INTRODUCTION TO DATABASE

S. Mamatha Jajur

INTRODUCTION

Database and database systems are essential component of everyday life in modern society. Daily, most of us encounter several activities that involve some interaction with a database. For example, if we go to bank to deposite or withdraw funds, if we make a hotel or airline reservation to access computerized library catalog to search book, or if we make online purchase etc.

Database and database technology have major impact on the growing use of computers. Databases play a critial role in almost all areas where computers are used, including business, electronic commerce, engineering, medicine, law, education, and library science etc.

A **database** is a collection of related data. By data, we mean known facts that can be recorded and that have implicit meaning. A database has the following implicit properties:

- A database represents some aspect of the real world, sometimes called the miniworld or universe of discourse 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

- A database has some source from which data is derived some degree of interaction with events in real world, and an audience that is actively interested in it.
- The end users of database may perform bainess transactions or events may happen that cause the change in database. Thus database must be true reflection of miniworld that it represents.
- A database can be of any size and complexity. A small/simple database like list of names & addresses or a huge/complex database like Amazon's database.
- A database may be generated and maintained manually or it may be computerized.

DATABASE MANAGEMENT SYSTEM (DBMS)

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 following processes:

- **Defining** a database involves specifying the data types, structures, and constraints of the data to be stored in the database
- **Constructing** the database is the process of storing the data on some storage medium that is controlled by DBMS.
- **Manipulating** includes querying, updating and repost generation
- **sharing** abbacss multiple users and programs to access database simultaneously.

DBMS also provides other important functions : **System protection** against hardware or software malfunction (or crashes) ; **Security protection** against unauthorized or malicious access and **Maintaining** the database system by allowing the system to evolve as requirements change over time.

Most DBMS's are very complex software systems. The database and DBMS software together form a **database system**.

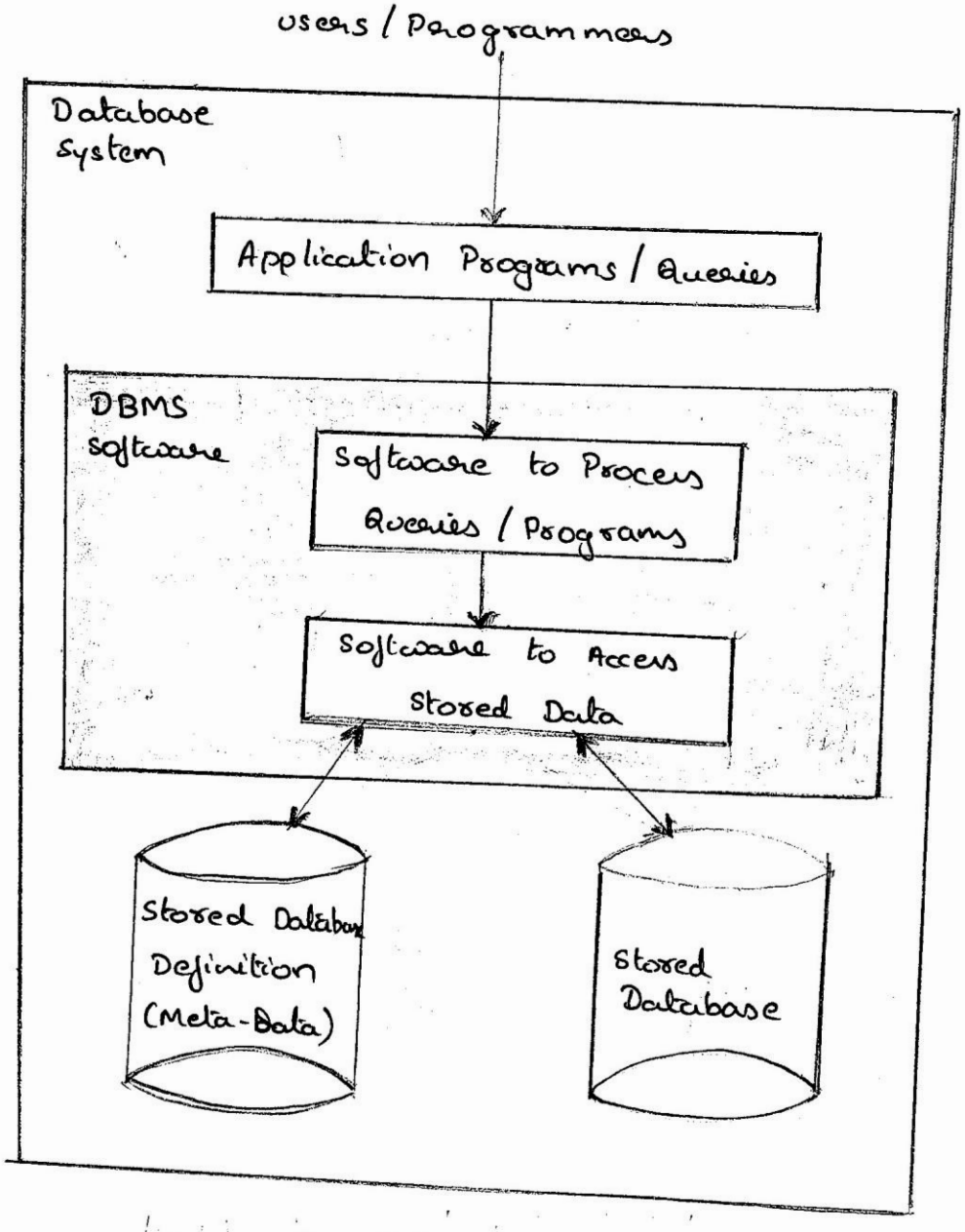


Fig : A Simplified Database System Environment

AN EXAMPLE

Consider UNIVERSITY database example for maintaing information concerning students, coourses, and grades in University entironment. Fig shows database structure & sample data stored.

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, COURSE file stores on each course and so on.

To define this database, we must specify the structure of the records of each file by specifying the different types of data elements (attributes) to be stored in each record. For exc: STUDENT file includes student's nam student_number and so on.

We must also specify a data type for each data element within a record. Ex: student name is string of alphabetic characters.

To construct the UNIVERSITY database, we store data to represent each student, coourse, section, grade report & prerequisite as record in the appropriate file. Notice that records in the various files may be related and may have many relationship among the records.

Database manipulation involves quering & updating. Examples of queries are:

- Retrieve the trawscripts of student named 'smith'.
- Lis all prequisites of the 'Database' course.

Examples of update are :

- change the class of 'smith' to sophomore
- Enter a grade of 'A' for 'smith' in 'Database' section of last semester.

STUDENT

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

COURSE

Course_name	Coursenumber	Credit_hour	Department
Computer_Science	CS1310	4	CS
Data Structure	CS3320	4	CS
Database	CS3380	3	MATH
Discrete Maths	MATH2410	3	CS

SECTION

Section_ID	Coursenumber	Sem	Year	Instructor
85	MATH2410	Fall	04	King
92	CSB10	Fall	04	Anderson
112	CS3320	Spring	05	Knuth
119	MATH2410	Fall	05	Chang

GRADE_REPORT

studentnumber	Section_ID	Grade
17	112	B
17	119	C
18	85	A
8	92	A

PREREQUISITE

Coursenumber	Prerequisitie_no
CS 3380	CS3320
CS3380	MATH2410

CHARACTERISTICS OF THE DATABASE APPROACH

A number of characteristics distinguish the database approach from the traditional approach of programming with files. In traditional file processing, each user defines and implements the files needed for a specific software application as part of programming the application. For example, one user, the grade reporting office, may keep a file on students & their grades. A second user, the accounting officer, may keep track of students fees & their payments. Although both users are interested in data about students, each user maintains separate files – and programs to manipulate these files.

The result of this approach is, **data redundancy**, which not only wastes storage space but also makes it more difficult to keep changing data items consistent with one another, as a change to one copy of a data item must be made to all of them (duplication-of-effort). Results in **inconsistency** when one (or more) copies of a datum are changed but not others.

In the database approach, a single repository of data is maintained that is defined once and then accessed by various users. In file systems, each application is free to name data elements independently. In contrast in a database, names or labels of data are defined once, and used repeatedly by queries, transactions and applications.

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

1) SELF-DESCRIBING NATURE OF A DATABASE SYSTEM

A database system includes - in addition to data-base itself but also complete definition or description of the database structure and constraints. This definition is stored in 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 data. The information stored in the catalog is called **meta-data**.

The catalog is used by DBMS software and also by database users who need information about database structure. A general purpose DBMS software package is not written for a specific database application.

In traditional file-processing, data definition is typically part of application program themselves. Hence, these programs are constrained to work with only one specific database, whose structure is declared in application programs.

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_no	XXXNNN	PREREQUISITE

Fig : An example database catalog

INSULATING BETWEEN PROGRAMS AND DATA, & DATA ABSTRACTION

Program-data-Independence: In traditional file processing the structure of data files is embedded in the applicatio programs, so many changes to the structure of a file may require changing all programs that access that file In contrast, DBMS access programs do not require such changes in most cases as the structure of data files is stored in the DBMS catalog separately from access programs. This property is called **Program data Independence**

Ex: In file system if need to add new field, say data of birth, the program will not work and must be changed. But in DBMS, we need to change the description of STUDENT in catalog to reflect the changes, no programs are changed.

User can define operations on data as a part of database definition. The interface of an operation includes operation name & data types of arguments. The implementation of the operation is specified separately & can be changed without affecting the interface. Users application programs can operate on data by invoking the operations through interface names & arguments, regardless of how the operations are implemented. This property is called **Program-operation Independence**.

The characteristic that allow program-data independence and program-operation independence is called **Data Abstraction**. DBMS provides user with conceptual representation of data using **Data Model**, a type of data abstraction.

SUPPORT OF MULTIPLE VIEWS OF THE DATA

A database typically has 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 is not explicitly stored.

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

Ex : One user of STUDENT database, may be interested in printing details of each student and second user, may be interested in only students enrolled in particular course

SHARING OF DATA AND MULTIUSER TRANSACTION PROCESSING

A multiuser DBMS, must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and maintained in a single database. The DBMS must include **concurrency control** software to ensure that several users trying to update the same data do so in a controlled manner so that result of the updates is correct.

A fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly and efficiently.

A **transaction** is an executing program or process that includes one or more database access such as reading, updating of database records. Each transaction is supposed to execute a logically correct

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 transaction, even though hundreds of transactions may be executin concurrently.
* The **atomicity** property ensures that either all the database operations in a transaction are executed or none are.

ADVANTAGES OF USING THE DATABASE APPROACH

Database approach came into existence due to the bottlenecks of file processing system. In the database approach, data is stored at a central location and is shared among multiple users. Thus, the main advantage of DBMS is **centralized data management**. The centralized nature of database system provides several advantages :

CONTROLLED DATA REDUNDANCY

During database design, various views of different users are integrated and each logical data item is stored at central location. This eliminates replicating the data item in different files and ensures consistency and saves the storage space. However, in practice, it is sometimes necessary to use controlled redundancy to improve the performance of queries. In such cases, DBMS should be capable of controlling this redundancy in order to avoid data inconsistencies.

RESTRICTING UNAUTHORIZED ACCESS

A DBMS should provide a security and authorization subsystem, which DBA uses to create accounts and to specify account restrictions. The restriction like retrieve data or updata data. Then, the DBMS should enforce these restrictions automatically

PROVIDING PERSISTENT STORAGE FOR PROGRAM OBJECTS

Databases can be used to provide peersistent storage for program objects and data structures. The object-oriented database systems are compatible with programming languages such as c++ and Java and the DBMS software automatically peyforms any necessary conversions of complex structures into a dorsnat suitable for file storage and also from file format to program variable structure.

PROVIDING STORAGE STRUCTURES AND SEARCH TECHNIQUES FOR EFFICIENT QUERY PROCESSING.

Database systems must provide capabilities for efficiently executing queries and updates. The DBMS, provides **indexes**, specialized data structures to speed up disk search for the desired records. Indexes are typically based on tree data structures or hash data structures, suitably modified for disk search.

DBMS provides **buffering** module that maintains parts of database in main memory buffers.

DBMS provides **query processing and optimization** module, responsible for choosing efficient query execution plan for each query based on existing storage structure.

PROVIDING BACKUP AND RECOVERY

DBMS must provide facilities for recovering from hardware and software failures. The **backup and recovery subsystem** of the DBMS is responsible for recovery. In case of system failure, recovery subsystem is responsible for making sure that database is restored to the state it was in before the transaction started executing. Or ensure that transaction is resumed from point at which it was interrupted.

PROVIDING MULTIPLE USER INTERFACES

DBMS facilitates many types of users with varying levels of technical knowledge used a database, a DBMS should provide a variety of user interfaces. These include query languages for casual users, programming language interfaces for application programmers, forms & commands codes for parameteric users, & menu-deriven interface and natural language interfaces for standalone users. And also provides graphical user interface (GUI).

ENFORCING INTEGRITY CONSTRAINTS

Most Database applications have certain integrity constraints that must hold for the data. The DBMS should provide capabilities for defining and enforcing these constraints. For ex: Specifying a data type for each data item.

REPRESENTING COMPLEX RELATIONSHIPS AMONG DATA

A DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, & to retrieve & update

data easily and efficiently.

PERMITTING INFERENCING AND ACTIONS USING RULES

Some database systems provide capabilities for defining deduction rules for inferencing new information from the stored database facts. such systems are called deductive database system.

BRIEF HISTORY OF DATABASE APPLICATIONS

- Early Database Applications :
  - Hierarchical and Network Models were introduced in mid 1960s and dominated during the seventies.
  - A bulk of worldwide database processing still occurs using these models.
- Relational Model based systems :
  - Relational model was originally introduced in 1970, was heavily researched and experimented with IBM Research and several universities.
  - object-oriented and emerging applications
- Object-Oriented DBMS were introduced in late 1980s and early 1990s to cater to the need of complex data processing in CAD & other applications.
- Many relational DBMSs have incorporated object database, concepts, leading to new category called as object-relational DBMSs (ORDBMS) in 1980s.
- Extended relational systems add further capabilities (ex for multimedia data, XML and other data types).
- Data on the Web and E-commerce applications :

Web contains data in HTML with links among pages. This has given raise to a new set of applications and E-commerce using new standard like XML.

- Scripting programming languages such as PHP and JavaScript allow generation of dynamic Web pages that are partially generated from a database.

WHEN NOT TO USE A DBMS

The overhead costs of using DBMS are due to:

- High initial investment and possible need for additional hardware and software and training.
- The generality that DBMS provides for defining and processing data
- Overhead for providing security, concurrency control, recovery and integrity functions.
- When a DBMS may be unnecessary
  - If the database applications are simple, well defined and are not expected to change.
  - If there are stringent real time requirements that may be met because of DBMS overhead.
  - If access to data by multiple users is not required

DISADVANTAGES OF IN FILE PROCESSING SYSTEM

- Data redundancy and inconsistency.
- Difficulty in accessing data
- Data isolation
- Integrity problems
- Atomicity problems
- security problems
- Concurrent access is not possible

APPLICATION AREAS OF DATABASE SYSTEM

Database systems are widely used in different areas.

- Airlines and railways : Used for online reservations & displaying schedule information.
- Banking : Banks use database for customer inquiry, accounts loans and other transactions.
- Education : Schools & colleges use database for registration, results and other information.
- Telecommunications : Use database to store information about network, telephone numbers, record of calls & so on
- E-commerce : Database are used for keeping track of purchase order details, payment details etc.
- Health care : Database are used for maintaing patient health care details, treatments availabk etc
- Finance : Database are used store information like sales, purchases of stocks, bonds & other trading details
- Human Resources : Organizations use database for storing information about their employees, salaries, taxes etc
- Credit card transactions : Database are used for keeping track of purchases, generation of statements etc.

ACTORS ON THE SCENE

In large organizations, many people are involved in the design, use and maintenance of a large database with hundreds of users. We shall see different people who use the database.

* DATABASE ADMINISTRATORS

In a database environment, the **database administrator** is responsible for administering the primary resource i.e database and the secondary resource DBMS and other related software.

- The DBA is responsible for authorizing access to database coordinating and monitoring its use and acquiring software & hardware resources as needed.
- The DBA is accountable for problems such as security breaches and poor system response time
- In large organizations, DBA is assisted by a staff that carries out these functions.

* DATABASE DESIGNERS

Database designers are responsible for identifying the data to be stored in the database & for choosing appropriate structures to represent and store this data.

- These tasks are mostly undertaken before database is actually implemented & populated with data
- It is the responsiblity of database designers to communicate with all prospective users in order to understand their

requirements and to create all prospective database a design that meets these requirements.

* End Users

End users are the people whose jobs require access to database for querying, updating & generating reports. The database primarily exists for their use. There are several categories of end users:

- CASUAL END USERS occasionally access the database, but 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 make up a sizable portion of database end users. They constantly query and update database, using standard types of queries and updates called canned transactions that have been carefully programmed and tested. Ex: Bank tellers check am account balances & post withdrawals & deposits.
- SOPHISTICATED END USERS include engineers, scientists, business analysts and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their applications to meet their complex requirements.
- STANDALONE USERS maintain personal databases by using ready-made program package that provide easy-to-use menu-based or graphics-base interfaces.

* SYSTEM ANALYSTS AND APPLICATION PROGRAMMERS

System Analysts determine the requirements of end users, especially naive & parametric end users, & develop specifications for canned transactions that meet these requirements.

Application programmers implement these specifications as programs; then they test, debug, document & maintain these canned transaction.

* DBMS SYSTEM DESIGNERS AND IMPLEMENTERS design and implement the DBMS modules and interfaces as a software packages.

* TOOL DEVELOPERS design and implement tools - the software packages that facilitate database modeling & design, database system design and improved performance.

* OPERATORS AND MAINTENANCE PERSONNEL are responsible for the actual running and maintenance of hardware & software environment for the database system.

OVERVIEW OF DATABASE LANGUAGES AND ARCHITECTURE

The architecture of DBMS packages has evolved from the early monolithic systems, where the whole DBMS that are a software package was one tightly integrated systems, to the modern DBMS packages that are modular in design, with a client/server system architecture.

The evolution in computing trends, where large centralized mainframe computers are being replaced by hundreds of distributed workstations and personal computers connected via communication networks to various types of server machines - Web servers, database servers, file servers, application servers and so on.

DATA MODELS, SCHEMAS AND INSTANCES

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.

One of the main characteristics of database approach is to support data abstraction so that different users can perceive data at their preferred level

DATA MODEL : A data model is a collection of concepts that can be used to describe the structure of a database. - pro By structure of a database we mean the data types, relationships and constraints that apply to the data. Data model provides a means to achieve data abstraction.

Categories of Data Models.

The data models can be categorized according to the types of concepts they use to describe the database structure.

- High - level or Conceptual data models provides concepts that are close to the way many users perceive data. It uses concepts such as entities, attributes and relationships. An entity represents a real world object or concept, such as employee or a project. An attribute represents some property of interest that further describes an entity, such as employee 's name or salary. A relationship among two or more entities represent an association among the entities for ex. works-on relationship b/w employee & project.

  Ex : Entity - Relationship Model.

- Low-level or physical data models provide concepts that describe the details of how data is stored on the computer storage media, magnetic disks. Concepts provided by low-level data models are generally meant for computer specialists, not for end users.

  Physical data models describe how data is stored as files in the computer by representing information such as record formats, record orderings and access paths. Access path is a structure that makes search for particular database records efficient. An Index is an example of an access path that allows direct access to data using an index term or keyword.

- Representional or Implementation data models which provides concepts that may be easily understood by end users. Representation data model hide many details of data storage on disk but can be implemented on a computer system directly. They are most frequently used in traditional commerical DBMSs. It uses record structures & hence are also called record-based data models.

  Ex:

  - Relational data model, network data model, hierarchical data model, object data model.

