Entity Relationships and Databases

The following is excerpted from Chapter 6, “Data Modeling,” in *Business Systems Analysis and Design* by William S. Davis (1994, Belmont, CA: Wadsworth Publishing Company), a text used at one time in INLS 162, Systems Analysis.

Entities and Relationships

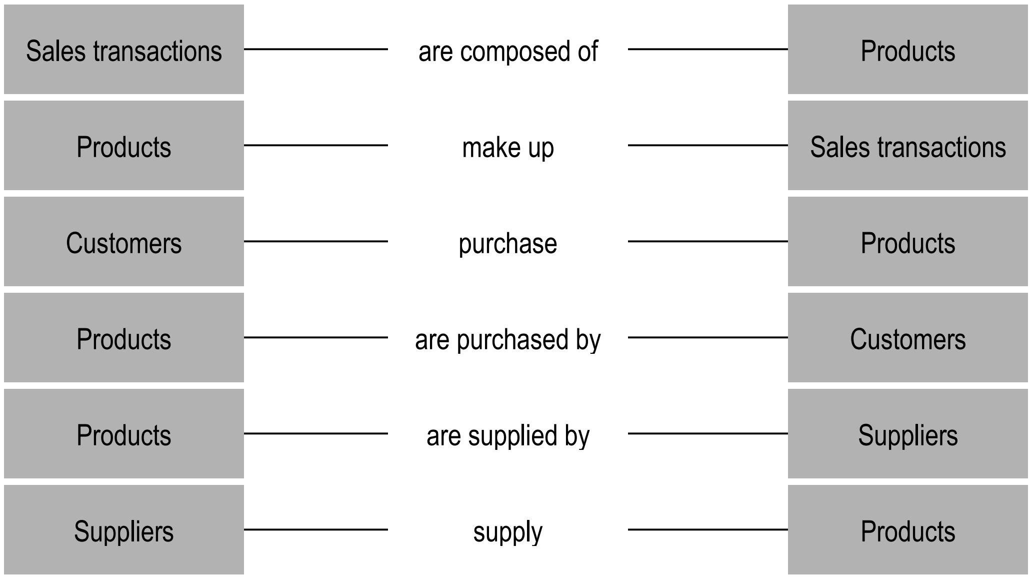
Entity-relationship diagrams were first proposed as a means of quickly obtaining, with minimum effort, a good sense of the structure of a database. Consequently, several key terms are taken from database theory.

An entity is an object (a person, group, place, thing, or an activity) about which data is stored. An occurrence is a single instance of an entity; for example, a particular 19-inch color TV set is a single occurrence of the entity called *Inventory*. An attribute is a property of an entity; for example, such attributes as stock number, description, and stock on hand might be associated with *Inventory*. Generally, the same set of attributes is associated with each occurrence of an entity, so every part in *Inventory* can be expected to have a stock number, a description, and a stock on hand. The set of attributes associated with an entity can be visualized as a table or a record.

A data element is an attribute that cannot be logically decomposed. A set of related data elements forms a data structure or a data composite; for example, the set of attributes associated with each occurrence of an entity is a data structure. The key to an entity is the attribute or group of attributes that uniquely distinguishes one occurrence from all other occurrences. A foreign key is a key to some other entity; for example, a *Supplier* code might be associated with the *Inventory* data.

A relationship links two entities and is shown by drawing a line between them as shown in figure 1.

figure 1



Logically, the relationship can be stated in the form of a sentence with a verb linking the two entities; for example, *Sales transactions* are composed of *Products* or *Products* make up *Sales transactions*.

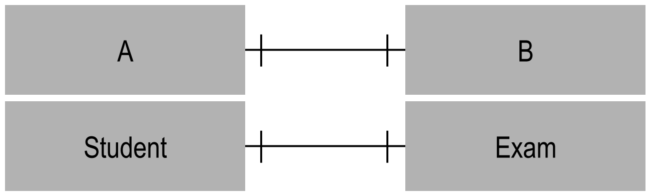
The act of creating such sentences is a good test of the relationship's validity; if you can't express the link, it might not exist. In most cases where the relationship is unclear, the sentence might be written alongside the relationship line as shown in figure 1.

