The O/R Problem:

Mapping Between

Relational and Object-Oriented

Methodologies

CIS515

Strategic Planning for Database Systems

Ryan Somma
StudenID: 9989060874
Ryeguy123@yahoo.com

Introduction

Relational databases are set of tables defining the columns of the rows they contain, which are also conceptualized as relations defining the attributes of the tuples they store. Relational Database Management Systems (RDBMS) abstract the normalized relational structures underlying the system away from the user, allowing them to focus on the specific data they wish to extract from it. Through queries, they provide users the capability of joining sets in relations into reports, transforming data into information. Through normalization, they eliminate redundancy, ensuring there is only one source for each data element in the system, and improve integrity through relationships based on data keys and validation (Rob and Coronel, 2009). Although there are alternatives to RDBMS's, the top five database management systems for 2004 to 2006 were all relational (Olofson, 2007).

Object orientation is a "set of design and development principles based on conceptually autonomous computer structures known as objects." Object-oriented programming (OOP) takes related data elements and functions, known as properties and methods, and encapsulates them into objects that may interact with other objects via messaging or act upon themselves. This strategy reduces complexity in the system, hiding it within objects that are easier to understand as concepts rather than exposing their workings within a morass of procedural code. Additionally, OOP promotes reuse of coding solutions with classes, which are built to model a generalized set of properties and methods that are implemented in object instances, which reuse the class code. Subclasses inherit properties and methods from parent classes, and then extend them with additional properties and methods, further increasing code-reusability (Rob and Coronel, 2009).

With the majority of enterprise databases structured on relational models and the majority of programming languages implementing object-orientation, a problem arises from these system architecture components, which must interact with one another, operating from two distinct paradigms. As a result, data stored relationally that object-oriented software needs to work with must be mapped from the relational model into the object-oriented one, and vice-versa. Database expert Akmal Chaudhri compares using an RDBMS with OOP to taking apart a Boeing 747 and storing all of its parts separately when not in use, and then retrieving all those parts and reassembling the aircraft when the user wants to fly it (Leavitt, 2009). This problem is known as the object-relational impedance mismatch, and software developers have wrestled with it for as long as the two strategies have existed.

Defining the O/R Impedance Mismatch

At first glance, differences between the two paradigms seem negligible, but the nuisances can have profound consequences. Relations are rigid, all rows in a table must have the same fields, but a "collection may contain objects of different classes through inheritance and polymorphism (Qint, 2009)." Classes encapsulate business logic with data, keeping everything in one place. Related relations are joined via keys comprised of data, while objects are joined through links. Objects have identities and states, which are separate and distinct, while tuples are identified by the state of their attributes. Two tuples of the same relation that have the same attribute values are considered identical, but two objects with the same property values are not identical, they are *equivalent*.

These subtle, yet important differences result in duplicate modeling in software development, one object-oriented, one relational, violating the Don't Repeat Yourself (DRY) Principle, which emphasizes having one source of authority for each bit of system knowledge

(Hunt and Thomas, 1999). When changes are made to the database model, the object model must change as well, which, in the maintenance phase, translates to changes in the database requiring modifications to the application software, violating the principle of orthogonality, that components should be designed in such a way that modifications to one component has little impact on the rest of the system. When application software is loosely coupled to the database software it interacts with, changes to one system will have minimal impact to the rest of the system.

Object-Relational Mapping Strategies

Currently software developers must either pursue third-party O/R mapping components and tools or develop their own mapping strategies in-house. Applications programmers working off RDBMS's sometimes spend more than 25 percent of their coding time mapping program objects to the database (Leavitt, 2009). There are four main techniques for O/R mapping that can require either a single table, multiple tables, or even generic database architectures. These strategies require changing the relational model to make it more convenient for the object model to build off of the database, and do not tackle the issue of multiple inheritance, that it is possible in many OOP's for a class to extend and inherit from more than one class, which further complicates the issue.