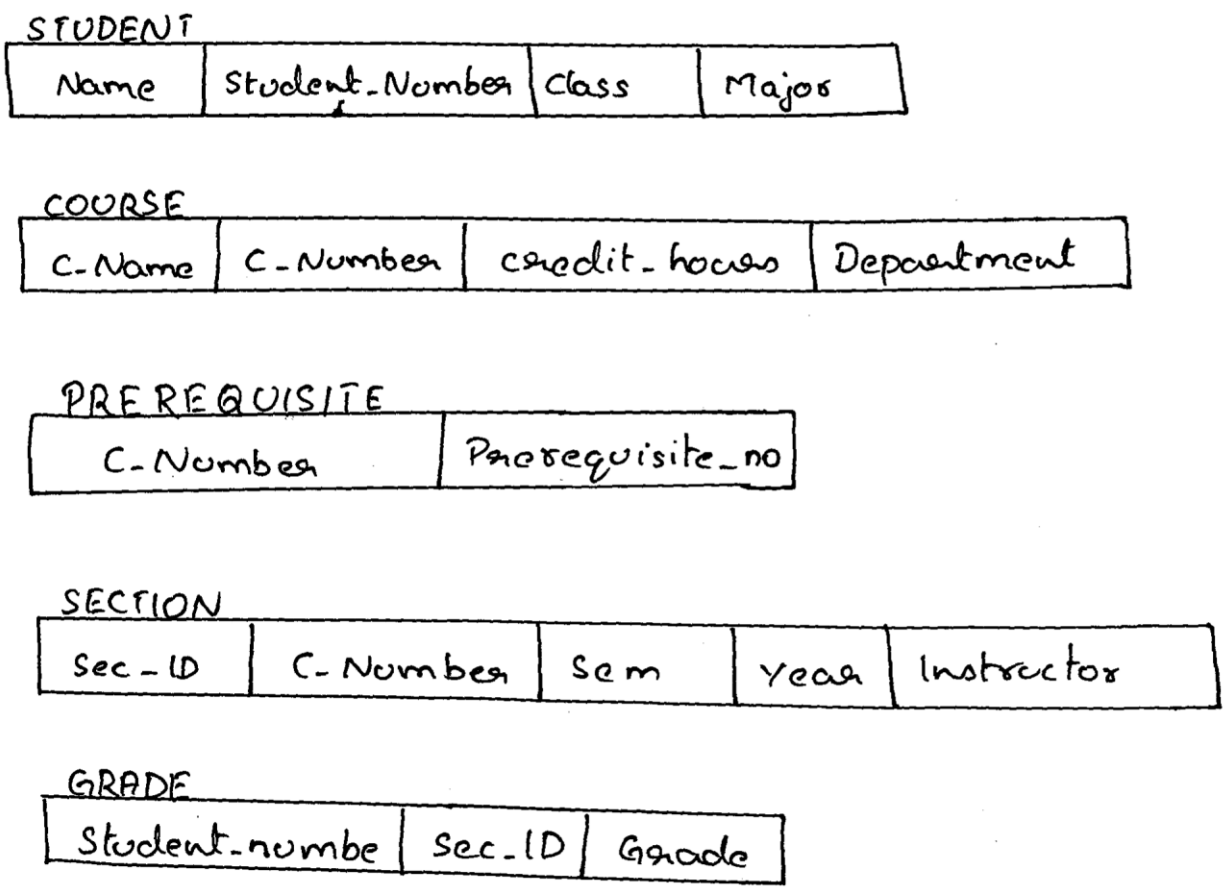
Schemas : The description of a database is called the database schema, which is specified during database design and is not expected to change frequently.

A displayed schema is called a **schema diagram**.

A schema diagram displays the structure of each record type but not actual instances of records.

Each schema object is called a **schema construct**

A schema diagram displays only some aspects of a schema, such as names of record types and data items, & some types of constraints.



SCHEMA DIAGRAM FOR THE DATABASE

Instances : The data in the database at a particular moment in time is called a database state or snapshot. It is also called the current set of occurences or **Instances** in the database.
Ex : the student construct will contain the set of individual student entities (records) as its instances.

When we define a new database, we specify sto its database schema only to the DBMS. At this point, the corresponding database state is the **empty state** with no data. we get the initial state of the database when the database is first populated or loaded with the initial data. From then on, every time an update operation is applied to the database, we get another database state. An any point in time, the database has a current state.

The DBMS is party responsible for ensuring that every state of the database is a valid state - i.e, a state that satisfies the structure & constraints specified in the schema.

The DBMS stores the description of schema constructs and constraints also called **meta-data** in the data DBMS catalog. so that DBMS software can refer to the schema whenever it needs to.
The schema is also called **Intension** and a database state is called an **extension** of the schema

THREE - SCHEMA ARCHITECTURE

The goal of the three-schema architecture is to separate the user applications and the physical database. In this architecture, schemas can be defined at the following three levels :

Internal level : 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.

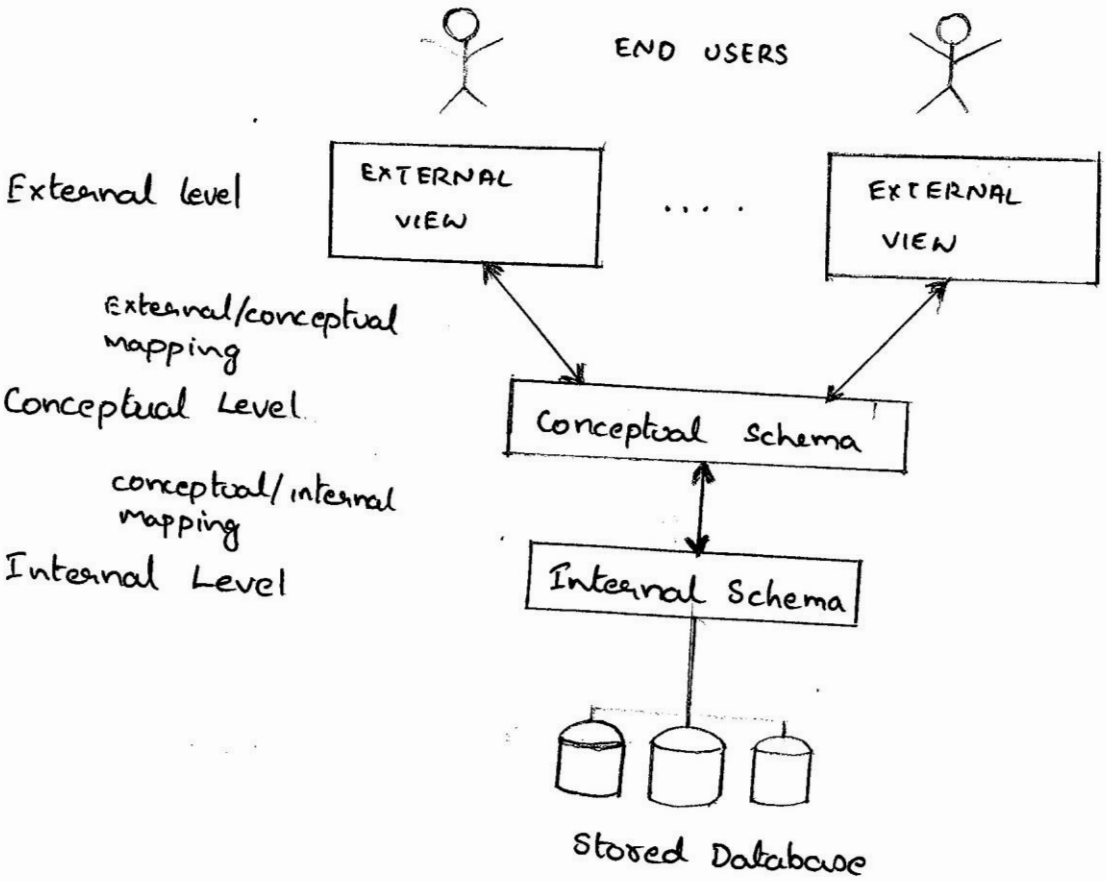


Fig : THREE SCHEMA ARCHITECTURE

Conceptual level : The conceptual level has an conceptual schema, which describe the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures & concentrates on describing entities, data types, relationships, user operations and constraints. Usually, a representational data model is used to describe the conceptual schema when a databa is implemented.

External or View level : The external or view level has 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. Each external schema is typically implemented using a representational data model, possibly based on external schema design in a high-leve data model.

The three-schema architecture is a convenient tool with which the user can visualize the schema levels in database system. Most DBMSs do not separate the three levels completely and explicitly, but support the three-schema architecture to some extent.

In DBMS based on the three-schema architecture each user group refers only to its external schema. Hence, the DBMS must transform a request specified on an external schema into a request against 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 process of transforming requests and results between levels are called **mappings** these mappings may be time-consuming.

DATA INDEPENDENCE

Data Independence is the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. There are two types of data Independence :

- **LOGICAL 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 record type or data item), to change constrain or to reduce the database (by removing a record type or data item). In the last case, external schemas that refer only to the remaining data should not be affected. Only view definition & mappings need to changed for this.

  After the conceptual schema undergoes a logical reorganization, application program that references the external schema constructs must work as before.

• PHYSICAL DATA INDEPENDENCE

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 internal schema may be needed because some physical files were reorganized, ex by creating additional access structures, to improve the performance of retrieval or update. Generally, physical data independence exists in most databases and file environments in which storage details are hidden from the user. Applications remain unaware of these details.

The three-schema architecture can make it easier to achieve true data independence, both physical and logical.

DATABASE LANGUAGES

• DATA DEFINITION LANGUAGE (DDL)

Data definition language (DDL) is used by DBA and by database designers to define conceptual and internal schemas for the database and any mappings between the two. The DBMS will have a DDL compiler whose function is to process DDL statements in order to identify descriptions of the schema constructs and to store the schema description in the DBMS catalog.

- ## STORAGE DEFENITION LANGUAGE (SDL)

Storage definition language is used to specify the internal schema. The mappings btw two schemas may be specified in either one of these languages. In most relational DBMSs today, there is no specific language that performs the role of SDL. Instead, internal schema is specified by a combination of parameters & specification related to storage - the DBA staff typically controls indexing and mapping of data storage.

- ## VIEW DEFINITION LANGUAGE (VDL)

View definition language is used to specify user views and their mappings to the conceptual schema, but in most DBMSs the DDL is used to define both conceptual & external schemas. In relational DBMSs, SQL is used in the role of VDL to define user or application views as result of predefined queries.

