1. Overview1. Overview

This standard describes the IDEF1X modeling language (semantics and syntax) and associated rules and techniques, for developing a logical model of data. IDEF1X is used to produce information models which represent the structure and semantics of information within an enterprise.

1.1 Scope1.1 Scope

IDEF1X is used to produce a graphical information model which represents the structure and semantics of information within an environment or system. Use of this standard permits the construction of semantic data models which may serve to support the management of data as a resource, the integration of information systems, and the building of computer databases.

In addition to the standard specification, this document provides two informative annexes. Annex A provides an example of a process of developing an IDEF1X model introduced in the original ICAM program. Annex B introduces a fomalization to the IDEF1X language written by R. G. Brown of the Database Design Group.

1.2 Purpose1.2 Purpose

This information modeling technique is used to model data in a standard, consistent, predictable manner in order to manage it as a resource. The primary objectives of this standard are:

a) To provide a means for completely understanding and analyzing an organization's data resources;

b) To provide a common means of representing and communicating the complexity of data;

c) To provide a method for presenting an overall view of the data required to run an enterprise;

d) To provide a means for defining an application-independent view of data which can be validated by users and transformed into a physical database design;

e) To provide a method for deriving an integrated data definition from existing data resources.

2. Definitions2. Definitions

2.1 Alias: A nonstandard name for an entity or domain (attribute).

2.2 Assertion: A statement that specifies a condition that must be true.

2.3 Attribute: A property or characteristic that is common to some or all of the instances of an entity. An attribute represents the use of a domain in the context of an entity.

2.4 Attribute, Inherited: An attribute that is a characteristic of a category entity by virtue of being an attribute in its generic entity or a generic ancestor entity.

2.5 Attribute, Migrated: A foreign key attribute of a child entity.

2.6 Attribute, Non-key: An attribute that is not the primary or a part of a composite primary key of an entity. A non-key attribute may be a foreign key or alternate key attribute.

2.7 Attribute, Optional: A non-key attribute of an entity that may be null in any instance of the entity.

2.8 Attribute, Owned: An attribute of an entity that has not migrated into the entity.

2.9 Attribute Value: A value given to an attribute in an entity instance.

2.10 Category Cluster: A set of one or more mutually exclusive categorization relationships for the same generic entity.

2.11 Category Discriminator: An attribute in the generic entity (or a generic ancestor entity) of a category cluster. The values of the discriminator indicate which category entity in the category cluster contains a specific instance of the generic entity. All instances of the generic entity with the same discriminator value are instances of the same category entity. The inverse is also true.

2.12 Conceptual Schema: See Schema

2.13 Constraint: A rule that specifies a valid condition of data.

2.14 Constraint, Cardinality: A limit on the number of entity instances that can be associated with each other in a relationship.

2.15 Constraint, Existence: A condition where an instance of one entity cannot exist unless an instance of another related entity also exists.

2.16 Database: A collection of interrelated data, often with controlled redundancy, organized according to a schema to serve one or more applications.

2.17 Data Model: A graphical and textual representation of analysis that identifies the data needed by an organization to achieve its mission, functions, goals, objectives, and strategies and to manage and rate the organization. A data model identifies the entities, domains(attributes), and relationships (or associations) with other data, and provides the conceptual view of the data and the relationships among data.

2.18 Data Type: A categorization of an abstract set of possible values, characteristics, and set of operations for an attribute. Integers, real numbers, character strings, and enumerations are examples of data types.

2.19 Domain: A named set of data values (fixed, or possibly infinite in number) all of the same data type, upon which the actual value for an attribute instance is drawn. Every attribute must be defined on exactly one underlying domain. Multiple attributes may be based on the same underlying domain.

2.20 Enterprise: An organization that exists to perform a specific mission and achieve associated goals and objectives.

2.21 Entity: The representation of a set of real or abstract things (people, objects, places, events, ideas, combination of things, etc.) that are recognized as the same type because they share the same characteristics and can participate in the same relationships.

2.22 Entity, Category: An entity whose instances represent a sub-type or sub-classification of another entity (generic entity). Also known as sub-type or sub-class.

2.23 Entity, Child: The entity in a specific connection relationship whose instances can be related to zero or one instance of the other entity (parent entity).

2.24 Entity, Generic: An entity whose instances are classified into one or more sub-types or sub-classifications (category entity). Also known as super-type or super-class.

2.25 Entity Instance: One of a set of real or abstract things represented by an entity. The instance of an entity can be specifically identified by the value of the attribute(s) participating in its primary key.

2.26 Entity, Parent: An entity in a specific connection relationship whose instances can be related to a number of instances of another entity (child entity).

2.27 Existence Dependency: A constraint between two related entities indicating that no instance of one (child entity) can exist without being related to an instance of the other (parent entity). The following relationship types represent existence dependencies: identifying relationships, categorization relationships and mandatory non-identifying relationships.

2.28 External Schema: See Schema

2.29 Functional Dependency: A special kind of integrity constraint that applies within the confines of a single entity "R", where each "X" value of "R" has associated with it at most one "Y" value of "R" (at any one time). Attributes "X" and "Y" may be composite.

2.30 Glossary: A set of definitions of entities and domains (attributes).

2.31 IDEF1X Diagram: See View Diagram.

2.32 IDEF1X Model: A set of one or more IDEF1X views, often represented as view diagrams which depict the underlying semantics of the views, along with definitions of the entities and attributes used in the views. See Data Model.

2.33 Identifier Dependency: A constraint between two related entities that requires the primary key in one (child entity) to contain the entire primary key of the other (parent entity). The following relationship types represent identifier dependencies: Identifying relationships, categorization relationships.

2.34 Key, Candidate: An attribute, or combination of attributes, of an entity whose values uniquely identify each entity instance.

2.35 Key, Alternate: Any candidate key of an entity other than the primary key.

2.36 Key, Composite: A key comprised of two or more attributes.

2.37 Key, Compound: Same as Key, Composite.

2.38 Key, Foreign: An attribute, or combination of attributes of a child or category entity instance whose values match those in the primary key of a related parent or generic entity instance. A foreign key results from the migration of the parent or generic entities primary key through a specific connection or categorization relationship.

2.39 Key, Migrated: Same as Foreign Key.

2.40 Key Migration: The modeling process of placing the primary key of a parent or generic entity in its child or category entity as a foreign key.

2.41 Key, Primary: The candidate key selected as the unique identifier of an entity.

2.42 Key, Split: A foreign key containing two or more attributes, where at least one of the attributes is a part of the entities primary key and at least one of the attributes is not a part of the primary key.

2.43 Normal Form: The condition of an entity relative to satisfaction of a set of normalization theory constraints on its attribution. A specific normal form is achieved by successive reduction of an entity from its existing condition to some more desirable form. The procedure is reversible.

a) First Normal Form (1NF) - An entity is in 1NF if and only if all underlying simple domains contain atomic values only.

b) Second Normal Form (2NF) - An entity is in 2NF if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key.

c) Third Normal Form (3NF) - An entity is in 3NF if and only if it is in 2NF and every attribute that is not a part of the primary key is non-transitively dependent (mutually independent) on the primary key. Two or more attributes are mutually independent if none of them is functionally dependent on any combination of the others.

2.44 Normalization: the process of refining and regrouping attributes in entities according to the normal forms.

2.45 Null: A condition where a value of an attribute is not applicable or not known for an entity instance.

2.46 Relationship: An association between two entities or between instances of the same entity.

2.47 Relationship Cardinality: The number of entity instances that can be associated with each other in a relationship. See Constraint, Cardinality.

2.48 Relationship, Categorization (Category): A relationship in which instances of both entities represent the same real or abstract thing. One entity (generic entity) represents the complete set of things the other (category entity) represents a sub-type or sub-classification of those things. The category entity may have one or more characteristics, or a relationship with instances of another entity not shared by all generic entity instances. Each instance of the category entity is simultaneously an instance of the generic entity.