One strategy is to map the entire object-class hierarchy to a single table (Ambler, 2006). For instance, if the classes "student" and "teacher" extend the parent class "person," then there would be a single relation in the database, "person," that holds both student and teacher rows. The object-model would be able to identify which rows to load into student and which teacher objects via a "category" column that identifies each tuple appropriately. The problem with this strategy comes with the way it impacts database normalization. Students and teachers will

require very different attributes, and a single table comprising both would include multiple partial dependencies, like student_id for students and employee_id for teachers, violating second normal form (Rob and Coronel, 2009).

Another strategy is to have one relation per concrete class (Ambler, 2006). Instead of creating a single person table that includes all objects that may inherit from the abstract class, two tables are created, “student” and “teacher”, to represent the two instantiable classes. This technique is midway between the single table solution and having a one-for-one relationship between classes and tables. Implementing this strategy can lead to headaches further down the development cycle in the even a parent class is modified. For instance, if the “person” class were modified to add a “DateOfBirth” property, then every table in the database supporting a concrete class extended from “person” would need to have this attribute added to it.

The most straightforward technique is to maintain a one-for-one relationship between tables in the database and classes in the application (Ambler, 2006). This strategy requires numerous tables, not only one for each class, but also tables and key attributes to maintain the relationships between classes, as in the case of many-to-many relationships between objects. When a concrete class is instantiated, the chain of tables representing the classes it inherits from must be joined in a query to successfully populate the class properties.

The least straightforward technique, but the one that most loosely couples the objects to the relational model, is to implement a completely generic database. Here, instead of the relations being concretely defined in the database, the relational model allows for the entry of any type of structure (Ambler, 2006). In other words, instead of a “person” table with columns defining the attributes of a person object, there is a “person” table with the minimum amount of identifying data, a “person_attributes” table, where the possible attributes are listed and a

“person_attributes_values” table, where the values for attributes for person instances are stored. Columns become rows in this database model, and as objects require new properties, it is simply a matter of adding more rows to the “person_attributes” table.

Adopting such a relational model has the advantage of making the object-oriented model the authority on the business entities and relegates the relational model to supporting generic objects. The problem with such a model is its complexity in design and implementation. The “person_attributes_values” table must be generic enough to store a wide variety of data types, which means either storing everything as a VARCHAR and then casting the data to DATE, INTEGER, or other data type in a view or writing database procedures to loop through values cast to desired datatypes when they are needed in a specific sequence, since an ORDER BY will sort the date ‘01/01/2001’ as coming before ‘02/02/1901’ when they are compared as strings of characters.

Another issue with this strategy is the complexity of writing SQL queries to pull the desired data from the table structures. With each attribute stored as a row in the “person_attributes” table, multiple joins must be made to this table for each attribute we wish to pull from the “person_attributes_values” table. Querying the database to populate ten properties in an object requires ten joins to the “person_attributes” and “person_attributes_values” tables as well as the “person” table, like so:

```
SELECT
        pav1.value AS name
        ,pav2.value AS address
        ,pav3.value AS phone
FROM
        person p
        INNER JOIN person_attributes_value pav1 ON
        p.person_id = pa1.person_id
                INNER JOIN person_attributes pa1 ON
                pav1.attribute_id = pa1.attribute_id
                AND pa1.attribute_name = “name”
        INNER JOIN person_attributes_value pav2 ON
        p.person_id = pa2.person_id
                INNER JOIN person_attributes pa2 ON
                pav2.attribute_id = pa2.attribute_id
```

```
                AND pa2.attribute_name = “address”
        INNER JOIN person_attributes_value pav3 ON
        p.person_id = pa3.person_id
                INNER JOIN person_attributes pa3 ON
                pav3.attribute_id = pa3.attribute_id
                AND pa3.attribute_name = “phone”
WHERE
        p.employee_id = 1234;
```

This example only pulls three attributes, meaning a query to pull nine attributes would be three times as long. Three table joins to pull data so atomic as to only represents one attribute in one tuple of a more entity-specific table could quickly bog down a database with data retrieval, which could lead to the DBA having to construct denormalized tables updated via rules. The situation could be further exacerbated by the way the generic database model obfuscates the entities and their attributes from users accessing the database directly, further lending to the need for a denormalized table to reveal this hidden structure.

