## Data Modeling (Entity Relationship Diagrams)

To create a relational database, the first step is to specify the requirements. Then the requirements can be used to create multiple tables. After the database has been created, Standard Query Language (SQL) can be used to gain access to the data across the tables. This lecture belongs to the first step, i.e. abstracting business processes into database requirements.

## Entities

When system requirements are specified, the decision makers specify a set of variables that they are interested in. From these specification emerges a list of variables/fields. To specify a database, we need to partition the variables of interest into multiple tables. We can decide about the tables in a database by divide the list of fields into entities. Then each entity will represent a table and each field in the table is an attribute of the entity. According to Merriam Webster Dictionary an entity is something that has an independent, separate, or self-contained existence. It could be just about anything: a person, a disease, a time frame. It can have an objective or conceptual reality. It can be something alive or dead. It can be a large concept such as organizations or a small concept such as a purchase. In this lecture, we learn to divide system requirements into an underlying set of entities.

An entity can be thought of as a class of data. Each entity has a name, a definition, a type. In addition, each entity has a set of attributes that describe the various characteristics of the entity. Each attribute also has a name, a definition, a type and constraints. The attribute types are text, numeric, binary and date types. Field and attributes are different name for the same thing. Entity and table are different name for the same thing. In the context of relationship diagrams, the words entity and attributes are used. In the context of physical database work, the words table and field are used.

A list of fields is divided into entities by examining common features of the fields. Each field is considered an attribute or characteristic of an entity. The database designer thinks through a set of attributes that describe a particular entity. For example, the three fields "Patient's first name," "Patient's last name" and "Patient's birthday" seem to suggest an entity called "Patient." They are about the patient. The two fields "Type of diagnosis" and "Name of diagnosis" seem to be about diagnosis and therefore they suggest an entity called "Diagnosis." Review the entire list of fields and divide it into as many entities that makes sense to you. Keep in mind that a list of entities should also make sense to others and therefore entities should be named in self apparent ways. An entity called "Patient" is about patients and an entity called "Diagnosis" is about diagnoses. The names imply what the entity is about.

Hernandez in his book Database Designers for Mere Mortals (pages 193-198) suggests the following rules for naming an entity:

1. "Create a name that accurately, clearly and unambiguously identifies the subject of a table.
2. Use the minimum number of words necessary to convey the subject of the table.

3. Do not use words that convey physical characteristics of the database. Avoid using words such as 'File', 'Record,' and 'Table.'
4. Do not use acronyms and abbreviations.
5. Do not use proper names or other words that will unduly restrict the data that can be entered into the table.
6. Do not use a name that implicitly or explicitly identifies more than one concept (e.g. facility or branch entity).
7. Do not use the plural form of the name.

Always include a brief description of the entity. The description should tell the definition of the entity and say why it is important to track this entity. The definition should be sufficiently clear to help the reader set up expectations about what types of fields might be included under the entity. For example, the entity "Patient" may be defined as: "The clients in the court diversion program who have mental illness." The description may continue with stating why it is important to track this information: "Information about the patients need to be kept in order to track if their court-ordered treatment is working and has been followed."

The process of dividing fields into entities is difficult. In some ways there is no right or wrong set of entities. Two database designers may arrive at different set of entities for a health care business and they may be both right. Design, after all, is an art form and not an exact science. But at the same time there are a number of rules that should be followed that would make the database more useful and more efficient. Some of these rules are the following:

