these languages' constructs for an overall languages' review and assessment.

Extended Object Language (EOL)

Extended Object Language or EOL is designed with three major features in mind; they must allow creation, deletion and modification of classes, manipulation of objects such as insert, delete and update of objects, and querying of objects from databases by selecting objects with filtering criteria and other querying features. Most of the language constructs designed and implemented in EOL languages are borrowed from ODL/OQL and SQL3. EOL constructs such as CREATE, ALTER, DROP, INSERT, DELETE, UPDATE and SELECT are designed and implemented in this research project in order to provide a complete set of constructs for defining, manipulating and querying of objects. In order to provide a complete management environment, additional features are added into the system in order to support Multidimensional Database Systems besides relational database system. EOL consists of two sets of language. They are Extended Object Definition Language (EODL) and Extended Object Query Language (EOQL). In addition, a set of EOL Application Programmable Interface (API) that consist of object-oriented classes are designed and implemented within the architecture to carry out features stated above so that developers can manipulate objects and communicate with the database systems in an application. The EOL Application Programmable Interface (API) is basically classes with methods (**Table 1**) that implement constructs in native host languages such as Java and C++ to provide similar types of features and functions. Each construct is designed around object-orientation features such as simple object, complex object, inheritance, collection such as SET, BAG, ARRAY and LIST, and association. The following figure highlights some of the methods that are designed and imple-

mented in the architecture in order to enhance overall database functionalities and performance of the EOL API.

Table 1 Summary of EOL API

EOL commands group	Class Schema Management	Inserting objects	Removing objects	Modifying objects	Selecting objects
EOL commands	CREATE ALTER DROP	INSERT	DELETE	UPDATE	SELECT
Object-oriented features					
Simple object	-	-	-	-	-
Complex object	-	-	-	-	-
Inheritance tree	-	Trace()	DelChild()	-	FilterClass()
Collection object	-	Loop()	Loop() DelAll()	UpdAll()	FindColl() OnlyOID()
Association object	-	Search() Locate() InsertLink()	DelAll()	UpdAll()	FindLink() OnlyOID()

Extended Object Definition Language

EODL provides CREATE construct to define types such as CLASS, INTERFACE and STRUCT in its grammar. “CREATE INTERFACE calSalary { … }” is an example which creates an interface that contains only abstract operations whereas “CREATE CLASS person { … }” defines both data members and interfaces of person class. STRUCT operator only defines data members of type without operations. “CREATE STRUCT staff { … }” defines a staff type that consists of only data members.

To support inheritance feature, EXTENDS operator is used to specify a parent type from which it inherits. In EODL, EXTENDS could be used to inherit CLASS and INTERFACE. EOL can automatically identify different type of parent classes. “CREATE CLASS human { … } EXTENDS calSalary” defines a human class that is inherited from calSalary INTERFACE whereas “CREATE CLASS human { … } EXTENDS person” defines a human class that is inherited from person CLASS. EODL does not require additional operators, which create a higher learning cycle/curve for users and developers.

EODL supports collection operators; they are SET<t>, BAG<t>, LIST<t>, ARRAY<t> and DICTIONARY<t>. To define a data member that holds a set of instances of string, simply create an attribute of “degree SET<string>” that defines a *degree* data member that holds a set of string instances.

As for association link, EODL has three operators that manage relationships between classes. They are REF, UNARY and BINARY; which REF is used to refer to a type, UNARY is to specify a single directional relationship and BINARY is to specify a two-way relationship link. As for a one-to-many and many-to-many relationship, use of collection operators such as SET, BAG, LIST and ARRAY allow a class’s member to define these relationships.

