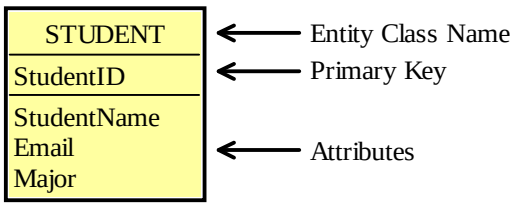
Entity-Relationship Diagrams
Prepared for Students of
CMST 283/332

I. Basic Symbols

An entity class (or entity set) represents a table. It has attributes (representing the columns of the table) and an identifier (representing the primary key). It is drawn as a three-part rectangle. For example, the table:

STUDENT(StudentID, StudentName, Email, Major)

Is represented by this entity class:



Each item within the class is an entity or entity instance. For example, if the STUDENT table contains the data:

111	Tom	tom@myu.edu	MATH
222	Dick	dick@myu.edu	BUS
333	Harry	harry@myu.edu	LIFE

Tom, Dick and Harry are the entities within it.

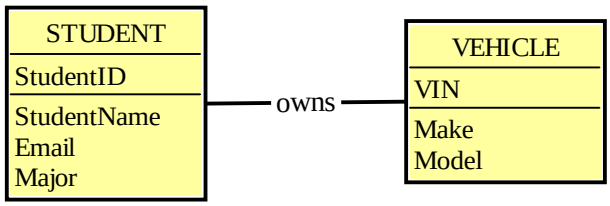
A relationship is an association between entities in different classes. For example, suppose we have, along with our table of students, a table of vehicles:

VEHICLE(VIN, Make, Model)

Containing data:

FM1	Ford	Mustang
PGP1	Pontiac	Grand Prix

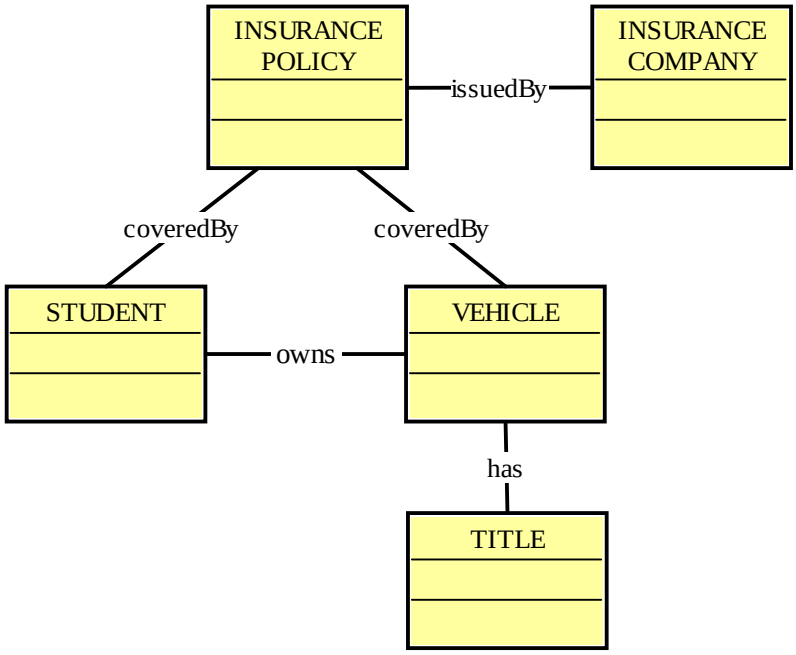
Ownership is a relationship between students and vehicles. If Tom owns the Mustang and Dick owns the Grand Prix, these are relationship instances. Relationships are drawn as named lines between entity classes, as follows:



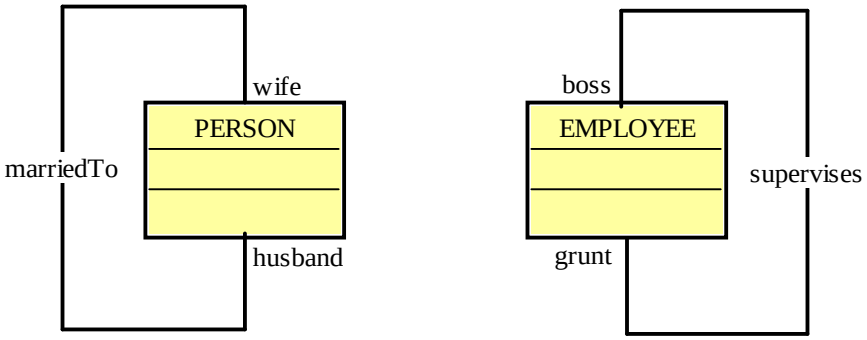
II. Degree

The degree of a relationship specifies the number of entity classes connected by it. In our diagrams, each relationship has one of three possible degrees.