- ## DATA MANIPULATION LANGUAGE (DML)

Data manipulation language (DML) is used to manipulate the database. Typical manipulations include retrieval, insertion, deletion and modification of the data. DML provides set of operations or a language 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 to be embedded in a general purpose programming language. In later case, DML statements must be identified within the program so that they can be extracted by a precompiler & processed by DBMS. It is also called as set-at-a-time DML.

- A **low-level** or **procedural DML** must be embedded in a general purpose programming language. This typically retrieves individual records or objects from database and processes each separately. Thus it is also called as record-at-a-time DML

DATA INTERFACES

User friendly interface provided by DBMS are

- Menu-Based Interface for Web Clients or Browsing

These interfaces provide user with lists of options (menus) that lead user through formation of a request.

- FORM Based Interfaces

These interface displays a form to each user. Users can fill out all of the forms entries to insert new data. Usually designed for Naive users.

- Graphical User Interfaces

A GUI interface displays a schema to user in diagrammatic form. The user can specify a query by manipulating the diagram.

- Natural language Interfaces.

These interfaces accepts requests written in English or some other language & attempt to understand them. Usually has its own schema as well as dictionary of important words.

- 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.

- Interfaces for parametric Users

These have a small set of operations that they must perform repeatedly. Systems analysts & programmers design & implement a special interface for each known class of naive users, having small set of abbreviatid commands

- Interfaces for the DBA.

Most database systems contain privileged commands that can be used only by 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.

DATABASE SYSTEM ENVIRONMENT

Database System is a complex software system. The figure illustrates the typical DBMS components. The figure is divided into two parts. The top part of the figure refers to various users of database environment and their interfaces. The lower part shows the internals of DBMS responsible for storage of data and processing of transactions.

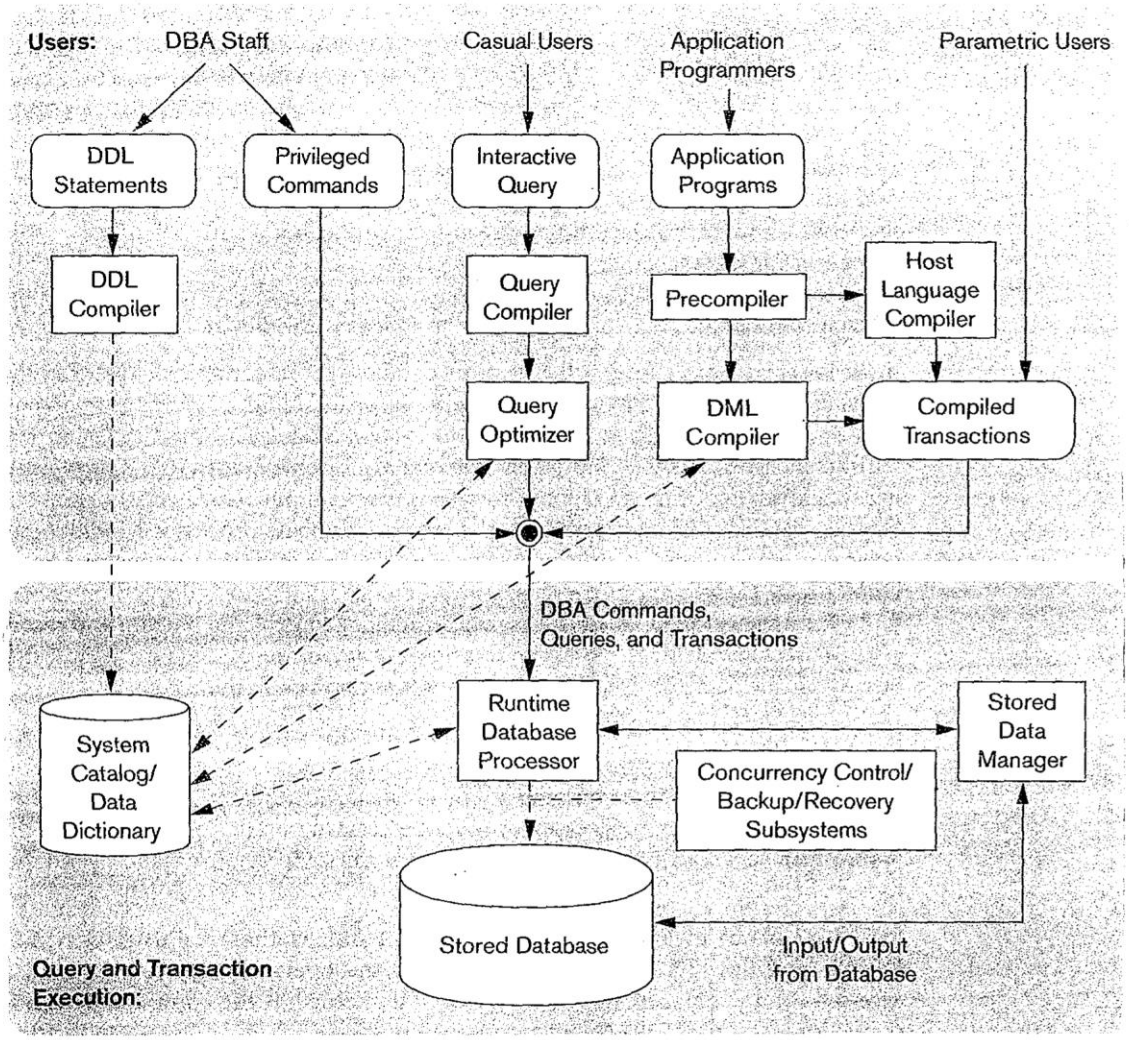


Fig : COMPONENTS MODULES OF A DBMS AND THEIR INTERACTIONS

Users top part

→ **DBA staff** works on defining the database and tuning it by making changes to its definition using DDL and other privileged commands.

The DDL compiler processes schema definitions, specified in DDL, and stores descriptions of schemas in DBMS catalog.

→ The **system catalog** include information such as names and sizes of files, names & data types of data items, storage details of each file, mapping information among schemas, & constraints and many other informations that are need by DBMS modules.

→ **Casual users** and persons with occasional need for information from the database interact using interactive query interface. with interactive queries. These queries are parsed & validated by a **query compiler** that compiles them into an internal form. This internal query is subjected to **query** optimization using **query optimizer** which is concerned with rearrangement & posible reording of operations, elimination of redundancies & use of correct algorithms & indexes during execution. It generates executable code that performs necessary operations for query and makes calls on the runtime processor.

8 → **Application programmers** write programs in host languages such as Java, C or C++ that are submitted to a precompiler. The precompiler extracts DML commands from an application program. These commands are sent to the DML compiler for compilation into object code for database access. The rest of the program is sent to the host language compiler. The object codes for DML commands and rest of the program are linked, forming **canned** transaction whose executable code include calls to runtime database processor.

→ Canned transactions are executed repeadly by **parametric** users, who simply supply the parameters to the transactions.

→ The **runtime database processor** executes - the privileged commands, the executable query plans and canned transactions with runtime parameters. It works with system catalog and storage data manager. It handles other aspects like data transfer, management of buffers in main memory.

→ The **database** and the **DBMS catalog** are usually stored on disk. Access to the disk controlled primarily by the operating system, which schedules read/write. DBMS have their own buffer management module to schedule read/write

→ A higher-level **stored data manager** module of DBMS controls access to DBMS information that is stored on disk whether it is part of database or catalog. It uses

→ basic operating system services for carrying out low-level input/output operations b/w disk and main memory.

9 → Concurrency control and backup and recovery systems are integrated into the working of the runtime database processor for purpose of transaction management.

Database System Utilities

DBMS have database utilities that help the DBA manage the database system.

- **Loading**. A loading utility is used to load existing data files into database. Usually, the current (source) format of data file and desired (target) database file structure are specified to the utility, which then automatically reformats data & stores it in database. Transferring data from one DBMS to another is becoming common. Such tools are called conversion tools.
  Ex: IMS (hierarchical DBMS), IDMS (network DBMS)
- Backup. A backup utility creates a backup copy of database, usually dumping entire database onto tape or other mass storage medium. Incremental backups are also often used, where only changes since previous backup are recorded. Internal back up are complex, but saves storage space.
- Database storage reorganization. This utility can be used to reorganize set of databases files into different file organizations & create new access paths to improve performance.

- Performance monitoring. such a utility monitors database usage and pr. provides statistics to DBA The DBA uses statistics in making decesions such as wether or not be reorganizeat files or whether to add or drop indexes to improve performance.

UNIT 3

CONCEPTUAL DATA MODELING USING ENTITIES AND RELATIONSHIPS

Conceptual modeling is a very important phase in designing a successful database application Generally, the team database application refers to a particular database and the associated programs that implement the database queries and updates. For example, a BANK database application that keeps track of customer accounts would include programs that implement database updates corresponding to customer deposits & withdraw

The Entity - Relationship model (ER model), is a popular high-level conceptual data model which is used frequently for conceptual design of database applications. and many database design tools employ its concepts.

USING HIGH-LEVEL CONCEPTUAL DATA MODELS FOR DATABASE DESIGN

The simplified Database Design process is as shown in figure. The first step is **requirements collection** and **analysis**, during which database designers interview prospective database users to understand and document their data requirements. These requirements should be specified in as detailed and complete a form as possible.

