Entity-Relationship Model

The entity-relationship (E-R)

Data model perceives the real world as consisting of basic objects, called *entities*, and *relationships* among these objects. It was developed to facilitate database design by allowing specification of an *enterprise schema*, which represents the overall logical structure of a database. The E-R data model is one of several semantic data models; the semantic aspect of the model lies in its representation of the meaning of the data. The E-R model is very useful in mapping the meanings and interactions of real-world enterprises onto a conceptual schema. Because of this usefulness, many database-design tools draw on concepts from the E-R model.

Basic Concepts

The E-R data model employs three basic notions:

- Entity sets.
- Relationship sets.
- Attributes.

Entity Sets

*An **entity** is a "thing" or "object" in the real world that is distinguishable from all other objects.*

For *example*, each person in an enterprise is an entity. An entity has a set of properties, and the values for some set of properties may uniquely identify an entity.

For instance, a person may have a *person-id* property whose value uniquely identifies that person. Thus, the value 677-89-9011 for *person-id* would uniquely identify one particular person in the enterprise. Similarly, *loans* can be thought of as entities, and loan number L-15 at the Perryridge branch uniquely identifies a loan entity. An entity may be concrete, such as a person or a book, or it may be abstract, such as a loan, or a holiday, or a concept.

An **entity set** is a set of entities of the same type that share the same properties, or attributes. The set of *all persons who are customers at a given bank*,

For *example*, can be defined as the entity set *customer*. Similarly, the entity set *loan* might represent the set of all loans awarded by a particular bank. The individual

entities that constitute a set are said to be the *extension* of the entity set. Thus, all the individual bank customers are the extension of the entity set *customer*.

Entity sets do not need to be disjoint. For example, it is possible to define the entity set of all employees of a bank (*employee*) and the entity set of all *customers* of the bank (*customer*). A *person* entity may be an *employee* entity, a *customer* entity, both, or neither.

An entity is represented by a set of **attributes**. Attributes are descriptive properties possessed by each member of an entity set. The designation of an attribute for an entity set expresses that the database stores similar information concerning each entity in the entity set; however, each entity may have its own value for each attribute.

Possible attributes of the *customer* entity set are *customer-id*, *customer-name*, *customer-street*, and *customer-city*. In real life, there would be further attributes, such as street number, apartment number, state, postal code, and country, but we omit them to keep our examples simple. Possible attributes of the *loan* entity set are *loan-number* and *amount*.

Each entity has a **value** for each of its attributes. For instance, a particular customer entity may have the value **321-12-3123** for *customer-id*, the value Jones for *customer-name*, the value Main for *customer-street*, and the value Harrison for *customer-city*.

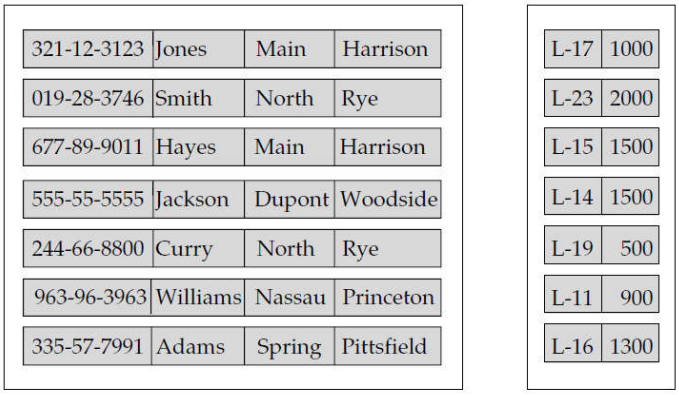
The *customer-id* attribute is used to uniquely identify customers, since there may be more than one customer with the same *name*, *street*, and *city*. In the United States, many enterprises find it convenient to use the *social-security* number of a person1 as an attribute whose value uniquely identifies the person. In general the enterprise would have to create and assign a unique identifier for each customer.

For each attribute, there is a set of permitted values, called the **domain**, or **value set**, of that attribute. The domain of attribute *customer-name* might be the set of all text strings of a certain length. Similarly, the domain of attribute *loan-number* might be the set of all strings of the form "L-*n*" where *n* is a positive integer.

