ER Modeling

There are four primary steps to designing a database:

1. As with any design problem, the first step is always to understand the problem and analyze the requirements.

2. In a database design problem, the next step is to design a high-level conceptual model (using an ER diagram).

3. Next, the conceptual model is translated into a logical schema. It should be noted that a specific ER diagram may have multiple translations depending upon the design decisions that are made in order to satisfy the requirements. We will discuss this point again at a later time.

4. The final step is to verify the design with the actual users in order to correct any mistakes (misunderstandings) prior to implementation.

NOTE: The process is iterative. In other words, upon completing step 4, you may discover that you have to make modifications to the conceptual model and subsequently make changes to the logical schema as well.

Each of the four steps is further discussed in the following sections.

You cannot design a database without first taking the time to understand the problem and the requirements. In order to make sure that you really understand the problem, write a summary (in narrative form) of the problem that you are trying to solve and the requirements that the user needs/wants as part of the solution. The summary should identify the entities, their attributes and the relationships that exist between the entities.

You should also identify the queries and reports that will need to be generated.

Another benefit of writing a summary of the problem and requirements is that you can use it to:

1. Make sure that you understand the requirements.
2. Communicate the scope of the project with your users.
3. Check your design.

Understanding the requirements is essential. Without understanding the requirements a developer runs the risk of creating features and functions that will not meet the end users expectations. Meeting the user's expectations is primary concern is system development If a system does not meet user expectations, it may be considered a failure. Communicating the scope of the project is equally important. Scope defines how far reaching the project will be. There are many instances where financial resource or schedule does not permit one to include all features and functions in the scope. When this occurs, requirement management must be used. This is moving or eliminating requirements in the project. Once you have a final summary of the requirements, you will use the summary to create a conceptual schema (i.e. an ER diagram).

Another important process in database design is normalization. We will cover normalization forms in another lecture

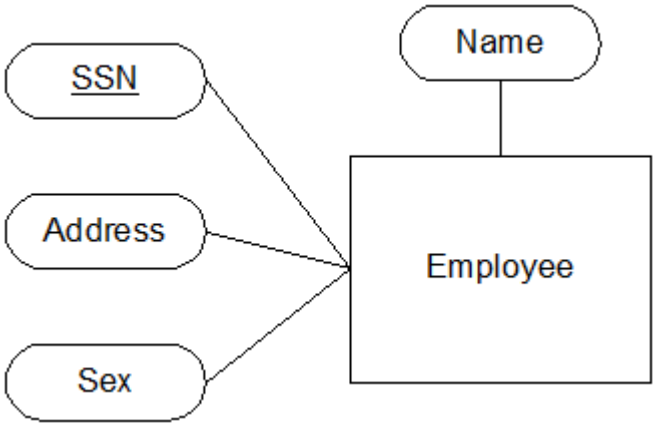
Database design has several phases: some are done in sequence while others are done in parallel. For example, **HCI (Human Computer Interaction)** Analysis and the Functional Analysis are done in parallel with the Conceptual, Logical and Physical database design.

HCI Analysis: GUI (Graphical User Interface) design, designing the screens that the users will use to access the database

Functional Analysis: designing any programs that will be required to interact with the database (e.g. external programs, triggers, stored procedures, etc.)

We will focus on the Requirements Analysis, Conceptual, Logical and Physical design.

When drawing the entity on the ER diagram, the entity is represented as a rectangle with the name of the entity located inside of the rectangle. The attributes are represented as an oval with the name of the entity located inside of the oval and a line connecting the oval to the entity. The name of the primary key attribute is also underlined. The Diagram Below illustrates this. Employee is the entity. Name, SSN, Address and Sex are the attributes being declared in the entity. Each attribute is connected to the entity by a line. The primary key, in this instance SSN is underlined.



