VOLUME 5, 2012

ISSN 2045-3345

RESEARCH JOURNAL OF ECONOMICS, BUSINESS AND ICT

ADVANTAGES OF OBJECT-ORIENTED OVER RELATIONAL DATABASES ON REAL-LIFE APPLICATIONS

Hakan Bütüner[I]
Industrial Management and Engineering Co., Istanbul, Turkey

ABSTRACT

Business, manufacturing and other kinds of databases have been implemented by the relational database technology. The transition from one generation to the next has always been necessitated by the ever-increasing complexity of database applications and the cost of implementing, maintaining, and extending these applications. Accordingly, whatever the application areas of the relational database technology are, the basic shortcomings and weaknesses of this discipline are valid.

On the other hand, the object-oriented database technology forms a good basis for all kinds of database applications, e.g., CAD / CAM / CASE / CIM systems, knowledge management systems, and others, due to its flexible, effective, and extensible data modeling and application formalism.

In this study, two popular database technologies, the relational and object-oriented, are briefly introduced and their general characteristics are discussed. The purpose of this study is to point-out and compare the important features of relational and object-oriented databases, for the sake of understanding the pros and cons of these technologies before implementing them.

UDC CLASSIFICATION & KEYWORDS

■ 004.65 ■ OBJECT-ORIENTED ■ APPLICATIONS ■ DATABASES ■ RELATIONAL

INTRODUCTION

Business, manufacturing and other kinds of databases have been implemented by the relational database technology. The transition from one generation to the next has always been necessitated by the ever-increasing complexity of database applications and the cost of implementing, maintaining, and extending these applications. Accordingly, whatever the application areas of the relational database technology are, the basic shortcomings and weaknesses of this discipline are valid.

However, object-oriented technology, which is the fundamental advancement in database technology, satisfies the objective of reducing the difficulty of designing and implementing very large and complex database requirements.

In this study, two popular database technologies, the relational and object-oriented, are briefly introduced and their general characteristics are discussed. The purpose of this study is to point-out and compare the important features of relational and object-oriented databases, for the sake of understanding the pros and cons of these technologies before implementing them.

Object-Oriented vs. Relational Databases

The relational database model requires the data or real-world entity to be organized into dimensional arrays called tables (see Figure 1.) According to the article Relational Databases (1992), Baker, in summarized form, stated the following statements. A typical database will contain many tables. A table consists of columns that are commonly referred to as attributes and rows that are called records. The domain of an attribute contains only a limited primitive data type (such as characters, integers, fixed-length strings, fixed- and floating-point numbers, Boolean, and date), and an attribute may only have a single fixed length value. Every table will have one or more attributes designated as a key. This is necessary so that each record can be uniquely identified.

One of the characteristics of the relational technology is that relationships between tables are based on the same contents (values) of the key attribute(s) of the relations and operations, such as joins and sorts. Also, relationships between tables are not explicitly defined. External software applications and/or database languages such as SQL must perform this function. Before the implementation of operations, such as joining two or more tables, the entities are grouped into tables, and the attributes in each of the tables must undergo a process called normalization.

According to the article Introduction to Object-Oriented Databases (1990), Kim, we can summarize his statements within the following paragraphs. Object-oriented database models require the data or real-world entity to be organized as objects. A record in the relational model corresponds roughly to an object in object-oriented models (see Figure 2.) However, while a record only has a state, an object encapsulates both state (data) and methods, and thus can represent structure as well as behavior. Therefore, each object can encapsulate many properties (attributes) and methods (that operate on its related properties) associated with it, which improves semantic consistency by means of encapsulation. Methods are used to operate and change the object and its properties and/or to return part of its state. Examples include creations of new instances, deletions, retrieving instances, updating instances and/or their properties. These methods and attributes in an object are invoked from the outside of the object by means of message passing. Thus, the interface of an object, which is a collection of messages to which the object responds by returning an object, is public. Users are free to ignore the implementation details of methods of the object, which is private.

