

# Glossary

## ***access layer***

The portion of the Enterprise Objects Framework that creates enterprise objects from a relational database and provides the classes and protocols that allow your application to interact with a database server.

## ***adaptor***

An object that connects an Enterprise Objects Framework application to a particular database server. The adaptor level classes provide a server-independent interface for working with relational database systems, while server-specific subclasses encapsulate the behavior of particular database servers.

## ***association***

Instances of the EOAssociation class that tie a single user interface object to a value corresponding to a named property (key) in an enterprise object or objects. Associations communicate with controllers to ensure that the values displayed in the user interface remain synchronized with the objects' values. Associations can also be used to link controllers. See *interface layer* and *controller*.

## ***attribute***

In Entity-Relationship modeling, an identifiable characteristic of an entity. For example, `lastName` could be an attribute of an `Employee` entity. An attribute typically corresponds to a column in a database table. See *flattened attribute*, *derived attribute*, *simple attribute*, *entity*, and *relationship*.

## ***channel***

An object used to perform select, fetch, insert, delete, and update operations on database rows or records. A channel corresponds to the cursor used by some relational databases. Operations performed by a channel take place within the scope of transactions controlled or tracked by a context. A context may work with several channels, but a channel is associated with only one context. See *context*.

## ***column***

In a relational database, the dimension of a table that holds values for a particular attribute. For example, a table that contains employee records might have a column titled "LastName" that contains the values for each employee's last name. See *attribute*.

## ***compound primary key***

In a database table, the group of columns whose values, taken in combination, are guaranteed to uniquely identify each row. See *primary key*.

## ***compound relationship***

A relationship that is based on more than one pair of join attributes. See *relationship* and *join*.

## ***connection dictionary***

The information needed to connect to a particular database server. Models include a connection dictionary. See *model*.

## ***context***

An object that manages a single transaction scope on the database server. A context can have one or more channels. See *channel* and *transaction*.

***controller***

In most NEXTSTEP applications, a custom class that manages part of an application, typically by mediating between the application's data and the user interface. The Enterprise Objects Framework provides an EOController class that takes the place of this custom class. EOController objects work with EOAssociation objects to coordinate the values displayed in the user interface with the values in enterprise objects. See *model-view-controller* and *association*.

***data dictionary***

In relational databases, the system tables that describe the organization of data in a particular database.

***data source***

An object conforming to the EODataSources protocol that provides a controller in the interface layer with enterprise objects. A data source presents the interface layer with a standard interface for accessing data, regardless of how the data is stored. All data sources have the ability to fetch, insert, update, and delete enterprise objects.

***database server***

A data storage and retrieval system. Database servers typically run on a dedicated computer and are accessed by client applications over a network.

***derived attribute***

An attribute that doesn't map directly to a single column in the root table of the entity. For example, a derived attribute can be based on another attribute that's modified in some way, such as an attribute **bonus** that's the result of a calculation performed on a **salary** attribute. See *root table*, *simple attribute*, *flattened attribute*, and *attribute*.

***destination entity***

In describing a relationship between two entities, the entity to which the relationship points. See *relationship*.

***enterprise object***

An Objective C object that conforms to the key-value coding protocol, whose properties (data) can map to stored data. An enterprise object brings together stored data with the methods for operating on that data. See *key-value coding* and *property*.

***Enterprise Objects Framework***

An object-oriented framework and associated tools that help you create applications that interact with database servers.

***entity***

In Entity-Relationship modeling, a distinguishable object about which data is kept. For example, you could have an **Employee** entity with attributes such as **lastName**, **firstName**, **address**, and so on. An entity typically corresponds to a table in a relational database; an entity's attributes in turn correspond to a table's columns. See *attribute* and *table*.

***Entity-Relationship modeling***

A discipline for examining and representing the components and interrelationships in a database system. Also known as E-R modeling, this discipline factors a database system into entities, attributes, and relationships.

***EOModeler***

One of the tools associated with the Enterprise Objects Framework. You use the EOModeler application to create models that map an object model to the database schema.

***EOPalette***

In Interface Builder, the collection of objects displayed in the Palettes window that you use in creating Enterprise Objects Framework applications.

***fault***

An object that's created for the destination of a relationship whenever an object that includes the relationship is fetched from the database. A fault object is a stand-in for the enterprise object (or in the case of a to-many relationship, array of objects) it represents. Fault objects transform themselves into the actual enterprise objects and fetch their data the first time they're accessed. See *relationship*.