Cardinality

For a variety of reasons, some relationships are more stable and easier to maintain than others. Cardinality, a measure of the related entity's relative number of occurrences, is an important predictor of the strength of the relationship.

In a **one-to-one** relationship (figure 2), each occurrence of entity A is associated with one and only one occurrence of entity B, and each occurrence of entity B is associated with one and only one occurrence of entity A. For example, imagine an instructor who maintaining examination data on each student.

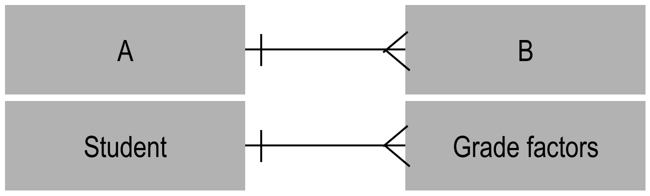
figure 2



There are two entities: *Students* and *Exams*. For each *Student*, there is one and only one *Exam*, and for each *Exam*, there is one and only one *Student*. Graphically, a **one-to-one** relationship is described by drawing short crossing lines at both ends of the line that links the two entities.

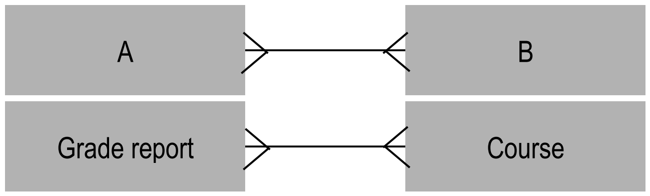
In a **one-to-many** relationship (figure 3), each occurrence of entity A is associated with one or more occurrences of entity B, but each occurrence of entity B is associate with only one occurrence of entity A. For example, your grade in most courses is based on numerous grade factors (exams, papers, projects). A given *Student* has several *Grade factors*, but a given *Grade factor* is associated with one and only one *Student*. Graphically, a **one-to-many** relationship is shown by drawing a short crossing line at the "one-end" and a small triangle (sometimes called a crow's foot) at the "many-end" of the line that links the entities.

figure 3



In a **many-to-many** relationship (figure 4), each occurrence of entity A is associated with one or more occurrences of entity B, and each occurrence of entity B is associated with one or more occurrences of entity A. For example, your end-of-term *Grade report* can list several *Courses*, and a given *Course* can appear on many students' *Grade reports*. Graphically, a **many-to-many** relationship is shown by drawing a crow's foot at both ends of the line that links the entities.

figure 4



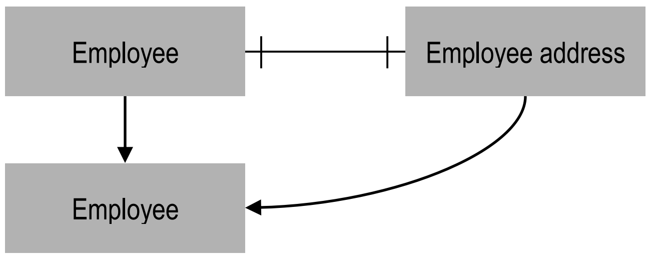
Other types of relationships are possible, but we will concentrate on **one-to-one**, **one-to-many**, and **many-to-many** relationships.

Analyzing Relationships

One-to-many relationships tend to be the most stable. Consequently, a primary objective of entity-relationship modeling is to convert **one-to-one** and **many-to-many** relationships into **one-to-many** relationships.

