

UNIT 1 : INTRODUCTION

Database systems have become an essential component of life in modern society, in that many frequently occurring events trigger the accessing of at least one database: bibliographic library searches, bank transactions, hotel/airline reservations, grocery store purchases, etc.,

Traditional vs. more recent applications of databases:

The applications like bank transactions, hotel/airline reservations, grocery store purchases, etc., are all "traditional" ones for which the use of rigidly-structured textual and numeric data suffices.

Recent advances have led to the application of database technology to a wider class of data. Examples include **multimedia** databases (involving pictures, video clips, and sound messages) and **geographic** databases (involving maps, satellite images).

Database :

Database is a collection of related data.

Properties of Database :

A database has the following implicit properties:

- 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.
- A database is a logically coherent collection of data with some inherent meaning.
- A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

Database Management System :

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.

- (i) **Defining** a database involves specifying the data types, structures, and constraints of the data to be stored in the data base. 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**.
- (ii) **Constructing** the database is the process of storing the data on some storage medium that is controlled by the DBMS.

(iii) **Manipulating** 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.

(iv) **Sharing** a database allows multiple users and programs to access the database simultaneously.

Other important functions provided by the DBMS includes :

(i) **Protecting the database : Protection** includes *system protection* against hardware or software malfunction (or crashes) and *security protection* against unauthorized or malicious access.

(ii) **Maintaining it over a long period of time** : The DBMS must be able to **maintain** the database system by allowing the system to evolve as requirements change over time.

Database System

The database and DBMS software together a **database system**. The below Figure shows the database system.

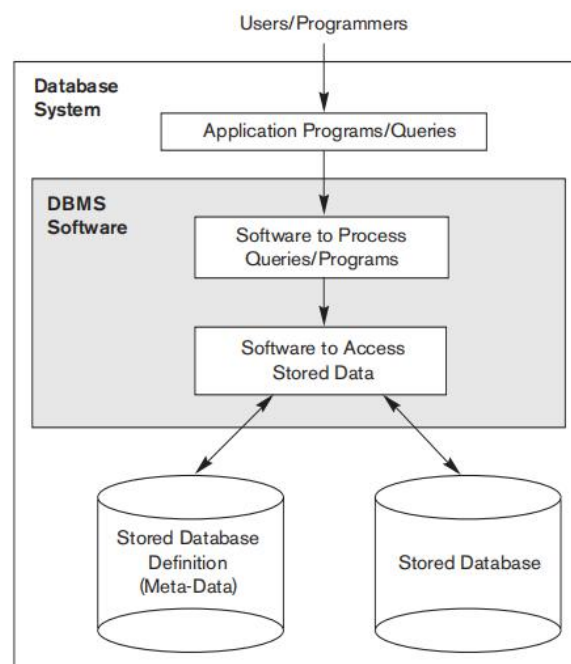


Fig : A Simplified Database Environment

Example :

Let us consider a **UNIVERSITY database** for maintaining information concerning students, courses, and grades in a university environment.

Below Figure shows the database structure and a few sample data for such a database.

The database is organized as five files, each of which stores **data records** of the same type. The STUDENT file stores data on each student, the COURSE file stores data on each course, the SECTION file stores data

on each section of a course, the GRADE_REPORT file stores the grades that students receive in the various sections they have completed, and the PREREQUISITE file stores the prerequisites of each course.

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 (b) : A database that stores student and course information.

Requirements Specification and Analysis:

- Design of a new application for an existing database or design of a brand new database starts off with a phase called **requirements specification and analysis**.
 - (i) **Conceptual design:** These requirements (which include design interactions, experience and strategies) are documented in detail.
 - (ii) **Logical design:** The design is then translated to a **logical design** that can be expressed in a data model implemented in a commercial DBMS. (Logical relationship, constraints etc.).
 - (iii) The final stage is **Physical design:** 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.

Characteristics of the Database Approach:

The main characteristics of the database approach versus the file-processing approach are :

(i) Self-Describing Nature of a Database System :

- ✓ A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints.
- ✓ This definition is stored in the *DBMS 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 (Figure a).*
- ✓ The catalog is used by the DBMS software and also by database users who need information about the database structure. A general-purpose DBMS software package is not written for a specific database application.
- ✓ Therefore, 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.
- ✓ The DBMS software must work equally well with any number of database applications—for example, a university database, a banking database, or a company database—as long as the database definition is stored in the catalog.

RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
....
....
....
Prerequisite_number	XXXXNNNN	PREREQUISITE

Fig (c) : An example of a database catalog for the database in Figure (b)

(ii) 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 that file. By contrast, 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. We call this property ***program-data independence***.

In some types of database systems, such as object-oriented and object-relational systems users can define operations on data as part of the database definitions. An operation (also called a function or method) is specified in two parts.

- ✓ The interface (or signature) of an operation includes the operation name and the data types of its arguments (or parameters).
- ✓ The implementation (or method) of the operation is specified separately and can be changed without affecting the interface.

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*.

The characteristic that allows program-data independence and program-operation independence is called *data abstraction*.

Data Item Name	Starting Position in Record	Length in Characters (bytes)
Name	1	30
Student_number	31	4
Class	35	1
Major	36	4

Fig (d) : Internal storage format for a STUDENT record, based on the database catalog in Figure (c)

(iii) Support of Multiple Views of the Data:

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 Figure (b) may be interested only in accessing and printing the transcript of each student; the view for this user is shown in Figure (e1). 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 (e2).

Below figure shows the Two views derived from the database in Figure (b).

TRANSCRIPT

Student_name	Student_transcript				
	Course_number	Grade	Semester	Year	Section_id
Smith	CS1310	C	Fall	08	119
	MATH2410	B	Fall	08	112
Brown	MATH2410	A	Fall	07	85
	CS1310	A	Fall	07	92
	CS3320	B	Spring	08	102
	CS3380	A	Fall	08	135

(a)

Fig (e1): The TRANSCRIPT view

COURSE_PREREQUISITES

Course_name	Course_number	Prerequisites
Database	CS3380	CS3320
		MATH2410
Data Structures	CS3320	CS1310

(b)

Fig (e2): The COURSE_PREREQUISITES view.

(iv) Sharing of Data and Multiuser Transaction Processing :

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 software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.

Actors on the Scene:

The people whose jobs involve the day-to-day use of a large database - *actors on the scene*.

The people who work to maintain the database system environment but who are not actively interested in the database contents as part of their daily job - *workers behind the scene*

(i) Database Administrators :

- ✓ In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software.
- ✓ Administering these resources is the responsibility of the **database administrator (DBA)**.
- ✓ The DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed.
- ✓ The DBA is accountable for problems such as security breaches and poor system response time.
- ✓ In large organizations, the DBA is assisted by a staff that carries out these functions.

