Encyclopedia of Database Systems, Editors-in-chief: Özsu, M. Tamer; Liu, Ling, Springer, 2009.

OBJECT DATA MODELS

Susan D. Urban
Arizona State University
http://www.public.asu.edu/~surban

Suzanne W. Dietrich
Arizona State University
http://www.public.asu.edu/~dietrich

SYNONYMS

ODB (Object Database), OODB (Object-Oriented Database), ORDB (Object-Relational Database)

DEFINITION

An object data model provides support for objects as the basis for modeling in a database application. An object is an instance of a class, which is a complex type specification that defines both the state of its instance fields and the behavior provided by its methods. Object features also include a unique object identifier that can be used to refer to the object, as well as the organization of data into class hierarchies that support inheritance of state and behavior. The term object data model encompasses the data model for both object-oriented databases (OODBs) and object-relational databases (ORDBs). OODBs use an object-oriented programming language as the database language and provide inherent support for the persistence of objects with typical database functionality. ORDBs extend relational databases by providing additional support for objects.

HISTORICAL BACKGROUND

The relational data model developed in the 1970's, providing a way to organize data into tables with rows and columns [4]. Relationships between tables were defined by the concept of foreign keys, where a column (or multiple columns) in one table contained a reference to a primary key value (unique identifier) in another table. The simplicity of the relational data model was complemented by its formal foundation on set theory, thus providing powerful algebraic and calculus-based techniques for querying relational data.

Initially, relational data modeling concepts were used in business-oriented applications, where tables provided a natural structure for the organization of data. Users eventually began to experiment with the use of relational database concepts in new application domains, such as engineering design and geographic information systems. These new application areas required the use of complex data types that were not supported by the relational model. Furthermore, database designers were discovering that the process of normalizing data into table form was affecting performance for the retrieval of large, complex, and hierarchically structured data, requiring numerous join conditions to retrieve data from multiple tables. Around the same time, object-oriented programming languages (OOPLs) were also beginning to develop, defining the concept of user-defined classes, with instance fields, methods, and encapsulation for information hiding [14].

The OOPL approach of defining object structure together with object behavior eventually provided the basis for the development of Object-Oriented Database Systems (OODBs) in the mid-1980's. The *Object-Oriented Database System Manifesto*, written by leading researchers in the database field, was the first document to fully outline the characteristics of OODB technology [1]. OODBs provided a revolutionary concept for data modeling, with data objects organized as instances of user-defined classes. Classes were organized into class hierarchies, supporting inheritance of attributes and behavior. OODBs differed from relational technology through the use of internal object identifiers, rather than foreign keys, as a means for defining relationships between classes. OODBs also provided a more seamless integration of database and programming language technology, resolving the impedance mismatch problem that existed for relational database systems. The impedance mismatch problem refers to the disparity that exists between set-oriented relational database access and iterative one-record-at-a-time host language access. In the OODB paradigm, the OOPL provides a uniform, object-oriented view of data, with a single language for accessing the database and implementing the database application.

The relational database research community responded to the development of OODBs with the *Third Generation Database System Manifesto*, defining the manner in which relational technology can be extended to support object-oriented capabilities [12]. Rowe and Stonebraker developed Postgres as the first object-relational database system (ORDB), illustrating an evolutionary approach to integrating object-oriented and relational concepts [10]. ORDB concepts parallel those found in OODBs, with the notion of user-defined data types, object tables formed from user-defined types, hierarchies of user-defined types and object tables, rows of object tables with internal object identifiers, and relationships between object tables that use object identifiers as references.

Today, several OODB products exist in the market, and most relational database products provide some form of ORDB support. The following section elaborates on the common features of object data models and then differentiates between OODB and ORDB modeling concepts.