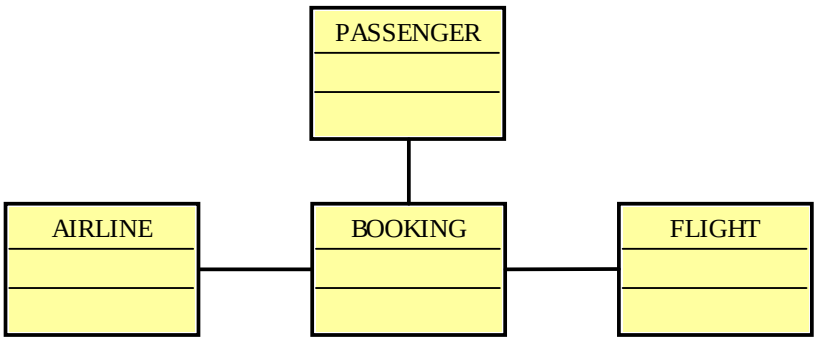
II.A. A binary relationship has degree = 2.



II.B. A unary relationship has degree = 1. For clarity, especially with unary relationships, you can specify a role at each end of the relationship. (Your textbook calls unary relationships *recursive* relationships.)



II.C. A ternary relationship has degree = 3. Such a relationship is drawn as a fourth entity class.



In the example above, BOOKING is the ternary relationship. It connects a particular passenger with a particular flight on a particular airline. For instance, suppose the three entity classes contained:

PASSENGER	
Tom	tom@mymail.net
Mary	mary@mymail.net

AIRLINE	
UN	United
DE	Delta

FLIGHT			
UN1	MCI	PHL	10:00 AM
UN2	PHL	MCI	8:00 AM
UN3	PHL	MDT	12:00 PM
DE1	MCI	PHL	10:15 AM

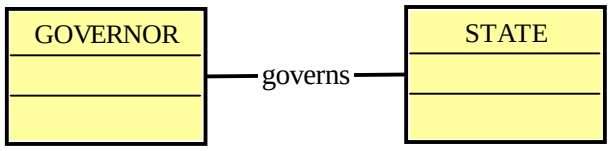
The BOOKING relationship holding Tom's booking for Kansas City to Harrisburg and Mary's round-trip booking between Kansas City and Philadelphia appears as:

BOOKING			
Tom	UN	UN1	June 6, 2014
Tom	UN	UN3	June 6, 2014
Mary	DE	DE1	July 1, 2014
Mary	UN	UN2	July 15, 2014

III. Cardinality

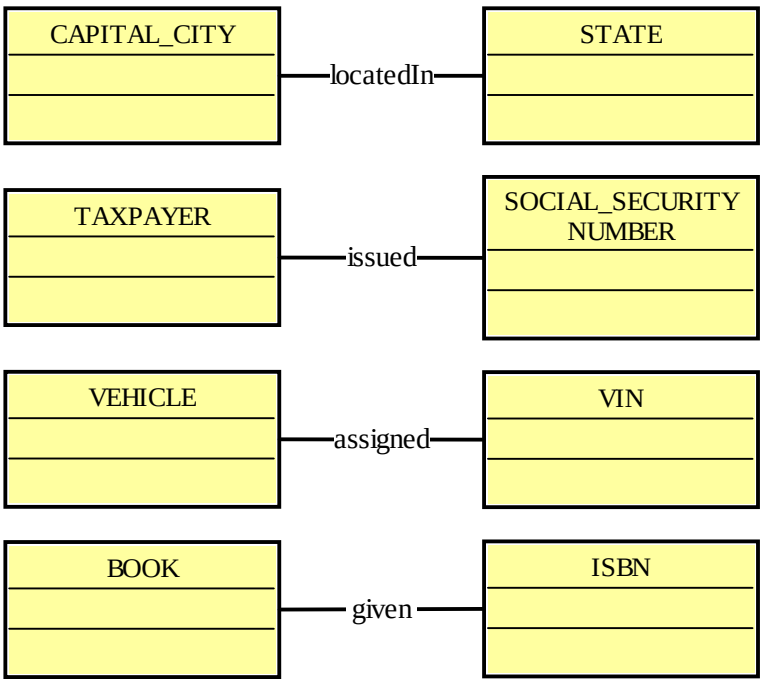
Pretend we have a relationship between entity classes *A* and *B*. The cardinality of the relationship specifies how many entities in *A* can be related to a single entity in *B*. Any relationship has one of three possible cardinalities: one-to-one, one-to-many or many-to-many.

