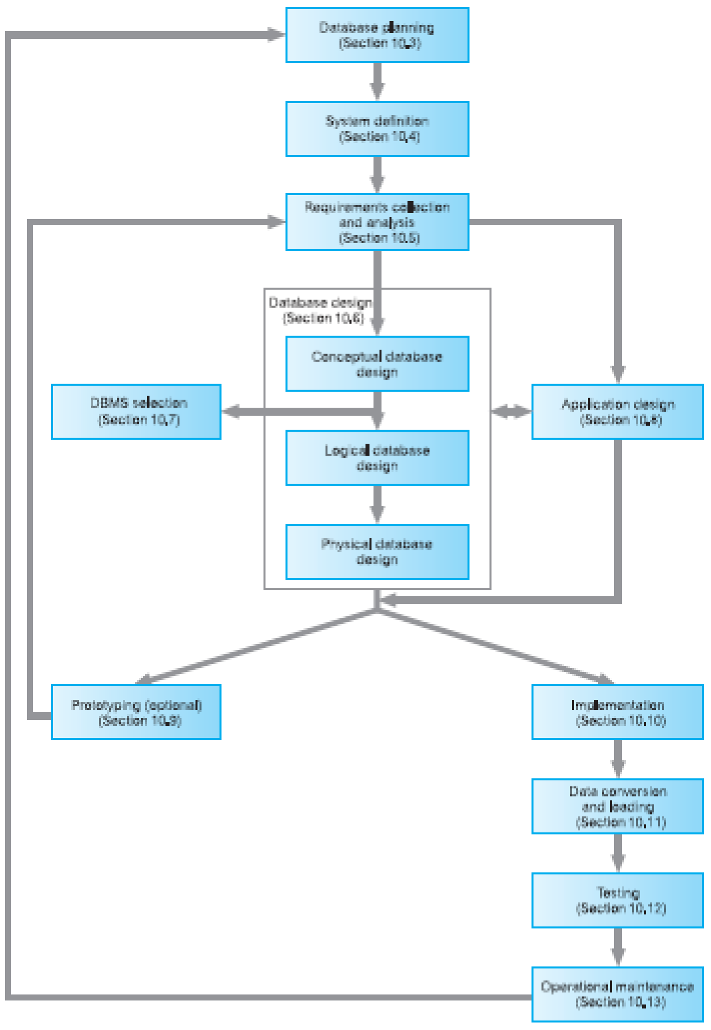
DATABASE SYSTEM DEVELOPMENT LIFECYCLE:


Database planning (Section 10.3)
System definition (Section 10.4)
Requirements collection and analysis (Section 10.5)
Database design (Section 10.6)
Conceptual database design
DBMS selection (Section 10.7)
Application design (Section 10.8)
Logical database design
Physical database design
Prototyping (optional) (Section 10.9)
Implementation (Section 10.10)
Data conversion and loading (Section 10.11)
Testing (Section 10.12)
Operational maintenance (Section 10.13)

Database Planning:

The management activities that allow the stages of the database system development lifecycle to be realized as efficiently and effectively as possible.

An important first step in database planning is to clearly define the mission statement for the database system. The mission statement defines the major aims of the database system. Those driving the database project within the organization (such as the Director and/or owner) normally define the mission statement. A mission statement helps to clarify the purpose of the database system and provide a clearer path towards the efficient and effective creation of the required database system.

Once the mission statement is defined, the next activity involves identifying the mission objectives. Each mission objective should identify a particular task that the database system must support. The assumption is that if the database system supports the mission objectives, then the mission statement should be met.

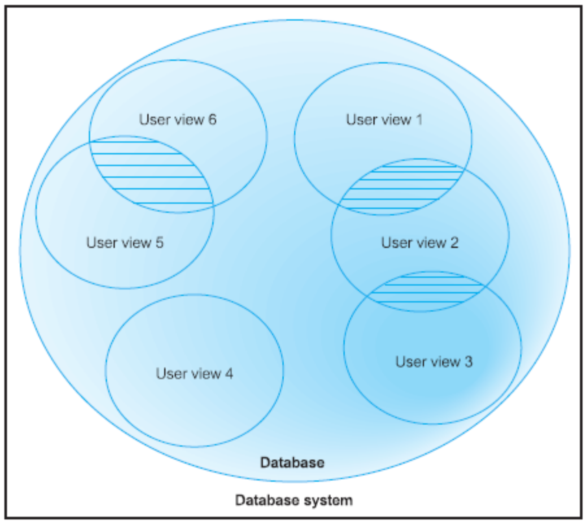
Database planning should also include the development of standards that govern how data will be collected, how the format should be specified, what documentation will be needed, and how design and implementation should proceed.

System Definition:

Describes the scope and boundaries of the database system and the major user views.

User Views: Defines what is required of a database system from the perspective of a particular job role (such as Manager or Supervisor) or enterprise application area (such as marketing, personnel, or stock control).

A database system may have one or more user views. Identifying user views is an important aspect of developing a database system because it helps to ensure that no major users of the database are forgotten when developing the requirements for the new database system.

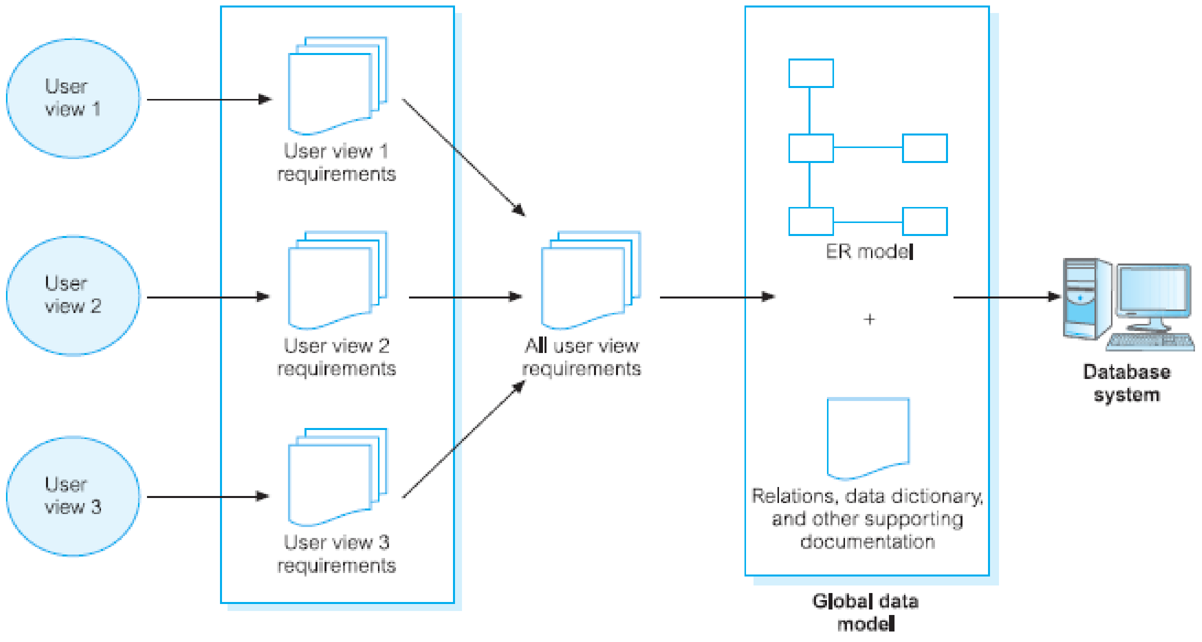


Representation of a database system with multiple user views: user views

There are three main approaches to managing the requirements of a database system with multiple user views:

- the centralized approach
- the view integration approach
- a combination of both approaches

