Data Requirements and Data modeling

The data model focuses on what data should be stored in the database while the function model deals with how the data is processed. In this chapter, we'll look into details of data modeling.

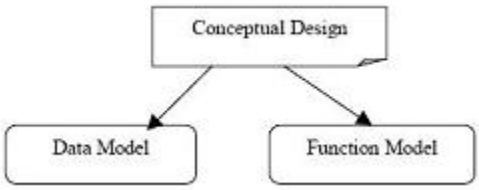


Fig 7.1 Elements of Conceptual Design

Prior to data modeling, we'll talk of basics of database design process. The database design process can be described as a set of following steps.

- **Requirement collection:** Here the database designer interviews database users. By this process they are able to understand their data requirements. Results of this process are clearly documented. In addition to this, functional requirements are also specified. Functional requirements are user defined operations or transaction like retrievals, updates, etc, that are applied on the database.
- **Conceptual schema:** Conceptual schema is created. It is the description of data requirements of the users. It includes description of data types, relationships and constraints.
- Basic data model operations are used to specify user functional requirements.
- Actual implementation of database.
- Physical database design. It includes design of internal storage structures and files.

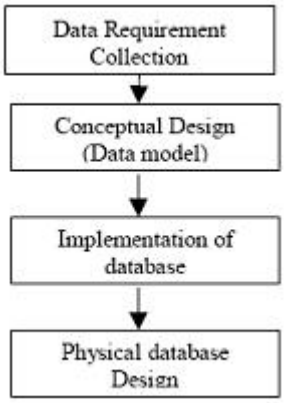


Fig 7.2 Overall database design process

There are various data models available. They fall in three different groups.

- Object-based logical models
- Records-based logical models
- Physical-models

Object-Based Logical Models

Object-based logical models are used in describing data at the logical and view levels. The main characteristic of these models is that they **provide flexible structuring capabilities and allows data constraints to be specified explicitly**. Many different models fall into this group. They are following.

- Entity-relationship model
- Object-oriented model

Record-Based Logical Models

Records-based logical models are used in describing data at the logical and view levels. They are used to **specify the overall logical structure of the database** and to **provide a higher-level description of the implementation**.

In record-based models, the database is structured in fixed-format records of several types. Each record type defines a fixed number of fields, or attributes, and each field is usually of a fixed length. The use of fixed-length records simplifies the physical-level implementation of the database.

The following models fall in this group.

- Relational model
- Network model
- Hierarchical model

Physical Data Models

Physical data models are used to describe data at the lowest level. A few physical data models are in use. Two such models are:

- Unifying model
- Frame-memory model

Physical data models capture aspects of database-system implementation.

E-R Data Modeling Technique

To understand the process of data modeling we'll study Entity Relationship model. Peter P. Chen originally proposed the Entity- Relationship (ER) model in 1976. The ER model is a **conceptual data model** that views **the real world as a construct of entities and associations or relationships between entities**.

A basic component of the model is the Entity-Relationship diagram, which is used to **visually represent data objects**. The ER modeling technique is frequently used for **the conceptual design of database applications** and many database design tools employ its concepts.

ER model is easy to understand. Moreover **it maps easily to relational model**. The constructs used in ER model can **easily be transformed into relational tables**.

We can compare ER diagram with a flowchart for programs. Flow chart is a tool for designing a program; similarly ERD is a tool for designing databases. Also an ER diagram shows the kind and organization of the data that will be stored in the database in the same way a flowchart chose the way a program will run.

E-R Model concept

The ER data modeling techniques is **based on the perception of a real world that consists of a set of basic objects called entities, and of relationships among these objects**. In ER modeling, **data is described as entities, relationships, and attributes**.

Entities and Attributes

One of the basic components of ER model is entity. **An entity is any distinguishable object about which information is stored**. These objects can be person, place, thing, event or a concept. Entities contain descriptive information. Each entity is distinct.