FOUNDATIONS

Characteristics of Object Data Models

An object is one of the most fundamental concepts of the object data model, where an object represents an entity of interest in a specific application. An object has state, describing the specific structural properties of the object. An object also has behavior, defining the methods that are used to manipulate the object. Each method has a signature that includes the method name as well as the method parameters and types. The state and behavior of an object is expressed through an object type definition, which provides an interface for the object. Objects of the same interface are collected into a class, where each object is viewed as an instance of the class. A class definition supports the concept of encapsulation, separating the specification of a class from the implementation of its methods. The implementation of a method can therefore change without affecting the class interface and the way in which the interface is used in application code.

When an object of a class is instantiated, the object is assigned a unique, internal object identifier, or oid [6]. An oid is immutable, meaning that the value of the identifier cannot be changed. The state of an object, on the other hand, is mutable, meaning that the values of object properties can change. In an object data model, object identity is used as the basis for defining relationships between classes, instead of using object state, as in the relational model. As a result, the values of object properties can freely change without affecting the relationships that exist between objects. Object-based relationships between classes are referred to as object references.

Classes in an object model can be organized into class hierarchies, defining superclass and subclass relationships between classes. A class hierarchy allows for the inheritance of the state and behavior of a class, allowing subclasses to inherit the properties and methods of its superclasses while extending the subclass with additional properties or behavior that is specific to the subclass. Inheritance hierarchies provide a powerful mechanism to represent generalization/specialization relationships between classes, which simplify the specification of an object schema, as well as queries over the schema.

As an example of the above concepts, consider the Publisher application described in Figure 1 using a Unified Modeling Language (UML) class diagram [11]. A Book is a class that is based on an object type that defines the state of a book (isbn, title, and listPrice), as well as the behavior of a book (the method calcBookSales for calculating the total sales of a book based on customer purchases). Publisher, Person, Author, and Customer are additional classes, also having state and behavior. Since authors and customers are specific types of people, the Author and Customer classes are defined to be subclasses of Person. Since personName and address are common to authors and customers, these attributes are defined at the Person level and inherited by instances of the Author and Customer classes. Furthermore, Author introduces additional state and behavior that is specific to authors, defining the date (authorSince) when an author first wrote a book as well as a method (calcAuthorBookSales) for calculating the total sales of the author's books. The Customer class similarly introduces state and behavior that is specific to a customer.

Relationships are also defined between the classes of the application:

- A book is authored by one or more authors; an author writes many books.
- A book is published by one publisher; a publisher publishes many books.
- A book is bought by many customers; a customer buys many books, also recording the date of each purchase.

For each relationship, specific instances of each class are related based on the object identity of each instance. For example, a book will establish a relationship to the publisher of the book using the oid of the publisher. In the relational model, the publisher name would be used as a foreign key to establish the relationship. If the publisher name changes, then the change in name must be propagated to the book that references the publisher. In the object data model, such changes in state do not affect relationships between objects since the relationship is based on an immutable, internal object identity.

A generic object model, such as the one shown in Figure 1, can be mapped to either an object-oriented data model or an object-relational data model. The following subsections use the Publisher application in Figure 1 to illustrate and explain OODB and ORDB approaches to object data modeling.