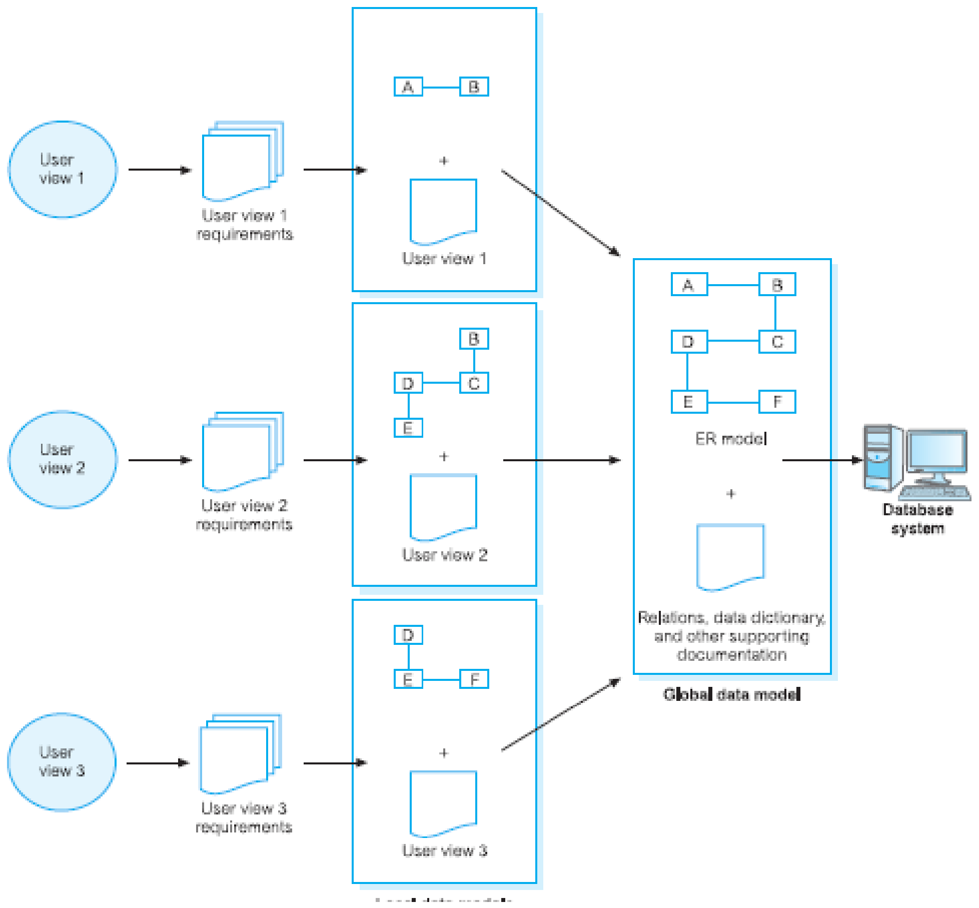
Centralized Approach: Requirements for each user view are merged into a single set of requirements for the new database system. A data model representing all user views is created during the database design stage.



The centralized (or one-shot) approach involves collating the requirements for different user views into a single list of requirements. The collection of user views is given a name that provides some indication of the application area covered by all the merged user views. In the database design stage, a global data model is created, which represents all user views.

View Integration Approach: Requirements for each user view remain as separate lists. Data models representing each user view are created and then merged later during the database design stage.

The view integration approach involves leaving the requirements for each user view as separate lists of requirements. In the database design stage, we first create a data model for each user view. A data model that represents a single user view (or a subset of all user views) is called a local data model. Each model is composed of diagrams and documentation that formally describes the requirements of one or more—but not all—user views of the database. The local data models are then merged at a later stage of database design to produce a global data model, which represents all user requirements for the database.



Database Design:

The process of creating a design that will support the enterprise’s mission statement and mission objectives for the required database system.

The two main approaches to the design of a database are:

- “bottom-up”
- “top-down.”

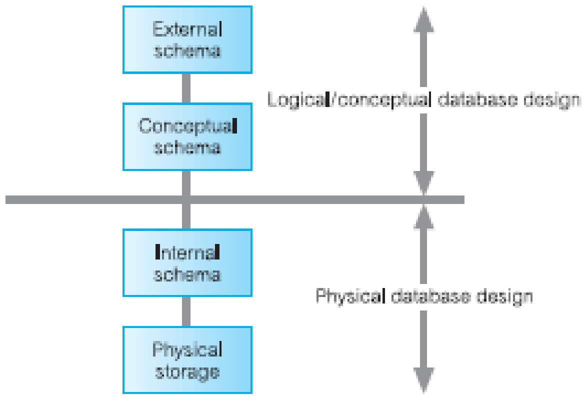
The bottom-up approach begins at the fundamental level of attributes (that is, properties of entities and relationships), which through analysis of the associations between attributes are grouped into relations that represent types of entities and relationships between entities.

A more appropriate strategy for the design of complex databases is to use the top-down approach. This approach starts with the development of data models that contain a few high-level entities and relationships and then applies successive top-down refinements to identify lower-level entities, relationships, and the associated attributes.

There are other approaches to database design, such as the inside-out approach and the mixed strategy approach. The inside-out approach is related to the bottom- up approach, but differs by first identifying a set of major entities and then spreading out to consider other entities, relationships, and attributes associated with those first identified. The mixed strategy approach uses both the bottom-up and top-down approach for various parts of the model before finally combining all parts together.

Database Design:

- Conceptual Database Design
- Logical Database Design
- Physical Database Design



Conceptual Database Design: The process of constructing a model of the data used in an enterprise, independent of all physical considerations.

Logical Database Design: The process of constructing a model of the data used in an enterprise based on a specific data model, but independent of a particular DBMS and other physical considerations.

Physical Database Design: The process of producing a description of the implementation of the database on secondary storage; it describes the base relations, file organizations, and indexes used to achieve efficient access to the data, and any associated integrity constraints and security measures.

DBMS Selection:

The selection of an appropriate DBMS to support the database system.

The steps involved in database selection are:

- Define Terms of Reference of study
- Shortlist two or three products
- Evaluate products
- Recommend selection and produce report

Application Design:

The design of the user interface and the application programs that use and process the database.

Database and application design are parallel activities of the database system development lifecycle. In most cases, it is not possible to complete the application design until the design of the database itself has taken place.

The steps involved in application design are:

- Transaction Design
- User interface Design Guidelines

Transaction Design: An action, or series of actions, carried out by a single user or application program, that accesses or changes the content of the database.

The purpose of transaction design is to define and document the high-level characteristics of the transactions required on the database, including:

- data to be used by the transaction
- functional characteristics of the transaction
- output of the transaction
- importance to the users
- expected rate of usage

User interface Design Guidelines:

- Meaningful title
- Comprehensible instructions
- Logical grouping and sequencing of fields
- Visually appealing layout of the form/report
- Familiar field labels
- Consistent terminology and abbreviations
- Consistent use of color
- Visible space and boundaries for data entry fields
- Convenient cursor movement
- Error correction for individual characters and entire fields
- Error messages for unacceptable values
- Optional fields marked clearly
- Explanatory messages for fields
- Completion signal

Prototyping:

Building a working model of a database system.

A prototype is a working model that does not normally have all the required features or provide all the functionality of the final system. The main purpose of developing a prototype database system is to allow users to use the prototype to identify the features of the system that work well or are inadequate, and if possible to suggest improvements or even new features to the database system.

There are two prototyping strategies:

- requirements prototyping - uses a prototype to determine the requirements of a proposed database system, and once the requirements are complete, the prototype is discarded
- evolutionary prototyping - used for the same purposes, the important difference is that the prototype is not discarded, but with further development becomes the working database system

Implementation:

The physical realization of the database and application designs.

The database implementation is achieved using the DDL of the selected DBMS or a GUI, which provides the same functionality while hiding the low-level DDL statements. The application programs are implemented using the preferred third- or fourth generation language (3GL or 4GL).

Data Conversion and Loading:

Transferring any existing data into the new database and converting any existing applications to run on the new database. This stage is required only when a new database system is replacing an old system.

Testing:

The process of running the database system with the intent of finding errors.

Before going live, the newly developed database system should be thoroughly tested. This is achieved using carefully planned test strategies and realistic data, so that the entire testing process is methodically and rigorously carried out.

Operational Maintenance:

The process of monitoring and maintaining the database system following installation.

- Monitoring the performance of the system. If the performance falls below an acceptable level, tuning or reorganization of the database may be required.
- Maintaining and upgrading the database system (when required). New requirements are incorporated into the database system through the preceding stages of the lifecycle.

