"User defined types and nested tables in object relational databases"

Bernadette-Marie Byrne*
Department of Computing,
Oxford Brookes University, UK
bbyrne@brookes.ac.uk,
tel 01865484521

Mary Garvey,
School of Computing and IT,
University of Wolverhampton, UK
m.garvey@wlv.ac.uk
tel : 01902 321483

Track: Technical Issues in Information Systems

Abstract

There has been much research and work into incorporating objects into databases with a number of object databases being developed in the 1980s and 1990s. During the 1990s the concept of object relational databases became popular, with object extensions to the relational model. As a result, several relational databases have added such extensions.

There has been little in the way of formal evaluation of object relational extensions to commercial database systems. In this work an airline flight logging system, a real-world database application, was taken and a database developed using a regular relational database and again using object relational extensions, allowing the evaluation of the relational extensions.

Keywords:

Relational Databases, Object-Relational Databases, User-defined Types, Nested tables.

Introduction

The relational model was first proposed by Codd (1970). The first Database Management System (DBMS) based on his ideas was an IBM product released in 1980. The Relational DBMS (RDBMS) grew in popularity despite performance problems compared with the existing navigational databases of the time and, over the last 25 years, has come to dominate the database market. With the rise in popularity of object oriented programming in the late 1980s developers began to propose alternatives and extensions to RDBMS in order to include support for objects. The two main proposals were for object databases and object-relational (OR) databases; the latter adding object support to the existing relational model. Initially, object databases were seen as the best way forward and several commercial databases were produced which are still in use today. However, the expected success of object databases did not materialise and they are no longer seen as the best solution for incorporating objects into databases.

There has since been much agreement among database experts that Object-Relational databases should be the way forward, even if there is disagreement on the form it should take. Carey & DeWitt (1996) predicted that OR databases would soon become dominant. There has been, however, little sign of the expected adoption of OR in the IT industry, although there have been a number of small-scale DBMSs developed with OR features. One of the advantages of the ORDBMS is that it is usually compatible with existing relational databases and therefore the new technology is easy for users to adopt without major investment and recoding of existing applications.

One of the first languages to incorporate many of the ideas of object-oriented (OO) programming was SIMULA in the 1960s (Stefik & Bobrow, 1984), but it was in the 1990s that OO programming became widespread. With the increasing importance of the Internet to applications and the emergence of new web-oriented languages such as Java and the .NET framework, OO programming has become the norm. One of the problems of object-oriented (OO) applications is that of providing persistence of objects – objects need to survive beyond the running of an application (Garvey, Jackson, & Roberts, 2002). Much work was done on developing "object-oriented databases" (OODB) with three systems in particular being developed: Gemstone, Vbase and Orion. In addition to the need for persistence of objects, Stonebraker et al. (1990) assert that the relational model does not adequately support many applications, especially non-business data processing applications which include different types of media, such as image video and complex objects, although many data processing systems are also not well supported. A new model of database system was required, the so-called "third generation" database system based on object-oriented concepts.

There has been an abundance of papers defining systems based on object-oriented concepts during the 1980s such as Albano, Cardelli & Orsini (1985) and Carey & DeWitt (1988). A consideration of the features of both object-oriented systems and database management systems led to the definition of the two leading object-oriented database manifestos. The first is often referred to as the Atkinson manifesto, which comes from an object-oriented programming language (OOPL) background and was one of the first attempts at defining an object-oriented database (Atkinson et al., 1989). This outlined a number of "mandatory" features of an object-oriented database system, including:

- Support for complex objects
- Object Identity
- Encapsulation
- Support for types or classes
- Class or Type Hierarchies
- Overriding, Overloading and Late Binding
- Computational Completeness
- Extensibility
- Persistence
- Secondary Storage Management
- Concurrency
- Recovery
- Ad hoc query facility

A standard for OO database systems, called the Object Data Standard, was produced in 1993 by the Object Database management Group (ODMG), initially called ODMG-93 (Cattell et al., 1993), with later versions being ODMG 2.0 (Cattell et al., 1997) and 3.0 (Cattell et al., 2000). The standard included Object Data Language (ODL), Object Query Language (OQL) and programming interfaces for C++ and Java (Carey & DeWitt, 1996).

