NEW SQL STANDARD FOR OBJECT-RELATIONAL DATABASE APPLICATIONS

Eric Pardede and **J. Wenny Rahayu**
LATROBE UNIVERSITY, AUSTRALIA

David Taniar
MONASH UNIVERSITY, AUSTRALIA

In this paper we examine the impact of the newest generation of Structured Query Language (SQL), which is the standard language for Relational Model. The new standard of SQL4 adds powerful object-oriented data structures to Object-Relational Database (ORDB), which is designed and implemented on Relational Model, to be used in diverse applications. This paper offers some opportunities of ORDB usage in many emerging database technologies such as Statistical and Scientific Database, Web Database, Multimedia Database, etc. Its aim is to show how standardization's impact on the model that can be useful for many real world applications.

Structured Query Language (SQL) is used for database definition and manipulation [15]. It was originally developed for Relational Model, a data model that was very popular until the mid 90s. However, many people believe that this model does not have enough capability to capture the real world problems and needs to be enriched with OO concepts [12]. It brings a new era of *Object-Relational Database* (ORDB), which captures the semantic richness of OO Conceptual Model (OOCM) and the simplicity of Relational Model implementation.

In ORDB, the OOCM is transformed into Logical Model and is implemented into relations using Object-Relational DBMS (ORDBMS) (see Fig.1). The design phase for ORDB still follows the conventional relational model, which does not have the capability to model full OOCM. It is understandable since SQL as the design language at that stage did not preserve full OOCM features. To answer the problem, ANSI/ISO has introduced new SQL standards called SQL3 [7, 15]. At present they have been working on the prototype of the SQL4 that has even more extensions [16]. Now the ORDB transformation can be performed efficiently and thoroughly, we have the challenge to make it applicable for new database problems.

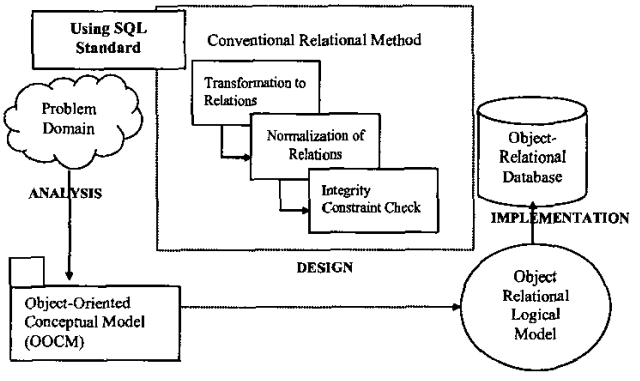


Figure 4. *Object-Relational Database*

0-7803-8172-6/03/$17.00 ©2003 IEEE.

The aims of this paper are to provide the taxonomy of data structures brought by SQL4 standard and to show how they can be used in ORDB for emerging database applications. It is important to emphasize that this paper shows challenges and opportunities as the impacts of new standards.

After this introduction, we show some background on SQL and ORDB in the first section. In the second section we classify new SQL data type, which enrich data structures in ORDB. In the third section we show how some database problems can be answered by ORDB with the new SQL standard. Finally our paper will be concluded in the fourth section.

Background

In this section we provide SQL historical information, from the introduction of the language as a standard to the development of the newest generation. It is followed by the background of the database model supported by new SQL, the ORDB.

Structured Query Language (SQL)

SQL was introduced in 1970 and has emerged as the standard language for Relational Database (RDB) [15]. The 1992 revision, SQL2, has been widely used by all RDBMS products. In 1993 an attempt to develop a new standard was started since the DBMS vendors had enhanced their existing relational products with OO features. The existing standard had become somewhat obsolete because it provided no support for OO features.

Many of the vendors created their own language extension of SQL to retrieve and manipulate data. For example, POSTGRES [22] provides an extended SQL called POSTQUEL query with the ability to capture the concept of abstract data type, inheritance, and object identity. Another example is Starburst [14, 21] that extends the relational algebra and supports user-defined operations and complex types.

Extension to SQL by vendors can be used for their own products. However, there was no standard that can be used by all ORDB users and was acceptable to all parties. Therefore, it is crucial to develop a new SQL that can capture all OO features in Relational Model. The new SQL is then developed and called SQL3 or SQL-99 [7].

SQL3 has been characterized as "OO SQL" and it has become the foundation for several ORDBMS such as Oracle8. Ironically, many believe that SQL2 will still be used in the future [5], since many researchers and practitioners still have unsettled arguments on many SQL3 issues. Therefore, the standardized body ANSI/ISO has started to review SQL3 and aims to release a new SQL version, SQL4, in few more years. At the time of writing, this version is still an ongoing work and no release date has been announced [16]. SQL4 has added some features to SQL3 and it has also reviewed some of its previous versions.

