UNIT - I

Introduction to databases and Conceptual Modeling

1.1 Introduction

Applications of Traditional Database

In everyday life we are doing several activities that involve some interaction with a database.

Eg. Bank - to deposit or withdraw funds,
Hotel / Airline / Rail Reservation
To access a computerized library catalog
Online (web) shopping - such as a book, toy, or computer

Even purchasing items at a supermarket often automatically updates the database that holds the inventory of grocery items.All these activities will involve some computer program accessing a database. The above mentioned examples are comes under Traditional database applications.

Traditional database applications, in which most of the information that is stored and accessed is either textual or numeric.

Databases play a critical role in almost all areas where computers are used, including business, electronic commerce, engineering, medicine, genetics, law, education, and library science.

Applications of Advanced Database

Advances in technology have led to exciting new applications of database systems.

1. **Multimedia databases (**store images, audio clips, and video streams digitally)
2. **Geographic information systems (GIS)** can store and analyze maps, weather data, and satellite images.
3. **Datawarehouses** and **online analytical processing (OLAP)** systems are used in many companies to extract and analyze useful business information from very large databases to support decision making.
4. **Real-time** and **active database technology** is used to control industrial and manufacturing processes.
5. **Database search techniques** are applied by World Wide Web search engines (to improve the search for information that is needed by users browsing the Internet).

DATA: Data means facts that can be recorded and that have implicit meaning.

Eg. In olden days, Phone book or indexed address book which contains the names, telephone numbers, and addresses of the people we know, is nothing but data.

The same data we might have stored it on a hard drive, using a personal computer and software such as Microsoft Access or Excel.

Today's phone which allows us to store ph.no., mail id's, address etc.

(Here we use the word data for both singular and plural. In standard English for plural – data, singular - datum)

Information

- Processed data is called information
- The purpose of data processing is to generate the information required for carrying out the business activities.

DATABASE: **A database is a collection of related data** with an implicit meaning stored together to serve multiple applications**.** The overall purpose of the database system is to record and maintain information.

(The preceding definition of database is quite general; for example, we may consider the collection of words that make up this page of text to be related data and hence to constitute a database. However, the common use of the term database is usually more restricted.)

IMPLICIT PROPERTIES OF THE DATABASE :

1. A database **represents some aspect of the real world**, sometimes called the miniworld or the **universe of discourse (UoD).** Changes to the miniworld are reflected in the database. (If a person's phone number changed means it should update in my phone database)

2. A database is a **logically coherent (interrelated) collection of data** with some inherent meaning. A random collection of data cannot be said to as a database. (In a library, Computer books database means it should have only those books, it should not contain a single record of Civil books. Means logically it is not same)

3. A database is **designed, built, and populated with data for a specific purpose**. It has an intended group of users and some predefined applications in which these users are interested. In short, a database has some source from which data is derived, some degree of interaction with events in the real world, and an audience that is actively interested in its contents. The end users of a database may perform business transactions (eg, a customer buys a camera) or events may happen (eg, student's achievement - winning a tournament / India wining world cup) that cause the information in the database to change.

4. **Database must be accurate and reliable** at all times, means changes must be reflected in the database as soon as possible.

5. **A database may be generated and maintained manually or it may be computerized.**
    - For example, a Department library card catalog is a database that may be created and maintained manually. But College library means it should be maintained with the help of softwares, means should be computerized.
    - A computerized database may be created and maintained either by a group of application programs written specifically for that task or by a database management system.

6. **Size / Complexity: A database can be of any size and of varying complexity**.

    It can be range from simple (student details of a single section) to huge/complex (eg. Amazon's database that keeps track of all his products customers and suppliers)

[Eg, the list of student / employee names and addresses referred to a section or small company may consist of only a few hundred records, each with a simple structure.

On the other hand,

- the computerized catalog of a large library may contain half a million entries organized under different categories—by primary author's last name, by subject, by book title—with each category organized alphabetically.

- the list of student / employee names and addresses referred to a college or a big company like IBM / TCS where many branches in many countries available, will have a complex structure.

- Still more complex means database of Department of Income tax which has to maintain past few years returns of each taxpayer and also current return of every citizen who is/was filing the form. So huge amount of information must be organized and managed so that users can search for, retrieve, and update the data as needed.

- An example of a large commercial database is Amazon.com. It contains data for over 20 million books, CDs, videos, DVDs, games, electronics, apparel, and other items. The database occupies over 2 terabytes (a terabyte is 1012 bytes worth of storage) and is stored on 200 different computers (called servers). About 15 million visitors access Amazon.com each day and use the database to make purchases. The database is continually updated as new books and other items are added to the inventory and stock quantities are updated as purchases are transacted. About 100 people are responsible for keeping the Amazon database up-to-date.]

DBMS : **A database management system (DBMS) is a collection of programs that enables users to create and maintain a database**.

The DBMS is a general-purpose software system that facilitates the processes of

- defining
- constructing
- manipulating and
- Sharing databases among various users and applications.

Other important functions provided by the DBMS include

- protecting the database and
- maintaining it over a long period of time

Database and DBMS software together is referred to as a database system.

Database is a collection of related data, the database is created and maintained either by a programs group of application programs written specifically for the tasks or by a database management system (DBMS).

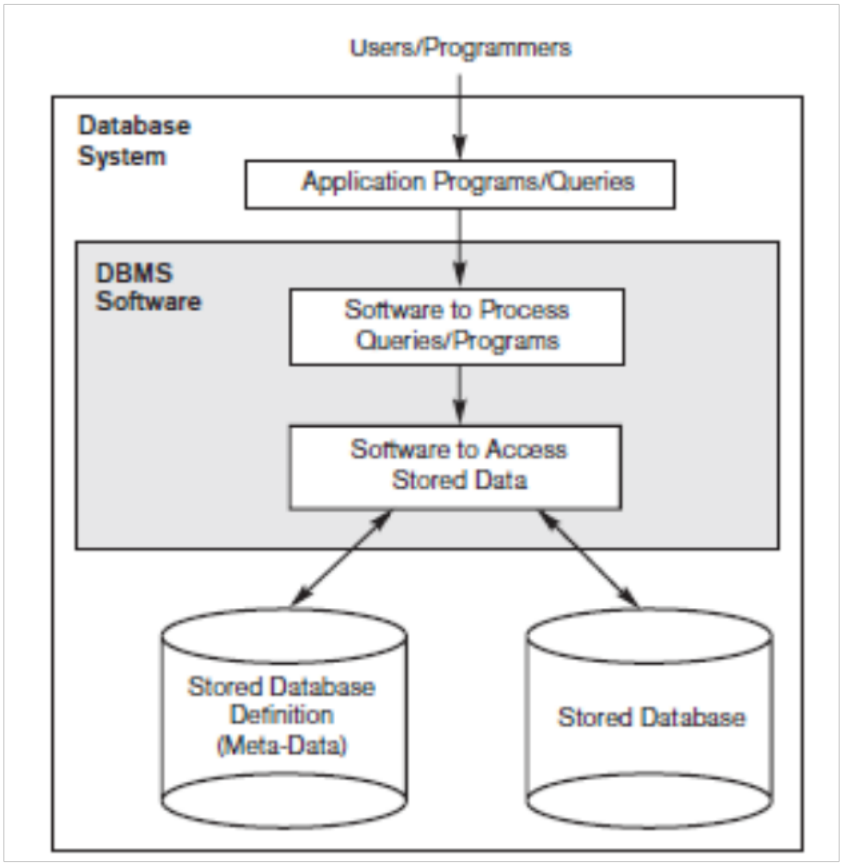

Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database

Fig 1.1: A simplified database system environment

<u>Defining a database</u> involves specifying

- the data types, structures, and constraints of the data to be stored in the database.
- **The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called meta-data.**

<u>Constructing</u> the database

- is the process of storing the data on some storage medium (eg. Magnetic disc) that is controlled by the DBMS.

<u>Manipulating</u> a database includes functions such as

- **querying** the database to retrieve specific data,
- **updating** the database to reflect changes in the miniworld, and
- **generating reports** from the data.

<u>Sharing</u> a database allows multiple users and programs to access the database simultaneously.

Protection includes

- System protection: Preventing database from being corrupted when hardware or software failures occur
- security protection: Preventing unauthorized or malicious access to database.

Maintaining A typical large database may have a life cycle of many years, so the DBMS must be able to maintain the database system by allowing the system to evolve as requirements change over time.

QUERY

An application program accesses the database by sending queries or requests for data to the DBMS. A query typically causes some data to be retrieved; a transaction may cause some data to be read and some data to be written into the database.
[It is not absolutely necessary to use general-purpose DBMS software to implement a computerized database. We could write our own set of programs to create and maintain the database, in effect creating our own special-purpose DBMS software. In either case—whether we use a general-purpose DBMS or not—we usually have to deploy a considerable amount of complex software. In fact, most DBMSs are very complex software systems.]

An Example

A UNIVERSITY database for maintaining information concerning students, courses, and grades in a university environment.

STUDENT file - stores data on each student
COURSE file - stores data on each course
SECTION file - stores data on each section of a course
GRADE_REPORT file- stores the grades that students receive in the various sections they have completed
PREREQUISITE file - stores the prerequisites of each course show in Fig 1.2.

The database is organized as five files / relation / table has a set of named fields / attributes / columns, each of which is specified of the same data type. (In addition to a datatype, we might put further restrictions upon a field, eg., GRADE_REPORT must have a value from the set {'A','B'…'F'})

Each table will be populated with data in the form of records / tuples/ rows, each of which represents some entity (in the miniworld) or some relationship between entities.

Defining the database, we must specify the structure of the records of each file by specifying the different types of **data elements** to be stored in each record.

Constructing the database, the records in the various files may be related.

Manipulation of database, involves querying and updating.

[Defining **the database** Each STUDENT record includes data to represent the student's Name, Student_number, Class (such as freshman or '1', sophomore or '2', and so forth), and Major (such as mathematics or 'MATH' and computer science or 'CS'). A **data type** for each data element within a record should be specified. For example, we can specify that Name of STUDENT is a string of alphabetic characters, Student_number of STUDENT is an integer, and Grade of GRADE_REPORT is a single character from the set {'A', 'B', 'C', 'D', 'F', 'I'}. Coding scheme can also be used to represent the values of a data item. For example, we represent the Class of a STUDENT as 1 for freshman, 2 for sophomore, 3 for junior, 4 for senior, and 5 for graduate student.

Constructing the database For example, the record for Smith in the STUDENT file is related to two records in the GRADE_REPORT file that specify Smith's grades in two sections. Similarly, each record in the PREREQUISITE file relates two course records: one representing the course and the other representing the prerequisite.
Most medium-size and large databases include many types of records and have many relationships among the records.