Michael Stonebraker and others came up with an alternative proposal for a different kind of OODB based on the relational model, known as the *"Third-Generation Database System Manifesto"* (Stonebraker et al., 1990). The type of system proposed was initially known as an "extended relational" database system, but is now known as an "object relational" (OR) database system. The first object-relational DBMS was POSTGRES in the late 1980s and is described in Rowe and Stonebraker (1987), but a formal proposal for the object relational database model was not produced until the 1990s (Stonebraker & Moore, 1996), (Stonebraker, Brown, & Moore, 1999). This proposal purports to be very different to Atkinson et al., yet it covers much of the same ground. However, there are certain fundamental differences, Stonebraker et al. (1990) believes that any third generation DBMS should subsume second-generation systems. They believe that an application should only (rather than optionally) access the database through a query language and that procedural, programmatic access to data is not desirable. They highlight the desirability of *“location independence”*, which is essentially physical data independence. This is particularly important where pointers are used: the pointers should point to a unique identifier (an object ID), which is independent of physical location. They also suggest that SQL is a *“Reasonable candidate”* for the new functions that they detail – they suggest extending SQL for OODBMS, rather than developing a new language, although this is partly for commercial reasons concerning the popularity of SQL.

In summary, Stonebraker et al’s proposal takes the main features of Atkinson et al but sees them as an extension of the relational model, with access to the data exclusively through an extended version of SQL.

C J Date (2000) criticises current implementations of OR databases. He refers to *“Two Great Blunders”* which many vendors of Object Relational Databases have made. The first great blunder he describes as equating relations with object classes, when in fact domains should be equated with object classes and the second great blunder is mixing pointers with relations.