While many of these solutions reduce the load on the application’s control-layer to map relational data into objects, it does put restrictions on database development, dictating to the database architect what is allowed in their solution. This can cause problems because the DBA operates under an entirely different realm of requirements than the software developer. They cannot denormalize tables simply to make it easier to map relations to classes, nor should they add columns to a relation that serve no other purpose than to categorize tuples so that they identify with specific concrete classes. At the other end of the spectrum, with the generic database, they should not be required to take on such complexity and performance impacts in their solution simply to accommodate any possible object-model the software developers come up with.

More Conclusive Solutions

For these many reasons, Ted Neward called the O/R Impedance Mismatch the “Vietnam of Computer Science” problems, a quagmire that countless hapless developers have ventured

into, only to find the problem so complex and nuanced as to make architecting a comprehensive solution impossible, but because so much time and effort has been invested in solving the problem, developers don't want to abandon it. According to Neward, developers have the choice to either abandon OOP, abandon RDBMS, accept having to write and maintain code to map manually between object-oriented and relational models, accept the limitations of O/R Mapping software solutions, integrate relational concepts, like sets, into programming languages, or incorporate relational concepts into the framework, as in having objects encapsulate data sets that can interact smoothly with the relational database (Neward, 2006). Jeff Atwood, of the *Coding Horror* blog, makes his opinion clear on what he thinks is the correct solution:

> Personally, I think **the only workable solution to the ORM problem is to pick one or the other:** either abandon relational databases, or abandon objects. If you take the O or the R out of the equation, *you no longer have a mapping problem.*
>
> It may seem crazy to abandon the traditional Customer object—or to abandon the traditional Customer table—but picking one or the other is a totally sane alternative to the quagmire of classes, objects, code generation, SQL, and stored procedure that an ORM "solution" typically leaves us with (emphasis in original) (Atwood, 2006).

Atwood goes on to say that he also stands with the relational model as the solution of choice, arguing that "objects are overrated," but the success and adoption of OOP clearly illustrates its effectiveness as a methodology. So why not the opposite? Why should databases remain relational, and not adopt object-orientation, which would also solve the O/R problem? While the advantages object orientation has conferred upon the programming world through reduced effort, improved productivity, and greater extensibility, object-orientation principles have failed to take root in database architectures. Despite a wide variety of Object Databases (ODBMS) being available for use, some early optimism for such systems, and even adoption in the marketplace, Object-Oriented databases have languished in the world of information technology.

History of the ODBMS

The concept of Object-Oriented Databases began in the early 1980s, most notably with the ORION Research Project at Microelectronics and Computer Technology Corporation (MCC). By the late 1980s, a variety of commercial ODBMS's had become available, such as Matisse, GemStone, Vbase, and Objectivity/DB. In 1991, the Object Data Management Group (ODMG) was founded with five major OODBMS vendors. The ODMG worked throughout the 1990s to establish standards, such as the Object Query Language (OQL), but, unlike SQL, it was never adopted as a formal standard by ANSI, ISO, or other notable organization. In 2000, the market for ODBMS products peaked at approximately $100 million, and proceeded to decline afterwards. Today ODBMS's survive primarily as open-source projects, such as db4o and Orient (Grehand and Barry, 2005); although, commercial products, such as Versant Object Database, remain in production (Versant, 2005).