(ii) 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.
- ✓ It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements and to create a design that meets these requirements.
- ✓ The final database design must be capable of supporting the requirements of all user groups.

(iii) End Users :

- ✓ **End users** are the people whose jobs require access to the database for querying, updating, and generating reports.

The 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 function revolves around constantly querying and updating the database, called **canned transactions**—that have been carefully programmed and tested.
- **Standalone users** maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces.

(iv) 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 standard 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—commonly referred to as **software developers or software engineers**.

Workers behind the Scene :

- **DBMS system designers/implementers:** provide the DBMS software that is at the foundation of all this!
- **Tool developers:** design and implement software tools facilitating database system design, performance monitoring, creation of graphical user interfaces, prototyping, etc.
- **Operators and maintenance personnel:** responsible for the day-to-day operation of the system.

Advantages of Using the DBMS Approach

1. Controlling Redundancy
2. Restricting Unauthorized Access
3. Providing Persistent Storage for Program Objects
4. Providing Storage Structures for Efficient Query Processing
5. Providing Backup and Recovery.
6. Providing Multiple User Interfaces
7. Representing Complex Relationships Among Data
8. Enforcing Integrity Constraints
9. Permitting Inference and Actions Via Rules

10. Potential for enforcing standards
11. Reduced application development time
12. Flexibility to change data structures
13. Availability of up-to-date information
14. Economies of scale.

Database System Concepts and Architecture

One fundamental characteristic of the database approach is that it provides some level of *data abstraction*.

Data abstraction generally refers to the suppression of details of data organization and storage, and the highlighting of the essential features for an improved understanding of data.

Data model—a collection of concepts that can be used to describe the structure of a database.

Categories of Data Models:

Data models have been categorized according to the types of concepts they use to describe the database structure.

- (i) ***High-level or conceptual data models*** provide concepts that are close to the way many users perceive data

- Conceptual data models use concepts such as entities, attributes, and relationships.

Entity : Represents a real-world object or concept.

Ex : Project, Car, Employee form the miniworld that is described in the database.

Attribute: Property that describes an entity, such as the employee's name or salary.

Relationship: Association between two or more entities.

Ex : a works-on relationship between an employee and a project.

- (ii) ***low-level or physical data models*** provide concepts that describe the details of how data is stored on the computer storage media.

- (iii) ***Representational (or implementation) data models*** which provide concepts that may be easily understood by end users. Representational data models hide many details of data storage on disk but can be implemented on a computer system directly.

Data Models:

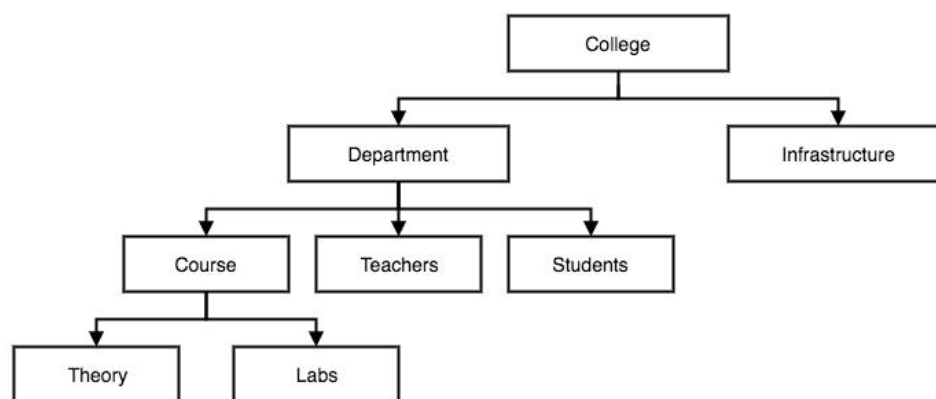
A Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system.

Types of Data Models:

1. Hierarchical Model
2. Network Model
3. Relational Model
4. Object oriented Model
5. E-R Model.

1. Hierarchical Model :

This database model organizes data into a tree-like-structure, with a single root, to which all the other data is linked. The hierarchy starts from the Root data, and expands like a tree, adding child nodes to the parent nodes. In this model, a child node will only have a single parent node.

**2. Network Model:**

This is an extension of the Hierarchical model. In this model data is organised more like a graph, and are allowed to have more than one parent node.

In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships.

This was the most widely used database model, before Relational Model was introduced.



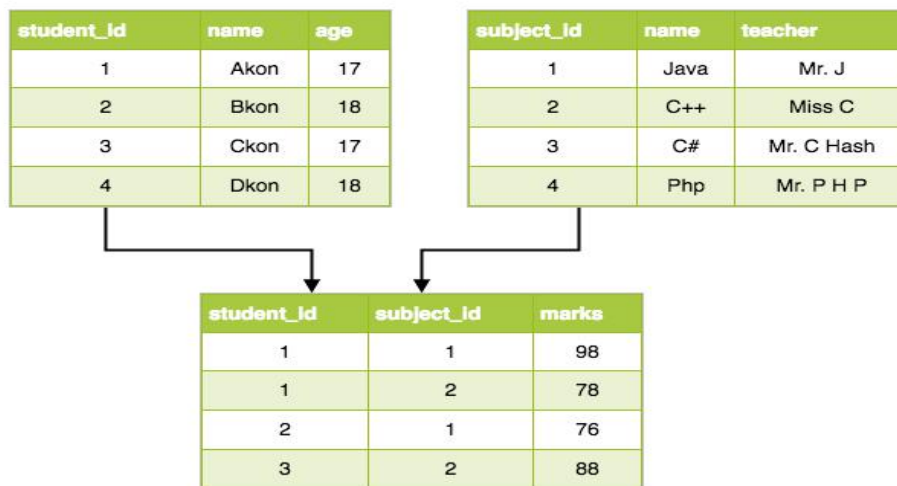
3. Relational Model :

In this model, data is organised in two-dimensional tables and the relationship is maintained by storing a common field.

This model was introduced by E.F Codd in 1970, and since then it has been the most widely used database model, infact, we can say the only database model used around the world.

The basic structure of data in the relational model is tables. All the information related to a particular type is stored in rows of that table.

Hence, tables are also known as relations in relational model.