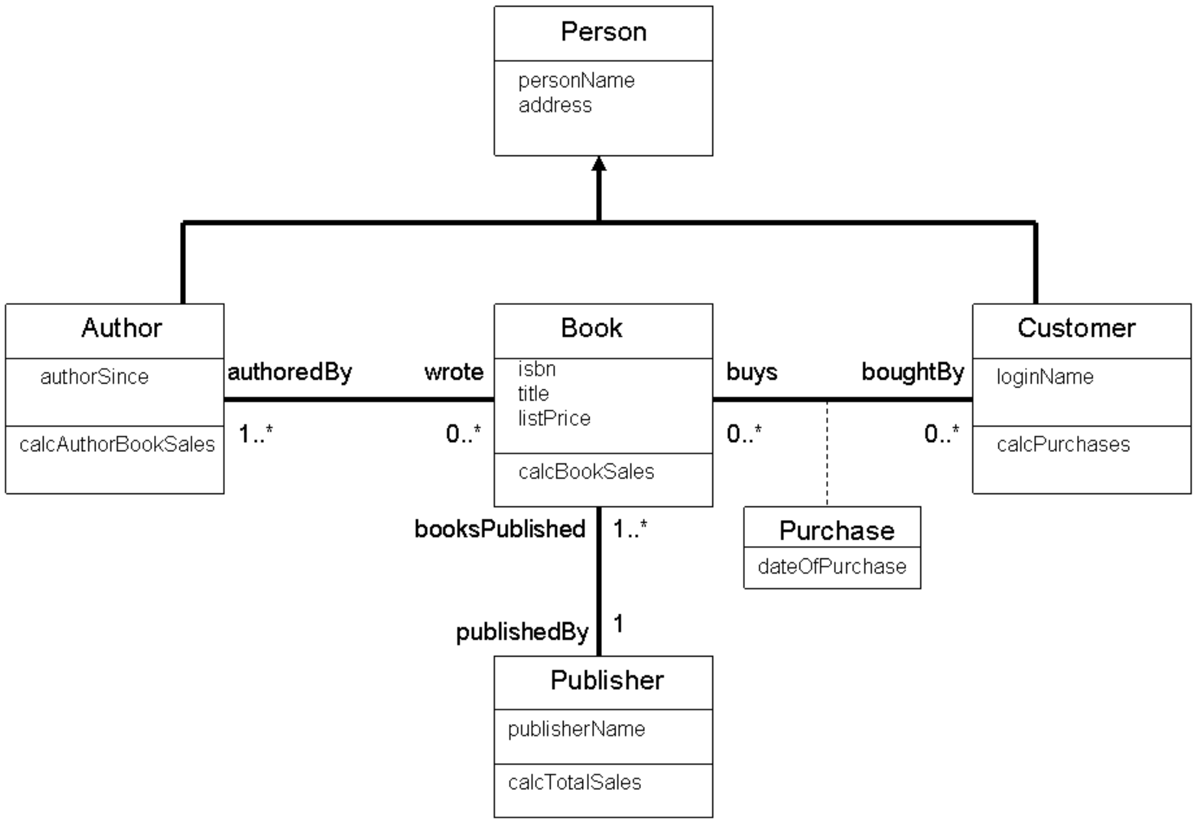

Figure 1: The Publisher Object Data Model

Object-Oriented Data Model
An object-oriented database (OODB) is a term typically used to refer to a database that uses objects as a building block and an object-oriented programming language as the database language. The database system supports the persistence of objects along with the features of concurrency and recovery control with efficient access and an ad hoc query language.

The Object Data Standard [2] developed as a standard to describe an object model, including a definition language for an object schema, and an ad-hoc query language. The object model supports the specification of classes having attributes and relationships between objects and the behavior of the class with methods. The Object Definition Language (ODL) provides a standard language for the specification of an object schema including properties and method signatures. A property is either an attribute, representing an instance field that describes the object, or a relationship, representing associations between objects. In ODL, relationships represent bidirectional associations with the database system being responsible for maintaining the integrity of the inverse association. An attribute can be used to define a unidirectional association. If needed, the association can be derived in the other direction using a method specification. The decision is based on trade-offs of storing and maintaining the association versus deriving the inverse direction on demand.