A database thus includes a collection of entity sets, each of which contains any number of entities of the same type. Figure shows part of a bank database that consists of two entity sets: *customer* and *loan*.

Formally, an attribute of an entity set is a function that maps from the entity set into a domain. Since an entity set may have several attributes, each entity can be described by a set of (attribute, data value) pairs, one pair for each attribute of the entity set. For *example*, a particular *customer* entity may be described by the set *{*(*customer-id*, 677-89-9011), (*customer-name*, Hayes), (*customer-street*, Main), (*customer-city*, Harrison)*}*, meaning that the entity describes a person named Hayes whose customer identifier is 677-89-9011 and who resides at Main Street in Harrison. We can see, at this point, an integration of the abstract schema with the actual enterprise being modeled. The attribute values describing an entity will constitute a significant portion of the data stored in the database.

An attribute, as used in the E-R model, can be characterized by the following attribute types.



321-12-3123	Jones	Main	Harrison
019-28-3746	Smith	North	Rye
677-89-9011	Hayes	Main	Harrison
555-55-5555	Jackson	Dupont	Woodside
244-66-8800	Curry	North	Rye
963-96-3963	Williams	Nassau	Princeton
335-57-7991	Adams	Spring	Pittsfield

customer

L-17	1000
L-23	2000
L-15	1500
L-14	1500
L-19	500
L-11	900
L-16	1300

loan

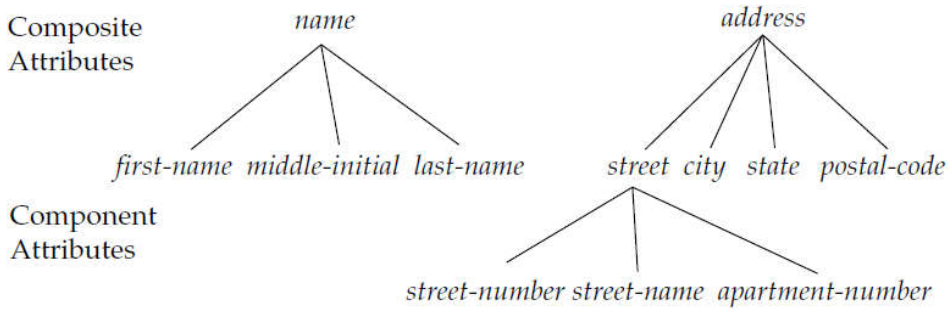
Entity sets *customer* and *loan*.

Simple and Composite Attributes

In our examples thus far, the attributes have been simple; that is, they are not divided into subparts. **Composite** attributes, on the other hand, can be divided into subparts (that is, other attributes). For example, an attribute *name* could be structured as a composite attribute consisting of *first-name*, *middle-initial*, and *last-name*. Using composite attributes in a design schema is a good choice if a user will wish to refer to an entire attribute on some occasions and to only a component of the attribute on other occasions. Suppose we were to substitute for the *customer* entity-set attributes

Customer-street and *customer-city* the composite attribute *address* with the attributes Street, *city*, *state*, and *zip-code*. Composite attributes help us to group together related attributes, making the modeling cleaner.

Note also that a composite attribute may appear as a hierarchy. In the composite attribute *address*, its component attribute *street* can be further divided into *street-number*, *street-name*, and *apartment-number*.
Figure depicts these examples of composite attributes for the *customer* entity set.



Composite attributes *customer-name* and *customer-address*.

Single-valued and multivalued attributes

The attributes in our examples all have a single value for a particular entity. For instance, the *loan-number* attribute for a specific loan entity refers to only one loan number. Such attributes are said to be **single valued**. There may be instances where an attribute has a set of values for a specific entity considers an *employee* entity set with the attribute *phone-number*. An employee may have zero, one, or several phone numbers, and different employees may have different numbers of phones.

This type of attribute is said to be **multivalued**. As another example, an at tribute *dependent-name* of the employee entity set would be multivalued, since any particular employee may have zero, one, or more dependent(s).

Where appropriate, upper and lower bounds may be placed on the number of values in a multivalued attribute. For example, a bank may limit the number of phone numbers recorded for a single customer to two. Placing bounds in this case expresses that the *phone-number* attribute of the *customer* entity set may have between zero and two values.

