JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2003**

Vol. 2, No. 4, July-August 2003

Achievements and Weaknesses of Object-Oriented Databases

Sikha Bagui, Department of Computer Science, University of West Florida, U.S.A.

Abstract

Object-oriented database systems began developing in the mid-80's out of a necessity to meet the requirements of applications beyond the data processing applications which were [are] served by relational database systems. This paper serves as an overview on the achievements of object-oriented database technology so far, and also discusses the weaknesses that have to be yet resolved by the object-oriented database community before object-oriented database technology can become as widespread as relational databases.

1 INTRODUCTION

Object-oriented database systems, which can be considered fifth-generation database technology, began developing in the mid-80's out of a necessity to meet the requirements of applications beyond the data processing applications, which characterized relational database systems (fourth-generation database technology). Attempts to use relational database technology for advanced applications like computer aided design (CAD), computer aided manufacturing (CAM), software engineering, knowledge-based systems, and multimedia systems, quickly exposed the shortcomings of relational database systems [22], [8]. The need to perform complex manipulations on existing databases and a new generation of database applications generated a need that would be better satisfied by object-oriented databases (OODBs).

Many definitions of object orientation and object-oriented databases have been developed over the years ([3], [21], [33], [9], [10], [20], [25]), but we will define object-oriented databases as databases that integrate object orientation with database capabilities. Object orientation allows a more direct representation and modeling of real-world problems, and database functionality is needed to ensure persistence and concurrent sharing of information in applications.

Today there are over 25 object-oriented database products on the market, including, GemStone from Servio Corporation, ONTOS from ONTOS, ObjectStore from Object

Cite this article as follows: Sikha Bagui: "Achievements and Weaknesses of Object-Oriented Databases", in *Journal of Object Technology*, vol. 2, no. 4, July-August 2003, pp. 29-41.
http://www.jot.fm/issues/issue_2003_07/column2

Design, Inc., and many others [26]. In addition, relational database management systems from Oracle, Microsoft, Borland, Informix, and others have incorporated object-oriented features into their relational systems. A lot of these products have been around since the mid to late '80's, and after almost one and half decades of development, the lack of maturity of a lot of these products has contributed to the slow acceptance of OODBs into the real worldwide market of today. Most current OODBs are still not full-fledged database systems comparable to current relational database systems (RDBs) [26]. In this paper we will discuss some of the achievements and weaknesses of the present OODBs.

In section two of this paper we briefly present the OODB model. In section three we discuss the achievements of OODBs, and in section four we present the weaknesses of current OODBs.

2 THE OBJECT-ORIENTED DATABASE (OODB) MODEL

Object-oriented database systems evolved from a need to satisfy the demand for a more appropriate representation and modeling of real world entities, so OODBs provide a much richer data model than conventional (relational) databases. The OODB paradigm is based on a number of basic concepts, namely object, identity, class, inheritance, overriding, and late binding [2], [4], [24], [32], [37].

In the object-oriented data model (OODM), any real world entity is represented by only one modeling concept – the object. An object has a state and a behavior associated with it. The state of an object is defined by the value of its properties (attributes). Properties can have primitive values (like strings and integers) and nonprimitive objects. A nonprimitive object would in turn consist of a set of properties. Therefore objects can be recursively defined in terms of other objects. The behavior of an object is specified by methods that operate on the state of the object.

Each object is uniquely identified by a system-defined identifier (OID). Objects with the same properties and behavior are grouped into classes. An object can be an instance of only one class [6], [7] or an instance of several classes [12], [14].

Classes are organized in class hierarchies. A subclass inherits properties and methods from a superclass, and in addition, a subclass may have specific properties and methods. In some systems, such as ORION[5], a class may have more than one superclass (multiple inheritance), while in others it is restricted to only one superclass (single inheritance).

Most models allow for overriding inherited properties and methods. Overriding is the substitution of the property domain with a new domain or the substitution of a method implementation with a different one [8].