***fetch***

In Enterprise Objects Framework applications, to retrieve data from the database server into the client application, usually into enterprise objects.

***flattened attribute***

A special kind of derived attribute that you add from one entity to another by traversing a relationship. For example, employees work for departments; you can add an attribute (such as **departmentName**) from the **Department** entity to the **Employee** entity as a flattened attribute. A flattened attribute is normally implemented by joining the tables corresponding to the source and destination entities whenever the attribute's data is fetched. See *relationship*, *attribute*, *derived attribute*, and *simple attribute*.

***flattened relationship***

A way of giving a source entity access to the relationships of a destination entity. For example, facilities have departments, which in turn have employees. You can add the relationship **toEmployee** that's between the **Department** and **Employee** entities to the **Facility** entity as a flattened relationship. See *relationship*.

***foreign key***

An attribute in an entity that gives it access to rows in another entity. This attribute must be the primary key of the related entity. For example, an **Employee** entity could contain the foreign key **deptID**, which matches the primary key in the entity **Department**. You could then use **deptID** as the source attribute in **Employee** and as the destination attribute in **Department** to form a relationship between the entities. See *key*, *primary key*, and *relationship*.

***full outer join***

A join operation in which all source records from both tables are included in the result—even those that don't satisfy the join condition. This type of join is not supported by most database servers. See *join* and *join condition*.

***generic record***

An instance of the **EOGenericRecord** default enterprise object class. A generic record has properties that map to stored data, but unlike a custom enterprise object, it adds no behavior to that data. Like custom enterprise objects, generic records conform to the key-value coding protocol; see *key-value coding*.

***inner join***

A join operation in which only the source records that satisfy the join condition appear in the result. See *join* and *join condition*.

***interface layer***

The portion of the Enterprise Objects Framework that provides a standard mechanism for displaying data from enterprise objects in the user interface, and that coordinates the flow of data between the user interface and enterprise objects. See *controller*, *association*, and *model-view-controller*.

***join***

An operation that provides access to data from two tables at the same time, based on the values contained in related columns. The results of a join are determined by the type of join in combination with the join operator. The types of joins supported by the Enterprise Objects Framework are inner join, full outer join, left outer join, and right outer join; note that not all join types are necessarily supported by all databases. See *inner join*, *full outer join*, *left outer join*, *right outer join*, *join operator*, *join semantic*, and *join condition*.

### ***join condition***

The restriction a join operator applies to a particular type of join to specify the criteria by which records are included in the results of the join. See *join* and *join operator*.

### ***join operator***

An operator used in a join that specifies how the destination attribute relates to the source attribute. Join operators are less than, greater than, equal to, less than or equal to, greater than or equal to, and not equal to. See *join* and *join condition*.

### ***join semantic***

The type of a join. Possible join types are inner join, full outer join, left outer join, and right outer join; note that not all join types are necessarily supported by all databases. See *join*, *join operator*, and *join condition*.

### ***key***

Depending on the context, the term "key" can have different meanings in the Enterprise Objects Framework. For NSDictionary objects and objects that conform to the key-value coding informal protocol (such as enterprise objects), "key" refers to the identifier used by other parts of the Framework to access data as key-value pairs. For this reason, enterprise object class properties are sometimes referred to as "class keys." The term "key" is also used to refer to primary keys. See *property*, *primary key*, and *key-value coding*.

### ***key-value coding (EOKeyValueCoding informal protocol)***

The mechanism that allows the properties in enterprise objects to be accessed by name (that is, as key-value pairs) by other parts of the Framework.

### ***left outer join***

A join operation in which source records for which no destination record can be found are included, but not the reverse. See *join* and *join condition*.

### ***many-to-many relationship***

A relationship in which each record in the source entity may correspond to more than one record in the destination entity, and each record in the destination may correspond to more than one record in the source. For example, an employee can work on many projects, and a project can be staffed by many employees. See *relationship*.

### ***model***

An EOModel object that defines, in Entity-Relationship terms, the mapping between enterprise object classes and the database schema. This definition is typically stored in a file created with the EOModeler application. A model also includes the information needed to connect to a particular database server; see *connection dictionary*.

