**DATABASE MANAGEMENT SYSTEM**

**UNIT-I**

**Syllabus**

Introduction: An overview of database management system, database system Vs file system, Characteristics of database approach, DBMS architecture, data models, schema and instances, data independence.

Data Modeling using Entity Relationship Model: Entity, Entity types, entity set, notation for ER diagram, attributes and keys, Concepts of composite, derived and multivalued attributes, Super Key, candidate key, primary key, relationships, relation types, weak entities, enhanced E-R and object modeling, Sub Classes:, Super classes, inheritance, specialization and generalization.

**Introduction**

Database systems have become an essential component of everyday life in modern society and in that many frequently occurring activities involve the accessing of at least one database. e.g **in Finance, in University, in library searches, in bank transactions, in hotel/airline/Railway reservations, in grocery store purchases, in online (Web) purchases** etc.

**Types of Databases and Database Applications**

* A traditional database applicationwhere most of the information is stored and accessed is either **numeric or textual**.
* More recent applications of databases: Recent advances have led to the application of database technology to a wider class of data:
* **Multimedia** Databases involving pictures, video clips, and sound messages.
* **Geographic Information Systems** (GIS) involving maps, satellite images.
* **Analytical Database:** Involves Data Warehouses & on-line analytical processing (OLAP) systems that are used in many companies to extract and analyse useful information from very large database for decision making.
* **Operational Database:** On-line Transaction processing systems(OLTP) to modify the data online usually track real time information
* **Real-time and Active Databases** is used in controlling industrial and manufacturing processes.

**Definitions**

**Data:** Data are facts that can be collected through observation & measurement such as amount, qty, rollno, name, telephone no etc. and recorded which have some implicit meaning. The data are organized in the form of characters, fields, record, files and databases. There are 2 types of data:

* It is the collection of information needed by the organization.
* Metadata - is information about the data i.e data about data.

**Information:** It refers to the processed data.

**Knowledge:** It refers to the ability to use information to achieve the desired needs/results.

**File:** A file is a collection of information/ sequence of records.

**Databases:**

* A database is a collection of related files that are usually logically related to each other, designed to meet the information needs of an organisation. In other words, it is a collection of all inter-related data.
* The major feature of database is that the actual data are separated by the programs that use that data.
* It represents some aspect of the real (or an imagined) world, called the **miniworld** or **Universe of Discourse (UOD)**. Changes to the miniworld are reflected in the database. For example, a UNIVERSITY miniworld concerned with students, courses, course sections, grades, and course prerequisites.

**Computerized vs. manual database**

**A** computerized database system may be created and maintained either by a group of application programs or by database management systems.

**A** manual system on the other hand may be generated and maintained manually. E.g. the library card catalog-based systems used in library.

**Size and Complexity of the database**

A database can be of any size and of varying complexity. e.g. one person's recipe database) to being huge/complex (e.g., Amazon's database that keeps track of all its products, customers, and suppliers now contains storage above 2 TB).

**Database Vs File System**

In the database approach, a single repository of data is maintained that is defined once and then accessed by various users. While in the file processing system, any changes to the structure of a file may require changing all programs that access the file.

**Database Management System**

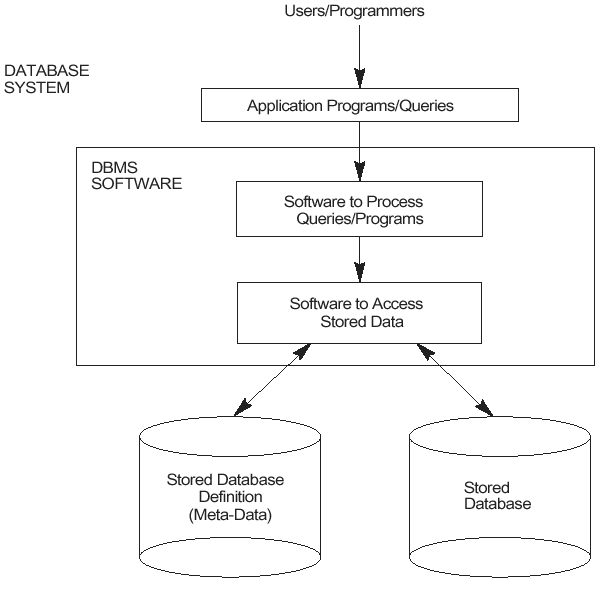
**Definition:** A **database management system** (DBMS) is a collection of interrelated data and a set of programs to access those data. Hence a DBMS is a *general purpose* software system that provides an environment that is ***both convenient and efficient*** to use in each of the following (with respect to a database):

1. **Defining** a database involves specifying the data types, structures and constraints for the data to be stored in the database.
2. **Constructing** a database is the process of storing the data on some medium (e.g., magnetic disk) that is controlled by the DBMS.
3. **Manipulating** a database includes functions such as
4. Querying the database to retrieves some specific data
5. Updating the database i.e insertions, deletions and modifications to its content that reflects changes in the mini world.
6. Generating reports from data.
7. **Sharing** allowing multiple users and programs to access the database "simultaneously".
8. **System protection** preventing database from becoming corrupted when hardware or software failures occur.
9. **Security protection** preventing unauthorized or malicious access to database.

**Database system**

A database together with the DBMS software is referred to as a **database system** shown in fig 1.1 below:

**Figure 1.1: A simplified database system environment**



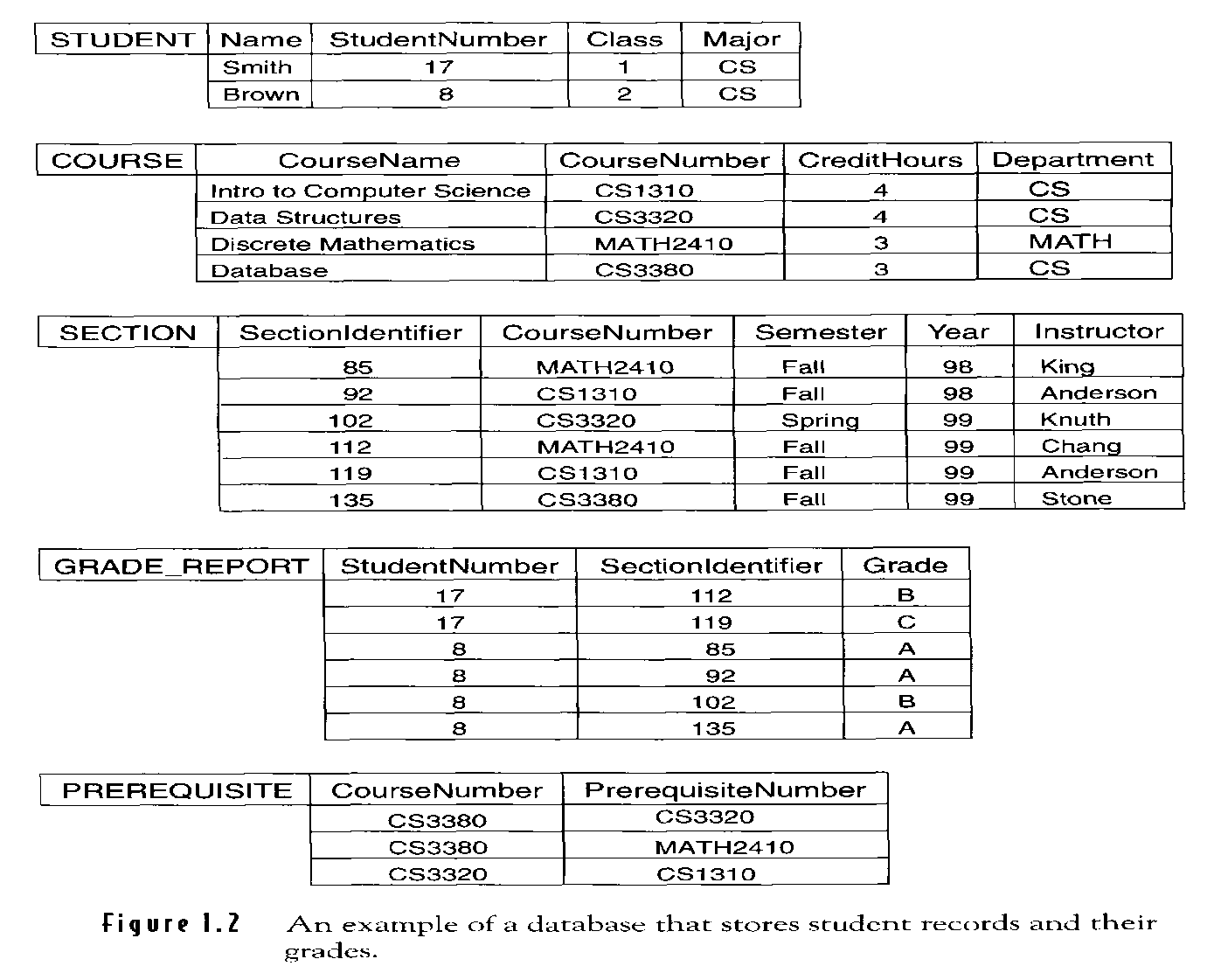
**Mini-world:** Some part of the real world about which data is stored in a database. Eg. student grades and transcripts at a university.

**Example of a Database (with a Conceptual Data Model)**

* Mini-world for the example:
* Part of a UNIVERSITY environment.
* Some mini-world *entities*:
* STUDENTs
* COURSEs
* SECTIONs (of COURSEs)
* (academic) DEPARTMENTs
* INSTRUCTORs

**Some mini-world *relationships*:**

* SECTIONs *are of specific* COURSEs
* STUDENTs *take* SECTIONs
* COURSEs *have prerequisite* COURSEs
* INSTRUCTORs *teach* SECTIONs
* COURSEs *are offered by* DEPARTMENTs
* STUDENTs *major in* DEPARTMENTs



**DBMS Vs File Processing System(FPS)**

**Question: Disadvantages of File Processing System/Advantages of DBMS**

**Advantages of using a DBMS**

* **Controlling the Data Redundancy:** Data redundancy means storing same data multiple number of times (normally occurs in the "file processing" approach) leads to the W**asted Storage Space**, D**uplication of effort** (when multiple copies of a datum need to be updated), and data **inconsistency**.