Entity

- A "thing"; an object
- People, places, tangible physical thing, organizations, events, concepts
- Naming convention → a singular noun is used instead of plural form for better clarity
- Key Attribute → an attribute that uniquely identifies each instance of an entity
- Descriptive Attributes → any non-key attribute connected to the entity

Each entity has attributes. Each attribute has a specific value and a domain. A domain is the field type and set of allowable values. In the example below, the SSN attribute for the Employee entity can only contain a numeric value. The Age attribute for the Employee entity can only contain values between 18 and 70 (inclusively). The specific value that the attributes have depends upon the specific instance of the entity (i.e. the values will vary from one row to another).

- **Properties of the entity**

Entity Name: Car				
Attributes	VIN	Color	Model	…
Values	123XYZ…	Silver	Honda	…

- **Each attribute has…**
  - a value
  - domain → allowable values (i.e. field type)

Entity Name: Employee					
Attributes	SSN	Name	Age	Address	…
Values	123456789	Jane Doe	39	1 Ohm Dr…	…
Domain	numbers only	alphabetic characters	18 - 70		

Attributes

- English grammar components for attributes
    - Noun
    - Adjective
    - Adverb
- Every entity type should have an entity identifying attribute (EID) to uniquely identify an instance of the entity
- An EID cannot be null (this characteristic is known as an Entity Constraint). Later the EID will become the primary key of the table created from the entity
- Attribute Classifications
    - Describe the role the attribute will play in the database
    - Describe the data type of the attribute along with any constraints

Types of Attributes

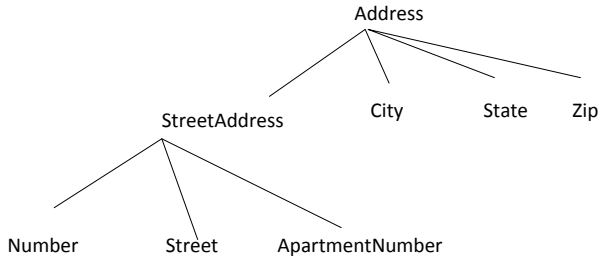
- **Composite vs. Simple (Atomic)** – Composite attributes are compound attribute composed of more than one attribute. For example, name can divided into smaller atomic elements such as first, middle, and last. An atomic attribute like last cannot be divided into any smaller component.
- **Single-valued vs. Multivalued** – an attribute like first name is single valued, a person can have only on first name. On the other hand, a person may have more than one phone number, more than one address and more than one degree that they have earned in school. When there can be more than one value for an attribute, it should be declared as a multivalued attribute.

- **Stored vs. Derived** – Some data is directly stored into the attributes of a database. This data becomes fields in the instances of a table. Some data is a direct result of a calculation. Examples of this is Gross Pay, it is a result of multiplying pay rate by hours worked. Another example is a person's age. A person's age changes every year. It would not be practical to store a person's age into a field of instance because it would have to be updated every year on their birthday. A better way would be to derive age from the person's birthday whenever age is required in a query. When a field is calculated and displayed in a result set of a query, it is a derived attribute.

- Complex

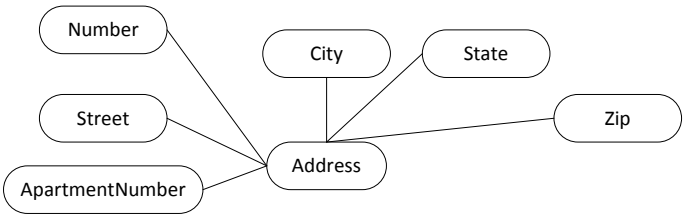
On an ER diagram, a composite attribute is drawn by drawing a line from the entity to the composite attribute. The subparts of the composite attributes are then drawn in separate ovals, which are directly connected to the composite attribute. This can be seen in the illustration below. Address is subdivided into street address, city, state and zip. Something to take note of is that the subcomponent of address Street Address is composite attribute inside of a composite attribute. It is further subdivided into the attribute Number, Street, and Apartment Number.