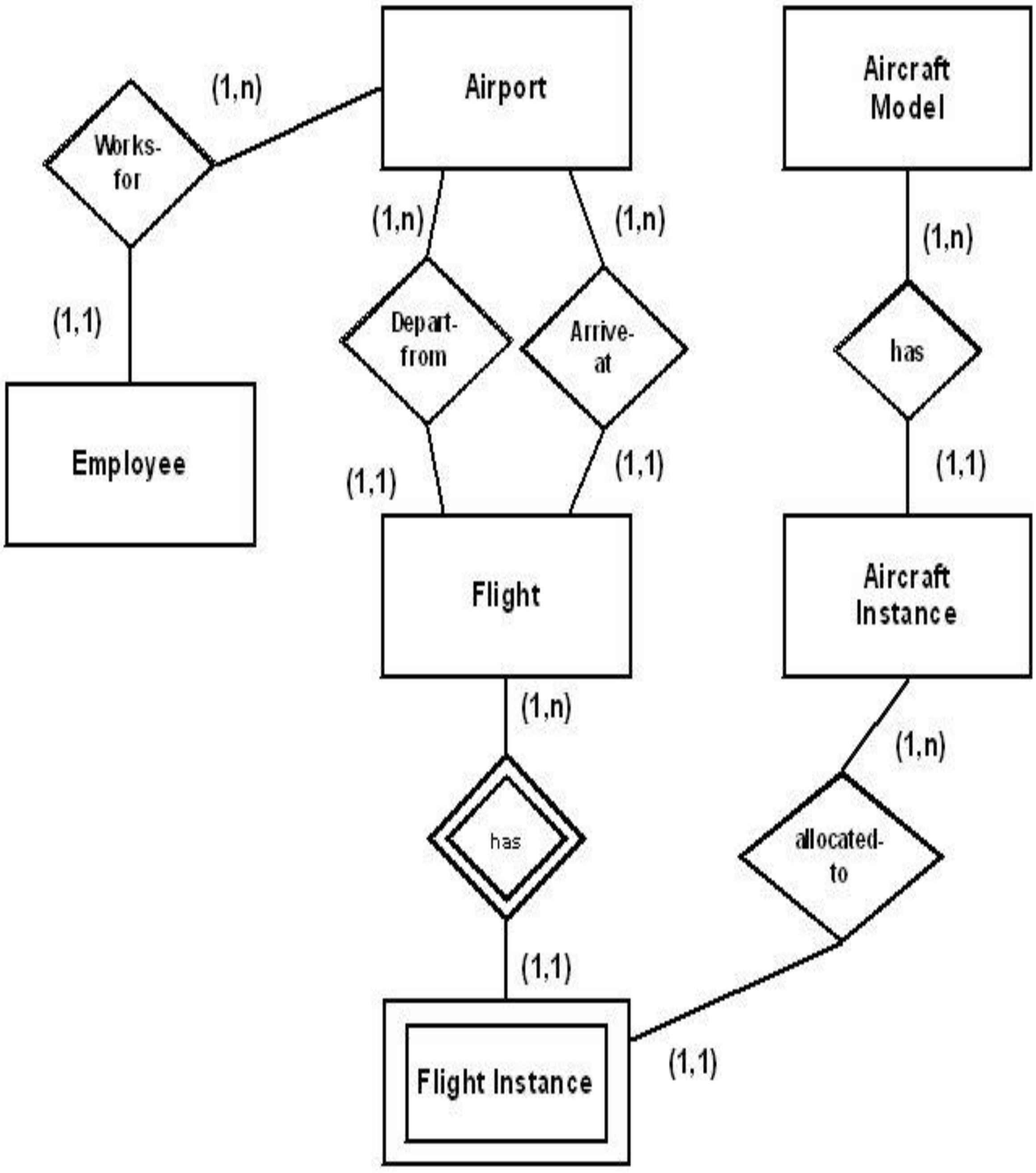
Implementation

The work described in this paper is based on a project that investigated object relational features (Millichamp, 2004). This paper considers two of these features, firstly User-Defined Types and secondly the use of Nested tables. The use of subtypes and the ‘ref’ structure will be the subject of a further paper. Figure 1 shows part of the entity relationship diagram for the Airline system which was implemented firstly using a standard relational database and again using some of the OR features.

User-Defined Types

A user-defined type (UDT) is a datatype used as an alternative to the DBMSs built-in types such as varchar, number, date etc. In practice it consists of one or more attributes, each of which can be of a built-in type, or another UDT.

Figure 1: Airline System



For example a ‘contact’ type was created with four attributes: tel1, tel2, fax and mobile. Each of these attributes was of telephone type. The telephone type consists of three attributes: int_code, local_code and number, each of type varchar2, with a character datatype being chosen rather than an integer to allow spaces and leading zeros in the telephone numbers. This creates a structure illustrated in table 1 below.

Contact	Tel1	Int_code
		Local_code
		Num
	Tel2	Int_code
		Local_code
		Num
	Fax	Int_code
		Local_code
		Num
	Mobile	Int_code
		Local_code
		Num