Derived attribute

The value for this type of attribute can be derived from the values of other related attributes or entities. For instance, let us say that the *customer* entity set has an attribute *loans-held*, which represents how many loans a customer has from the bank. We can derive the value for this attribute by counting the number of *loan* entities associated with that customer.

As another example, suppose that the *customer* entity set has an attribute *age*, which indicates the customer's age. If the *customer* entity set also has an attribute *date-of-birth*, we can calculate *age* from *date-of-birth* and the current date (*system date*).

Thus, *age* is a derived attribute. In this case, *date-of-birth* may be referred to as a *base* attribute, or a *stored* attribute. The value of a derived attribute is not stored, but is computed when required.

Null Value

An attribute takes a **null** value when an entity does not have a value for it. The *null* value may indicate "not applicable"—that is, that the value does not exist for the

entity. For *example*, one may have no middle name. *Null* can also designate that an attribute value is unknown. An unknown value may be either *missing* (the value does exist, but we do not have that information) or *not known* (we do not know whether or not the value actually exists).

For instance, if the *name* value for a particular customer is *null*, we assume that the value is missing, since every customer must have a name. A null value for the *apartment-number* attribute could mean that the address does not include an apartment number (not applicable), that an apartment number exists but we do not know what it is (missing), or that we do not know whether or not an apartment number is part of the customer's address (unknown).

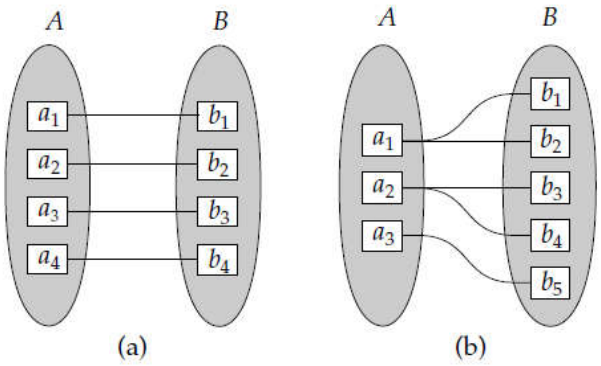
Mapping Cardinalities

Mapping cardinalities, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set.

Mapping cardinalities are most useful in describing binary relationship sets, although they can contribute to the description of relationship sets that involve more than two entity sets. In this section, we shall concentrate on only binary relationship sets.

For a binary relationship set R between entity sets A and B, the mapping cardinality must be one of the following:

- **One to one**. An entity in A is associated with *at most* one entity in B, and an entity in B is associated with *at most* one entity in A.
- **One to many**. An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with *at most* one entity in A.
- **Many to one**. An entity in A is associated with *at most* one entity in B. An entity in B, however, can be associated with any number (zero or more) of entities in A.
- **Many to many**. An entity in A is associated with any number (zero or more) of entities in B, and an entity in B is associated with any number (zero or more) of entities in A.



Mapping cardinalities. (a) One to one. (b) One to many.

The appropriate mapping cardinality for a particular relationship set obviously depends on the real-world situation that the relationship set is modeling.

As an illustration, consider the *borrower* relationship set. If, in a particular bank, a loan can belong to only one customer, and a customer can have several loans, then the relationship set from *customer* to *loan* is one to many. If a loan can belong to several customers (as can loans taken jointly by several business partners), the relationship set is many to many.

Keys

A *key* allows us to identify a set of attributes that suffice to distinguish entities from each other. Keys also help uniquely identify relationships, and thus distinguish relationships from each other.

Super key

A **super key** is a set of one or more attributes that, taken collectively, allow us to identify uniquely an entity in the entity set. For example, the *customer-id* attribute of the entity set *customer* is sufficient to distinguish one *customer* entity from another. Thus, *customer-id* is a superkey. Similarly, the combination of *customer-name* and *customer-id* is a superkey for the entity set *customer*. The *customer-name* attribute of *customer* is not a superkey, because several people might have the same name.