Object-Relational Database (ORDB)

The newest SQL version is aimed to facilitate a database model that was developed in the 90s, called Object-Relational Model. This model is developed to answer the limitation of Relational Model. Even though relational model can separate the logical and physical components of database model, it does not have the modelling capability to capture the semantics of complex applications such as encapsulation, different types of relationships, complex types, etc. Moreover, it gives little or no support for handling and manipulating the data stored in the tables, since the dynamic aspects of the model were largely ignored and left to the application programmer [4].

To answer the limitation of Relational Model, OO Model has become an intense topic of interest in computer science, due to the richness of the OO concepts, such as extensibility, information hiding, reusability, and inheritance [2, 4, 20]. They provide an excellent basis for modelling, because the object structures permit analysts and designers to focus on a problem at a high level of abstraction but with a resulting design that can be easily implemented. Since the last decade, more software has been written using the OO paradigm. Many prototypes as well as commercial OODBMS such as O2, Versant, POET, ONTOS, Objectivity, GemStone, and ObjectStore have been developed by both industrial and research laboratories around the world. [3, 11, 19, 23].

Nevertheless, OO Databases (OODB) are still not as widely used as RDB that rest on a firm formal foundation. [24] reports that the OODBMS market is a factor of 100 smaller in comparison to the RDBMS market, and it is expected that this figure will be maintained in the next decade. It is a fact that RDB still largely dominate the database community. RDBMS technology is considered mature, and has been the basis of a large number of applications around the world. It is also known that in reality about 95% of object-based development systems are still using RDBMS engine as its persistence mechanism [1]. However, the relational approach, when used to model real world problems, is not nearly strong enough to model all the different kinds of relationships, both static and dynamic. This also includes the fact that the relational model has a lack of

semantic features, an inability to represent knowledge other than simple facts, and an inability to represent complex structures and operations [12].

These reasons have stimulated the emergence of a new approach in the development of database systems, namely the Object-Relational approach. In general this approach is a method of combining both OO and relational approaches with the aim of incorporating the advantages and eliminating their drawbacks. It has significant benefits in the areas of semantic data modelling, since it captures more extensive static aspects of the domain and also the dynamic aspect. This rich semantics is lacking in the relational model. On the other hand, in the implementation of the data model, there are major strengths of the existing RDBMS that OODBMS does not have. These include the wide spread acceptance as well as the simplicity of the query processing.

ORDB transforms OO model into Object-Relational logical model before it is implemented in ORDBMS. The logical design and implementation are performed using conventional relational model design supported by SQL. With the existence of new data structures in SQL4, the ORDB transformation is now becoming more complete and therefore it should become more useful for many database applications. In the next section we will start classifying the new SQL4 data types for ORDB.

New SQL Extension Data Types

SQL4 classifies the data types into three main classes: predefined, constructed, and user-defined [15]. It has few extensions from SQL3, but there are a large number of additional data types from SQL2, which is a pure relational model language. Fig.2 illustrates the complete data types supported by current SQL.

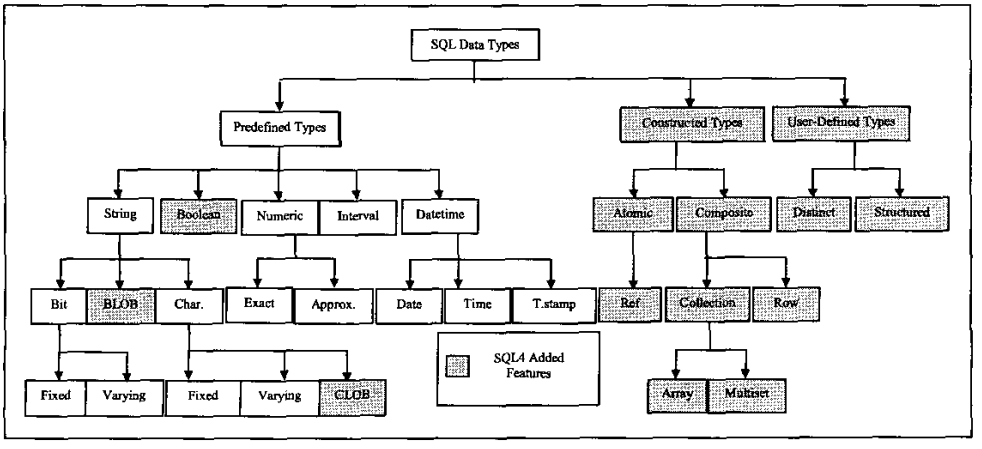


Figure 2. *SQL4 Data Types Classification*

In this section we focus on those that clearly affect the data structures. Its aim is to show how the new data structures can be useful in later section. In addition, we also show how ORDBMS products facilitate the additional data types [9, 10, 18]. It is crucial to include products, since the readiness of vendors is a big issue in implementing software standard including SQL. Many features have been deemed immature since the vendors were not able to have agreement on the semantic or syntax. On the other hand, different implementation of data types in ORDBMS products has also increased the importance of new SQL standard. At this stage, we encounter problems to transfer database from one ORDBMS to another because they have their own data type extension. With the same design following SQL, we can resolve this problem.