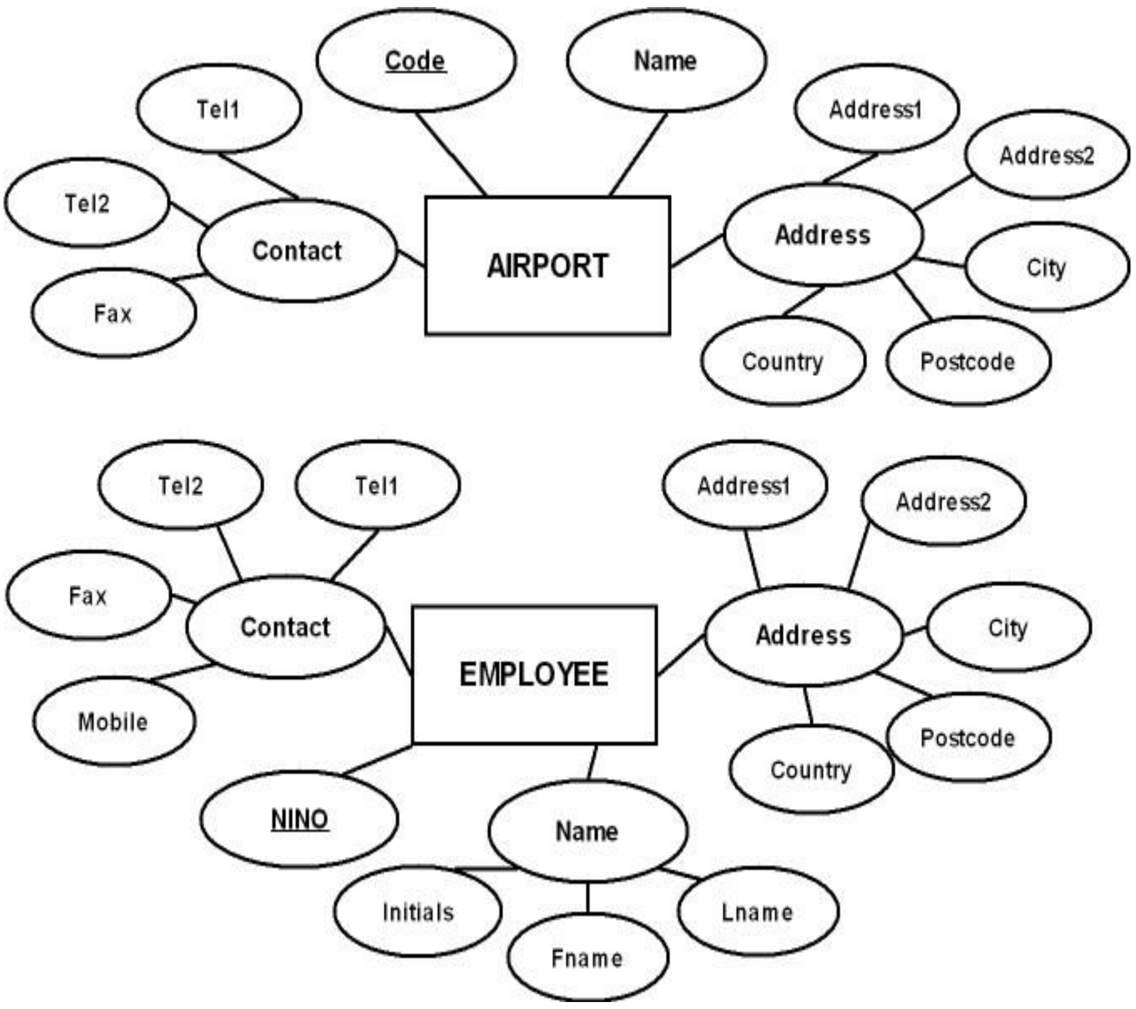
Table 1: Contact structure

The use of a UDT allows the construction of complex data structures. It also allows object types to be used in different tables – the contact type for example was used in four different tables in the completed system.

A type is created using the `Create Type` statement. There are different naming conventions for types, but the most common is to add the suffix ‘_T’ to the type name, such as ‘CONTACT_T’ for the contact type.

The EER diagram shows composite attributes for name, contact numbers and address in the Employee and Airport entities, as shown below. These can easily be mapped to UDTs and one UDT was created for each composite attribute. The composite attributes are shown in Figure 2 below.

Figure 2: Examples of a Composite Attribute



Additionally, a type was created for telephone numbers, with attributes for the international code, local STD code and the number itself. This allows the user to extract whichever part of the number they require and perhaps create a custom format from a combination of codes and number.

The contact type has attributes of the PHONE_T type. It is possible to have unlimited levels of nested types in a hierarchy.

The final types created were:

```
CREATE TYPE Name_t AS OBJECT ( /* person's name */
        lname       VARCHAR(20),
        fname       VARCHAR(20),
        initials    VARCHAR(5))
/
CREATE TYPE Address_t AS OBJECT (/* postal address */
        address1    VARCHAR(20),
        address2    VARCHAR(20),
        city        VARCHAR(20),
        postcode    VARCHAR(10),
        country         VARCHAR(20))
/
CREATE TYPE Phone_t AS OBJECT (/* phone numbers */
        init_code       VARCHAR(5),
        local_code      VARCHAR(10),
        num             VARCHAR(10))
/
CREATE TYPE Contact_t AS OBJECT (/* contact phone numbers */
        tel1            PHONE_T,
        tel2            PHONE_T,
        fax             PHONE_T,
        mobile          PHONE_T)
/
```

The use of UDTs was one of the most successful of the new features. It allows complex structures to be created, such as the contact type and reflects the requirements of OO applications. The `CREATE TYPE` statement is used to create the specification of a user defined type. To use the new type in a database, it can either form the basis of a table, or the type can be used as if it were a built-in datatype, for example, the following defines the AIRPORT and EMPLOYEE tables:

```
CREATE TABLE Airport (
        code            NUMBER(4) PRIMARY KEY,
        name            VARCHAR(10),
        contact         CONTACT_T,
        address         ADDRESS_T);

CREATE TABLE Employee (
        nino            NUMBER(4) PRIMARY KEY,
        name            NAME_T,
        address         ADDRESS_T,
        contact         CONTACT_T,
        worksFor        NUMBER(4) REFERENCES Airport(code));
```