Entity-Relationship Diagram

As we saw briefly, an **E-R diagram** can express the overall logical structure of a database graphically. E-R diagrams are simple and clear—qualities that may well

account in large part for the widespread use of the E-R model. Such a diagram consists of the following major components.

- **Rectangles**, which represent entity sets
- **Ellipses**, which represent attributes
- **Diamonds**, which represent relationship sets
- **Lines**, which link attributes to entity sets and entity sets to relationship sets
- **Double ellipses**, which represent multivalued attributes
- **Dashed ellipses**, which denote derived attributes
- **Double lines**, which indicate total participation of an entity in a relationship set
- **Double rectangles**, which represent weak entity sets

Consider the entity-relationship diagram in Figure, which consists of two entity sets, *customer* and *loan*, related through a binary relationship set *borrower*. The attributes associated with *customer* are *customer-id*, *customer-name*, *customer-street*, and *customer-city*. The attributes associated with *loan* are *loan-number* and *amount*. In Figure, attributes of an entity set that are members of the primary key are underlined.

The relationship set *borrower* may be many-to-many, one-to-many, many-to-one, or one-to-one. To distinguish among these types, we draw either a directed line (→) or an undirected line (—) between the relationship set and the entity set in question.

• A directed line from the relationship set *borrower* to the entity set *loan* specifies that *borrower* is either a one-to-one or many-to-one relationship set, from *customer* to *loan*; *borrower* cannot be a many-to-many or a one-to-many relationship set from *customer* to *loan*.

Weak Entity Sets

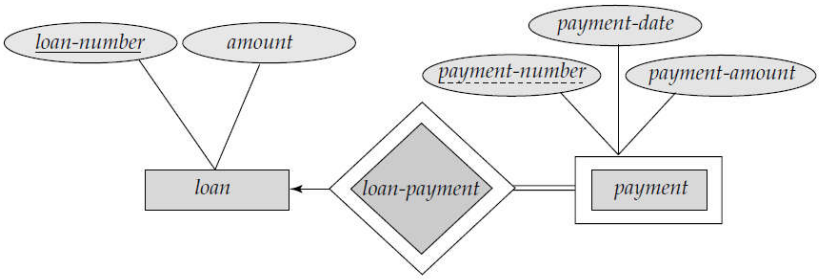
An entity set may not have sufficient attributes to form a primary key. Such an entity set is termed a **weak entity set**. An entity set that has a primary key is termed a **strong entity set**.

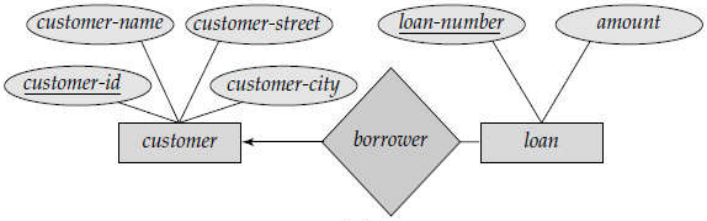
As an illustration, consider the entity set *payment*, which has the three attributes: *payment-number*, *payment-date*, and *payment-amount*. Payment numbers are typically sequential numbers, starting from 1, generated separately for each loan. Thus, although each *payment* entity is distinct, payments for different loans may share

the same payment number. Thus, this entity set does not have a *primary key*; it is a weak entity set.

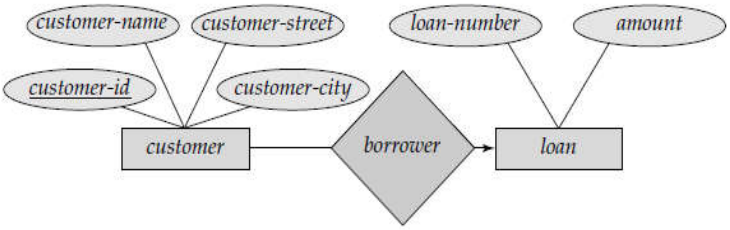
For a weak entity set to be meaningful, it must be associated with another entity set, called the *identifying* or *owner entity set*. Every weak entity must be associated with an *identifying entity*; that is, the weak entity set is said to be *existence dependent* on the *identifying entity set*.

In our example, the identifying entity set for ***payment*** is ***loan***, and a relationship *loan-payment* that associates *payment* entities with their corresponding *loan* entities is the identifying relationship.

