

Haramaya University



COLLEGE OF COMPUTING AND INFORMATICS
DEPARTMENT OF COMPUTER SCIENCE
FUNDAMENTAL OF DATABASE SYSTEM
HANDOUT!!

JULY 2021

CHAPTER 1: INTRODUCTION

- ✓ **DATABASE** is shared collection of logically related data (and a description of this data), designed to meet the information needs of an organization. (Centralized System)
- ✓ A collection of related data.
- ✓ **Data:** Known facts that can be recorded and have an implicit meaning.
- ✓ **Database Management System (DBMS):** A software package/ system to facilitate the creation and maintenance of a computerized database.
- ✓ **Database System:** The DBMS software together with the data itself. Sometimes, the applications are also included. Its elements are the data in the database, Hardware (client and server computers), Software (Application program and DBMS, DDL, DML, TML etc.) and users (end users, application programmers, database administrator)

DATABASE DEVELOPMENT LIFE-CYCLE

- ✚ Database planning
- ✚ Data analysis and requirements
- ✚ Define Problems and Constraints
- ✚ Database design
- ✚ DBMS selection
- ✚ Prototyping
- ✚ Implementation
- ✚ Testing
- ✚ Operational maintenance

➤ Database Planning

- ✓ Analyze the Company Situation
 - What is the organization's general operating environment, and what is its mission within that environment?
 - What is the organization's structure?
- ✓ Identifies work to be done; the resources with which to do it; and the money to pay for it all.
- ✓ Integrated with the overall IS strategy of the organization.

➤ Data analysis and requirements

- ✓ Designer's efforts are focused on
 - ✚ Information needs,
 - ✚ Information sources.
- ✓ Sources of information for the designer
 - ✚ Developing and gathering end user data views
 - ✚ Direct observation of the current system:
 - ✚ existing system and desired output
- ✓ The designer must identify the company's business rules and analyze their impacts.

➤ Define Problems and Constraints

- ✚ How does the existing system function?
- ✚ What input does the system require?
- ✚ What reports does the system generate?
- ✚ How is the system output used? By Whom?
- ✚ What are the operational relationships among business units?
- ✚ What are the limits and constraints imposed on the system?

➤ Database Design

- ✚ Gather user and system requirements
- ✚ Create a **conceptual model** of the database using the **Entity Relationship model** that is based on the user requirements
- ✚ Convert this **conceptual model (E-R)** into a **logical database model** - we will use the Relational model. *Data model mapping*
 - ✚ Normalize the Relational model of the database.
 - ✚ Implement the normalized relations as tables in a relational database – this is the **Physical Database design** and implementation.
 - ✚ Choose a DBMS - in our case, we will use MS-SQL Server, a desktop relational DBMS.

✚ Design:

- ✚ in database designing more emphasis is given to this phase.
- ✚ The phase is further divided into three sub-phases: -

⚠. Conceptual Design: concise description of: -

- ✓ Entities and
- ✓ Relationship between entities.

B. Logical Design: a higher-level conceptual abstraction with selected specific data model to implement the data structure.

- ✚ Attributes
- ✚ Primary key
- ✚ Foreign key
- ✚ Mapping E-R diagram into relational schema.

C. Physical Design: involves the actual design of a database according to the requirements that were established during logical modelling.