An entity may be physical or abstract. A person, a book, car, house, employee etc. are all physical entities whereas a company, job, or a university course, are abstract entities.

Fig. 7.3 Physical and Abstract Entity

Another classification of entities can be independent or dependent (strong or weak) entity.

Entities are classified as independent or dependent (in some methodologies, the terms used are strong and weak, respectively). An independent entity is one, which does not rely on another entity for identification. **A dependent entity is one that relies on another entity for identification. An independent entity exists on its own whereas dependent entity exists on the existence of some other entity**. For example take an organization scenario. Here department is independent entity. Department manager is a dependent entity. It exists for existing depts. There won't be any department manager for which there is no dept.

Some entity types may not have any key attributes of their own. These are called **weak entity** types. Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with some of their attribute values. For example, take the license entity. It can't exist unless it is related to a person entity.

Attributes

After you identify an entity, then you describe it in real terms, or through its attributes. Attributes are basically properties of entity. We can use attributes for identifying and expressing entities. For example, Dept entity can have DeptName, DeptId, and DeptManager as its attributes. A car entity can have modelno, brandname, and color as its attributes.

A particular instance of an attribute is a value. For example, "Bhaskar" is one value of the attribute Name. Employee number 8005 uniquely identifies an employee in a company.

The value of one or more attributes can uniquely identify an entity.

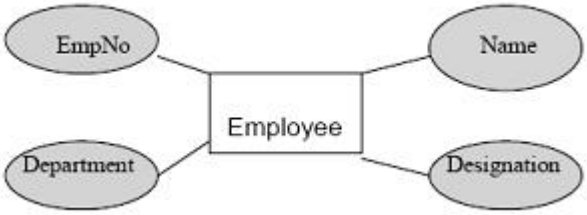


Fig 7.4 Entity and its attributes

In the above figure, employee is the entity. EmpNo, Name, Designation and Department are its attributes.

An entity set may have several attributes. Formally each entity can be described by set of <attribute, data value> pairs.

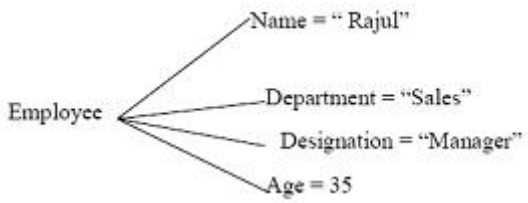


Fig. 7.5 Employee entity and its attribute values

Types of Attributes

Attributes can be of various types. In this section, we'll look at different types of attributes. Attributes can be categorized as:

- Key or non key attributes
- Required or optional Attributes
- Simple or composite Attributes

- Single-valued and multi-valued Attributes
- Stored, Coded or derived Attributes

Key or non-key attributes

Attributes can be classified as **identifiers or descriptors**. Identifiers, more commonly called **keys or key attributes uniquely identify an instance of an entity**. If such an attribute doesn't exist naturally, a new attribute is defined for that purpose, for example an ID number or code. A descriptor describes a non-unique characteristic of an entity instance.

An entity usually has an attribute whose values are distinct for each individual entity. This attribute uniquely identifies the individual entity. Such an attribute is called a key attribute. For example, in the Employee entity type, EmpNo is the key attribute since no two employees can have same employee number. Similarly, for Product entity type, ProdId is the key attribute.

There may be a case when one single attribute is not sufficient to identify entities. Then a combination of attributes can solve this purpose. We can form a group of more than one attribute and use this combination as a key attribute. That is known as a composite key attribute. When identifying attributes of entities, identifying key attribute is very important.

Required or optional Attributes

An attribute can be required or optional. When it's required, we must have a value for it, a value must be known for each entity occurrence. When it's optional, we could have a value for it, a value may be known for each entity occurrence. For example, there is an attribute EmpNo (for employee no.) of entity employee. This is required attribute since here would be no employee having no employee no. Employee's spouse is optional attribute because an employee may or may not have a spouse.