In 1995, a group of researchers from a variety of academic institutions published "The Object-Oriented Database System Manifesto," a paper which attempted to define what constituted an ODBMS in the same manner E.F. Codd defined a relational database in twelve rules over two decades earlier (Codd, 1970). The *Manifesto* defines a list of "mandatory" characteristics that an ODBMS must support in order to satisfy the requirements of an object-oriented system as well as a database management system. Among these mandatory characteristics is support for complex objects, like tuples, lists, and arrays, in addition to basic objects, such as integers, characters, booleans, and floats; considered data types in a relational model, these are objects in the OO model, instantiable, extendable, and inheritable (Atkinson, et.al, 1995). The ODBMS must support object identities, that is, rather than identifying tuples in a table based on primary keys or object properties, objects themselves have identities and may be

shared or copied; such as two people living together would share the same address object (Halpin and Morgan, 2008).

These mandatory requirements are, for the most part, functionality that is supported by relational databases despite the differences in conceptualizing them. Where the *ODBMS Manifesto* draws true distinctions between its system and others is in the requirements for computational completeness and persistence. At the time of their writing, SQL was not a computationally-complete language, and the *Manifesto* was attempting to bring the power of OOP into the DBMS. At the other end, was an attempt to bring persistence into OOP:

> This requirement is evident from a database point of view, but a novelty from a programming language point of view. Persistence is the ability of the programmer to have her/his data survive the execution of a process, in order to eventually reuse it in another process. Persistence should be orthogonal, i.e., each object, independent of its type, is allowed to become persistent as such (i.e., without explicit translation). It should also be implicit: the user should not have to explicitly move or copy data to make it persistent (Atkinson, et.al, 1995).

This passage illustrates the major shortcoming of object-oriented programming that a relational database fills: the need to persist the state of object properties across user sessions. A database management system can function independently, but an object-oriented software application cannot exist without a database if it needs persistence.

Authors of the *Manifesto* were unable to reach a consensus on whether certain features should be required or optional in an ODBMS, such as view definition and derived data, database administration utilities, integrity constraints, and schema evolution facility (Atkinson, et.al, 1995). These are important to note because they are standard characteristics of modern RDBMS's, but it is difficult to conceptualize how some of these would be implemented in an ODBMS. For instance, relational models have normalization standards to ensure data integrity, but objects, with their multiple instances and use of links rather than data keys to connect them, appear to violate the basic principles of normalization.

In a 2002 interview, David Maier, a professor at Portland State University and major contributor to ODBMS theory and practice, attributed a great deal of ODBMS lack of popularity to the lack of modeling support and standards:

> I think one of the reasons that [object-relational databases] are not the big hit is [that] having all of these new features in [the ORDBMS] has expanded the [schema] design space, and there's not a good design theory the way there was [with] normal form relations and so forth for relational databases… It turns out that object modeling techniques like UML aren't [very well] suited to the object-relational models, because [the] object models that the object-relational [databases] provide don't quite match up [with what UML offers] (Winslett, 2002).

Dr. Winslett also notes the long period of time it took to solidify the standard for Object Query Language (OQL), specifically figuring out how to optimize OQL queries, saying, "Yeah, if we could have done in two years what we did in ten, it might have made a big difference… it took proportionally longer in academia to figure out a lot of these [query optimization] issues… (Winslett, 2002)"

Even had the Object Data Management Group (ODMG) solidified its standards quickly, it is unknown if the standard would have been adopted, as, unlike the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) (OMG, 2009), the ODMG's Object Data Mangement Standard (ODM 3.0) is not made freely available to the public, but is only available through purchasing the book "The Object Data Standard ODMG 3.0," published just before the ODMG disbanded (ODBMS 2009).

Also while OQL was intended to be the standard technique for seeking objects in an ODBMS, few, if any modern ODBMS's have adopted this standard in its proposed format (Leavitt, 2009). For instance, the Versant ODBMS uses Versant Query Language 7.0 (VQL) (Versant, 2005) and db4o uses Query-by-Example (QBE) and other programming-style methods (db40, 2009).