III.A. One-to-one relationship: any single entity in *A* is related to at most one entity in *B* and vice-versa. We write *one-to-one* as *1:1*.



At any given instant of time, each governor is governor of exactly one state and each state has exactly one governor.

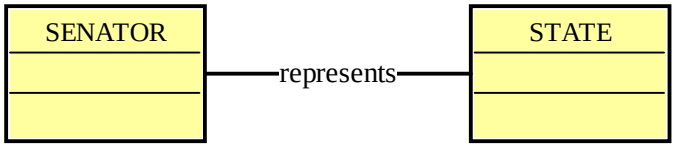
Here are some more 1:1 relationships:



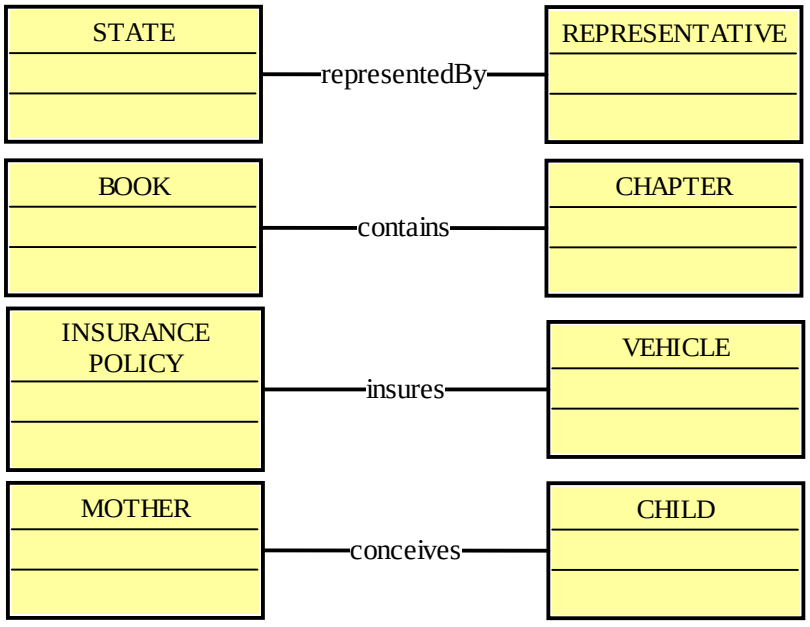
III.B. <u>One-to-many relationship</u>: any single entity in *A* is related to zero, one or more entities in *B* but any single entity in *B* is related to at most one entity in *A*. We write *one-to-many* as *1:N*.

At any given instant of time, each state has 2 senators but each senator can be elected from only one state.

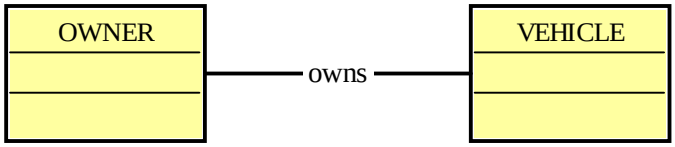
Some designers follow a convention of always putting the "many" entity class on the left (or top), as in:



We could call this a *many-to-one* relationship, and many authors do so, but it is equivalent to the one-to-many relationship of the opposite direction. Here are more 1:*N* relationships with the "one entity class" placed on the left:

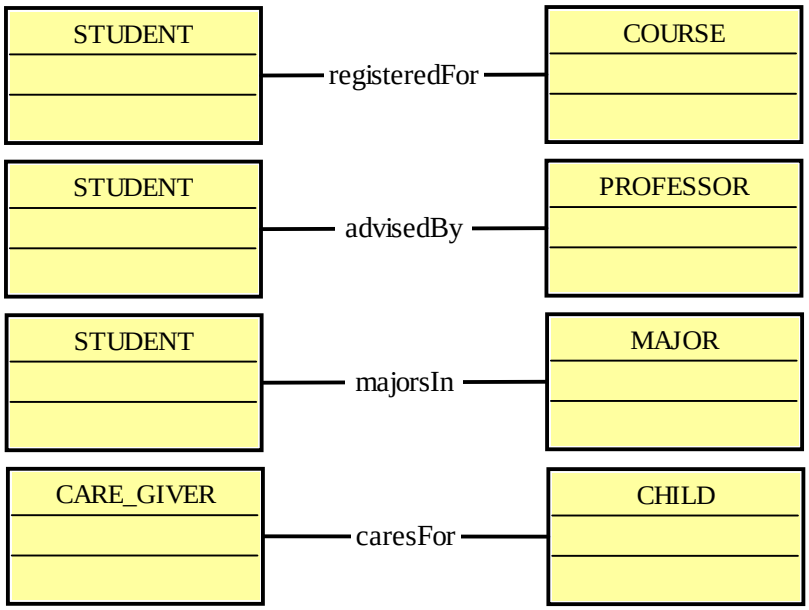


III.C. Many-to-many relationship: any single entity in *A* is related to zero, one or more entities in *B* and vice-versa. We write *many-to-many* as *N:M*.



At any given instant of time, an owner can own several vehicles and each vehicle can have several owners (say husband and wife).

Other *N*:*M* relationships:



IV. Modality

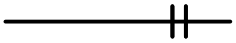
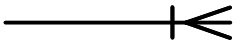
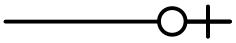
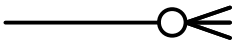
Pretend we have a relationship between entity classes *A* and *B*. The modality of the relationship specifies whether or not an entity in *A* must be related to an entity in *B* and vice-versa.

The two modalities are mandatory and optional. For example, a taxpayer has a social security number but a number may exist that hasn't been assigned to a taxpayer. A vehicle must be owned by somebody but there may be somebody who doesn't own a vehicle.

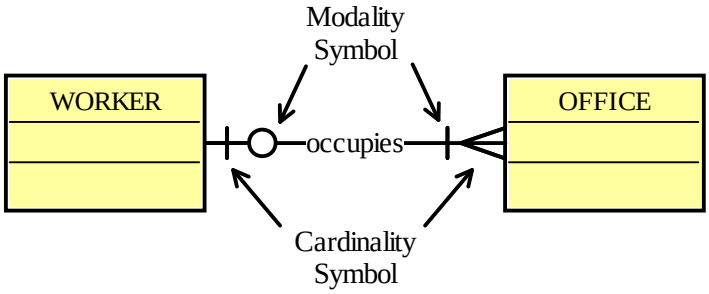
Modality can be referred to as the relationship's *minimum cardinality*, the minimum number of entities that must participate in a relationship instance. An optional relationship has minimum cardinality of 0 whereas a mandatory relationship has minimum cardinality of 1. For example, a vehicle has at least 1 owner but a person may own 0 vehicles. Similarly the relationship's cardinality can be referred to as its *maximum cardinality*.

V. Drawing Cardinality and Modality

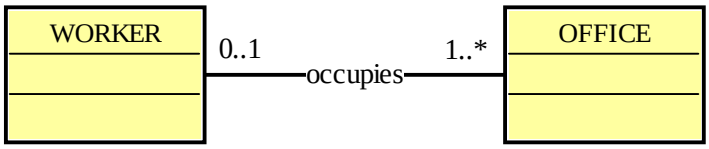
To draw cardinality and modality, you need two symbols for cardinality (one or many) and two symbols for modality (optional or mandatory), for a total of 2•2 = 4 symbols. There are two styles of symbols. The older style is the crow's foot style; the newer is the UML style (UML stands for Unified Modeling Language).

Left Symbol = Modality	**Right Symbol = Cardinality**	**Crow's Foot**	**UML**	**Meaning**
Mandatory	One		1..1	Exactly 1
Mandatory	Many		1..*	One or more
Optional	One		0..1	Zero or one
Optional	Many		0..*	Zero or more

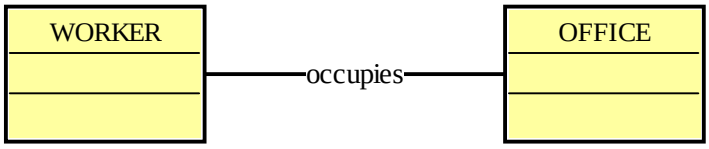
For your ERD to be complete, you must draw these symbols on both sides of each relationship line. For the crow's foot style, the cardinality symbol is always the one closest to the entity class. For example, the following diagram specifies that a worker has one or more offices and an office has zero or one occupants.