To illustrate the data types we will use a running example (see Fig.3) throughout the section. In each sub-section we add attributes implemented using one new SQL data type. We will use CREATE TABLE and CREATE TYPE as the basic SQL statements with the basic syntax as follows.

```
CREATE [OR REPLACE] TYPE <object schema>
   (attribute   attribute type, ....,
    attribute   attribute type);
```

```
CREATE TABLE <table schema>
    (attribute   attribute type CONSTRAINT attribute PRIMARY KEY, ....,
     attribute   attribute type);
```

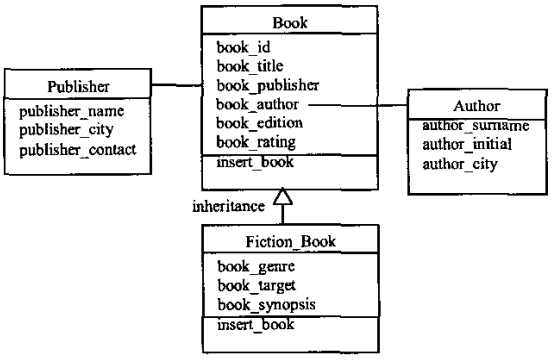


Figure 3. *An Example - Book*

Reference (REF) Type

REF type is a data type, whose value can be used to address a site holding another value. The site pointed to can be another constructed data type or user-defined type in a typed table. The value of this data type is called REF value and it is a unique strongly typed value. It can only contain value that point to a specified type and it does not have referential constraint attached to it.

The operator used for the data type is REF operator. It takes the REF value and returns the value held by the site it identifies. If for some reasons the site has been destroyed, the REF operator will return null value. The data type can also be used to obtain a value referenced by a REF value using DEREF operator. Furthermore, it can return a value acquired from invoking and SQL-invoked method.

The example below shows the use of REF type in tables. Notice that the `book_publisher` is a REF type that holds a value pointing to a user-defined type `Publisher`. Oracle and DB2 both implement the data type with the same syntax. Informix on the other hand, has not provided REF type among its extended type. Therefore, we need to use a referential key or *foreign key* (FK). In this case `book_publisher` of `Book` table is the FK to `Publisher` table.

```
SQL:
    CREATE TYPE Publisher
        (publisher_name CHARACTER VARYING(50),
         publisher_city CHARACTER VARYING(20),
         publisher_contact NUMBER);

    CREATE TABLE Book
        (book_id CHARACTER VARYING(5) CONSTRAINT book_book_id_pk PRIMARY KEY,
         book_title CHARACTER VARYING(50),
         book_publisher REF(Publisher));

Oracle and DB2:
    CREATE TABLE Book
        (book_id VARCHAR(5) PRIMARY KEY,
         book_title VARCHAR(50),
         book_publisher REF(Publisher));

Informix:
    CREATE TABLE Publisher
        (publisher_name VARCHAR(50),
         publisher_city VARCHAR(20),
         publisher_contact NUMBER);
```

```
CREATE TABLE Book
    (book_id VARCHAR(5) PRIMARY KEY,
     book_title VARCHAR(50),
     book_publisher VARCHAR(50),
     FOREIGN KEY book_publisher REFERENCES Publisher);
```

Row Type

Row type is constructed data type that contains a sequence of attribute names and their data types. Since it is defined like a flat table, a row type inside a table resembles a nested table. Furthermore, it is possible in current SQL to have varying levels of nesting. It becomes a powerful means to capture real world problems since they can rarely be represented by a simple flat table.

SQL provides named and unnamed row types. The named row type is quite similar to user-defined type, where we create the type before using it in a relation. The only difference is that named row type has no methods associated with it.

Values of different row types are assignable and comparable to each other if and only if both row types have the same degree and the attributes of both row types in every ordinal is assignable and comparable to each other.