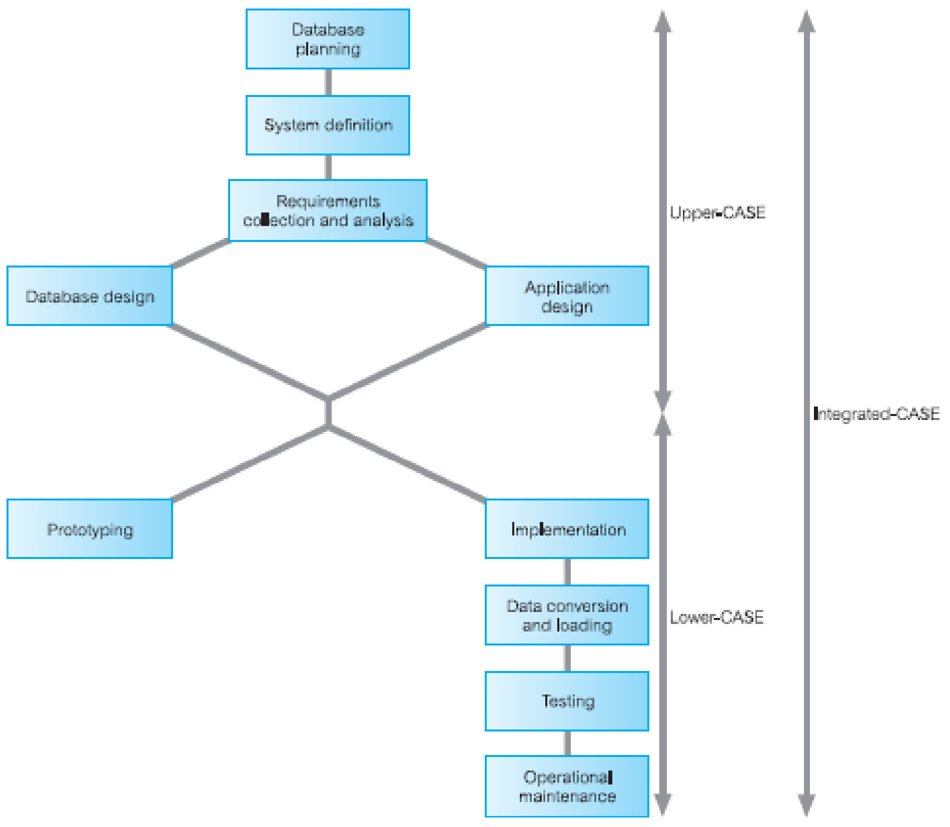
CASE Tools:

A computer-aided software engineering (CASE) tool is a software package that provides support for the design and implementation of information systems. It can document a database design and provide invaluable help in maintaining the consistency of a design. By integrating many of the techniques used to document a system design including the data dictionary, data flows, and entity relationships, CASE tool can increase the consistency and accuracy of a database design. It can also ease the task of creating the diagrams that accompany a system design.

There is no software in the world that can examine a database environment and identify the entities, attributes, and relationships that should be represented in a database. The model created with CASE tool is therefore only as good as the analysis of the database environment provided by the people using the tool.

CASE support may include:

- a data dictionary to store information about the database system’s data
- design tools to support data analysis
- tools to permit development of the corporate data model, and the conceptual and logical data models
- tools to enable the prototyping of applications



Requirements Collection:

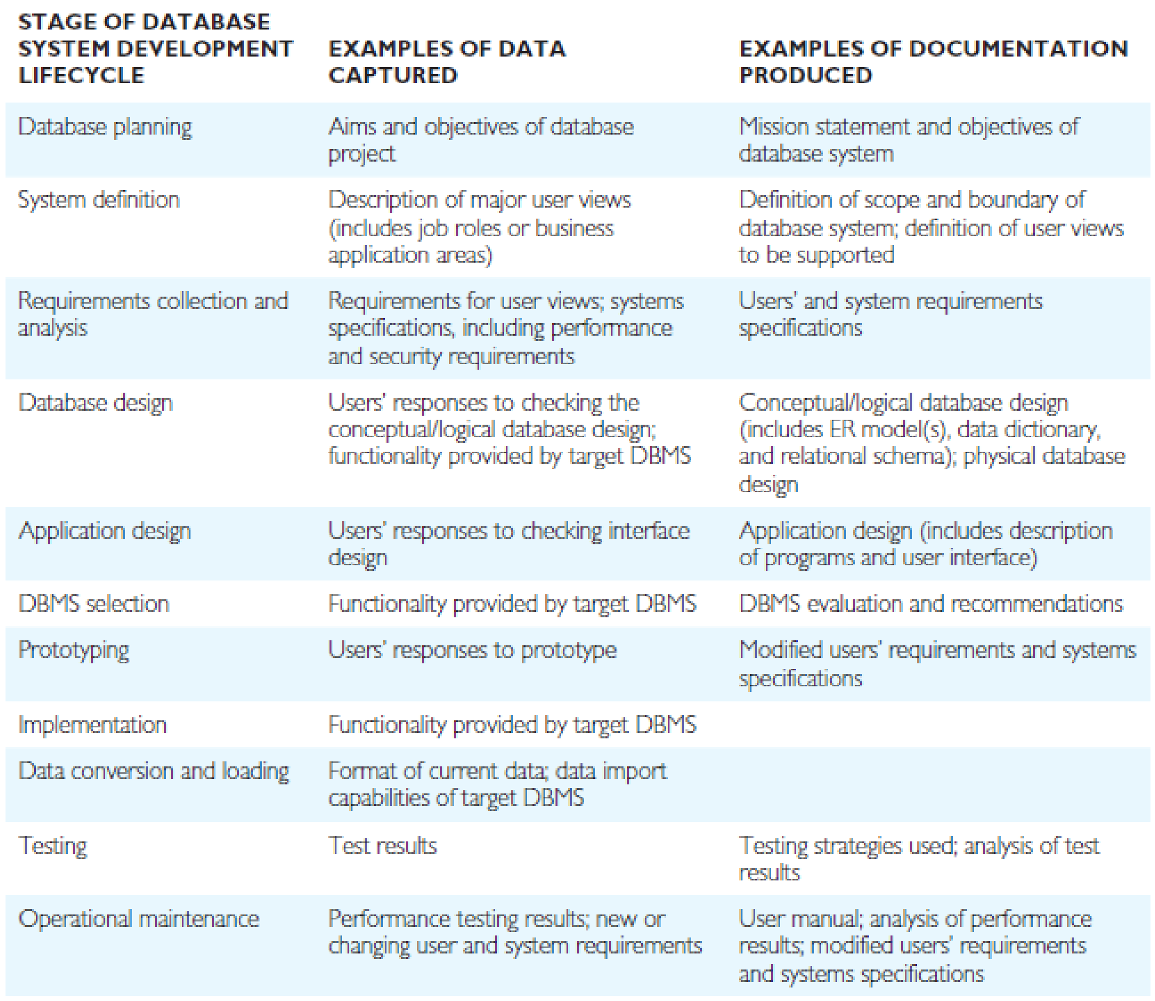
Fact Finding Technique:

The formal process of using techniques such as interviews and questionnaires to collect facts about systems, requirements, and preferences. Fact-finding is particularly crucial to the early stages of the lifecycle, including the database planning, system definition, and requirements collection and analysis stages.

There are five commonly used fact-finding techniques:

- examining documentation
- interviewing
- observing the enterprise in operation
- research
- questionnaires

the below diagram depicts the examples of data captured in each of the stages of the development life cycle.