3 ACHIEVEMENTS OF THE OBJECT-ORIENTED DATABASE MODEL

OODBs allow representation of complex objects in a more straightforward way than relational systems. In this section we will discuss some of the achievements of OODBs so far: OODBs allow users to define abstractions, facilitate the development of some relationships, eliminate the need for user defined keys, have developed a new set of equality predicates, eliminate the need for joins in some cases, have performance gains over the RDB model in some situations, and have support for versioning and long-duration transactions. Finally, object algebra has been developed, although it may not be as developed as relational algebra yet.

OODBs allow users to define abstractions

OODBs have the ability to define new abstractions and to control the implementation of these abstractions. The new abstractions can match the data structures needed for intricate tasks – new abstract data types. That is, OODB packages today allow the user to create a new class with attributes and methods, have the classes inherit attributes and methods from superclasses, create instances of the class each with a unique object identifier, retrieve the instances either individually or collectively, and load and run methods [26]. OODMs also allow the definition of objects as aggregates of other objects, and aggregates can be nested at several levels. Properties too can have complex structures, and can be defined using the collection constructor. Furthermore, they can have nonprimitive objects as values, allowing deeply nested object structures [8].

Multivalued properties are used in the OODMs to express complex data structures. In the relational model this is obtained by using additional relations [8] and joins.

An example of an OODB package that includes all of the above features would be ENCORE [38]. The data model in ENCORE is based primarily on data abstraction. ENCORE allows subtyping (inheritance), encapsulation, complex structures, object identity, and late binding of methods. ENCORE also has the ability to relate objects by means of properties. In ENCORE, a property `p` relates an object `x` to a set of objects `S` without making any statement about how this relationship is computed. This could be computed by a direct reference to the identity of `S` (or its members), or it could be computed by matching of values for some other properties as a join.

OODBs facilitate development of some relationships

OODBs offer the feature of inverse relationships to express a mutual reference between two objects (a binary relationship). This system ensures referential integrity by establishing corresponding reference as soon as a reference is created. It is even possible to automatically propagate deletion via these references [15]. An example of an OODB

package that supports the automatic maintenance of inverse relationships is ObjectStore [35].

OODBs eliminate need for user defined keys

The OODB model has an OID that it is automatically generated by the system and that guarantees uniqueness to each object. This, in addition to eliminating the need for user defined keys in the OODB model, has brought other advantages to OODBs: 1) the OID cannot be modified by the application; 2) as discussed in [23] and [37], the notion of object identity provides a separate and consistent notion of identity, which is independent of how an object is accessed or modeled with descriptive data. Therefore, two objects are different if they have different OIDs, even if they have the same structures and the same values for all their properties. In the RDB model, where object identification is supported by user-defined keys, those objects would be considered the same object [8].

Development of equality predicates

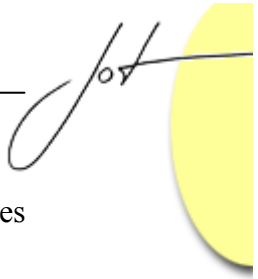
In RDBs, equality is always based only on values. In RDBs, two tuples are the same entity if all key attributes have the same values. In OODBs, however, different types of equality have been developed and defined [37], [8]:

1. *Identity equality of objects*: Two objects, S_1 and S_2 are equal if they have the same object (that is, if they have the same OID).
2. *Value equality of objects*: This can be determined in two ways: (a) Two primitive objects are equal if they have the same value. (b) Two nonprimitive objects are equal if they have the same number of properties, and if, for any property p_i of S_1 there exists a property p_j of S_2 that is equal in value.
3. *Value equality of properties*.
4. *Identity equality of properties*.

OODBs reduce need for Joins

The capability of navigating through object structures and the resulting path expressions in object attributes gives us a new perspective on the issue of joins in OODBs. The relational join is a mechanism that correlates two relations on the basis of values of a corresponding pair or attributes in the relations. Since two classes in an OODB may have corresponding pairs of attributes, the relational join (or, explicit join) may still be necessary in OODBs. For example, suppose we have a class `Student` and a class `School`, and both have attributes `Name` and `Age`. Although the `Name` and `Age` attributes of the class `School` may not have the domains of `Name` and `Age` attributes of class `Student` and vice versa, we may wish to relate the two classes on the basis of values of these attributes (e.g. *find all the student objects whose age is less than the age of the school the student goes to*).