One-to-one *Relationships*

figure 5



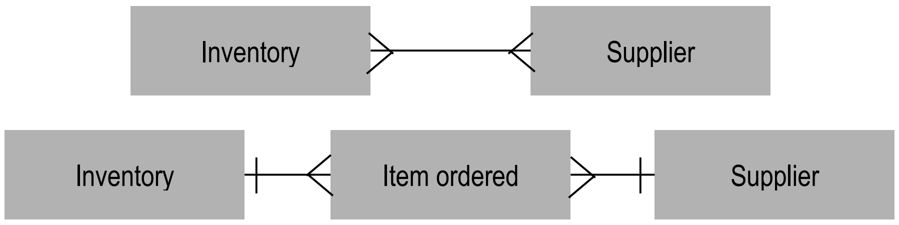
One-to-one relationships can often be collapsed by merging them. For example, figure 5 shows a **one-to-one** relationship linking a set of employee data with that employee's address. You might be able to imagine an application in which it makes sense to keep the home address separate from the other employee data, but it seems reasonable to treat *Address* as an attribute of *Employee*. Generally, entities that share a **one-to-one** relationship should be merged unless there is a good reason to keep them separate.

Note that not all **one-to-one** relationships can be collapses. For example, imagine a relationship between *Athlete*s and *Drug test*s. There is one *Drug test* per *Athlete* and one *Athlete* per *Drug test*, so the relationship is clearly **one-to-one**. You might argue that *Drug test* is an attribute of *Athlete*, but what if the law requires that *Drug test* data be kept confidential? Because merging the data would make it relatively easy to link a specific person to a specific test result, merging them would probably violate the law, so there is a good logical reason to maintain separate entities.

Many-to Many Relationships

Many-to many relationships can cause maintenance problems. For example, figure 6 shows a **many-to-many** relationship between *Inventory* and *Supplier*. Each product in *Inventory* can have more than one *Supplier*, and each *Supplier* can carry more than one product. If you were to store a list of *Supplier*s in *Inventory*, adding or deleting a *Supplier* might mean updating several *Inventory* occurrences. Likewise, listing products in *Supplier* could mean changing several *Supplier* occurrences if a single product were added or deleted.

figure 6



One solution is to create a new entity that has a **one-to-many** relation ship with *both* original entities. For example, imagine a new entity called *Item ordered* (figure 6). If you need 100 television sets, you might order 50 from *Supplier* A, 25 from *Supplier* B, and 25 from *Supplier* C, so a given product in *Inventory* can appear on several active *Item ordered*. However, each *Item ordered* is for one and only one product. Likewise, a given *Supplier* can appear on several active *Item ordered*, but each *Item ordered* lists one and only one *Supplier*. Note that a give *Item ordered* links a specific product in *Inventory* with a specific occurrence of *Supplier*. The **many-to-many** relationship has been converted to two **one-to-many** relationships.

How does that affect data maintenance? Imagine that *Inventory*, *Item ordered*, and *Supplier* are three files. The file called *Item ordered* holds product codes and *Supplier* codes, and it is indexed on both fields. Detailed information on products and/or *Supplier*s can be obtained by accessing the *Inventory* file or the *Supplier* file, as appropriate. Dropping a product affects *Inventory* and *Item ordered*, but not *Supplier*. Adding a *Supplier* affects *Supplier* and *Item ordered*, but not *Inventory*. Because *Item ordered* is indexed on both keys, it is easy to maintain.

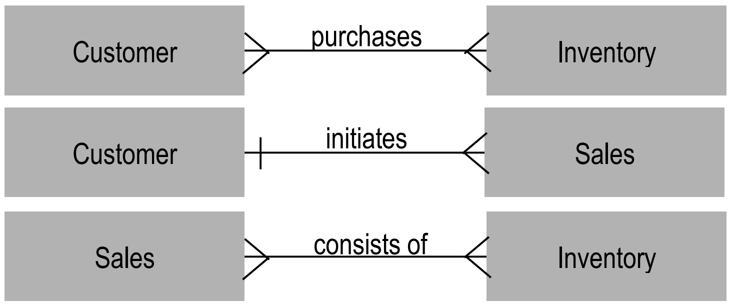
Creating an Entity-Relationship Diagram

Sales Data

Assuming we are analyzing a system, we have some things that we can see exist. If the system is a business of some sort, one can assume that we have three entities: *Sales*, *Inventory*, and *Supplier*. First, why are they entities?