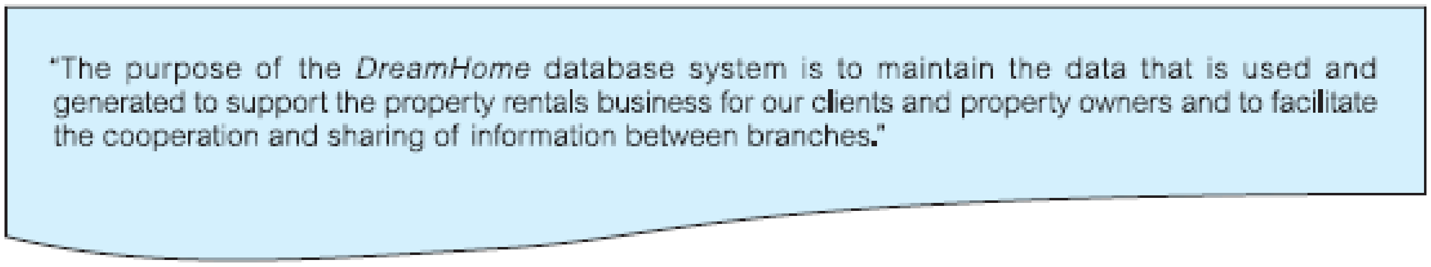


STAGE OF DATABASE SYSTEM DEVELOPMENT LIFECYCLE	EXAMPLES OF DATA CAPTURED	EXAMPLES OF DOCUMENTATION PRODUCED
Database planning	Aims and objectives of database project	Mission statement and objectives of database system
System definition	Description of major user views (includes job roles or business application areas)	Definition of scope and boundary of database system; definition of user views to be supported
Requirements collection and analysis	Requirements for user views; systems specifications, including performance and security requirements	Users' and system requirements specifications
Database design	Users' responses to checking the conceptual/logical database design; functionality provided by target DBMS	Conceptual/logical database design (includes ER model(s), data dictionary, and relational schema); physical database design
Application design	Users' responses to checking interface design	Application design (includes description of programs and user interface)
DBMS selection	Functionality provided by target DBMS	DBMS evaluation and recommendations
Prototyping	Users' responses to prototype	Modified users' requirements and systems specifications
Implementation	Functionality provided by target DBMS	
Data conversion and loading	Format of current data; data import capabilities of target DBMS	
Testing	Test results	Testing strategies used; analysis of test results
Operational maintenance	Performance testing results; new or changing user and system requirements	User manual; analysis of performance results; modified users' requirements and systems specifications

DreamHome Case Study:

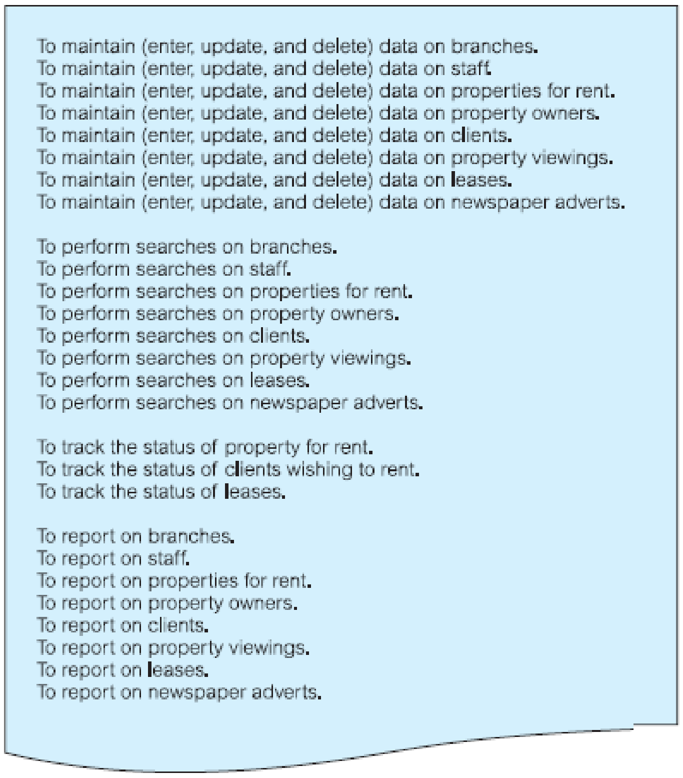
DreamHome specializes in property management, taking an intermediate role between owners who wish to rent out their furnished property and clients of DreamHome who require to rent furnished property for a fixed period. DreamHome currently has about 2000 staff working in 100 branches.

Mission Statement for the Case Study:



"The purpose of the *DreamHome* database system is to maintain the data that is used and generated to support the property rentals business for our clients and property owners and to facilitate the cooperation and sharing of information between branches."

Mission Objectives for the Case Study:



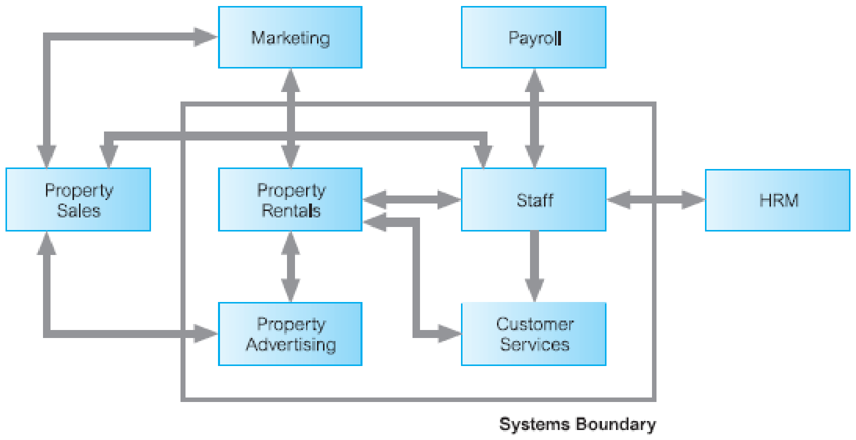
To maintain (enter, update, and delete) data on branches.
To maintain (enter, update, and delete) data on staff.
To maintain (enter, update, and delete) data on properties for rent.
To maintain (enter, update, and delete) data on property owners.
To maintain (enter, update, and delete) data on clients.
To maintain (enter, update, and delete) data on property viewings.
To maintain (enter, update, and delete) data on leases.
To maintain (enter, update, and delete) data on newspaper adverts.

To perform searches on branches.
To perform searches on staff.
To perform searches on properties for rent.
To perform searches on property owners.
To perform searches on clients.
To perform searches on property viewings.
To perform searches on leases.
To perform searches on newspaper adverts.

To track the status of property for rent.
To track the status of clients wishing to rent.
To track the status of leases.

To report on branches.
To report on staff.
To report on properties for rent.
To report on property owners.
To report on clients.
To report on property viewings.
To report on leases.
To report on newspaper adverts.

System boundary for the Case Study:



ENTITY-RELATIONSHIP MODELING:

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 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.

An entity is represented by a set of attributes. **Attributes** are descriptive properties possessed by each member of an entity set.

Attribute domain: The set of allowable values (data type) for one or more attributes.

Types of Attributes:

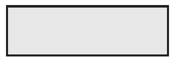
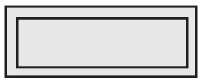
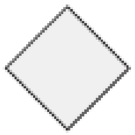
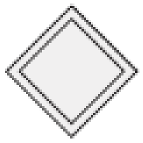
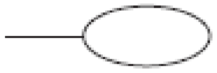
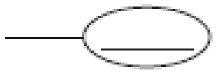
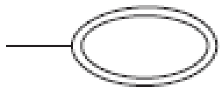
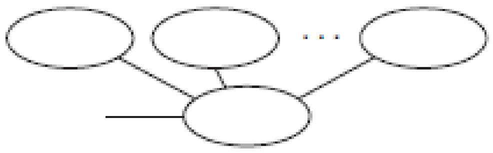
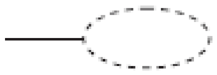
Simple Attribute: An attribute composed of a single component with an independent existence.

Composite Attribute: An attribute composed of multiple components, each with an independent existence.

Single-Valued Attributes: An attribute that holds a single value for each occurrence of an entity type.

Multi-Valued Attributes: An attribute that holds multiple values for each occurrence of an entity type.

Derived Attributes: An attribute that represents a value that is derivable from the value of a related attribute or set of attributes, not necessarily in the same entity type.