Modern ODBMS's

ODBMS's are alive and well in the modern world of Information Technology; they are simply nowhere near as popular as was predicted when they were conceived, and their growth in adoption appears to have stalled. Organizations such as Exxon, CERN, Citibank, and Siemens use ODBMS's in domains such as manufacturing, scientific research, financial services, data warehousing, and process control (Objectivity, 2005), taking advantage of the semantic relationships provided. Switching to an ODBMS from an RDBMS allows users to discard the application code required for O/R mapping. According to an Objectivity white paper, one user "literally discarded one-third of his application" as a result of this (Objectivity, 2005).

A review of the features supported by the Versant ODBMS finds most RDBMS features present, such as object level locking (equivalent to table, page or row-level locking in RDBMS), schema management, distribution at the object level (equivalent to horizontal and vertical fragmentation in RDBMS), recovery, security, replication and user management. Additionally, the Versant ODBMS supports version management at the object level and checking out objects to provide local access to users over long periods of time as they make modifications. Complications arise in the Verant ODBMS when making changes to an inheritance tree, such as making a modification to the model to insert a new class in a chain, which requires backing up child classes and copying them back into the chain modified to the new hierarchy (Versant, 2005).

Not all ODBMS's are truly object-oriented. For instance, the Informix Illustra and UniSQL ODBMS's are actually an OO application layered over a relational database, leading some to refer to them as "Object-Relational" DBMS's (ORDBMS) (Objectivity, 2005). In such

cases, the OO application is merely acting as a O/R Mapping tool, failing to take true advantage of object-oriented modeling.

A major criticism of modern ODBMS's is their lack of support for taking disparate data and recombining it in such a way to produce new relationships in a persistent manner, such as when a user creates a view in a relational database:

> …commercial object databases lack support for high-level database application programming where facilities are required not just for the persistent storage and retrieval of data, but for the management of complex interrelated collections of data over long periods of time. Concepts well known from database systems such as constraints and triggers are usually absent and support for object and schema evolution minimal at best (Norrie et al, 2008).

Even were this missing functionality provided, it might also be the sheer complexity of object-oriented models that keeps this database model from mainstream adoption. A comparison of database models that included hierarchical, network, relational, entity-relationship, and object-oriented models found that, while OO models have the advantage of adding semantic content and promoting data integrity through inheritance, they require a complex navigation system, a steep learning curve, and high system overhead that slows transactions (Rob and Coronel, 2009). Considering this inherent complexity in the context of an overwhelming familiarity with relational databases in the marketplace, and we may assume that most organizations find that the advantages conferred from ODBMS's are insufficient to justify scrapping existing RDBMS's (Leavitt, 2009).

Conclusions

Simplicity and familiarity are features, and there are no dramatic shortcoming with relational databases that provide the necessary impetus for migrating to a new model, especially considering the complexity and adjustment required to move into object-orientation. For these exact same reasons, it does not make sense for programmers to migrate away from object-orientation, as OOP has revolutionized software development and relational-models do not

provide the same benefits in a programming context. With relational models being best suited for realms of data and object-orientation the best solution for software development, developers will continue to grapple with the issue of mapping data between the two paradigms. Adhering to the principle of orthogonality means programmers should model their software according to the best principles of object-orientation and DBA's should architect their relations according to the principles of normalization and data integrity. In the meantime, developers will continue to work on better strategies for O/R Mapping, finding the best practices at least, if not an ultimate solution.