Manipulation of database
Examples of queries are as follows:
■ Retrieve the transcript(s) of student named 'Smith'
■ List the names of students who took the section of the 'Database' course offered in fall 2008 and their grades in that section
■ List the prerequisites of the 'Database' course

Examples of updates include the following:
■ Change the class value of 'Smith' to sophomore
■ Insert a new section for the 'Database' course for this semester
■ Remove from the SECTION , instructor as ' Chang '

A query/ update must be conveyed to the DBMS in a precise way(via the query language of the DBMS) in order to be processed.

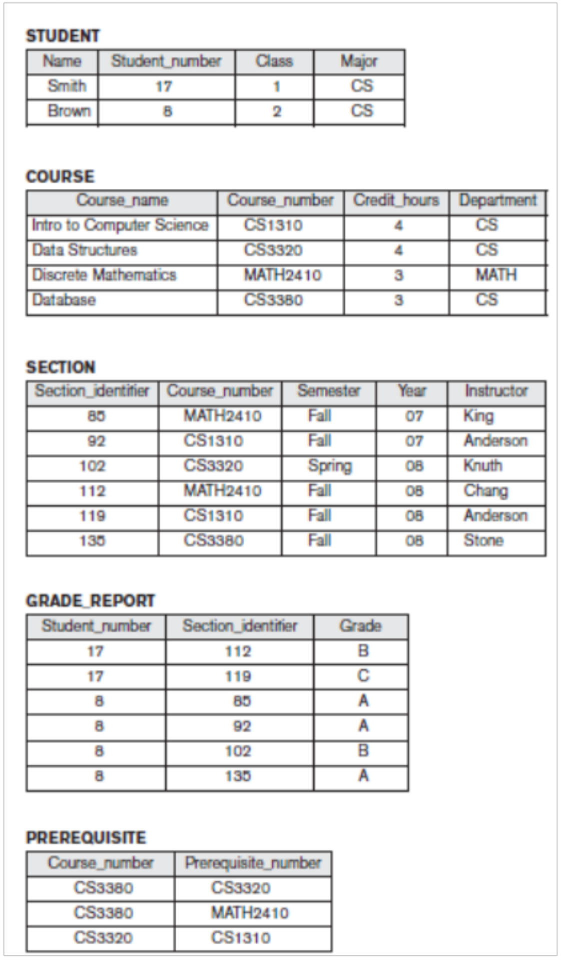
Developing a new database or design of a new application for an existing database proceeds in phases, including **requirements specification and analysis and various levels of design**.

Conceptual design	Entity-Relationship model
logical design	Relational Data Model
physical design	**F**ile structures

Conceptual design that can be represented and manipulated using some computerized tools so that it can be easily maintained, modified, and transformed into a database implementation.

Logical design that can be expressed in a data model implemented in a commercial DBMS.

Physical design, last level, during which further specifications are provided for storing and accessing the database. The database design is implemented, populated with actual data, and continuously maintained to reflect the state of the mini-world.



STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Fig 1.2: Student Database

1.2 Characteristics of the database approach

Database approach Vs File processing 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.

For example, one user, the grade reporting office, may keep a file on students and their grades. Programs to print a student's transcript and to enter new grades into the file are implemented as part of the application.

A second user, the accounting office, may keep track of students' fees and their payments. Although both users are interested in data about students, each user maintains separate files-and programs to manipulate these files-because each requires some data not available from the other user's files. This redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common data up to date.

File processing approach: Data redundancy and inconsistency are the drawbacks

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**

1.2.1 Self-Describing Nature of a Database System

The database contains a complete definition or description of the database structure and constraints. (This is the fundamental characteristic of the database approach)

This definition is stored in the DBMS **system catalog,** which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data.

The information stored in the catalog is called **meta-data,** and it describes the structure of the primary database.

The catalog is used by both DBMS software and users.

A DBMS catalog stores the description of the database. The description is called meta-data . This allows the DBMS software to work with different databases.
Catalog: structure of each file, type & storage format of each data item, constraints on data

[Users - Who need to know the names of tables and attribures, and sometimes data type information and other things

DBMS software- which certainly needs to "know" how the data is structured / organized in order to interpret it in a manner consistent with that structure.

A general-purpose DBMS software package is not written for a specific database application, and hence it must refer to the catalog to know the structure of the files in a specific database, such as the type and format of data it will access. Hence the structure of the data cannot be "hard-coded" in its programs.

In traditional file processing, data definition is typically part of the application programs themselves. Hence, these programs are constrained to work with only one specific database, whose structure is declared in the application programs. For example, an application program written in c++ may have struct or class declarations, and a COBOL program has Data Division statements to define its files. Whereas file-processing software can access only specific databases, DBMS software can access diverse databases by extracting the database definitions from the catalog and then using these definitions.]

1.2.2 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. [If we decide to change the structure of the data (eg., by adding the first two digits to the YEAR field, in order to make the program Y2K complaint), every application in which a description of that file's structure is hard-coded must be changed.]

In contrast, DBMS access programs do not require such changes in most cases, because the structure of data files is stored in the **DBMS catalog** separately from the access programs. This property is called as **program-data independence.**

For example, a file access program may be written in such a way that it can access only STUDENT records of the structure shown in Fig 1.2. If we want to add another piece of data to each STUDENT record, say the BirthDate, such a program will no longer work and must be changed. This is possible because the user can define operations on data as part of the database definitions.

Operation is specified in two parts

1) **Interface:** An operation include the operation name and the data type of its argument.

2) **Implementation:** operation is specified separately and can be changed without affecting the interface.

- **Insulation between programs and data: Called program-data independence.**

Allows changing data storage structures without having to change the DBMS access programs

Data Abstraction:

- **A data model is used to hide storage details and present the users with a conceptual view of the database.**
- **Programs refer to the data model constructs rather than data storage details.**

[By contrast, in a DBMS environment, it is just need to change the description of STUDENT records in the catalog to reflect the inclusion of the new data item BirthDate; no programs are changed. The next time a DBMS program refers to the catalog, the new structure of STUDENT records will be accessed and used.

In other words, the DBMS provides a conceptual or logical view of the data to application programs, so that the underlying implementation may be changed without the programs being modified(This is referred to as program-data independence)

Also, which access paths (eg., indexes) exist are listed in the catalog, helping the DBMS to determine the most efficient way to search for items in response to a query.

The characteristic that allows program-data independence is called **data abstraction**. A DBMS provides users with a conceptual representation of data that does not include many of the details of how the data is stored or how the operations are implemented.

Informally, a data model is a type of data abstraction that is used to provide this conceptual representation. The data model uses logical concepts, such as objects, their properties, and their interrelationships, that may be easier for most users to understand than computer storage concepts. Hence, the data model hides storage and implementation details that are not of interest to most database users.]

1.2.3 Support of Multiple Views of the Data

A database is accessed by many users, each of whom may require a different perspective or view of the database. A view may be a subset of the database or it may

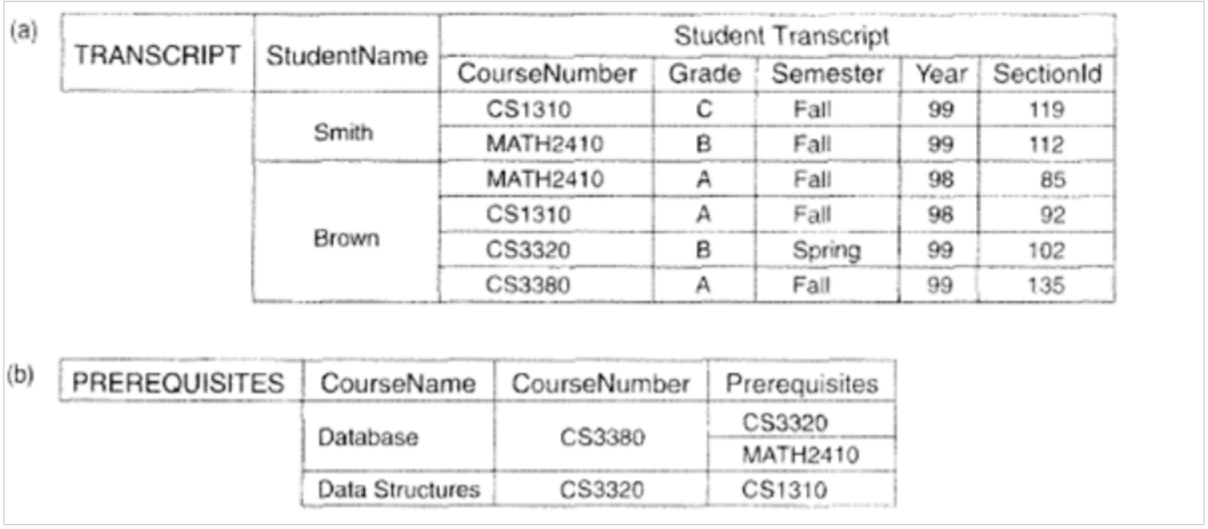
contain virtual data that is derived from the database files but without actually stored. The user need not be aware of whether the data they refer is stored or derived.

A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views.

For example, one user of the database of Fig 1.2 may be interested only in accessing and printing the transcript of each student; the view for this user is shown in Figure 1.3a.

A second user, who is interested only in checking that students have taken all the prerequisites of each course for which they register, may require the view shown in Figure l.4b.

- **Each user may see a different view of the database, which describes only the data of interest to that user.**
- **A good DBMS has facilities for defining multiple views. This is not only convenient for users, but also addresses security issues of data access.(eg., The Registrar's office view should not provide any means to access financial data)**



(a)

TRANSCRIPT	StudentName	Student Transcript				
		CourseNumber	Grade	Semester	Year	SectionId
	Smith	CS1310	C	Fall	99	119
		MATH2410	B	Fall	99	112
	Brown	MATH2410	A	Fall	98	85
		CS1310	A	Fall	98	92
		CS3320	B	Spring	99	102
		CS3380	A	Fall	99	135

(b)

PREREQUISITES	CourseName	CourseNumber	Prerequisites
	Database	CS3380	CS3320
			MATH2410
	Data Structures	CS3320	CS1310

Fig: 1.3 Two views derived from the database in Figure 1.2 (a) The STUDENT TRANSCRIPT View (b) The COURSE PREREQUISTES View

1.2.4 Sharing of Data and Multiuser Transaction Processing

A multiuser DBMS 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. It must also include concurrency control to ensure that several users trying to update the same data that will results of the updates is correct.

Eg: Airline flight reservation such applications are called as online transaction processing [OLTP] applications, when several reservation clerks try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one clerk at a time for assignment to a passenger. **A fundamental role of multiuser DBMS**

software is to ensure that concurrent transactions operate correctly.