Symbol	**Meaning**
	Entity
	Weak Entity
	Relationship
	Indentifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute

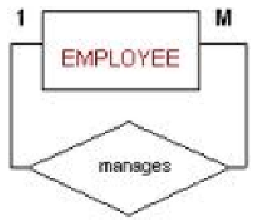
A **relationship** is an association among several entities. Relationship set is a set of relationships of the same type.

The association between entity set is referred to as **participation**. That is, the entity sets E1, E2, . ..,En participate in relationship set R.

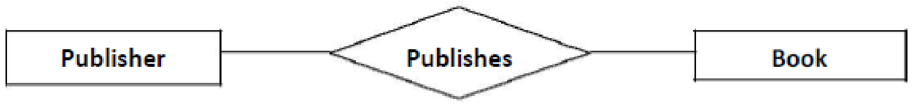
A uniquely identifiable association that includes one occurrence from each participating entity type. A relationship occurrence indicates the particular entity occurrences that are related. Relationship type and Relationship occurrences are one and the same.

Degree of a relationship: The number of participating entity types in a relationship. Binary relationship set is of degree 2; a tertiary relationship set is of degree 3.

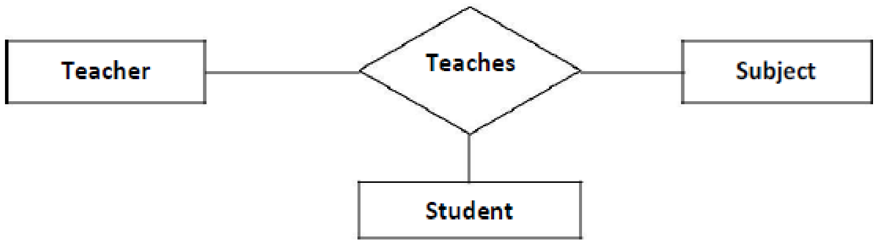
Unary relationship: A unary relationship exists when an association is maintained within a single entity.



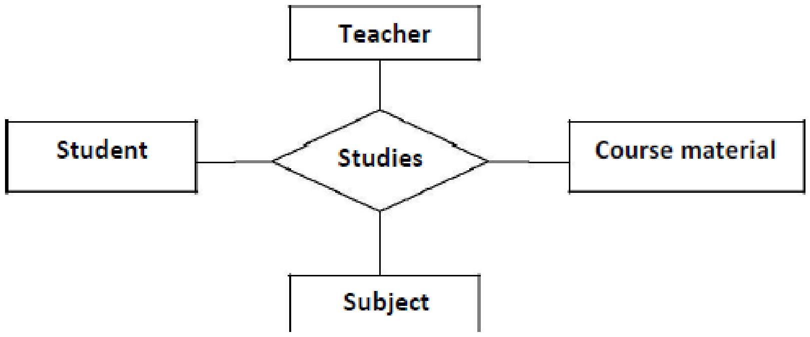
Binary relationship: A binary relationship exists when two entities are associated.



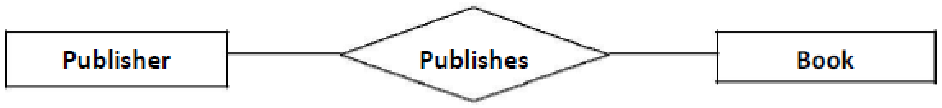
Ternary relationship: A ternary relationship exists when there are three entities associated.



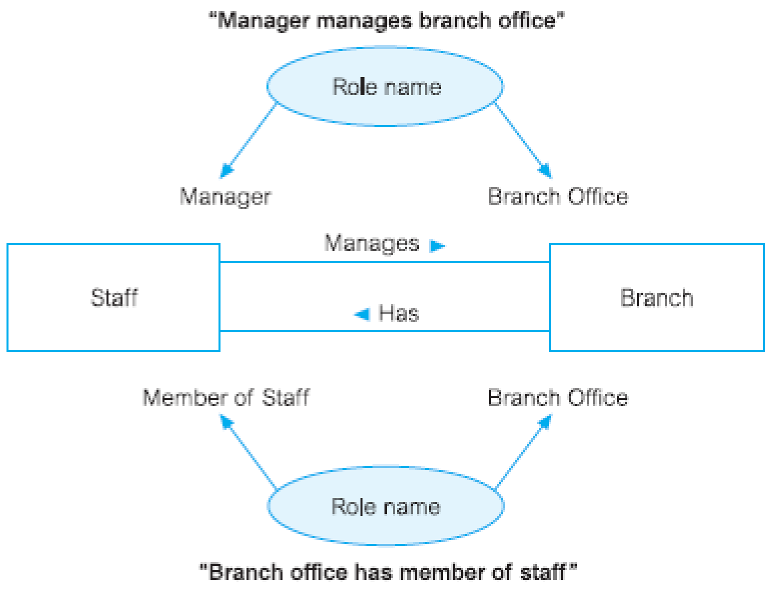
Quaternary relationship: A quaternary relationship exists when there are four entities associated.



Entity role: The function that an entity plays in a relationship is called that entity's role. A role is one end of an association. In the below ER model, the publisher entity plays the publishes role.



Recursive Relationship: A relationship type in which the same entity type participates more than once in different roles.



Keys:

Super Key: Super key is a single attribute or a group of multiple attributes that can uniquely identify each occurrence of an entity type.

Candidate Key: The minimal set of attributes that uniquely identifies each occurrence of an entity type.

Primary Key: The candidate key that is selected to uniquely identify each occurrence of an entity type.

Composite Key: A candidate key that consists of two or more attributes.

Foreign Key: Foreign key is an attribute which is a Primary key in its parent entity, but is included as an attribute in another entity. A Foreign key generates a relationship between the parent entity and the child entity.

Alternate or Secondary Key: Alternate keys are those candidate keys which are not the Primary key.

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**.

Weak entity set is associated with another entity set called the identifying or owner entity set. i.e., weak entity set is said to be existence dependent on the identifying entity set. Identifying entity set is said to own the weak entity set.

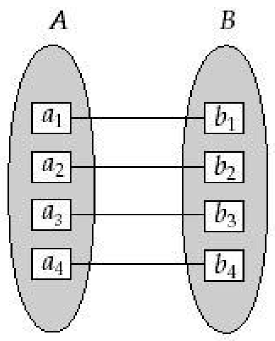
The relationship among the weak and identifying entity set is called the **identifying relationship**.

STRUCTURAL CONSTRAINTS:

Multiplicity: The number (or range) of possible occurrences of an entity type that may relate to a single occurrence of an associated entity type through a particular relationship.

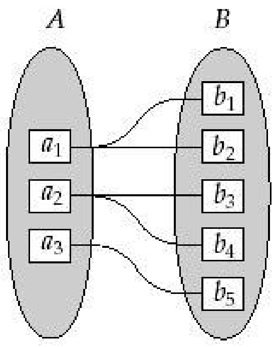
- one-to-one (1:1)
- one-tomany (1:*)
- many-to-many (*:*)

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.



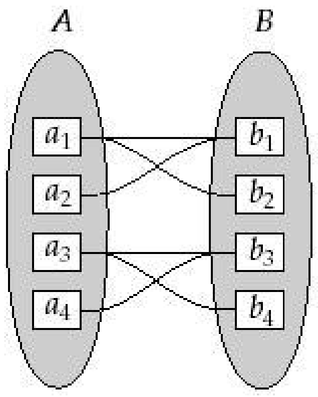
Example: Relationship between Manager and Branch of a Bank.

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



Example: Relationship between Department and Employee.

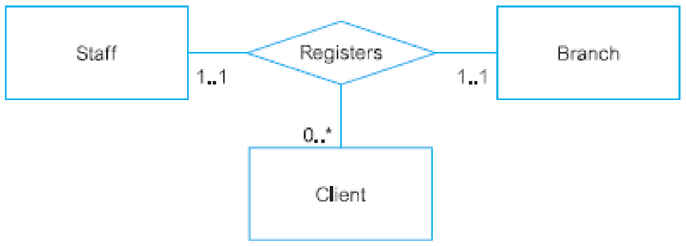
Many-to-many: An entity in A is associated with any number (zero or more) of entities in B, and an entity in B is associated with any number (zero or more) of entities in A.



Example: Relationship between Supplier and Products.