But, as mentioned above, path expressions can reduce the need for joins of classes significantly, as compared to RDBs [26]. There are also times when the need for the relational join can be eliminated [26]. For example, when the domain of an attribute of a



class A is class B, the fetching of the OIDs of objects in a class that are stored as values of an attribute in another class eliminate the need for an implicit join between objects.

Therefore, in OODBs there is a distinction between the implicit join, derived from the hierarchical nesting of objects, and the explicit join, which is similar to the relational join where two objects are explicitly compared by using either the value or the identity equality. Furthermore, all explicit joins (using the *value* equality or the *identity* equality) cannot be defined in relational query language because any predicate in RDBs can only involve atomic attributes [8].

Performance gain using OODBs

Although most current OODBs are not full-fledged database systems comparable to current RDBs, OODBs have a few sources of performance gain over RDBs:

1. In an OODB, the value of an attribute of an object X, whose domain is another object Y, is the object identifier (OID) of the object Y. Therefore, if an application has already retrieved object X and now would like to retrieve object Y, the database system may retrieve object Y by looking up its OID. If the OID is a physical address of an object, the object may be retrieved directly; if the OID is a logical address, the object may be fetched by looking up a hash table entry (assuming that the system maintains a hash table that maps an OID to is physical address) [26]. This would not be possible so easily in RDBs, since RDBs do not maintain OIDs.
2. A second source of performance gain in OODBs over RDBs is that most OODBs convert the OIDs stored in an object to memory pointers when the object is loaded into memory. Since RDBs do not store OIDs, they cannot store memory pointers to other tuples. The facility to navigate through memory-resident objects is a fundamentally absent feature in RDBs, and the performance drawback that results from it cannot be neutralized by simply having a large buffer space in memory. Therefore, for applications that require repeated navigation through linked objects loaded in memory, OODBs can dramatically outperform RDBs [26].
3. Also, even if OODBs are not indexed, it may be convenient to execute arbitrary queries that suit the object structure by sequential scan – that is, exploit the reference paths between objects. When queries are formulated in the direction not supported by references, the query will be processed by sequential scan [15]. However, queries that are formulated on object relationships not directly modeled by references are executed inefficiently.

Support for versioning or long-duration transactions

Versioning and long-duration transactions are missing in RDBs. Few OODBs offer versioning and long-duration transactions, though with limited facilities only [26].

Development of Object Algebra

Though not as developed and mature as relational algebra, object algebra has been developed that defines five fundamental object-preserving [30] operators: *union*, *difference*, *select*, *generate* and *map*. Other operators like *intersection* may be defined from these fundamental operators. Equivalence preserving transformation rules for logical optimization of object algebra expressions are derived in [34] and [35]. While the mapping process for the *union*, *difference* and *map* operators are primarily one-to-one, the mapping for the *select* and *generate* operations is one-to-many [35]. Object preservation means that algebra operators return objects that belong to predefined classes in the database, and do not create new objects. *Union* returns objects that are in both sets P or Q or both. *Difference* returns the set of objects that are in set P and not in set Q. *Select* returns a subset of an input set. *Generate* generates objects from those in the input sets. *Map* returns a set of objects resulting from each sequence application [35].

4 WEAKNESSES OF THE OBJECT-ORIENTED DATABASE MODEL

The expectation was that object-oriented technology would bring a quantum jump to database technology. But, in spite of the achievements of OODBs discussed above, OODBs have not been able to make a major impact because of weaknesses still present in OODB model and technology.

In OODBs there is a lack of basic features that users of database systems have become accustomed to, and therefore expect. The features include, lack of interoperability between RDBs and OODBs, minimal query optimization, lack of standard query algebra, lack of query facilities, no support for views, security concerns, no support for dynamic class definition changes, limited support for consistency constraints, limited performance tuning capabilities, little support for complex objects, limited integration with existing object-oriented programming systems, limited performance gains, among others.