### ***modeling objects***

A general term that refers to objects of the classes used to build models: EOModel, EOEntity, EOAttribute, EORelationship, and EOJoin.

### ***model-view-controller***

An object-oriented paradigm that originated in Smalltalk, in which an application's data (model) is kept separate from the user interface (view) by an intermediate layer (controller). This paradigm is made explicit in the Enterprise Objects Framework's architecture. An application's enterprise objects are the model, while the classes of the interface layer compose the controller system for the application. See *interface layer*, *controller*, and *association*.

### ***primary key***

In a database table, the column whose values are guaranteed to uniquely identify each row. In the Enterprise Objects Framework, the term "primary key" is also sometimes used to refer to the *value* of a primary key for a specific enterprise object. For example, an Employee enterprise object might have the property **empID** with the value "115". Depending on the context, either the property or its

value might be referred to as the object's primary key. See *key*, *compound primary key*, *foreign key*, and *relationship*.

### ***property***

Depending on the context, the term "property" can have different meanings in the Enterprise Objects Framework. An entity's attributes and relationships are called properties. The instance variables and methods in an enterprise object class that are accessed by key-value coding methods are also called properties. An enterprise object's "class properties" typically include instance variables that map to the properties of a corresponding entity. Enterprise object properties are sometimes referred to as "keys" to indicate that the object's data is accessed as key-value pairs. See *key-value coding* and *key*.

### ***qualifier***

An object that holds information used to restrict a select operation to enterprise objects or rows whose attributes meet certain criteria.

### ***record***

The set of values that describes a single instance of an entity; in a relational database, a record is equivalent to a row.

### ***reflexive relationship***

A relationship in which the source and destination entities are the same. For example, the **Employee** entity could have a **toManager** relationship to itself to provide access to the manager of a given employee.

### ***relational database***

A database designed according to the relational model, which uses the discipline of Entity-Relationship modeling and the data design standards called normal forms.

### ***relationship***

A link between two entities that's based on attributes of the entities. For example, the **Department** and **Employee** entities could have a relationship based on the **deptID** attribute as a foreign key in **Employee**, and as the primary key in **Department** (note that though the join attribute **deptID** is the same for the source and destination entities in this example, it doesn't have to be). This relationship would make it possible to find the employees for a given department. See *to-one*, *to-many*, *many-to-many*, *primary key*, and *foreign key*.

### ***right outer join***

A join operation in which destination records for which no source record can be found are included, but not the reverse. See *join* and *join condition*.

### ***root table***

The table associated with an entity from which all of the entity's relationships originate. An entity's root table contains all of the entity's modifiable attributes; though an entity may hold references to attributes in other tables, only the attributes in its root table can be modified. See *relationship*, *attribute*, *flattened attribute*, *derived attribute*, and *entity*.

### ***row***

In a relational database, the dimension of a table that groups attributes into records.

### ***simple attribute***

An attribute that corresponds directly to a single column in the database. See *attribute*, *derived attribute* and *flattened attribute*.

### ***snapshot***

An NSDictionary object that records an enterprise object's state when it's fetched from the database. Snapshots are used in updates to ensure that the data in the corresponding database row was not changed since the object was fetched.

### ***source entity***

In describing a relationship between two entities, the entity from which the relationship originates. See *relationship*.

***table***

A two-dimensional set of values corresponding to an entity. The columns of a table represent characteristics of the entity and the rows represent instances of the entity.

***to-many relationship***

A relationship in which each source record has zero to many corresponding destination records. For example, a department has many employees.

***to-one relationship***

A relationship in which each source record has exactly one corresponding destination record. For example, each employee has one job title.

***transaction***

A logical unit of work that can contain one or more database operations (operations are insert, update, and delete). A transaction transforms the database from one consistent state to another without externally exposing any inconsistencies at intermediate points. A transaction is completed by either committing the changes, or, if any portion of the transaction fails, by rolling the database back to its previous consistent state.

***uniquing***

A behavior of the Enterprise Objects Framework used to uniquely identify enterprise objects. Uniquing ensures that if an object already exists in memory, another instance of it isn't created when a row with the same primary key is fetched from the database. So, for example, if two employee objects have the same manager, a single instance of the manager object resides in memory, and both employee objects refer to it. See *primary key*.

***update***

In Enterprise Objects Framework applications, to modify records in the database server based on changes to the corresponding records in the client application.