2.49 Relationship, Connection: Same as Relationship, Specific Connection.

2.50 Relationship, Identifying: A specific connection relationship in which every attribute in the primary key of the parent entity is contained in the primary key of the child entity.

2.51 Relationship, Mandatory Non-identifying: A non-identifying relationship in which an instance of the child entity must be related to an instance of the parent entity.

2.52 Relationship Name: A verb or verb phrase which reflects the meaning of the relationship expressed between the two entities shown on the diagram on which the name appears.

2.53 Relationship, Non-specific: An relationship in which an instance of either entity can be related to a number of instances of the other.

2.54 Relationship, Non-identifying: A specific connection relationship in which some or all of the attributes contained in the primary key of the parent entity do not participate in the primary key of the child entity.

2.55 Relationship, Optional Non-identifying: A non-identifying relationship in which an instance of the child entity can exist without being related to an instance of the parent entity.

2.56 Relationship, Parent-Child: Same as Relationship, Specific Connection.

2.57 Relationship, Specific Connection: A relationship where a number of instances of one entity (child entity) can be related to zero or one instance of the other entity (parent entity). In a specific connection relationship the primary key of the parent entity is contributed as a foreign key to the child entity.

2.58 Role Name: A name assigned to a foreign key attribute to represent the use of the foreign key in the entity.

2.59 Schema: A definition of data structure:

a) Conceptual Schema: A schema of the ANSI/SPARC Three Schema Architecture, in which the structure of data is represented in a form independent of any physical storage or external presentation format.

b) External Schema: A schema of the ANSI/SPARC Three Schema Architecture, in which views of information are represented in a form convenient for the users of information; a description of the structure of data as seen by the user of a system.

c) Internal Schema: A schema of the ANSI/SPARC Three Schema Architecture, in which views of information are represented in a form specific to the data base management system used to store the information: a description of the physical structure of data.

2.60 Semantics: The meaning of the syntatic components of a language.

2.61 Synonym: A word, expression, or symbol accepted as a figurative or symbolic substitute for another word or expression; that is, an alternative name for the same thing. (See Alias)

2.62 Syntax: Structural components or features of a language and rules that define relationships among them.

2.63 Verb Phrase: A phrase used to name a relationship, which consists of a verb and words which comprise the object of the phrase.

2.64 View: A collection of entities and assigned attributes (domains) assembled for some purpose.

2.65 View Diagram: A graphic representation of the underlying semantics of a view.

3. IDEF1X Syntax and Semantics3. IDEF1X Syntax and Semantics

This section will discuss the levels of IDEF1X models, their content, the rules that govern them, and the various forms in which a model may be presented. It will also discuss the semantics (or meaning) of each component of an IDEF1X diagram, the graphical syntax for representing the component, and the rules governing its use. Although the components are highly interrelated, each one is discussed separately without regard for the actual sequence of construction. An IDEF1X model is comprised of one or more views (often presented in view diagrams representing the underlying semantics of the views), and definitions of the entities and domains (attributes) used in the views. These are described in this section. Annex A discusses a procedure for building an IDEF1X model which will conform to the defined syntax and semantics.

Each IDEF1X model must be accompanied by a statement of purpose (describing why the model was produced), a statement of scope (describing the general area covered by the model), and a description of any conventions the authors have used during its construction. Author conventions must not violate any of the rules governing model syntax or semantics.

The components of an IDEF1X view are:

a) Entities
   1) Identifier-Independent Entities
   2) Identifier-Dependent Entities

b) Relationships
   1) Identifying Connection Relationships
   2) Non-Identifying Connection Relationships
   3) Categorization Relationships
   4) Non-Specific Relationships

c) Attributes/Keys
   1) Attributes
   2) Primary Keys
   3) Alternate Keys
   4) Foreign Keys

d) Notes

The section provides a general description of IDEF1X. Each modeling construct is described in terms of its general semantics, syntax, and rules.

3.1 Entities3.1 Entities

Entities represent the things of interest in an IDEF1X view. They are displayed in view diagrams, and defined in the glossary.

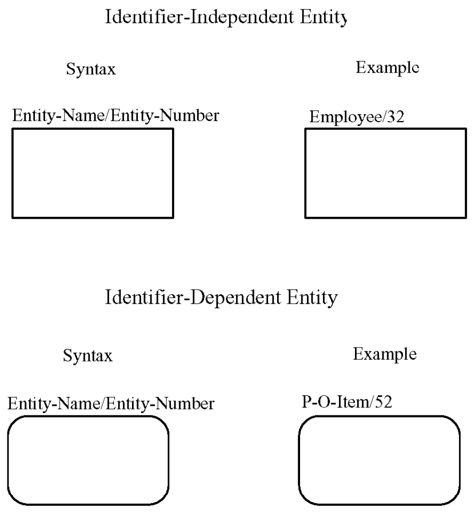
3.1.1 Entity Semantics3.1.1 Entity Semantics

An entity represents a set of real or abstract things (people, objects, places, events, ideas, combinations of things, etc.) which have common attributes or characteristics. An individual member of the set is referred to as an "entity instance." A real world object or thing may be represented by more than one entity within a view. For example, John Doe may be an instance of both the entity EMPLOYEE and BUYER. Furthermore, an entity instance may represent a combination of real world objects. For example, John and Mary could be an instance of the entity MARRIED-COUPLE.

An entity is "identifier-independent" or simply "independent" if each instance of the entity can be uniquely identified without determining its relationship to another entity. An entity is "identifier-dependent" or simply "dependent" if the unique identification of an instance of the entity depends upon its relationship to another entity.

3.1.2 Entity Syntax3.1.2 Entity Syntax

An entity is represented as a box as shown in Figure 1. If the entity is identifier-dependent, then the corners of the box are rounded. Each entity is assigned a label which is placed above the box. The label must contain a unique entity name. A positive integer number may also be assigned to an entity as part of the label. The number would be separated by a slash ("/").



Identifier-Independent Entity
Syntax
Example
Entity-Name/Entity-Number
Employee/32
Identifier-Dependent Entity
Syntax
Example
Entity-Name/Entity-Number
P-O-Item/52

Figure 1. Entity Syntax

The entity name is a noun phrase that describes the set of things the entity represents. The noun phrase is in singular form, not plural. Abbreviations and acronyms are permitted, however, the entity name must be meaningful and consistent throughout the model. A formal definition of the entity and a list of synonyms or aliases must be defined in the glossary. Although an entity may be drawn in any number of diagrams, it only appears once within a given diagram.

3.1.3 Entity Rules3.1.3 Entity Rules

a) Each entity must have a unique name and the same meaning must always apply to the same name. Furthermore, the same meaning cannot apply to different names unless the names are aliases.

b) In a key-based or fully-attributed view, an entity has one or more attributes which are either owned by the entity or migrated to the entity through a relationship. (See Foreign Keys in Section 3.9.)

c) In a key-based or fully-attributed view, an entity has one or more attributes whose values uniquely identify every instance of the entity. (See Primary and Alternate Keys in Section 3.8.)

d) An entity can have any number of relationships with other entities in the view.

e) If an entire foreign key is used for all or part of an entity's primary key, then the entity is identifier-dependent. Conversely, if only a portion of a foreign key or no foreign key attribute at all is used for an entity's primary key, then the entity is identifier-independent. (See Foreign Key Semantics in Section 3.9.1.)

f) In a view, an entity is labeled by either its entity name or one of its aliases. It may be labeled by different names (i.e., aliases) in different views in the same model. (See Glossary in Section 3.12.)

g.) No view can contain two distinctly named entities in which the names are synonymous. Two names are synonymous if either is directly or indirectly an alias for the other, or there is a third name for which both names are aliases (either directly or indirectly).