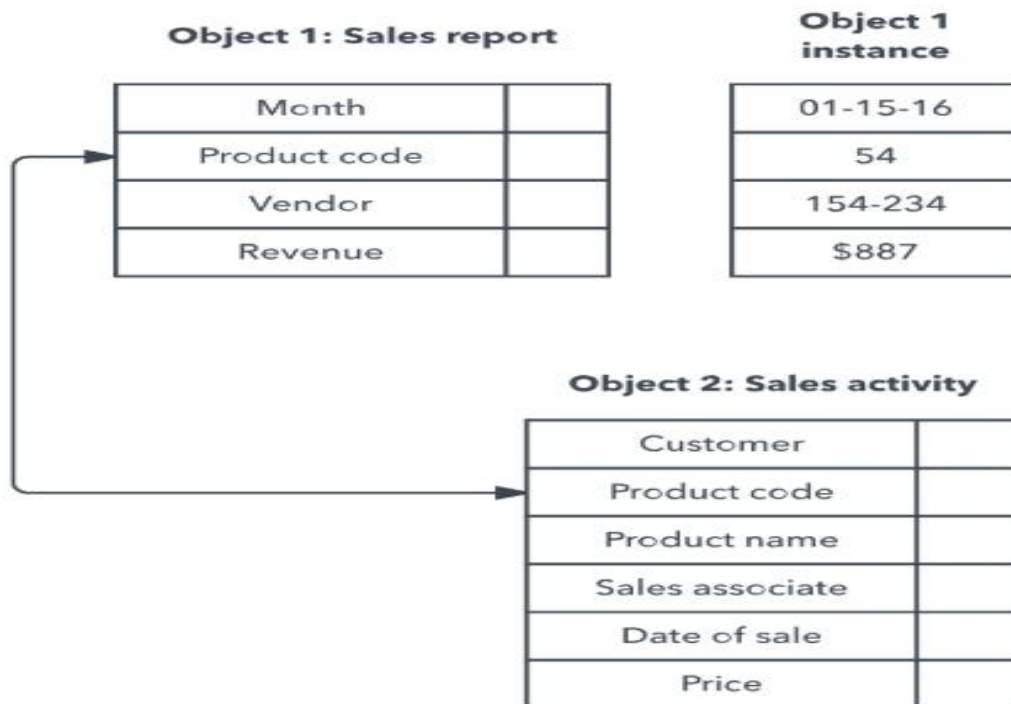
4. Object Oriented Data Model:

In the object oriented data model (OODM), both data and their relationships are contained in a single structure known as an object.

In turn, the OODM is the basis for the object-oriented database management system (OODBMS).

The Components of the Object Oriented Data Model

- An object is an abstraction of a real-world entity.
- Attributes describe the properties of an object.
- Objects that share similar characteristics are grouped in classes. A class is a collection of similar objects with shared structure (attributes) and behavior (methods).
- Classes are organized in a class hierarchy. The class hierarchy resembles an upside-down tree in which each class has only one parent.
- Inheritance is the ability of an object within the class hierarchy to inherit the attributes and methods of the classes above it.



Schemas, Instances, and Database State:

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

Schema Diagram A displayed schema is called a schema diagram. Figure 2a shows a schema diagram for the database shown in Figure (b) in chap 1.

Schema Construct: structure of each record type but not the actual instances of records.

Ex : each object in the schema—such as STUDENT or COURSE

Snapshot / Database State: The data in the database at a particular moment in time is called a **database state** or **snapshot**. (or) It is also called the *current* set of **occurrences** or **instances** in the Database.

The **schema** is sometimes called the **intension**.

Database state is called an **extension** of the schema.

Schema Evolution: It is the ability of a database system to respond to changes in the real world by allowing the schema to evolve.

For example, we may decide that another data item needs to be stored for each record in a file, such as adding the Date_of_birth to the STUDENT schema in Figure 2a. This is known as **schema evolution**.

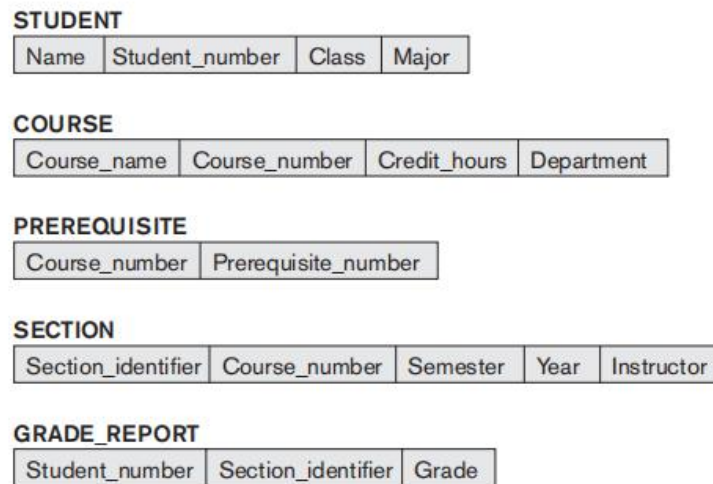


Fig 2a: Schema diagram for the database in fig(a) in chapter 1.

Three-Schema Architecture and Data Independence :

The Three-Schema Architecture :

The goal of the three-schema architecture, illustrated in Figure 2b, is to separate the user applications from the physical database.

In this architecture, schemas can be defined at the following three levels:

1. **The internal level** has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
2. **The conceptual level** has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints.
3. **The external or view level** includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group.

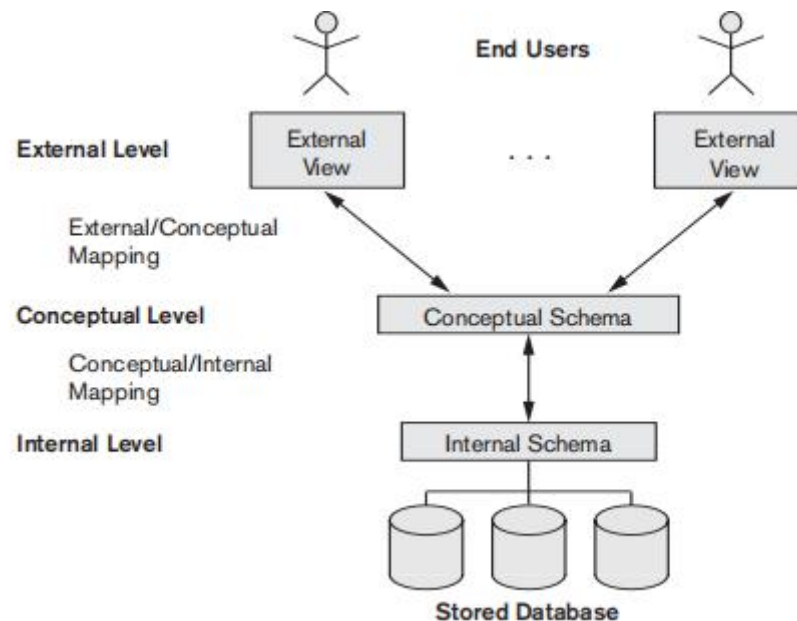


Fig 2b : Three Schema Architecture.