1. No two entities should be about the same things. For example, calling one entity "Patient" and another "Client" may create both confusion and inefficiency. The inefficiency is obvious as same information has to be entered both places. But this is not the worst of the problems. Even more important than inefficiency is confusion in where to look for data. Users of the database will be confused because they would not know if data on patients will be under the table called "Patient" or under the table called "Client."
2. Do not assign the same field in two different entities. This too will create inefficiency and confusion. Take for example, the patient's name. We may correctly decide that this information belongs to the entity "Patient." We may also erroneously assume that the patient's name should also be kept in the entity "Diagnosis." After all, patients have diagnoses. The question in your mind should always be where would future user's of the database look for the name of the patient, in the table "Patient" or in the table "Diagnosis." Certainly these entities are related but that is not how we decide where to put various fields. By putting it in two places, we are just creating confusion of where information is kept. If not sure where fields belong, always choose the place that is most commonly thought of as the place for the data. Meet and satisfy the intuitions of future users of the database.
3. Make sure that time based events are separated from time independent fields. For example, a patient's diagnoses may seem at first glance to belong to the entity "Patients." But all fields in the "Patients" entity seem to be time invariant: first name is not likely to change very often. But a patients' diagnosis changes every visit. Putting these two fields in the same entity will create a problem later when frequent changes needs to be made to

diagnoses but not first name. Instead create a separate entity, perhaps "Visits." Visits will be time based and each visit may contain one or more diagnoses. The field "Patient's diagnosis" needs to reside with an entity that is clearly time dependent, like "Visits."

4. If several entities share the same fields, see if you can partition the shared fields into a separate entity. For example, in the design of a database for a mental health court, the mental health employees, the court staff, the patient, the lawyer and the judge all have a series of fields called "First name," "Last name," "Middle initial," "Birthday" and "City of Birth". One way to make the design more efficient is to create a new entity called "Person" and keep these shared fields in it. Then if an employee falls ill or is seen in the court, their name does not need to be entered several times once as an employee, next as a patient and last as a court client. The fields are entered into the table "Person" and the role this person is playing may be entered into other tables.
5. Include look-up entities as you proceed. In a number of situations the possible range of a field is well defined ahead of time. For example, the field "Observed gender" in the entity "Patient" may have three possibilities: male, female, and unknown. These fixed responses are entered into a separate entity called "Gender" with two fields: code and code description. Then the table "Gender" is considered a look up table for entering the values in the field "Observed Gender" in the table "Patient." Clearly in any database numerous look up tables may exist in order to simplify entering values for various fields. It is important to keep track of these look up tables.

Once you have settled on a list of entities, review the fields within these entities for completeness. Many entities suggest new fields that have not been thought about. For example, a characteristic of a patient is his age. This suggests that the field "Birthday" should be added to the entity "Patient," if it is not already there. Once a complete set of fields have been identified review them again to see if they fit the list of entities. Share the list with organizational members, to see if the naming makes sense to them and if all relevant concepts have been captured. Get as much input as you go along designing the database.

End the process with clear documentation. Make sure that each entity has a name, a description, a statement of why it is important to track the entity, and a list of fields that belong to it.

## Entity Relationships

The last step in creating entity relationship diagrams is the specification of the relationships among the entities. Just as every object in the real world has some kind of relationship to one or more objects so too the entities in a database are related to other entities. The nature of relationships between entities is usually implied in the very definition of the entity. Despite the obviousness of these relationships, it is important to review all entities and specify how they relate to each other. There are at least three types of relationships possible:

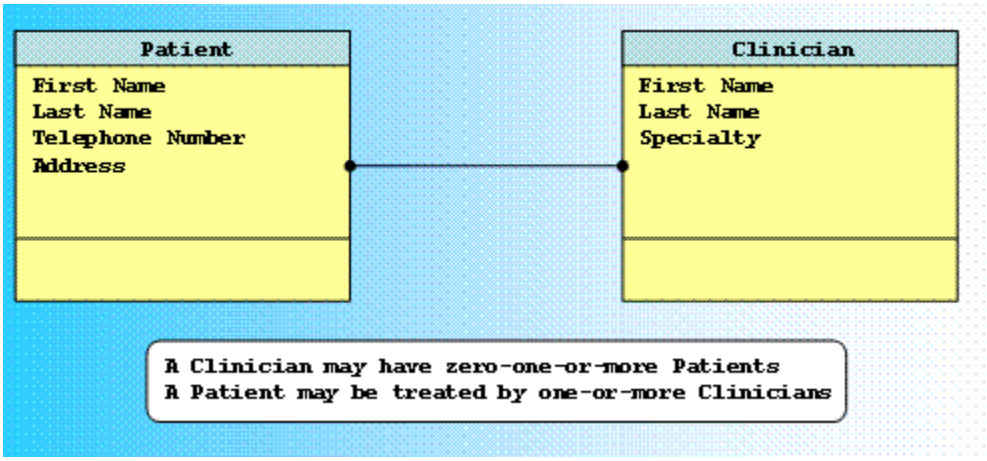
1. One-to-one, where one entity corresponds to exactly another entity. For example, a table about patient's death has a one-to-one relationship with the table "Person."
2. One-to-many, where one instance of one entity can be repeatedly used by another. For example the look up table Gender may be repeatedly used in the table "Patient."

3. Many-to-many where one instance of both entities can be repeatedly used by another. For example, tables "Patient" and "Clinician" have many-to-many relationships as a clinician may have many patient and a patient may have many clinicians.

Sometimes, the relationship between two entities is not clear. The most common cause is that a third entity is missing. This often occurs when two entity have many-to-many relationship. For example, the entity Patient and the entity Clinician have, as mentioned earlier, many-to-many relationship. It is difficult to show these relationships inside a database in a way that can easily be manipulated. An alternative is to show a new table that links these two tables to each other and has one-to-many relationship to each of the tables. For example, we can make a new table called Visit. Within a visit a patients is diagnosed. Both the patients and the clinicians identity are kept in the visit table. The Visit table has one-to-many relationship with either patient or clinician table. Sometimes, as we specify the relationships among entities, a new entity must be defined.

Linkages between entities are part of the business rules that databases should capture. In our example, the business rule for the linkage between a Clinician and a Patient is that a clinician may have zero, one, or, more patients. The business rule for the linkage between the Patient and the Clinician is that a patient may have one, or, more clinicians. Note that these are the business rules that someone may have specified. In a different information system someone could decide that a patient can only have one clinician at a time, or that the number of clinicians dealing with a patient must always be 3, or some other similar rule. The important point is that entities can be linked to each other, and that the nature of the linkage is part of the business rules of the system.

To make tracking of information simpler, many modeling languages have standardized how entities and relationships are shown. A common approach is to show entities as boxes with their names as their labels. Inside the entity box the fields are listed. In the Figure below two entities are shown: the Patient and the Clinician entities.




The figure also shows that the two entities are related to each other. The graphical representation of data linkages depends on the modeling language one uses. In IDEF1X, a modeling language, one can represent many-to-many relationships by using a line with a large black dot at both

ends.  This icon means that there is a many-to-many linkage between the entities connected.  If there is a one-to-many relationship the large dot is out in the side of entity with many instances.

In Access database, the line shows the relationship between the two tables and the shared field shows the nature of the relationship. The arrow shows if the relationship is one-to-many, with the many side shown by the direction of the arrow.

As with the specification of the entities discussed at the beginning of this lecture, the documentation of the relationships is part of the logical information model.  The format for documenting the linkages among entities includes the name of both entities, the verb phrase that describes the semantics of the linkage and the cardinality of the linkage (i.e. whether one-to-one, one-to-many or many-to-many).  The statement of the cardinality can be made in plain English.  All relationships must be documented before proceeding to the physical design of the database.

## Summary

In this lecture we have seen how one can take an information flow and conceptualize one or more entities out of it.  Data entities have a name, a type, a definition and attributes.  Any change in the system requirements must be faithfully and completely captured by the corresponding changes in the entity relationship diagram.  This may be either by adding or refining the definition of entities or attributes.  We reviewed the rules for how entities can be found from a list of fields/attributes.