A DBMS should provide the capability to automatically enforce the rule that no inconsistencies are introduced when data is updated. Most DBMS provide facilities for controlling the data redundancy using the concepts of keys and normalization.

* **Restricting Unauthorized Access:** A DBMS should provide a **security and authorization subsystem**, which is used for specifying restrictions on user accounts. Common kinds of restrictions are to allow read-only access (no updating), or access only to a subset of the data.
* **Providing Persistent Storage for Program Objects:** Object-oriented database systems make it easier for complex runtime objects (e.g., lists, trees) to be saved in secondary storage so as to survive beyond program termination and to be retrievable at a later time.
* **Permitting Inferencing and Actions Via Rules:** Some database systems provide capabilities for defining deduction rules for inferencing new information from the stored database. Such system is called **deductive** database system; one may specify *declarative* rules that allow the database to infer new data.
* **Providing Multiple User Interfaces: As lots of** different users with various levels of technical knowledge use a DBMS. Hence a DBMS should provide a variety of user interfaces e.g Query languages for casual users, Programming language interfaces for application programmers, forms and/or command codes for parametric users, menu-driven interfaces for stand-alone users.
* **Representing Complex Relationships Among Data:** A database may include numerous varieties of data that are interrelated in many ways. A DBMS should have the capability to represent such relationships and to retrieve related data quickly.
* **Enforcing Integrity Constraints:** Most database applications have certain integrity constraints that must hold for data. A DBMS should provide the capabilities for defining and enforcing these constraints. Perhaps the most fundamental constraint on a data item is its data type, which specifies the universe of values from which its value may be drawn.

**e**.g., a **Grade** field could be defined to be of type Grade Type, which, say, we have defined as including precisely the values in the set { "A", "A-", "B+", ..., "F" }. **Age** field could be defined to be of type **int** which including the values between 18 & 65.

* **Providing Backup and Recovery:** The subsystem having this responsibility ensures that recovery is possible in the case of a system crash during execution of one or more transactions.

**Characteristics of the Database Approach**

A number of characteristics distinguish the database approach from the traditional approach of programming with files. In traditional file processing, each user defines and implements the files needed for a specific software application as part of programming the application. In the database approach, a single repository of data is maintained that is defined once and then is accessed by various users. The main characteristics of the database approach versus the file-processing approach are the following:

* Self-describing nature of a database system
* Insulation between programs and data, and data abstraction
* Support of multiple views of the data
* Sharing of data and multiuser transaction processing

**File Processing Systems (FPS) vs. DBMS**

|  |  |  |
| --- | --- | --- |
| **Characteristics** | **File systems** | **DBMS** |
| **1.Self-describing nature.** | In traditional file processing system data definition is the part of the application program.  These programs contain only one specific database, whose structure is declared in the application program. E.g in C program using structure we can create a record. | In Database Management System a single data repository (i.e. Database) is used. So, redundancy is reduced.  The structure and constraints of the database are defined in the **DBMS catalog.** Itcontains the information such as i) the structure of each file ii) the type iii) storage format of each data item and iv) various constraints on the data.  The information stored in the catalog is  called **meta-data**, and it describes the structure of the primary database. |
| **2.Insulation between Programs and Data, and Data Abstraction** | In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access this file. | Whereas in DBMS access programs do not require such changes in most cases.  The structure of data files is stored in the DBMS catalog separately from the access programs and this property is called P**rogram Data Independence**.  **Program operation independence:**  In some types of database systems, such as object-oriented, a user can define operations on data as part of the database definition.  User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed program-operation independence.  **Data abstraction:**  The characteristic that allows P**rogram Data Independence** and **Program Operation Independence** is called Data Abstraction. |
| **Support of multiple views of data** | All users could see all the data. | The DBMS may allow different users to see different "views" of the DB, according to the perspective each one requires.  **A view** may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored. |
| **Data sharing and Multiuser Transaction Processing** | Inconsistency could result due to uncontrolled concurrent multiuser access. | A multiuser DBMS must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and maintained in a single database.  The DBMS must include   * **Concurrency control** within the DBMS guarantees that each transaction is correctly executed or aborted. * **Recovery subsystem** ensures each completed transaction has its effect permanently recorded in the database. * **OLTP** (Online Transaction Processing) is a major part of database applications (allows hundreds of concurrent transactions to execute per second) |

**DBMS ARCHITECTURE**

* The idea was first described by the ANSI/SPARC committee in late 1970's.
* Also called **ANSI/SPARC architecture** or **Three-Schema architecture**.
* The purpose of DBMS is to provide user interface with an abstract view of the data called **Data Abstraction**.
* A DBMS can be organized into 3 separate levels of abstraction.

1. **Internal Level or Physical Level**

* It is the lowest level of abstraction that describe internal schema i.e how

the data are actually stored in the database.

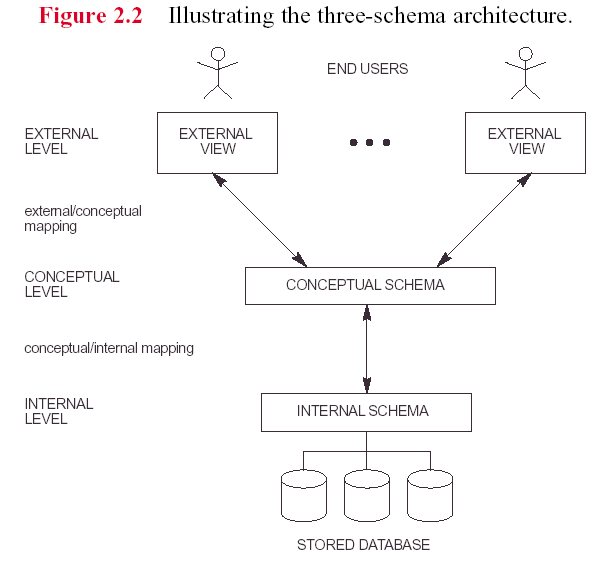
* It uses a physical data model.
* Concerned with the physical storage, actual bytes (size) for data items, access mechanism (sequential, random, indexed) etc.
* Data encryption & data decryption techniques.

1. **Conceptual Level or Logical Level**

* It describes the structure and constraints of the entire DB for all users—***what*** type of data is stored in the Database.
* It uses a *conceptual* or an *implementation* data model.

1. **External Level or View Level**

* It is the highest level of abstraction.
* It describes the user views.
* Typically uses the same data model as the conceptual level.



**Mappings**

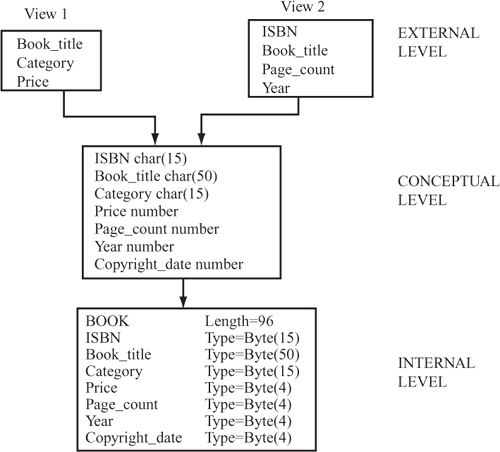
* The process of transforming requests and results between levels are called mapping. Programs refer to an *external* schema, and are mapped by the DBMS to the internal schema for execution. 2 types of **mappings** are required in a database system with 3 different views:
* mappingbetween **external/conceptual**  (providing **logical** data independence)
* mappingbetween **conceptual/internal** (providing **physical** data independence)

The main advantage of Three-schema architecture is that it provides data independence. ***Data independence*** is the ability to change the schema at one level of the database system without having to change the schema at the other levels. Data independence is of two types, namely, ***logical data independence***and***physical data independence***.

* **Logical data independence**: It is the ability to change the conceptual schema without affecting the external schemas or application programs. The conceptual schema may be changed due to change in constraints or addition of new data item or removal of existing data item, etc., from the database. The separation of the external level from the conceptual level enables the users to make changes at the conceptual level without affecting the external level or the application programs. For example, if a new data item, say **Edition** is added to the **BOOK** file, the two views (*view 1* and *view 2)* shown in the [Figure below](javascript:moveTo('ch01fig02');) are not affected.
* **Physical data independence**: It is the ability to change the internal schema without affecting the conceptual or external schema. An internal schema may be changed due to several reasons such as for creating additional access structure, changing the storage structure, etc. The separation of internal schema from the conceptual schema facilitates physical data independence.

To understand the **three-schema architecture**, consider the three levels of the **BOOK** file in *Online Book* database as shown in figure below. In this figure, two views (*view 1* and *view 2*) of the **BOOK** file have been defined at the external level. Different database users can see these views. The details of the data types are hidden from the users. At the conceptual level, the **BOOK** records are described by a type definition. The application programmers and the DBA generally work at this level of abstraction. At the internal level, the **BOOK** records are described as a block of consecutive storage locations such as words or bytes. The database users and the application programmers are not aware of these details; however, the DBA may be aware of certain details of the physical organization of the data.

**Fig. Three levels of Online Book database (BOOK file)**



Logical data independence is more difficult to achieve than the physical data independence because the application programs are always dependent on the logical structure of the database. Therefore, the change in the logical structure of the database may require change in the application programs.

**Advantages and disadvantages**

* The three -Schema Architecture can make it easier to achieve true data independence
* Mappings create an overhead during compilation or execution of a query or a program
* Not implemented fully by DBMSs

**DATA MODELS**

**Data Model**

* A collection of concepts/conceptual tools that can be used to describe the structure of a database means the data types, relationships, and constraints that should hold on the data.
* Also includes basic operations for specifying retrievals and updates on the database.