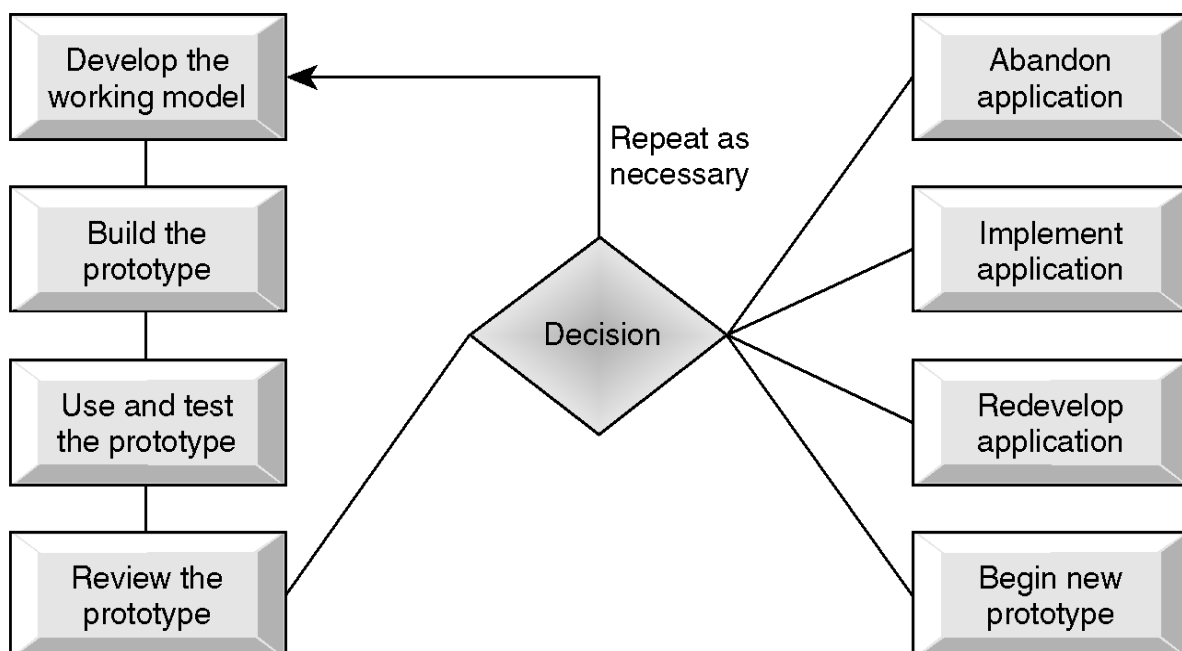
- ✚ Table names
- ✚ Column names
- ✚ Column data types

➤ DBMS SELECTION

- ✚ The selection of an appropriate DBMS to support the database application.
- ✚ Also design the user interface and the application programs using the selected DBMS.

➤ **PROTOTYPING:** Building a working model of a database application.

- Purpose
 - ✚ To identify features of a system that work well, or are inadequate
 - ✚ To suggest improvements or even new features
 - ✚ To clarify the users' requirements
 - ✚ To evaluate the feasibility of a particular system design.



Prototype Development Method Stages

➤ Implementation

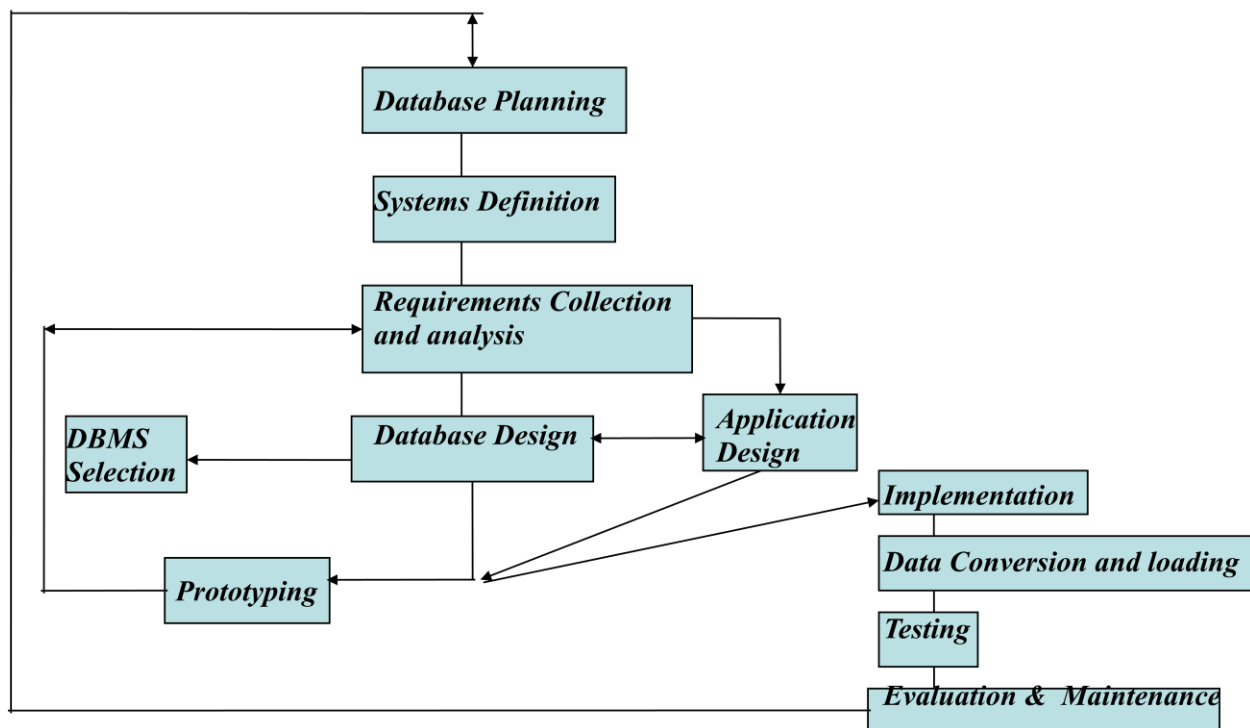
- The physical realization of the database and application designs.
 - Use DDL of DBMS to create database schemas and empty database files.
 - Use DML to create any specified user views.