3.2 Domains3.2 Domains

A "Domain" represents a named and defined set of values that one or more attributes draw their values from. In IDEF1X domains are defined separately from entities and views in order to permit their reuse and standardization throughout the enterprise.

3.2.1 Domain Semantics3.2.1 Domain Semantics

A domain is considered a class for which there is a fixed, and possibly infinite, set of instances. For example, *State-Code* would be considered a domain, where the set of allowable values for the domain would satisfy the definition of a state-code (e.g. the unique identifier of a state) and might consist of the two-letter abbreviations of the states. Another example of a domain might be *Last-Name,* which has a near-infinite set of possible values which must be composed of alphabetic characters [A-Z, a-z].

Domains are considered immutable classes whose values do not change over time. In contrast, entities are time-varying classes, their instance data varies over time as the data is modified and maintained. As immutable classes, domain instances always exist in principle. Take, for example, the domain *Date*, each instance of date did or will exist, however all instances of date might not be used as instances in an entity containing a date domain.

Each domain instance has a unique value in some representation--that is, unique within that domain. The *State-Code* domain could use a number of representations: Full-Name [Alabama, Alaska, Arizona, ...], Abbreviations [AL, AK, AZ, ...] or State Number [1, 2, 3, ...]. Each instance representation must be unique within the domain. Using the first letter of the state as the representation would be invalid [A, A, A].

There are two basic types of domains, base domain and typed domain.

A base domain may be assigned a data type which may be one of the following: Character, Numeric or Boolean. Other data types such as dates, times, binary, etc. may also be used, but the IDEF1X standard includes the basic three as defaults.

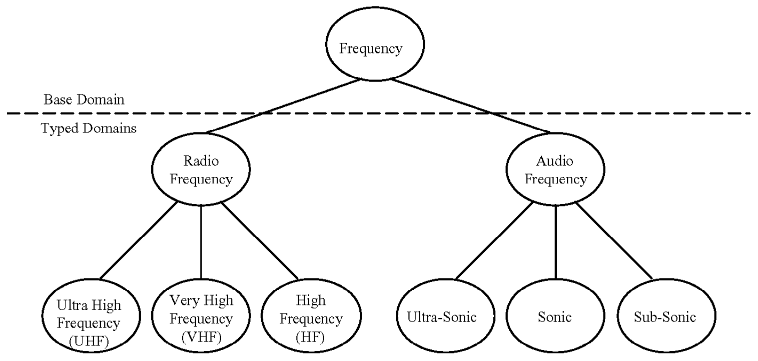
Base domains may also be assigned a domain rule. Domain rules are used to provide the acceptable values of a domain. The two most common domain rules are the value list, and the range rules.

The value list domain rule defines the set of all acceptable instance values for a domain. Attributes of a domain with a value list domain rule are only valid if their instance values are a part of the value list. A common use of this rule is to define lists of coded values such as *State-Code* or *Titles* [Mr, Mrs, ...].

The range domain rule defines the set of all acceptable instance values for a domain where the instance values are constrained by a lower and/or upper boundary. A good example of the range domain rule is *Azimuth* which must be between -360^{o} to $+360^{o}$.

Finally, for a domain, the domain rule may be left unspecified. In this case the domain is only constrained by those rules associated with its data type or those of its super-type if it has either. *Last-Name* is an example of a domain without a domain rule, it can take on any allowable character value.

An instance of a base domain is considered to exist if it is of the specified data type, if any, and satisfies its domain rule.



Typed domains are sub-types of base domains or other typed domains, which may further constrain the acceptable values for the domain. A typed domain exists, if it is of the data type, and satisfies the domain rules of its supertype domain. In this way, a hierarchy of domains can be defined, with increasingly tighter domain rules going down the hierarchy.

A domain hierarchy is a generalization hierarchy. Note that, unlike the category structure for entitles, there is no implication that domain sub-types are mutually exclusive.

In the example shown in Figure 2, the *Frequency* domain might have a data type of numeric with no domain rule. The *Audio-Frequency* domain would have a representation of Hertz (Hz) with a range domain rule added, which constrained the valid instance data to between 1 and 250,000 Hz. The *Sonic* domain would further constrain the range to that of human hearing or 20 to 20,000 Hz with an additional range domain rule. The Radio-Frequency domain would have a range rule of 1 to 300,000,000.0 Hz. Note that a domain instance must comply with all of the domain rules of its parent domain (if any) and with its own domain rules.

Figure 2. Example of a Domain Hierarchy

3.2.2 Domain Syntax3.2.2 Domain Syntax

There is no current concrete syntax for domains in IDEF1X. Domain information should be stored in the glossary as informal notation stating the data type for base domains, sub-typing relationships for typed domains, and the domain rules and representation for all domains.

3.2.3 Domain Rules3.2.3 Domain Rules

a) A domain must have a unique name and the same meaning must always apply to the same name. Furthermore, the same meaning cannot apply to different names unless the names are aliases.

b) A domain is either a base domain or a typed domain.

c) A base domain may have one of the following types: character, numeric, or boolean.

d) A domain may have a domain rule.

e) A domain rule is stated either as a range or a value list.

f) The range rule constrains valid instances with a lower and/or upper value.

g) The value list rule constrains valid instances to one member of a set of values.

h) A typed domain is a sub-type of a base domain or another typed domain.

i) A typed domain references the domain it is a sub-type of.

j) No domain can be directly or indirectly a sub-type of itself.

3.3 Views3.3 Views

An IDEF1X “view” is a collection of entities and assigned domains (attributes) assembled for some purpose. A view may cover the entire area being modeled, or a part of that area. An IDEF1X model is comprised of one or more views (often presented in view diagrams representing the underlying semantics of the views), and definitions of the entities and domains (attributes) used in the views.

3.3.1 View Semantics3.3.1 View Semantics

In IDEF1X, entities and domains are defined in a common glossary and mapped to one another in views. In this way an entity such as EMPLOYEE may appear in multiple views, in multiple models, and have a somewhat different set of attributes in each. In each view, it is required that the entity EMPLOYEE mean the same thing. The intent is that EMPLOYEE be the class of all employees. That is, individual things are classified as belonging to the class EMPLOYEE on the basis of some similarity. It is that sense of what it means to be an employee that is defined in the glossary. Similarly, the domain EMPLOYEE-NAME is defined once, and used as an attribute in appropriate views.

A view is given a name, and, optionally, additional descriptive information. Optional information may include the name of the author, dates created and last revised, level (ER, key-based, fully-attributed, etc.), completion or review status, and so on.

A textual description of the view may also be provided. This description may contain narrative statements about the relationships in the view, brief descriptions of entities and attributes, and discussions of rules or constraints that are specified.

3.3.2 View Syntax3.3.2 View Syntax

The constructs, i.e., entities, attributes (domains), relationships, and notes depicted in a view diagram must comply with all syntax and rules governing the individual constructs.

3.3.3 View Rules3.3.3 View Rules

a) Each view must have a unique name.

b) Author conventions adopted for a model must be consistent across all views included in the model.

c) Any view which contains syntax that is in conflict with any of the standards set forth in this document must be labeled “For Exposition Only” (FEO).

d) A model may contain views of different levels.

3.4 Attributes3.4 Attributes

A domain associated with an entity in a view is referred to as an “attribute” of the entity. Within an IDEF1X view, attributes are associated with specific entities.

3.4.1 Attribute Semantics3.4.1 Attribute Semantics

In an IDEF1X view, an "attribute" represents a type of characteristic or property associated with a set of real or abstract things (people, objects, places, events, ideas, combinations of things, etc.). An "attribute instance" is a specific characteristic of an individual member of the set. An attribute instance is defined by both the type of characteristic and its value, referred to as an "attribute value." An instance of an entity, then, will usually have a single specific value for each associated attribute. For example, EMPLOYEE-NAME and BIRTH-DATE may be attributes associated with the entity EMPLOYEE. An instance of the entity EMPLOYEE could have the attribute values of "Jenny Lynne" and "February 27, 1953." In another situation, an employee's birth date may be unknown when the entity instance is created, and the attribute may have no value for some period of time. In another case, an attribute may be applicable for one instance, but not for another. For example, a WINDOW entity might have the attribute COLOR. In one instance, COLOR may have the value "grey". In another, there may be no value (the attribute may be null) meaning that the window has no color (the window is clear, therefore the attribute is not applicable to this instance).