**Categories of Data Models**

* 1. **Conceptual or high-level data models**

Include all the major concepts and their relationship used in describing databases at the conceptual or view level. Examples are Entity-Relationship Model, Object-based data models. This model uses concepts such as

* **Entity**: Real-world object or concept (e.g., student, employee, course, department, event)
* **Attribute**: some property of interest describing an entity (e.g., height, age, color)
* **Relationship** is an interaction among entities (e.g., a works-on relationship between an employee and a project; an enrolled-in relationship between a student and a course section)
  1. **Physical or low-level data models**
* Provide concepts that describe details of how data is stored in the computer.
* Meant for computer specialists, not for typical end users.
  1. **Implementation or representational data models**
* Provide concepts that fall between the above two, balancing user views with some computer storage details.
* Most frequently used in traditional commercial DBMSs.
* Examples are Relational Data Model, Network Data Model & Hierarchical Data Model where data and relationships are represented by a collection of tables, links/pointers and tree respectively.

**Schemas, Instances, and Database State**

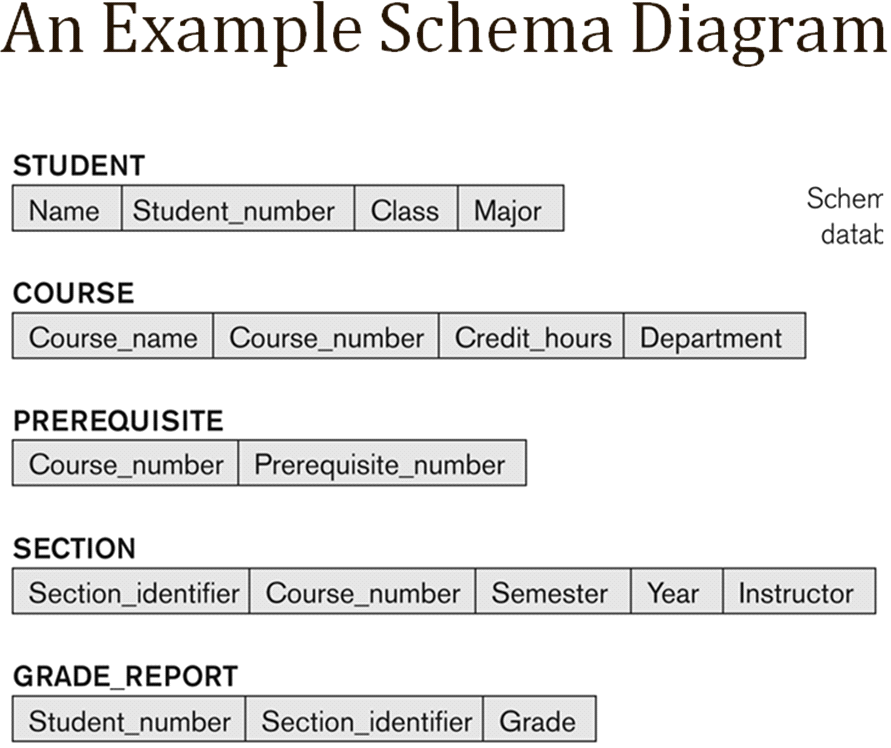
**Database Schema also called intension: T**he overall design of the database is called the database schema. i.e the description of a database and is not expected to change frequently.

**Schema Diagram**

* A diagrammatic display of (some aspects of) a database schema.

**Instances/Database State is also called extension i.e**

* The data in the database at a particular moment of time
* Also called the current set of occurrences or instances
* Every update operation changes the database from one state to another



**Distinction between Database Schema & Database State**

- The database schema changes very infrequently. The database state changes every time the

database is updated.

- Schema is also called intension. State is also called extension.

**Database Users**

**Users may be divided into:**

1. **Actors on the Scene-** Those who actually use and control the content. The jobs involve the day-to-day use of a large database. They are:

* **Database Administrators**
* **Database Designers**
* **End Users**
* **System Analysts and Application Programmers (Software Engineers)**
* **Database Administrators - DBA is a kind of database user with the following responsibilities:**

1. The primary responsibility of the D**atabase Administrator (DBA)** is to observe and manage (administer) the various resources (database itself, DBMS software and other related software).
2. He is responsible for authorizing access to the database, for coordinating and monitoring its use, and for acquiring software and hardware resources as needed.
3. The DBA is accountable for problems such as breach of security or poor system response time.

* **Database Designers- They** are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data. These tasks are mostly undertaken before the database is actually implemented and populated with data. It is the responsibility of database designers to communicate with all prospective database users, in order to understand their requirements, and to come up with a design that meets these requirements.
  + **End Users-** End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use. There are several categories of end users:

• **Casual end users** occasionally access the database, but they may need different information each time. They use a sophisticated database query language to specify their requests and are typically middle- or high-level managers or other occasional browsers.

• **Naive** or **parametric end users** their main job is to querying and updating the database using standard types of queries and updates—called **canned transactions—**that have been carefully programmed and tested. Eg Bank tellers check account balances and post withdrawals and deposits, reservation clerks for airlines, hotels check availability for a given request and make reservations.

• **Sophisticated end users** include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS so as to implement their applications to meet their complex requirements.

• **Stand-alone users** maintain personal databases by using ready-made program packages that provide easy-to-use menu- or graphics-based interfaces. An example is the user of a ***tax package*** that stores a variety of personal financial data for tax purposes, ***Tally***, a financial accounting package.

* **System Analysts and Application Programmers (Software Engineers)**

**System analysts** determine the requirements of end users, especially naive and parametric end users, and develop specifications for canned transactions that meet these requirements. **Application programmers** implement these specifications as programs; then they test, debug, document, and maintain these canned transactions. Such analysts and programmers (nowadays called **software engineers**) should be familiar with the full range of capabilities provided by the DBMS to accomplish their tasks.

1. **Workers behind the Scene :** We consider people who may be called "workers behind the scene"—those who work to maintain the database system environment, but who are not actively interested in the database itself. They include the following categories:

• **DBMS system designers and implementers** are persons who design and implement the DBMS modules and interfaces as a software package. A DBMS is a complex software system that consists of many components or **modules,** including modules for implementing the catalog, query language, interface processors, data access, concurrency control, recovery, and security. The DBMS must interface with other system software, such as the operating system and compilers for various programming languages.

• **Tool developers** include persons who design and implement **tools**—the software packages that facilitate database system design and use, and help improve performance. Tools are optional packages that are often purchased separately. They include packages for database design, performance monitoring, natural language or graphical interfaces, prototyping, simulation, and test data generation.

• **Operators and maintenance personnel** are the system administration personnel who are responsible for the actual running and maintenance of the hardware and software environment for the database system.

**The Database System Environment**

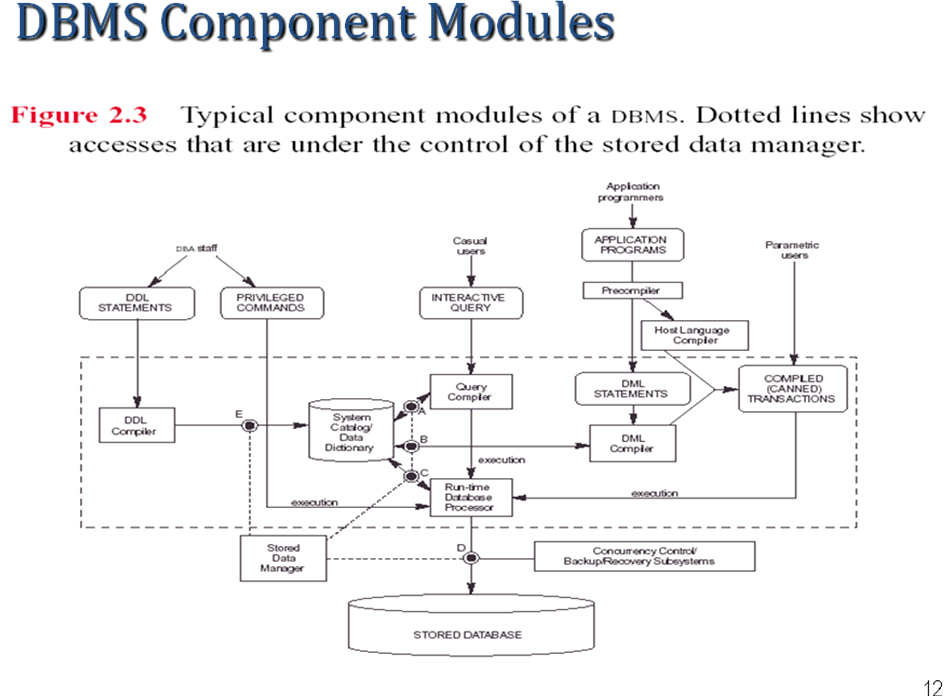
**DBMS Component Modules** The figure is divided into two halves. The top half of the figure refers to the various users of the database environment and their interfaces. The lower half shows the internals of the DBMS responsible for storage of data and processing of transaction.

The database and the DBMS catalog are usually stored on disk. Access to the disk is primarily controlled by ***operating system (OS),***which includes disk input/Output. A higher level ***stored data manager*** module of DBMS controls access to DBMS information that is stored on the disk. If we consider the top half of the figure, it shows interfaces to DBA staff, casual users, application programmers and parametric users.

The ***DDL compiler*** processes schema definitions, specified in the DDL, and stores the description of the schema in the DBMS Catalog. The catalog includes information such as names and sizes of the sizes of the files, data types of data of data items. Storage details of each file, mapping information among schemas and constraints.