➤ Testing

- ✚ The process of executing the application programs with the intent of finding errors.
 - Use carefully planned test strategies and realistic data.
 - Testing cannot show the absence of faults; it can show only that software faults are present.
 - Demonstrates that database and application programs *appear* to be working according to requirements.

➤ Operational Maintenance

- The process of monitoring and maintaining the system following installation.
- Monitoring the performance of the system.
- If performance falls, may require reorganization of the database.
- Maintaining and upgrading the database application (when required).
- Incorporating new requirements into the database application.



Data base development Life Cycle

1.2 File based versus Database approach

Before the advent of computerized file handling systems, a manual method of file handling system which is dependent on the use of file cabinets, drawers, pens, paper and pencil were used.

The limitation of the manual file handling system includes:

-  Prone to error
 -  Less secured
 -  Not sharable
 -  Difficult to update, retrieve, integrate
 -  You have the data but it is difficult to compile the information
-  Two computerized approaches evolved to overcome the limitations of the manual file handling system: -



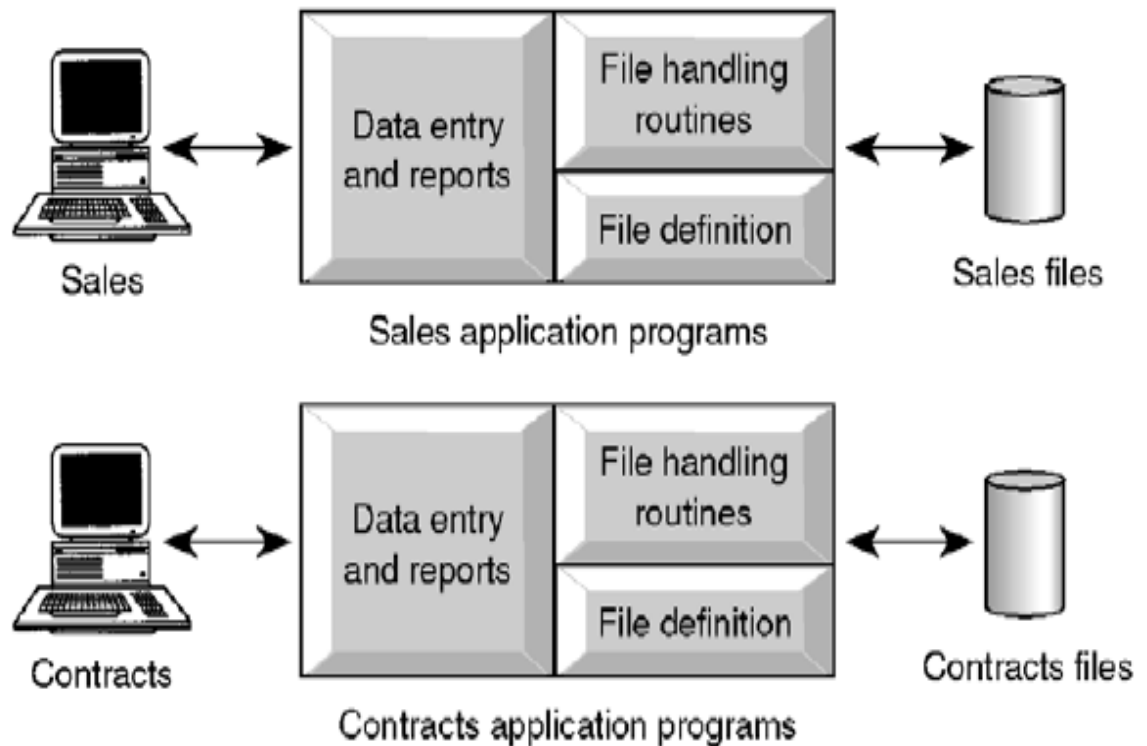
File based approach decentralised



Database approach centralised

➤ **FILE-BASED APPROACH**

-  File based systems were an early attempt to computerize the manual filing system.
-  This approach is the decentralized computerized data handling method.
-  A collection of application programs performs services for the end-users.
-  In such systems, every application program that provides service to end users define and manage its own data.
-  Such systems have number of programs for each of the different applications in the organization.
-  Since every application defines and manages its own data, the system is subjected to serious data duplication problem.
-  File, in traditional file-based approach, is a collection of records which contains logically related data.



➤ Limitations of the Traditional File Based approach

- ✚ Limited data sharing
- ✚ Lengthy development and maintenance time
- ✚ Duplication or redundancy of data
- ✚ Data dependency on the application
- ✚ Incompatible file formats between different applications and programs creating inconsistency.
- ✚ The most significant problem experienced by the traditional file-based approach of data handling is the “**anomalies**”. We have three types of **anomalies**;