- The three schemas are only descriptions of data
- The stored data that actually exists is at the physical level only.
- In a DBMS based on the three-schema architecture, each user group refers to its own external schema.
- Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database.
- If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view.
- The processes of transforming requests and results between levels are called **mappings**.

Data Independence

Data Independence is defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level.

We can define two types of data independence:

- Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item).
- Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well. Changes to the internal schema may be needed because some physical files were reorganized.

Database Languages and Interfaces

Database languages:

- (i) **Data Definition Language (DDL)**: is used by the DBA and by database designers to define both schemas.
- (ii) **Storage Definition Language (SDL)**: is used to specify the internal schema. The mappings between the two schemas may be specified in either one of these languages.
- (iii) **View Definition Language (VDL)**: To specify user views and their mappings to the conceptual schema, but in most DBMSs *the DDL is used to define both conceptual and external schemas*. In relational DBMSs, SQL is used in the role of VDL to define user or application **views** as results of predefined queries
- (iv) **Data Manipulation Language (DML)**: Once the database schemas are compiled and the database is populated with data, users must have some means to manipulate the database. Typical manipulations include retrieval, insertion, deletion, and modification of the data. The DBMS provides a set of operations or a language called the **(DML)** for these purposes.

There are two main types of DMLs.

- a) **High-Level Or Nonprocedural DML** can be used on its own to specify complex database operations concisely. Many DBMSs allow high-level DML statements either to be entered interactively from a display monitor or terminal or to be embedded in a general-purpose programming language. In the latter case, DML statements must be identified within the program so that they can be extracted by a precompiler and processed by the DBMS.
- b) A **Lowlevel Or Procedural DML** *must* be embedded in a general-purpose programming language. This type of DML typically retrieves individual records or objects from the database and processes each separately., it needs to use programming language constructs, such as looping, to retrieve and process each record from a set of records.
 - Low-level DMLs are also called **record-at-a-time** DMLs because of this property.
 - DL/1, a DML is a low-level DML that uses commands such as GET UNIQUE, GET NEXT, or GET NEXT WITHIN PARENT to navigate from record to record within a hierarchy of records in the database.
 - Highlevel DMLs, such as SQL, can specify and retrieve many records in a single DML statement; therefore, they are called **set-at-a-time** or **set-oriented** DMLs.
 - A query in a high-level DML often specifies *which* data to retrieve rather than *how* to retrieve it; therefore, such languages are also called **declarative**.
 - Whenever DML commands, whether high level or low level, are embedded in a general-purpose programming language, that language is called the **host language** and the DML is called the **data sublanguage**.

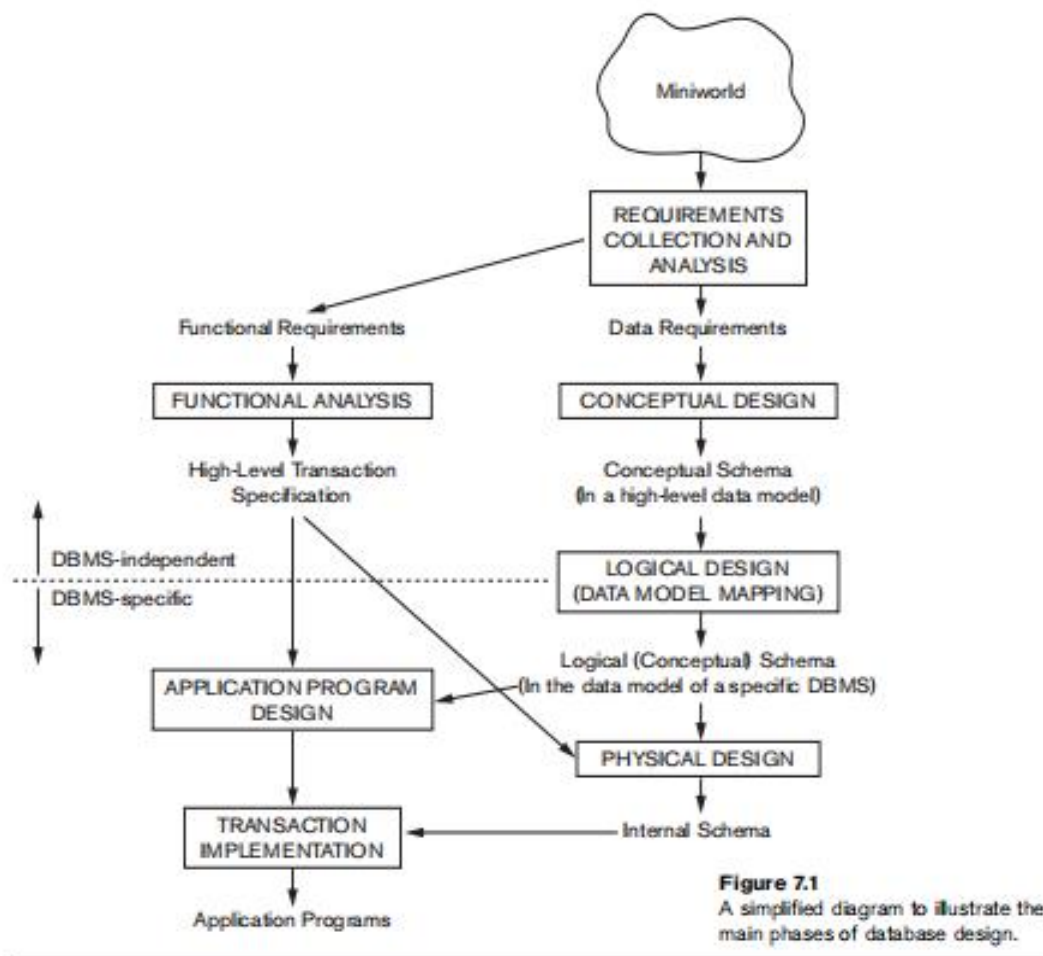
- On the other hand, a high-level DML used in a standalone interactive manner is called a **query language**.

