Object-Oriented Database Technology

Robert Milton Underwood, Jr.

Table of Contents

Section	Page
Abstract	3
Benefits of object-oriented technology	6
Relational vs. object-oriented database technology	8
Approaches to building object-oriented databases	10
Required features of object-oriented databases	11
It should be a DBMS	11
It should be an object-oriented system	13
Optional features of object-oriented databases	17
Other choices to consider during system design	18
Conclusion	19
References	21
Appendix	23

Abstract

As advances in computer-related technology improve, increasingly larger files are able to be created, transmitted, and stored electronically. It thus becomes more apparent that object-oriented technology, and object-oriented databases in particular, are needed to warehouse the files or "objects." About 88 percent of organizations use relational databases, yet about 55 percent plan to acquire object-oriented databases at some point in the future (Betts, 1997). Management should therefore plan carefully and understand the benefits of object-oriented technology. Technology experts within a firm should also understand the essential elements of object-oriented databases so that they can decide whether or not to recommend to management the adoption of a pure object-oriented system, or a hybrid system.

Object-Oriented Database Technology

Most widely used database software uses some form of client/server technology with relational databases. The relational database paradigm, whether centralized or distributed, maintains that only data, and not procedures, should be stored. A major objective of conventional database technology is to make the data completely independent from the procedures (Martin & Leben, 1995). A recently developed form of database, called an object-oriented database, is becoming more frequently used. In contrast to the relational database model, an object-oriented database stores *objects*, which consist of data as well as procedures (a.k.a., methods[1]) that are used to perform operations on that data.

In general terms, an *object* has a group of characteristics that can be stored as data, which can then be processed as information in a number of ways (Hernandez, 1997). Large-scale database management systems (DBMS) are increasingly in demand to support groups of users in collaborative work environments (Huh & Kim & Chung, 1999). Since database models change along with the reality that is captured in them, their dependent models and views should also evolve.

Object-oriented approaches to programming were introduced as early as 1966 with the emergence of the simulation language Simula67 (Schach, 1996). At the time Simula67 was introduced, the technology was considered too radical for practical use. It

[1] According to Martin and Leben (1995), a *method* specifies the way in which an operation that is performed on the data associated with an object is encoded in computer software. A method consists of procedures that can be executed in a computer to access or manipulate the data associated with an object type.

basically lay dormant until the early 1980's when it was essentially resurrected within the context of modularity.

The concept of *inheritance* was first introduced in Simula67. Inheritance is supported by most object-oriented programming languages, such as C++. The benefit of the concept of inheritance is that new data types can be defined as *extensions* of previously defined types, thereby avoiding the need to have to define new data types from scratch for each new project. To put it simply, inheritance derives a new data type from an existing data type.

The two main reasons for the rapid increase in interest in object-oriented programming over the last several years for applications software development have been (a) the wider availability of these languages plus supporting environments for the object-oriented paradigm, and (b) the emergence of increasingly higher powered hardware (Henderson-Sellers & Edwards, 1994).

A formal proposal for what constitutes an object-oriented database was developed in 1989 at the First International Conference on Deductive and Object-oriented Databases (Henderson-Sellers & Edwards, 1994). What was agreed upon at and subsequent to the conference was a list of the features that an object-oriented database should support (see Appendix).

According to Martin and Leben (1995), object-oriented databases will probably coexist, rather than replace, relational databases, because each has features that are beneficial. The important features of relational databases include security and integrity, but businesses cannot operate while relying only on the data types available in relational databases as they represent only about 10 percent of the data that is available for storage

(Betts, 1997). Some companies may choose to hybridize by putting pure object front-ends on relational databases, thereby gaining the development benefits of objects and the security benefits of relational databases.

Benefits of object-oriented technology

The technology for object-oriented databases first evolved from a need to support object-oriented programming. C++ programmers needed a store for data that remained after a process terminated. According to Martin and Leben (1995), object-oriented databases became important for certain types of applications with complex data, such as computer-aided design (CAD) and computer-aided engineering (CAE). Understanding the benefits (see Table 1) of object-oriented technology over older technologies (Martin & Odell, 1992) can help management understand the potential strengths of object-oriented databases.