Simple and composite Attributes

Composite attributes can be divided into smaller subparts. These subparts represent basic attributes with independent meanings of their own. For example, take Name attributes. We can divide it into sub-parts like First_name, Middle_name, and Last_name.

Attributes that can’t be divided into subparts are **called Simple or Atomic attributes**. For example, EmployeeNumber is a simple attribute. Age of a person is a simple attribute.

Single-valued and multi-valued attributes

Attributes that can have single value at a particular instance of time are called single-valued. A person can’t have more than one age value. Therefore, age of a person is a single-values attribute. A multi-valued attribute can have more than one value at one time. For example, degree of a person is a multi-valued attribute since a person can have more than one degree. Where appropriate, upper and lower bounds may be placed on the number of values in a multi-valued attribute. For example, a bank may limit the number of addresses recorded for a single customer to two.

Stored, coded, or derived Attributes

There may be a case when two or more attributes values are related. Take the example of age. Age of a person can be calculated from person’s date of birth and present date. Difference between the two gives the value of age. In this case, age is the derived attribute.

The attribute from which another attribute value is derived is called stored attribute. In the above example, date of birth is the stored attribute. Take another example, if we have to calculate the interest on some principal amount for a given time, and for a particular rate of interest, we can simply use the interest formula

Interest=NPR/100;

In this case, interest is the derived attribute whereas principal amount(P), time(N) and rate of interest(R) are all stored attributes.

Derived attributes are usually created by a formula or by a summary operation on other attributes.

A coded value uses one or more letters or numbers to represent a fact. For example, the value Gender might use the letters "M" and "F" as values rather than "Male" and "Female".

The **attributes reflect the need for the information** they provide. In the analysis meeting, the participants should list as many attributes as possible. Later they can weed out those that are not applicable to the application, or those clients are not prepared to spend the resources on to collect and maintain. The participants come to an agreement, on which attributes belong with an entity, as well as which attributes are required or optional.

Entity Types

An entity set is **a set of entities of the same type that share the same properties, or attributes**. For example, all software engineers working in the department involved in the Internet projects can be defined as the entity set InternetGroup. The individual entities that constitute a **set are called extension of the entity set**. Thus, all individual software engineers of in the Internet projects are the extensions of the entity set InternetGroup.

Entity sets don't need to be disjointed. For example, we can define an entity set Employee. An employee may or may not be working on some Internet projects. In InternetGroup we will have some entries that are there in Employee entity set. Therefore, entity sets Employee and InternetGroup are not disjoint.

A database usually **contains groups of entities that are similar**. For example, employees of a company share the same attributes. However, every employee entity has its own values for each attribute. An entity type defines a set of entities that have same attributes. A name and a list of attributes describe each entity type.

Fig. 7.6 shows two entity types Employee and Product. Their attribute list is also shown. A few members of each entity type are shown.