The example of row type is shown by the following example. For SQL we show the named row type. The attribute `book_author` is a row type, which in our example is included in a type attribute. It can only be included as table attribute. Oracle has not provided row type, but we can use another type called nesting for similar purpose. In the following example `book_author` is stored as nested table. Informix provides row type exactly like it is standardized by SQL. DB2 has not implemented the type although we can use a user-defined-type without methods to suit the purpose.

```
SQL:
    CREATE ROW TYPE Author_Row_Type
        (author_surname CHARACTER_VARYING(20),
         author_initial CHAR(3),
         author_city CHARACTER VARYING(15)));

    CREATE TABLE Book
        (book_id CHARACTER VARYING(5) CONSTRAINT book_book_id_pk PRIMARY KEY,
         book_title CHARACTER VARYING(50),
         book_publisher REF(Publisher),
         book_author AUTHOR_ROW_TYPE);

Oracle:
    CREATE TYPE Author_Table_Type AS OBJECT
        (author_surname VARCHAR(20),
         author_initial CHAR(1),
         author_city VARCHAR(15));

    CREATE TABLE Book
        (book_id VARCHAR(5) PRIMARY KEY,
         book_title VARCHAR(50),
         book_publisher REF(Publisher),
         book_author AUTHOR_TABLE_TYPE)
         NESTED TABLE book_author AS Book_Author_Tab;

Informix:
    CREATE TABLE Book
        (book_id VARCHAR(5) PRIMARY KEY,
         book_title VARCHAR(50),
         book_publisher VARCHAR(50),
         book_author ROW (author_surname VARCHAR(20),
                          author_initial CHAR(3),
                          author_city VARCHAR(15)),
         FOREIGN KEY book_publisher REFERENCES Publisher));

DB2:
    CREATE TYPE Author_Type
        (author_surname VARCHAR(20),
         author_initial CHAR(3),
```

```
    author_city VARCHAR(15)));

CREATE TABLE Book
    (book_id VARCHAR(5) PRIMARY KEY,
     book_title VARCHAR(50),
     book_publisher REF(Publisher),
     book_author AUTHOR_TYPE);
```

Array Type

Array is a constructed data type that can hold composite elements of similar type. The number of elements in a collection is called the cardinality. The element data type can be a predefined type, another constructed type, or a user-defined type. A collection constructor is identified by the keyword that determines the type of the collection and the element data type.

Array provides an ordering semantic of the elements. It specifies the maximum cardinality, although it can contain elements less than that number. It is more flexible than array in many programming languages, which forces the users to have elements as many as the maximum cardinality. However, it is less flexible than a list, another type of collection type in OOCM, which enables users to have an unlimited number of elements. As array captures ordering semantic, it requires the ordinal position information of each element. This position is called an *index*. Index enables users to perform an operation in a specific element of an array.

The following example shows the array in a table. Attribute `book_edition` is an array since it has to be implemented in order without duplication. The element data type is INTEGER, although we can also have an array of extended data type. Oracle provides array type, while Informix implements list. On the other hand DB2 does not include list nor array type for its extended data type.

```
SQL:
    CREATE TABLE Book
        (book_id CHARACTER VARYING(5) CONSTRAINT book_book_id_pk PRIMARY KEY,
         book_title CHARACTER VARYING(50),
         book_publisher REF(Publisher),
         book_author AUTHOR_ROW_TYPE,
         book_edition INTEGER ARRAY[10]);

Oracle:
    CREATE TABLE Book
        (book_id VARCHAR(5) PRIMARY KEY,
         book_title VARCHAR(50),
         book_publisher REF(Publisher),
         book_author AUTHOR_TABLE_TYPE,
         book_edition VARRAY OF INTEGER(10))
         NESTED TABLE book_author AS Book_Author_Tab;

Informix:
    CREATE TABLE Book
        (book_id VARCHAR(5) PRIMARY KEY,
         book_title VARCHAR(50),
         book_publisher VARCHAR(50),
         book_author ROW (author_surname VARCHAR(20),
                          author_initial CHAR(1),
                          author_city VARCHAR(15)),
         book_edition LIST(INTEGER),
         FOREIGN KEY book_publisher REFERENCES Publisher));
```

Multiset Type

Multiset is the newest collection type added to SQL. It is an opposite of a list, because it contains elements that can be duplicated and do not need an ordering semantic. In OOCM this collection type is known as bag. Since a multiset does not require ordering, it does not have ordinal position or index. Nevertheless, the cardinality of multiset is still important in case users perform an assignment or a comparison between multiset.

The following example shows multiset in SQL table. Attribute `book_rating` is implemented as multiset because it is assumed that a book can have same rating several times. In addition, we do not really concern about the order of the rating given. Since this data type is a fairly new addition in SQL, not many ORDBMS provide multiset type. Among our sample, Informix is the only product that has provided multiset type. The syntax is similar to the syntax of list.

```
SQL:
   CREATE TABLE Book
       (book_id CHARACTER VARYING(5) CONSTRAINT book_book_id_pk PRIMARY KEY,
        book_title CHARACTER VARYING(50),
        book_publisher REF(Publisher),
        book_author AUTHOR_ROW_TYPE,
        book_edition INTEGER ARRAY[10],
        book_rating INTEGER MULTISET);

Informix:
   CREATE TABLE Book
       (book_id VARCHAR(5) PRIMARY KEY,
        book_title VARCHAR(50),
        book_publisher VARCHAR(50),
        book_author ROW (author_surname VARCHAR(20),
                         author_initial CHAR(1),
                         author_city VARCHAR(15)),
        book_edition LIST(INTEGER),
        book_rating MULTISET(INTEGER),
        FOREIGN KEY book_publisher REFERENCES Publisher));
```

Structured User-Defined Type (UDT)

UDT comprises a number of attributes and routines (see Fig.4). It enables users to define and to support the storage and manipulation of complex structure. The internal structure is encapsulated, so they are not accessible directly to the users. Access to instances or attribute is done through their routines.