Table 1
Benefits of Object-oriented Technology

Designers think in terms of behavior of objects, not small details. Encapsulation[2] allows the small details to be hidden and makes complex classes[3] easy to use.
Classes are designed so that they can be reused again and again. To make the most or reuse, classes can be built so that they can be customized.
Classes designed for repeated reuse are more stable.
Software built from stable classes is likely to have fewer bugs than software built from scratch
Applications are created from preexisting parts, thereby improving the speed of design time.
Designs are often of higher quality because they are created from already-proven components.
Programs built of smaller pieces are easier to create.
Maintenance may be easier because the maintenance programmer usually changes one method of one class at a time.
Object-oriented CASE tools make large changes during the mid-lifecycle of development easier. They also make system refinement easier at all stages of development.

[2] Encapsulation designs a data type's representation, and also its *behavior*, in one encapsulated entity (Stevens, 1994).
[3] According to Connolly and Begg (1999), classes are blueprints for defining a set of similar objects. Objects that have the same attributes can be grouped together to form a class.

Object-oriented analysis may model the enterprise or application area in a way that is closer to reality than conventional analysis.
Object-oriented methodologies encourage better communication between programmers and lay clients, because clients think in terms of objects and events rather than in traditional programming structures.
A graphical user interface is beneficial because it is easier to click on an icon than to remember numerous commands.
Classes are designed to be independent of platforms, hardware, and software environments.
It may be easier to share software from numerous vendors. Also, software developed independently by different vendors should be able to work together and appear as a single unit to the end user.
Benefits can be realized in client-server computing: a server class may be used by many different clients, and these clients access server data with the class methods, thereby helping ensure that the data is not corrupted.
Object-oriented design is the key to large-scale distributed computing. Classes in one machine will interact with classes in other machines without needing to know where the classes reside.
Parallel computing means that objects on different processors will be able to execute at the same time, each acting independently.
There is a higher level of database automation with an object-oriented design. The data structures in object-oriented databases are linked to methods that take automatic actions. In a sense, an object-oriented database has intelligence built into it in the form of methods, whereas a basic relational database does not.
Object-oriented databases have demonstrated significantly higher performance than relational databases for certain applications with very complex data structures.
Businesses can conveniently create their own libraries of classes that reflect their company standards and application needs, and can add to these libraries as policies and needs change.

Despite the benefits of object-oriented technology, it may take some businesses longer to adapt than others. Consider companies in the insurance industry. According to representatives of Miller Freeman, Inc. (1999), many insurance companies would like to convert from COBOL mainframe-based applications to component-based object-oriented frameworks, but are indecisive due to the complexity and costs of such a project. One strategy, perhaps, is to externalize certain business functions such as billing, rating, reporting, and underwriting, by building these as stand-alone components through object-oriented development technology and object-oriented programming.

Relational vs. object-oriented database technology

Object-oriented databases have been designed to support diverse data types rather than just the simple tables, columns and rows of relational databases. Image files, sound clips, video clips, and unformatted text are the data types that object-oriented databases have been designed to store.

There are numerous differences (Martin & Leben, 1995) between relational and object-oriented database technology. For instance, the main goal of relational database technology is to be able to have data be physically reorganized without affecting how it is used. The main goal of object-oriented database technology is encapsulation and class independence; classes can be reorganized without affecting how they are used.

Companies considering adopting applications with object-oriented technology should make sure that they understand what their needs are. In an object-oriented database management system, the characteristics of object-orientation need to be combined with database management requirements including persistency, secondary storage management, concurrency, recovery, and an ad-hoc query facility (Henderson-Sellers, 1992).

According to Baer (1999), object-oriented databases can outperform relational databases at working with complex relationships among data. A relational database, which must store data in columns, rows, and tables, is usually a suitable choice when relationships among different data types are fairly fixed. But for very complex queries, such as those performed by engineers who must consider numerous contingencies and possibilities during design phases, a relational database may become overloaded. The problem for management becomes more critical when the actual business of the company

changes more quickly than a database can be redesigned to deliver needed information in a timely manner. In many industries deregulation, increased competition, and the demand for improved customer service are creating the complex data relationships for which object-oriented technology and object-oriented databases can excel (Baer, 1999). An object-oriented database system is a desirable alternative to a relational database system in certain industries with applications that require tremendous modeling power, especially those industries with companies needing statistical and scientific databases. These applications typically maintain large amounts of data, and additionally, often want to keep old versions of data (Nørvåg, 1999).