Casual users and persons with occasional need of information from database interact using some form of interface which is ***interactive query interface***. The queries are parsed, analyzed for correctness of the operations for the model. The names of the data elements and so on by a ***query compiler*** that compiles them into internal form. The internal query is subjected to query optimization.The query optimizer is concerned with rearrangement and possible recording of operations, eliminations of redundancies.

Application programmer writes programs in host languages. The ***pre-compiler*** extracts DML commands from an application program



**Database System Utilities**

* ***Loading*** data stored in files into a database. Includes data conversion tools.
* ***Backing up*** the database periodically on tape.
* ***Reorganizing*** database file structures.
* ***Report generation*** utilities.
* ***Performance monitoring*** utilities.
* Other functions, such as ***sorting*, *user monitoring*, *data compression***, etc.

**Data Modeling Using the Entity-Relationship Model**

**INTRODUCTION Conceptual Database Design**

The main phases of database design are:

* **Requirements Collection and Analysis:** purpose is to produce a description of the users' requirements.
* **Conceptual Design**: purpose is to produce a *conceptual schema* for the database, including detailed descriptions of ***entity types*, *relationship types*, and *constraints*.** All these are expressed in terms provided by the data model being used.
* **Implementation**: purpose is to transform the conceptual schema (which is at a high/abstract level) into a (lower-level) representatonal/implementational model supported by whatever DBMS is to be used.
* **Physical Design**: purpose is to decide upon the internal storage structures, access paths (indexes), etc., that will be used in realizing the representational model produced in previous phase.

**Example COMPANY Database**

We need to create a database schema design based on the following (simplified) **requirements** of the COMPANY Database:

* Employees, departments, and projects
* Company is organized into departments
* Department controls several projects
* Employee: require each employee’s name, Social Security number, address, salary, sex (gender), and birth date
* Keep track of the dependents of each employee

**3.3 Entity-Relationship (ER) Model**

The ER model is a high-level conceptual data model. The ER model was introduced by Peter Chen in 1976, and is now the most widely used conceptual data model.

It represents real world situations using concepts, which are commonly used by people. It allows defining a representation of the real world at logical level.ER model has no facilities to describe machine-related aspects.

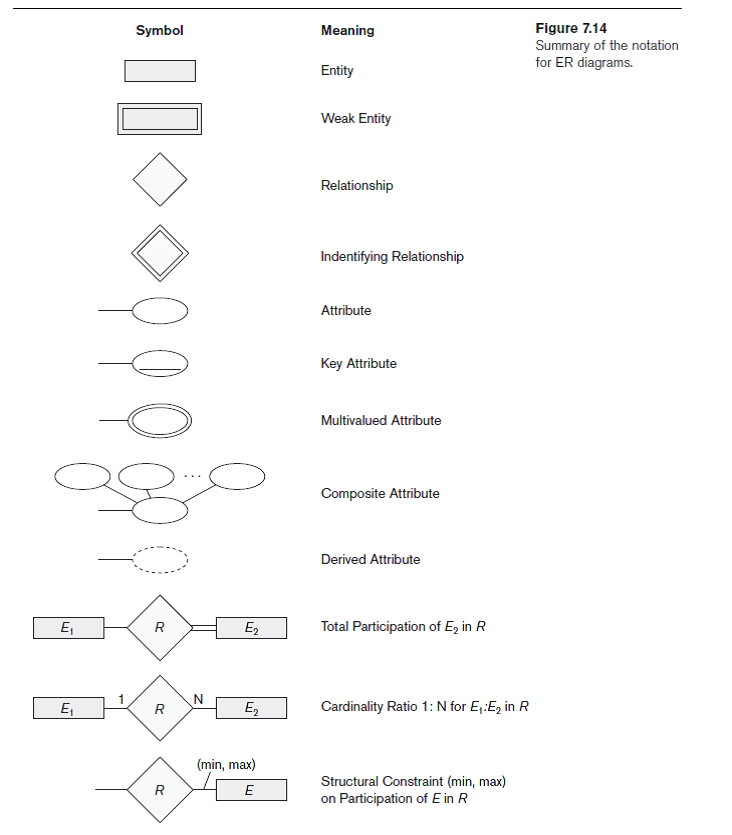
In ER model the logical structure of data is captured by indicating the grouping of data into entities. The ER model also supports a top-down approach by which details can be given in successive stages.

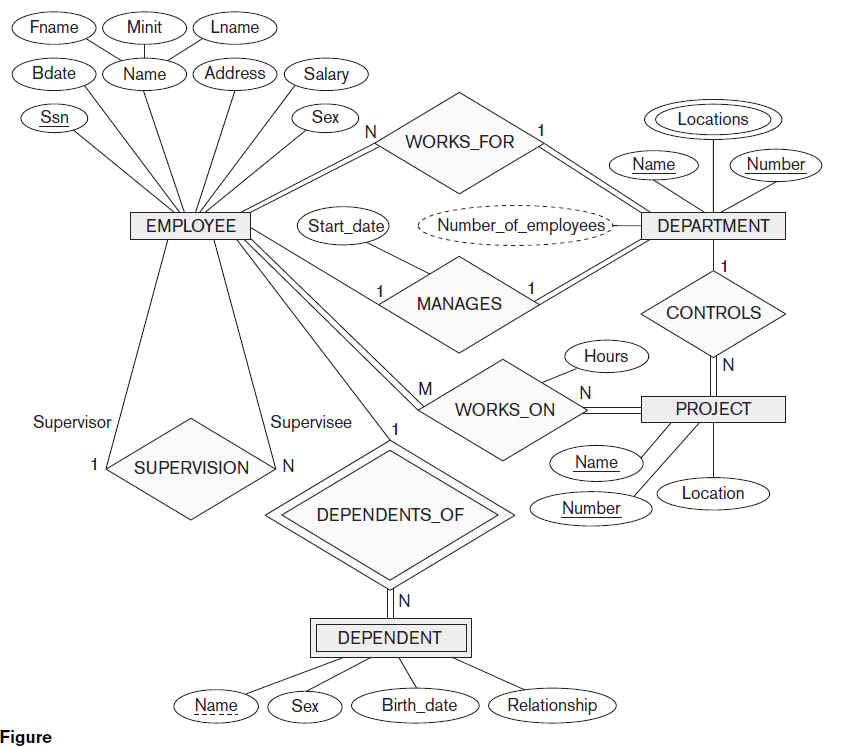
In the ER model, the main concepts are **entity**, **attribute**, and **relationship**.

**Entity Types, Entity Sets, Attributes, and Keys**

* **Entities and Attributes**
* **Entity Types, Entity Sets, Keys, and Value Sets**
* **Initial conceptual Design for the COMPANY Database**

**SUMMARY OF ER DIAGRAM SYMBOLS**





**Figure: An ER Diagram of COMPANY database**

**1. Entities and Attributes**

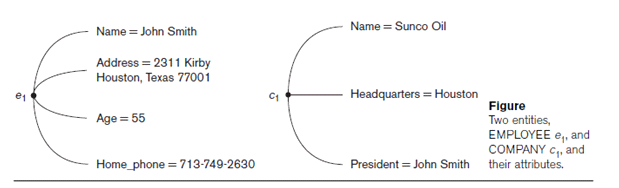
**Entity**: The basic object that the ER model represents is an entity which represents some "thing" (in the mini world) that is of interest to us, i.e. about which we want to maintain some data. An entity could represent a physical object (e.g., house, person, automobile, widget) or a less tangible concept (e.g., company, job, academic course, business transaction).

**Attribute:**

* An attribute is a property or characteristic of an entity type. Each attribute has a value drawn from some domain (set of meaningful values).
* In ER diagrams place attributes name in an ellipse with a line connecting it to its associated entity
* Attributes may also be associated with relationships
* An attribute is associated with exactly one entity or relationship

For example, an employee entity may be described by the employee’s name, age, address, salary, and job particular entity will have a **value** for each of its attributes. The attribute values that describe each entity become a major part of the data stored in the database.

Figure below shows two entities and the values of their attributes. The employee entity e1 has four attributes: Name, Address, Age, and HomePhone; their values are "John Smith," "2311 Kirby, Houston, Texas 77001," "55," and "713-749-2630," respectively. The company entity c1 has three attributes: Name, Headquarters, and President; their values are "Sunco Oil," "Houston," and "John Smith," respectively.



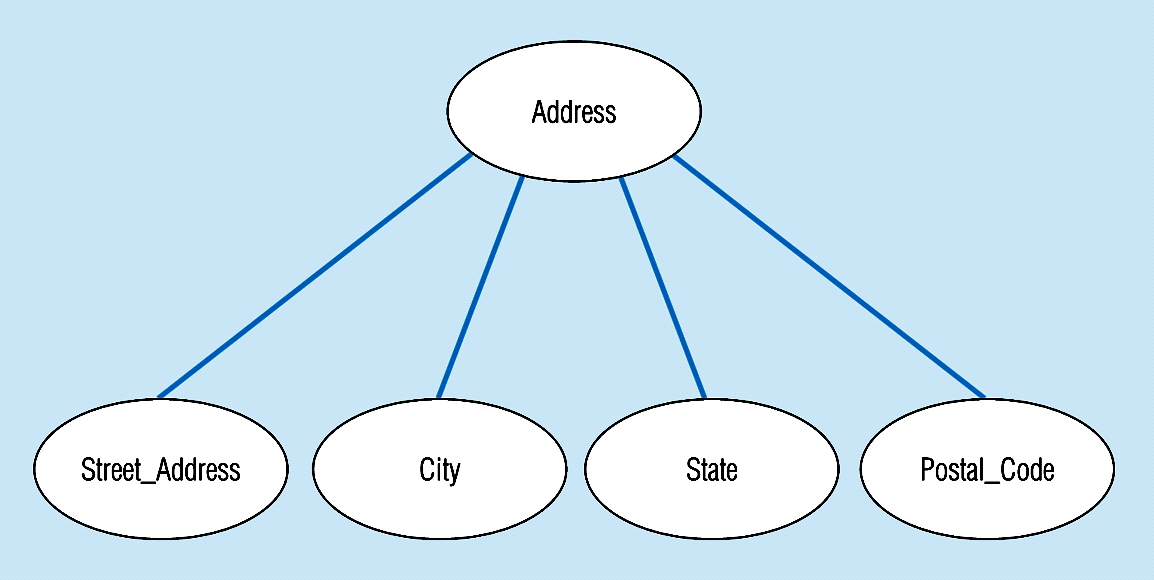
Several types of attributes occur in the ER model are:

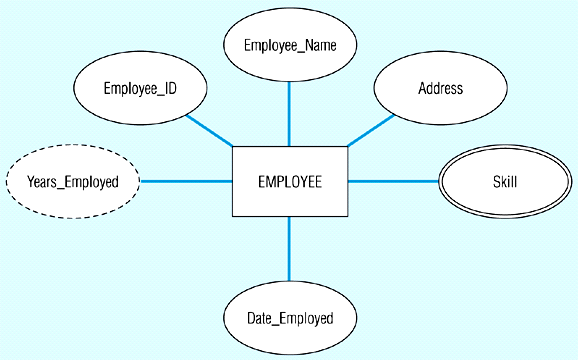
* **Composite versus Simple(atomic)**
* **Single-valued versus Multivalued**
* **Stored versus Derived.**
* **Complex attributes**
* **Null values**
* **Composite Versus Simple (Atomic) Attributes**

**Composite attributes** can be divided into smaller subparts, which represent more basic attributes with independent meanings. For example, the Address attribute of the employee entity shown in Figure 03.03 can be sub-divided into Street\_Address, City, State, and Postal\_code, with the values "2311 Kirby," "Houston," "Texas," and "77001."

Attributes that are not divisible are called **simple** or **atomic attributes.** Composite attributes can form a hierarchy; for example, Address can be subdivided into simple attributes Street\_Address, city, state and postal\_code as shown in Figure a. The value of a composite attribute is the concatenation of the values of its constituent simple attributes.

**Figure a: A Composite attribute b. Multivalued attribute (Skill) and Derived attribute (Years\_Employed)**





* **Single-Valued versus Multivalued Attribute**

Most attributes have a single value for a particular entity; such attributes are called **single-valued.** For example, Age is a single-valued attribute of person.

It frequently happens that there is an attribute that may have more than one value for a given instance, e.g. EMPLOYEE may have more than one Skill. A **multivalued attribute** isone that may take on more than one value – it is represented by an ellipse with double lines. **Figure b** shows an entity with a multivalued attribute (**Skill**) and derived attribute (Years\_Employed)

* **Stored versus Derived Attributes**

In some cases two (or more) attribute values are related—for example, the Age and BirthDate attributes of a person. For a particular person entity, the value of Age can be determined from the current (today’s) date and the value of that person’s BirthDate. The Age attribute is hence called a **derived attribute** and is said to be **derivable from** the BirthDate attribute, which is called a **stored attribute.**

* Some attribute values can be calculated or derived from others
* e.g.if no\_of\_experience needs to be calculated for EMPLOYEE, it can be calculated using Date\_of\_joining and Today's\_Date
* d attribute values.

A derived attribute is signified by an ellipse with a dashed line. **Figure b** shows an entity

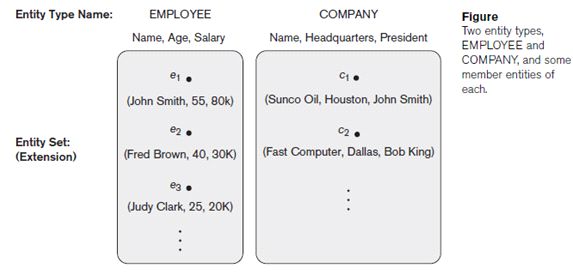
with derived attribute (Years\_Employed)

* **Complex attributes-** A complex attribute is an attribute that is both composite and multi valued.
* **The Null value**: In some cases a particular entity might not have an applicable value for a particular attribute Or that value may be unknown. *Example*: The attribute *DateOfDeath* is not applicable to a living person and its correct value may be unknown for some persons who have died. In such cases, we use a special attribute value (non-value), called **null**.

**2. Entity Types, Entity Sets, Keys, and Value Sets**

A database usually contains groups of entities that are similar. For example, a company employing hundreds of employees may want to store similar information concerning each of the employees. These employee entities share the same attributes, but each entity has *its own value(s)* for each attribute.

**Entity type :** An **entity type** defines a *collection* (or *set*) of entities that have the same attributes. Each entity type in the database is described by its name and attributes. The figure below shows two entity types, named EMPLOYEE and COMPANY, and a list of attributes for each. A few individual entities of each type are also illustrated, along with the values of their attributes.



**Entity set** : The collection of all entities of a particular entity type in the database at any point in time is called an **entity set;** the entity set is usually referred to using the same name as the entity type. For example, EMPLOYEE refers to both a *type of entity* as well as the current *set of all employee entities* in the database.

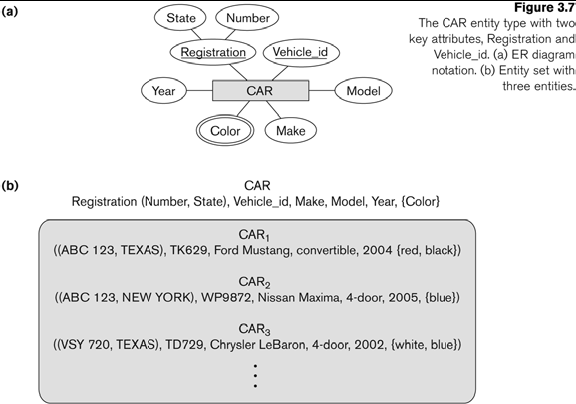
An entity type is represented in ER diagrams (Figure 3.2) as a rectangular box enclosing the entity type name. Attribute names are enclosed in ovals and are attached to their entity type by straight lines. Composite attributes are attached to their component attributes by straight lines. Multivalued attributes are displayed in double ovals.

An entity type describes the **schema** or **intension** for a *set of entities* that share the same structure. The collection of entities of a particular entity type are grouped into an entity set, which is also called the **extension** of the entity type.

**Key Attributes of an Entity Type**

An important constraint on the entities of an entity type is the **key** or **uniqueness constraint** on attributes. An entity type usually has an attribute whose values uniquely identifies each individual entity in the collection. Such an attribute is called a **key attribute,** and its values can be used to identify each entity uniquely. For example, in a STUDENT entity rollno is the **key attribute** because no 2 students have the same roll numbers. In ER diagrammatic notation, each key attribute has its name **underlined** inside the oval.

Some entity types have more than one key attribute. E.g. Entity Type CAR with two keys and a corresponding Entity Set



**Super Key, Candidate Key and Primary Key**

**Primary Key: A primary key** is an attribute or a set of attributes of a relation schema used to uniquely identify each row in a relation. It cannot contain null value. eg of a relational schema **employee**:

**employee(empno, name, job, deptno),** here empno is a primary key.

**Unique Key: A unique key** is an attribute or a set of attributes of a relation schema used to uniquely identify each row in a relation. It may contain null value.

**Super Key:** A superkey is a set of one or more attributes that taken collectively, allows us uniquely identify a row in a relation . eg {empno}, {empno,name}, {empno,deptno}, {name,deptno} etc all these sets are super key.

**Candidate Key:** A candidate key is a minimal set of attributes necessary to identify a row. It is also called a minimal superkey. Examples of superkeys in this schema would be {empno}, {empno, Name}, {empno, Name, job}, and {empno, Name, deptno} etc.

In a real database we do not need values for all of those attributes to identify row. We only need, for our example, the set {empno}. This is a **minimal superkey**—that is, a minimal set of attributes that can be used to identify a single row. Therefore, empno is a candidate key.

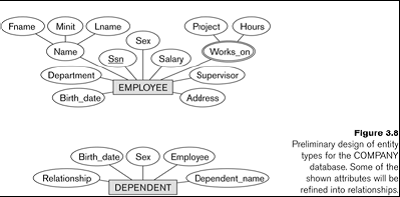
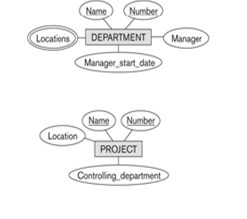
**Value Sets (Domains) of Attributes**

Each attribute of an entity type is associated with a value set or **domain** of values, which specifies the set of values that may be assigned to that attribute. E.g. in CAR entity, the domain for year attribute is {2004,2005,2002}.

* **Initial conceptual Design for the COMPANY Database**

Based on the requirements, we can identify four initial entity types in the COMPANY database:

DEPARTMENT, PROJECT, EMPLOYEE, DEPENDENT



**Relationships, Relationship Types, Roles and Structural Constraints**

**Relationship** An interaction/association between two or more entities based on a key attributes.

**Relationship Types, Sets and Instances**

A **Relationship type *R***among *n* entity types *E1, E2, ..., En* defines a set of associations or a relationship set-among entities from these entity types. Mathematically **the** **relational set R** is a set of relationship instances ***ri***

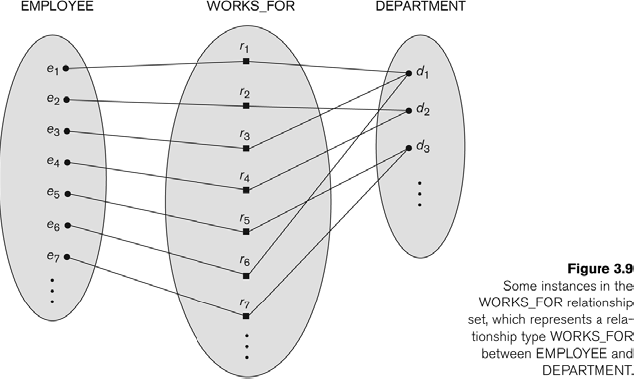
**Relationship instance *ri***

• Each *ri* associates *n* individual entities (*e1, e2, ..., en*)

• Each entity *ej* in *ri* is a member of entity set *Ej*

• Relationships uniquely identified by keys of participating entities.