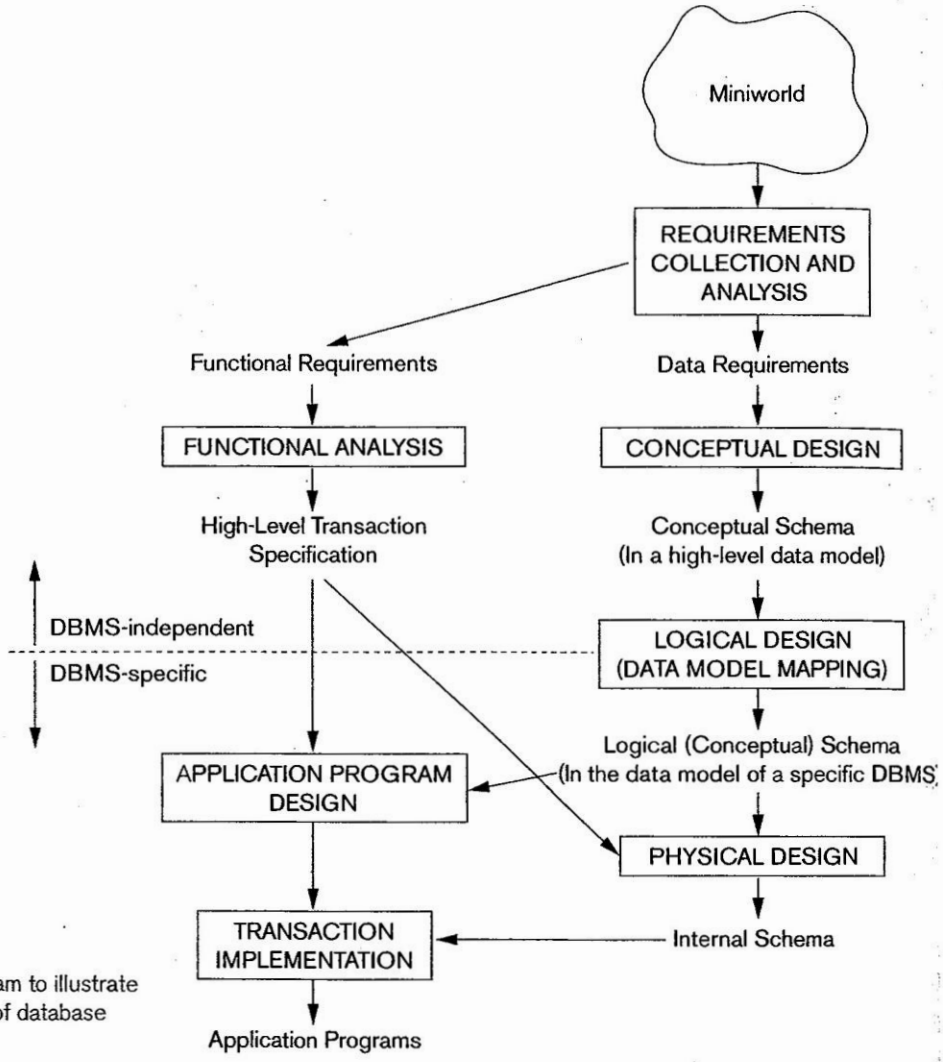


Figure 7.1
A simplified diagram to illustrate the main phases of database design.

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

After this next step is to create conceptual schema for the database, using high-level conceptual data model. This step is called **conceptual design**. The conceptual schema is a concise description of data requirements of the user, and include detailed description of entity types, relationships & constraints. It does not include implementation details.

The high-level conceptual schema can also be used as a reference to ensure that all user's data requirements are met and that the requirements do not conflict.

During or after 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 conceptual schema meets all identified functional requirements.

The next step in database design is the actual implementation of database using commercial DBMS. For which we use implementation data model - such as relational or object-relational database model - so conceptual schema is transformed from high-level data model into **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 DBMS.

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

ENTITIES

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

ATTRIBUTES

Attributes are the particular properties that describe entity. A particular entity will have a value for each of its attributes. The attribute values become major part of data stored in database.

Ex: An EMPLOYEE entity may be described by attributes like employee's name, age, address, salary & job.

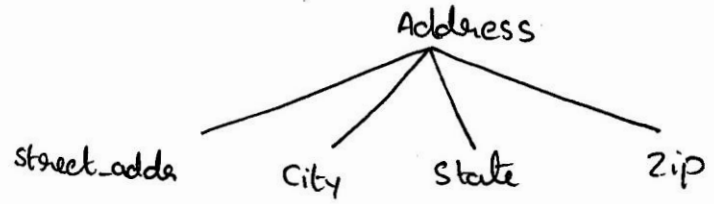
In ER diagrams, attributes names are enclosed in ovals & are attached to their entity type by straight line.

TYPES OF ATTRIBUTES

- COMPOSITE VERSUS SIMPLE (ATOMIC) ATTRIBUTES

Composite attributes can be divided into smaller subparts, which represents more basic attributes with independent meanings.

Ex: Address of the EMPLOYEE entity can be subdivided into Street_addr, City, State and Zip.



Simple attributes are not divisible into smaller subparts. For ex Age of the employee entity type.

- SINGLE-VALUED VERSUS MULTIVALUED ATTRIBUTES

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

In some cases an attributes can have a set of values for the same entity. Such attributes are called **Multivalued** attribute. In ER diagram, multivalued attributes are displayed in double ovals.
Ex: Colors attribute for a car, college_degrees attribute for a person.

- STORED VERSUS DERIVED ATTRIBUTES

In some cases, two (or more) attributes values are related - for example, the Age and Birth_date attribute of a person. For particular person entity, the value of Age can be determined from current date and the value of that person's Birth_date. The Age attribute is hence called a **derived attribute** and is said to be derived from the Birth_date attribute which is called **stored attribute**. In ER diagram, derived attributes are represented by dotted line.

- COMPLEX ATTRIBUTES

The composite and multivalued attributes can be nested arbitrarily. We can represent arbitrary nesting by grouping components of the composite attributes b/w parentheses () and separating components with commas, & by displaying

multivalued attributes b/w braces { }. Such attributes are called **complex** attributes.

Ex: If a person can have more than one residence and each residence can have a single address & multiple phones, an attribute Address_Phone for a person.

{Address_phone({Phone(Area_code, Phone_number)}, Address(street_address(Number, Street, Apartment_number), City, state, zip))}

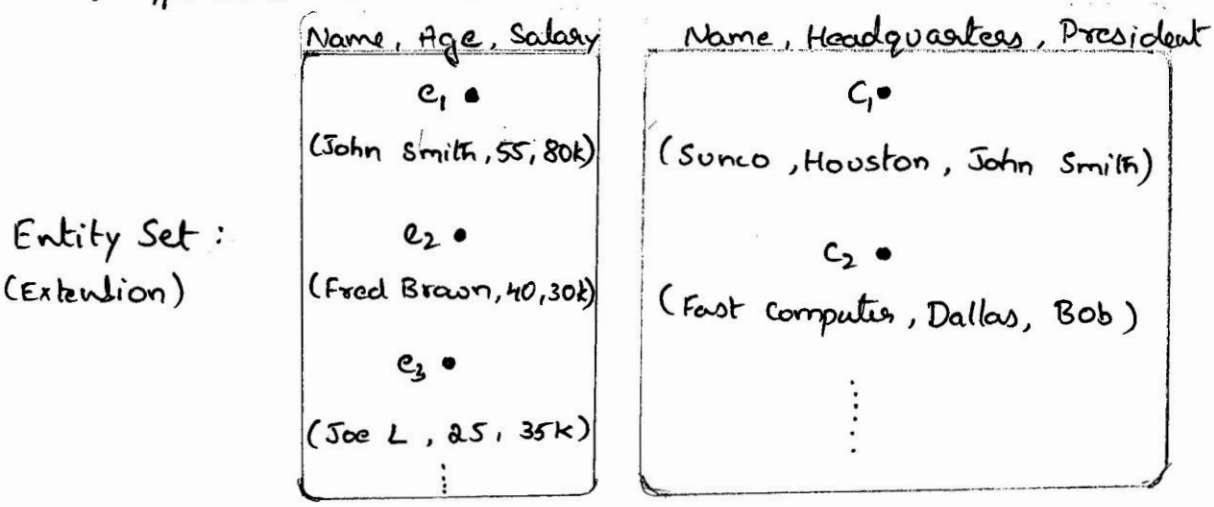
ENTITY TYPE AND ENTITY SET

An **Entity Type** defines a collection (or set) of entities that have the same attributes. Each entity type in the database is described by its name and attributes. In ER diagrams, an Entity Type is represented as a rectangular box enclosing the entity type name.

Ex: A company employing hundreds of employees may want to store similar information concerning each of employees. These employee entities share same attributes, but each entity has its own value(s) for each attribute. Thus EMPLOYEE and COMPANY are two entity types.

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 same name as entity type.

Ex: Employee refers to both a type of entity as well as current set of all employee entities in the database.



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

KEY ATTRIBUTES OF AN ENTITY TYPE

An entity type usually has an attribute 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. In ER diagrammatic notation, each key attribute has its name underlined inside the oval.

Ex : For the PERSON entity type, a typical key attribute is SSN.
For the STUDENT entity type, key attribute is USN.

Specifying that an attribute is a key of an entity type means that the preceding uniqueness property must hold for every entity set of the entity type. It is not the property of particular extension, rather, it is a constraint on all extensions of entity type.

Sometimes several attributes together form a key, meaning that the combination of attributes values must be distinct for each entity. If a set of attributes possesses this property, we define it as composite attribute and designate it as key attribute of entity type. Such a composite key must be minimal. i.e all components attributes must be included in the composite attribute to have the uniqueness property. Superfluous attributes must not be included in a key.

Ex : Registration attribute & vehicle_id are key attribute of CAR entity type. Registration is a composite key formed from state, area & number.

VALUE SETS (DOMAINS) OF ATTRIBUTES.

Each simple attribute of an entity type is associated with a value set (or domain of values), which specifies the set of values that may be assigned to that attribute for each individual entity.

Ex : 1) value set for Name attribute to be set of string of alphabetic characters separated by blank characters.

2) Age of EMPLOYEE to be set of integer numbers btw 16 and 70.

Value sets are typically specified using basic data types available in most programming languages, such as integer, string

VALUE SETS (DOMAINS) OF ATTRIBUTES

Each simple attribute of an entity type is associated with a value set (or domain of values), which specifies the set of values that may be assigned to that attribute for each individual entity.

Ex: Name attribute to the the set of strings of alphabetic characters separated by blank characters.

Value sets are not displayed in ER diagrams. Value sets are typically specified using the basic data types available such as integer, string, Boolean, float, enumerated type, date, time & other concepts are also employed. Mathematically, an attribute A of entity type E whose value set is V can be defined as a function from E to the power set P(V) of V.

$$A : E \rightarrow P(V)$$

We refer to the value of attribute A for entity e as A(e). This definition can have single valued, multivalued as well as NULL.