The concept of a transaction has become central to many database applications. A **transaction** is an executing program or process that includes one or more database accesses, such as reading or updating of database records. Each transaction is supposed to execute a logically correct database access if executed in its entirety without interference from other transactions.
The DBMS must enforce several transaction properties. The **isolation property** ensures that each transaction appears to execute in isolation from other transactions, even though hundreds of transactions may be executing concurrently. The **atomicity property** ensures that either all the database operations in a transaction are execute

[Like in OS concepts , in DBMS also simultaneous access of computer resources by multiple users/ processes is a major source of complexity. Arising form this is the need for concurrency control, which is supposed to ensure that several users trying to update the same data do so in a " controlled " manner so that the results of the updates are as though they were done in some sequential order(rather than interleave, which could result in data incorrect).

This given rise to the concept o a transaction, which is a process that makes one or more accessed to a database and which must have the appearance of executing in isolation form all other transactions (even ones that access the same data at the same time) and of being atomic (in the sense that, if the system crashes in the middle of its execution, the database contents must be as though it did not execute at all). Application such as airline reservation systems are known as online transaction processing applications.

1.3 Data Models, Schemas and Instances

Database System Concepts and Architecture: In basic client/server DBMS architecture, the system functionality is distributed between two types of modules.

A **client module,** typically designed so that it will run on a user workstation or personal computer. Typically, application programs and user interfaces that access the database run in the client module. Hence, the client module handles user interaction and provides the user-friendly interfaces such as forms- or menu-based GUIs (graphical user interfaces).

A **server module**, typically handles data storage, access, search, and other functions. A characteristic of the database approach is that it provides a level of data abstraction, by hiding details of data storage that are not needed by most users.

A **data model** - is a collection of concepts that can be used to describe the conceptual /

logical structure of a database. The model provides the necessary means to achieve the abstraction.

The structure of a database is characterized by data types, relationships, and constraints that hold for the data. Models also include a **set of basic operations** for specifying retrievals and updates.

Data models are changing to include concepts to specify the behaviour of the database application. This allows designers to specify a set of user defined operations that are allowed.

1.3.1 <u>Categories of Data Models</u> (based on degree of abstractness)

Data models can be categorized in multiple ways.

- **High level / conceptual data models** – provides a view close to the way users would perceive the data; uses concepts such as
    - **Entity** – represents a real world object or concept (eg., student, employee, course, department, event)
    - **Attribute** - represents property of interest that describes an entity, such as name, salary, height, age, colour.
    - **Relationship** – among two or more entities, represents an association among two or more entities. (works-on relationship between an employee and a project)

- **Low-level / Physical data models** – provide concepts that describe the details of how data is stored in the computer system, such as record formats, record orderings and access paths(indexes). These concepts are generally meant for the specialist, and not the end user.
  An **access path** is a structure that makes the search for particular database data records efficient.

  An **index** is an example of an access path that allows direct access to data using an index term or a keyword.

- **Representational / implementational data models** – provide concepts that may be understood by the end user but not far removed from the way data is organized. Representational data models are used most frequently in commercial DBMSs. They include relational data models, and legacy models such as network and hierarchical models. Also called **record-based** model as the data is represented by using record structures in this model. Intermediate level of abstraction.

- **Object data models** group (ODMG) – a group of higher level implementation data models closer to conceptual data models.

1.3.2 Schemas, Instances and Database State

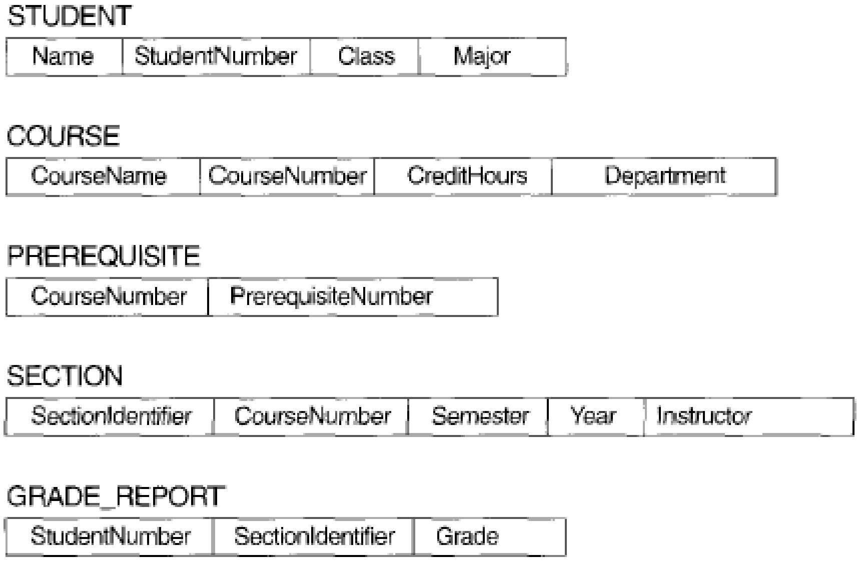
The description of a database is called the database schema. The schema is specified during database design, and is not expected to change frequently.

Data models have conventions for displaying schemas as diagrams. A displayed schema is called a schema diagram. Each object in the schema is called a **schema construct.** Schema diagrams display only some aspects of a schema, such as names and some constraints.

The actual data stored in the database may change frequently, every time records are added or updated. The data in the database at a particular time is called **the database state or snapshot.**

Database Schema vs Database State

When a database is defined, the schema is specified to the DBMS. The database state at this point is the empty state with no data. The initial state of the database is when the database is first populated or loaded with the initial data. Every time data is added/removed/updated, there is a new database state. The DBMS is responsible for ensuring every state is a valid state, a state that satisfies the structure and constraints specified in the schema.



STUDENT

Name	StudentNumber	Class	Major

COURSE

CourseName	CourseNumber	CreditHours	Department

PREREQUISITE

CourseNumber	PrerequisiteNumber

SECTION

SectionIdentifier	CourseNumber	Semester	Year	Instructor

GRADE_REPORT

StudentNumber	SectionIdentifier	Grade

Fig 1.4 S**chema diagram**

The DBMS stores the descriptions of the schema constructs and constraints,

called the **meta data, in the DBMS catalogue**.
The schema is called the intension, and the database state is called an extension of the schema.

Application requirements change occasionaly, which is one of the reasons why software maintenance is important. On such occasions, a change to a database's schema may be called for. An example would be to add a Date_of_Birth field/ attribute to the STUDENT table. Making changes to a database schema is known as **schema evolution**. Most modern DBMS's support schema evolution operations that can be applied while a database is operational.

- Database State: Refers to the **content of a database at a moment** in time.
- Initial Database State: Refers to the database state when it is initially loaded into the system.
- Valid State: A state that satisfies the structure and constraints of the database.
- Distinction
  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

1.4 Database Languages and Interfaces

- A DBMS supports a variety of users, so it must provide appropriate languages and interfaces each category of users.

1.4.1 DBMS Languages

- DDL – the **data definition language**, used by the DBA and database designers to define the conceptual and internal schemas.
  - The DBMS has a DDL compiler to process DDL statements in order to identify the schema constructs, and to store the description in the catalogue.

- In databases where there is a separation between the conceptual and internal schemas
  DDL - is used to specify the conceptual schema, and SDL, **storage definition language** - is used to specify the internal schema.

- For true three-schema architecture, VDL, **view definition language**, is used to specify the user views and their mappings to the conceptual schema.
  But in most DBMSs, the DDL is used to specify both the conceptual schema and the external schemas.
  [Once the schemas are compiled, and the database is populated with data, users need to manipulate the database. Manipulations include retrieval, insertion, deletion and modification.]

- The DBMS provides operations using the DML, **data manipulation language**, used for performing operations such as retrieval and update upon the populated database.

- In most DBMSs, the VDL, DML and the DML are not considered as separate languages, but a comprehensive integrated language for conceptual schema definition, view definition and data manipulation. Storage definition is kept separate to fine-tune the performance, usually done by the DBA staff.

- An example of a comprehensive language: SQL, which represents a VDL, DDL, DML as well as statements for constraint specification, etc.

Data Manipulation Languages (DMLs)

Two main types:

High-level/Non procedural

- Can be used on its own to specify complex database operations.
- DMBSs allow DML statements to be entered interactively from a terminal, or to be embedded in a programming language. If the commands are embedded in a general purpose programming language, the statements must be identified so they can be extracted by a pre-compiler and processed by the DBMS.

Low Level/Procedural

- Must be embedded in a general purpose programming language.
- Typically retrieves individual records or objects from the database and processes each separately.
- Therefore it needs to use programming language constructs such as loops.
- Low-level DMLs are also called record at a time DMLS because of this.
- High-level DMLs, such as SQL can specify and retrieve many records in a single DML statement, and are called set at a time or set oriented DMLs.
- High-level languages are often called declarative, because the DML often specifies what to retrieve, rather than how to retrieve it.

DML Commands

- When DML commands are embedded in a general purpose programming language, the programming language is called the **host language** and the DML is called the **data sub-language**.
- High-level languages used in a standalone, interactive manner is called a query language.
- Casual end users use high-level query language to specify requests, where programmers usually use embedded DML.
- Parametric end users usually interact with user-friendly interfaces, which can also be used by casual users who don’t want to learn the high-level languages.

1.4.2 DBMS Interfaces

Types of interfaces provided by the DBMS include:

Menu-Based Interfaces for Web Clients or Browsing

- Present users with list of options (menus)
- Lead user through formulation of request
- Query is composed of selection options from menu displayed by system.

Forms-Based Interfaces

- Displays a form to each user.
- User can fill out form to insert new data or fill out only certain entries.
- Designed and programmed for naïve users as interfaces to canned transactions.

Graphical User Interfaces

- Displays a schema to the user in diagram form. The user can specify a query by manipulating the diagram. GUIs use both forms and menus.

Natural Language Interfaces

- Accept requests in written English, or other languages and attempt to understand them.
- Interface has its own schema, and a dictionary of important words. Uses the schema and dictionary to interpret a natural language request.

Interfaces for Parametric Users

- Parametric users have small set of operations they perform.
- Analysts and programmers design and implement a special interface for each class of naïve users.
- Often a small set of commands included to minimize the number of keystrokes required. (I.e. function keys)

Interfaces for the DBA

- Systems contain privileged commands only for DBA staff.

Include commands for creating accounts, setting parameters, authorizing accounts, changing the schema, reorganizing the storage structures etc

1.4 Using high-level conceptual data models for database design

Figure 1.5 shows a simplified overview of the database design process. The first step shown is **requirements collection and analysis**. During this step, the database designers interview prospective database users to understand and document their **data requirements**. The result of this step is a concisely written set of users' requirements.