By equating a UDT as an object class, it avoids Date's first great blunder. It provides a reusable structure that can simplify application development whether OO is used or not and allows queries where the structure of the object is reflected in the query; for example:-

```
SELECT E.Contact.Tel1.num FROM Employee e;
```

One of the biggest difficulties in using UDTs and row types is that there are major difficulties in altering types. For example, a PERSON_T type may have the attributes firstname, lastname, age, but it is subsequently decided that it would be better to use the date of birth (DOB) rather than the age. Any tables where are dependent on the PERSON_T type will be affected. However, it was found that it was difficult to make changes to a type, once other objects, such as another type, or table, made any reference to it. The only reliable way to alter a type was to backup any dependent tables and then drop the tables and the type. The new type and tables can then be created and the data copied from the back up tables. This is not a simple thing to do and carries a real risk of losing data if not carefully managed. It is also extremely time-consuming.

Queries with UDTs

Creating queries highlights some of the advantages of OR features. The structure of an object is seen in the way it is referenced in a query. For example, a fax number is referenced as t.contact.fax.num (where t is the table alias). The use of it can be seen in this query where a UK format telephone number is generated from the default international format:

```
SELECT e.name.fname || '  ' ||
        e.name.lname AS "Contact",
        '0' || a.contact.tel1.local_code || '  ' ||
        a.contact.tel1.num  AS "Telephone"
FROM Airport a, Employee e
WHERE code='AMM'
        AND e.worksfor = a.code;
```

Contact	Telephone
Fred Jones	0161 253 4400

The | | characters are used as concatenators, while column aliases are in quotes.

Nested Tables

The flight table is central to the database, with relationships to several tables, as can be seen in Figure 1. The flight instance is shown as a weak entity of flight. A flight can take place for example at the same time and day every week for a period of time. A flight instance is that flight on a particular day. The flight instance weak entity was mapped to a nested table within the Flight table; the aircraft instance entity was mapped to a nested table in the same way.

The nested table construct allows a weak entity to be represented by a nested table rather than an independent table. It effectively allows multiple instances of an attribute (also known as ‘repeating groups) to be stored in a single attribute. This can be illustrated as:

FlightNo	**Scheduled Arrival Time**	**Flight Instance**		
B747	09.20	**Reg**	**Capacity**	**Date**
		GRYTW	371	09/02/06
		GYERT	367	10/02/06
		GIYERT	369	11/02/06
		GGYTT	362	12/02/06
A333	10.15	GKOOP	213	09/02/06
		GYERT	213	10/02/06
		GIYTH	211	11/02/06

The tables were created using a series of SQL statements. OR features only work with object types, the `FlightInstance_T` type was created first, then a "table type" based on this type was created, called `FlightInstance_TAB`, which is used in the `Flight` table definition. Elements of a nested table are stored separately, so a further table, `instance_table`, is created to store its values, as seen below:

```
CREATE TYPE FlightInstance_T AS OBJECT (
        Reg             VARCHAR(10),
        Capacity        NUMBER(4),
        Fdate           DATE)
/
CREATE TYPE FlightInstance_Tab AS TABLE OF FlightInstance_T
/
CREATE TABLE Flight (
        FlightNo                    VARCHAR(4),
        ScheduledArriveTime         VARCHAR(5),
        FlightInstances             FlightInstance_Tab)
NESTED TABLE flightInstances STORE AS instance_table;
```

Queries on nested tables

With queries on nested queries, the TABLE construct is used to un-nest a nested table. This creates a temporary table for the matching results from the nested table, which is then joined to the parent table and can be projected as a normal table. For example:

```
SELECT F.FlightNo, F.ScheduledArriveTime, T.Reg, T.fDate
FROM Flight F, TABLE(flightInstances) T
WHERE F.FlightNo = 'B747';
```

It was found that is was not possible to drop a nested table, even though a parent table may be dropped. For example it was possible to drop the flight table with a drop table command but the software being used would not allow a nested table to be dropped.

```
Drop table flight; /* (the parent table) */
```

Table dropped

```
drop table instance_table;
           *
ERROR at line 1:
DROP of nested tables not supported
```

This is not a desirable situation. Any table or object may need to be dropped. In particular, some changes to a table involve backing up the data, dropping the table and recreating the table with the changes and copying the data back in.