Interoperability between RDBs and OODBs

For OODBs to make a major impact on the database market, following has to be done:

1. OODBs have to be made full-fledged database systems, sufficiently compatible with RDBs – a migration path is needed to allow the coexistence and the gradual migration from the current products to new products;
2. Application development tools and database access tools have to be developed for such database systems;
3. Architectures of the RDBs and OODBs have to be unified;
4. The data models of the RDBs and OODBs have to be unified [26].

Minimal query optimization

One of the biggest problems in OODBs is the optimization of declarative queries. The additional complexity of the object-oriented data model (OODM) complicates the optimization of OODBs queries [16]. This additional complexity is due to:

1. Additional data types – The user definition of new types and classes through inheritance can both assist and thwart optimization of queries. An example of where it helps could be a query involving the intersection of `Employees` and `Supervisors`. If `Employee` is a superclass of `Supervisor`, the optimizer can assume that `Supervisors` are a proper subset of `Employees` and simplify the join to the set of `Supervisors` [16]. An example of where additional data types deter optimization could involve the union of `Students` and `Employees`, with `Person` being a superclass of both. If we wanted to find all supervisors of students and employees, we would perform the union first and then apply a `supervisor()` [16].
2. Changing variety of types – queries may be based on operations over collections, but optimizations pertaining to sets (or multisets or lists, etc) need to be combined with optimizations over the types of objects contained in the sets. An object-oriented query optimizer must be able to apply optimizations specific to the types, and optimizations that look at relationships between objects of different types [28].
3. Complex objects, methods and encapsulation add to the complexity of query processing in the OODBs. Complex objects create path expressions that complicate query processing. The building of indices for path expressions, especially in the face of arbitrary methods in the path complicates query processing. This is an even harder problem if methods have side-effects. Another problem with path expressions is that they suggest an execution order of the path methods, which may be a very inefficient order. As an example, the path `Orders.part.name` may best be evaluated right to left if there are many orders but few parts, and vice versa if there are many parts with few orders. Also, a path may sometimes be more efficiently processed using a Join. Consider, for example, a query involving the path `s.comp.name` where `s` is in `students`. It might be more efficient (if there are few companies, `comp`) to first compute the `name` property for each `company` and store this result in a tuple. The part from `student` to a `company` name would then involve joining `students` with the set of tuples by matching the `comp` property of a `student` with the `company` attribute of a tuple.
4. OODBs query languages support the use of nested structures, which may again highly complicate the optimization process, turning it from a local problem to a global one – requiring global knowledge of the entire query expression.
5. Object identity – when objects have identities, there is a question as to what constitutes equality of two objects [28]. This carries over to the language where equality operations are used in predicates and where a decision must be made

concerning the creation of new objects by a query. The optimizer for object-oriented models must be able to deal with the creation of new objects and with alternative definitions for equivalence.

Due to all the problems discussed above, optimization of object-oriented queries is extremely hard to solve and is still in the research stage. Today's OODBs offer rather simple optimization strategies. The optimization of joins is also another issue that needs more attention.

Lack of standard query algebra

Lack of query algebra standards is another major weakness of OODBs. This also impairs query optimization. Several different formal query languages of algebras and calculi which have been proposed for OODBs [36], [35], [17], [27] and [18]. These algebras and calculi differ in several respects and expressibility and support for optimizing rewrite rules. Most of these algebras are variable based, i.e. use variables for temporary results. One OODB package, KOLA, is purely functional and variable free. In [11] the author argues that KOLA algebra allows more powerful rule systems to be built due to its variable freeness.

In RDBs, there is close correspondence between algebra operations and low level primitives of the physical system[31]. The mappings between relations and files, and tuples and records have contributed to this strong correspondence. However, in OODBs there is no analogous, intuitive correspondence between object algebra operators and physical system primitives. Any discussion of execution plan generation too, must first define the low level object manipulation primitives [35].

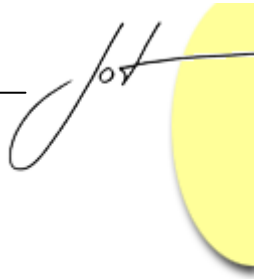
Lack of query facilities

Most OODBs suffer from the lack of query facilities [26]. In those few systems that provide significant query facilities, the query language is not ANSI SQL compatible. The query facilities do not include nested sub-queries, set queries (union, intersection, difference), aggregation functions and GROUP BY, or joins of multiple classes – facilities fully supported in the RDBs [26].