These requirements should be specified in as detailed and complete a form as possible.

In parallel with specifying the data requirements, it is useful to specify the known **functional requirements** of the application. These consist of the user-defined **operations** (or **transactions**) that will be applied to the database, including both retrievals and updates.

Once the requirements have been collected and analyzed, the next step is to create a **conceptual schema** for the database, using a high-level conceptual data model. This step is called **conceptual design**. The conceptual schema is a concise description of the data requirements of the users and includes detailed descriptions of the entity types, relationships, and constraints; these are expressed using the concepts provided by the high-level data model. Because these concepts do not include implementation details, they are usually easier to understand and can be used to communicate with nontechnical users. The high-level conceptual schema can also be used as a reference to ensure that all users' data requirements are met and that the requirements do not conflict. This approach enables database designers to concentrate on specifying the properties of the data, without being concerned with storage and implementation details. This makes it is easier to create a good conceptual database design.

During or after the conceptual schema design, the basic data model operations can be used to specify the high-level user queries and operations identified during functional analysis. This also serves to confirm that the conceptual schema meets all the identified functional requirements. Modifications to the conceptual schema can be introduced if some functional requirements cannot be specified using the initial schema.

The next step in database design is the actual implementation of the database, using a commercial DBMS. Most current commercial DBMSs use an implementation data model—such as the relational or the object-relational database model—so the conceptual schema is transformed from the high-level data model into the implementation data model. This step is called **logical design** or **data model mapping**; its result is a database schema in the implementation data model of the DBMS. Data model mapping is often automated or semi automated within the database design tools.

The last step is the **physical design** phase, during which the internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files are specified. In parallel with these activities, application programs are designed and implemented as database transactions corresponding to the high-level transaction specifications.

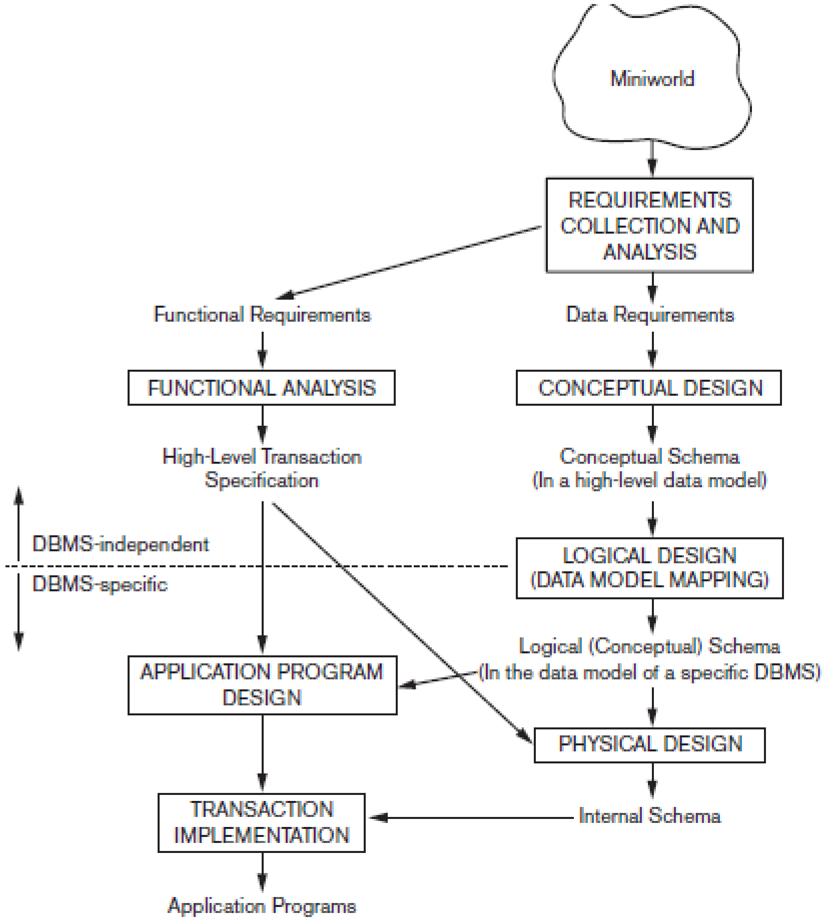

Miniworld
REQUIREMENTS COLLECTION AND ANALYSIS
Functional Requirements
Data Requirements
FUNCTIONAL ANALYSIS
CONCEPTUAL DESIGN
High-Level Transaction Specification
Conceptual Schema (In a high-level data model)
DBMS-independent
DBMS-specific
LOGICAL DESIGN (DATA MODEL MAPPING)
Logical (Conceptual) Schema (In the data model of a specific DBMS)
APPLICATION PROGRAM DESIGN
PHYSICAL DESIGN
TRANSACTION IMPLEMENTATION
Internal Schema
Application Programs

Fig 1.5: Main phases of Database Design

1.5 A Sample Database Application

In this section we describe a sample database application, called COMPANY, which serves to illustrate the basic ER model concepts and their use in schema design. We list the data requirements for the database here, and then create its conceptual schema step-by-step as we introduce the modeling concepts of the ER model. The COMPANY database keeps track of a company's employees, departments, and projects. Suppose that after the requirements collection and analysis phase, the database designers provide the following description of the *miniworld*—the part of the company that will be represented in the database.

The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.

- A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
- We store each employee's name, Social Security number,2 address, salary, sex (gender), and birth date. An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department. We keep track of the current number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee (who is another employee).
- We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, sex, birth date, and relationship to the employee.

1.6 Entity types, attributes, keys (E-R Model)

Basic concepts of ER model

The ER model describes data as entities, relationships, and attributes.

Entity

- The basic object of the ER model is an entity
- An entity represents a 'thing' of interest to us (in the real world) about which we want to maintain some data (for the representation in a database.)
- An entity may be object with a physical existence eg., person, car, house, employee, student
- An entity may be object with conceptual existence eg., a company, a job, a course

Attributes

- An entity is represented by as set of properties/ characteristics called attributes.
- The attributes are useful in describing the properties of each entity in the entity set.
- These properties reflects the sort of information which is stored in the database. [Every entity will have some set of properties. These properties are used to distinguish one entity from another entity. These properties termed as **attributes**.]
  - eg the attributes of the entity of

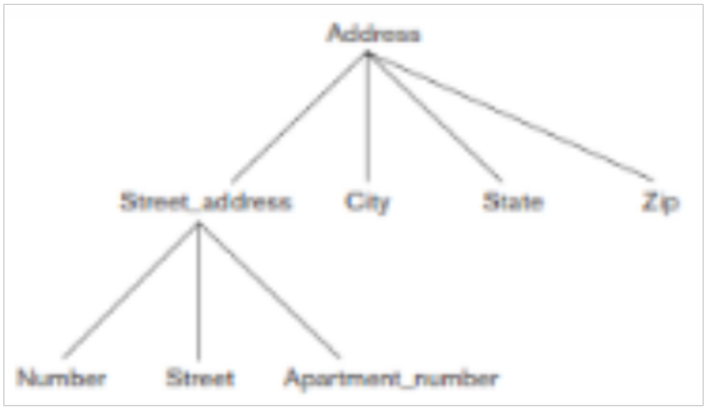
a student	-	Reg. No., Name, Course
an employee	-	employee's name, age, address, salary, job
a vehicle	-	vehicle number, make, capacity etc,.

Several types of attributes

1. Simple vs. Composite
2. Single-valued vs. Multi-valued
3. Stored vs. Derived
4. Null attribute
5. Complex

1. Simple vs. Composite Attributes

- **Simple** or **atomic attributes** are those attributes which cannot be further divided into subparts (indivisible or indecomposible).
  eg, Reg no. of a student is unique and cannot be further divided
- Composite attributes can be divided into smaller subparts, which represent more basic attributes, which have their own meanings.
  Example of a composite attribute
  1 A B**irthDate** attribute can be viewed as being composed of (sub-) attributes for month, day, and year
  2 An **Address.** Address can be broken down into a number of subparts, such as Street Address, City, State, ZipCode. Street Address may be further broken down by Number, Street Name and Apartment number. As this suggests, composition can extend to a depth of two (as here) or more.
- To describe the structure of a composite attribute, one can draw a tree or as a text
  BirthDate(Month,Day,Year)
  Name (Firstname, Middlename, Lastname)
  Address (streetAddr (StrNum,StrName,AptNum), City, State, Zip)



[Composite attributes can be used if the attribute is referred to as the whole, and the atomic attributes are not referred to. For example, if you wish to store the Company Location, unless you will use the atomic information such as Postal Code, or City separately from the other Location information (Street Address etc) then there is no need to subdivide it into its component attributes, and the whole Location can be designated as a simple attribute.]

2. Single-Valued vs. Multi-valued Attributes

- **Single-valued attributes** are attributes which have a single value for any entity, such as a car only has one model, a student has only one ID number, an employee has only one data of birth.

- **Multi-valued attributes** are the attributes which can have multiple values for a single entity, for example, a doctor may have more than one specialty (or may have only one specialty), a customer may have more than one mobile phone number, or they may not have one at all.
- Multi-valued attributes may have a lower and upper bounds to constrain the number of values allowed. For example, a doctor must have at least one specialty, but no more than 3 specialties.

3. Stored vs. Derived Attributes

- In some cases two (or more) attributes are related.
- If an attribute can be calculated using the value of another attribute, they are called **derived** attributes.
- The attribute that is used to derive the attribute is called a **stored** attribute.
- Derived attributes are not stored in the file, but can be derived when needed from the stored attributes.
- **eg** 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 a derived attributes and is said to be derivable form the Birthdate attributes, which is called a Stored attributes.

4. Null Valued Attributes

- A null value attributes is used when an attributes does not have any value.
- A person who does not have a mobile phone would have null stored at the value for the Mobile Phone Number attribute.
- Null can also be used in situations where the attribute value is unknown. There are two cases where this can occur, one where it is known that the attribute is valued, but the value is missing, for example hair color. Every person has a hair color, but the information may be missing. Another situation is if mobile phone number is null, it is not known if the person does not have a mobile phone or if that information is just missing.

