Introduction

Enterprise Objects Framework brings the benefits of object-oriented programming to database application development. You can use the Framework to build feature-rich, graphical database applications with reusable software components that tightly couple business information with the business processes managing that information.

One of the most significant problems developers face when using object-oriented programming languages with SQL databases is the difficulty of matching static, two-dimensional data structures with the extensive flexibility afforded by objects. The features of object-oriented programming—such as encapsulation and polymorphism—and their benefits—like fewer lines of code and greater code reusability—are often negated by the programming restrictions that come with accessing SQL databases within an object-oriented application.

Enterprise Objects Framework solves this problem by providing tools for defining an object model and mapping it to a data model. This allows you to create objects that encapsulate both data and the methods for operating on that data, while taking advantage of the data access services provided by the Framework that make it possible for these objects to persist in a relational database.

The flexible, three-tier architecture provided by the Framework allows you to build robust, scalable, client/server applications. Objects at each of the three tiers (user interface, enterprise objects, and data store) can be deployed to take advantage of network resources. For example, data might be stored in a relational database running on a fault-tolerant database server with gigabytes of disk storage, while enterprise objects run on high-end compute servers. Partitioning the application to make best use of available resources allows complex applications to achieve maximum performance.

The components of Enterprise Objects Framework fully embrace the three-tier architecture, which means that portions of the Framework can be used selectively to meet specific application requirements. For example, the components that provide users with the ability to interactively manipulate enterprise objects can be used independently to provide applications with an undo capability. You can use a custom data store (such as a flat-file system) in place of a relational database to store data for enterprise objects. Or you can make use of the database adaptors separate from the rest of the Framework components to provide direct access to relational databases for your applications.

Enterprise Objects Framework offers these additional benefits:

Flexibility. An enterprise object isn't constrained by the physical location of data. Its mapping can extend across tables, and its data isn't confined to the object's mapping to a physical database. Further, the mapping of an enterprise object to the database can be dynamically controlled at run time.

Modularity. Depending on the needs of your application, you can create simple applications that require little or no code, program selected components while accepting the default behavior of other components, or use selected components independent of the rest of the Framework.

Extensibility. Enterprise Objects Framework's classes are public and extensible. For example, you can provide your own data source, or add support for a new user interface object.

Enterprise Objects Framework Documentation

The Enterprise Objects Framework documentation set includes the *Enterprise Objects Framework Reference* and the *Enterprise Objects Framework Developer's Guide*.

Entity-Relationship Modeling

Enterprise Objects Framework relies heavily on the concepts embodied in traditional Entity-Relationship modeling. Specifically, Entity-Relationship modeling terminology is used by the Enterprise Objects Framework classes to describe the mapping between stored data and enterprise objects.

For a discussion of these concepts and of relational databases, see the Appendix, "Entity-Relationship Modeling."