References

Ambler, Scott (2006). *Mapping Objects to Relational Databases: O/R Mapping in Detail.* AgileData.org. Retrieved from agiledata.org on September 12, 2009 at: http://www.agiledata.org/essays/mappingObjects.html

Atkinson, Malcom; Bancilhon, Francois; DeWitt, David; Dittrich, Klaus; Maier, David; and Zdonik, Stanley (1995). *The Object-Oriented Database System Manifesto.* Carnegie Mellon School of Computer Science. Retrieved from cmu.edu August 15, 2009 at: http://www.cs.cmu.edu/afs/cs.cmu.edu/user/clamen/OODBMS/Manifesto /htManifesto/Manifesto.html

Atwood, Jeff (2006). *Object-Relational Mapping is the Vietnam of Computer Science.* Coding Horror, Programming and Human Factors. Retrieved from CodingHorror.com August 22, 2009 at: http://www.codinghorror.com/blog/archives/000621.html

db4o (2009). *db40 Tutorial.* db40.com. Retrieved from db40.com September 13 2009 at: http://www.db4o.com/about/productinformation/resources/db4o-6.3-tutorial-java.pdf

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

Grehan, Rick and Barry, Doug (2005). *Introduction to ODBMS: Short History.* Object Database Management Systems Portal. Retrieved August 15, 2009 at: http://www.odbms.org/Introduction/history.aspx

Halpin, Terry and Morgan, Tony (2008). *Information Modeling and Relational Databases.* Morgan Kaufmann, Burlington, MA.

Hunt, Andrew and Thomas, David (1999). *The Pragmatic Programmer.* Addison-Wesley, Reading, MA.

Leavitt (2009). *Industry Trends: Whatever Happened to Object-Oriented Databases?* Leavitt Communications Inc. Fallbrook, CA. Retrieved from leavcom.com August 22, 2009 at: http://www.leavcom.com/db_08_00.htm

Neward, Ted (2006). *The Vietnam of Computer Science.* Blogride, Ted Neward's Technical Blog. Retrieved from tedneward.com August 22, 2009 at: http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx

Norrie, Moira; Grossniklaus, Michael; Decurins, Corsin; Spindler, Alexandre; Vancea, Andrei, and Leone, Stefania (2008). *Semantic Data Management for db40.* Institute for Information Systems, ETH Zurich, Switzerland. Retrieved from odbms.org on September 13, 2009 at: http://www.odbms.org/download/042.02%20Norrie%20 Semantic%20Data%20Management%20for%20db4o%20March%202008.PDF

Objectivity (2005). *Hitting the Relational Wall: An Objectivity White Paper.* Objectivity, Inc. Retrieved from objectivity.com September 13, 2009 at: http://www.objectivity.com/pages/downloads/whitepaper/pdf/RelationalWall.pdf

ODBMS Portal (2009). *Object Data Management Group.* ODBMS.org. Retrieved from odbms.org August 22, 2009 at: http://www.odbms.org/odmg/

Olofson, Carl (2007). *Worldwide RDBMS 2006 Vendor Shares: Preliminary results for the Top 5 Vendors.* IDC, Framingham, MA. Retrieved from Microsoft.com September 12, 2009 at: http://download.microsoft.com/download/A/B/9/AB93175B-BA6A-4332-AFBF-FE4C3749BBEC/IDC%202006%20DB%20Marketshare%20206061.pdf

OMG Portal (2009). *Object Management Group: Common Object Request Broker.* OMG.org. Retrieved from omg.org August 22, 2009 at: http://www.omg.org/spec/CORBA/

Qint (2009). *Objects vs Relational Database Concepts*. Qint Software, Germering, Germany. Retrieved from reportsanywhere.info August 22, 2009 at: http://reportsanywhere.info/joria/doc/objectsvsrecords.html

Rob, Peter and Coronel, Carlos (2009). *Database Systems: Design, Implementation, and Management, Eighth Edition.* Course Technology, Boston MA.

Versant (2005). *VERSANT Database Fundamentals Manual.* Retrieved from versant.com August 16, 2009 at: http://www.versant.com/developer/resources/objectdatabase/documentation/database_fund_man.pdf

Winslett, Marianne (2002). *David Maier Speaks Out.* ACM SIGMOD series of interviews. Retrieved from sigmod.org August 29, 2009 at: http://www.sigmod.org/record/issues/0212/maier-oct-16.pdf