DBMS Interfaces:

- (i) **Menu-Based Interfaces for Web Clients or Browsing.** These interfaces present the user with lists of options (called **menus**) that lead the user through the formulation of a request.
- (ii) **Forms-Based Interfaces.** A forms-based interface displays a form to each user. Users can fill out all of the **form** entries to insert new data, or they can fill out only certain entries, in which case the DBMS will retrieve matching data for the remaining entries.
- (iii) **Graphical User Interfaces.** A GUI typically displays a schema to the user in diagrammatic form. The user then can specify a query by manipulating the diagram. In many cases, GUIs utilize both menus and forms. Most GUIs use a **pointing device**, such as a mouse, to select certain parts of the displayed schema diagram.
- (iv) **Natural Language Interfaces.** These interfaces accept requests written in English or some other language.
- (v) **Speech Input and Output.** Limited use of speech as an input query and speech as an answer to a question or result of a request is becoming commonplace. Applications with limited vocabularies such as inquiries for telephone directory, flight arrival/departure, and credit card account information are allowing speech for input and output to enable customers to access this information.
- (vi) **Interfaces for Parametric Users.** Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly. For example, a teller is able to use single function keys to invoke routine and repetitive transactions such as account deposits or withdrawals, or balance inquiries. Systems analysts and programmers design and implement a special interface for each known class of naive users. Usually a small set of abbreviated commands is included, with the goal of minimizing the number of keystrokes required for each request.
- (vii) **Interfaces for the DBA.** Most database systems contain privileged commands that can be used only by the DBA staff. These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, and reorganizing the storage structures of a database.

Data Modelling Using the Entity-Relationship (ER) Model

Main Phases Of Database Design :



The above fig 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 database users to understand and document their **data requirements**. 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 description of the data requirements of the users and includes detailed descriptions of the entity types, relationships, and constraints.

The next step in database design is the actual implementation of the database, 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.

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.

Entity Types, Entity Sets, Attributes, and Keys :

Entities and Attributes :

Entity : is a *thing* in the real world with an independent existence. An entity may be an object with a physical existence (for example, a particular person, car, house, or employee) or it may be an object with a conceptual existence (for instance, a company, a job, or a university course).

Attribute : The properties that describe entity.

For example, an EMPLOYEE entity may be described by the employee's name, age, address, salary, and job.

Types of Attributes :

Several types of attributes occur in the ER model:

- a. Simple versus Composite,
- b. SingleValued versus Multivalued, &
- c. Stored versus Derived.
- d. Null Value for an Attribute.
- e. Complex attribute.

a. Composite versus Simple (Atomic) Attributes :

Composite attributes : can be divided into smaller subparts.

For example, the Address attribute of the EMPLOYEE entity shown in below Figure can be subdivided into Street_address, City, State, and Zip,3 with the values '2311 Kirby', 'Houston', 'Texas', and '77001.'

Simple Or Atomic Attributes: The attributes that are not divisible are called Simple Or Atomic Attributes.

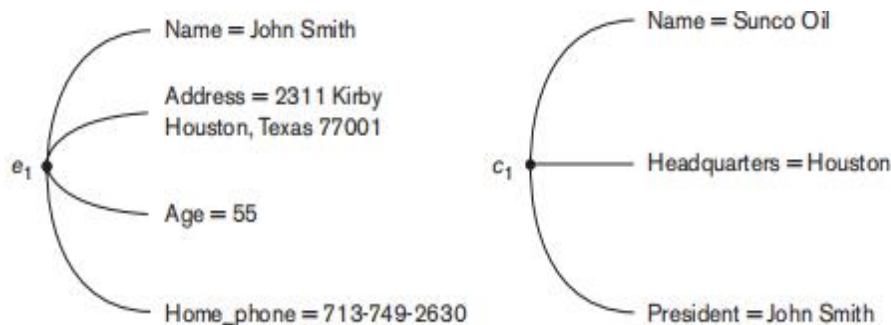


Figure 7.3
Two entities,
EMPLOYEE e_1 , and
COMPANY c_1 , and
their attributes.

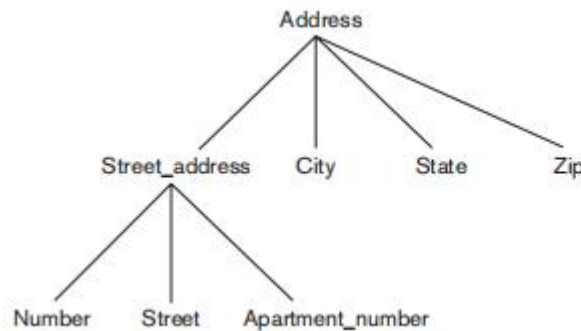


Figure 7.4
A hierarchy of composite attributes.

b. Single-Valued versus Multivalued Attributes.

Single Valued Attributes: The attributes have a single value for a particular entity; such attributes are called **single-valued**.

For example, Age is a single-valued attribute of a person.

Multivalued Attributes: A multivalued attribute may have a multiple value for a particular entity; such attributes are called **multi-valued**.

For example, the degree of a person.

c. Stored versus Derived Attributes.

In some cases, two (or more) attribute values are related—for example, the Age and Birth_date 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 Birth_date.

The Age attribute is hence called a **derived attribute (attribute derived from other attribute)** and is said to be **derivable from** the Birth_date attribute, which is called a **stored attribute**.

d. NULL Values.

It is a value to be considered when a particular entity may not have an applicable value for an attribute.

For example, the Apartment_number attribute of an address applies only to addresses that are in apartment buildings and not to other types of residences, such as single-family homes.

Similarly, a College_degrees attribute applies only to people with college degrees. For such situations, a special value called NULL is created.

The unknown category of NULL can be further classified into two cases.

The first case arises when it is known that the attribute value exists but is missing—for instance, if the Height attribute of a person is listed as NULL.

The second case arises when it is not known whether the attribute value exists—for example, if the Home_phone attribute of a person is NULL.

e. Complex Attributes:

Combination of a composite and multivalued attributes. It can be represented as arbitrary nesting by grouping components of a composite attribute between parentheses () and separating the components with commas, and by displaying multivalued attributes between braces { }. Such attributes are called complex attributes.

For example, if a person can have more than one residence and each residence can have a single address and multiple phones, an attribute Address_phone for a person can be specified as shown in Figure 7.5 Both Phone and Address are themselves composite attributes.

```
{Address_phone( {Phone(Area_code,Phone_number)},Address(Street_address
(Number,Street,Apartment_number),City,State,Zip) )}
```

Figure 7.5
A complex attribute:
Address_phone.

Entity Types, Entity Sets, Keys, and Value Sets :

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.

Figure(aa) shows two entity types: EMPLOYEE and COMPANY, and a list of some of the attributes for of their attributes.