Start with the definition of an entity: an object (a person, group, place, thing, or activity) about which data are stored. Since any business must involve a *Customer*, a *Customer* is an object and *Customer* is probably an entity. Other data elements (*Stock number*, *Description*, and *Unit price*) describe a product in *Inventory*, so *Inventory* must be another entity and *Customer* purchase *Inventory* defines their relationship (figure 7). Since one *Customer* can purchase many products and a given product can be purchased by many *Customer*s, the relationship is **many-to-many**.

figure 7

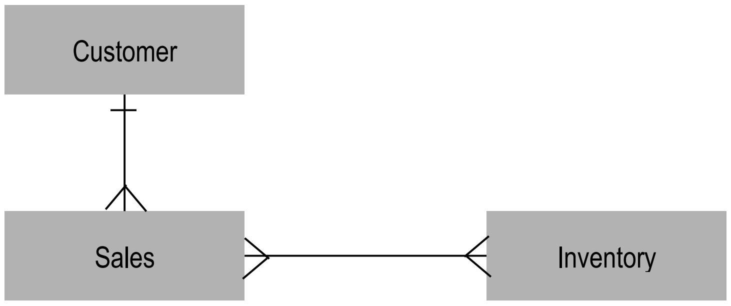


Other data elements we might encounter could include *Invoice number*, *Date of sale*, *Quantity*, *Item total*, *Subtotal*, *Sales tax*, and *Total due* are clearly not attributes of *Customer* or *Inventory*. Instead, they describe the sale itself, so *Sales* must be another entity. The *Sales* entity is related to both *Customer* and *Inventory*. A quick review of the other sale-related documents should convince you that most of their fields either describe a product in *Inventory*, are derived from *Sales*, or are computed upon demand, so the existing data suggests no additional entities.

As figure 7 shows, the *Sales* entity is related to the other two as follows: *Customer* initiates *Sales*; *Sales* consist of *Inventory*

The first relationship is **one-to-many** (figure 8). A given *Customer* can have many *Sales* transactions, but a given Sale is associated with one and only one *Customer*. The second relationship is **many-to-many** because a given sale can include several products from *Inventory* and a given product in *Inventory* can appear in many *Sales*.

figure 8



To resolve that **many-to-many** relationship, create a new entity, *Item sold*, that has a **one-to-many** relationship with both *Sales* and *Inventory* (figure 8). A given *Sales* transaction can list many *Item sold*, but a given *Item sold* is associated with one and only one *Sales* transaction. A given product in *Inventory* can appear in many *Item sold*, but a given *Item sold* lists one and only one product. (Think of an *Item sold* as one line in a list of products purchased on a *Sales* invoice.)

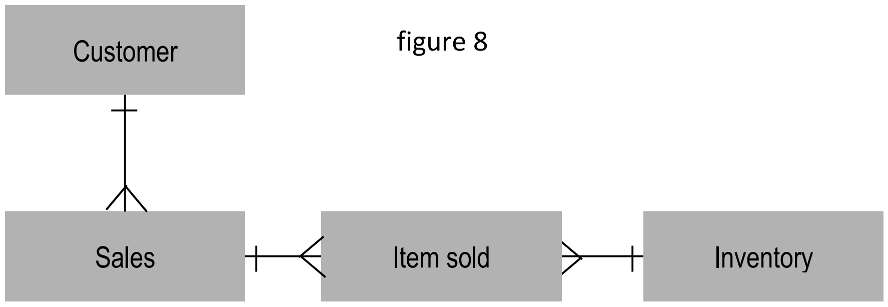


figure 8

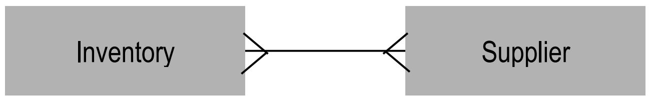
Note that the set of relationships described in figure 8 also resolves the **many-to-many** relationship between *Customer* and *Inventory*. Those two entities are related through *Sales* and *Item sold*, and each of the intermediate relationships is **one-to-many**.