One useful application of object-oriented database technology is with signature files. A signature is a bit string, which is generated by applying some hash function[4] on some or all of the attributes of an object. In many of the new application areas for database systems, data is viewed as a collection of objects. Another typical characteristic of the new applications is that a large percentage of query accesses are perfect match accesses on one or more words in text strings from the objects or rows in the databases. For these types of accesses, signature files can be used to reduce the query cost (Nørvåg, 2000). The main disadvantage of traditional signature files is that the signature file maintenance can be costly. If one of the attributes contributing to the signature in an object is modified, the signature file also has to be updated. With an object-oriented database, instead of storing the signatures in distinct signature files, the signatures are stored with the objects. Management can realize savings with this method. The use of signatures is quite clear-cut in queries involving only value attributes. But in an object-

[4] A hash function, according to Connolly and Begg (1999), "calculates the address of the page in which a record is to be stored based on one or more fields in the record." Therefore, with hash files, "records do not have to be written sequentially to the file."

oriented database, it is also possible to access data from a general programming language like C++ or Java (Nørvåg, 2000).

Approaches to building object-oriented databases

Database applications that do not work effectively, that do not meet business requirements, or that hinder business objectives, cannot be of much help to a company (Schur, 1994). It is therefore of critical importance that the database chosen for a company be designed in a fashion that is complimentary to the company objectives. Management should plan well prior to accepting or creating a database system, and should communicate with representatives from all levels of the corporate hierarchy, from upper management to technology staff to end-users, to make sure that all pertinent design criteria are considered and understood. According to Martin and Leben (1995), there are three main approaches to creating object-oriented databases. The first approach considers the use of conventional and existing database software. The database developer using this method would add a layer for processing object-oriented requests and for storing methods. There are two positive benefits of this approach. The first benefit is that already existing code can be used, allowing the object-oriented database to be implemented quicker. The second benefit follows directly from the first benefit in that cost savings are realized when the development is quicker.

The second main approach to creating object-oriented databases is to add capabilities to existing database software. With this approach, the existing tools and techniques associated with relational database technology can be used to create a new object-oriented database. Pointers may be added to relational tables that link them to large binary objects (e.g. video images).

The third main approach to creating object-oriented databases is to produce a completely new architecture that is made to accommodate the full features of object-oriented technology. While the market may be somewhat limited at the present time for databases that are considered completely within the parameters of object-oriented technology, there are numerous software companies who have gone ahead with the development and marketing of such products.

Required features of object-oriented databases

There are certain main features and characteristics that a system must have to qualify as an object-oriented database system (Atkinson & Bancilhon & DeWitt & Dittrich & Maier & Zdonik, 1995). Managers need to understand these features and the optional features described in the section following this one so that they can make sound decisions if the company should decide to adopt an object-oriented DBMS. Required features are features that a system absolutely *must have* in order to be considered an object-oriented database system. An object-oriented system must satisfy two broad criteria: (a) it should be a DBMS, and (b) it should be an object-oriented system. The features of each broad criterion are described below.

It should be a DBMS

The system should be consistent with features necessary for a true DBMS. In other words, it is not just a database where files are stored and retrieved, it is a database management system which has all of the basic features of a database plus the capabilities to be a system that management can use to aid in the decision-making process. A DBMS should have the following.

Persistence Persistence is the ability of the programmer to have data survive the execution of a process, in order to be able to reuse it in another future process. Persistence should be orthogonal (Atkinson, et. al, 1995), which means that each object should be allowed to become persistent, independent of its type. Another feature of persistence is that the user should not have to use *move* or *copy* commands to make the data persistent.

Secondary storage management Secondary storage management is usually supported through the features of index management, data clustering, data buffering, access path selection, and query optimization. The end-user does not see these features because they are just performance features. It is important that these certain features are invisible in order to allow the programmer to avoid having to write code to maintain indices, allocate disk storage, or move data between disk and main memory. This invisibility effectively ensures that there is a clear independence between the logical and physical level of the database system.

Concurrency Concurrency is the ability of numerous users to be able to access the system at the same time. There should be an agreeable coexistence with controlled sharing among users as they work at the same time on the database system.

Recovery If the hardware or software experiences failure, it should recover by being able to bring itself back to a stable state with reasonably coherent data.

Ad Hoc Query Facility The point of having an ad hoc query facility is to have a simple way of querying data. It does not necessarily have to be done with a query language, because other methods (e.g. like having a graphical browser) could suffice. The main idea is to have the user be able to ask simple queries to the database system in a

simple form. This facility could be supported by the Data Manipulation Language, or a form of it. According to Atkinson, et al, (1995) the ad hoc query facility should comply with three criteria: (a) It should be high level and should be able to express simple queries concisely. (b) It should be efficient. That means that it should be able to be optimized. (c) It should be independent of applications, and should be able to work on any type of database. This last criterion eliminates writing additional operations on each user-defined type.