In the figure belowemployees e1, e3, and e6 work for department d1; employees e2 and e4 work for department d2 and e5 and e7 work for d3.



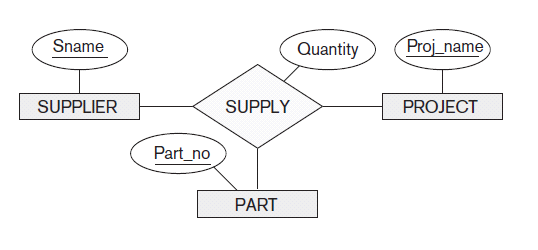
**Relationship Degree, Role Names, and Recursive Relationships**

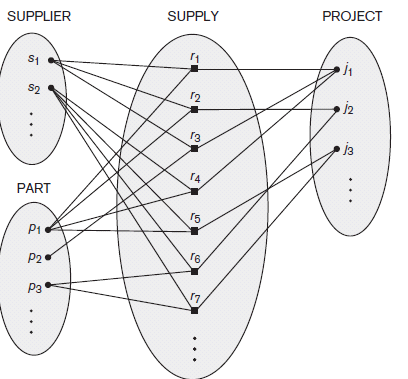
**Degree of a relationship type**

It is the number of participating entity types. e.g. a relationship type of degree 1 is unary, a relationship type of degree 2 is binary, a relationship type of degree 3 is ternary , a relationship type of degree N is called N-ary.

* Is\_Married\_To’ is a one-to-one relationship between instances of the PERSON entity type
* ‘Manages’ is a one-to-many relationship between instances of the EMPLOYEE entity type

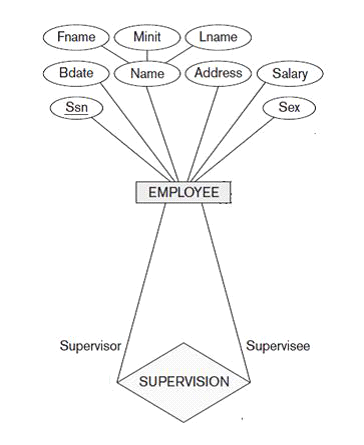
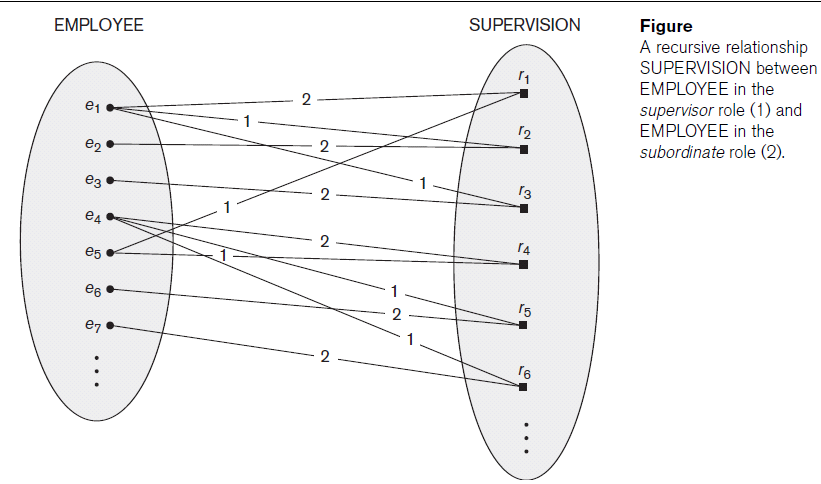
**Example of degree 3 (ternary) relationship called SUPPLY**





**Role Names and Recursive Relationships**

* Each entity type in a relationship plays a particular role. The **role name** specifies the role that a participating entity type plays in the relationship and explains what the relationship means.
* In a recursive relationship type, both participations are same entity type in different roles.
* For example, SUPERVISION relationships between EMPLOYEE (in role of supervisor or boss) and (another) EMPLOYEE (in role of subordinate or worker).
* In following figure, first role participation labeled with 1 and second role participation labeled with 2.
* In ER diagram, need to display role names to distinguish participations.

A relationship type can have attributes:

For example, HoursPerWeek of WORKS\_ON, Hours of WORKS\_ON

**Constraints on Relationship Types**

* Relationship types have certain constraints that limit the possible combination of entities that may participate in relationship.
* An example of a constraint is that if we have the entities Doctor and Patient, the organization may have a rule that a patient cannot be seen by more than one doctor. This constraint needs to be described in the schema. Another example , if the company has a rule that each employee must work for exactly one department, then this constraint needs to be described in the schema.
* There are **two main types of relationship constraints**

1. **Cardinality ratio** for binary relationship constraints and
2. **Participation constraints** and Existence Dependencies.

**I. Cardinality ratio for Binary Relationship**

Binary relationships are relationships between exactly two entities. The cardinality ratio specifies the maximum number of relationship instances that an entity can participate in. The possible cardinality ratios for binary relationship types are: 1:1, 1:N, N:1, M:N. Cardinality ratios are shown on ER diagrams by displaying 1, M and N on the diamonds.

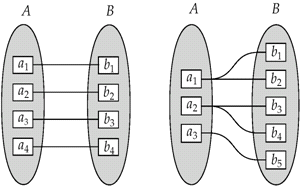
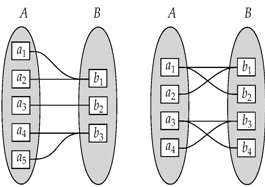
**1. One to One**: An entity in A is associated with exactly one entity in B. And one entity in B is associated with exactly one entity in A.

**2. One to Many**: An entity in A is associated with any no. of entities in B. An entity in B can be associated with at most one entity in A.

**3. Many to One**: An entity in A is associated with at most one entity in B. An entity in B can be associated with any no. of entity in A.

**4. Many to Many**: An entity in A is associated with any no. of entities in B. An entity in B can be associated with any no. of entities in A.

**Figure:**

**a. 1 : 1**  b. **1 : M c. M : 1 d. M : M**

**II. Participation Constraints and Existence Dependencies**

The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type. The constraint specifies the minimum number of relationship instances that each entity can participate in. There are two types of participation constraints:

**1. Total Participation:**

* If an entity can exist, only if it participates in at least one relationship instance, then that is called **total participation**, meaning that every entity in one set, must be related to at least one entity in a designated entity set.
* An example would be the Employee and Department relationship. If company policy states that every employee must work for a department, then an employee can exist only if it participates in at lest one relationship instance (i.e. an employee can’t exist without a department)
* It is also sometimes called an **existence dependency**.
* Total participation is represented by a double line, going from the relationship to the dependent entity.

**2. Partial Participation**:

* If only a part of the set of entities participate in a relationship, then it is called **partial participation.**
* Using the Company example, every employee will not be a manager of a department, so the participation of an employee in the “Manages” relationship is partial.
* Partial participation is represented by a single line.

**Attributes of Relationship Types**

Relationship types can have attributes similar to entity types. For example, in the relationship **Works\_On**, between the Employee entity and the Department entity we would like to keep track of the number of hours an employee works on a project. Therefore we can include **Number\_of\_Hours** as an attribute of the relationship.

Another example is for the “manages” relationship between employee and department,

we can add Start Date as an attribute of the Manages relationship.

**Weak Entity Types**

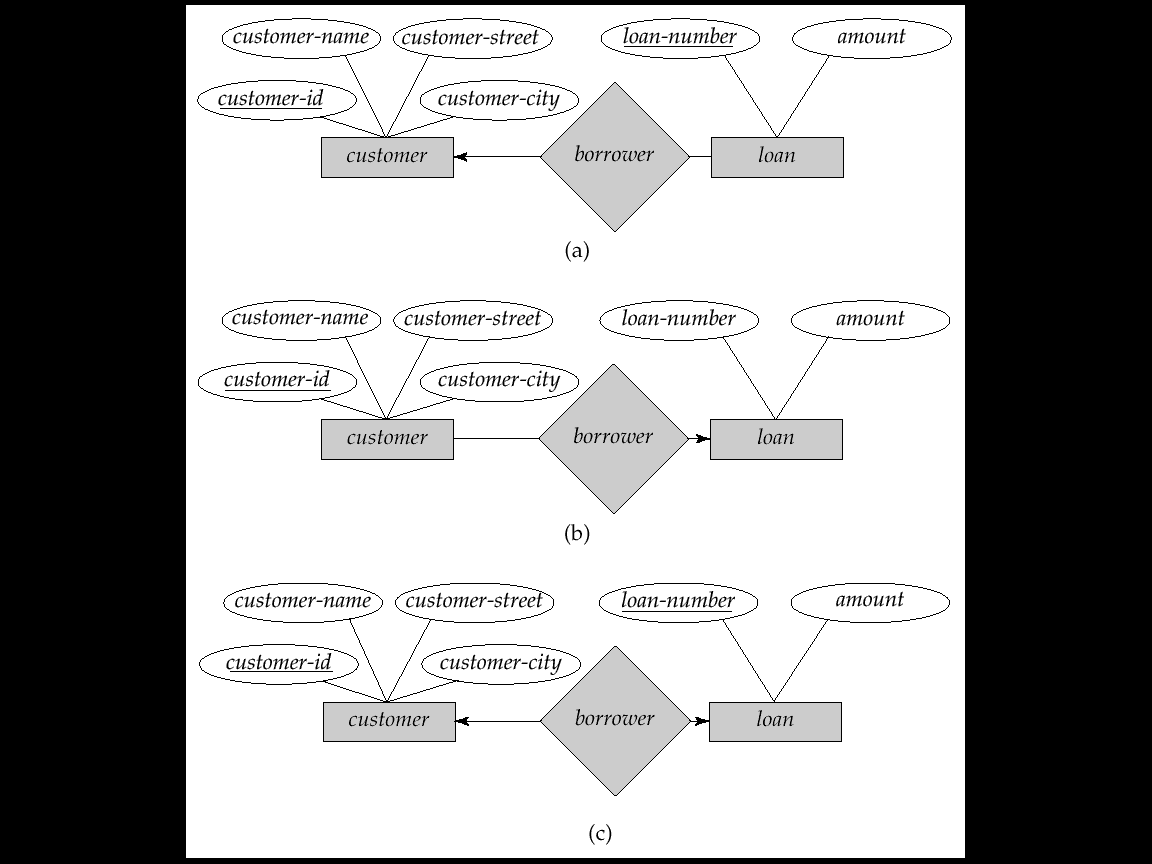
* Entity types that do not have key attributes are called **weak entity** types. The regular entity type that do have a key attributes are called **strong entity**.
* Entities that belong to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. This entity type is called an **identifying or owner entity** type.
* The relationship that relates the identifying entity type with the weak entity type is called an **identifying relationship**.
* A weak entity type always has a total participation constraint with respect to the identifying relationship, because a weak entity cannot exist without its owner.
* Not all existence dependencies result in a weak entity type; if an entity has a key attribute then it is not a weak entity.
* A weak entity type usually has a partial key, which is the set of attributes that can uniquely identify weak entities that are ***related to the same owner entity.***