Also, there is no object query standard, though there have been efforts to come up with an object SQL [15]. SQL3 maybe still a few years away [26].

No support for views in OODBs

OODBs do not support views. Although there have been several proposals [1], [14], [19], [13], [29], there is little agreement as to how a view mechanism should operate in OODBs. The development of an object-oriented view capability is complicated by such model features as object identity. What are the identities of the objects in a view? On the other hand, there has also been the argument that data encapsulation and inheritance make explicit view definitions unnecessary [15].



Security concerns with OODBs

While RDBs support authorization, most OODBs do not support authorization [26]. RDBs allow users to grant and revoke privileges to read or change the definitions and tuples in relations and views [26]. If OODBs are going to expand into more business oriented fields, this feature has to be improved.

Some OODBs require users to explicitly set and release locks. RDBs automatically set and release locks in user processing query and update statements [26].

No support for dynamic class definition changes with OODBs

In addition to the fact that no single standard data model has yet been developed for OODBs, most OODBs do not allow dynamic changes to the database schema, such as adding a new attribute or method to a class, adding a new superclass to a class, dropping a superclass from a class, adding a new class, and dropping a class. RDBs allow the user to dynamically change the database schema using the ALTER command; a new column may be added to a relation, a relation may be dropped, a column can sometimes be dropped from a relation [26].

Most OODBs do not offer automatic management of class extensions either. If a class extension is needed, the user has to define a collection for it and keep it up to date on insertions and deletions[15].

Limited support for consistency constraints in OODBs

There are no mechanisms to declare key properties of attributes (for example, an attribute of a class cannot be declared the primary key of the class) or uniqueness constraints, explicit consistency constraints, pre and postconditions of methods [15]. Although all of this could be done using methods, explicit consistency constraints would be more user friendly, less error prone, and more easily accessible for inspection and modification.

Limited performance tuning capability in OODBs

Most of the OODBs offer limited capabilities for parameterized performance tuning [26]. RDBs allow the installer to tune system performance by providing a large number of parameters that can be set by the system administrator. The parameters include the number of memory buffers, the amount of free space reserved per data page for future insertions of data, and so forth [26].

Little support for complex objects

The full functionality of complex objects is not yet fully supported. One can navigate across the reference and code one's own operations using it, but there are no predefined generic operations exploiting different reference semantics. All references are to independent objects, and the semantics of special relationships within complex objects are hidden within the user-supplied operations [15].

Limited integration with existing OO programming systems

It is difficult to re-write object-oriented programs for persistent data management. Several problems arise: 1) naming conflict; 2) class hierarchies have to rewritten; 3) OODBs tend to overwrite system operations [15].

Limited performance gain over RDBs

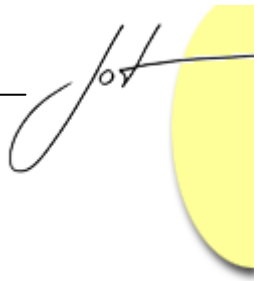
If all database applications required only OID lookups with database objects or memory-pointers chasing other objects in memory, two to three orders of magnitude performance advantage for OODBs over RDBs would be valid [26]. However, most applications that require OID lookups also have database access and update requirements that RDBs have been designed to meet. These requirements include bulk database loading; creation, update, and delete of individual objects (one at a time); retrieval from a class of one or more objects that satisfy certain search conditions; joins of more than one class; transaction commit, and so forth. For such applications, OODBs do not have any performance advantages over RDBs.

Other features that OODBs do not yet support

Examples of other features that OODBs do not yet support are triggers, meta data management features [15], constraints such as UNIQUE and NULL [26].

5 CONCLUSION

Due to the weaknesses of OODBs discussed above, OODBs have not been able to keep up with the expectations of providing all the important features that targeted OODB application areas would like to use. The term OODB has become a misnomer for most current OODBs. Most current OODBs are closer to being merely persistent storage systems for some object-oriented programming language than database systems [26]. So, though the OODM is richer than the relational data model in many respects, the OODM has not matured enough, and to date, the weaknesses of OODB systems outweigh the achievement of OODB systems.