- Composite
    - Can be divided into smaller subparts



(See Fig. 3.4)

- Simple (Atomic) - Cannot be divided into smaller subparts
    - The attributes below cannot be divided into smaller subparts



Relationships – in our class, a relationship in the context of database management is defined in our class an association between two or more entities. When creating ER Diagrams, a relation is identified by a diamond shape. A relationship is typically given a name of a meaningful verb. The name given to the relationship, will typically covey the role that the relationship plays between the entities.

- An association between entities
- Relationship Name
    - meaningful verb
- Role name
    - signifies the role that an entity plays in each relationship instance
- Symbol

A relationship is identified by a Diamond with the relationship name displayed within the diamond. Each relationship has a property known as the degree of the relationship, which describes the number of entities that are involved in the relationship. If an entity has a relationship with itself, then the degree is one, and the relationship is known as a unary (recursive) relationship. If the relationship involves only two entities, then the degree of the relationship is 2, and the relationship is known as a binary relationship. If the relationship involves three entities, then the degree of the relationship is 3, and the relationship is known as a ternary relationship.

- Degree of relationship
  - the number of participating entities
- Unary (Recursive) Relationship
  - degree of 1
- Binary Relationship
  - degree of 2
- Ternary Relationship
  - degree of 3

Relationships: Constraints

Cardinality ratio (max. cardinality)- Ultimately, relationships are about joining tables together to extract meaningful information from a database using SELECT queries. We will learn that this is based on **Functional Dependency** (FD). To truly represent a relationship you need to make sure that the appropriate foreign keys are in place in the schema to allow joins between tables. Remember a **Foreign Key** is a **Primary Key** in another table. However, every time you construct a new query involving these tables you will need to specify the appropriate join conditions. Even if you define foreign keys, the **Database Management System (DBMS)** will not figure out the joins for you. This makes the relational model very flexible, but also makes queries hard to write for the novice.

All relationships turn into foreign keys. In the stable translation method that we will be studying they also turn into an extra table. They should also turn into “not null” constraints, a primary ky and foreign key should never contain null values.

To represent cardinality as a 1:M relationship, take the primary key of the table on the “1” side and insert it as a foreign key into the table on the “M” side. This is the most basic use of a foreign key. It may make sense to rename the foreign key to reflect its relationship to the table you are inserting it into.

To represent cardinality as a 1:1 relationship you have a choice. Ask yourself whether it makes more sense to leave this as two separate tables, or to join them together to make one big table. This will depend on whether records will usually exist in both tables, how data will be accesses, if joins will be made constantly or only occasionally when data is queried, etc. If the relationship is mandatory on both sides, there is probably little point in representing it as two tables. Assuming you chose to retain two tables for a 1:1 relationship, you must choose which table will receive the primary key of the other as a foreign key. If the relationship is mandatory for one entity but not the other, then put foreign key into the table for which participation is mandatory. If it is mandatory for neither, ask which is more likely to exist or to be queried. Put the foreign key in the table that will be accessed less frequently. This minimizes the number of joins required when querying.

If the 1:1 or 1:M relationship is mandatory, you will need a "not null" constraint on the foreign key. If the foreign key can never be null, this means it must always refer to a row in the related primary key table. Therefore, because of referential integrity, it means that a row cannot exist in the foreign key table without a related row on the other side of the relationship in the primary key table.

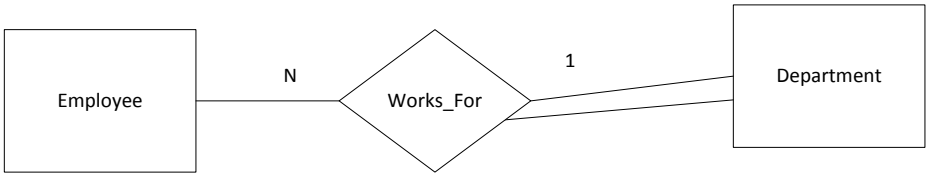
A weak entity will have a foreign key as part of its own primary key. If it has a 1:M relationship with the strong entity that defines it, it will also have a partial key of its own, to tell it apart from other weak entities associated with the same strong entity. (For example, different instances of the same flight number in a reservation table). This partial key, together with the primary key of the defining entity inserted as a foreign key, together make up the primary key of the table for the weak entity.