Nested tables represent what Codd describes as a non-simple domain and which are also known as ‘’repeating groups”. It should be noted that Codd does not say that non-simple domains cannot be allowed in the relational model, but that *“the possibility of eliminating non-simple domains appears worth investigating”* (Codd, 1979).

A nested table was used to store instances of a flight. A number of problems were encountered. The set of flights embedded with the Flight Instance nested table ideally should be referenced by the Aircraft_Instance table. Using standard relational features, this could be provided by a FOREIGN KEY constraint. Alternatively, a leading DBMS provider has an object-relational datatype called a REF, which encapsulates references to row objects of a specified object type. From a modeling perspective, REFs provide the ability to capture an association between two row objects. Behind the scenes, the REF is implemented by the use of object identifiers.

It was found that it is not possible to create a REF to a nested table, which meant the intended structure of the flight instance nested table containing a REF to the aircraft instance nested table was not possible. Instead a foreign key was used. However, despite extensive research and experimentation, no way was found to add foreign key constraints to a nested table, with an error being returned stating that referential constraints are not allowed on nested table columns!

The authors assume this is because the nested table in fact is embedded in the “outer” table, such as FLIGHT and is not visible without reference to this table.

This means that although the attribute aircraft in the flight instance nested table acts as a foreign key, it is not constrained as such, a situation which is far from satisfactory as the data can become inconsistent.

Conclusion

There was little wrong with the relational version of the database; it was relatively straightforward to design and set up and the structure seemed to support the requirements of the application. The EER diagram seen in figure 1 can be implemented using standard relations, using a mapping technique for the weak entities, such as seen in Elmasri & Navathe (2000). However, the OR features do appear to offer certain advantages, such as: the OO format of columns, support of complex objects, the simplicity of REF types for joining tables and the support for composite attributes; and this project set to investigate the effectiveness of these object-relational features.

Creating queries for the OR database proved to be challenging and time-consuming. The queries use a more complex structure, particularly where un-nesting tables is involved along with a hierarchy of joined nested and non-nested tables and it takes time to come to an understanding of the syntax of such queries. This may be less of a problem for experienced database designers, but may cause problems for some application developers creating queries to embed into their application.

The usual implementation of OR features uses Object IDs (OIDs) to identify each tuple, a construct used, in particular, with nested tables. This suggests the vendor, at least in part, identifies objects as relations, with tuples as instances, thus committing Date's first great blunder. However, some vendors imply that in fact the OID relates to the user-defined type (i.e. domain). This makes sense when it is considered that (for example) nested tables can only be used in conjunction with object tables and object rows – a relevant relation must be based on a UDT.

Date also refers to the "second great blunder": mixing pointers with relations. One of his examples of such a mistake is nested tables. Although the table appears to be nested, it is actually a separate table, with pointers from the parent table. The OID for each related tuple is used for the pointers.

Date explains that, while there is no problem with tuple or relationvalued attributes, this only applies when the attribute is indeed a value. Since pointers require addresses, they can, by their very nature, only point to variables rather than values and in this case they point to tuple variables. The relational model does not allow for tuple variables, only relation variables and by introducing the concept of a tuple variable Date asserts that mixing pointers and variables therefore "seriously undermines the conceptual integrity of the relational model" (Date, 2000).

The OR database was extremely difficult to implement and took a length of time out of proportion to the complexity of the requirements. While it supported complex objects, its very complexity created problems. This was not a complex application: while the EER diagram was not easy to get right, it was nothing that any competent database designer would baulk at, yet when implementing it as an OR database, many difficulties were encountered. This certainly suggests that where very complicated systems are required, OR database systems must be considered to capture the structure of the database, but simpler systems may be better suited to the simpler relational approach.

The project found that although the classic example of a weak entity could be implemented with a nested table, this approach is not suitable when used with a chain of weak entities (Byrne & Garvey, 2004). The inability to reference the "inner table" makes the technique only suitable when the nested table does not take part in any other relationship.

It has already been stated that the term "Object Relational" is used for relational databases with object extensions. Date (2000), however, argues that the term is a misnomer, as is even the term "extensions". Some so-called extensions, such as REF, are in fact orthogonal with the relational model. He also argues that some "extensions" such as Abstract Data Types (ADT) are not extensions or object relational features, as they should be an integral part of the relational model anyway. He criticizes SQL for its failure to implement domains properly: the additional support for ADTs in SQL:1999 merely improves SQL's implementation of the relational model.