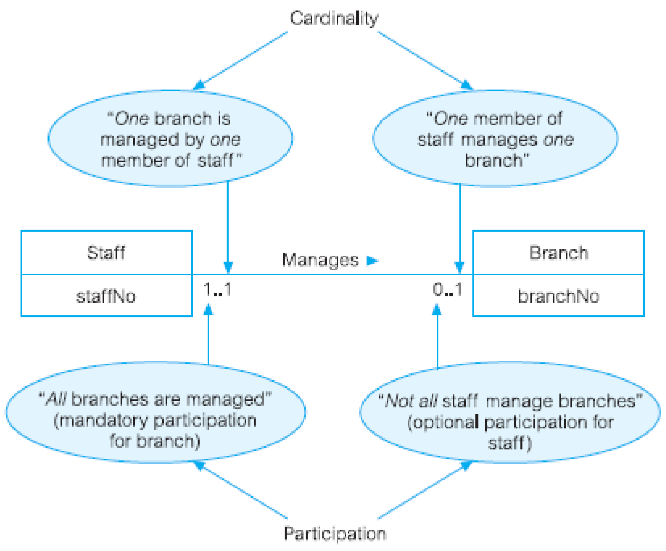
Cardinality and Participation Constraints: Multiplicity actually consists of two separate constraints known as cardinality and participation.

Cardinality: Describes the maximum number of possible relationship occurrences for an entity participating in a given relationship type.



Participation: Determines whether all or only some entity occurrences participate in a relationship

The participation constraint represents whether all entity occurrences are involved in a particular relationship (referred to as mandatory participation) or only some (referred to as optional participation).


Cardinality
"*One* branch is managed by *one* member of staff"
"*One* member of staff manages *one* branch"
Staff
staffNo
Manages ►
Branch
branchNo
1..1
0..1
"*All* branches are managed" (mandatory participation for branch)
"*Not all* staff manage branches" (optional participation for staff)
Participation

Problems with ER Models:

Fan Trap: Where a model represents a relationship between entity types, but the pathway between certain entity occurrences is ambiguous.

Chasm Trap: Where a model suggests the existence of a relationship between entity types, but the pathway does not exist between certain entity occurrences

ENHANCED ER MODEL

It is a diagrammatic technique for displaying the following concepts

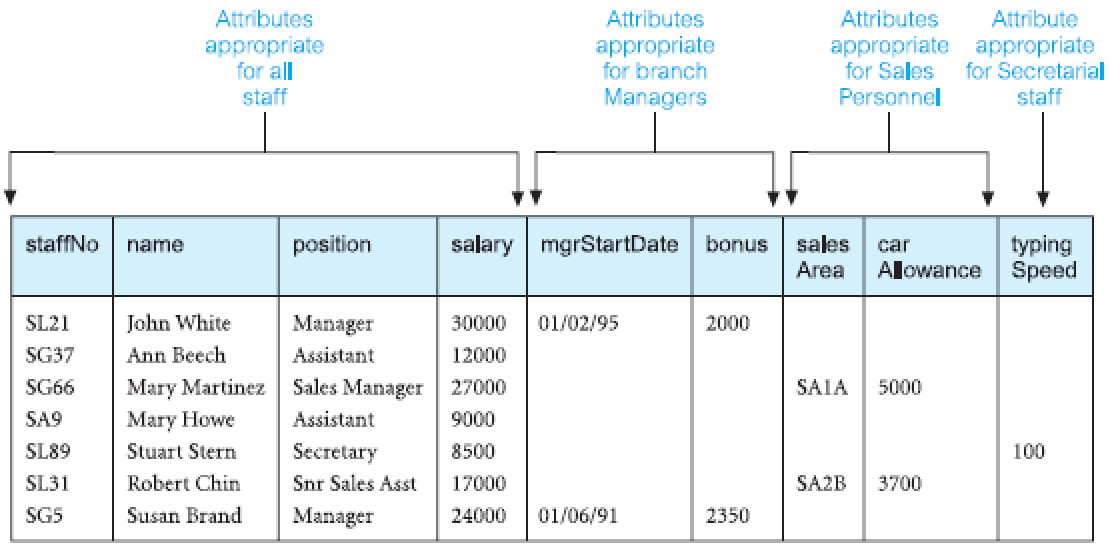
- Sub Class and Super Class
- Specialization and Generalization
- Aggregation
- Composition

Super Class: An entity type that includes one or more distinct subgroupings of its occurrences, which must be represented in a data model.

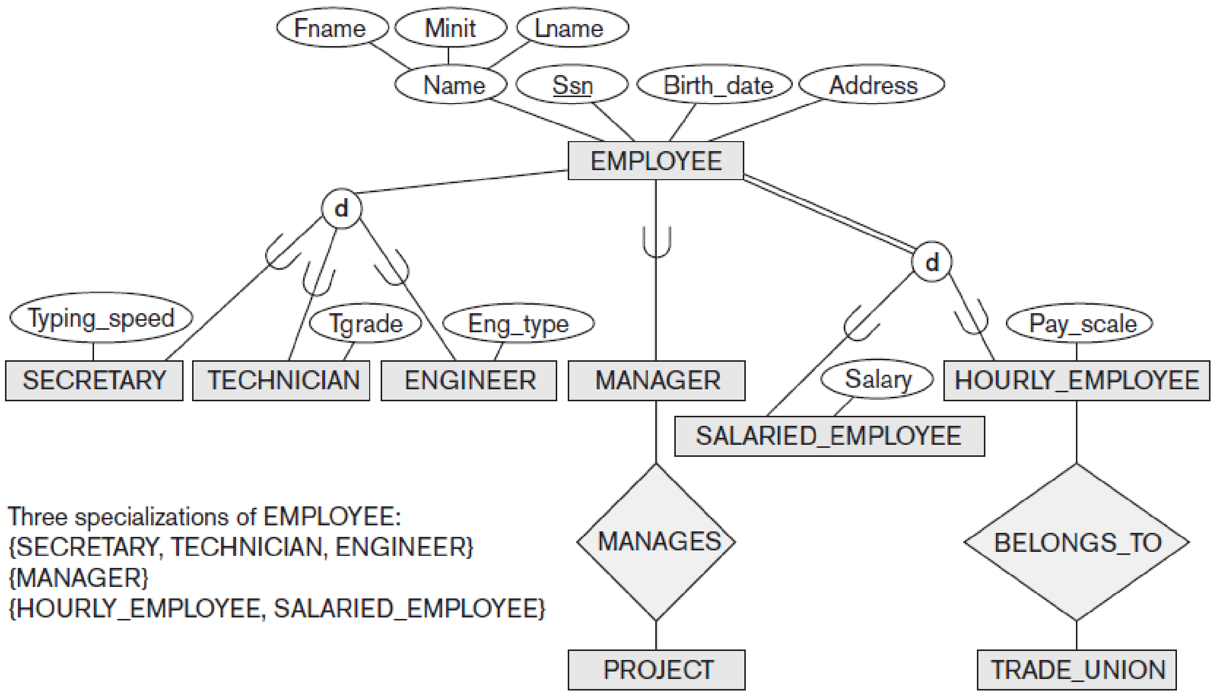
Sub Class: A distinct subgrouping of occurrences of an entity type, which must be represented in a data model.

Specialization: Specialization is the process of defining a set of subclasses of an entity type. The process of maximizing the differences between members of an entity by identifying their distinguishing characteristics. The set of subclasses that forms a specialization is defined on the basis of some distinguishing characteristic of the entities in the superclass.

Generalization: A reverse process of abstraction. Suppress the differences among several entity types. The process of minimizing the differences between entities by identifying their common characteristics. Identify their common features, and generalize them into a single superclass of which the original entity types are special subclasses.