5. Complex Attributes

- Complex attributes are attributes that are nested in an arbitrary way.
- For example a person can have more than one residence, and each residence can have more than one phone, therefore it is a complex attribute that can be represented as:
- Multi-valued attributes are displayed between braces { }
- Complex Attributes are represented using parentheses ( )

E.g. {**AddressPhone**({Phone(AreaCode, PhoneNumber)}, Address(StreetAddress(Number, Street, ApartmentNumber), City, State, Zip))}

Entity Types and Entity Sets

- An **entity type** defines a collection of entities that have the same attributes. Each entity type in the database is described by its **Name and attributes**. The entity shares the same attributes, but each entity has its own value for each attribute.
- The collection of all entities of a particular entity type stored in the database at any point in time is called an **entity set**. The entity set (Student) can be referred to using the same name as the entity name.
- The fig shows two entity type

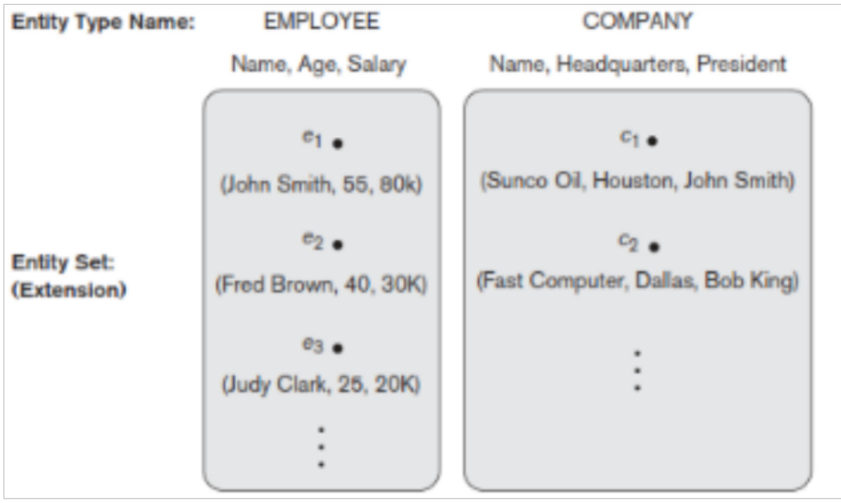


Figure 7.6
Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

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

Key Attributes of an Entity Type

- **A key attribute** is an attribute whose values are distinct / unique for each individual entity in the entity set.
- The values of the key attribute can be used to identify each entity uniquely.
- Sometimes a key can consist of several attributes together, where the combination of attributes is unique for a given entity. This is called a **composite key**.
- Composite keys should be minimal, meaning that all attributes must be included to have the uniqueness property.
- Specifying that an attribute is a key of an entity type means that the uniqueness property must hold true for every entity set of the entity type.
- An entity can have more than one key attribute, and some entities may have no key attribute. Those entities with no key attribute are called **weak entity types**.

Value Sets (Domains) of Attributes

- Each simple attribute of an entity is associated with a value set or domain of values -
  which specifies the set of values that may be assigned to that attribute for each

entity.
For example, - date of birth must be before today’s date, and after 01/01/1900
- the Student Name attribute must be a string of alphabetic characters.
- the age an employee should be within the range 23 - 60

- Value sets are not specified in ER diagrams.

Entity Types and Key Attributes

Entities with the same basic attributes are grouped or typed into an entity type.

For example, the entity type EMPLOYEE and PROJECT.

An attribute of an entity type for which each entity must have a unique value is called a key attribute of the entity type. For example, SSN of EMPLOYEE.

A key attribute may be composite.

Vehicle Tag Number is a key of the CAR entity type with components (Number,State).

An entity type may have more than one key.

The CAR entity type may have two keys: VehicleIdentificationNumber (popularly called VIN)

VehicleTagNumber (Number, State), license plate number.

Each key is underlined

Entity set is the collection of all entities of a particular entity type stored in the database.

Displaying an Entity type

In ER diagrams, an entity type is displayed in a rectangular box

Attributes are displayed in ovals.

Each attribute is connected to its entity type

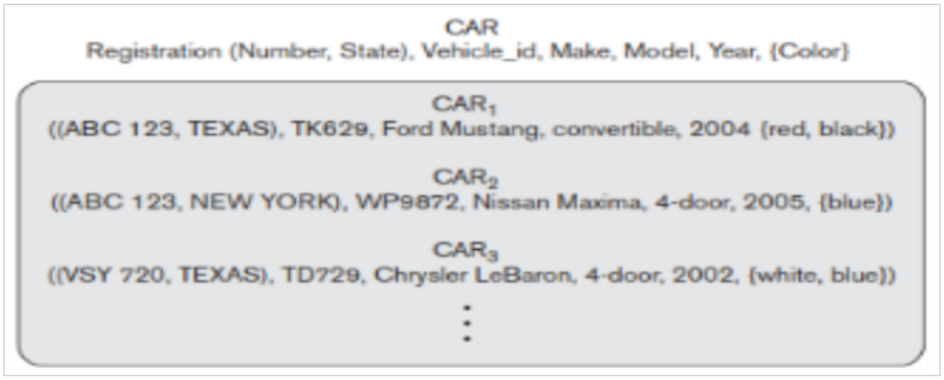
Components of a composite attribute are connected to the oval representing the composite attribute.

Each key attribute is underlined.

Multivalued attributes displayed in double ovals.

This example shows three CAR entity instances in the entity set for CAR

Same name (CAR) used to refer to both the entity type and the entity set.



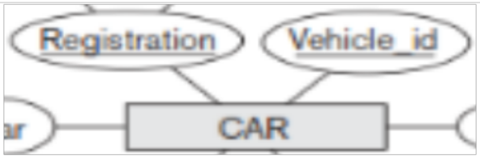
CAR
Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}

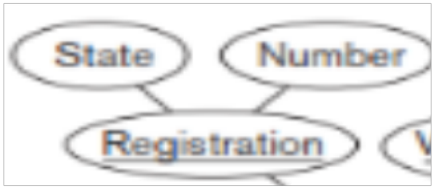
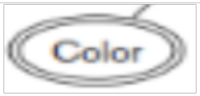
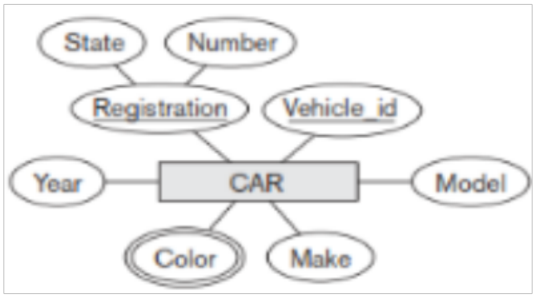
CAR_1
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

CAR_2
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

CAR_3
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

⋮

Car is an entity type, so entity type name should represent with a rectangular box	
Registration and vehicle_id are the key attributes, means which are unique	

Registration attributes is an example of a composite key formed from two simple component attributes, Registration Number, State. Components of a composite attributes are shown between parenthesis ( )	
Multivalued attributes are shown between set braces { }	
Car entity is having 6 attributes Registration, Vehicle_id, color, Make, year, model Registration, Vehicle_id - key attributes Registration - composite attributes Color - Multivalued attributes State, Number, Vehicle_id, Make, year, model - single valued attribues	 State Number Registration Vehicle_id Year CAR Model Color Make

Initial Conceptual Design of the COMPANY Database

We can now define the entity types for the COMPANY database, based on the requirements described in Section 1.5. After defining several entity types and their attributes here, we refine our design in next section, after we introduce the concept of a relationship. According to the requirements listed in Section 1.5, we can identify four entity types—one corresponding to each of the four items in the specification (see Figure 1.6):

1. An entity type DEPARTMENT with attributes Name, Number, Locations, Manager, and Manager_start_date. Locations is the only multivalued attribute. We can specify that both Name and Number are (separate) key attributes because each was specified to be unique.
2. An entity type PROJECT with attributes Name, Number, Location, and Controlling_department. Both Name and Number are (separate) key attributes.
3. An entity type EMPLOYEE with attributes Name, Ssn, Sex, Address, Salary, Birth_date, Department, and Supervisor. Both Name and Address may be composite attributes; however, this was not specified in the requirements. We must go back to the users to see if any of them will refer to the individual components of Name—First_name, Middle_initial, Last_name—or of Address.
4. An entity type DEPENDENT with attributes Employee, Dependent_name, Sex, Birth_date, and Relationship (to the employee).

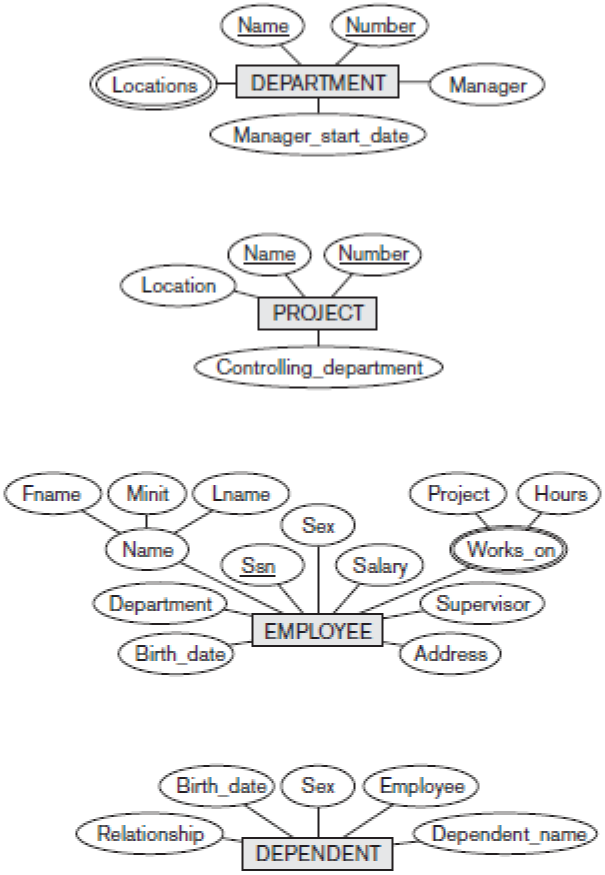
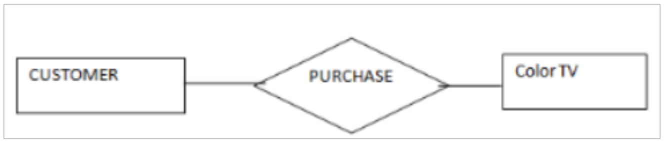


Fig 1.6: Preliminary design of entity types for the COMPANY database.

1.7 Relationship

A **relationship relates two or more distinct entities with a** specific meaning or A relationship is an association between entities where the association includes one entity from each participating entity type.

eg., We can define a relationship which associates the CUSTOMER with an item Color TV



Note: Whenever an attribute of one entity type refers to an entity(of the same type or different type),we say that a relationship exists between the two entities.

EG., A STUDENT entity being associated to an ACADEMIC_COURSE entity via, say an ENROLLED_IN relationship.