An entity must have an attribute or combination of attributes whose values uniquely identify every instance of the entity. These attributes form the "primary-key" of the entity. (See Section 3.8.) For example, the attribute EMPLOYEE-NUMBER might serve as the primary key for the entity EMPLOYEE, while the attributes EMPLOYEE-NAME and BIRTH-DATE would be non-key attributes.

In a key-based or fully-attributed view, every attribute is owned by only one entity. The attribute MONTHLY-SALARY, for example, might apply to some instances of the entity EMPLOYEE but not all. Therefore, a separate but related category entity called SALARIED-EMPLOYEE might be identified in order to establish ownership for the attribute MONTHLY-SALARY. Since an actual employee who was salaried would represent an instance of both the EMPLOYEE and SALARIED-EMPLOYEE entities, attributes common to all employees, such as EMPLOYEE-NAME and BIRTH-DATE, are owned attributes of the EMPLOYEE entity, not the SALARIED-EMPLOYEE entity. Such attributes are said to be "inherited" by the category entity, but are not included in its list of attributes: they only appear in the list of attributes of the generic entity.

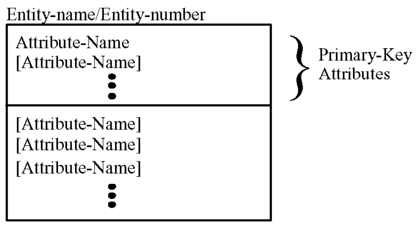
In addition to an attribute being "owned" by an entity, an attribute may be present in an entity due to its "migration" through a specific connection relationship, or through a categorization relationship. (See Section 3.9.) For example, if every employee is assigned to a department, then the attribute DEPARTMENT-NUMBER could be an attribute of EMPLOYEE which has migrated through the relationship to the entity EMPLOYEE from the entity DEPARTMENT. The entity DEPARTMENT would be the owner of the attribute DEPARTMENT-NUMBER. Only primary key attributes may be migrated through a relationship. The attribute DEPARTMENT-NAME, for example, would not be a migrated attribute of EMPLOYEE if it was not part of the primary key for the entity DEPARTMENT.

3.4.2 Attribute Syntax3.4.2 Attribute Syntax

Each attribute is identified by the unique name of its underlying domain. The name is expressed as a noun phrase that describes the characteristic represented by the attribute. The noun phrase is in singular form, not plural. Abbreviations and acronyms are permitted, however, the attribute name must be meaningful and consistent throughout the model. A formal definition and a list of synonyms or aliases must be defined in the glossary.

Attributes are shown by listing their names, inside the associated entity box. Attributes which define the primary key are placed at the top of the list and separated from the other attributes by a horizontal line. See Figure 3.

Attribute And Primary Key Syntax



Example

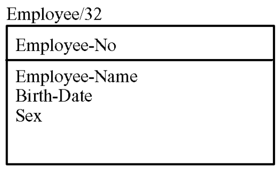


Figure 3. Attribute and Primary Key Syntax

3.4.3 Attribute Rules3.4.3 Attribute Rules

a) An attribute is a mapping from an entity in a view to a domain. It must have a unique name and the same meaning must always apply to the same name. Furthermore, the same meaning cannot apply to different names unless the names are aliases.

b) An entity can own any number of attributes. In a key-based or fully-attributed view, every attribute is owned by exactly one entity (referred to as the Single-Owner Rule).

c) An entity can have any number of migrated attributes. However, a migrated attribute must be part of the primary key of a related parent entity or generic entity.

d) Every instance of an entity must have a value for every attribute that is part of its primary key.

e) No instance of an entity can have more than one value for an attribute associated with the entity (referred to as the No-Repeat Rule).

f) Attributes that are not part of a primary key are allowed to be null (meaning not applicable or not known) provided that they are clearly marked by the symbol "(O)" (an upper case O, for optional) following the attribute name.

g) In a view, an attribute is labeled by either its attribute name or one of its aliases. If it is an owned attribute in one entity, and a migrated attribute in another, it either has the same name in both or has a role name or an alias for a role name as the migrated attribute. An attribute may be labeled by different names (i.e., aliases) in different views within the same model.

h) No view can contain two distinctly named attributes in which the names are synonymous. Two names are synonymous if either is directly or indirectly an alias for the other, or there is a third name for which both names are aliases (either directly or indirectly).

3.5 Connection Relationships3.5 Connection Relationships

In an IDEF1X view, connection relationships are used to represent associations between entities.

3.5.1 Specific Connection Relationship Semantics3.5.1 Specific Connection Relationship Semantics

A “specific connection relationship”or simply “connection relationship” (also referred to as a “parent-child relationship”) is an association or connection between entities in which each instance of one entity, referred to as the parent entity, is associated with zero, one, or more instances of the second entity, referred to as the child entity, and each instance of the child entity is associated with zero or one instance of the parent entity. For example, a specific connection relationship would exist between the entities BUYER and PURCHASE-ORDER, if a buyer issues zero, one, or more purchase orders and each purchase order must be issued by a single buyer. An IDEF1X view diagram depicts the type or set of relationship between two entities. A specific instance of the relationship associates specific instances of the entities. For example, “buyer John Doe issued Purchase Order number 123” is an instance of a relationship.

The connection relationship may be further defined by specifying the cardinality of the relationship. That is, the specification of how many child entity instances may exist for each parent instance. Within IDEF1X, the following relationship cardinalities can be expressed from the perspective of the parent entity:

a) Each parent entity instance may have zero or more associated child entity instances.

b) Each parent entity instance must have at least one associated child entity instance.

c) Each parent entity instance can have zero or one associated child instance.

d) Each parent entity instance is associated with some exact number of child entity instances.

e) Each parent entity instance is associated with a specified range of child entity instances.

Cardinality can also be described from the perspective of the child entity and will be further enumerated in the sections that follow.

3.5.1.1 Identifying Relationship Semantics3.5.1.1 Identifying Relationship Semantics

If an instance of the child entity is identified by its association with the parent entity, then the relationship is referred to as an “identifying relationship”, and each instance of the child entity must be associated with exactly one instance of the parent entity. For example, if one or more tasks are associated with each project and tasks are only uniquely identified within a project, then an identifying relationship would exist between the entities PROJECT and TASK. That is, the associated project must be known in order to uniquely identify one task from all other tasks. (See Foreign Keys in Section 3.9.) The child in an identifying relationship is always existence-dependent on the parent, i.e., an instance of the child entity can exist only if it is related to an instance of the parent entity.

3.5.1.2 Non-Identifying Relationship Semantics3.5.1.2 Non-Identifying Relationship Semantics

If every instance of the child entity can be uniquely identified without knowing the associated instance of the parent entity, then the relationship is referred to as a “non-identifying relationship.” For example, although an existence-dependency relationship may exist between the entities BUYER and PURCHASE-ORDER, purchase orders may be uniquely identified by a purchase order number without identifying the associated
buyer.

3.5.2 Specific Connection Relationship Syntax

A specific connection relationship is depicted as a line drawn between the parent entity and the child entity with a dot at the child end of the line. The default child cardinality is zero, one or many. A “P” (for positive) is placed beside the dot to indicate a cardinality of one or more. A “Z” is placed beside the dot to indicate a cardinality of zero or one. If the cardinality is an exact number, a positive integer number is placed beside the dot. If the cardinality is a range, the range is placed beside the dot. See Figure 4. Other cardinalities (for example, more than 3, exactly 7 or 9), are recorded as notes placed beside the dot.