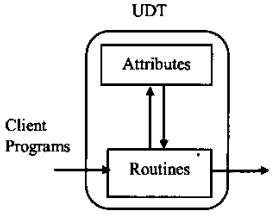


Figure 4. *Encapsulation of Attributes and Routines*

A UDT can be instantiated into instances. Instances are generated by the system-provided constructor. The manipulation of UDT attributes or methods is done through the instance. In SQL, we can have a UDT that cannot be instantiated indicated by NOT INSTANTIABLE keyword. For such type we usually do instantiation through the sub type.

Sub-type is the result of UDT inheritance structure. Inheritance enables users to create a type under an existing type. The lower type inherits the attributes and behaviours of all types above its hierarchies. The inheritance however, can be limited by overriding some properties of the super types.

SQL4 supports type and table inheritance. Type inheritance is the declaration of a UDT under another type. This type will be called the sub type and it will inherit all the fields of its super type. The type hierarchy has pushed the complexity into the data type definitions, and made the table structure simple and easy to use. Type inheritance creates relationships among the structure of the tables, but the tables remain independent in terms of the data they contain. Table inheritance on the other hand provides a different way of linking the table contents. It turns the table much closer to like object classes since the tables themselves have a parallel hierarchy. In this inheritance, when a table is declared under its super table, it inherits more than just column structure. They include the keys, the integrity constraints, triggers, indexes, etc. In table inheritance, the sub tables will be treated like a nested collection of rows and a query on a table applies to all rows included in the set.

The following example shows how UDT is provided by SQL. We use the same example as before, however we now work on the type instead of the table. To create sub types, we have to declare the higher type as NOT FINAL such as in `book_type`. If we decide it to be at the end of an inheritance structure, we can declare it FINAL, as in `fiction_type`.

```
SQL:
```

```
CREATE TYPE Book_Type
    (book_id CHARACTER VARYING(5),
     book_title CHARACTER VARYING(50),
     book_publisher REF(Publisher),
     book_author AUTHOR_ROW_TYPE,
     book_edition INTEGER ARRAY[10],
     book_rating INTEGER MULTISET,

     PROCEDURE Insert_Book(
        new_book_id IN VARCHAR2,
        new_book_title IN VARCHAR2);

       BEGIN
          INSERT INTO Book (book_id, book_title)
          VALUES(new_book_id, new_book_title);
       END;

)INSTANTIABLE NOT FINAL/

CREATE TYPE Fiction_Type UNDER Book_Type
    (book_genre CHARACTER VARYING(20),
     book_target CHARACTER VARYING(20),
     book_synopsis CHARACTER VARYING(200),

     OVERRIDING PROCEDURE Insert_Book(
        new_book_id IN VARCHAR2,
        new_book_title IN VARCHAR2);

       BEGIN
          . . .
       END;

)INSTANTIABLE FINAL /
```

Oracle implements UDT the same as the SQL standard. It can take form as *value UDT* or *object UDT*. The former means that the type is bound to a table as column object, like `book_author` attribute in the following example. The latter on the other hand is appeared in object table as special kind of table, like it is shown by table `Book of book_type`. For object UDT, Oracle provides every row object a unique object identifier. DB2 supports structured type almost the same as Oracle. However, in DB2 we cannot have value UDT, where the type is implemented as a column of a table or a view.

Structured type in Informix can be associated with opaque data type. This data type stores a single value, which is the memory contents of the data structure, implemented as C structure and C routines. Since the content of the data type is the memory, we need to specify the memory allocated to the data type. To interact with the internal structure, the database server uses the routines. This data type practices encapsulation of structured type in SQL, however, the implementation of using a structure outside the database is very different with other DBMS.

```
Oracle:
   CREATE TYPE Author_Type AS OBJECT
       (author_surname VARCHAR(20),
        author_initial CHAR(1),
        author_city VARCHAR(15)));

   CREATE TYPE Book_Type
       (book_id VARCHAR(5),
        book_title VARCHAR(50),
        book_publisher REF(Publisher),
        book_author AUTHOR_TYPE,
        book_edition VARRAY OF INTEGER(10),

        MEMBER PROCEDURE Insert_Book(
           new_book_id IN VARCHAR2,
           new_book_title IN VARCHAR2))/

   CREATE OR REPLACE TYPE BODY Book_Type AS
```

```
        MEMBER PROCEDURE Insert_Book(
            new_book_id IN VARCHAR2,
            new_book_title IN VARCHAR2) IS

           BEGIN
              INSERT INTO Book (book_id, book_title)
              VALUES (new_book_id, new_book_title);
          END Insert_Book;
        END;/

    CREATE TABLE Book OF Book_Type
        (book_id NOT NULL PRIMARY KEY);

Informix:
    CREATE OPAQUE TYPE Book_Opaque_Type
    (INTERNALLENGTH=VARIABLE,MAXLEN=4096);
```