It should be an object-oriented system

The second broad criteria that a database system must satisfy in order to be considered an object-oriented database is that it should be a true object-oriented system. Benefits of object-oriented technology were identified in Table 1. Mandatory object-oriented features present in databases would include the following.

Complex objects Complex objects are built from simpler objects by applying constructors to them. Simple objects include objects such as integers, characters, and byte strings. Constructors that the object-oriented DBMS should have include sets, arrays, and tuples (a.k.a. rows). Sets are important because they are a typical way of representing collections. Arrays are important because they capture order in data. Tuples are important because they are a natural way of indicating the properties of an entity. One other thing to note about constructors is that they should apply to any object. Constructors of relational databases do not apply to all objects.

Object identity While object identity has been common for a long time in programming languages, it has only recently been seen in databases. The essence of object identity is that an object has an independent existence from its value. There are

two ways to look at object equivalence: (a) two objects can be identical, or (b) they can have the same value. Two implications follow this representation: *object sharing* and *object updates*. *Object sharing* indicates that two objects can share a component. Consider the example of two Realtors that made sales at 123 Main Street. Three interpretations may exist to define the reality of the situation: (a) They either sold the same house, albeit at different times, (b) they sold the same house at the same time working jointly, or (c) they sold two houses with the same street address in two different cities. In an identity-based model the different possibilities can share the common part. The quality of *object updating* allows objects to be updated if they share a relationship. In the example mentioned above, if the two Realtors did indeed sell the same home working cooperatively, then updates to one Realtor's file (i.e. value) would also update the other Realtor's file. With relational databases (i.e. value-based systems), the sub-objects would have to be individually updated.

Encapsulation An object should encapsulate both program and data. Encapsulation provides a form of logical data independence wherein the implementation of a type can be changed without any of the programs using that type. What results is that the application programs are protected from implementation changes in the lower layers of the database system. The appropriate encapsulation is obtained when only the operations are visible and the data and the implementation of the operations are hidden in the objects.

Types and Classes There are two main categories of object-oriented systems, those supporting the concept of class, and those supporting the concept of type. A *type* summarizes the common features of a set of objects with the same characteristics. It has

two parts, the interface and the implementation. Only the interface part is visible to the end users. A *class* has similar specifications as *type* but it is more of a run-time concept. There are two aspects of class: an object factory and an object warehouse. As its name implies, an object factory is used to create objects. The object warehouse refers to the attaching to the class of the set of objects that are instances of the class. The end-user can change the warehouse by applying operations on all elements of the class.

Class or Type Hierarchies The basic assumption with this feature is that classes or types should be able to inherit features, in code, from previously defined programs. Inheritance allows the building of an orderly hierarchy of classes, and when several abstract data types have characteristics in common, programmers can design their common qualities into a single base class and divide their unique characteristics into unique derived classes. (Stevens, 1994). Inheritance helps to give a precise description of the real world and it helps in factoring out shared specifications and implementations in applications. Inheritance facilitates in making code reusable, because every program is at the level at which the largest number of objects can share it. According to Atkinson, et. al. (1995), there are four types of inheritance: (a) substitution, (b) inclusion, (c) constraint, and (d) specialization.

Overriding, overloading, and late binding In an object-oriented system, the display operation is defined at the object type level. The display that the end-user sees, for example, could be in the form of a graph, or a bitmap image. The display can be used differently on graphs and pictures, and yet the implementation of the operation is redefined for each of the types according to type. The redefinition is referred to as overriding. The redefinition results in a single display denoting different programs

(referred to as overloading). The display operation is applied to each one of the different types of elements, and the system picks the appropriate implementation at run-time. An advantage noticed is that the application programmer does not have to worry about needing a different program for graphs and for images. Also, the code is simpler since there is not case statement on types. In order to offer the functionality described, the system cannot bind operation names to programs at the time of compilation. That means that operation names must be translated into program addresses at run-time. This delayed translation is referred to as late-binding.