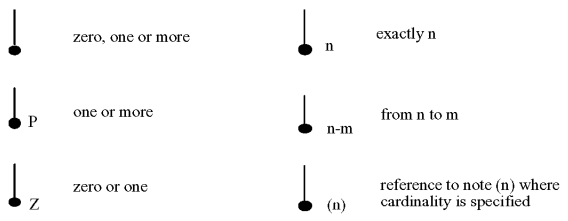
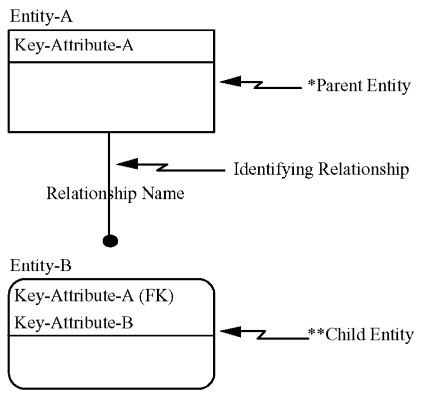


Figure 4. Relationship Cardinality Syntax

3.5.2.1 Identifying Relationship Syntax3.5.2.1 Identifying Relationship Syntax

A solid line depicts an identifying relationship between the parent and child entities. See Figure 5. If an identifying relationship exists, the child entity is always an identifier-dependent entity, represented by a rounded corner box, and the primary key attributes of the parent entity are also migrated primary key attributes of the child entity. (See Foreign Keys in Section 3.9.)



* The Parent Entity in an Identifying Relationship may be an Identifier-Independent Entity (as shown) or an Identifier-Dependent Entity depending upon other relationships.

** The Child Entity in an Identifying Relationship is always an Identifier-Dependent Entity.

Figure 5. Identifying Relationship Syntax

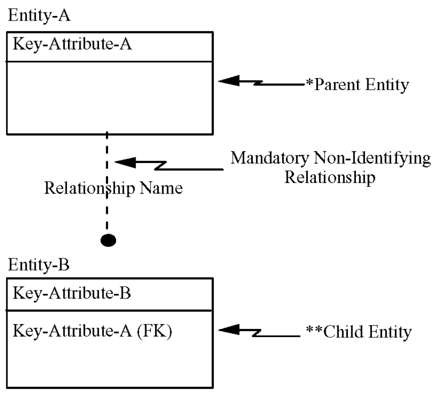
The parent entity in an identifying relationship will be identifier-independent unless the parent entity is also the child entity in some other identifying relationship, in which case both the parent and child entity would be identifier-dependent. An entity may have one or more relationships with other entities. However, if the entity is a child entity in any identifying relationship, it is always shown as an identifier-dependent entity with rounded corners, regardless of its role in the other relationships.

3.5.2.2 Non-Identifying Relationship Syntax3.5.2.2 Non-Identifying Relationship Syntax

A dashed line depicts a non-identifying relationship between the parent and child entities. Both parent and child entities will be identifier-independent entities in a non-identifying
relationship unless either or both are child entities in some other relationship which is an identifying relationship.

3.5.2.3 Mandatory Non-Identifying Relationship Syntax

In a mandatory non-identifying relationship, each instance of the child entity is related to exactly one instance of the parent entity. See Figure 6.



* The Parent Entity in a Mandatory Non-Identifying Relationship may be an Identifier-Independent Entity (as shown) or an Identifier-Dependent Entity depending upon other relationships.

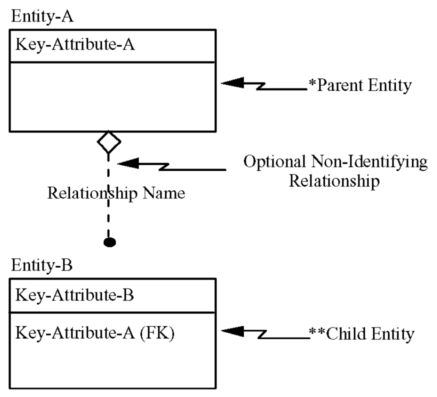
** The Child Entity in a Mandatory Non-Identifying Relationship will be an Identifier-Independent Entity unless the entity is also a Child Entity in some Identifying Relationship.

Figure 6. Mandatory Non-Identifying Relationship Syntax

3.5.2.4 Optional Non-Identifying Relationship Syntax3.5.2.4 Optional Non-Identifying Relationship Syntax

A dashed line with a small diamond at the parent end depicts an optional non-identifying relationship between the parent and child entities. See Figure 7. In an optional non-identifying relationship, each instance of the child entity is related to zero or one instances of the parent entity.

An optional non-identifying relationship represents a conditional existence dependency. An instance of the child in which each foreign key attribute for the relationship has a value must have an associated parent instance in which the primary key attributes of the parent are equal in value to the foreign key attributes of the child.



* The Parent Entity in a Optional Non-Identifying Relationship may be an Identifier-Independent Entity (as shown) or an Identifier-Dependent Entity depending upon other relationships.

** The Child Entity in a Optional Non-Identifying Relationship will be an Identifier-Independent Entity unless the entity is also a Child Entity in some Identifying Relationship.

Figure 7. Optional Non-Identifying Relationship Syntax

3.5.3 Specific Connection Relationship Label3.5.3 Specific Connection Relationship Label

A relationship is given a name, expressed as a verb or verb phrase placed beside the relationship line. The name of each relationship between the same two entities must be unique, but the relationship names need not be unique within the model. The relationship name for a specific connection relationship is usually expressed in the parent-to-child direction, such that a sentence can be formed by combining the parent entity name, relationship name, cardinality expression, and child entity name. For example, the statement "A project funds one or more tasks" could be derived from a relationship showing PROJECT as the parent entity, TASK as the child entity with a "P" cardinality symbol, and "funds" as the relationship name. When a relationship is named from both the parent and child perspectives, the parent perspective is stated first, followed by the symbol "/" and then the child perspective. Note that the relationship must still hold true when stated from the reverse direction if the child-to-parent relationship is not named explicitly. From the previous example, it is inferred that "a task is funded by exactly one project." The child perspective here is represented as "is funded by". The full relationship label for this example, including both parent and child perspectives, would be "funds / is funded by". The parent perspective should be stated for all specific connection relationships.

A second method may be used for naming the relationship from the child perspective. The direct object of the phrase may be used in place of the entire verb phrase. The verb phrase is completed when reading the relationship by inserting the implied term "has". A pattern for reading a relationship described in this style is "A <child entity name> has <cardinality> <phrase object> <parent entity name>". Using the previous example of the relationship between project and task, the direct object of the phrase is "funding" The full relationship label then becomes "funds / funding". The reverse relationship is read "a task has exactly one funding project".

3.5.4 Specific Connection Relationship Rules3.5.4 Specific Connection Relationship Rules

a) A specific connection relationship is always between exactly two entities, a parent entity and a child entity.

b) In an identifying relationship, and in a mandatory non-identifying relationship, each instance of a child entity must always be associated with exactly one instance of its parent entity.

c) In an optional non-identifying relationship, each instance of a child entity must always be associated with zero or one instance of its parent entity.

d) An instance of a parent entity may be associated with zero, one, or more instances of the child entity depending on the specified cardinality.

e) The child entity in an identifying relationship is always an identifier-dependent entity.

f) The child entity in a non-identifying relationship will be an identifier-independent entity unless the entity is also a child entity in some identifying relationship.

g) An entity may be associated with any number of other entities as either a child or a parent.

h) Only non-identifying relationships may be recursive, i.e., may relate an instance of an entity to another instance of the same entity.

3.6 Categorization Relationships3.6 Categorization Relationships

Categorization relationships are used to represent structures in which an entity is a "type" (category) of another entity.