There is one possible source of confusion about the *Inventory* entity that might need clarification. A specific 19-inch color TV set is an example of a single occurrence of that entity, but *Inventory* might hold 100 or more virtually identical TV sets. For *Inventory* control purposes, tracking TV sets (a *class* of occurrences) is probably good enough. However, a *Customer* purchases a specific TV set (identified, perhaps, by concatenating the serial number to the stock number). Thus a given *Item sold* lists one and only one occurrence of *Inventory*.

Supplier and Inventory Data

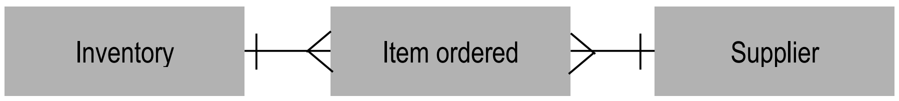
But there seems to also be a **many-to-many** relationship between *Inventory* and *Supplier* (figure 9). It is **many-to-many** because a given product can have many *Supplier*s and a given *Supplier* can supply many products.

figure 9



Many-to-many relationships must be resolved, so add a new entity called *Item ordered* to the model (figure 10). You now have two **one-to-many** relationships. Check the relationships in figure 10 and convince yourself that they really are **one-to-many**.

figure 10

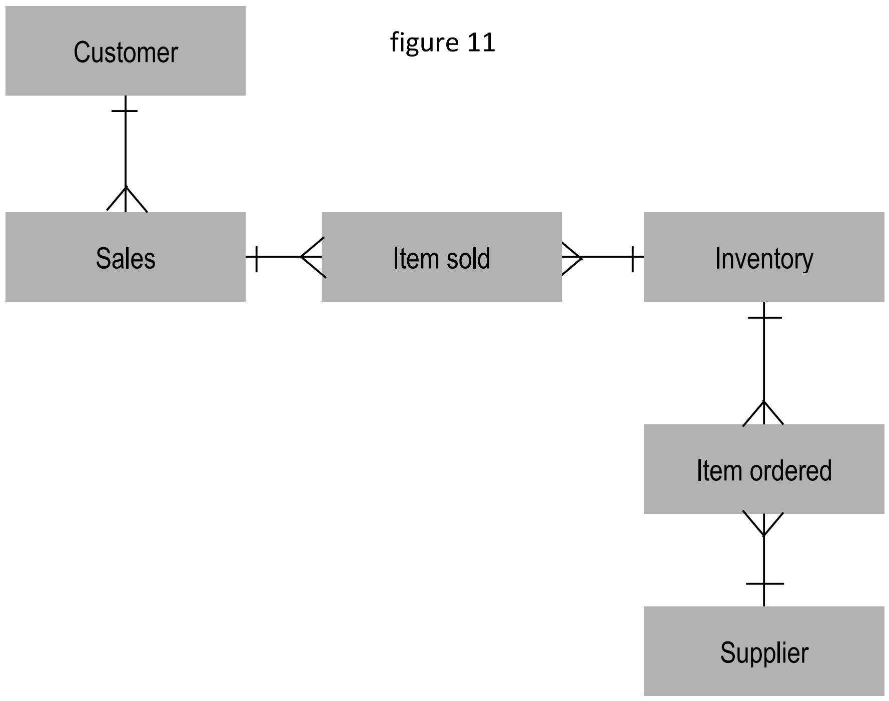


The entity called *Item ordered* will hold foreign keys that link a product to a *Supplier* and a *Supplier* to a product.