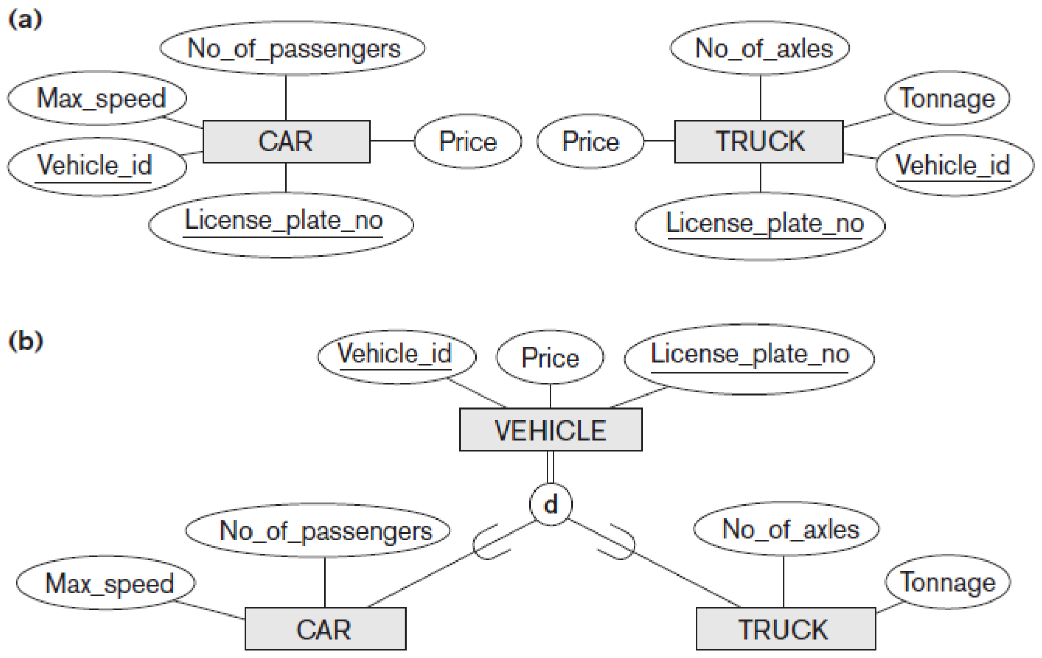

Attributes appropriate for all staff
Attributes appropriate for branch Managers
Attributes appropriate for Sales Personnel
Attribute appropriate for Secretarial staff

staffNo	name	position	salary	mgrStartDate	bonus	sales Area	car Allowance	typing Speed
SL21	John White	Manager	30000	01/02/95	2000			
SG37	Ann Beech	Assistant	12000					
SG66	Mary Martinez	Sales Manager	27000			SA1A	5000	
SA9	Mary Howe	Assistant	9000					
SL89	Stuart Stern	Secretary	8500					100
SL31	Robert Chin	Snr Sales Asst	17000			SA2B	3700	
SG5	Susan Brand	Manager	24000	01/06/91	2350			



Three specializations of EMPLOYEE:
{SECRETARY, TECHNICIAN, ENGINEER}
{MANAGER}
{HOURLY_EMPLOYEE, SALARIED_EMPLOYEE}

Generalization. (a) Two entity types, CAR and TRUCK.
(b) Generalizing CAR and TRUCK into the superclass VEHICLE.



Constraints on Specialization/Generalization: In some specializations we can determine exactly the entities that will become members of each subclass by placing a condition on the value of some attribute of the superclass. Such subclasses are called predicate-defined (or condition-defined) subclasses.

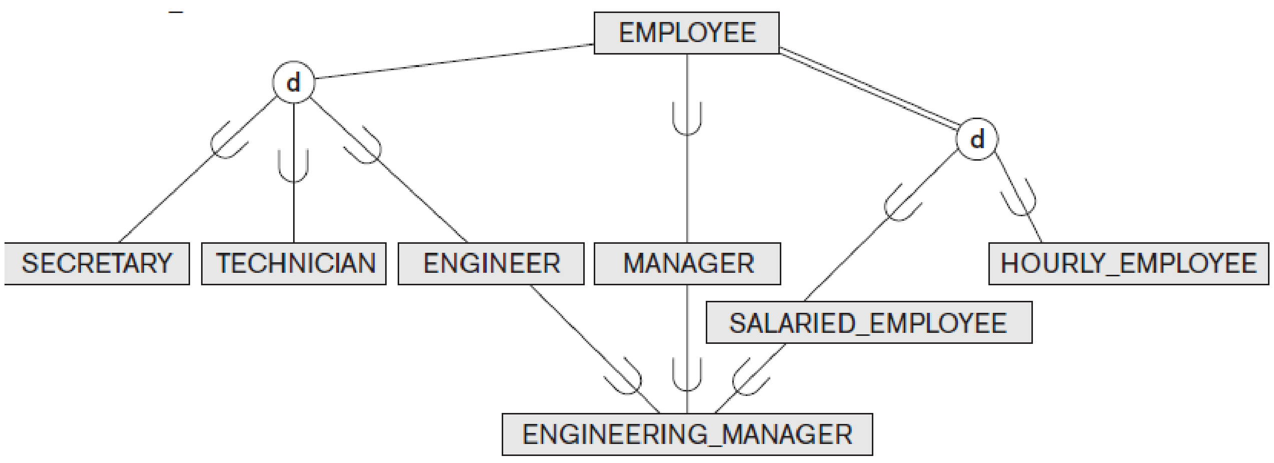
- **Participation Constraints:** Determines whether every member in the superclass must participate as a member of a subclass
- **Disjoint Constraints:** Describes the relationship between members of the subclasses and indicates whether it is possible for a member of a superclass to be a member of one, or more than one, subclass.

If all subclasses in a specialization have their membership condition on the same attribute of the superclass, the specialization itself is called an attribute-defined specialization. The attribute is called the defining attribute of the specialization. When we do not have a condition for determining membership in a subclass, the subclass is called user-defined subclass.

A total specialization constraint specifies that every entity in the superclass must be a member of at least one subclass in the specialization. A partial specialization, which allows an entity not to belong to any of the subclasses.

A specialization Lattice:

A set of entities that are at two or more levels is called as a specialization lattice. It is also called as multi-level specialization.



A specialization lattice with shared subclass
ENGINEERING_MANAGER.

Aggregation:

- Aggregation is an abstraction concept for building composite objects from their component objects.
- Represents a "has-a" or "is-part-of" relationship between entity types, where one represents the "whole" and the other the "part."

  Three Cases

  - We aggregate attribute values of an object to form the whole object
  - We represent an aggregation relationship as an ordinary relationship
  - Combining objects that are related by a particular relationship instance into a *higher-level aggregate object*
    - *IS-A-PART-OF and IS-A-COMPONENT-OF*

Composition:

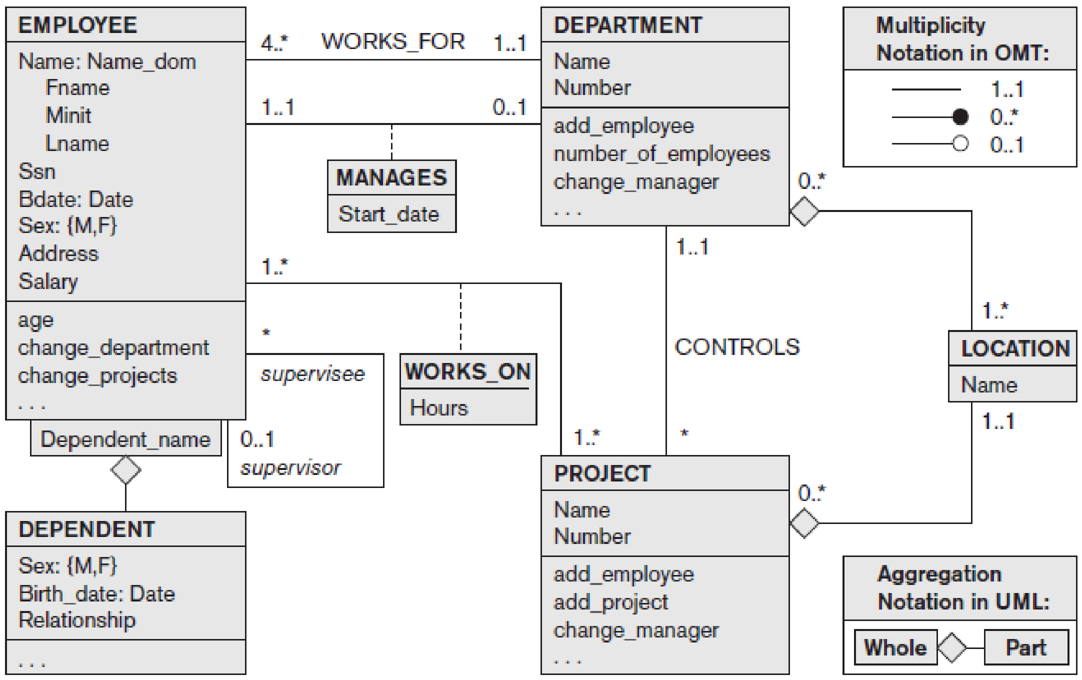
A specific form of aggregation that represents an association between entities, where there is a strong ownership and coincidental lifetime between the "whole" and the "part."