3.6.1 Categorization Relationship Semantics3.6.1 Categorization Relationship Semantics

Entities are used to represent the notion of "things about which we need information." Since some real world things are categories of other real world things, some entities must, in some sense, be categories of other entities. For example, suppose employees are something about which information is needed. Although there is some information needed about all employees, additional information may be needed about salaried employees which is different, from the additional information needed about hourly employees. Therefore, the entities SALARIED-EMPLOYEE and HOURLY-EMPLOYEE are categories of the entity EMPLOYEE. In an IDEF1X view, they are related to one another through categorization relationships.

In another case, a category entity may be needed to express a relationship which is valid for only a specific category, or to document the relationship differences among the various categories of the entity. For example, a FULL-TIME-EMPLOYEE may qualify for a BENEFIT, while a PART-TIME-EMPLOYEE may not.

A "categorization relationship" is a relationship between one entity, referred to as the "generic entity", and another entity, referred to as a "category entity". A "category cluster" is a set of one or more categorization relationships. An instance of the generic entity can be associated with an instance of only one of the category entities in the cluster, and each instance of a category entity is associated with exactly one instance of the generic entity. Each instance of the category entity represents the same real-world thing as its associated instance in the generic entity. From the previous example, EMPLOYEE is the generic entity and SALARIED-EMPLOYEE and HOURLY-EMPLOYEE are the category entities. There are two categorization relationships in this cluster, one between EMPLOYEE and SALARIED-EMPLOYEE and one between EMPLOYEE and HOURLY-EMPLOYEE.

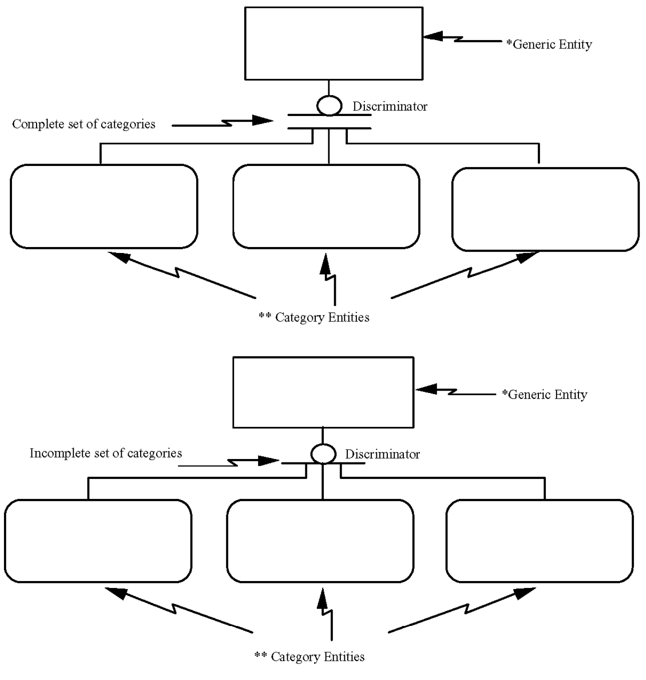
Since an instance of the generic entity cannot be associated with an instance of more than one of the category entities in the cluster, the category entities are mutually exclusive. In the example, this implies that an employee cannot be both salaried and hourly. However, an entity can be the generic entity in more than one category cluster, and the category entities in one cluster are not mutually exclusive with those in others. For example, EMPLOYEE could be the generic entity in a second category cluster with FEMALE-EMPLOYEE and MALE-EMPLOYEE as the category entities. An instance of EMPLOYEE could be associated with an instance of either SALARIED-EMPLOYEE or HOURLY-EMPLOYEE and with an instance of either FEMALE-EMPLOYEE or MALE-EMPLOYEE.

In a “complete category cluster”, every instance of the generic entity is associated with an instance of a category entity, i.e., all the possible categories are present. For example, each employee is either male or female, so the second cluster is complete. In an “incomplete category cluster”, an instance of the generic entity can exist without being associated with an instance of any of the category entities, i.e., some categories are omitted. For example, if some employees are paid commissions rather than an hourly wage or salary, the first category cluster would be incomplete.

An attribute in the generic entity, or in one of its ancestors, may be designated as the discriminator for a specific category cluster of that entity. The value of the discriminator determines the category of an instance of the generic. In the previous example, the discriminator for the cluster including the salaried and hourly categories might be named EMPLOYEE-TYPE. If a cluster has a discriminator, it must be distinct from all other discriminators.

3.6.2 Categorization Relationship Syntax3.6.2 Categorization Relationship Syntax

A category cluster is shown as a line extending from the generic entity to a circle which is underlined. Separate lines extend from the underlined circle to each of the category entities in the cluster. Each line pair, from the generic entity to the circle and from the circle to the category entity, represents one of the categorization relationships in the cluster. Cardinality is not specified for the category entity since it is always zero or one. Category entities are also always identifier-dependent. See Figure 8. The generic entity is independent unless its identifier has migrated through some other relationship.



* The Generic Entity may be an Identifier-Independent Entity (as shown) or an Identifier-Dependent Entity depending upon other relationships.

** Category Entities will always be Identifier-Dependent Entities.

Figure 8. Categorization Relationship Syntax

If the circle has a double underline, it indicates that the set of category entities is complete. A single line under the circle indicates an incomplete set of categories.

The name of the attribute used as the discriminator (if any) is written with the circle. Although the categorization relationships themselves are not named explicitly, each generic entity to category entity relationship can be read as “can be.” For example, an EMPLOYEE can be a SALARIED-EMPLOYEE. If the complete set of categories is referenced, the relationship may be read as “must be.” For example, an EMPLOYEE must be a MALE-EMPLOYEE or a FEMALE-EMPLOYEE. The relationship is read as “is a/an” from the reverse direction. For example, an HOURLY-EMPLOYEE is an EMPLOYEE.

The generic entity and each category entity must have the same primary key attributes. However, role names may be used in the category entities. (See Foreign Keys in Section 3.9.)

3.6.3 Categorization Relationship Rules3.6.3 Categorization Relationship Rules

a) A category entity can have only one generic entity. That is, it can only be a member of the set of categories for one category cluster.

b) A category entity in one categorization relationship may be a generic entity in another categorization relationship.

c) An entity may have any number of category clusters in which it is the generic entity. (For example, FEMALE-EMPLOYEE and MALE-EMPLOYEE may be a second set of categories for the generic entity EMPLOYEE.)

d) The primary key attribute(s) of a category entity must be the same as the primary key attribute(s) of the generic entity. However, role names may be assigned in the category entity.

e) All instances of a category entity have the same discriminator value and all instances of different categories must have different discriminator values.

f) No entity can be its own generic ancestor, that is, no entity can have itself as a parent in a categorization relationship, nor may it participate in any series of categorization relationships that specifies a cycle.

g) No two category clusters of a generic entity may have the same discriminator.

h) The discriminator (if any) of a complete category cluster must not be an optional attribute.

A category entity cannot be a child entity in an identifying connection relationship unless the primary key contributed by the identifying relationship is completely contained within the primary key of the category, while at the same time the category primary key satisfies rule d above.

3.7 Non-Specific Relationships3.7 Non-Specific Relationships

Non-specific relationships are used in high-level Entity-Relationship views to represent many-to-many associations between entities.

3.7.1 Non-Specific Relationship Semantics3.7.1 Non-Specific Relationship Semantics

Both parent-child connection and categorization relationships are considered to be specific relationships because they define precisely how instances of one entity relate to instances of another entity. In a key-based or fully-attributed view, all associations between entities must be expressed as specific relationships. However, in the initial development of a model, it is often helpful to identify “non-specific relationships” between entities. These non-specific relationships are refined in later development phases of the model. The procedure for resolving non-specific relationships is discussed in Annex A Section A3.4.1.