New SQL for Emerging Applications

New problem domains in database usage have emerged along with the introduction of any real world problem. Nowadays the real world problems become more complex and to represent them we need sophisticated data model. Many database designers do not see ORDB as a data model that can be used for new problem domains. The main reason is a bad presumption on the capability of Relational Model as its predecessor.

In this section we show some emerging database applications. We also give direction on how ORDB with new SQL standard can be used to model these new applications. ORDB will not be able to answer all requirements for all applications, but at least the new standard has provided more capabilities to answer the requirements that could not be captured before.

Statistical and Scientific Database (SSDB)

SSDB emerged as one big research area in the 1980s. The applications are complex because they involve complex data and procedures. The data usually contains large, static, and a lot of null values. In addition the queries often include aggregation and sampling.

Traditional DBMS will impose some difficulties, if used for SSDB application. For SSDB statistical aspect there are two reasons. First, the data redundancy is profound and therefore the organization of data into tuples might be inefficient. Second, traditional DBMS has a lack of functionality. They only have simple aggregate functions such as SUM, AVERAGE, etc.

As an example, we use a statistical problem in an agricultural department. The department wants to store the data of production and consumption of different types of dairy product from different regions in different months. Using traditional DBMS we need composite keys of different categories and we might end up with lots of redundancy. SSDB usually implements this problem type in matrix. Now with new SQL standard in ORDB we can also develop matrix in nested row (see fig. 5). In addition, the methods of UDT can be used to answer the lack functionality of traditional DBMS.

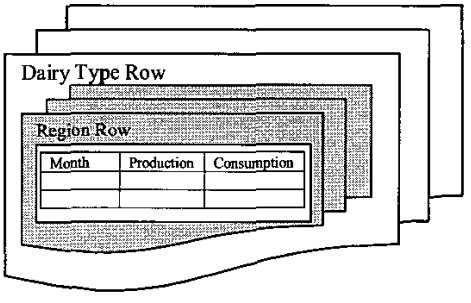


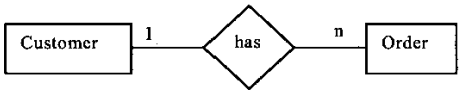
Figure 5. *Nested Rows for SSDB applications*

Web Database

Web database has become a major research area, especially since the introduction of *Extensible Markup Language* (XML) as the standard language for web and a tool for exchanging semi-structured data between

various systems and databases on the Internet. It is a very versatile language, since it can be extended easily using user-defined tags to capture complex data structure and relationship such as inheritance and association. XML now has provided its own database. However, many users still question the performance and simplicity of the database. That is why users still favor using existing database such as RDB and ORDB. For the second option, we then need further work to integrate the database into XML documents.

New SQL standard data types now have captured more complex data structures and relationships. Therefore, more integration between ORDB data structure and XML is required. For example, REF type in new SQL can be represented easily in XML so that we can reduce or even eliminate the old and expensive join process. In fig 6, we show a very simple association relationship between a type `Customer` and `Order`, and how we intermingle new SQL and XML.




```
XML Schema:
<xs:sequence>
    <xs:element name = "order" type= "
      orderType" minOccurs= "1" max Occurs =
      "unbounded" / >
</xs:sequence>
<xs:complexType name = "orderType">
    <xs:attribute name ="orderID"
      type="xs:integer"/>
    <xs:attribute name = "orderDate" type=
      "xs:date"/>
</xs:complexType>

New SQL:
CREATE TYPE ORDER
(order_id           varchar2(10),
 order_date         date,
 order_customer     REF customer_type)
```

Figure 6. *Association Relationship in XML and SQL*

Despite further work brought by SQL4 toward XML transformation, the standard has also given a significant benefit. The nested structure in SQL4 such as row and collection type allow user to map XML documents in more natural way. The element hierarchy of Document Type Definition (DTD) on XML can be directly mapped onto nested attributes in ORDB [8, 13, 26]. It is more direct than the transformation to relational database [6, 25].

At present, ANSI/ISO has a special committee working on SQL-XML integration [17]. The work that is planned to be released with SQL4, is called SQL/XML. It includes how to map between SQL data types and XML, how to query XML documents using SQL-based syntax, and related issues.