REFERENCES

[1] Abiteboul, S. and Bonner, A "Objects and views," in *Proc. ACM SIGMOD Int. Conf. On Management of Data*, pp. 238-247, 1991.

[2] Atkinson, M., et. al., "The object-oriented database system manifesto," in *Proc. Int. Conf. On Deductive and Object-Oriented Databases*, 1989.

[3] Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D, and Zdonik, S. "The Object-Oriented Database System Manifesto," in Bancilhon et. al. (eds.), *Building an Object-Oriented Database System: The Story of O2*. Morgan Kaufman, 1992.

[4] Bancilhon, F., "Object Oriented database systems," in *Proc. 7th ACM SIGART/SIGMOD Conf*., 1988.

[5] Banerjee, J., et. al., "Data model issues for object oriented applications," *ACM Trans. On Office Information Systems*, vol. 5, no. 1, Jan 1987.

[6] Banerjee, J., Kim, W., and Kim, K.C., "Queries in object oriented databases," in *Proc. IEEE Data Engineering Conf.,* Feb. 1988.

[7] Beech, D., "A Foundation for evolution and relational to object databases," in *Proc. Extended Data Base Technology*, Mar. 1988.

[8] Bertino, E., Negri, M., Pelagatti, G., and Sbattella, L., "Object-Oriented Query Languages: The Notion and the Issues," *IEEE Transactions on Knowledge and Data Engineering,* Vol. 4, No. 3, 1992.

[9] Brown, A.W., *Object-Oriented Databases, Applications in Software Engineering.* New York: McGraw-Hill, 1991.

[10] Cattell, R.G.G., *Object Data Management, Object-Oriented and Extended Relational Database Systems*. Reading, MA: Addison-Wesley, 1991.

[11] Cherniack, M., "Form(ers) over Function(s): The KOLA Query Algebra," Technical Report, Brown University, December, 1995.

[12] Cluet, S., et. al., "Reloop, An algebra based query language for an object-oriented database system," in *Proc. 1st Int. Conf. On Deductive and Object Oriented Databases*, Dec. 1989.

[13] Cruz, I.F., DOODLE: A visual language for object-oriented databases. In *Proc. ACM SIGMOD Int. Conf. On Management of Data*, pp. 71-80, 1992.

[14] Dayal, U., "Queries and views in an object-oriented data model," in *Proc. 2nd Int. Work. On Database Programming Languages*, June 1989.

[15] Dittrich, K.A., and Dittrich, K.R., "Where Object-Oriented DBMSs Should Do Better: A Critique Based on Early Experiences," in *Modern Database Systems:*

The Object Model, Interoperability and Beyond, pp. 238-252, Kim, W., ed., ACM Press, Addison Wesley, 1995.

[16] Erlingsson, U., "Object-Qriented Query Optimization," unpublished manuscript.

[17] Fegaras, L., and Maier, D., "Towards an Effective Calculus for Object Query Languages," *ACM SIGMOD International Conference on Management of Data*, San Jose, California, May, 1995.

[18] Fegaras, L., Maier, D. and Sheard, T., "Specifying Rule-based Query Optimizers in a Reflective Framework," in *Proc. of the 3rd International Conference on Deductive and Object-Oriented Databases*, Phoenix, Arizona, December, 1993.

[19] Heiler, S. and Zdonik, S., "Object Views: Extending the vision," in *Proc. 6th Int. Conf. On Data Engineering*, pp. 86-93, 1990.

[20] Hughes, J.G., *Object-Oriented Databases*. New York: Prentice –Hall, 1991.

[21] Khoshafian, S., "Insight Into Object-Oriented Databases," *Information and Software Technology*, vol. 32, no.4, 1990.

[22] Khoshafian, S. *Object-Oriented Databases*, New York: John Wiley & Sons, 1993.

[23] Khoshafian, S. and Copeland, G., "Object identity," in *Proc. 1st Int. Conf. On Object-Oriented Programming Systems, Languages, and Applications*, Oct. 1986.