Entity Sets: 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

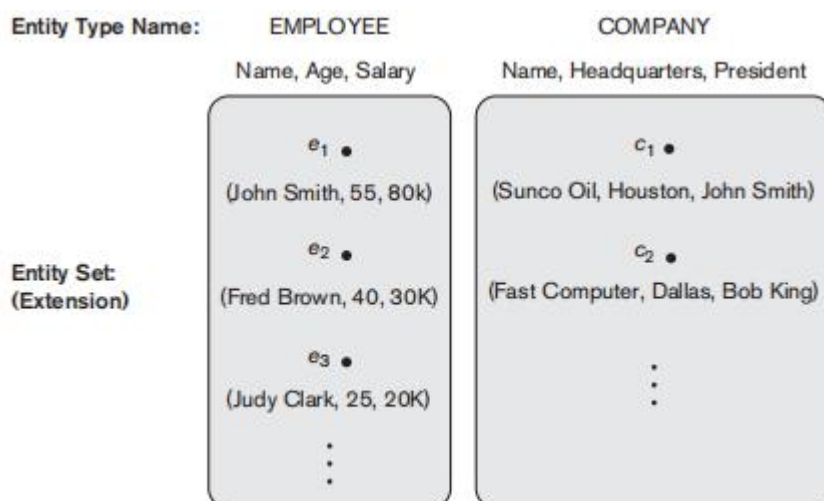


Fig (aa) : Two entity types,EMPLOYEE and COMPANY, and some member entities of each.

Key Attributes of an Entity Type: An entity type usually has one or more attributes whose values are distinct for each individual entity in the entity set. Such an attribute is called a **key attribute**, and its values can be used to identify each entity uniquely.

Ex : USN, SSN etc.

Some entity types have more than one key attribute. For example, each of the Vehicle_id and Registration attributes of the entity type CAR (shown in below Figure) is a key in its own right.

Weak Entity Type : Entity types that do not have key attributes of their own are called weak entity type.

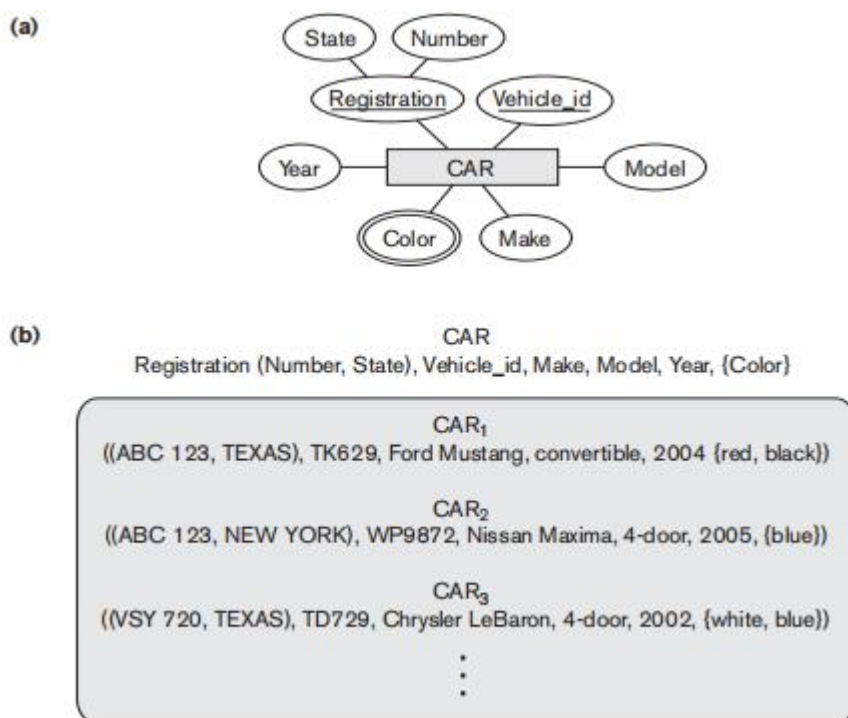


Fig: The CAR entity type with two key attributes, Registration and Vehicle_id. (a) ER diagram notation. (b) Entity set with three entities.

Value Sets (Domains) of Attributes: Each simple attribute of an entity type is associated with a value set (or domain of values).

It specifies the set of values that may be assigned to that attribute for each individual entity. In Figure (aa), if the range of ages allowed for employees is between 16 and 70, we can specify the value set of the Age attribute of EMPLOYEE to be the set of integer numbers between 16 and 70.

Value sets are not displayed in ER diagrams.

Initial Conceptual Design of the COMPANY Database

1. Define the entity types for the COMPANY database, based on the requirements described .
2. After defining several entity types and their attributes here, introduce the concept of a relationship.
3. According to the requirements listed in the above step , we can identify four entity types—one corresponding to each of the four items in the specification.

- a) 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.
- b) An entity type **PROJECT** with attributes Name, Number, Location, and Controlling_department. Both Name and Number are (separate) key attributes.
- c) An entity type **EMPLOYEE** with attributes Name, Ssn, Sex, Address, Salary, Birth_date, Department, and Supervisor. Both Name and Address may be composite attributes (Name—First_name, Middle_initial, Last_name—or of Address.)
- d) An entity type **DEPENDENT** with attributes Employee, Dependent_name, Sex, Birth_date, and Relationship (to the employee).

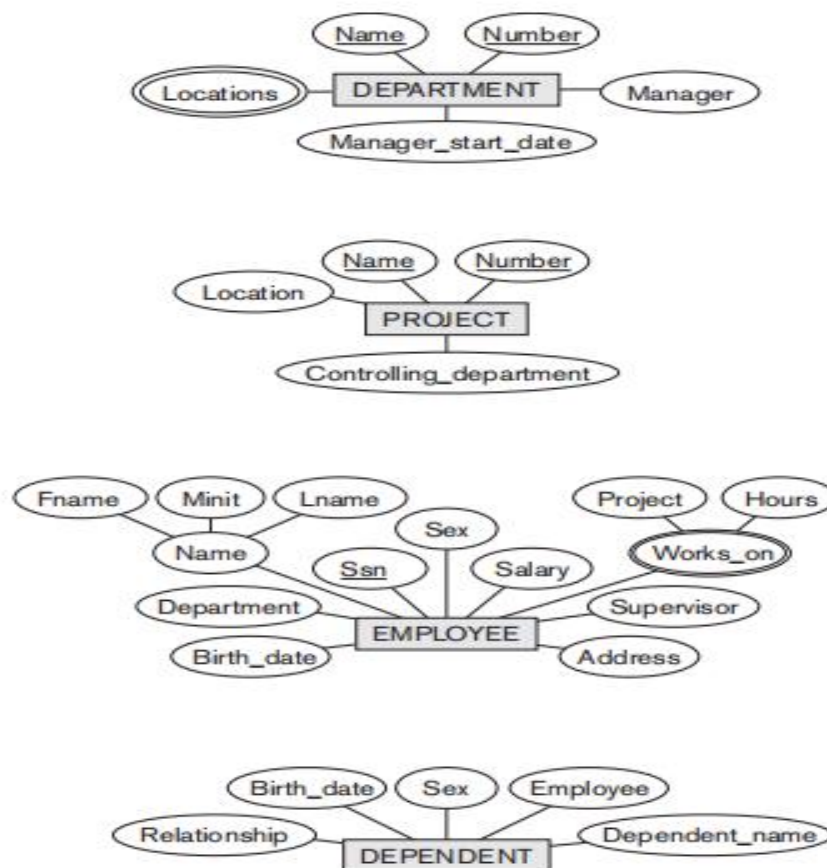


Fig : Preliminary design of entity types for the COMPANY database.