For demonstration purposes, let us consider the example of a company that manufactures ladders according to customer specifications. The ladder can have any number of rungs, as specified by the customer. Using the object-oriented approach, the Ladder would be modeled as an object with attributes such as the Number_of_Rungs, and methods such as the Create_Graphical_Representation, and Calculate_Tolerances. Upon invoking these methods by supplying the relevant parameters, the object would automatically provide a graphical representation of itself, including design changes that are necessary to

[I] hakan.butuner@imeco-tr.com

accommodate the change in variables (i.e., the Number_of_Rungs).

However, the same organization using the relational approach is semantically incapable and difficult to understand. For example, assume we need to define height, length, and width of the packages of parts. In the relational schema, these attributes may be defined independently or they may be defined as the attributes of another relation. In the first case the fact that they all belong together is lost, and in the second case a new user-defined key attribute that consists of four elements has to be defined.

Unlike the relational model, where relationships are not explicitly defined between tables, object-oriented models have several ways of specifying relationships between objects directly in the database structure, and these relationships are not implied by the object's contents (values).

First, objects can be grouped together in a table-like structure, called a class, to resemble the relational model's table. However, while a table group records that share the same set of attributes, a class groups objects (instances of the class) that share the same set of attributes and methods. Additionally, unlike tables, class(s) (its/their objects) can be the subset(s) of other class(s) (its/their objects) which are called class hierarchy (IS-A relationship). For example, the class Machinable_Part would be a subset of the class Part. The properties of the Machinable_Part class would automatically include those of the Part class. This feature is known as inheritance (allowing new objects to be built from old ones without having to redefine them from scratch) and is another special characteristic of object-oriented models. Inheritance allows each succeeding level of class(s) (its/their objects) to acquire the properties (attributes) of the higher-level class(s) (its/their objects) and to have additional local properties.

Figure 1: An illustration of the relational database model

CUSTOMER TABLE		
Customer #	Customer Name	Location
A	MFG Co.	New York
B	CIM Co.	Chicago

EXISTING ACCEPTED ORDER TABLE		
Order #	Order Date	Customer #
X5	121093	A
X6	122393	A
X12	20394	A

INVENTORY PRODUCT TABLE		
Product #	Product Name	Price
1	Table	300
2	Chair	200
3	VCR	275
4	Walkman	100
5	Radio	350
7	T	0

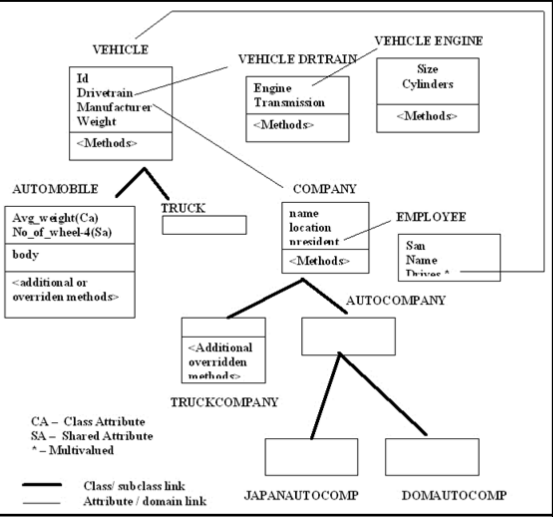
LINE ITEM TABLE			
Line Item	Order #	Product #	Qty.
P1	X5	1	5
P2	X5	2	4
P3	X5	1	2
P4	X5	7	2
P1	X6	1	6
P1	X12	5	3

Source: Bütüner (1994)

In addition, class(s) (its/their objects) can inherit operations (methods) from the higher-level class(s) (its/their objects) as well. In some cases, an inherited operation may need more programming code to take into account the additional properties of the lower-level class. This can be accomplished by refining the higher-level's operation. Only the additional programming code is included in the refined operation. The lower-level object automatically retrieves the higher-level's operation whenever the lower-level's operation is invoked.