**Example: Consider figure of ER Diagram of COMPANY**

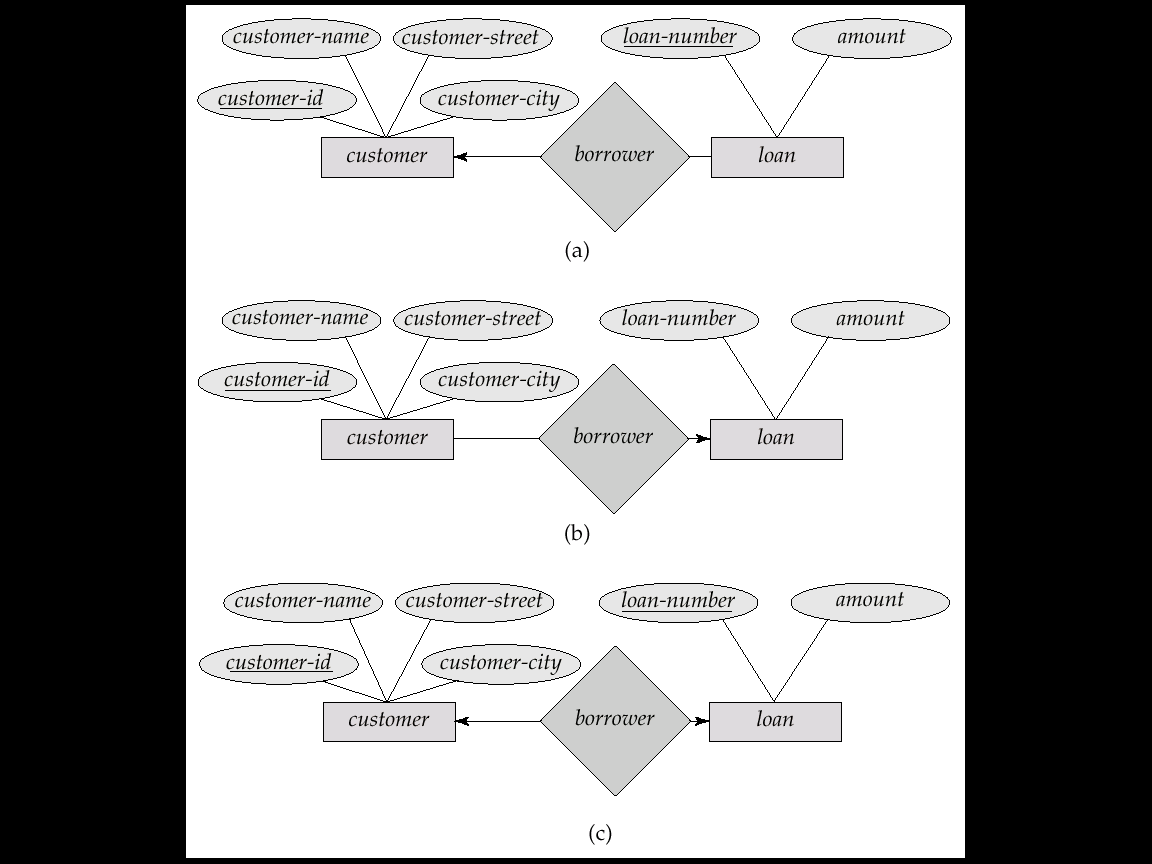
A DEPENDENT entity is identified by the dependent’s first name, and the specific EMPLOYEE with whom the dependent is related. Name of DEPENDENT is the partial key. DEPENDENT is a weak entity type. EMPLOYEE is its identifying entity type via the **identifying relationship** type DEPENDENT\_OF.

**Some examples of ER- Diagram**

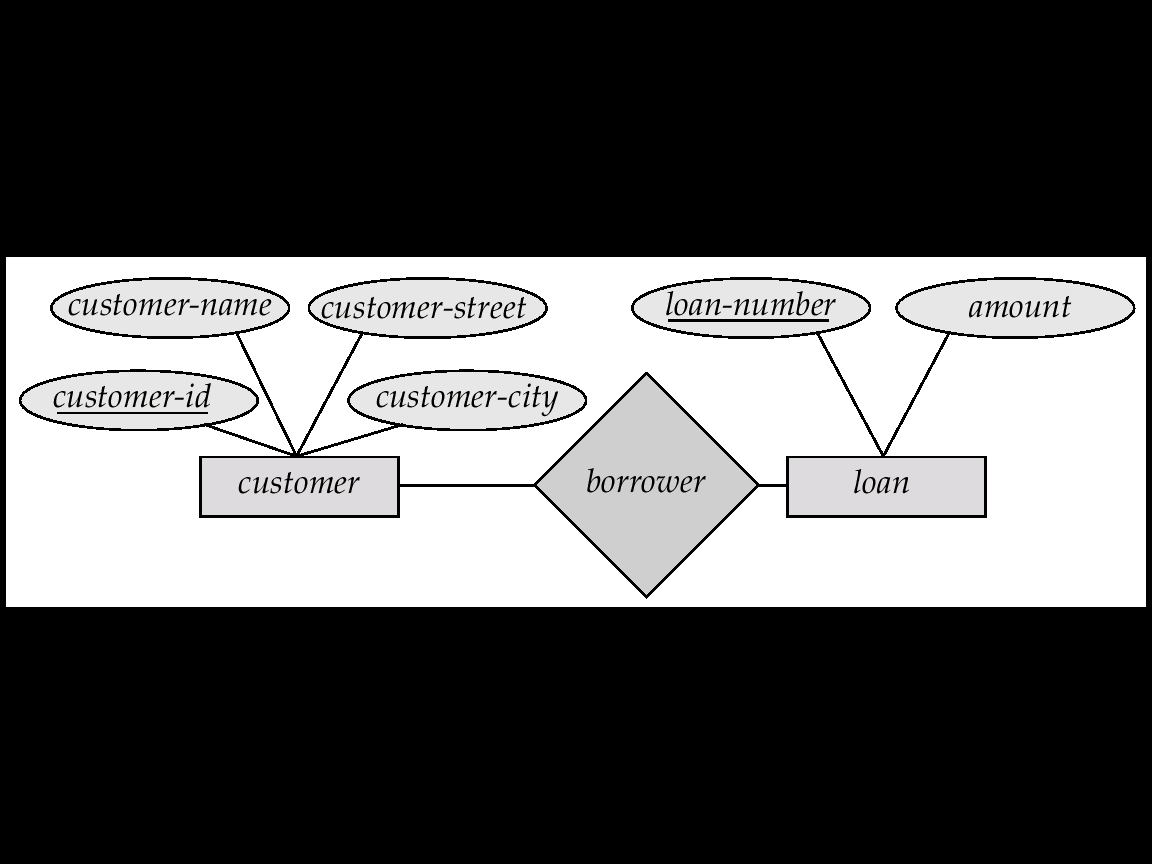
**1: M( ONE TO MANY)**



**M : 1(MANY TO ONE)**



**M : M(MANY TO MANY)**



**Enhanced Entity-Relationship and Object Modeling**

EER stands for **Enhanced ER** or **Extended ER**. The various EER Model Concepts