There is no direct representation of a M:N relationship in the relational model. You will need to turn each M:N relationship between two entities into a separate relation (table) of its own. This relation will usually have as its own primary key the combination of two foreign keys – each of these will be the primary key of one of the relations involved in this relationship. If a relationship has attributes then they need to go into a table. Where to put them depends on the type of the relationship. In a 1:1 or 1:M relationship, put them the same place the foreign

key goes (on the M side in 1:M). In a M:N relationship, put them in the new table you create for the relationship.

I advise you to avoid n-ary relationships in the first place (a single relationship including three or more entities). They can usually be better represented by using an additional entity and a set of binary relationships. If you had an n-ary relationship on your ERM then follow these steps to simplify them – then translate the simpler relationships and the new entity as normal.

- **Cardinality ratio (max. cardinality**
  - Cardinality is the Max. # of relationship instances that an entity can participate in
  - There are three types of cardinality - 1:1, 1:N, M:N
  - Symbol
    - ✓ ratio displayed on diamond
    - ✓ *Look across* notation for cardinality



Each department must have working for it one or more employees

Each employee may work for one and only one department

Rules for a Well-Defined ERD

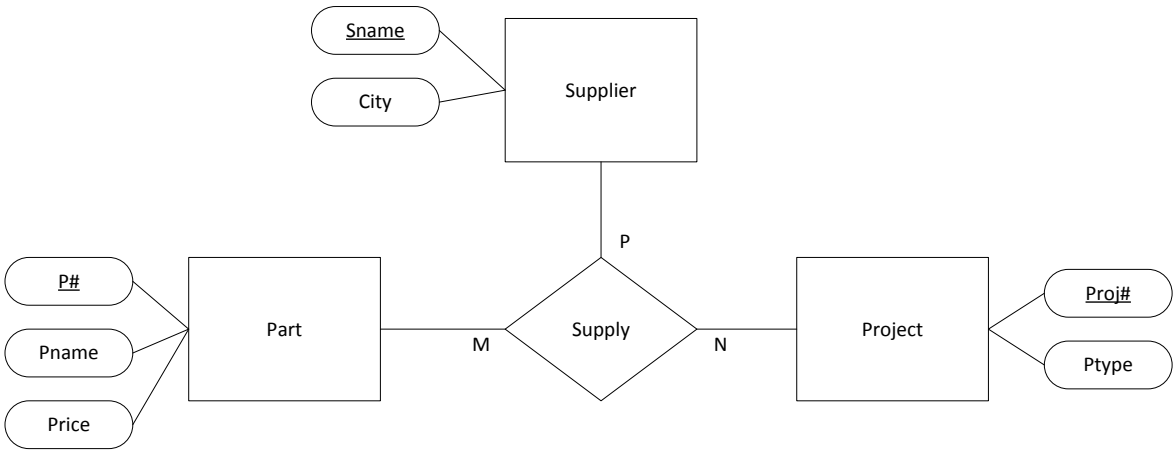
- **Basic Rules**
    - All entities & relationships must be connected
    - Each entity must have at least one relationship
    - All entity names must be unique
    - Use a singular noun for an entity name and an attribute name
    - A relationship cannot be directly connected to another relationship
    - Use a meaningful verb for a relationship name whenever possible
    - Every entity must have at least one unique key (atomic or concatenated
- Primary Key
    - No two instances can have the same value in the table
    - There should not be any redundant attribute in a key
    - Requirements & preferences
    - Known at all times
    - Must not have a null value (not null: entity constraint)
    - Should not be changed
    - Prefer a short one