A non-specific relationship, also referred to as a “many-to-many relationship,” is an association between two entities in which each instance of the first entity is associated with zero, one, or many instances of the second entity and each instance of the second entity is associated with zero, one, or many instances of the first entity. For example, if an employee can be assigned to many projects and a project can have many employees assigned, then the connection between the entities EMPLOYEE and PROJECT can be expressed as a non-specific relationship. This non-specific relationship can be replaced with specific relationships later in the model development by introducing a third entity, such as PROJECT-ASSIGNMENT, which is a common child entity in specific connection relationships with the EMPLOYEE and PROJECT entities. The new relationships would specify that an employee has zero, one, or more project assignments. Each project assignment is for exactly one employee and exactly one project. Entities introduced to resolve non-specific relationships are sometimes called "intersection" or "associative" entities.

A non-specific relationship may be further defined by specifying the cardinality from both directions of the relationship. Any of the cardinalities listed in Figure 4 in Section 3.5.2.1 may be used on either end of the relationship.

3.7.2 Non-Specific Relationship Syntax3.7.2 Non-Specific Relationship Syntax

A non-specific relationship is depicted as a line drawn between the two associated entities with a dot at each end of the line. See Figure 9. Cardinality may be expressed at both ends of the relationship as shown in Figure 4 in Section 3.4. A "P" (for positive) placed beside a dot indicates that for each instance of the entity at the other end of the relationship there are one or more instances of the entity at the end with the "P." A "Z" placed beside a dot indicates that for each instance of the entity at the other end of the relationship there are zero or one instances of the entity at the end with the "Z." In a similar fashion, a positive integer number or minimum and maximum positive integer range may be placed beside a dot to specify an exact cardinality. The default cardinality is zero, one, or more.

A non-specific relationship may be named in both directions. The relationship label is expressed as a pair of verb phrases placed beside the relationship line and separated by a slash, ("/.") The order of the verb phrases depends on the relative position of the entities. The first expresses the relationship from either the left entity to the right entity, if the entities are arranged horizontally, or the top entity to the bottom entity, if they are arranged vertically. The second expresses the relationship from the other direction, that is, either the right entity to the left entity or the bottom entity to the top entity, again depending on the orientation. Top-to-bottom orientation takes precedence over left-to-right, so if the entities are arranged upper right and lower left, the first verb phrase describes the relationship from the perspective of the top entity. The relationship is labeled such that sentences can be formed by combining the entity names with the phrases. For example, the statements "A project has zero, one, or more employees" and "An employee is assigned zero, one, or more projects" can be derived from a non-specific relationship labeled "has / is assigned" between the entities PROJECT and EMPLOYEE. (The sequence assumes the PROJECT appears above or to the left of the entity EMPLOYEE.)

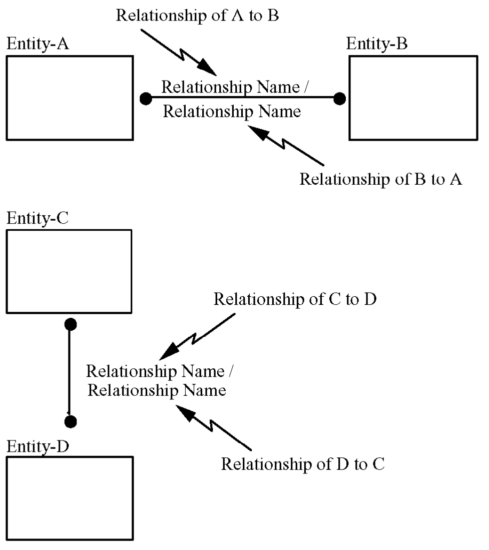


Figure 9. Non-Specific Relationship Syntax

3.7.3 Non-Specific Relationship Rules3.7.3 Non-Specific Relationship Rules

a) A non-specific relationship is always between exactly two entities.

b) An instance of either entity may be associated with zero, one, or more instances of the other entity depending on the specified cardinality.

c) In a key-based or fully-attributed view, all non-specific relationships must be replaced by specific relationships.

d) Non-specific relationships may be recursive, i.e., may relate an instance of an entity to another instance of the same entity.

3.8 Primary and Alternate Keys3.8 Primary and Alternate Keys

Primary and alternate keys represent uniqueness constraints over the values of entity attributes.

3.8.1 Primary and Alternate Key Semantics3.8.1 Primary and Alternate Key Semantics

A “candidate key” of an entity is one or more attributes whose value uniquely identifies every instance of the entity. For example, the attribute PURCHASE-ORDER-IDENTIFIER may uniquely identify an instance of the entity PURCHASE-ORDER. A combination of the attributes ACCOUNT-IDENTIFIER and CHECK-IDENTIFIER may uniquely identify an instance of the entity CHECK.

In key-based and fully-attributed views, every entity must have at least one candidate key. In some cases, an entity may have more than one attribute or group of attributes which uniquely identify instances of the entity. For example, the attributes EMPLOYEE-IDENTIFIER and EMPLOYEE-SOCIAL-SECURITY-NUMBER may both uniquely identify an instance of the entity EMPLOYEE. If more than one candidate key exists, then one candidate key is designated as the “primary key” and the other candidate keys are designated as “alternate keys.” If only one candidate key exists, then it is, of course, the primary key.

3.8.2 Primary and Alternate Key Syntax3.8.2 Primary and Alternate Key Syntax

Attributes which define the primary key are placed at the top of the attribute list within an entity box and separated from the other attributes by a horizontal line. See previously referenced Figure 3 in Section 3.4.

Each alternate key is assigned a unique integer number and is shown by placing the note “AK" plus the alternate key number in parentheses, e.g., “(AK1),” to the right of each of the attributes in the key. See Figure 10. An individual attribute may be identified as part of more than one alternate key. A primary key attribute may also serve as part of an alternate key.

Alternate Key Syntax

Attribute-Name (AKn)[(AKm). . .]

or

Attribute-Name (AKn[, AKm. . .])

Where n, m, etc., uniquely identify each Alternate Key that includes the associated attribute and where an Alternate Key consists o all the attributes with the same identifier.

Example

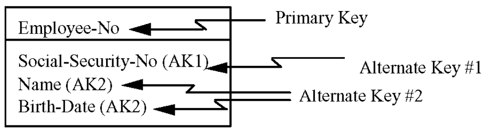


Figure 10. Alternate Key Syntax

3.8.3 Primary and Alternate Key Rules3.8.3 Primary and Alternate Key Rules

a) In a key-based or fully-attributed IDEF1X view, every entity must have a primary key.

b) An entity may have any number of alternate keys.

c) A primary or alternate key may consist of a single attribute or combination of attributes.

d) An individual attribute may be part of more than one key, either primary or alternate.

e) Attributes which form primary and alternate keys of an entity may either be owned by the entity or migrated through a relationship. (See Foreign Keys in Section 3.9.)

f) Primary and alternate keys must contain only those attributes that contribute to unique identification (i.e., if any attribute were not included as part of the key then every instance of the entity could not be uniquely identified; referred to as the Smallest-Key Rule).

g) If the primary key is composed of more than one attribute, the value of every non-key attribute must be functionally dependent upon the entire primary key, i.e., if the primary key is known, the value of each non-key attribute is known, and no non-key attribute value can be determined by just part of the primary key (referred to as the Full-Functional-Dependency Rule).

h) Every attribute that is not part of a primary or alternate key must be functionally dependent only upon the primary key and each of the alternate keys, i.e., no such attribute's value can be determined by another such attribute's value (referred to as the No-Transitive-Dependency Rule).

3.9 Foreign Keys3.9 Foreign Keys

Foreign keys are attributes in entities which designate instances of related entities.