Generalization allow us to impose structure onto the set of object types in the database, and the structural and behavioral relationships between objects can be seen more clearly and organized in a more modular manner. This allows easy modifications on the schema and allows the objects to be reused for other applications. Therefore, using the class hierarchy captures the semantics of categorization

Figure 2: An illustration of an object-oriented database model



Source: Kim (1990)

of components, and at the same time suspends multiple occurrences of the same attributes and methods by stating them once, which reduces the size of the schema considerably.

Second, unlike the relational model, the domain (value) of any attribute of a class (its objects) may be another class (its objects) that naturally leads to a directed graph of classes to resemble the hierarchical model's database. However, this graph, called an "aggregation hierarchy" (IS-PART-OF relationship), can be cyclic. Further, an attribute of an object may take a single or a set of values, which may let an object contains variable-length values. In this way, besides the basic data types provided by the relational model, designers can define a variety of data types (including array, set, point, text, graphical, image, etc.) to model complex data structures that closely match real-world objects and enhance the semantic contents of the schema. Furthermore, since an object can refer to another object, a method defined in one object can perform an operation on that other object. Consequently, object-oriented models provide a significant enhancement to the database designers to allow them to define additional data types and operations on these data types, called "abstract data types."

By the help of the article Bottom-up Design of Object-Oriented Databases (2001), Kappel, Rausch-Schott, and Retschitzegger, the following example has been formed. A concept closely related to the abstract data types is the "complex data object." For example, consider an order processing application. An object represents a line item in a customer order. An attribute within the line item indicates the product ordered. Within the relational model, this attribute would contain a value of the Product Number relation key identifier, thus providing a cross reference from the Line Item to the Inventory Product relation (see Figure 1). Within object-oriented models, the same attribute would contain a reference to the inventory product itself directly. Thus, we have an object that contains another object. In this example, restricting the line items' attribute to contain only the inventory products means that this attribute must be of type Inventory_Product. As mentioned above, we can also define new and/or use existing operators for the abstract data type Inventory_Product. These operators, for example, might change the values of the corresponding attribute.

Third, classes (their objects) can be connected in both of the previously outlined ways. They are the basic specifications offered by object-oriented models to represent the direct relationships between objects.

Based on the article OODBMS vs. Relational (1992), Loomis, we can summarize his statements within the following paragraphs. However, in the relational technology, there are two fundamental problems regarding the relationships. When the relational model is used, application entities have to be transformed and treated as tables. Additionally, many normalized tables could be required to represent a single real-world entity. Users and programmers typically want to manipulate objects, not tables, as tables have poor semantic expressions to represent real-world entities. At run time, entities from the tables need to be put back together to make application entities. As there is only a very weak notion of linkages between tables, putting those entities together in a meaningful way needs joining and sorting, in which the resulting view runs inefficiently, or in other words, slowly. Because the object-oriented technology can model and manipulate application entities directly, there is no need to transform application entities into tables; consequently, a joining process is not necessary, and therefore run time performance can be far better than with RDBMSs support.

Second, relationships are always associated with "integrity constraints" and other "rules." For example, don't delete customer entities that contain outstanding orders. This problem has meant building all the rules into an application code. More recently, some RDBMSs let the data dictionary store rules as procedures written in SQL and executed whenever changes were made to the database. However, this approach still falls short because without binding integrity constraints directly to database relationships in the first place, these rules cannot become a built-in part of the database, and their operations are restricted to those defined for the basic data types.

In the relational approach, entity integrity and referential integrity are enforced by SQL. It also allows certain types of data integrity checks, such as the attribute type and domain checks and some enforcement of "relation constraints" (e.g., attribute A must be equal or greater than attribute B), to be defined within the schema.