Note: In ER diagram, relationship type are displayed as DIAMOND shape boxes, which are connected by straight line to the rectangular box representing participating entity types relationship name is displayed in the diamond shaped box.

Relationship Types

Relationships of the same type are grouped or typed into a **relationship type.**

Relationship type Is the schema description of a relationship. Identifies the relationship name and the participating entity types. Also identifies certain relationship constraints. -A **relationship type** R among n **entity types** E1, E2, ..., E n defines a set of associations—or a **relationshipset**—among entities from these entity types.

As for the **entity types** and **entity sets**, a **relationship type** and its **corresponding relationship set** are customarily referred to by the same name, R.

Mathematically, the relationship set R is a set of **relationship instances** r i, where each r i associates n individual entities (e 1 , e 2 , ..., e n), and each entity e j in r i is a member of entity set E_j , $1 \leq j \leq n.$

where R - Relationship type or relationship set

e_j - n individual a entities

r i - relationship entities

E_j - Entity type

Hence, a relationship set is a mathematical relation on E1, E 2, ..., En. Each of the entity types E1, E 2, ..., En is said to be **participate in the relationship type R**. Similarly each of the individual entities e1, e 2, ..., en is said to be participate in the **relationship instance** r i = (e1, e 2, ..., en).

For example, consider a relationship type WORKS_FOR between the two entity type EMPLOYEEs and DEPARTMENT, which associates each employee with the department the employee works for

Each relationship instance (r i) in the relationship set WORKS_FOR associates one employee entity and one department entities.

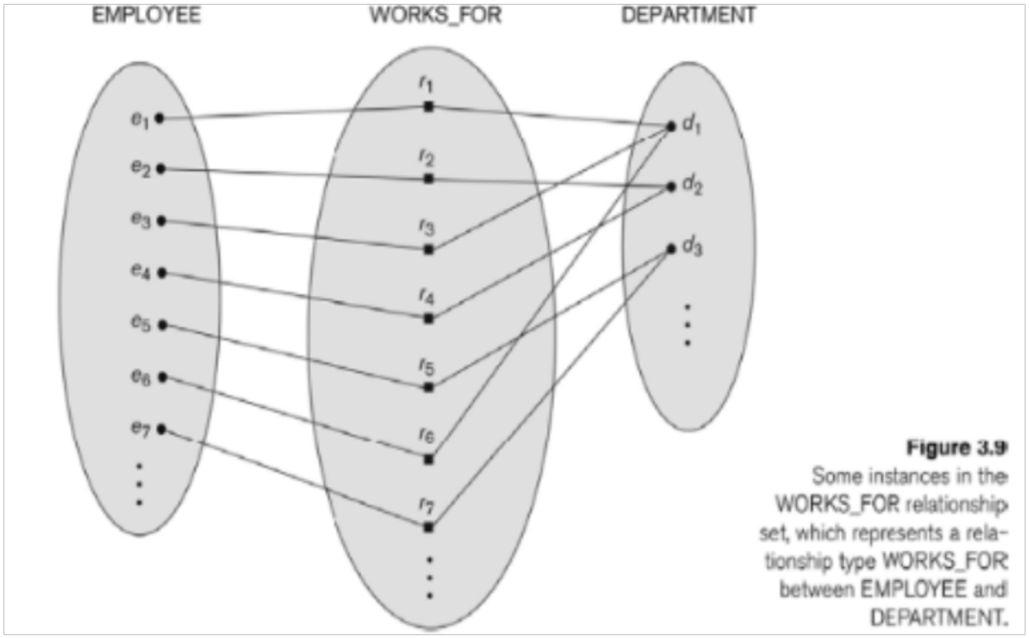
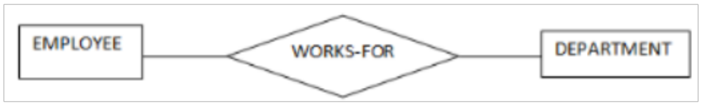


Figure 3.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

Some instances of the WORKS_FOR relationship between EMPLOYEE and DEPARTMENT

- employee e1, e3 and e6 work for department d1
- employee e2 and e4 work for d2
- employee e5 and e7 work for d3



In ER diagrams, we represent the relationship type as follows:

In the original Chen notation, the relationship is placed inside a diamond, Connected to the participating entity types via straight lines. As always, there are many notations in use today... e.g.managers manage employees

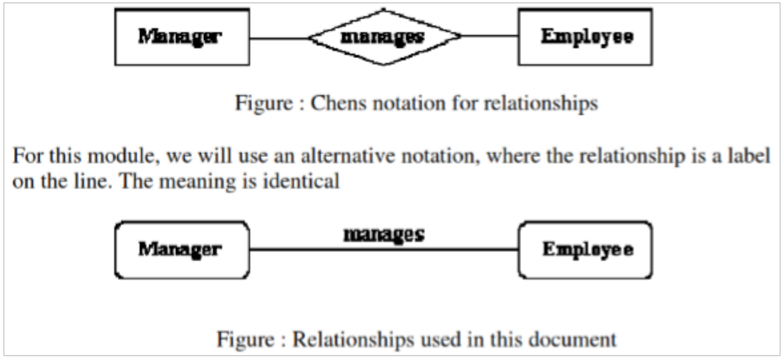


Figure : Chens notation for relationships

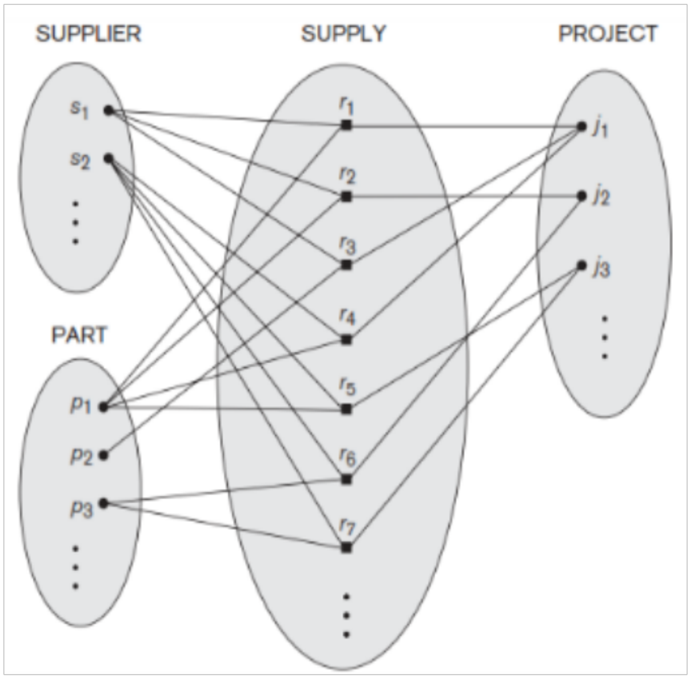
For this module, we will use an alternative notation, where the relationship is a label on the line. The meaning is identical

Figure : Relationships used in this document

Relationship Degree

The degree of a relationship type is the number of participating entity types.

A relationship type of degree two is called **binary**, and one of degree three is called **ternary**.
Hence, the WORKS_FOR relationship is of degree two.
An example of a ternary relationship is SUPPLY.



In supply eg., each relationship instance r_i associates three entities

Supplier	S
Part	P
Project	J

Relationship are generally be of any degree, but the onces most common are binary relationship.

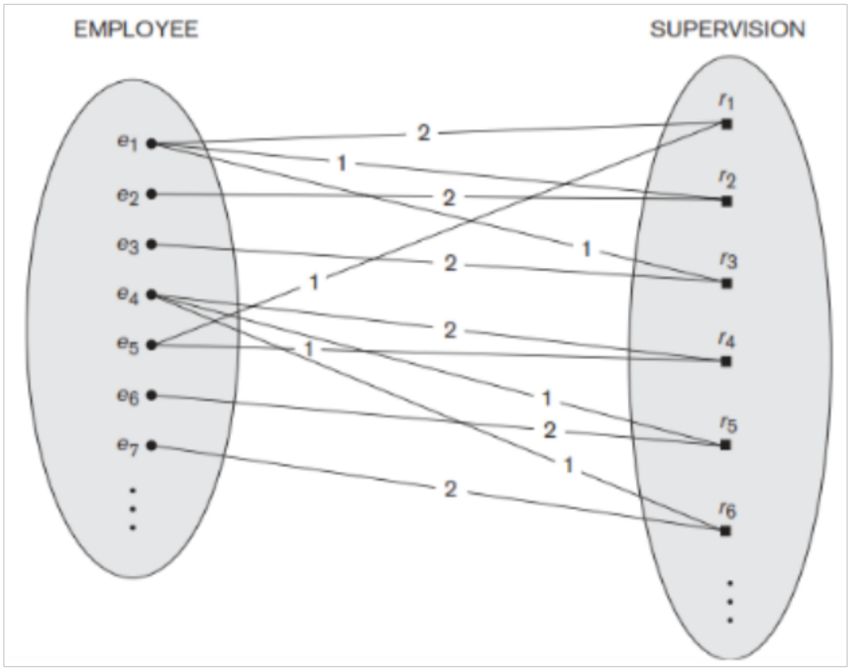
Both MANAGES and WORKS_ON are binary relationships.

Role Names and Recursive Relationships

Each entity type that participates in a relationship type plays a particular **role** in the relationship. The **role name** signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means.

For example, in the WORKS_FOR relationship type, EMPLOYEE plays the role of employee or worker and DEPARTMENT plays the role of department or employer. Role names are not technically necessary in relationship types where all the participating entity types are distinct, since each participating entity type name can be used as the role name.

However, in some cases the same entity type participates more than once in a relationship type in different roles. In such cases the role name becomes essential for distinguishing the meaning of the role that each participating entity plays. Such relationship types are called **recursive relationships**.