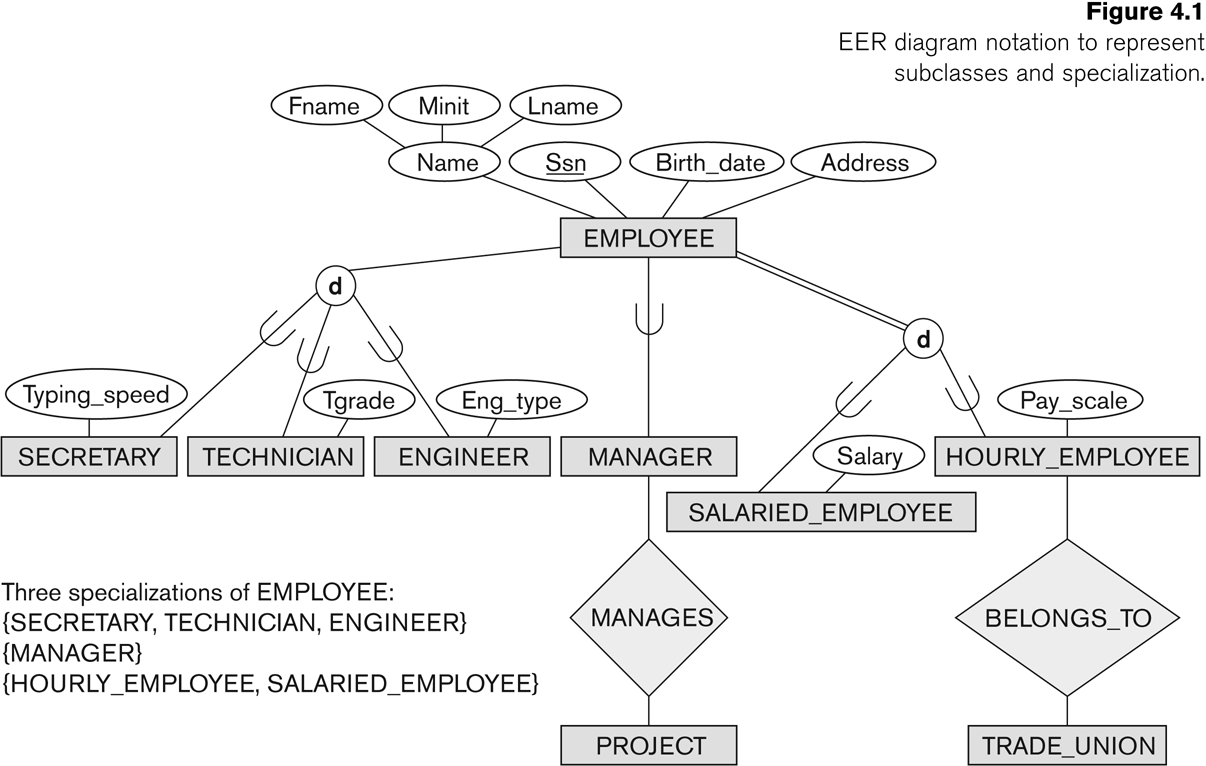
* Includes all modeling concepts of basic ER.
* Additional concepts:
* Subclasses/Superclasses and Attribute Inheritance
* Specialization/Generalization
* These are fundamental to conceptual modeling
* The additional EER concepts are used to model applications more completely and more accurately.
* EER includes some object-oriented concepts, such as inheritance.
* **Subclasses & Superclasses and Attribute Inheritance**

**Subclass** - Often an Entity has many subgrouping which are meaningful and need to be explicitly represented. These are called subclasses.

**Superclass** - Is the term used to describe the Entity that contains a subclass(s).

Example: EMPLOYEE entity type is grouped into SECRETARY, ENGINEER, MANAGER, TECHNICIAN, SALARIED\_EMPLOYEE, HOURLY\_EMPLOYEE.

Here EMPLOYEE is ***superclass***, SECRETARY, ENGINEER, etc. is ***subclass*** of EMPLOYEE.



The relationship between a superclass and any of its subclasses a ***superclass/subclass relationships*** or ***class/subclass relationships.*** Example: EMPLOYEE/SECRETARY,

EMPLOYEE/TECHNICIAN.

A ***class/subclass relationship*** is also called IS-A relationships

e.g. SECRETARY IS-A EMPLOYEE, TECHNICIAN IS-A EMPLOYEE, etc.

An entity that is member of a subclass represents the same real-world entity as some member of the superclass:

* The subclass member is the same entity in a distinct specific role
* An entity cannot exist in the database merely by being a member of a subclass; it must also be a member of the superclass
* A member of the superclass can be optionally included as a member of any number of its subclasses.

Examples: A salaried employee who is also an engineer belongs to the two subclasses:

ENGINEER, and

SALARIED\_EMPLOYEE

A salaried employee who is also an engineering manager belongs to the three subclasses:

MANAGER,

ENGINEER, and

SALARIED\_EMPLOYEE

It is not necessary that every entity in a superclass be a member of some subclass. Subclass is shown with an arc on the line pointing toward the superclass. EER diagram also has "d" (disjoint) in circle denoting unique specialization.

**Attribute Inheritance** An entity that is member of a subclass inherits

* All attributes of the entity as a member of the superclass.
* All relationships of the entity as a member of the superclass.

Example: SECRETARY (as well as TECHNICIAN and ENGINEER) inherit the attributes   
 Name, SSN, …, from EMPLOYEE. Every SECRETARY entity will have values for the

inherited attributes.

**2 . Specialization and Generalization**

**Specialization**

* Specialization is the process of defining a set of subclasses of a superclass.
* The set of subclasses is based upon some distinguishing characteristics of the entities in

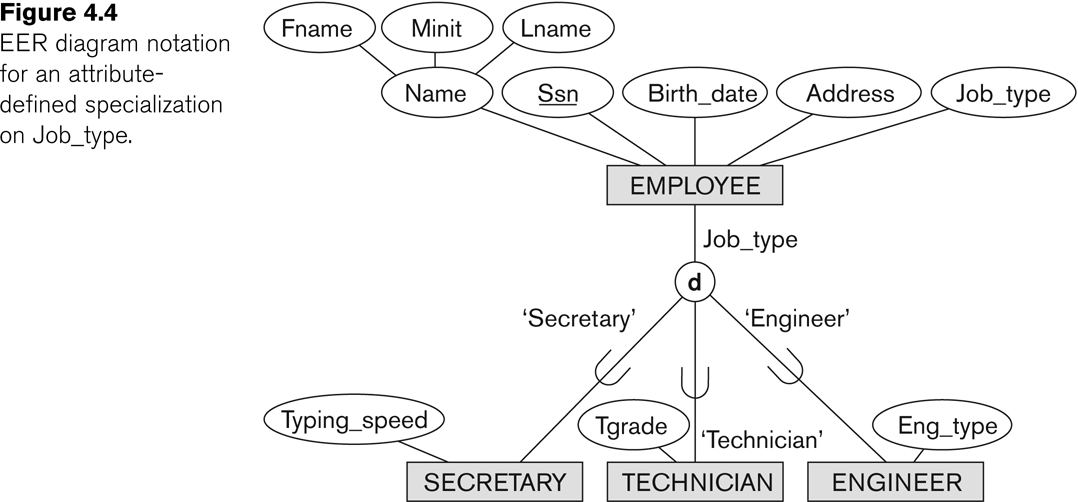
the superclass

Example: {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of EMPLOYEE based upon job type. Another specialization of EMPLOYEE based on method of pay is {SALARIED\_EMPLOYEE, HOURLY\_EMPLOYEE}.

Superclass/subclass relationships and specialization can be diagrammatically represented in EER diagrams

* Attributes of a subclass are called specific or local attributes.
* For example, the attribute TypingSpeed of SECRETARY

The subclass can also participate in specific relationship types. For example in fig 4.1, a relationship BELONGS\_TO of HOURLY\_EMPLOYEE



**Generalization**

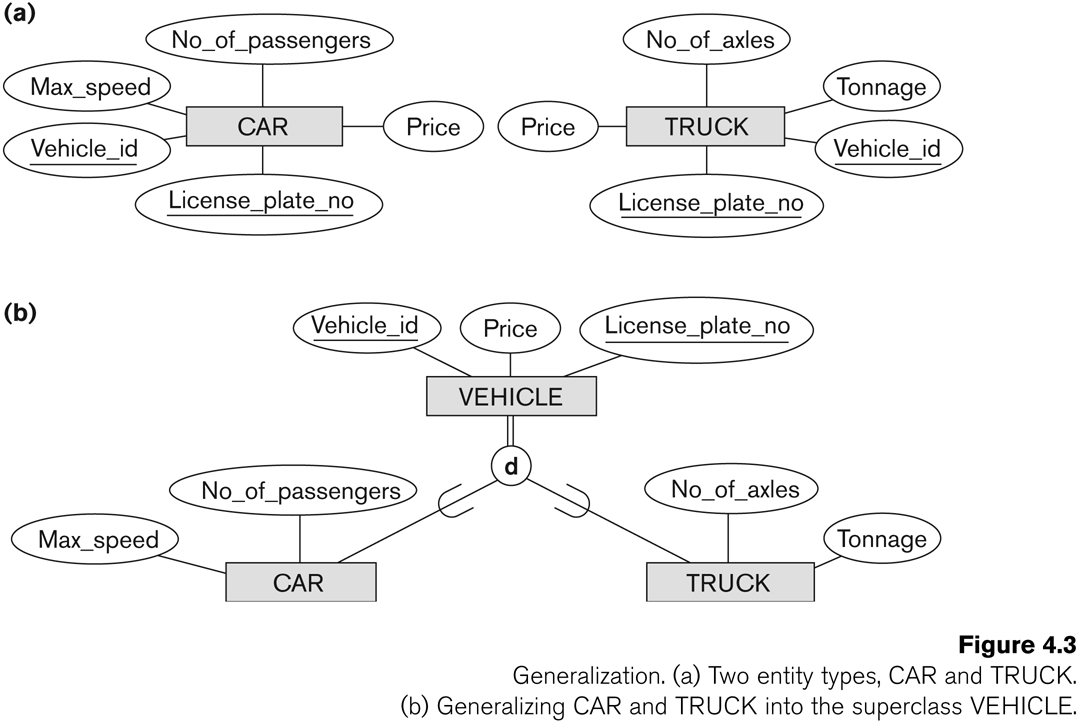
The reverse of the specialization process

* Generalization takes common features of subclasses and creates a superclass.
* Example: CAR, TRUCK generalized into VEHICLE; both CAR, TRUCK become subclasses of the superclass VEHICLE.

– We can view {CAR, TRUCK} as a specialization of VEHICLE

– Alternatively, we can view VEHICLE as a generalization of CAR and TRUCK.

Figure 4.3a shows CAR and TRUCK with several common attributes. Figure 4.3b shows VEHICLE superclass with CAR and TRUCK subclasses.



Diagrammatic notation are sometimes used to distinguish between generalization and specialization.

* An arrow pointing to the generalized superclass represents a generalization
* Arrows pointing to the specialized subclasses represent a specialization
* We do not use this notation because it is often subjective as to which process is more appropriate for a particular situation