EODL design includes constructs and operators to support object-oriented features that are easy to use, easy to understand and easy-to-capture rich semantics in a domain. As a result, the type definition allows methods to be defined using a mix of multiple object-oriented programming languages and data members.

```
CREATE CLASS student
{
        NAME string;
        STUDID integer;
        DEGREE SET<string>;
        PARENT REF(FAMILY_MEMBER);
FRIENDS SET<PEOPLE> BINARY PEOPLE IS_FRIEND_OF;
        void get_NAME();
        integer get_STUDID() { return STUDID; }
}
```

The example above defines CLASS student with two data members, NAME and STUDID and two methods get_NAME() and get_STUDID(). EODL is designed to identify whether the first symbol is a method's return type or an attribute name, if it is a return type which is inclusive of void, integer, real and string, then this part of the definition will be treated as a method, otherwise, it will be recognised as an attribute.

Extended Object Query Language

EOQL introduces many additional new database operators into the language set. New database operators and constructs are INSERT, DELETE, UPDATE, EXPLORE, EXPLORECluster, OID, UP and DOWN. These additional database constructs each correspond to the database API services at the heart of the Extolware persistent object framework. The following is a list of examples that demonstrate the use of these new database operators.

The insert into *staff(name:"John", age:34, staffid:"id123")* inserts an object by creating a staff object with the name "John", age 34 and a staffid of "id123".

For an insert statement that involves collection and relationship attribute, to insert a *lecturers* object which contains a collection attribute called *qualification* and a relationship attribute called *lectures*, the *insert into lecturers(staffid:"id123", name:"John", age:34, Col:qualification:["Bsc", "Mcs", "Phd"], Rel:lectures:[oid["courses@12345"], courses(courseid:"cse1301", coursename:"C programming", credithours:6)])* statement uses a database operator *Col* for a collection attribute and a relationship operator *Rel* is used for a relationship attribute.

To update a people instance, the *update people set people.age:24, gender:"male" where born="1979" and sex="m"* updates people's age to 24, gender to "male" for all people instances that fulfil the WHERE clause.

The selection of objects is carried out using a select operator. The *select lecturers.qualification.* from lecturers where lecturers.staffid = "id123"* selects objects from lecturers class that fulfil staffid that equals to "id123". Notice that path expression is expressed using a dot delimiter.

To delete any lecturers instances that contain name equals to nil, the following statement *delete lecturers.name.* from lecturers where lecturers.name=nil* carries out this database activity.

In short, EOL provides two layers of services to users and developers. EOL is designed for those who wants to access to persistent objects without the knowledge of programming. Moreover, EOL API is developed for developers who want to manipulate objects within the application kernel. In addition, methods are programmed in either Java or C++ native host language; no additional set of non-procedural language is required.

SQL3 Database Language

SQL3 starts to change from year 1999 and this effort is due to increasingly higher demand for database constructs that could manage complex object, abstract data type, inheritance, polymorphism, encapsulation and other object-oriented related features (Han, 2001). SQL3 standard has proposed an eight-part language. They are Framework, Foundation, Call-Level Interface (CLI), Persistent stored modules (SQL/PSM), Bindings, Transactions, Temporal and Object. These six-part languages form the foundation of the SQL3's core with its extension (Eisenberg & Melton, 2001).

The SQL Framework provides an overview of the standard and components that make up SQL3 (Seytabla, 2001). To handle new data types and complex data requirement, SQL Foundation contains triggers, recursion, new data types, user-defined tables, subtables, abstract data type, BLOB, CLOB, row type, supertype and subtype, collection types such as ARRAY, SET, LIST and MULTILIST. SQL Call-Level Interface or SQL/CLI bind SQL database and user-interface language which supports both static and dynamic SQL statements by passing parameters whenever necessary. The result returned by such a call to the database is returned to the calling routine within the application kernel. SQL Persistent Stored Modules support functions, procedures and modules, which these routines are either written in SQL or some 3GL. It gives power to DBMS and performance (Fortier, 1999). SQL Binding allows dynamic invocation of SQL statements within the application kernel through the use of SQL EXEC statements. SQL Transactions provides features so that database state maintains ACID properties during database management activities. This is a critical requirement in a distributed database environment. Object-oriented database management systems have the requirement to handle temporal information effectively by the database engine. SQL Temporal supports temporal information and provides features such as version, history and interval. One of the key features of object-orientation requirement is the management of unique object identity information within the database engine itself. SQL Object provides a means so that database management systems can understand application-specific data types. This feature supports object-orientation requirement such as object identity and row object. To support object-oriented features within SQL3, a few language extensions are in the discussion state and these new proposals might become the soon to be released SQL4 (Eisenberg, & Melton, 2001)