- Primary keys in the ER model
    - The PK of an Entity, the EID, becomes the primary key in the table
        - ✓ Primary Key is the same as the Entity Identifier
    - PK of a relationship
        - ✓ Depends upon the cardinality
            - 1:1 → could be the PK of either side
            - 1:N → same PK as the N-side
            - M:N → concatenation of PK of both sides

Ternary Relationship

- A relationship among 3 entity types
- Requirements
    - All three entities must always occur at the same time
    - We need to process queries using these three entities
- Types of Cardinality
    - Can be 1:1:1, 1:M:1, 1:M:N or M:N:P

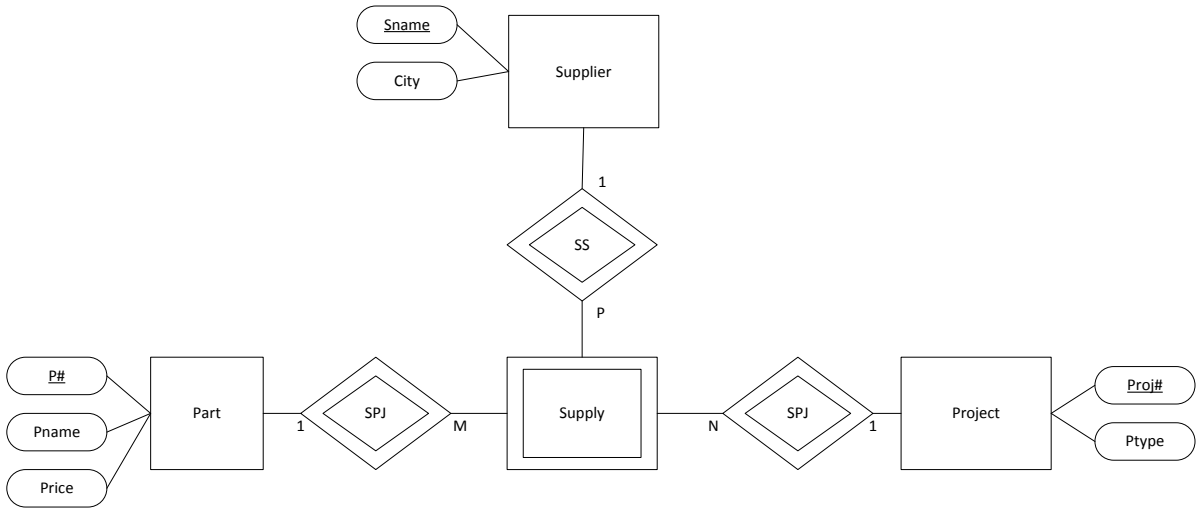
Ternary Example



- Semantics
    - For a given pair of Supplier & Project, there are many Parts
    - For a given pair of Parts & Supplier, there are many Projects
    - For a given pair of Parts & Projects, there are many Suppliers

- Conversion to Binary Relationships
    - The cardinality becomes 1:N in all three relationships