Within the relational model, the notion of primary keys means that record identification (record uniqueness) is based on the value of fields in the row, and changes typically would not be allowed to the record's primary key or identifier. Relationships within and among tables are implemented using the key values. If the value of a primary key field changes, then the row becomes a different row (the entity identity independent from entity status is not supported), and a search for and change of all relations-relevant records, containing the identifier, has to be performed by the specific updating program in order to maintain the integrity among the data elements. Also, unlike the object-oriented approach, if you delete a record stored in a relational database, other records may be left with references to the deleted one and may now be incorrect. The integrity of the data thus becomes suspect and creates inconsistent versions.

In the object-oriented technology, object identifiers are logical pointers and never reused. An object identifier uniquely identifies an object and is generated by the system. It verifies the existence of the object and the class to which the object belongs when the system attempts to send it messages. This lets the system attach methods to objects that can enforce arbitrary integrity rules. More specifically, in OODB technology only the object identifier(s) of the instance(s) of a class are stored within the instance variables (attributes), not the attribute values themselves, except primitive attributes (in which only values are stored). Therefore, it is permissible to store any type of data in an object's instance variable by storing only its object identifier. Relationships among objects are implemented using the internal identifiers. Thus, unlike RDBMSs, if any attribute value associated with the referenced object is changed, this change is automatically propagated to all other objects that refer to it. Consequently, even an attribute value that would be considered the object's identifier can be modified without any problems, and the object still maintains its identity.

In general, this concept ensures the maintenance of integrity among the data elements and makes objects easy to change, as relationships are explicitly defined through the hidden identifiers. Also, due to the encapsulation, object-oriented database models guarantee consistency and relationship constraints between the operations executed to the objects of any given data type by different applications. In object-oriented models, other explicit constraints (such as data scopes, data units, data types checking, etc.) can also be enforced.

Also, in the object-oriented approach, replacing multiple occurrences of compound keys reduces the size of the database as well, but this reduction may be offset by increased space utilization due to fixed size and smaller, and uniform system-defined object identifiers. In the relational schema, all the elements of the compound key appear for each of these references without any syntactic indication that they, together, reference a part.

Finally, though the object-oriented approach can form references based on identity, this is not the sole basis for relationships in the model, as it can relate objects through the values of properties as well. In this way, it provides a framework for unifying value-based and identity-based access.

By the help of the article "Object Database Technology: What Is It, What Are Its Advantages and Who Is Using It?" (1992), Blakey, we can summarize his statements within the following paragraphs. As a result, we can say that the object-oriented approach provides clear mapping of conceptual images. Once we identify the necessary classes, we work on their attributes (defined with both instance and class variables) and behaviors (defined with both instance and class methods). It is a very organized process. This is possible because we have a good initial understanding of the semantics of the schema in terms of the objects; i.e., it is better structured, and a simple mapping to the logical view of data makes it more intuitive. Additionally, as data is organized into tables in the relational approach, this makes these systems inflexible and resistant to change. For this reason, they are ineffective for storing data, especially data that is not table oriented. On the other hand, object-oriented database models do not have a rigid structure, as the schema can be set, updated, and extended more flexibly. This gives them so much power, such as higher productivity, better quality, and more flexibility. For example implementation on one side of a wall does not constrain the design on the other.

The explicit enforcement of object-oriented technology, such as class hierarchy and inheritance (generalization), aggregation (inter-object) relationships, methods, and encapsulation and message passing have stimulated a reinvestigation into the architectural concepts that are also provided by RDBMSs. These functionalities are persistence data storage and transaction management and manage sharing through concurrency control (which is more comprehensive by object-oriented technology, including database activities which require interactive and cooperative transactions that may last for long periods, also known as long transactions). They also protect data through backup and recovery; grant only authorized access to data through security schemes; and tune performance through a variety of storage and access-path options. Additionally, they provide administration facilities, schema management, and a query facility. (These concepts are also greater than RDBMSs.)

The relational technology allows attributes to be added into existing tables, but dropping attributes or moving them to other tables is seldom permitted. However, in object-oriented technology, there are two types of schema evolution: changes to the class definition and changes to the structure of a class hierarchy. The types of changes include creation and deletion of a class, alteration of the IS-A relationship between classes, addition and deletion of instance variables and methods, etc.