Figure 2 provides an ODL specification of the Publisher application. Each class has a named extent, which represents the set of objects of that type. The Author and Customer classes inherit from the Person class, extending each subclass with specialized attributes. The Book class has the isbn attribute that forms a key, being a unique value across all books. The association between Author and Book is represented as an inverse relationship, and the cardinality of the association is many-to-many since an author can write many books and a book can be written by multiple authors. The set collection type models multiple books and authors. Since the Purchase association class from Figure 1 has an attribute describing the association, Purchase is modeled in ODL using reification, which is the process of transforming an abstract concept, such as an association, into a class. As a result, the Purchase class in Figure 2 represents the many-to-many association between Book and Customer. Each instance of the Purchase class represents the purchase of a book by a customer. The Purchase class has the dateOfPurchase instance field, as well as two relationships indicating which Book (bookPurchased) and which Customer (purchasedBy) is involved in the purchase. The inverse relationships in Book (boughtBy) and Customer (buys) are related to instances of the Purchase class.

```
class Person
(extent people)
{attribute Name personName,
 attribute AddressType address
};

class Author extends Person
(extent authors)
{attribute Date authorSince,
 relationship set<Book> wrote
    inverse Book::authoredBy,
 public float calcAuthorBookSales();
};

class Customer extends Person
(extent customers)
{attribute string loginName,
relationship set<Purchase> buys
    inverse Purchase::purchasedBy,
public float calcPurchases();
};

class Purchase
(extent purchases)
{attribute Date dateOfPurchase,
 relationship Book bookPurchased
    inverse Book::boughtBy,
 relationship Customer purchasedBy
    inverse Customer::buys;
);
```

```
class Book
(extent books,
 key isbn)
{attribute string isbn,
 attribute string title,
 attribute float listPrice,
 relationship set<Author> authoredBy
    inverse Author::wrote,
relationship set<Purchase> boughtBy
    inverse Purchase::bookPurchased,
relationship Publisher publishedBy
    inverse Publisher::booksPublished,
public float calcBookSales();
};

class Publisher
(extent publishers)
{attribute string publisherName,
 relationship set<Book> booksPublished
   inverse Book::publishedBy,
public float calcTotalSales();
};
```

Figure 2: ODL Schema of the Publisher Application

This ODL specification forms the basis of the definition of the object schema within the particular OOPL used with the OODB, such as C++, Java, and Smalltalk. The specification of the schema and the method implementation using a given OOPL is known as a language binding. In some OODB products, the ODL specification of the properties of the class are used to automatically generate the definition of the schema for the OOPL being used.

The standard also includes a declarative query language known as the Object Query Language (OQL). The OQL is based on the familiar select-from-where syntax of SQL. The select clause defines the structure of the result of the query. The from clause specifies variables that range over collections within the schema, such as a class extent or a multivalued property. The where clause provides restrictions on the properties of the objects that are to be included in the result. Object references are traversed through the use of dot notation for single-valued properties and through the from clause for multivalued properties.

Consider a simple query that finds the name of a publisher of a book given its isbn:

```
select  b.publishedBy.publisherName
from    books b
where   b.isbn = “0-13-042898-1”;
```

This OQL query looks quite similar to SQL. In the from clause, the alias b ranges over the books extent. The where clause locates the book of interest. The select clause provides a path expression that navigates through the publishedBy single-valued property to return the name of the publisher.

Consider another query that finds the title and sales for books published by Springer-Verlag:

```
select  title:   b.title,
        sales:  b.calcBookSales()
from    p in publishers,
        b in p.booksPublished
where   p.publisherName = “Springer-Verlag”
```

This query illustrates the alternative syntax for the alias in the from clause, using the syntax “variable in collection”. The alias p ranges over the publishers extent, whereas the alias b ranges over the multivalued relationship booksPublished of each publisher that satisfies the where condition. The select clause returns the name of each field and its value, where sales returns the results of a method call.

Object-Relational Data Model

An object-relational database (ORDB) refers to a relational database that has evolved by extending its data model to support user-defined types along with additional object features. An ORDB supports the traditional relational table in addition to introducing the concept of a typed table, which is similar to a class in an OODB. A typed table is created based on a user-defined type (UDT), which provides a way to define complex types with support for encapsulation. UDTs and their corresponding typed tables can be formed into class hierarchies with inheritance of state and behavior. The rows (or instances) of a typed table have object identifiers that are referred to as object references. Object references can be used to define relationships between tables that are based on object identity.

Figure 3 presents an ORDB schema of the Publisher application that is defined using the object-relational extensions to the SQL standard. The type personUdt is an example of specifying a UDT. The UDT defines the structure of the type by identifying attributes together with their type definitions. The phrase “instantiable not final ref is system generated” defines three properties of the type:

1. “instantiable” indicates that the type supports a constructor function for the creation of instances of the type. The phrase “not instantiable” can be used in the case where the type has a subtype and instances can only be created at the subtype level.
2. “not final” indicates that the type can be specialized into a subtype. The phrase “final” can be used to indicate that a type cannot be further specialized.
3. “ref is system generated” indicates that the database system is responsible for automatically generating an internal object identifier. The SQL standard supports other options for the generation of object identifiers, which include user-specified object-identifiers as well as identifiers that are derived from other attributes.