Eg.: A newspaper displays an advertisement

UML Class Diagram:

In UML class diagrams,

- a class (similar to an entity type in ER) is displayed as a box that includes three sections:
  - The top section gives the class name (similar to entity type name)
  - the middle section includes the attributes
  - last section includes operations that can be applied to individual objects
- Operations are not specified in ER diagrams.



- The designer can optionally specify the domain (or data type) of an attribute if desired, by placing a colon (:) followed by the domain name or description
- A composite attribute is modeled as a structured domain
- A multivalued attribute will generally be modeled as a separate class
- Relationship types are called associations in UML terminology, and relationship instances are called links

Associations:

A binary association is represented as a line connecting the participating classes, and may optionally have a name

- A relationship attribute, called a link attribute, is placed in a box that is connected to the association's line by a dashed line
- The (min, max) notation is used to specify relationship constraints, which are called multiplicities in UML terminology
  - Multiplicities are specified in the form *min..max*, and an asterisk (*) indicates no maximum limit on participation

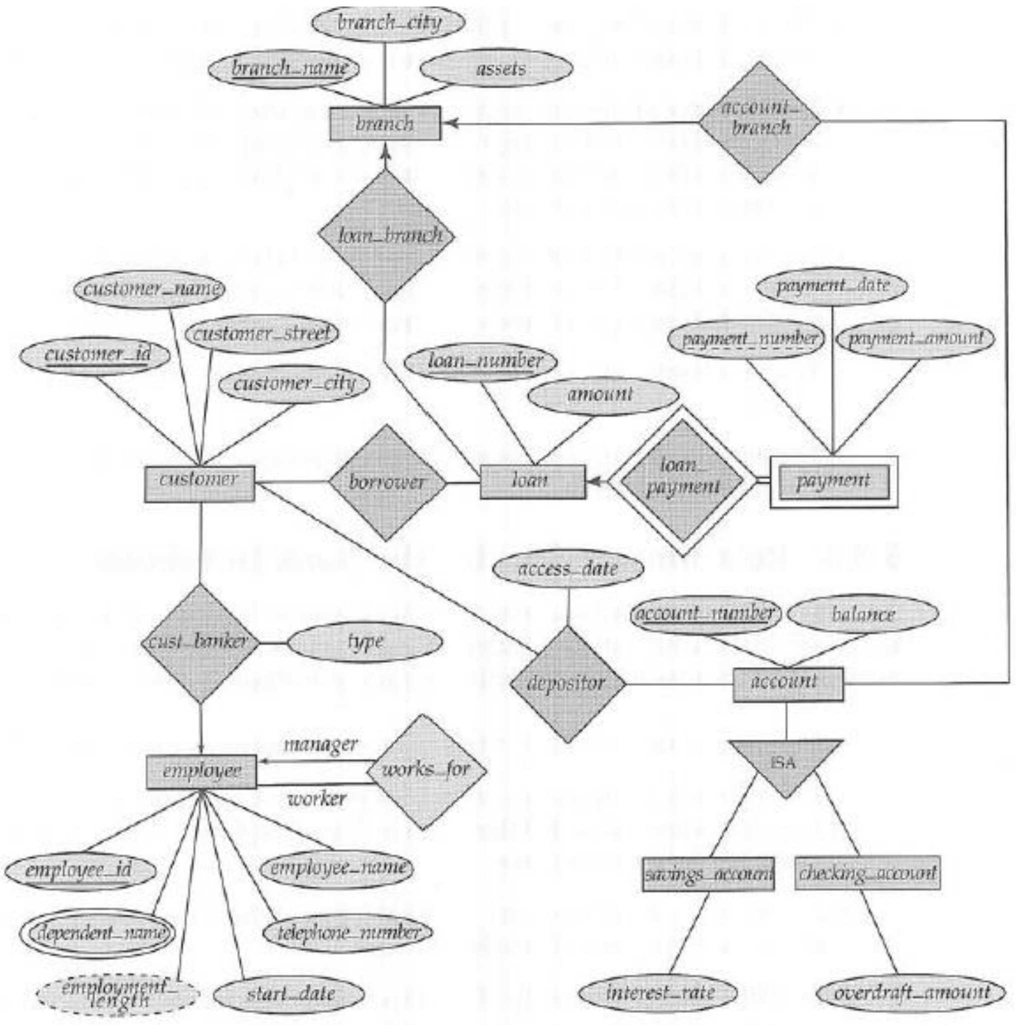
In UML, there are two types of relationships:

- Association
- Aggregation

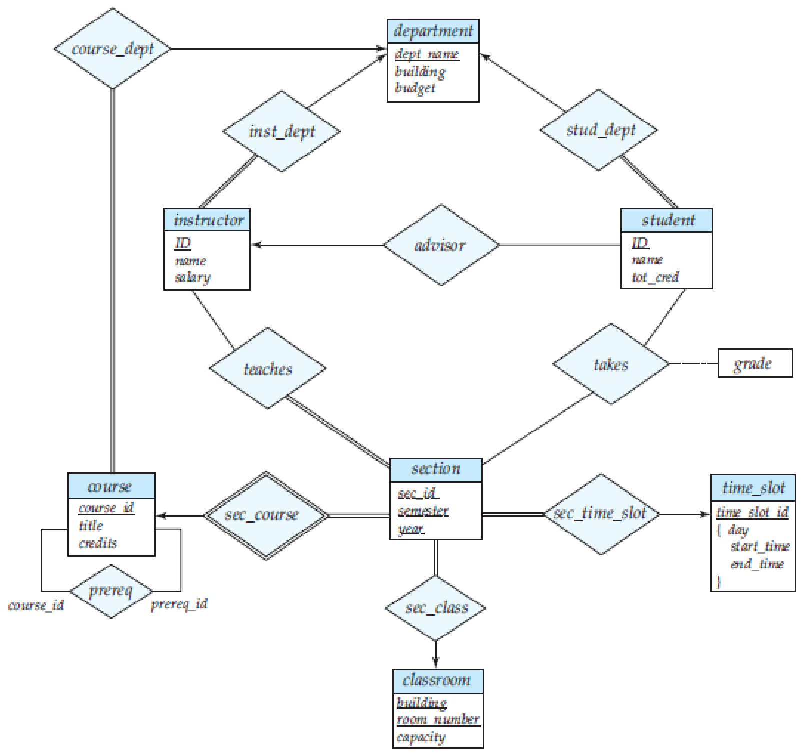
Aggregation is meant to represent a relationship between a whole object and its component parts, and it has a distinct diagrammatic notation.

UML also distinguishes between unidirectional and bidirectional associations/aggregations. In the unidirectional case, the line connecting the classes is displayed with an arrow to indicate that only one direction for accessing related objects is needed. If no arrow is displayed, the bidirectional case is assumed, which is the default. In addition, relationship instances may be specified to be ordered.

ER Model for a Banking System:



ER Model for a University System


course_dept
department
dept_name
building
budget
inst_dept
stud_dept
instructor
ID
name
salary
advisor
student
ID
name
tot_cred
teaches
takes
grade
section
sec_id
semester
year
course
course_id
title
credits
sec_course
sec_time_slot
time_slot
time_slot_id
{ day
start_time
end_time
}
prereq
course_id
prereq_id
sec_class
classroom
building
room_number
capacity