Despite following the recommendations of leading database experts, the OR features still do not work well overall, despite the success and benefits of certain features such as UDTs. Apart from the complexity of using the features and the cumbersome SQL constructs, too many problems were found. In particular in this paper we have mentioned the problems of constraints in nested tables; the difficulty in Ref attributes pointing to nested tables and the lack of relational inheritance in subtypes will be the topic of subsequent papers. It is difficult to say whether the problems are caused by the OR features compromising the integrity of the relational model or whether it is poor implementation by the vendors. The OR features used do not provide a satisfactory overall solution. Any developer should think carefully before using such features if the database is to provide a robust basis for storing and managing data.

References

Albano, A., Cardelli, L., & Orsini, R. (1985). GALILEO: A strongly-typed, interactive conceptual language. *ACM Transactions in Database Systems, 10*(2), 230-260.

Atkinson, M. P., Bancilhon, F., DeWitt, D. J., Dittrich, K. R., Maier, D., & Zdonik, S. B. (1989). *The Object-Oriented Database System Manifesto.* Paper presented at the Deductive and Object-Oriented Databases, Kyoto, Japan.

Byrne, B., & Garvey, M. (2004, 5th-7th May). *Weak Entities in Conceptual Modeling.* Paper presented at the 9th UKAIS, Glasgow, Scotland.

Carey, M. J., & DeWitt, D. J. (1988, June 1988). *A Data Model and Query Language for EXODUS.* Paper presented at the ACM SIGMOD International Conference on Management of Data, Chicago.

Carey, M. J., & DeWitt, D. J. (1996). *Of Objects and Databases: A Decade of Turmoil.* Paper presented at the Proceedings of 22nd International Conference on Very Large Databases, Mumbai (Bombay), India.

Cattell, R. G. G., Atwood, T., Duhl, J., Ferran, G., Loomis, M., & Wade, D. (1993). *The Object database standard, ODMG-93. Release 1.1*. San Francisco, Calif.: Kaufmann.

Cattell, R. G. G., Barry, D., Bartels, D., Berler, M., Eastman, J., Gamerman, S., et al. (1997). *The Object Database Standard: ODMG 2.0*. CA: Morgan Kaufmann.

Cattell, R. G. G., Barry, D., Berler, M., Eastman, J., Jordan, D., Russell, C., et al. (2000). *The Object Data Standard: ODMG 3.0*. San Francisco, Calif: Morgan Kaufmann.

Codd, E. F. (1970). A Relational Model for Large Shared Data Banks. *Communications of the ACM, 13*(6), 377-387.

Codd, E. F. (1979). Extending the Database Relational Model to capture more meaning. *ACM Transactions on Database Systems, 4*(4).

Date, C. J. (2000). *An Introduction to Database Systems* (Seventh ed.): Addison-Wesley.

Elmasri, R., & Navathe, S. (2000). *Fundamentals of Database Systems* (Third ed.): Addison-Wesley.

Garvey, M., Jackson, M., & Roberts, M. (2002). *Using Persistent Java to Construct a GIS.* Paper presented at the 4th International Conference on Enterprise Information Systems, Ciudad Real, Spain.

Millichamp, A. J. (2004). *Is there a future for object-relational databases? An evaluation of an object-relational database developed using a major commercial database management system.* University of Wolverhampton, Wolverhampton.

Rowe, L. A., & Stonebraker, M. (1987, September 1-4, 1987). *The POSTGRES Data Model.* Paper presented at the 13th International Conference on Very Large Data Bases (VLDB'87), Brighton, UK.

Stefik, M., & Bobrow, D. G. (1984). Object-Oriented Programming: Themes and variations. *AI Magazine, 6*(4).

Stonebraker, M., Brown, P., & Moore, D. (1999). *Object-relational DBMSs: tracking the next great wave*: Morgan Kaufmann.

Stonebraker, M., & Moore, D. (1996). *Object-Relational DBMSs the next great wave*: Morgan Kaufmann.

Stonebraker, M., Rowe, L. A., Lindsay, B. G., Gray, J., Carey, M. J., Brodie, M. L., et al. (1990, July 1990). *Third-Generation Database System Manifesto - The Committee for Advanced DBMS Function.* Paper presented at the Object-Oriented Databases: Analysis, Design & Construction (DS-4), Windermere, UK.