A NULL value is represented by empty set

For single-valued attributes, A(e) is restricted to singleton set for each entity e in E.

For composite ~~att~~ attribute A, value set V is power set of cartesian product of $P(V_1), P(V_2) \ldots P(V_n)$

$$V = P(P(V_1) \times P(V_2) \times \ldots \times P(V_n))$$

RELATIONSHIP TYPES

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 ex: The attribute Manager of Department refers to an employee who manages the department.

The attribute Controlling-department of Project refers to department that controls the project.

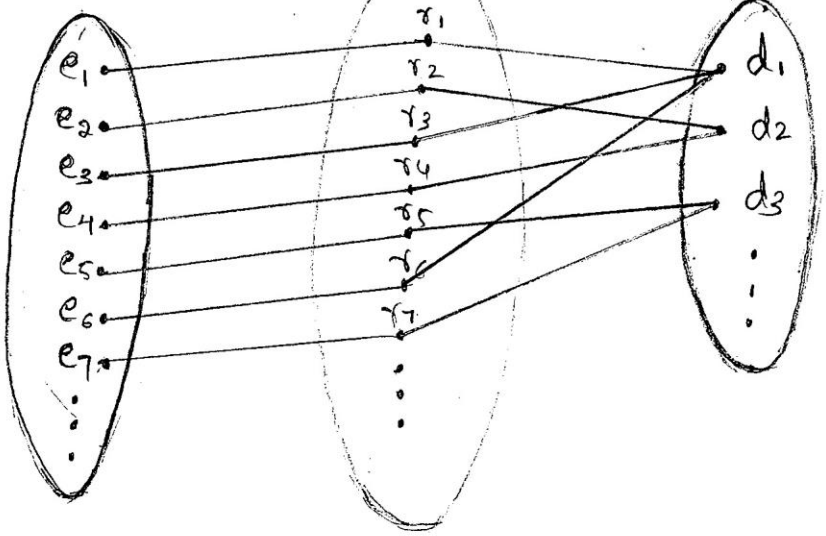
In ER model, these references should not be represented as attributes but as relationships.

A **relationship type** R among n entity types $E_1, E_2 \ldots E_n$ defines a set of associations - or a relationship set - among entities from these entity types.

Mathematically, the **relationship set** R is a set of relationship instances r_i, where each r_i associates n individual entities $(e_1, e_2, \ldots e_n)$ and each entity e_j in r_i is a member of entity type E_j, $1 \leq j \leq n$. Hence, relationship type is a mathematical relation on $E_1, E_2, \ldots E_n$. It can also be defined as a subset of Cartesian Product $E_1 \times E_2 \times \ldots \times E_n$. Each of the entity types $E_1, E_2 \ldots E_n$ is said to participate in the relationship type R. Similarly, each of the individual entities $e_1, e_2 \ldots e_n$ is said to participate in the relationship instance $r_i = (e_1, e_2 \ldots e_n)$

Each relationship instance r_i in R is an association of entities, where the association includes exactly one entity from each participating entity type.

For ex: Consider a relationship type WORKS_FOR b/w two entity types EMPLOYEE and DEPARTMENT, which associates each employee with the department for which employee works. Each relationship instance in the relationship set WORKS_FOR associates one EMPLOYEE entity & one DEPARTMENT entity. EMPLOYEE WORKS_FOR DEPARTMENT



In the miniworld represented by above fig, employees e_1, e_3 & e_6 work for department d_1; employees e_2 & e_4 work for department d_2; employees e_5 & 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 diagmond-shaped box.

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.

The relationship type of degree two is called **binary** and one of degree three is called **ternary**. Relationships can generally be of any degree.

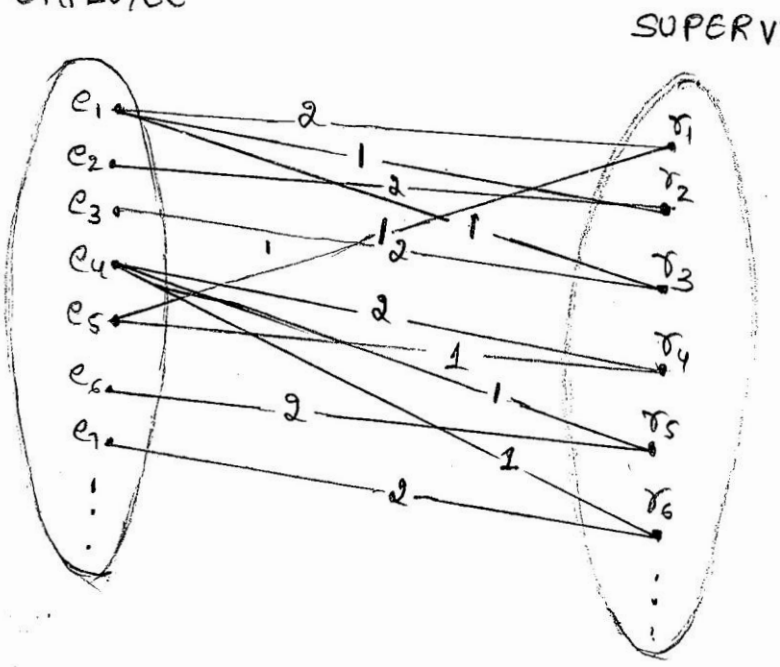
RELATIONSHIPS AS ATTRIBUTES

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.

Exc: 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 role name. However, in some case the same entity type participating more than once in a relationship type in different roles. In such cases the role name becomes essential for distinguishing meaning of each participation. Such relation-ship types are called **recursive relationships**.



Ex: The SUPERVISION relationship type relates an employee to a supervisor, where both employee & supervisor entities are members of same EMPLOYEE entity type. Hence EMPLOYEE entity type participates twice in SUPERVISION. once in role of **supervisor** (1)(boss) & once in role of **supervisee** (2) (subordinate).

CONSTRAINTS ON RELATIONSHIP TYPES

Relationship types usually have certain constraints that limit the possible combination of entities that may participate in the corresponding relationship set. There are two main types of relationship constraints :

- cardinality ratio and
- participation.

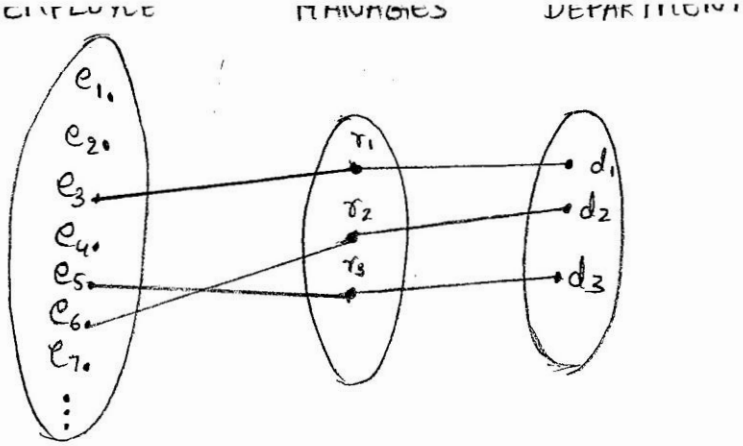
CARDINALITY RATIOS FOR BINARY RELATIONSHIPS

The **cardinality ratio** for a binary relationship specifies the maximum number of relationship instances that an entity can participate in.

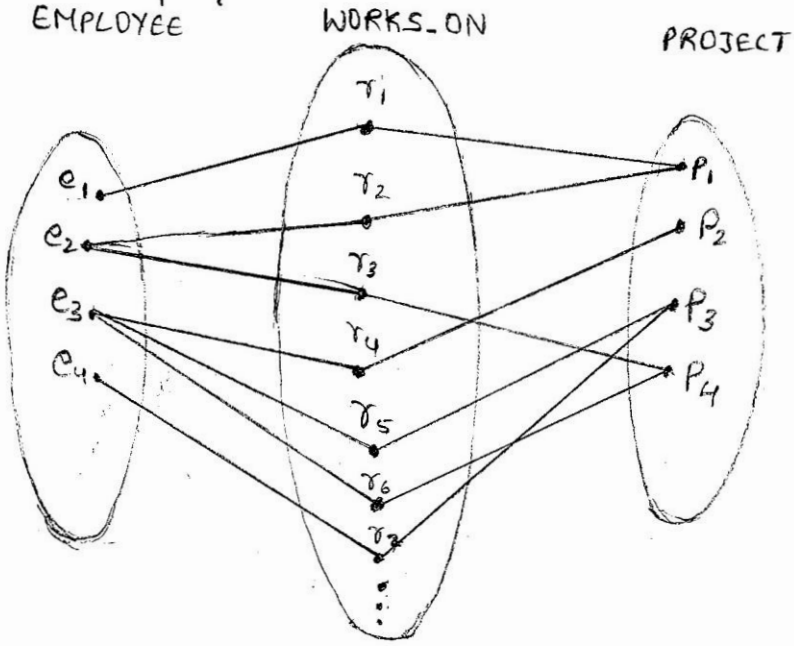
Ex 1. : In the WORKS-FOR binary relationship type, DEPARTMENT : EMPLOYEE is of cardinality ratio 1 : N, meaning that each department can be related to any number of employees. but an employee can be related to (work for) only one department.

The possible cardinality ratios for binary relationship types are 1 : 1, 1 : N, N : 1 and M : N.

Ex 2: Binary relationship MANAGES is 1 : 1 cardinality ratio i.e it relates department entity to employee who manages that department. Which means an employee can manage one deparment only & a department can have one manager only.



Ex 3 : They relationship type WORKS_ON is of cardinality ratio M:N because minicoorld rule is that an employee can coork on seaveral projects and a project can have seveeal employees.



Cardinality ratios for binary relationships are represented on ER diagrams by displaying 1, M and N on the diamonds.

PARTICIPATION CONSTRAINTS AND EXISTENCE DEPENDENCIES

The **Participation** constraint specifies whether the existence of entity depends on its being related to another entity via the relationship type. This constraint specifies minimum number of relationship instances that each entity can particate in, & is sometimes called minimom cardinality constraint.