Definition of the personUdt type is followed by the specification of the person typed table, which is based on the personUdt type. The person typed table automatically acquires columns for each of the attributes defined in personUdt. In addition, the person typed table has a column for an object identifier that is associated with every row in the table. The phrase “ref is personID” defines that the name of the object identifier column is personID. The definition of a typed table can add constraints to the columns that are defined in the type associated with the table. For example, personName is defined to be a primary key in the person typed table.

```
create type personUdt as
(personName         varchar(15),
 address            varchar(20))
 instantiable not final ref is system generated;

create table person of personUdt
(primary key (personName),
 ref is personID system generated);

create type authorUdt under personUdt as
(authorSince   varchar(10),
 wrote          ref (bookUdt) scope book array [10]
    references are checked on delete no action)
 instantiable not final
 method calcAuthorBookSales() returns decimal;

create table author of authorUdt under person;

create type customerUdt under personUdt as
(loginName     varchar(10),
 buys      ref (purchaseUdt) scope purchase array [50]
    references are checked on delete no action)
 instantiable not final
 method calcPurchases() returns decimal;

create table customer of customerUdt under person
(unique (loginName));

create type publisherUdt as
(publisherName     varchar(30),
 booksPublished    ref (bookUdt) scope book array (1000)
    references are checked on delete set null)
 instantiable not final ref is system generated
 method caclTotalSales() returns decimal;
```

```
create table publisher
(primary key (publisherName),
 ref is publisherID system generated);

create type purchaseUdt as
(dateOfPurchase  date,
 purchasedBy           ref (customerUdt) scope customer
    references are checked on delete cascade,
bookPurchased          ref (bookUdt) scope book
    references are checked on delete cascade)
 instantiable not final ref is system generated;

create table purchase of purchaseUdt
 (ref is purchaseID system generated);

create type bookUdt as
(isbn              varchar(30),
 title             varchar(50),
 listPrice         decimal,
 authoredBy    ref (authorUdt) scope author array[5]
    references are checked on delete no action,
 boughtBy      ref (purchaseUdt) scope purchase array[1000]
    references are checked on delete set null,
 publishedBy   ref (publisherUdt) scope publisher
    references are checked on delete no action)
 instantiable not final ref is system generated
 method calcBookSales() returns decimal;

create table book
(primary key (isbn),
 ref is bookID system generated);
```

Figure 3: ORDB Schema of Publisher Application

The authorUdt type is defined as a subtype of personUdt, as indicated by the “under personUdt” clause. In addition to defining the structure of the type, authorUdt also defines behavior with the definition of the calcAuthorBookSales method. Since authorUdt is a subtype of personUdt, authorUdt will inherit the object identifier (personID) defined in personUdt. For consistency, the author table is also defined to be a subtable of the person object table. The typed table hierarchy therefore parallels the UDT hierarchy. In a similar manner, customerUdt is defined to be a subtype of personUdt and the customer typed table, based on customerUdt, is defined to be a subtable of the person table. UDTs and typed tables are also defined for the Book and Publisher classes from Figure 1, as well as the (reified implementation of the) Purchase association class.

Figure 3 also illustrates the use of object references to represent identity-based relationships between UDTs. Recall from the object data model in Figure 1 that a book is published by one publisher; a publisher publishes many books. In an ORDB, this relationship is established through the use of reference types. In the bookUdt, the publishedBy attribute has the type ref(publisherUdt), indicating that the value of publishedBy is a reference to the object identifier (publisherID) of a publisher. In the inverse direction, the type of booksPublished in the publisherUdt is an array of ref(bookUdt), indicating that booksPublished is an array of object references to books. Each attribute definition includes a scope clause and a “references are checked” clause. Since a UDT can be used to define multiple tables, the scope clause defines the table of the object reference. The references clause specifies the same options for referential integrity of object references as originally defined for traditional relational tables.