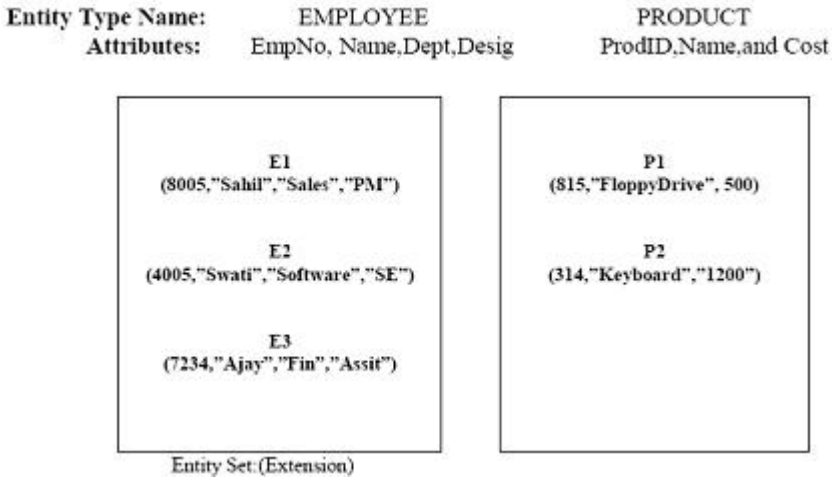


Fig. 7.6 Two entity types and some of the member entities of each

An entity type is represented in ER diagrams as rectangular box and the corresponding attributes are shown in ovals attached to the entity type by straight lines. See fig 7.4.

An entity type is **basically the schema** or intension or structure for the set of entities that share the same structure whereas the individual entities of a particular entity type are collectively called entity set. The entity set is also called the extension of the entity type.

Value Sets (domain) of Attributes

Each attribute of an entity type is associated with a value set. This **value set is also called domain**. The domain of an attribute is the collection of all possible values an attribute can have.

The value set specifies the set of values that may be assigned for each individual entity. For example, we can specify the value set for designation attribute as <"PM", "Assit", "DM", "SE">. We can specify "Name" attribute value set as <strings of alphabetic characters separated by blank characters>. The domain of Name is a character string.

Entity Relationships

After identification of entities and their attributes, the next stage in ER data modeling is to identify the relationships between these entities.

We can say **a relationship is any association, linkage, or connection between the entities of interest** to the business. Typically, a relationship is **indicated by a verb** connecting two or more entities. Suppose there are two entities of our library system, member and book, then the relationship between them can be "borrows".

Member borrows book

Each relationship has a name, degree and cardinality. These concepts will be discussed next.

Degree of an Entity Relationship Type

Relationships exhibit certain characteristics like **degree, connectivity, and cardinality**. Once the **relationships are identified their degree and cardinality** are also specified.

Degree: The degree of a relationship is **the number of entities associated with the relationship**. The **n-ary** relationship is **the general form for degree n**. Special cases are the binary, and ternary, where the degree is 2, and 3, respectively.

Binary relationships, the association between two entities are the most common type in the real world.

Fig 7.7 shows a binary relationship between member and book entities of library system

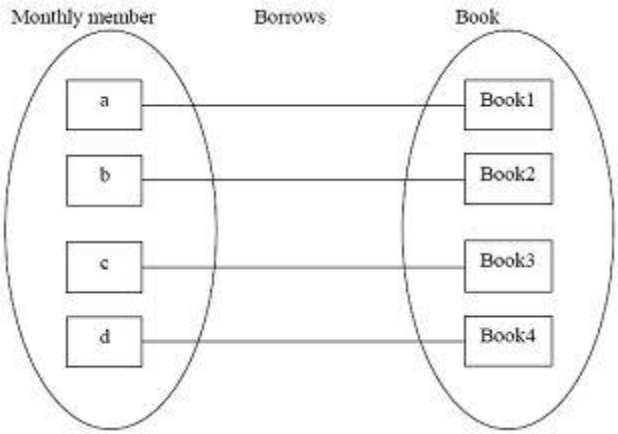


Fig. 7.7 Binary Relationship

A ternary relationship involves three entities and is used when a binary relationship is inadequate. Many modeling approaches recognize only binary relationships. Ternary or n-ary relationships are decomposed into two or more binary relationships.

Connectivity and Cardinality

By connectivity we mean how many instances of one entity are associated with how many instances of other entity in a relationship. **Cardinality is used to specify such connectivity**. The connectivity of a relationship describes the mapping of associated entity instances in the relationship. The values of connectivity are "one" or "many". The cardinality of a relationship is the actual number of related occurrences for each of the two entities. The basic types of connectivity for relations are: one-to-one, one-to-many, and many-to-many.

A **one-to-one (1:1)** relationship is when at most **one instance of an** entity A is associated with one instance of entity B. For example, each book in a library is issued to only one member at a particular time.