The relationship of SUPERVISION between SUPERVISION and EMPLOYEE where the employee entity type plays two roles

1 SUPERVISION (BOSS)

2SUPERVISE (Subordinates)

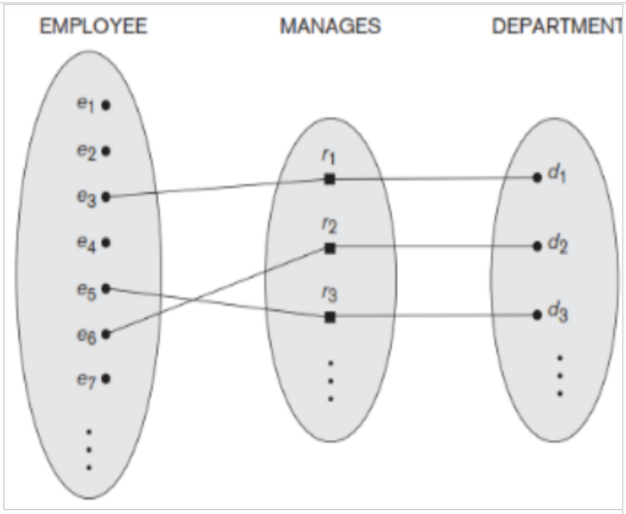
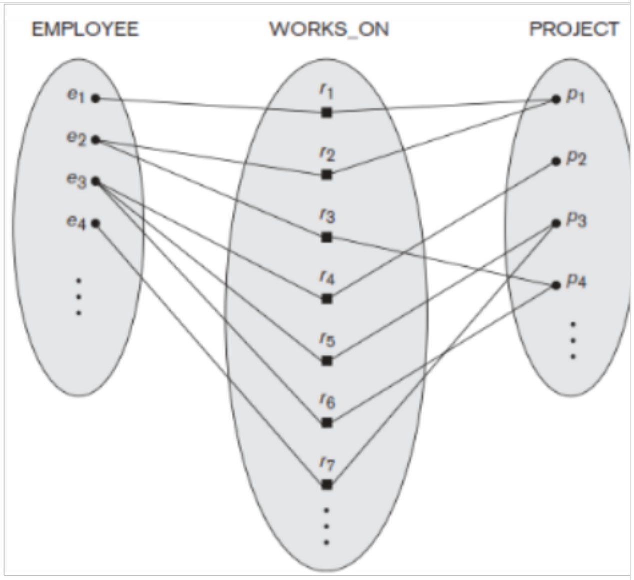
Constraints on Relationship Types

The development of a database using a entity relationship defines certain constraints on the development of the database. For example, if the company has a rule that each employee must work for exactly one department, then we would like to describe this constraint in the schema.

Two main types of binary relationship constraints:
 cardinality ratio
participation

Cardinality Ratios for Binary Relationships: The cardinality ratio for a binary relationship 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
Eg, in the WORKS_FOR binary relationship type, DEPARTMENT: EMPLOYEE is of cardinality ratio 1: N, [meaning that each department can be related to any number of employees, but an employee can be related to (work for) only one department. This means that for this particular relationship WORKS_FOR, a particular department entity can be related to any number of employees (N indicates there is no maximum number). On the other hand, an employee can be related to a maximum of one department.]

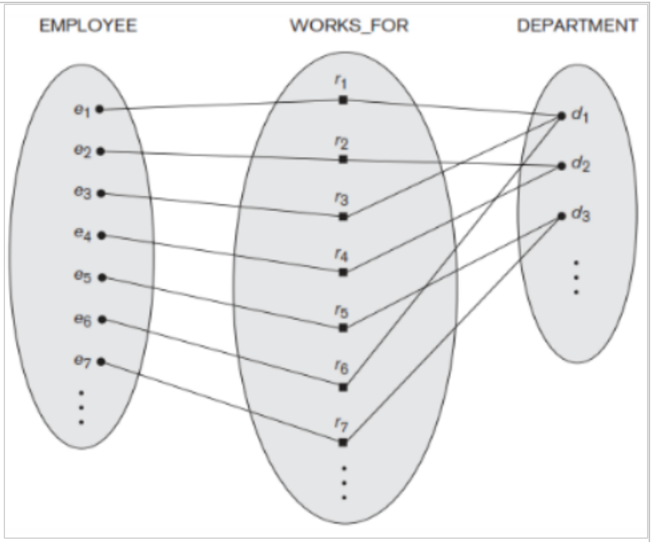
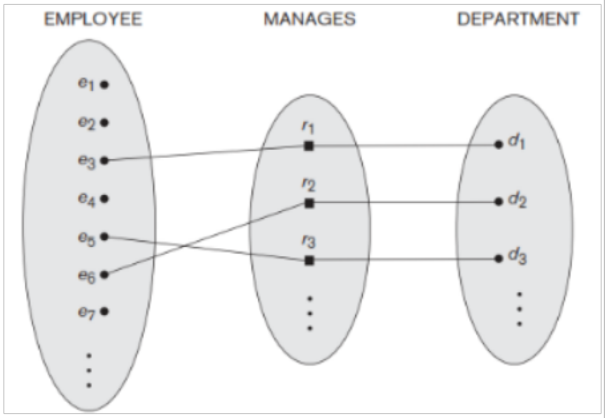
 EMPLOYEE MANAGES DEPARTMENT e_1 e_2 e_3 e_4 e_5 e_6 e_7 r_1 r_2 r_3 d_1 d_2 d_3	An example of a 1:1 binary relationship is MANAGES, which relates a department entity to the employee who manages that department. This represents—at any point in time—an employee can manage one department only and a department can have one manager only.
The relationship type WORKS_ON is of cardinality ratio M:N, because the mini-world rule is that an employee can work on several projects and a project can have several employees.	 EMPLOYEE WORKS_ON PROJECT e_1 e_2 e_3 e_4 r_1 r_2 r_3 r_4 r_5 r_6 r_7 p_1 p_2 p_3 p_4

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. This constraint specifies the minimum number of relationship instances that each entity can participate in, and is sometimes called the **minimum**

cardinality constraint.
There are two types of participation constraints—**total and partial**
In ER diagrams,

- **total participation (or existence dependency)** is displayed as a **double line** connecting the participating entity type to the relationship,
- **partial participation** is represented by a **single line**

If a company policy states that every employee must work for a department, then an employee entity can exist only if it participates in at least one WORKS_FOR relationship instance . Thus, the participation of EMPLOYEE in WORKS_FOR is called **total participation**, meaning that every entity in the total set of employee entities must be related to a department entity via WORKS_FOR. Total participation is also called **existence dependency**	 EMPLOYEE WORKS_FOR DEPARTMENT e_1 e_2 e_3 e_4 e_5 e_6 e_7 r_1 r_2 r_3 r_4 r_5 r_6 r_7 d_1 d_2 d_3
 EMPLOYEE MANAGES DEPARTMENT e_1 e_2 e_3 e_4 e_5 e_6 e_7 r_1 r_2 r_3 d_1 d_2 d_3	We do not expect every employee to manage a department, so the participation of EMPLOYEE in the MANAGES relationship type is **partial**, meaning that some or part of the set of employee entities are related to some department entity via MANAGES, but not necessarily all. We will refer to the cardinality ratio and

1:1 relationship, MANAGES	participation constraints, taken together, as the **structural constraints** of a relationship type.

Attributes of Relationship Types

Relationship types can also have attributes, similar to those of entity types. A good example, is WORKS_ON, each instance of which identifies an employee and a project on which he works. In order to record how many hours are worked by each employee works each project, we include Hours as an attribute of WORKS_ON relationship

Another example is to include the date on which a manager started managing a department via an attribute Start_date for the MANAGES relationship type

In the case of an M:N relationship type(such as WORKS_ON), allowing attributes is vital. In the case of an N:1, 1:N, or 1:1 relationship type, any attributes can be assigned to the entity type opposite from the 1 side. For example, Start_date attribute of the MANAGES relationship type can be given to either the EMPLOYEE or the DEPATMENT entity type.

1.8 Weak entity types

Weak entity - An entity that does not have a key attributes of their own are called weak entity types.

Strong entity - Regular entity types - that have a key attribute - are strong entity types.

Identifying or owner entity - Entities belonging 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. The identifying entity type is also sometimes called the **parent entity type or the dominant entity** type.

The relationship type that relates a weak entity type to its owner the **identifying relationship** of the weak entity type. The weak entity type is also sometimes called the **child entity type or the subordinate entity type.**

A weak entity does not have a distinguishing attribute of its own and mostly are dependent entities, which are part of some another entity.

A weak entity will always be related to one or more strong entities.

They can be also understood as multi-valued attributes

A weak entity type always has a total participation constraint (existence dependency) with respect to its identifying relationship because a weak entity cannot be identified without an owner entity. However, not every existence dependency results in a weak entity type. For example, a DRIVER_LICENSE entity cannot exist unless it is related to a PERSON entity, even though it has its own key (License_number) and hence is not a weak entity.

Consider the entity type DEPENDENT, related to EMPLOYEE, which is used to keep track of the dependents of each employee via a 1:N relationship . In our example, the attributes of DEPENDENT are Name (the first name of the dependent), Birth_date, Sex, and Relationship (to the employee). Two dependents of two distinct employees may, by chance, have the same values for Name, Birth_date, Sex, and Relationship, but they are still distinct entities. They are identified as distinct entities only after determining the particular employee entity to which each dependent is related. Each employee entity is said to own the dependent entities that are related to it.

A weak entity type normally has a partial key, which is the attribute that can uniquely identify weak entities that are related to the same owner entity. In our example, if we assume that no two dependents of the same employee ever have the same first name, the attribute Name of DEPENDENT is the partial key. In the worst case, a composite attribute of all the weak entity's attributes will be the partial key.

In ER diagrams, both a weak entity type and its identifying relationship are distinguished by surrounding their boxes and diamonds with double lines. The partial key attribute is underlined with a dashed or dotted line.

In the preceding example, we could specify a multivalued attribute Dependents for EMPLOYEE, which is a composite attribute with component attributes Name, Birth_date, Sex, and Relationship.

1.9 ER diagrams, naming conventions, design issues

1.9.1 Summary of Notation for ER Diagrams

In ER diagrams the emphasis is on representing the schemas rather than the instances. This is more useful in database design because a database schema changes rarely, whereas the contents of the entity sets change frequently. In addition, the schema is obviously easier to display, because it is much smaller.

Figure 1.7 displays the COMPANY **ER database schema** as an **ER diagram**. We now review the full ER diagram notation. Entity types such as EMPLOYEE, DEPARTMENT, and PROJECT are shown in rectangular boxes. Relationship types such as WORKS_FOR, MANAGES, CONTROLS, and WORKS_ON 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 by a straight line to its entity type or relationship type. Component attributes of a composite attribute are attached to the oval representing the composite attribute, as illustrated by the Name attribute of EMPLOYEE.

Multivalued attributes are shown in double ovals, as illustrated by the Locations attribute of DEPARTMENT. Key attributes have their names underlined. Derived attributes are shown in dotted ovals, as illustrated by the Number_of_employees attribute of DEPARTMENT. Weak entity types are distinguished by being placed in double rectangles and by having their identifying relationship placed in double diamonds, as illustrated by the DEPENDENT entity type and the DEPENDENTS_OF identifying relationship type. The partial key of the weak entity type is underlined with a dotted line.