Computational completeness Computational completeness simply means that one can express any computable function, using the Data Manipulation Language of the database system. Some languages are not considered complete. Structured Query Language (SQL) is an example of a language that is not considered complete. But just because a language is not complete (with regards to computations) does not mean that new languages have to be written. Computational completeness can be introduced through a reasonable connection to existing programming languages.

Extensibility Predefined types are part of a database system, and these types can be manipulated by programmers who create applications for the system. Extensibility means that there should be no distinction in usage between system-defined and user-defined types. While there probably may be a difference in the way system-defined and user-defined types are supported by the system, the differences should be indiscernible to the application and to the programmer.

The eight features summarized above are not necessarily a part of traditional relational databases, yet are mandatory features for a DBMS to be considered object-oriented.

Optional features of object-oriented databases

There are certain features that are not considered mandatory, but can be included to markedly improve a DBMS.

Multiple inheritance The quality of multiple inheritance is of an object-oriented nature, but it is not a mandatory feature. It does, however, make the system *more* object-oriented. If a system does support multiple inheritance, there will be numerous possible solutions for facing the problem of conflict resolution.

Type checking and type inferencing Typically, the greater the degree of type checking that the system will perform at the time of compiling, the better. The same goes for type inferencing. More type inferencing is better for a system, and the best situation is the one in which only the base types have to be declared while the system infers the temporary ones.

Distribution The database can be a distributed system, or not, depending upon the needs of the company and end-users.

Design transactions In most new applications, the transaction model of the classical business-oriented database system is not satisfactory, as transactions tend to be very long and the usual serializability criterion is not adequate. Therefore, many object-oriented databases support design transactions (viz. long transactions or nested transactions).

Versions Most of the new applications (e.g. CAD and CASE) involve a design activity and require some form of versioning. Therefore, many object-oriented databases support versions. Again, offering a versioning mechanism is not part of the mandatory requirements for the system, but it is a desirable feature to have.

Other choices to consider during system design

Programming paradigm There are different programming paradigms that can be used depending upon the preferences of the programmer and based upon the needs of the end-users. Programmers could choose the logic programming style, the functional programming style, or the imperative programming style. One possible option is that the DBMS be independent of the programming style, and would therefore support multiple programming paradigms. The choice of syntax is also open to the preferences of the programmer.

Representation system The representation system is defined by the set of constructors. Constructors include sets and lists, but they can be extended in many different ways.

Type system Encapsulation was a type facility that was considered mandatory, but other type formers (e.g. generic type, or arbitrary type) are flexible in choice. Also, programmers may want the option of whether or not the type system is considered second order.

Uniformity It may be unclear to database experts as to what level of uniformity one should expect of object-oriented systems. For instance, should a type be considered an object, or should a method be considered an object? The issue should be examined at three different levels. At the implementation level, the programmer must decide whether

type information should be stored as objects or whether an ad hoc system should be implemented. At the programming language level, it should be determined whether or not types are first class entities in the semantics of the programming language. At the interface level, it should be determined by management whether or not to present the end-user with a uniform view of types, objects, and methods.

Conclusion

An object-oriented DBMS can be simply described as a DBMS with an underlying object-oriented data model. Object-oriented database technology will probably not replace relational database technology, but will build on it. Relational database technology will probably continue to coexist with object-oriented technology, and many executives think that there will need to be open access between relational and object-oriented databases (Betts, 1997). Conventional relational databases will be used for certain computer applications while object-oriented databases will be used for other applications.

Different businesses will have different needs, and will therefore focus on different solutions. Businesses utilizing modern transaction processing systems realize that vast amounts of different types of data from numerous sources must be handled with complete reliability, and this is best achieved using an object-oriented model (Collins, 2000).

If a company is considering adopting an object-oriented database, management must plan carefully and should thoroughly consider all aspects of the decision making process. Plans prior to adoption should include estimates of costs related to initial research, implementation, and follow-up. With a careful study of the features of object-

oriented databases, managers will have available data with which to make prudent decisions. If an object-oriented database is adopted, management and the Information Technology Director of the company should work closely together during the implementation phase. Thoughtful planning will allow for a smooth transition from prior systems to new systems. The ultimate objective of management with regards to having an object-oriented database is to be able to use it as a tool to facilitate success in their chosen industry.

References

Atkinson, M., & Bancilhon, F., & DeWitt, D., & Dittrich, K., & Maier, D., & Zdonik, S. (1995, April). The Object-oriented Database System Manifesto, [On-line].

Baer, T. (1999, January 18). Object databases. Computerworld, [On-line].