Relationship Types, Relationship Sets, Roles, and Structural Constraints

- There are several implicit relationships among the various entity types.
- Whenever an attribute of one entity type refers to another entity type, some relationship exists.
- For example, the attribute Manager of DEPARTMENT refers to an employee who manages the department.

A Relationship Type : R among n entity types E_1, E_2, \dots, E_n defines a set of association

A Relationship Set : Entities from the entity types.

Relationship Instances : The relationship set R is a set of relationship instances r_i , where each r_i associates n individual entities (e_1, e_2, \dots, e_n), and each entity e_j in r_i is a member of entity set E_j .

For example, consider a relationship type WORKS_FOR between the two entity types EMPLOYEE and DEPARTMENT, which associates each employee with the department for which the employee works in the corresponding entity set. Each relationship instance in the relationship set WORKS_FOR associates one EMPLOYEE entity and one DEPARTMENT entity.

Figure(bb) illustrates this example, where each relationship instance r_i is shown connected to the EMPLOYEE and DEPARTMENT entities that participate in r_i .

In the miniworld represented by Figure(bb), employees e_1, e_3 , and e_6 work for department d_1 ; employees e_2 and e_4 work for department d_2 ; and employees e_5 and e_7 work for department d_3 .

In ER diagrams, relationship types are displayed as **diamond-shaped boxes**, which are connected by straight lines to the rectangular boxes representing the participating entity types. The **relationship name is displayed** in the **diamond-shaped box**

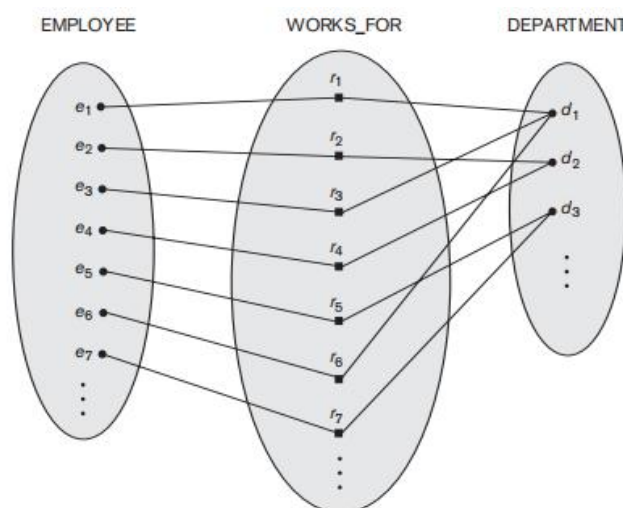


Fig (bb) : Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

Relationship Degree, Role Names, and Recursive Relationships

Degree of a Relationship Type: The degree of a relationship type is the number of participating entity types.

Ex : the WORKS_FOR relationship is of degree two.(refer fig bb). A relationship type of degree two is called binary, and one of degree three is called ternary.

An example of a ternary relationship is SUPPLY, shown in below Figure(cc), where each relationship instance r_i associates three entities—a supplier s , a part p , and a project j —whenever s supplies part p to project j .

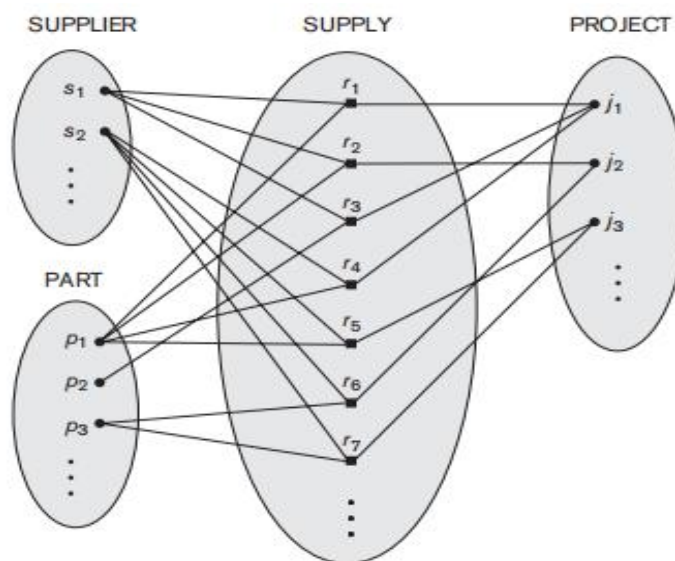


Fig (cc) : Some relationship instances in the SUPPLY ternary relationship set.

Constraints on Binary Relationship Types

Two main types of binary relationship constraints: **cardinality ratio and 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, and M:N.**

- a) **1:1 (One to One) cardinality ratio** : If an entity [a record] of one entity set is associated with maximum of one entity of the other entity set, then the relationship type is said to be one-to-one.

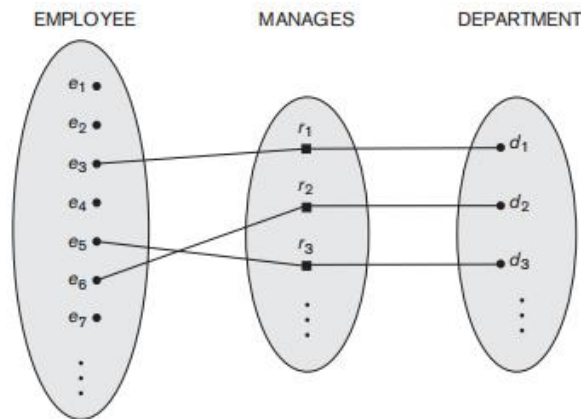


Fig : 1:1 Relationship,Manages

- b) **1:N (One to Many)** : If an entity of one entity set is associated with zero or more entities of the other entity set, then the cardinality ratio is said to be one-to-many from one side entity set to the many side entity set.