To establish the fact that a book is published by a specific publisher, the object identifier of publisher is retrieved to create the relationship:

```
update book
set publishedBy = (select publisherID
                   from publisher
                   where publisherName = “Prentice Hall”)
where isbn = “0-13-042898-1”;
```

A similar update statement can be used to establish the relationship in the inverse direction by adding the book oid to the array of object references of the publisher.

References can be traversed to query information about relationships. For example, to return the name of the publisher of a specific book, the following query can be used:

```
select publishedBy.publisherName
from book
where isbn = “0-13-042898-1”;
```

The dot notation in the select clause performs an implicit join between the book table and the publisher table, returning the name of the publisher. The deref() function can also be used to retrieve the entire structured type associated with a reference value. For example, the following query will return the full instance of the publisherUdt type, rather than just the publisherName:

```
select deref(publishedBy)
from book
where isbn = “0-13-042898-1”;
```

In this case, the result of the query is a value of type publisherUdt, containing the publisher name and the array of references to books published by the publisher.

KEY APPLICATIONS
Computer-Aided Design, Geographic Information Systems, Computer-Aided Software Engineering, Embedded Systems, Real-time Control Systems

CROSS REFERENCES
Semantic Data Model, Conceptual Data Model, Extended Entity Relationship Model, UML, ORM, Object Query Language, Database Design, Relational Data Model

RECOMMENDED READING
[1] Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D. and Zdonik, S. (1990): The Object-Oriented Database System Manifesto. Proc. of the 1st Int. Conf. on Deductive and Object-Oriented Databases, Kyoto, Japan. Elsevier Science Publishers B.V (NorthHolland) , 1990.

[2] Cattell, R. G. G., Barry, D. K., Berler, M., Eastman, J., Jordan, D., Russell, C., Schadow, O., Stanienda, T., and Velez, F., editors. (2000): The Object Data Standard: ODMG 3.0. Morgan Kaufmann 2000.

[3] Chaudhri, A. and Zicari, R., editors. (2000): Succeeding with Object Databases: A Practical Look at Today's Implementations with Java and XML. J. Wiley 2000.

[4] Codd, E. F. (1970) A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, vol. 13, no. 6.

[5] Dietrich, S. W. and Urban, S. D. (2005): An Advanced Course in Database Systems: Beyond Relational Databases. Prentice Hall 2005.

[6] Koshafian, S and Copeland, G (1986) Object Identity. ACM SIGPLAN Notices, vol. 20, no. 11.

[7] Loomis, M. E. S. and Chaudhri, A., editors. (1997): Object Databases in Practice: Prentice Hall 1997.

[8] Melton, J. (2002): Advanced SQL:1999: Understanding Object-Relational and Other Advanced Features. Morgan Kaufmann 2002.

[9] Object Database Management Systems: The Resource Portal for Education and Research, http://odbms.org/

[10] Rowe, L and Stonebraker, M. (1987) The Postgres Data Model. Proceedings of the 13th International Conference on Very Large Data Bases, Brighton, England

[11] Rumbaugh, J., Jacobson, I., and Booch, G. (1991): The Unified Modeling Language Reference Manual. Reading, MA. Addison-Wesley.

[12] Stonebraker, M., Rowe, L., Lindsay, B., Gray, J., Carey, M., Brodie, M., Bernstein, P., and Beech, D. (1990) Third Generation Database System Manifesto, SIGMOD Record, vol. 19, no. 3.

[13] Stonebraker, M. (1995): Object-Relational DBMSs: The Next Great Wave. Morgan Kaufmann 1995.

[14] Stroustrup, B (1997) The C++ Programming Language (3rd Edition). Reading, MA. Addison-Wesley.

[15] Zdonik, S.B. and Maier, D. (1990): Readings in Object-Oriented Database Systems. Morgan Kaufmann 1990.