The basic element of SQL3 is still table and this basic data structure could store data with a higher degree of flexibility, syntax and semantics. In SQL3 standard, a table does not have to adhere to normal form as Codd originally specified it. In SQL3, basic features and constructs are still supported and backward comparability is maintained. Extended features and constructs in SQL3 are summarised in Table 2. These newly added features are essential so that SQL3 can handle complex objects and a higher level of abstraction as outlined in Object Data Model standard. The following sections examine SQL3 constructs in detail (Fortier, P., 1999).

This discussion focuses on SQL3 commands that manage class schemas such as CREATE, ALTER and DROP, INSERT, DELETE, UPDATE and SELECT. For each construct, discussion will focus on operators used and functionalities proposed relevant to object-orientation features such as simple object, complex object, inheritance, collection and association (Fortier, 1999).

Object-oriented concepts were injected into SQL to support most but not all object-oriented features proposed in object-oriented data model. These features and constructs are designed to handle sophisticated data modelling requirement in a semantically rich domain. The primary basis for supporting object-oriented structures is user-defined types (ADT, named row types and distinct types), row and reference types constructors, collection type constructors, user-defined functions and procedures and BLOB and CLOB supports (Fortier, 1999).

SQL/PSM provides function, procedure and method to declare signature of an interface. SQL3 uses METHOD operator in CREATE TYPE construct to declare a method. Modifiers such as PUBLIC, PRIVATE and PROTECTED could be used on TYPE's attributes and FUNCTION, PROCEDURE and METHOD (Beech, 2001).

User-defined types are supported in SQL3 by construct CREATE TYPE. Tables in SQL3 could be defined as of a type defined by system or user-defined. Each instance in the table is row of object with an object identity. The construct used here is "CREATE TABLE x OF y_type" where y_type is defined by "CREATE TYPE" construct. Hence, each object is stored as a row in table x. Row type uses ROW in SQL3 to define a sequence of fields with the data type resembling table definition. This allows complete rows of such type to be stored in a variable. It could also represent types of rows in a table. Named row type uses "CREATE ROW TYPE x (…)" to define a user-defined data type without encapsulation structure. A named row type could also be used as type of rows in a table. To access an attributes of a user-defined type, one could use either '.' or dot notation or functional notation such as f(x).

REF operator is used to point to or refer to a type that is defined during compilation. REF type variable always associates to a specific structured type. An attribute such as "manager REF(EmployeeType)" would mean manager is a variable in which the value of such variable is of EmployeeType. To refer to the value of EmployeeType, use "pointer" or "->" such as "SELECT emps.manager->last_name" or "SELECT emps.manager..last_name" so that it follows to the manager's last_name attribute in order to access the value. References could be used in path expression to refer and navigate along a path from one table to the other table.

Inheritance feature uses UNDER to create subtype/supertype relationship for two tables. "CREATE TYPE employee UNDER person (…)" defines employee type as a subtype of supertype person.

Aggregation is supported in SQL3 through collection type such as SET(<type>), MULTISET(<type>), LIST(<type>) and ARRAY(<type>). Every row can also be nested. A number of predefined parameterised collection types can also defined in attributes of each row (Paul Fortier, 1999).

SQL3 extends a number of operators and constructs in order to support requirements of object data model so that complex data object and highly unstructured scenario could be captured easily and effectively. However, SQL3 is still not a widely accepted standard due to specification is still evolving and a lot of new features are adding into the SQL3 language and model.