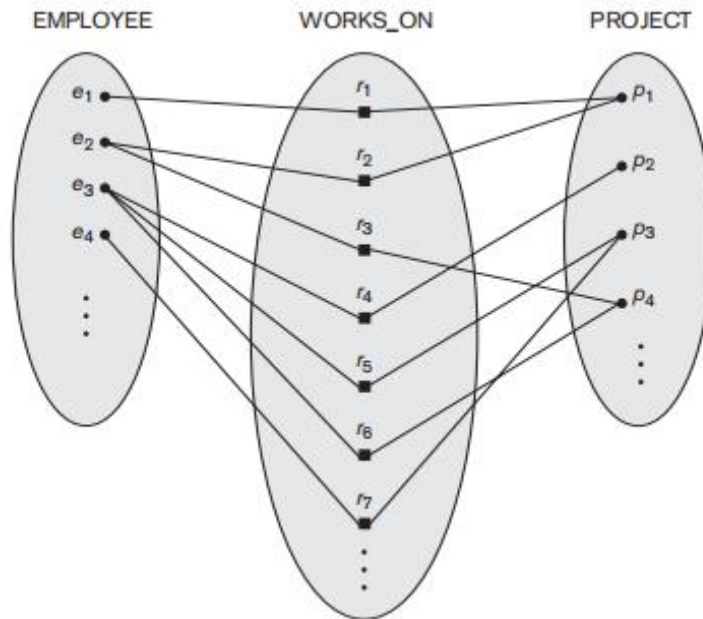


- c) **N:1 (Many to One)** : An entity in set A can be associated with at most one entity in set B and an entity in set B can be associated with any number (zero or more) of entities in set A.



- d) **M:N (Many to Many)** : An entity in set A can be associated with any number (zero or more) of entities in set B and an entity in set B can be associated with any number (zero or more) of entities in set A.





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—

- a. **Total And**
- b. **Partial**

- a. **Total participation** : 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**.
- b. **Partial Participaion** :, meaning that *some* or *part of the set* of employee entities are related to some entity
The cardinality ratio and participation constraints together, is called **structural constraints** of a relationship type.

In ER diagrams, total participation (or existence dependency) is displayed as a double line connecting the participating entity type to the relationship, whereas partial participation is represented by a single line.

Weak Entity Types












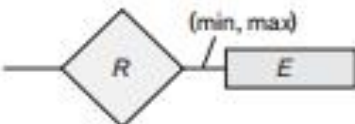
Weak entity types : Entity types that do not have key attributes of their own.

Regular entity types : Entity types that do have a key attribute.

A weak entity type always has a *total participation constraint* (existence dependency) with respect to its identifying relationship because a weak entity can not be identified without an owner entity.

Partial Key: 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*.

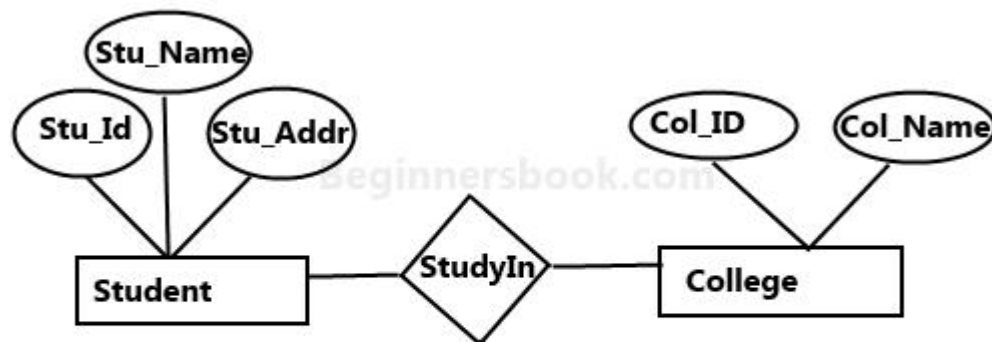
Notations for ER Diagram :

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

ER (Entity Relationship) Diagram

An **Entity–relationship model (ER model)** describes the structure of a database with the help of a diagram, which is known as **Entity Relationship Diagram (ER Diagram)**. An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.

Example : A simple ER Diagram



Sample E-R Diagram

In the above diagram we have two entities Student and College and their relationship. The relationship between Student and College is many to one as a college can have many students however a student cannot study in multiple colleges at the same time. Student entity has attributes such as Stu_Id, Stu_Name & Stu_Addr and College entity has attributes such as Col_ID & Col_Name.

Notations and their meaning in an E-R Diagram:

Rectangle: Represents Entity sets.

Ellipses: Attributes

Diamonds: Relationship Set

Lines: They link attributes to Entity Sets and Entity sets to Relationship Set

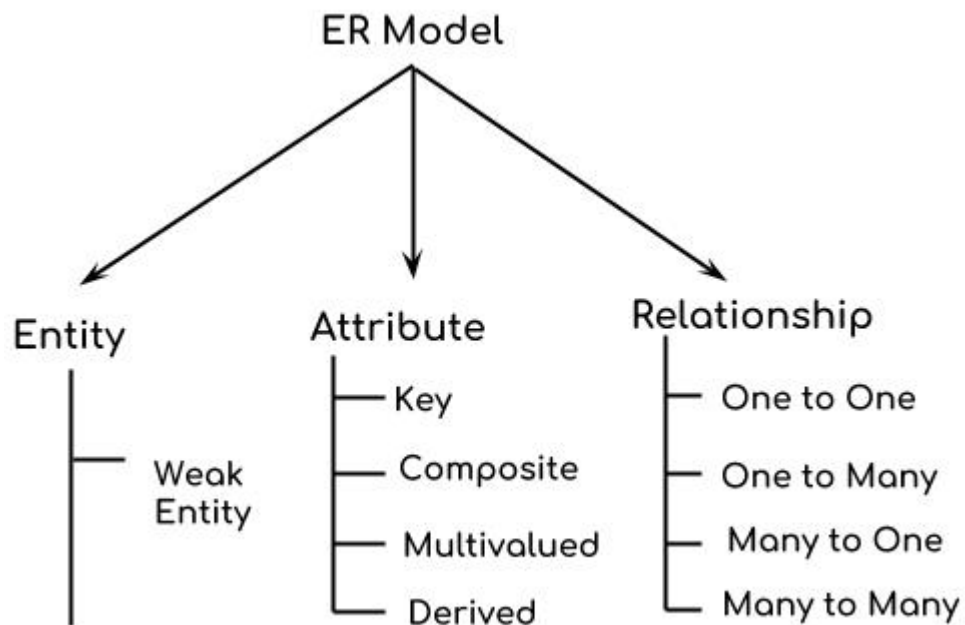
Double Ellipses: Multivalued Attributes

Dashed Ellipses: Derived Attributes

Double Rectangles: Weak Entity Sets

Double Lines: Total participation of an entity in a relationship set.

Components of ER Diagram



Example : ER diagram for Company Database