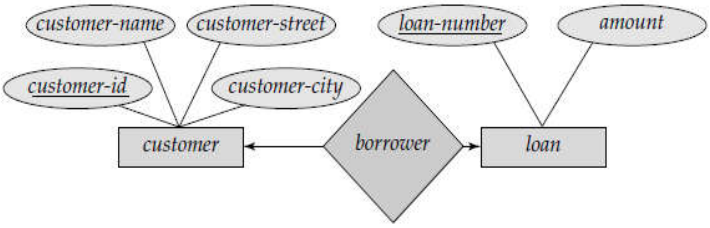




(a)

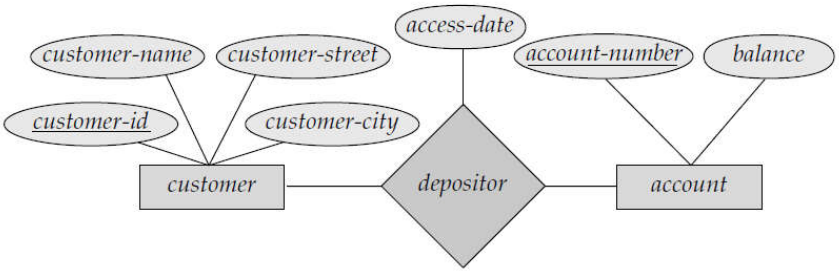


(b)

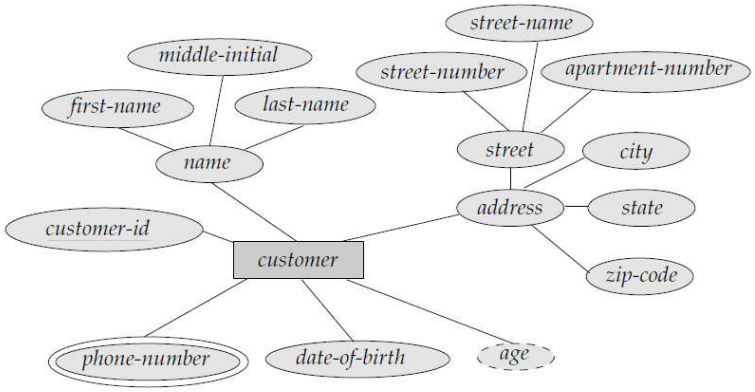


(c)

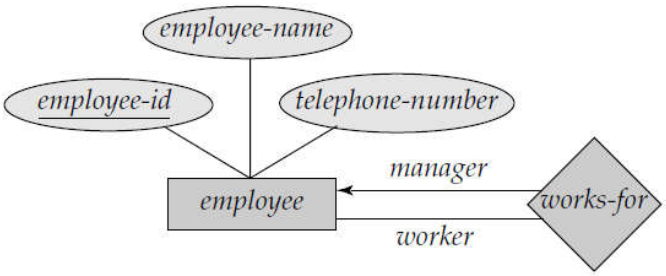
Relationships. (a) one to many. (b) many to one. (c) one-to-one.



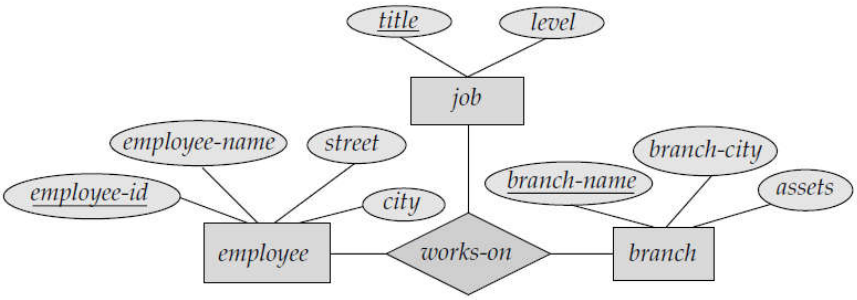
E-R diagram with an attribute attached to a relationship set.



E-R diagram with composite, multivalued, and derived attributes.



E-R diagram with role indicators.



E-R diagram with a ternary relationship.