Object Definition and Object Query Languages

Object Data Management Group or ODMG proposed ODMG Object Model as the standard so that database vendors are able to build object-oriented database management systems and applications that are portable and easily migrate with least changes from vendors to vendors. ODMG proposes Object Definition Language (ODL) and Object Query Language (OQL) and omit manipulation language, which intend to let developers glue these manipulation features within classes defined by developers. ODL is a definition language that specifies object's characteristics and types. It is designed for portability, language-independent, multi-vendor and simple.

ODL and OQL are database languages that allow a user to define and manipulate instances with minimum effort and learning cycle. The object-oriented data model, like relational and object-relational data model, complete with basic constructs and operators in order to manage and manipulate data and objects effectively. Hence, for the scope of the research, the discussion will cover class management and object manipulation with respect to simple object, complex object, inheritance, aggregation and association.

ODL is a specification language. It defines object's specification. Object Interchange Format or OIF is used to dump and load the current state of an object database system to or from a file or set of files. This is to allow ODMS from various vendors to comply to a set of standard, thus allowing persistent objects to migrate from one to the other.

The types defined by ODMG object model consist of specifications and implementations where specification is for the external and implementation is language-dependent so that such type could be implemented in any languages on any platform. For example, "interface EMPLOYEE {…}" defines an employee interface that consists of abstract behaviour only. As for "class PERSON {…}", it defines abstract behaviour and abstract state and "struct Complex {…}" defines only abstract state. Implementation of an object type consists of a representation and a set of methods. The representation is a data structure that is derived from the type's abstract state by a language binding; methods are functions and procedures that are derived from the type's abstract behaviour by the language binding.

Subtyping and inheritance is supported in ODMG object model as well. Interface uses ':' to carry out subtype and supertype relationship between parent and children interfaces. Classes use EXTENDS to define inheritance behaviour between object types. For example, "class EMPLOYEE extends PERSON : STAFF { … }" defines that EMPLOYEE class inherits all attributes and methods from class PERSON and inherits all STAFF interface's abstract behaviour.

Object identity (OID) is a unique key that uniquely identifies an object as soon as it is instantiated. An OID is part of the feature proposed by ODMG.

ODMG object model supports collection of object. Constructs such as SET<t>, BAG<t>, LIST<t>, ARRAY<t> and DICTIONARY<t> are collection constructs in ODMG's object model standard.

Relationships between classes are defined using RELATIONSHIP operator in ODL. These relationships could be either unary or binary. Cardinality between types could be one-to-one, one-to-many and many-to-many. The many side uses collection operator to hold instances of objects. A relationship allows traversal path between objects participating in the relationship.

ODL and OQL are database languages proposed by ODMG for management and manipulation of persistent objects modelled according to ODMG object model. This allows ODMS from vendors implement and deploy with minimum differences across multiple ODMS products.

EOL, SQL3 and ODL/OQL: A Design Evaluation

Database languages such as EOL, SQL3 and ODL/OQL are always designed with specific goals, audients, type of applications, problem domains and data model.

SQL3 is designed to maintain backward compatibility with SQL2 standard with full support of both relational data model as well as object-relational data model. As for ODL/OQL database language, it is designed to support ODMG data model and applications that deal with unstructured complex data objects. ODMG data model is implemented with database capabilities and object-oriented programming languages to provide a complete database development environment. As a result, object manipulation constructs are not part of ODL/OQL standard, which means no additional set of database languages needs to be learned and developers are supposed to create the manipulation capabilities within the class definitions to manage database operations.

The design consideration for EOL is very much different from SQL3 and ODL/OQL. EOL is designed to support a complete object-oriented view with persistence storage being either relational, object-relational, multidimensional or object-oriented data models. As a result, EOL provides full database constructs such as CREATE, INSERT, UPDATE, DELETE and SELECT in the lan-

guage. It provides class definitions management capabilities, object manipulation capabilities and object querying capabilities. In addition to this, a complete set of class API is also taken into consideration. The class API is language-dependent, which means class API is C++ and Java must be created to support these two languages. This is to provide better performance at the application kernel when it communicates with the database. The developers have another option that is to code EOL statements at the application kernel by sending such statements to the EOL Parser and Syntax Checker and result set will then returned by the host language's SQL library.