Multimedia Database

Multimedia Database involves accessing and manipulating stored information of different media such as text, graphic, animation, audio, image, video, and mixed data. Due to its large size, real-time and raw nature, we cannot treat multimedia database like traditional database. The diverse characteristic of various media objects has created the complexity. This area is one of the most progressive research areas in database since it affects many disciplines including education, marketing, retailing, entertainment, travel, etc.

With the support of new SQL, we can accommodate multimedia in ORDB. Not only storing them as Large Object, but even use constructed data type. For example we can store an image as a multiset of pixel and include it as an attribute of a UDT like it is shown in the data definition language below. Now we will be able to query, for example, the detail of a person who has certain eye color by searching the pixel color of the attribute *Photo*. Of course, further research has to be conducted on how to implement this idea efficiently.

```
CREATE TYPE Face_Type AS MULTISET(BIT);

CREATE TYPE Person_Type AS OBJECT
    (Surname CHARACTER VARYING (20),
     First_Name CARACTER VARYING (20),
     Age INTEGER,
```

```
Photo FACE_TYPE);
```

Beside the one explained in this section, SQL also has a special section on multimedia developed by a special committee. SQL/MM is aimed to develop a special set of multimedia library contained with all multi media data types. It is more powerful than using UDT like the previous example. Standard multimedia data type integrates cleanly and completely with the database, whereas UDT data type for multimedia may not be as seamlessly integrated and reusable [15]. At present, the SQL/MM only deals with full text media. Nevertheless, works for other media are still ongoing and hopefully can be included in SQL4 release [15].

Genome Data Management

Genome is a total complex genetic information that can be obtained about an entity. For example in a human being there are up to 300,000 genes with an estimated up to 4 billions nucleotides [5]. Identifying and sequencing this information are very important since it can bring answer to questions regarding genetics, medicine, anthropology, agriculture, etc.

Managing this data is however very challenging. The data is more complex, compared to other applications, since the amount and the range of variability are massive. Another problem is the rapid change of database schema in this area and the subjective representation nature of the data. It is even worse since the queries made by users, in this case biologists, can be very complex. So far there are several major genome-related databases such as GenBank, GDB, OMIM, EcoCyc, etc. Each of them has different content and also different technology.

GDB for example, has been developed using relational model. It associates a piece of information with a particular location of human genome. The mapping is a complicated task, not only because of the variability of the data source, but also it is not possible to map the data into nucleotide level. It creates a problem, since we do not know whether a particular information has been associated to a previous gen or similar problem.

ORDB with new data structures can improve the mapping process. We know that new SQL has provided array or list data type. It will be possible to store genome as multi level list. A genome will have sub-genome as its sub-list based on the proximity analysis. It can be done repetitiously down to nucleotide level (see Fig. 7). Now, if we have a new piece of information we do not have to map it to all genome, but only to groups that have similar information associated with them.

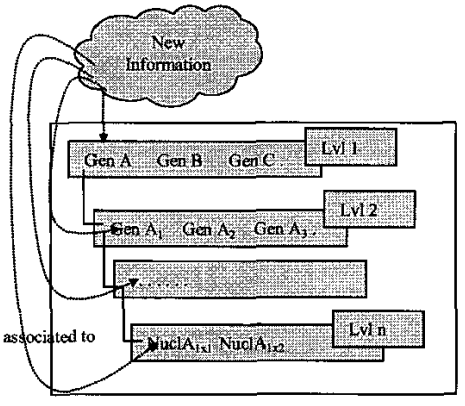


Figure 7. *Multilevel List for Genome Database*

Geographic Information System

GIS is used to collect, model, store, and analyse information describing physical properties of the geographical world [5]. It has two types of data: spatial and non-spatial data. Spatial data includes maps, digital images, regions, roads, and other physical data, while non-spatial data includes economic data, sales information, counts, etc. Using GIS we can retrieve information such as the air quality in a geographical area in different time span, etc..

GIS can be classified into different applications. We use one of them to illustrate the potential use of ORDB with new SQL standard. Geographical Object Application is one GIS application that has physical domain as an object of interest. For example we have physical domain `Power Plant` to determine the power consumption in different regions. With rich semantics of new SQL, a physical domain of GIS can be implemented as a structured type. In this type we separate the spatial data from the non-spatial data and handle the latter purely in ORDB. Fig.8 shows that `Power Plant` type has non-spatial attributes that might include year, month, consumption, etc. It also has methods for operations applicable only to that domain. Of course further research has to be done in this application to benefit from existing advances in ORDB technology. The further research includes new client-server architecture, new modeling and also integration technique between spatial and non-spatial data.