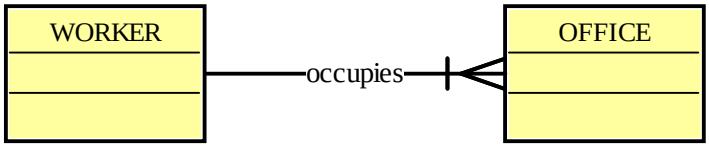
When using UML notation, the minimum cardinality is always on the left and the maximum cardinality is always on the right, as in:



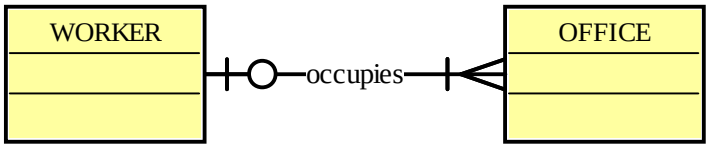
Pay close attention to the correct direction of the cardinality. The most common mistake that beginners make drawing E-R diagrams is to place the cardinality symbols on the wrong end of the relationship line. Here's how to do it correctly. Take, for instance, the WORKER and OFFICE entity classes:



First pretend you're a WORKER and ask yourself, "How many offices can I have?" Whatever the answer is, draw those symbols next to the OFFICE entity class. For example, if the answer is, "I must have an office and I can have more than one office," then draw:

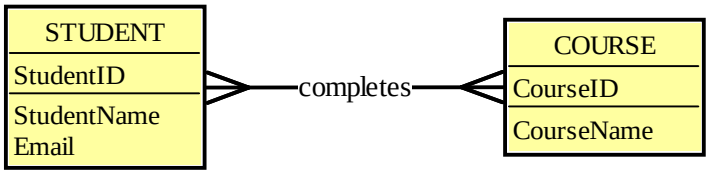


Now pretend you're an OFFICE and ask yourself, "How many workers can occupy me?" and draw those symbols next to the WORKER entity class. For example, if the answer is, "I may not be occupied and I can have only one occupant," then draw:



VII. Associative Classes

Consider the STUDENT and COURSE example:



Assume we have the following data and that both students have completed both courses:

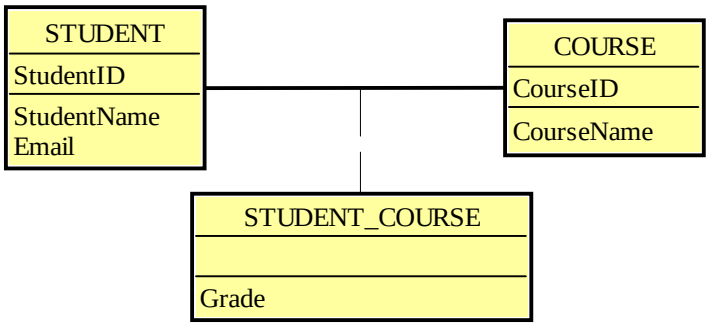
STUDENT		
111	Tom	tom@myu.edu
222	Mary	mary@myu.edu

COURSE	
CMST 180	Introduction to Database Systems
CMST 103	Program Design

STUDENT_COURSE		
Tom	CMST 180	B
Tom	CMST 103	C
Mary	CMST 180	A
Mary	CMST 103	B

The grades are **data associated** with the **relationship instance rather than** the **entity instance**. For example, Tom's grades cannot be put into the STUDENT table because Tom has more than one. Likewise, the grades for CMST 180 cannot be put into the course table because CMST 180 has more than one student. Each grade is associated with the student/course pair. Thus, it needs to be stored in a separate table that we call an associative class.