A **one-to-many (1:N)** relationship is when for one instance of entity A, there are zero, one, or many instances of entity B but for one instance of entity B, there is only one instance of entity A. An example of a 1:N relationships is

a department has many employees;
each employee is assigned to one department.

A **many-to-many (M:N)** relationship, sometimes called non-specific, is when for **one instance of entity A, there are zero, one, or many instances of entity B and for one instance of entity B there are zero, one, or many instances of entity A**. An example is employees may be assigned to no more than three projects at a time; every project has at least two employees assigned to it.

Here the cardinality of the relationship from employees to projects is three; from projects to employees, the cardinality is two. Therefore, this relationship can be classified as a many-to-many relationship.

If a relationship can have a cardinality of zero, it is an optional relationship. If it must have a cardinality of at least one, the relationship is mandatory. Optional relationships are typically indicated by the conditional tense. For example,

An employee may be assigned to a project.

Mandatory relationships, on the other hand, are indicated by words such as must have. For example,

a student must register for at least three courses in each semester.

Designing basic model and E-R Diagrams

E-R diagrams represent the schemas or the overall organization of the system. In this section, we'll apply the concepts of E-R modeling to our "Library Management System" and draw its E-R diagram.

In order to begin constructing the basic model, the modeler must analyze the information gathered during the requirement analysis for the purpose of: and

- classifying data objects as either entities or attributes,
- identifying and defining relationships between entities,
- naming and defining identified entities, attributes, and relationships,
- Documenting this information in the data document.
- Finally draw its ER diagram.

To accomplish these goals the modeler must analyze narratives from users, notes from meeting, policy and procedure documents, and, if lucky, design documents from the current information system.

E-R diagrams constructs

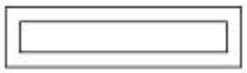
In E-R diagrams, entity types are represented by squares. See the table below. Relationship types are shown in diamond shaped boxes attached to the participating entity types with straight lines. Attributes are shown in ovals, and each attribute is attached to its entity type or relationship type by a straight line. Multivalued attributes are shown in double ovals. Key attributes have their names underlined. Derived attributes are shown in dotted ovals.

Weak entity types are distinguished by being placed in double rectangles and by having their identifying relationship placed in double diamonds.

Attaching a 1, M, or N on each participating edge specifies cardinality ratio of each binary relationship type. The participation constraint is specified by a single line for partial participation and by double lines for total participation. The participation constraints specify whether the

existence of an entity depends on its being related to another entity via the relationship type. If every entity of an entity set is related to some other entity set via a relationship type, then the participation of the first entity type is total. If only few member of an entity type is related to some entity type via a relationship type, the participation is partial.


ENTITY TYPE


WEAK ENTITY TYPE

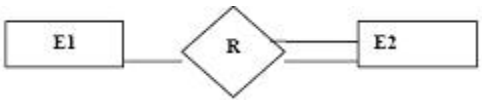

RELATIONSHIP TYPE


ATTRIBUTE

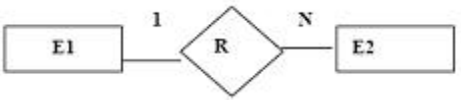

KEY ATTRIBUTE


MULTIVALUED ATTRIBUTE

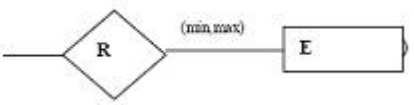

DERIVED ATTRIBUTE



TOTAL PARTICIPATION OF
E2 IN R



Cardinality Ratio 1:N FOR
E1:E2 IN R



Structural Constraint(Min,Max)
On Participation Of E In R

Naming Data Objects

The names should have the following properties:

- Unique,
- Have meaning to the end-user.
- Contain the minimum number of words needed to uniquely and accurately describe the object.

For entities and attributes, names are singular nouns while relationship names are typically verbs.