[24] Kim, W., "A foundation for object-oriented databases", MCC Tech. Rep., N. ACA-ST-248-88, Aug. 1988.

[25] Kim, W., *Introduction to Object-Oriented Databases*. Cambridge, MA: The MIT Press, 1991.

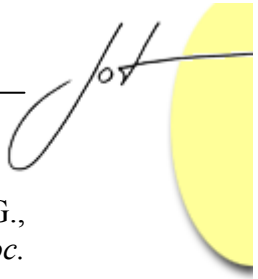
[26] Kim, W., "Object-Oriented Database Systems: Promises, Reality, and Future," in *Modern Database Systems: The Object Model, Interoperability and Beyond*, pp. 255-280, Kim, W., ed., ACM Press, Addison Wesley, 1995.

[27] Leung, T.W., Mitchell, G., Subramanian, B., Vance, B., Vandenberg, S.L. and Zdonik, S.B., "The Aqua Data Model and Algebra," Technical Report CS-93-09, Brown University, March, 1993.

[28] Mitchell, G., Zdonik, S.B., and Dayal, U., "Object-Oriented Query Optimization – What's the Problem?," Technical Report CS-91-41, Brown University, June, 1991.

[29] Rudensteiner, E.A., "Multiview: A methodology for supporting multiple views in object-oriented databases," in *Proc. 18th Int. Conf. On Very Large Databases*, pp. 187-198, 1992.

[30] Scholl, M. and Schek, H., "A relational object model," in Abiteboul, S. and Kanellakis, P.C., eds., in *Proc. 3rd Int. Conf. On Database Theory, volume 470 of Lecture Notes in Computer Science*, pp. 89-105, Springer Verlag, 1990.



[31] Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A., and Price, T.G., "Access path selection in a relational database management system," in *Proc. ACM SIGMOD Int. Conf. On Management of Data*, pp. 23-34, 1979.

[32] Stefik, M. and Bobrow, D.G, "Object-oriented programming: Themes and variations," *The AI Mag*., Jan 1986.

[33] Stonebraker, M., Rowe, L., Lindsay, B., Gray, J., and Carey, M. "Third-Generation Data Base System Manifesto." Memorandum N. UCB/ELB. M90/23, April. The Committee for Advanced DBMS Function, University of California, Berkeley, CA, 1990.

[34] Straube, D.D. and Ozsu, M.T., "Queries and query processing in object-oriented database sytems," *ACM Transactions on Information Systems,* vol. 8, no. 4, pp.387-430, October, 1990.

[35] Straube, D.D. and Ozsu, M.T., "Execution Plan Generation for an Object-Oriented Data Model," *Proceedings of the 2nd International Conference on Deductive and Object-Oriented Databases*, pp. 43, Munich, Germany, December 1991.

[36] Su, S.Y.W., Guo, M. and Lam, H., "Association Algebra: A Mathematical Foundation for Object-Oriented Databases," IEEE Transactions on Knowledge and Data Engineering, vol. 5, no. 5, pp. 775, October, 1993.

[37] Zdonik, S.B. and Maier, D., eds., *Readings in Object-Oriented Database Systems*, San Mateo, CA: Morgan Kauffman, 1989.

[38] Zdonik, S.B., and Wegner, P. "Language and Methodology for Object-Oriented Database Environments" in *Proc. of the Hawaii International Conference on System Sciences*, January, 1986.

About the author

Sikha Bagui is a Lecturer in the Department of Computer Science at the University of West Florida, Pensacola, Florida. She is the co-author of three books in databases, entitled, "*Learning SQL: A Step-by-Step Guide Using Oracle*", "*Learning SQL: A Step-by-Step Guide Using Access*" by Addison Wesley, and "*Database Design Using Entity Relationship Diagrams"* by CRC press. She has also published many research articles in journals such as Pattern Recognition, Database Management, Oracle Internals, Journal of Multimedia and Hypermedia. Her teaching interests include Database Systems, Advanced Databases, Data Mining, and programming languages. Email: bagui@uwf.edu.