Table 2 Summary of weaknesses and strengths of SQL3, EOL and ODL/OQL

	EOL	SQL3	ODL/OQL
General			
Type of database language	Definition/ Specification	Definition	Specification
Complete database API	Complete	Complete	No**
Lean additional non-procedural language	No	Yes	No
Object-oriented features supported			
Simple Object	Yes	Yes	Yes
Complex Object	Yes	Yes	Yes
Inheritance	Yes	Yes	Yes
Association/ Relationship	Yes	Yes	Yes
Aggregation/ Collection	Yes	Yes	Yes
Types			
- Interface	Yes	Yes*	Yes
- Class	Yes	No	Yes
- Struct	Yes	No	Yes
Design consideration			
Ease of use	Simple	Simple	Medium
Ease of understanding	Yes	Yes	Yes
Constructs and operators to support OO features	Yes	Yes	Yes
Constructs supported			
CREATE	Yes	Yes	Yes
INSERT	Yes	Yes	No
DELETE	Yes	Yes	No
UPDATE	Yes	Yes	No
SELECT	Yes	Yes	Yes

* Use row and column type ** Users must create or customise class manipulation methods

As shown in Table 2, it is obvious that these three database languages fulfil their defined objectives and provide all the essential object-orientation supports to an acceptable level.

SQL3 intends to accommodate, fulfill and support all object-oriented features by complicating its type construct. Non-procedural language proposed by SQL3 is creating a very stiff learning cycle to new users. SQL3 supports type level function, procedure and method signature with full implementation permitted, creating messy and unmanageable code segments.

ODL/OQL lacks of users level of object manipulation functionalities. To a certain extent, this creates slow acceptance among naïve users community. Migration of RDBMS application to

ODBMS applications is also at the down side due to technology issues such as poor database per formance and lack of formal foundation.

As for EOL, utilising features from both SLQ3 and ODL/OQL standards allow faster acceptance among users and developers. EOL provides complete set of constructs and operators to support object-oriented features while language-dependent EOL API is complementing the languages' performance at the application kernel.

An Implementation Evaluation on EOL and SQL3

EOL is implemented in Java programming language to provide the retrieval and storage functionalities to the proposed framework. The implementation of EOL has allowed an evaluation on the proposed design and database constructs introduced into the EOL grammar. The following describes the observations and results of the implementation:

1. The introduction of INSERT, DELETE and UPDATE database constructs has enabled the ease of object operations with the persistent stores. The users can save an object, delete objects and update objects without the need to learn an object-oriented programming language. The proposed EOL, however, is discovered that multi-level nested relationships and collections of objects are very confusing and lengthy. This is an area of the EOL that needs further rethinking and redesigning.
2. The computational completeness feature is lacking from the current EOL implementation. The result produced by EOL must be able to be used by other EOL statements. SQL3 supports this feature. The appropriate container to be used to hold the result produced by EOL statements must be considered. This is due to the different collection containers supported by the EOL grammar.
3. The complexity of the algorithms and the much to-be-improved performance of the EOL module when it interacts with the core of the framework is still an area to be improved.

The observations made from the implementation of EOL are a fruitful one. It allows a better understanding of the EOL design. Hence future works can be formulated and a better EOL system can be developed.

Conclusion

Experience gained from studying and analysing SQL3 and ODL/OQL is an extremely precious and challenging task. Designing Extended Object Language (EOL) to fulfil users and developers within a language paradigm is no easy job. Efforts will be invested into the simplified language constructs and operators while enhancing EOL's grammar capability and providing a much efficient set of EOL API in the future research plans for multiple object-oriented programming languages.