In Figure 1.7 the cardinality ratio of each *binary* relationship type is specified by attaching a 1, M, or N on each participating edge. The cardinality ratio of DEPARTMENT:EMPLOYEE in MANAGES is 1:1, whereas it is 1:N for DEPARTMENT: EMPLOYEE in WORKS_FOR, and M:N for WORKS_ON. The participation constraint is specified by a single line for partial participation and by double lines for total participation (existence dependency).

In Figure 1.7 we show the role names for the SUPERVISION relationship type because the same EMPLOYEE entity type plays two distinct roles in that relationship. Notice that the cardinality ratio is 1:N from supervisor to supervisee because each employee in the role of supervisee has at most one direct supervisor, whereas an employee in the role of supervisor can supervise zero or more employees. Figure 1.8 summarizes the conventions for ER diagrams.

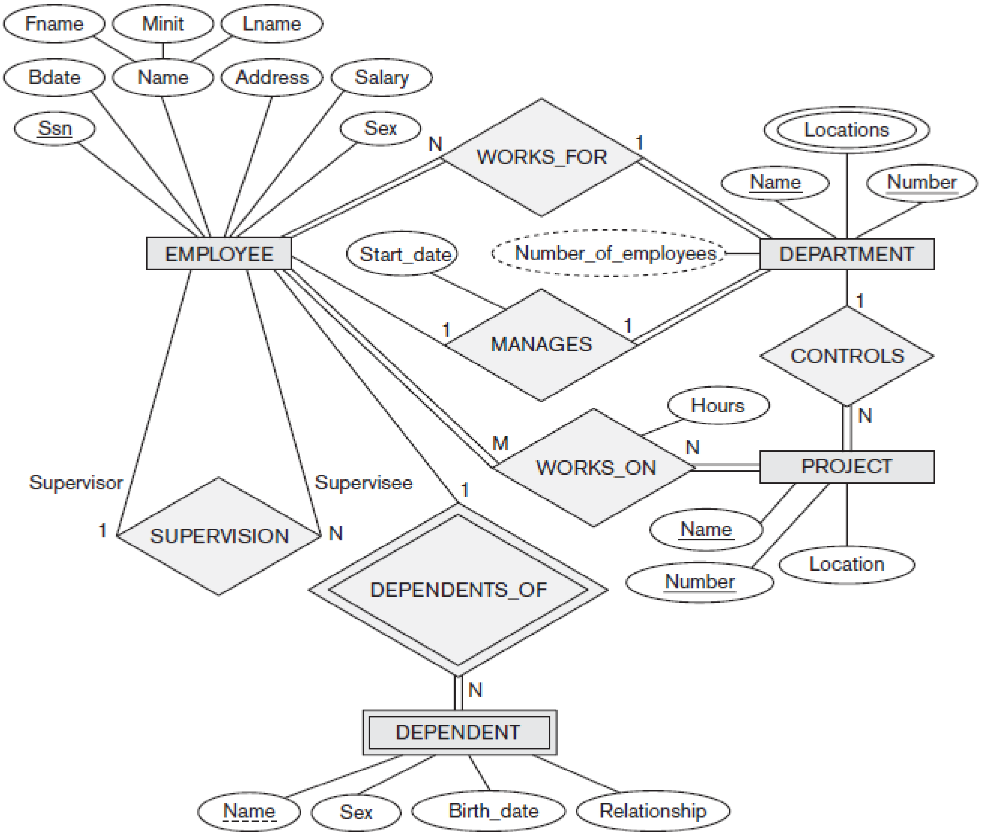


Fig. 1.7: An ER schema diagram for the COMPANY database

1.9.2 Proper Naming of Schema Constructs

When designing a database schema, the choice of names for entity types, attributes, relationship types, and (particularly) roles is not always straightforward. One should choose names that convey, as much as possible, the meanings attached to the different constructs in the schema. We choose to use *singular names* for entity types, rather than plural ones, because the entity type name applies to each individual entity belonging to that entity type. In our ER diagrams, we will use the convention that entity type and relationship type names are uppercase letters, attribute names have their initial letter capitalized, and role names are lowercase letters. We have used this convention in Figure 1.7.

As a general practice, given a narrative description of the database requirements, the *nouns* appearing in the narrative tend to give rise to entity type names, and the *verbs* tend to indicate names of relationship types. Attribute names generally arise from additional nouns that describe the nouns corresponding to entity types.

Another naming consideration involves choosing binary relationship names to make the ER diagram of the schema readable from left to right and from top to bottom. We have generally followed this guideline in Figure 1.7. To explain this naming convention further, we have one exception to the convention in Figure 1.7—the DEPENDENTS_OF relationship type, which reads from bottom to top. When we describe this relationship, we can say that the DEPENDENT entities (bottom entity type) are DEPENDENTS_OF (relationship name) an EMPLOYEE (top entity type). To change this to read from top to bottom, we could rename the relationship type to HAS_DEPENDENTS, which would then read as follows: An EMPLOYEE entity (top entity type) HAS_DEPENDENTS (relationship name) of type DEPENDENT (bottom entity type).

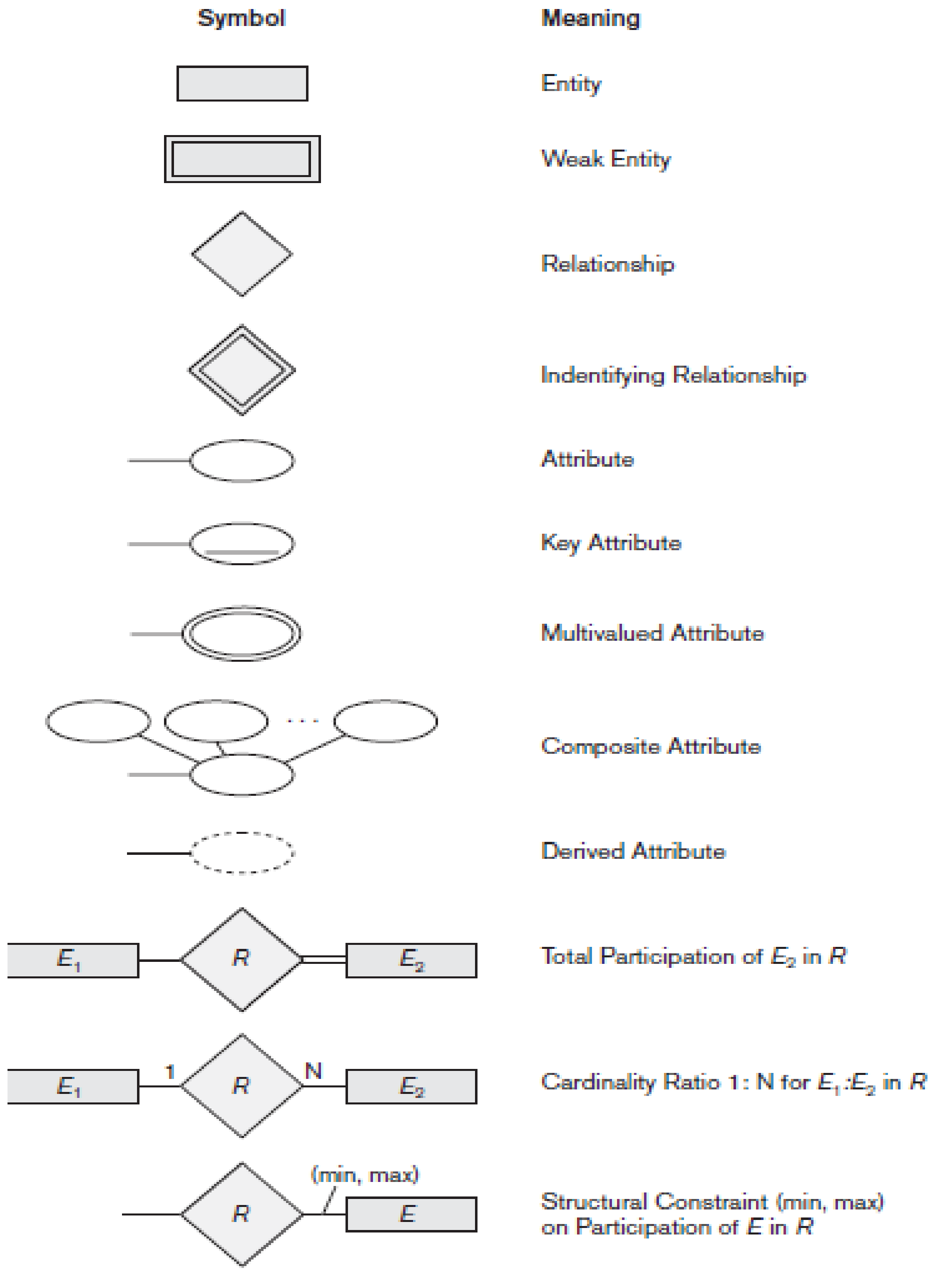

Symbol
Meaning
Entity
Weak Entity
Relationship
Indentifying Relationship
Attribute
Key Attribute
Multivalued Attribute
. . .
Composite Attribute
Derived Attribute
E_1
R
E_2
Total Participation of E_2 in R
E_1
1
R
N
E_2
Cardinality Ratio 1: N for E_1:E_2 in R
(min, max)
R
E
Structural Constraint (min, max)
on Participation of E in R

Fig. 1.8: Summary of the notations of the ER diagram

1.9.3 Design Choices for ER Conceptual Design

It is occasionally difficult to decide whether a particular concept in the miniworld should be modeled as an entity type, an attribute, or a relationship type. In this section, we give some brief guidelines as to which construct should be chosen in particular situations. In general, the schema design process should be considered an iterative refinement process, where an initial design is created and then iteratively refined until the most suitable design is reached. Some of the refinements that are often used include the following:

- A concept may be first modeled as an attribute and then refined into a relationship because it is determined that the attribute is a reference to another entity type. It is often the case that pair of such attributes that are inverses of one another are refined into a binary relationship. It is important to note that in our notation, once an attribute is replaced by a relationship, the attribute itself should be removed from the entity type to avoid duplication and redundancy.
- Similarly, an attribute that exists in several entity types may be elevated or promoted to an independent entity type. For example, suppose that several entity types in a UNIVERSITY database, such as STUDENT, INSTRUCTOR, and COURSE, each has an attribute Department in the initial design; the designer may then choose to create an entity type DEPARTMENT with a single attribute Dept_name and relate it to the three entity types (STUDENT, INSTRUCTOR, and COURSE) via appropriate relationships. Other attributes/ relationships of DEPARTMENT may be discovered later.
- An inverse refinement to the previous case may be applied—for example, if an entity type DEPARTMENT exists in the initial design with a single attribute Dept_name and is related to only one other entity type, STUDENT. In this case, DEPARTMENT may be reduced or demoted to an attribute of STUDENT.