3.9.1 Foreign Key Semantics3.9.1 Foreign Key Semantics

If a specific connection or categorization relationship exists between two entities, then the attributes which form the primary key of the parent or generic entity are migrated as attributes of the child or category entity. These migrated attributes are referred to as "foreign keys." For example, if a connection relationship exists between the entity PROJECT as a parent and the entity TASK as a child, then the primary key attributes of PROJECT would be migrated attributes of the entity TASK. For example, if the attribute PROJECT-ID were the primary key of PROJECT, then PROJECT-ID would also be a migrated attribute or foreign key of TASK.

A migrated attribute may be used as either a portion or total primary key, alternate key, or non-key attribute within an entity. If all the primary key attributes of a parent entity are migrated as part of the primary key of the child entity, then the relationship through which the attributes were migrated is an "identifying relationship." (See Section 3.5.1.1.) If any of the migrated attributes are not part of the primary key of the child entity, then the relationship is a "non-identifying relationship". (See Section 3.5.1.2.) For example, if tasks were only uniquely numbered within a project, then the migrated attribute PROJECT-ID would be combined with the owned attribute TASK-ID to define the primary key of TASK. The entity PROJECT would have an identifying relationship with the entity TASK. If on the other hand, the attribute TASK-ID is always unique, even among projects, then the migrated attribute PROJECT-ID would be a non-key attribute of the entity TASK. In this case, the entity PROJECT would have a non-identifying relationship with the entity TASK. When only a portion of a migrated primary key becomes part of the primary key of the child entity, with the remainder becoming non-key attribute(s) of the child, the contributed foreign key is called a "split key". If a key is "split", the relationship is non-identifying.

In a categorization relationship, both the generic entity and the category entities represent the same real-world thing. Therefore, the primary key for all category entities is migrated through the categorization relationship from the primary key of the generic entity. For example, if SALARIED-EMPLOYEE and HOURLY-EMPLOYEE are category entities and EMPLOYEE is the generic entity, then if the attribute EMPLOYEE-IDENTIFIER is the primary key for the entity EMPLOYEE, it would also be the primary key for the entities SALARIED-EMPLOYEE and HOURLY-EMPLOYEE.

In some cases, a child entity may have multiple relationships to the same parent entity. The primary key of the parent entity would appear as migrated attributes in the child entity for each relationship. For a given instance of the child entity, the value of the migrated attributes may be different for each relationship, i.e., two different instances of the parent entity may be referenced. A bill of material structure, for example, can be represented by two entities PART and PART-ASSEMBLY-STRUCTURE. The entity PART has a dual relationship as a parent entity to the entity PART-ASSEMBLY-STRUCTURE. The same part sometimes acts as a component from which assemblies are made, i.e., a part may be a component in one or more assemblies, and sometimes acts as an assembly into which components are assembled, i.e., a part may be an assembly for one or more component parts. If the primary key for the entity PART is PART-IDENT, then PART-IDENT would appear twice in the entity PART-ASSEMBLY-STRUCTURE as a migrated attribute. However, since an attribute may appear only once in any entity, the two occurrences of PART-IDENT in PART-ASSEMBLY-STRUCTURE are merged unless a role name is assigned for one or both.

3.9.1.1 Role Name Semantics3.9.1.1 Role Name Semantics

When an attribute migrates into an entity through more than one relationship, a "role name" may need to be assigned to each occurrence to distinguish among them. If an instance of the entity can have one value for one occurrence, and another value for another, then each occurrence must be given a different role name. On the other hand, if each instance of the entity must have the same value for two or more occurrences, they must each have the same name. From the previous example, role names of COMPONENT-PART-IDENT and ASSEMBLY-PART-IDENT should be assigned to distinguish between the two migrated PART-IDENT attribute occurrences. Although not required, a role name may also be used with a single occurrence of a migrated attribute to more precisely convey its meaning within the context of the child entity.

3.9.2 Foreign Key Syntax3.9.2 Foreign Key Syntax

A foreign key is shown by placing the names of the migrated attributes inside the entity box and by following each with the letters "FK" in parentheses, i.e., "(FK)." See Figure 11. If all migrated attributes belong to the primary key of the child entity, each is placed above the horizontal line and the entity is drawn with rounded corners to indicate that the identifier (primary key) of the entity is dependent upon an attribute migrated through a relationship. If any migrated attribute does not belong to the primary key of the child entity, the attribute is placed below the line, and the entity shape may be rounded or square-cornered depending on the presence or absence of identifying relationships to this entity. Migrated attributes may also be part of an alternate key.

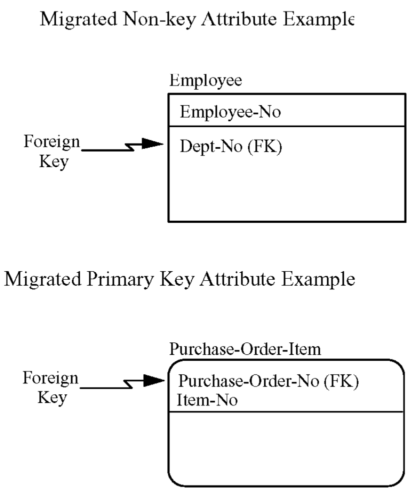


Figure 11. Foreign Key Syntax

3.9.2.1 Role Name Syntax3.9.2.1 Role Name Syntax

Role names, like domain (attribute) names, are noun phrases. A role name is followed by the name of the migrated attribute, separated by a period. See Figure 12.

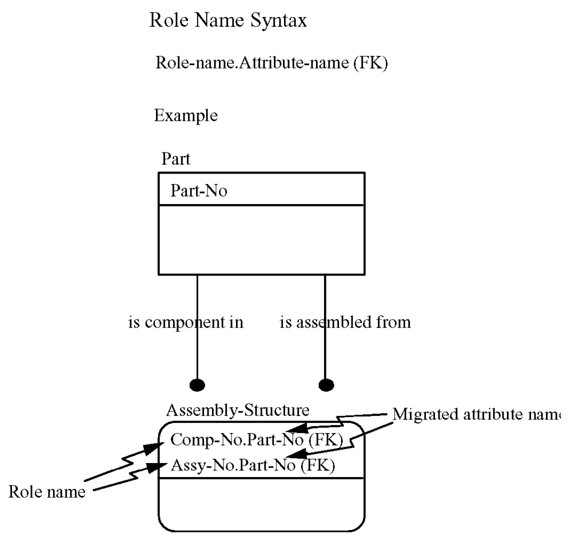


Figure 12. Role Name Syntax

3.9.3 Foreign Key Rules3.9.3 Foreign Key Rules

a) Every entity must contain a set of foreign key attributes for each specific connection or categorization relationship in which it is the child or category entity. A given attribute can be a member of multiple such sets. Further, the number of attributes in the set of foreign key attributes must be the same as the number of attributes of the primary key of the parent or generic entity.

b) The primary key of a generic entity must be migrated as the primary key for each category entity.

c) A child entity must not contain two entire foreign keys that identify the same instance of the same ancestor (parent or generic entity) for every instance of the child unless these foreign keys are contributed via separate relationship paths containing one or more intervening entities between the ancestor and the child.

d) Every migrated attribute of a child or category entity must represent an attribute in the primary key of a related parent or generic entity. Conversely, every primary key attribute of a parent or generic entity must be a migrated attribute in a related child or category entity.

e) Each role name assigned to a migrated attribute must be unique and the same meaning must always apply to the same name. Furthermore, the same meaning cannot apply to different names unless the names are aliases.

f) A migrated attribute may be part of more than one foreign key provided that the attribute always has the same value for these foreign keys in any given instance of the entity. A role name may be assigned for this migrated attribute.

g) Every foreign key attribute must reference one and only one of the primary key attributes of the parent. An attribute A references another attribute B if A=B or A is a direct or indirect sub-type of B. An attribute A is considered a sub-type of B if A is an alias for C and C is a sub-type of B, or A is a sub-type of C and C is an alias for B.

3.10 View Levels3.10 View Levels