References

Beech, D. (2001). *Can SQL3 Be Simplified?* Retrieved from http://jeffsutherland.org/x3h2/97-094.pdf

Cattell, R. G. G. & Barry, D. (1999). *The object data standard: 3.0.* Morgan Kaufmann.

DataPro. (1996). *Pick systems D3 server 7.0.* McGraw-Hill.

Date, C. J. (2000). *An introduction to database systems* (7th ed.). Addison-Wesley.

Date, C. J., Nelson, M., Bancilhon. F. & Mark, R. (1995). Objects and SQL: Strange relations? *SIGMOD 95.*

Eisenberg, A. & Melton, J. (2001). *SQL:1999, formerly known as SQL3.* http://dbs.uni-leipzig.de/en/lokal/standards.pdf

Fortier, P. (1999). *SQL 3 implementing the SQL foundation standard*. McGraw Hill.

Han, K.-J. (2001). *SQL3 standardization.* Kon-Kuk University, Department of Computer Engineering.

IBM Corp. (2001). IBM U2 nested relational databases. A White Paper.

Informix Software Inc. (2001a). The architecture of UniVerse. A White Paper.

Informix Software Inc. (2001b). UniVerse & UniData - Extended relational databases. A White Paper.

Lim, T. M. & Lee, S. P. (1997). Classes and objects management multidimensional DBMS data model. *IASTED International Conference Software Engineering* (SE '97) San Francisco, 2-6 Nov. 1997, California.

Lim, T. M. & Lee, S. P. (1998). Objects collection management in multidimensional DBMS data model. *IEEE - ISORC Kyoto International Conference Hall*, Kyoto Japan, April 20-22, 1998.

Lim, T. M. & Lee, S. P. (1999). Objects to multidimensional database wrapping mechanism. *Annual AoM/IaoM International Conference on Computer Science*, August 6-8, 1999, Westgate Hotel, San Diego, California.

Lim, T. M. & Lee, S. P. (2001). An object to R/MDBMS persistence framework, *REDECS2001 conferences,* Kuala Lumpur, Oct. 22-25, 2001 UniTen-UPM.

Lim, T. M. & Lee, S. P. (2002a). Object schema management facilities in an object wrapper. *The 2002 International MultiConferences in Computer Science*, Las Vegas (14 Joint Int. Conferences).

Lim, T. M. & Lee, S. P. (2002b). Object schema management tool. *Malaysia Journal of Computer Science*.

Lim, T. M. & Lee, S. P. (2002c). Object caching using XML document structure. *CITRA2002.*

Seytabla, J. E. (2001). W*hy we need SQL3.* SQL3.

Yugopuspito, P. & Araki, K. (2000). Evolution of relational database to object-relational database in abstract level. Kyushu University.

Biographies

Tong Ming, Lim is a PhD candidate at the Faculty of Computer Science and Information Technology, University of Malaya. He is working on the object-oriented wrapper in his final year. He worked in TAR College, as a GM and IT Director in two commercial software houses for about 10 years. He is currently lecturing in Monash University, Malaysia, on computer science courses such as database management system, artificial intelligence and computer graphic programming. He is an ACM member and IEEE member since 1993. He could be reached by lim.tong.ming@busit.monash.edu.my.

Sai Peck Lee is currently an associate professor at Faculty of Computer Science & Information Technology, University of Malaya. She obtained her Master of Computer Science from University of Malaya in August 1990, her Diplôme d'Études Approfondies (D. E. A.) in Computer Science from University of Pierre et Marie Curie (Paris VI) in July 1991 and her Ph.D. degree in Computer Science from University of Panthéon-Sorbonne (Paris I) in July 1994. Her current research interests include Software Engineering, Object-Oriented (OO) Methodology, Software Reuse and Framework-based Development, Information Systems and Database Engineering, OO Analysis and Design for E-Commerce Applications and Auction Protocols. She has published a number of research papers in several computer science journals as well as in local and international conferences. She is a member of IEEE Computer Society. She could be reached by saipeck@um.edu.my.