An associative class is only needed with an *N*:*M* binary relationship, and is drawn as follows:

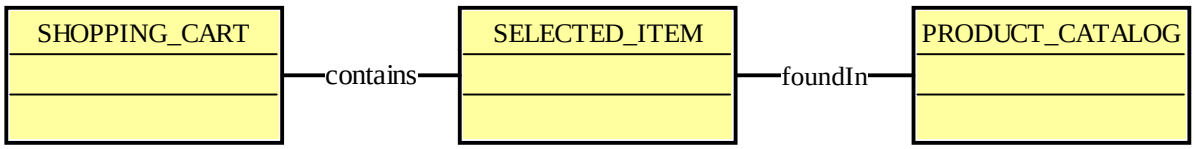


When an *N*:*M* binary relationship requires an associative class, there is no need to draw the modality/cardinality symbols.

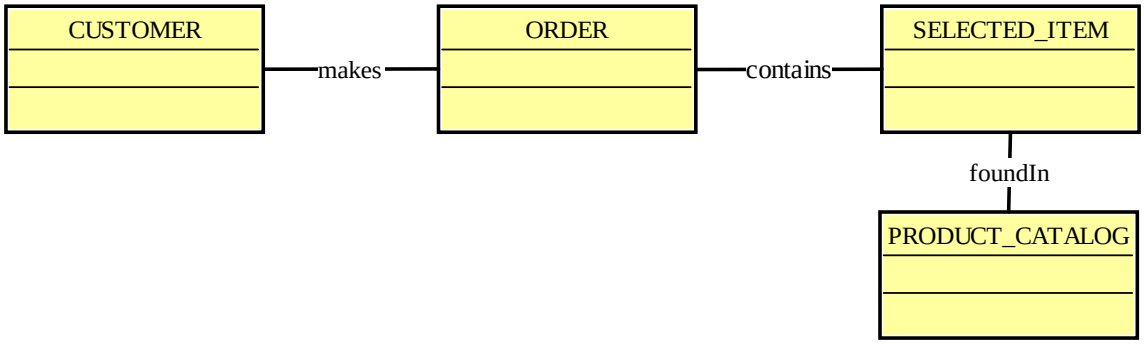
VIII. Weak vs. Strong Entities

VIII.A. A weak entity cannot exist in a database unless another type of entity also exists in that database. An entity that is not weak is strong; i.e. a strong entity's existence doesn't depend on the existence of another entity.

For example, think of a dotcom business, say Amazon.com. They have a database containing a set of products to sell – strong entities that exist in and of themselves. Pretend that you are browsing the online catalog and you place something it into a shopping cart. The shopping cart is a strong entity but it contains weak entities, the items you have selected to purchase, which can't exist unless there is a shopping cart in which to place them.

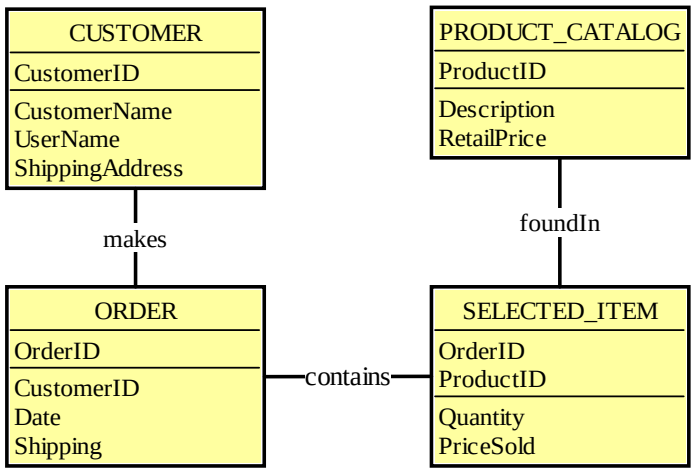


Most dotcom businesses allow you to create a shopping cart without being a registered user. However, once you decide to buy (i.e. click the Checkout button) then you must log into your account. Thus, when the shopping cart becomes an actual order, it becomes weak – dependent upon the existence of the customer who ordered the items.



In practice, I have never found the distinction between weak and strong entities to impact my table design, so I don't explicitly identify them within the ERD by using a different symbol. Many authors use a rounded rectangle for a weak entity class and a dashed line to connect it to the class on which it depends.

VIII.B. If the primary key of a weak entity contains the primary key of the entity on which it depends, it is an ID-dependent entity. In the dotcom example:



SELECTED_ITEM is an ID-dependent weak entity class because its primary key contains the primary keys of ORDER and PRODUCT_CATALOG. ORDER is a non-ID-dependent weak entity class because it has its own unique primary key (even though it has the CustomerID as a foreign key). CUSTOMER and PRODUCT_CATALOG are strong entity classes.