There are two types of participation constraints

- total
- partial

Total Participation : 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 emp 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**

Partial Participation : In MANAGES relationship, we do not expect every employee to manage a department, so that the participation of EMPLOYEE in 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.

The cardinality ratio and participation constraints taken together are refer to as **structural constraints** of a relationship type.

In ER diagrams, total participation is displayed as doubled line connecting participating entity type to relationship, whereas partial participation is represented by a single line.

ATTRIBUTES OF RELATIONSHIP TYPES

Relationship Types can also have attributes, similar to those of entity types.

Ex : To record the number of hours per week that an employee works on a particular project, we include an attribute Hours for the WORKS_ON relationship type.

→ Notice that attributes of 1:1 or 1:N relationship types can be migrated to one of the participating entity type.

Ex : Start_date attribute of for the MANAGES relationship can be attribute of either EMPLOYEE or DEPARTMENT, although conceptually it belongs to MANAGES. because it is 1:1

→ For 1:N relationship type, relationship attribute can be migrated only to entity type on N side of relationship

Ex : In WORKS_FOR relationship start_date indicate when employee started working for a department, this attribute can be included as an attribute of Employee.

→ For M:N, some attributes may be determined by combination of participating entities in relationship instance not by any single entity. Such attributes must be specified as relationship attributes.

Ex : Hours per week attribute of M:N relationship WORKS_ON.

WEAK ENTITY TYPES

Weak entity types are those entity types that do not have key attributes of their own.

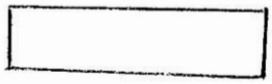
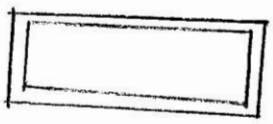
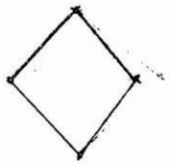
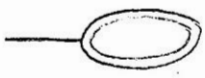
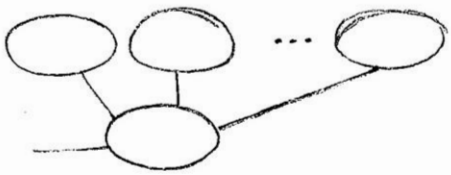
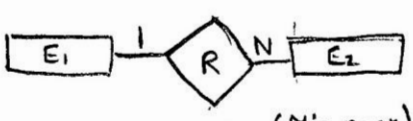
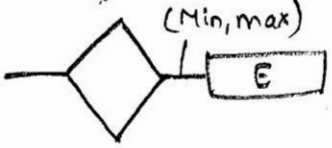
* Regular entity types that do have a key attribute are called strong entity types.
* Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. This other entity type is called as identifying or owner entity type and relationship type that relates a weak entity type to its owner is called as identifying relationship of the weak entity type.
* Weak entity type has always a total participation constraint w.r.t 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.

Ex: DEPENDENT entity type, related to EMPLOYEE, is used to keep track of dependents of employee via 1:N relationship. Two dependents of two distinct employees may, by chance, have same values for all attributes, but they are still distinct entities, because they are identified as distinct entities only after determining particular employee entity to which each dependent is related.

* A weak entity type normally has a partial key, which is the set of attributes that can uniquely identify weak entities that are related to same owner entity.
Ex: Name of DEPENDENT is the partial key.

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

SUMMARY OF THE NOTATION FOR ER DIAGRAMS

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

SPECIALIZATION AND GENERALIZATION IN EER

SPECIALIZATION is the process of defining a set of subclasses of an entity type, this entity type is called the superclass of the specialization. The set of subclasses that forms a specialization is defined on the basis of some distinguishing characteristic of the entities in superclass

Ex 1: The set of subclasses {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of the superclass EMPLOYEE that distinguishes among employee entities based on job type.

Ex 2: Another specialization of EMPLOYEE entity type may yield set of subclasses {SALARIED_EMPLOYEE, HOURLY_EMPLOYEE} distinguishes employees based on method of pay.

- EER notation for specialization :

  The subclasses that define specialization are attached by lines to a circle that represents specializati which is connected in turn to superclass. The subset symbol on each line connecting a subclass to the circle indicates the direction of superclass/subclass relationship.

- Attributes that apply only to entities of a particular subclass - such as typing speed of SECRETARY - are attached to the rectangle representing that subclass.

- Two reasons for including specialization
  → certain attributes may apply to some but not all entity of superclass.
  → Some relationship type may be participated in only

by entities that are members of subclass.

Ex: if only HOURLY_EMPLOYEES can belong to a trade union we can represent that fact by creating subclass HOURLY_EMPLOYEE of EMPLOYEE & relating subclass to an entity type TRADE_UNION via BELONGS_TO relationship type.

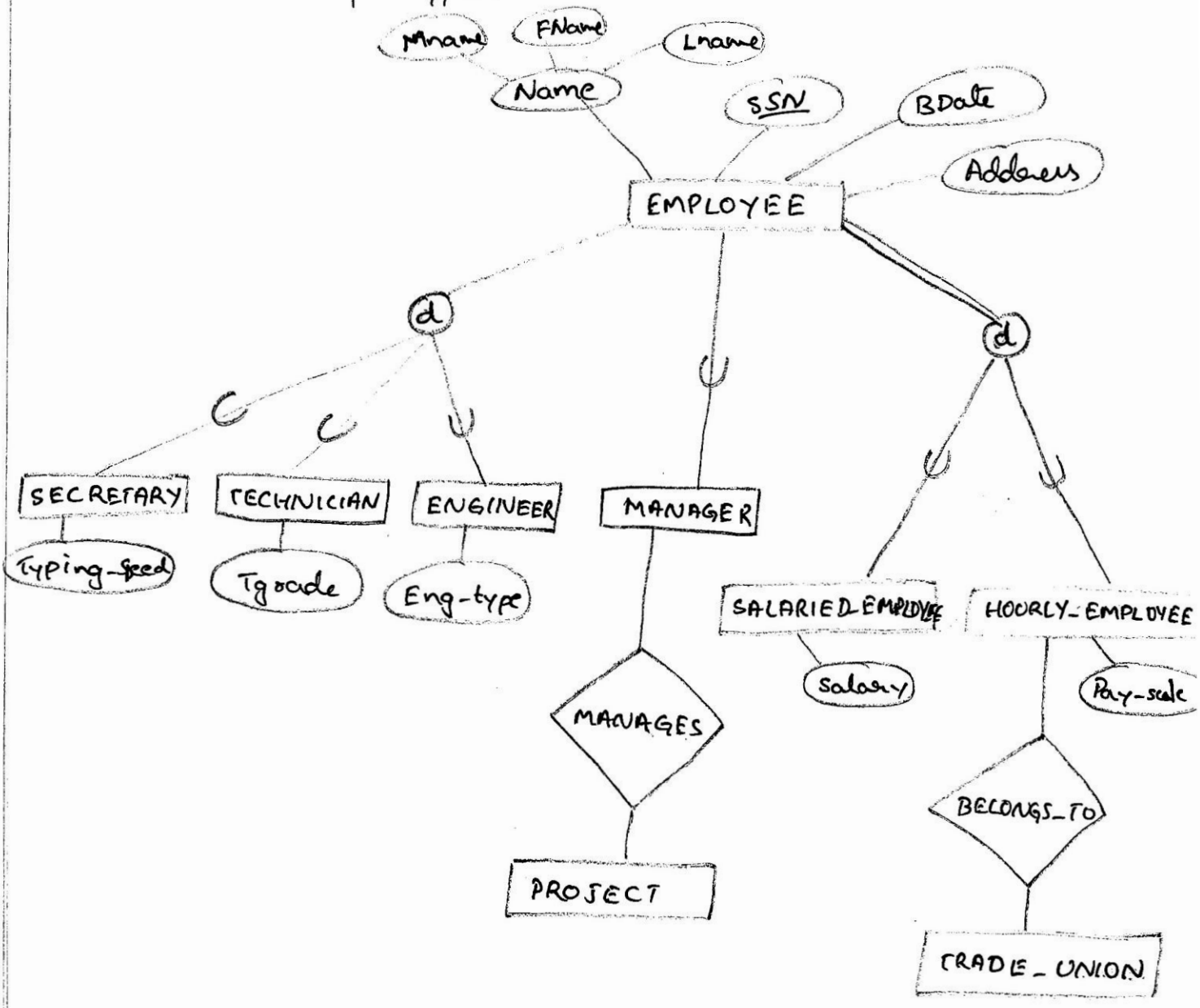


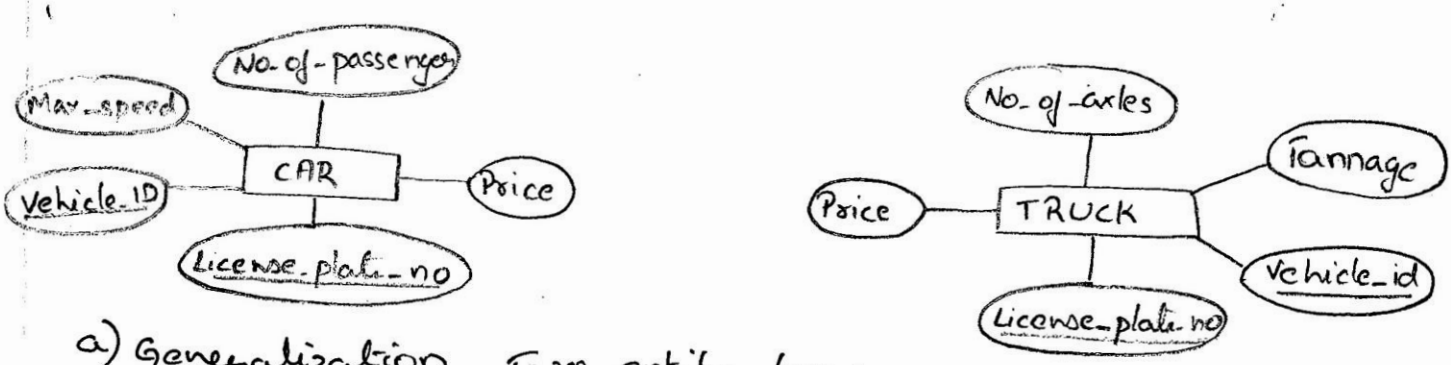
Fig: EER diagram notation to represent subclasses and Specialization.