E-R Diagram for library management system

In the library Management system, the following entities and attributes can be identified.

- ***Book*** -the set all the books in the library. Each book has a Book-id, Title, Author, Price, and Available (y or n) as its attributes.
- ***Member***-the set all the library members. The member is described by the attributes Member_id, Name, Street, City, Zip_code, Mem_type, Mem_date (date of membership), Expiry_date.
- ***Publisher***-the set of all the publishers of the books. Attributes of this entity are Pub_id, Name, Street, City, and Zip_code.
- ***Supplier***-the set of all the Suppliers of the books. Attributes of this entity are Sup_id, Name, Street, City, and Zip_code.

Assumptions: a publisher publishes a book. Supplier supplies book to library. Members borrow the book (only issue).

Return of book is not taken into account

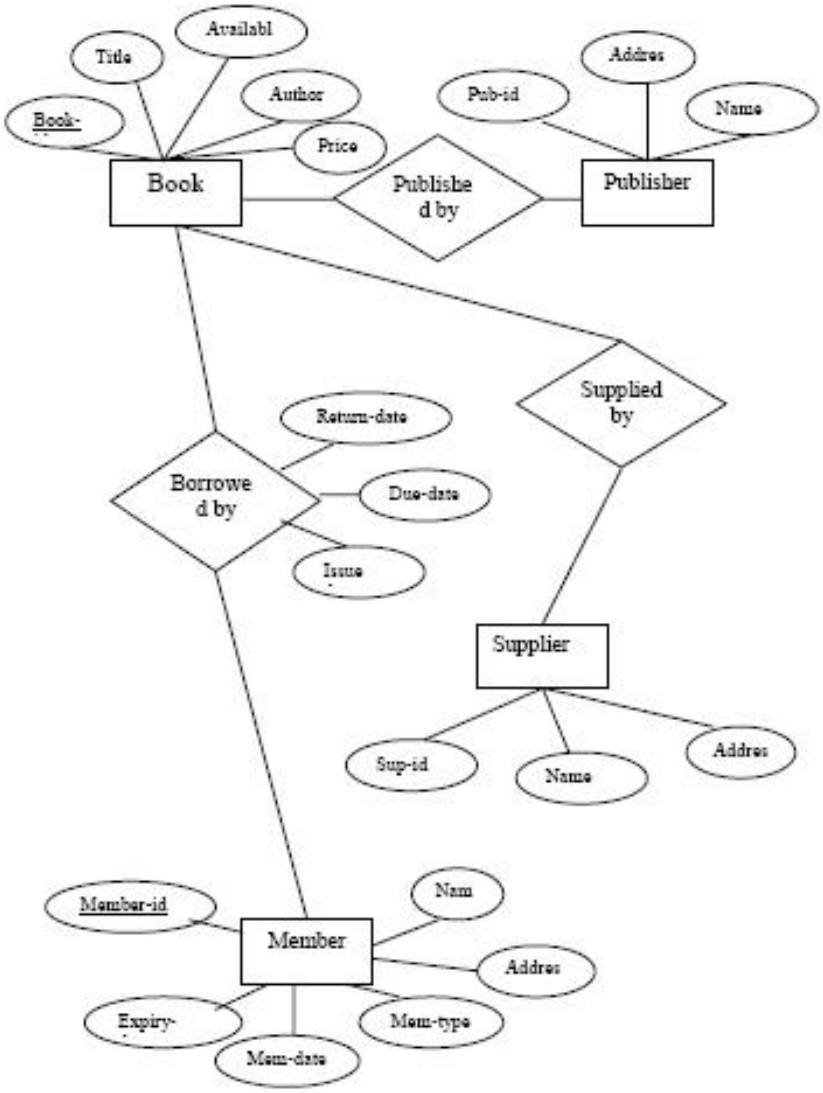


Fig. 7.8

E-R Diagram of Library Management System.