Betts, B. (1997, February 13). Objects of desire? Computer Weekly, [On-line].

Collins, R. (2000). Object-oriented transaction processing for mission-critical applications, [On-line].

Connolly, T., Begg, C., & Strachan, A. (1999). Database systems. Harlow, England: Addison-Wesley. 986.

Henderson-Sellers, B. (1992). A book of object-oriented knowledge: object-oriented analysis, design and implementation: a new approach to software engineering (pp. 1-4). New York: Prentice Hall.

Henderson-Sellers, B., and Edwards, J. (1994). Book two of object-oriented knowledge: the working object: object-oriented software engineering: methods and management. (pp. 30-35). Sydney, Australia: Prentice Hall.

Hernandez, M. (1997). Database design for mere mortals. Reading, Massachusetts: Addison-Wesley.

Huh, S., Kim, H., and Chung, Q. (1999, August). Framework for change notification and view synchronization in distributed model management. Omega, [On-line].

Martin, J., & Leben, J. (1995). Client-server databases: enterprise computing. Upper Saddle River, New Jersey: Prentice Hall.

Martin, J., & Odell, J. (1992). Object-oriented analysis & design. Englewood Cliffs, New Jersey: Prentice Hall.

Miller Freeman, Inc. (1999, October). Expert Opinion: short-term strategies for upgrading to component-based, object-oriented technology. Insurance & Technology, [On-line].

Nørvåg, K. (1999). Temporal Object Database Systems. [On-line].

Nørvåg, K. (2000, June 16). Fine-Granularity Signature Caching in Object Database Systems. [On-line].

Schach, S. (1996). Classical and object-oriented software engineering (3rd ed., pp. 140, 170). Chicago: Irwin.

Schur, S. (1994). The database factory: active database for enterprise computing. New York: John Wiley & Sons.

Stevens, A. (1994). Welcome to programming: from mystery to mastery (pp. 260-265). New York: MIS:Press.

Appendix

The following list of possible features for an object-oriented database management system was created at and shortly after the *First International Conference on Deductive and Object-oriented Databases* in 1989 (Henderson-Sellers & Edwards, 1994). The importance of the list is that it reveals the efforts at comprehensiveness with which the authors of the list gave to the model that would lead to the future development of object-oriented databases.

TRANSACTION PROPERTIES
- Atomicity (everything is done, or nothing is done)
- Consistency (a transaction changes a database from one consistent state to another consistent state)
- Isolation (the internal state of a transaction is not visible to other transactions)
- Durability (the results of a transaction are lasting)
- Long transactions
- Nested transactions
- Shared transactions
- Nonblocking read consistency

LOCKING AND CONCURRENCY CONTROL
- Concurrency control
- Lock escalation
- Promotable locks
- Locks set and release
- Granularity of locks
- Deadlocks

SECURITY AUTHORIZATION
- Role authorization
- Implicit authorization
- Positive authorization
- Negative authorization
- Strong authorization
- Weak authorization
- Authorization objects
- Private database
- Time and day

QUERY CAPABILITY
- Query language
- Query optimization
- Indexing
- Queries in programs
- Queries preferred

Query language completeness
Specification of collections
SQL

DATA INDEPENDENCE/SCHEMA MODIFICATION
Runtime types
Tools for schema transformations
Update time
Changes to attributes or data members
Changes to database methods
Changes to superclass/subclass relationships
Versions of schema

DISTRIBUTED DATABASE SYSTEMS
Location independence
Local autonomy at each site
No reliance on a central site that can result in a single point of reference
Fragmentation independence
Replication independence
Distributed query processing
Distributed transaction management
Hardware independence
Operating system independence
Network independence
Wide area network (WAN) support
Database management system independence
Continuous operation

OBJECT MODEL
Objects
Binding and polymorphism[5]
Encapsulation
Identity
Types and classes
Inheritance and delegation
Relationships and attributes
Literals
Nontraditional objects
Aggregates
Composite or complex objects
Integrity
Schema extensibility
Database operation extensibility

[5] Polymorphism is present when a derived class customizes the behavior of the base class to meet the requirements of the derived class (Stevens, 1994).

- Object languages

ARCHITECTURE

- Architecture type
- Methods execution location
- Query processing location
- Shared access coordination location
- Multiple interfaces
- Memory utilization
- Safety
- Disk media protection
- Backup/recovery facilities
- Change notification
- Version
- Configuration management