GENERALIZATION

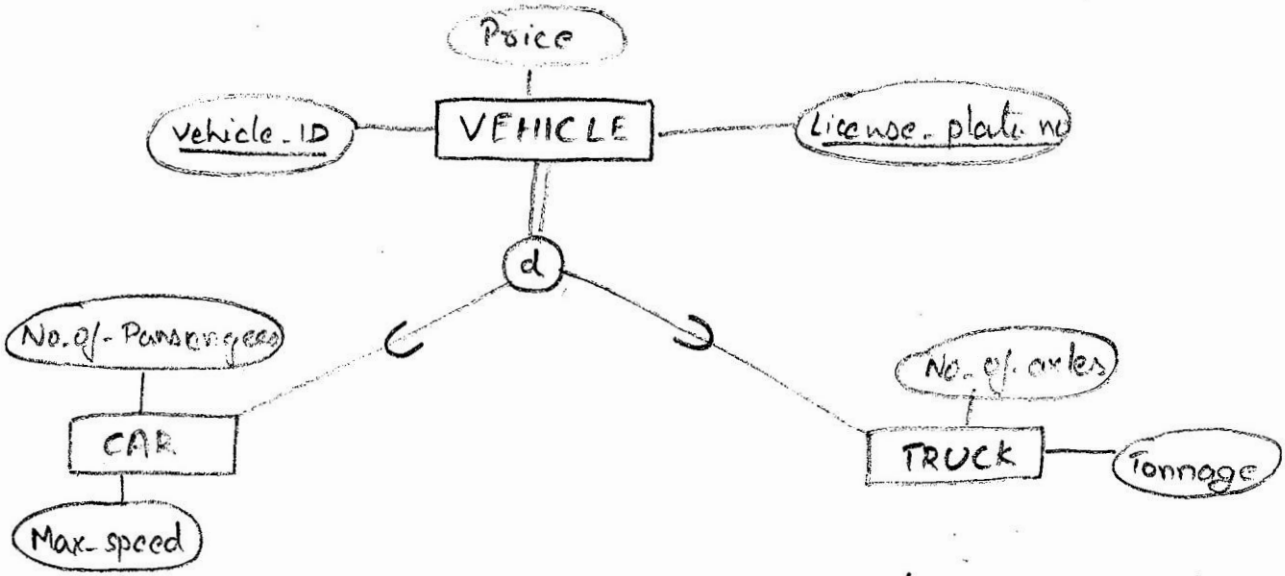
Generalization is the process of defining a generalized entity type from the given entity types.

We can think of a reverse process of abstraction in which we suppress the differences among several entity types, identify their common features, and generalize them into a single superclass of which original entity types are special subclasses.

Ex: Consider the entity types CAR and TRUCK shown in below fig. Because they have several common attributes, we can generalize them into entity type VEHICLE. Both CAR & TRUCK are now subclasses of generalized superclass VEHICLE.



a) Generalization. Two entity types CAR & TRUCK.



b) Generalizing CAR & TRUCK into the superclass VEHICLE

An Example Database Application : COMPANY

Consider an example database application, called COMPANY which serves to illustrate the basic ER model concepts and their use in schema design. The COMPANY database keeps track of a company's employee departments and projects. Here we list out the data requirements collection & analysis phase, the database designers provide following description of the miniworld :

- The company is organized into departments. Each department has a unique name, unique number, & a particular employee who manages the department. We keep track of start date when that employeee began managing the department. A department 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, address, salary, sex, and date of birth. An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by same department. We keep track of number of hours per week that an employee works on each project. We also keep track of direct supervisor of each 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.

Schema Diagram for COMPANY database

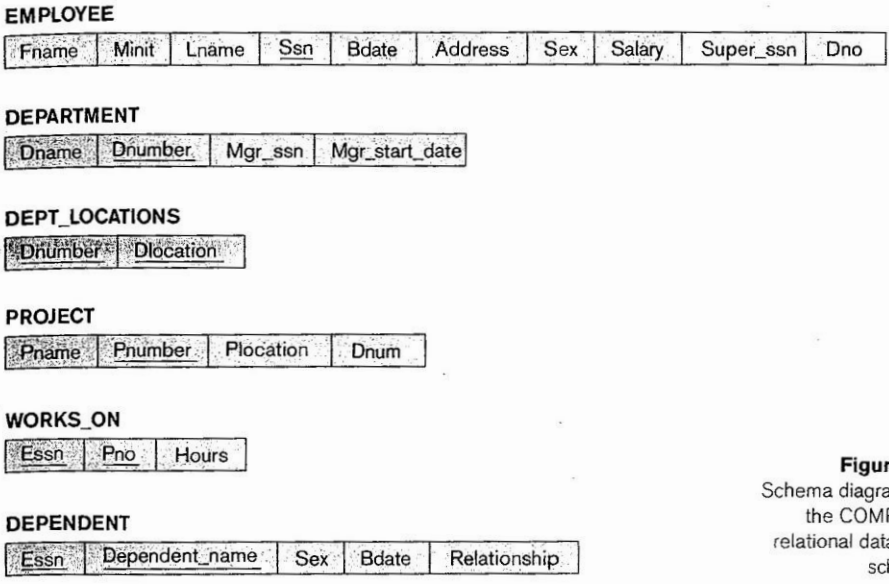


Figure 5.5
Schema diagram for the COMPANY relational database schema.

Fig : Schema diagram for the COMPANY relational database.

13 ER Diagram for the COMPANY database

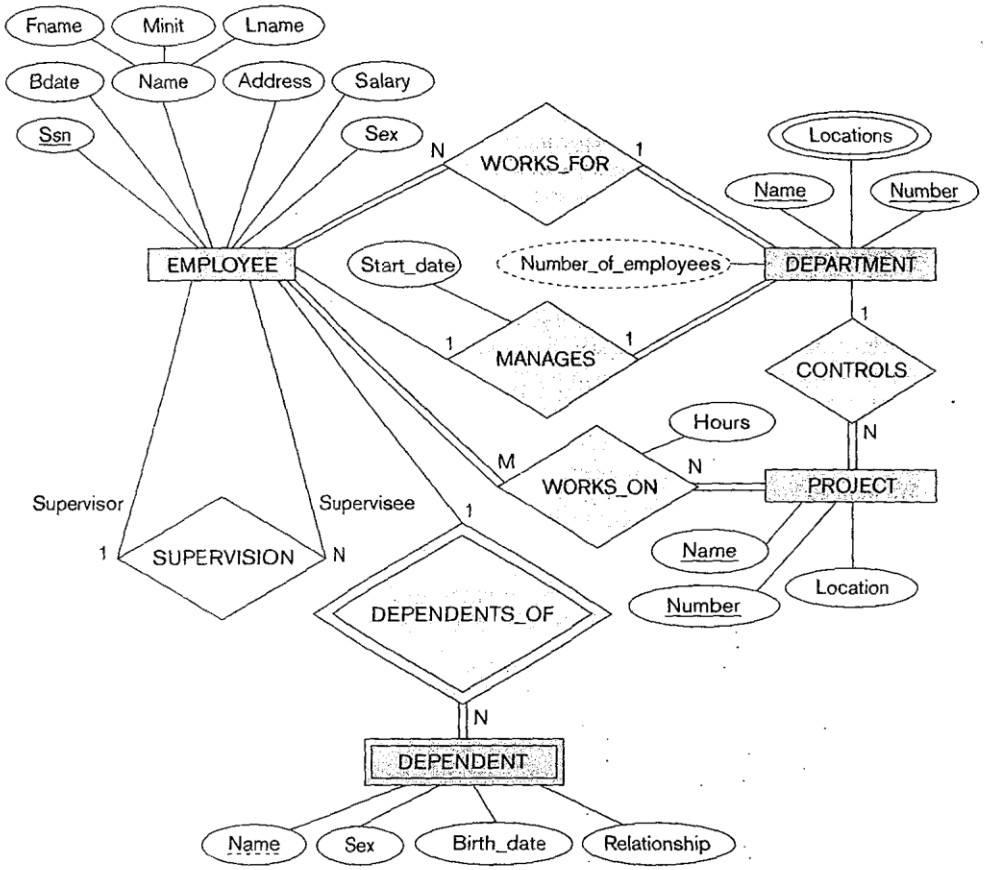


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Fig : An ER schema diagram for the COMPANY database.

MODULE 1 QUESTION BANK

1. Define the following terms :
   a) Database
   b) DBMS
   c) Database system
   d) DBA
   e) Data Abstraction
   f) Data Independence

2. What are the advantages of a database system? What are the various costs and risk factors involved in implementing database system ?

3. Explain the characteristics of the database approach and how it differs from traditional file system.

4. Discuss the disadvantages of file processing system that led to the development of database system.

5. What do you understand by the term data abstraction ?

6. Discuss the various areas in which database system is used.

7. Who is DBA ? List the various responsibilities of DBA.

8. What are the different types of database users who interact with database system?

9. Explain the three-level architecture of DBMS.

10. Write a short note on data definition language and data manipulation language.

11. Explain various functional components

11. Explain the various functional components of a DBMS with the help of suitable diagram.

12. Write a short note on conceptual data modeling.

13. Define the following terms

a) Entity
b) Entity type
c) Entity set
d) Attribute
e) Key attribute
f) Domain
g) composite key
h) Degree of relationship
i) Relationship Type
j) Relationship set
k) Weak entity

14. What are the various types of attributes? Discuss the various situations in which an attribute can use a null value.

15. What is a data model? Explain different categories of Data Models.

16. Explain the following with suitable example
a) schemas b) Instance or database state
c) schema diagram d) Intension e) Extension.

17. What are relationship type and relationship instance?

18. What is identifying relationship?

19. What is the significance of using role name in the description of relationship types? In what situations are role names necessary?

20. Discuss various structural constraints that are applied on relationship types.

21. What is meant by recursive relationship type ? Ex

22. Discuss the E-R symbols that are used in E-R diagrams to represent various types of information.

23. What is a weak entity type ? How are weak entity types represented in E-R diagram ?

24. Discuss specialization and generalization with the help of an example.

25. Explain cardinality Ratios for binary relationships.

26. Explain participation constraints