In RDBMSs, SQL is used to define, manipulate and control data stored in a database, and there are relational algebra (that enables the user to extract selected records and attributes within a single relation) and relational calculus (that facilitates the manipulations of two or more relations) and set-oriented operations serving as standard guidelines for designing relational databases. Users can generate multiple views by means of SQL operation joins from several tables.

Besides some of its "drawbacks" described below, data manipulation in RDBMSs is quite well defined and standardized:

(a) SQL only provides a small number of general operations for queries and updates on the relational database records. The operations are the same regardless of the semantics of the entity involved.

(b) SQL does not permit updates to views that represent joins.

(c) SQL permits adding attributes, but dropping attributes or moving them to other relations is seldom permitted.

(d) SQL is mismatched with the common programming languages that deal with record-at-a-time logic, not table-at-a-time logic.

According to the article "Object-Oriented Application Frameworks" (1997), Fayad and Schmidt has stated and concluded the following paragraphs:

In object-oriented technology, unlike RDBMSs, all objects provide their own set of operations, with some sharing through classes, inheritance, and aggregation mechanisms. Accordingly, object-oriented models are more powerful in their capability of explicitly expressing application semantics. Thus, within an OODB, operations or methods can be defined to perform updates or queries on objects using a message-passing concept. However, unlike object-oriented technology, within the relational technology, the message passing concept can be handled by writing externally defined programs, and/or by a programmer manipulating the data only through predefined operations of SQL (which requires the user to know specific table and attribute names).

Since operations are contained within the objects (encapsulation), i.e., in the database itself, much less of a programming code is required for an external application. This also provides an appropriate basis for the development of portions of general-purpose reusable code (modular decomposition) of an object that can be reused in different applications and can reduce the application development effort. Hence, object orientation enhances logical data independence, i.e., independence of applications from database implementation. Object-oriented databases are thus active while relational databases are passive. In relational technology the application programs must redefine relationships and operations for every application. In short, use of object-oriented concepts ensures better extensibility, reusability, and maintenance of software.

In general, in object-oriented technology, data definition, manipulation, and control languages are based on object-oriented concepts (previously outlined). The OODBMSs interfaces can be well integrated with the object-oriented programming languages and there is no impedance mismatch, as both use the same object-oriented concepts. In fact, the whole concept of a separate database and programming languages can disappear. However, today there are no agreements on standard language and standard guidelines for designing object-oriented databases.

Object-oriented technology has some other features which are not present in the relational technology, such as the concepts of temporal evaluation of data (i.e., versioning and change notification of data for designs when they evolve); derived attributes that are computed from other attributes; polymorphism; dynamic binding; etc.

Conclusion

It is difficult to represent and implement a variety of applications using relational database technology because of the shortcomings mentioned above. These applications include computer-aided design, engineering, software engineering and manufacturing systems (CAD, CAE, CASE, and CAM); knowledge-based systems (expert systems and expert system shells); multimedia systems that manage images, graphics, voice, and textual documents; statistical and scientific modeling and analysis programs; information systems; and so on. However, object-oriented technology, which is the fundamental advancement in database technology (because of the features outlined previously), satisfies the objective of reducing the difficulty of designing and implementing very large and complex database requirements.

References

Kappel, G., Rausch-Schott, S., Retschitzegger, W. Bottom-Up Design of Active Object-Oriented Databases. Communications of the ACM, 2001.

Fayad, Mohamed E., Schmidt, Douglas C. Object-Oriented Application Frameworks. Communications of the ACM, 1997.

Loomis, Mary E. S. ODBMS versus Relational. JOOP Focus on ODBMS, 1992.

Blakey, Adrian. Object Database Technology: What Is It, What Are Its Advantages and Who Is Using It? CALS Journal, 1992.

Baker, Henry G. Relational Databases. Communications of the ACM, 1992.

Kim, Won. Introduction to Object-Oriented Databases. 1st ed. Cambridge, Massachusetts: The MIT Press, 1990.