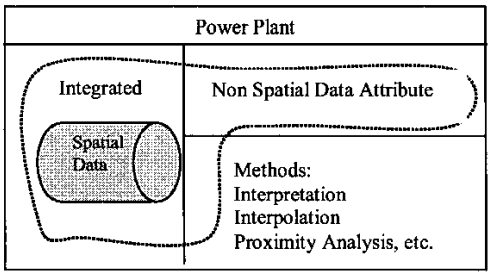


Figure 8. *Separated Data and Methods in GIS*

Conclusion and Future Works

A standard can impact upon a formal model and furthermore provides opportunities for practical applications. In this paper, we have shown how SQL4 standard has enriched Object-Relational Model with many new data structures. SQL4 also tries to reunite different design and implementation of vendor specific ORDB into one common standard. We foresee that many emerging database domains and applications can now derive benefit from ORDB supported by new SQL4.

This paper classifies SQL4 data types and gives suggestions on how they can be used for some database problems. As we only provide ideas, the paper challenges further research on the design and implementation of the new data types in general ORDB and also in the specific aforementioned database problems.

Beside the usage of ORDB in new application, a further research can be done to new SQL support on XML and Multimedia. In addition, research to improve the existing SQL for further existence is also open.

References

[1] S.W. Ambler, "Mapping Objects to Relational Databases", *Building Object Applications That Work*, SIGS Books, 1997

[2] G. Booch, *Object-Oriented Analysis and Design with Applications*, 2nd Ed, Benjamin/Cummings, 1994

[3] O. Deux, "The Story of O2", *TKDE* 2(1), IEEE Computer Society, 1990, pp. 91-108

[4] T.S. Dillon and P.L. Tan, *Object-Oriented Conceptual Model*, Prentice Hall, 1993

[5] R. Elmasri and S.B. Navathe, *Fundamentals of Database Systems*, 3rd Ed., Addison Wesley, 2000

[6] D. Florescu and D. Kossmann, "Storing and Querying XML Data using an RDMBS", *IEEE Data Engineering Bulletin* 22(3), 1999, pp.27-34 (1999)

[7] P. Fortier, *SQL3 Implementing the SQL Foundation Standard*, McGraw Hill, 1999

[8] W-S. Han, K-H. Lee, and B S. Lee, "An XML Storage System for Object-Oriented/Object-Relational DBMSs", *Journal of Object Technology* 2(1), pp.113-126 (2003)

[9] IBM DB2, http://www.software.ibm.com/cgi-bin/db2www/library/, 2003

[10] Informix, http://www-3.ibm.com/software/data/informix/, 2003

[11] W. Kim, *Introduction to Object-Oriented Databases*, The MIT Press, 1990

[12] W. Kim, *Modern Database Systems*, Addison-Wesley, 1995

[13] M. Klettke and H. Meyer, "XML and Object-Relational Database Systems - Enhancing Structural Mappings Based on Statistics", *WebDB*, Springer-Verlag, 2000, pp.151-170

[14] B.G. Lindsay and L.M. Haas, "Extensibility in the Starburst Experimental Database System", *IBM Symposium: Database Systems of the 90s*, Springer-Verlag, 1990, pp. 217-248

[15] J. Melton, A.R. Simon and J. Gray, *SQL: 1999 - Understanding Relational Language Components*, Morgan Kauffman, 2001

[16] J. Melton, (ed.), *Database Language SQL – Part 2 Foundation,* ISO-ANSI WD 9075-2, ISO, Working Group WG3, August 2002

[17] J. Melton, (ed.), *Database Language SQL – Part 14 XML-Related Specifications (SQL/XML),* ISO-ANSI WD 9075-14, ISO, Working Group WG3, August 2002

[18] Oracle, www.oracle.com, 2003

[19] J. Robie, J. Lapp and D. Achach, "XML Query Language (XQL)", *The Query Languages Workshop*, 1998

[20] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, 1991

[21] P.M. Schwarz, W. Chang, J.C. Freytag, G.M. Lohman, J. McPherson, C. Mohan, and H. Pirahesh, "Extensibility in the Starburst Database System", *OODBS* 1986, IEEE Computer Society, 1986, pp. 85-92

[22] M. Stonebraker, "Object Management in Postgres Using Procedures", *OODBS* 1986, IEEE Computer Society, 1986, pp. 66-72

[23] M. Stonebraker, "The Postgres DBMS", *SIGMOD* 1990, ACM Press, 1990, pp. 394

[24] M. Stonebraker and D. Moore, *Object-Relational DBMSs The next great wave*, Morgan Kaufmann, 1996

[25] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D.J. DeWitt, and J.F. Naughton. "Relational Databases for Querying XML Documents: Limitations and Opportunities", *VLDB* 1999, Morgan-Kauffman, pp.302-314

[26] T. Shimura, M. Yoshikawa, and S. Uemura, "Storage and Retrieval of XML Documents Using Object-Relational Databases", *DEXA* 1999, Springer-Verlag, 1999, pp. 206-217