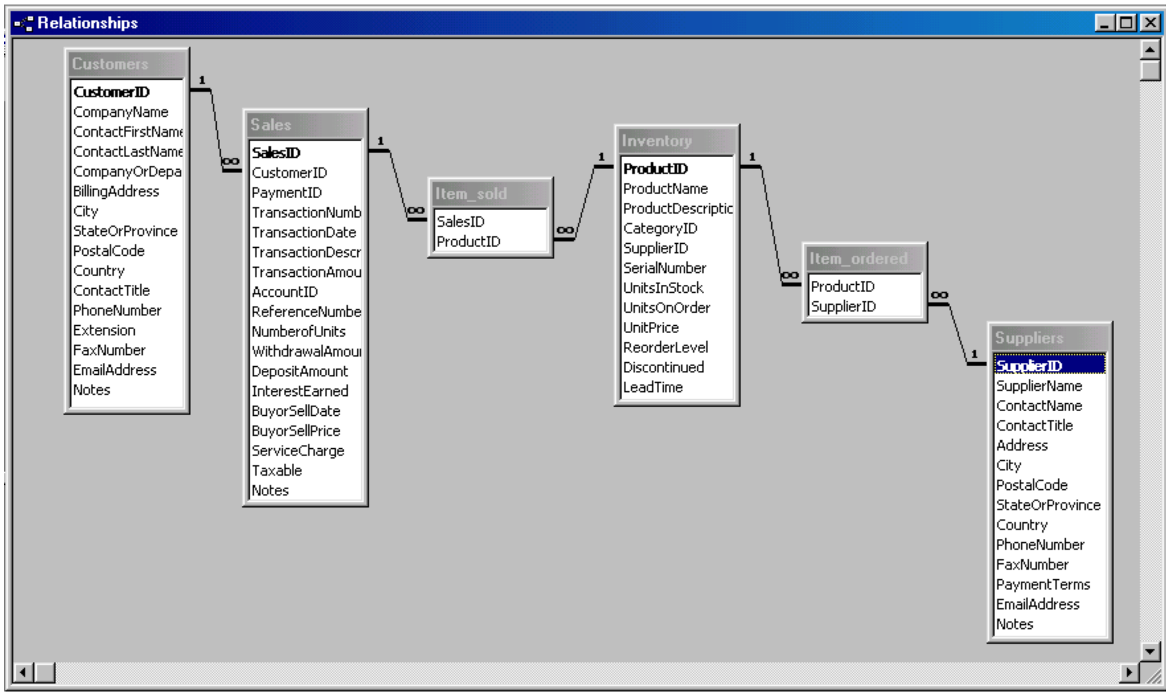
Completing the Model

The *Inventory* entity is related to both *Item sold* and *Item ordered*, so you can combine the two partial diagrams to form a single entity-relationship model (figure 11).

figure 11



Translated into Access terms, the relationship could look like the following, which was entirely created using the Table Wizard. Note that in the creation of the Customer/Sales relationship, the Wizard placed the Primary Key from Customers into Sales as a Foreign Key. However, in the construction of the Item_sold and Item_ordered tables, the Primary Keys from the related tables had to be placed by the Wizard as foreign keys into the connecting tables during the definition of the fields in these two tables.


Relationships
Customers
CustomerID
CompanyName
ContactFirstName
ContactLastName
CompanyOrDepa
BillingAddress
City
StateOrProvince
PostalCode
Country
ContactTitle
PhoneNumber
Extension
FaxNumber
EmailAddress
Notes
1
∞
Sales
SalesID
CustomerID
PaymentID
TransactionNumb
TransactionDate
TransactionDescr
TransactionAmou
AccountID
ReferenceNumbe
NumberofUnits
WithdrawalAmou
DepositAmount
InterestEarned
BuyorSellDate
BuyorSellPrice
ServiceCharge
Taxable
Notes
1
∞
Item_sold
SalesID
ProductID
∞
1
Inventory
ProductID
ProductName
ProductDescriptic
CategoryID
SupplierID
SerialNumber
UnitsInStock
UnitsOnOrder
UnitPrice
ReorderLevel
Discontinued
LeadTime
1
∞
Item_ordered
ProductID
SupplierID
∞
1
Suppliers
SupplierID
SupplierName
ContactName
ContactTitle
Address
City
PostalCode
StateOrProvince
Country
PhoneNumber
FaxNumber
PaymentTerms
EmailAddress
Notes