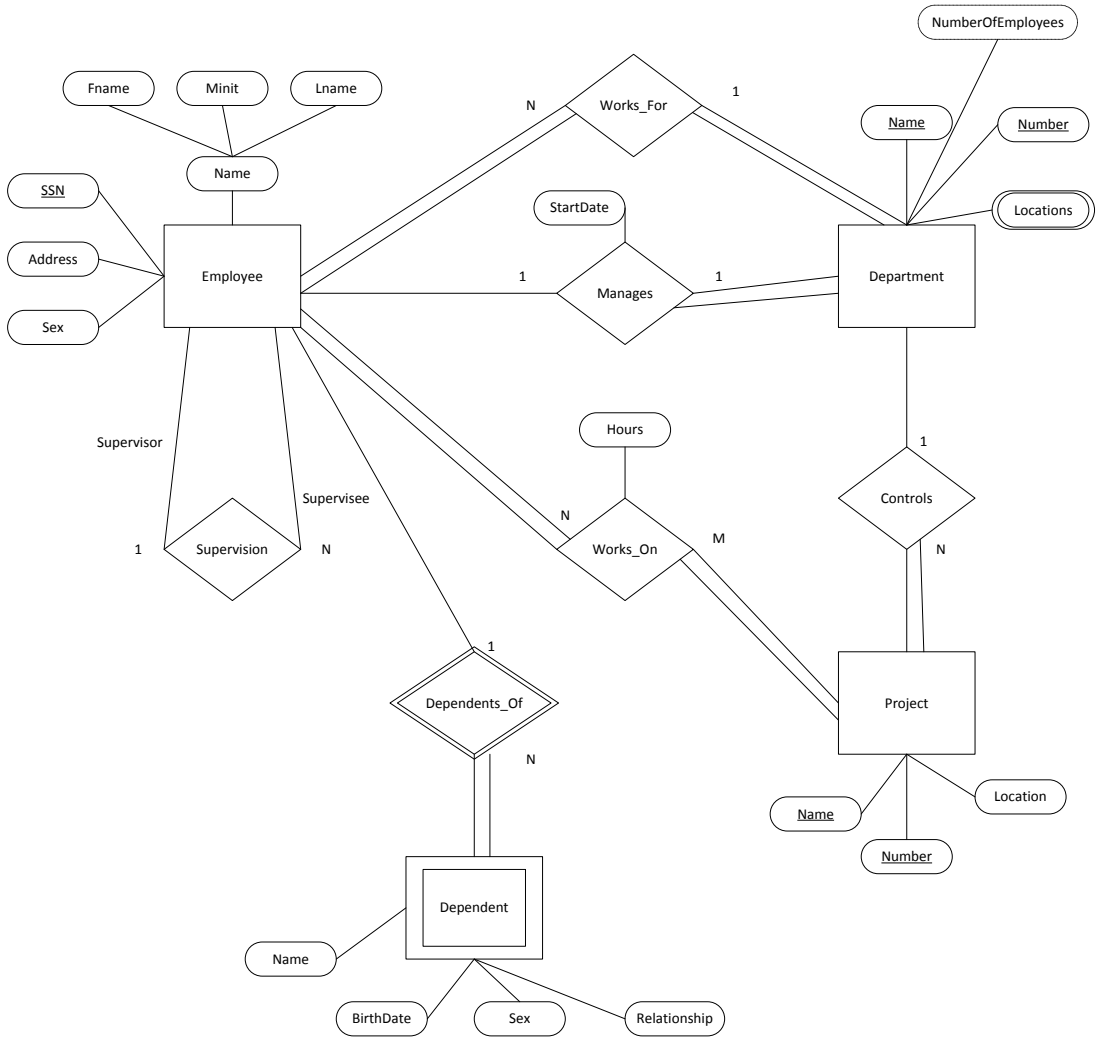
The illustration below is an example of an ER diagram that converts the Ternary Model illustrated previously example into multiple binary relationships. In this example three binary relationships are created to replace the one ternary relationship. There is a 1:M relationship between supplier and supply, a 1:M relationship between Part and Supply, and a 1:M relationship between project and supply. This provides a framework for less complex queries on the database.



When converting a ternary relationship into a set of binary relationships the process is identical to the process outlined for binary relationships.

1. Gather & understand requirements
2. Identify each entity
3. Draw ERD without attributes
4. Identify each relationship
5. Identify & draw cardinality constraints
6. Assign attributes
7. Assign keys
8. Check model

Conclusion



Practice reading this diagram. Write out the sentences that describe the relationships between the entities. For example:

Each employee must work for one and only one department. Each department must have working for it one or more employees.

Each employee may manage one and only department, and each department must be managed by one and only one employee.

We will practice ER Diagraming for several weeks

Last Modified by Craig Nelson 09/7/2013