 1. **Modification Anomalies**: a problem experienced when one or more data value is modified on one application program but not on others containing the same data set.
 2. **Deletion Anomalies**: a problem encountered where one record set is deleted from one application but remain untouched in other application programs.
 3. **Insertion Anomalies**: a problem experienced whenever there is new data

item to be recorded, and the recording is not made in all the applications.

- And when same data item is inserted at different applications, there could be errors in encoding which makes the new data item to be considered as a totally different object.

➤ THE DATABASE APPROACH

- ✓ To become more effective, a new approach was required by the name database approach.
- ✓ A Database is shared collection of logically related data (and a description of this data), designed to meet the information needs of an organization. (Centralized System)
- ✓ The purpose of database is to: -
 - ✚ store information and
 - ✚ to allow users to retrieve and update that information on demand.
- ✓ Database is designed once and used simultaneously by many users.
- ✓ Each database application will perform the combination of:
 - Creating database,
 - Reading,
 - Updating and
 - Deleting data.

➤ Characteristics of database approach

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

➤ Benefits of the database approach

- ✚ **Data can be shared**: two or more users can access and use same data instead of storing data in redundant manner for each user.
- ✚ **Improved accessibility of data**: by using structured query languages, the users can easily access data without programming experience.
- ✚ **Redundancy can be reduced**: isolated data is integrated in database to decrease the redundant data stored at different applications.
- ✚ **Inconsistency can be avoided**: controlled data redundancy will avoid inconsistency of the data in the database to some extent.
- ✚ **Transaction support can be provided**: basic demands of any transaction support systems are implanted in a full-scale DBMS.
- ✚ **Integrity can be maintained**: data at different applications will be integrated together with additional constraints to facilitate shared data resource.

- ✚ **Security measures can be enforced**: the shared data can be secured by having different levels of clearance and other data security mechanisms.
- ✚ **Centralized information control**: it can be controlled and managed at the central level.

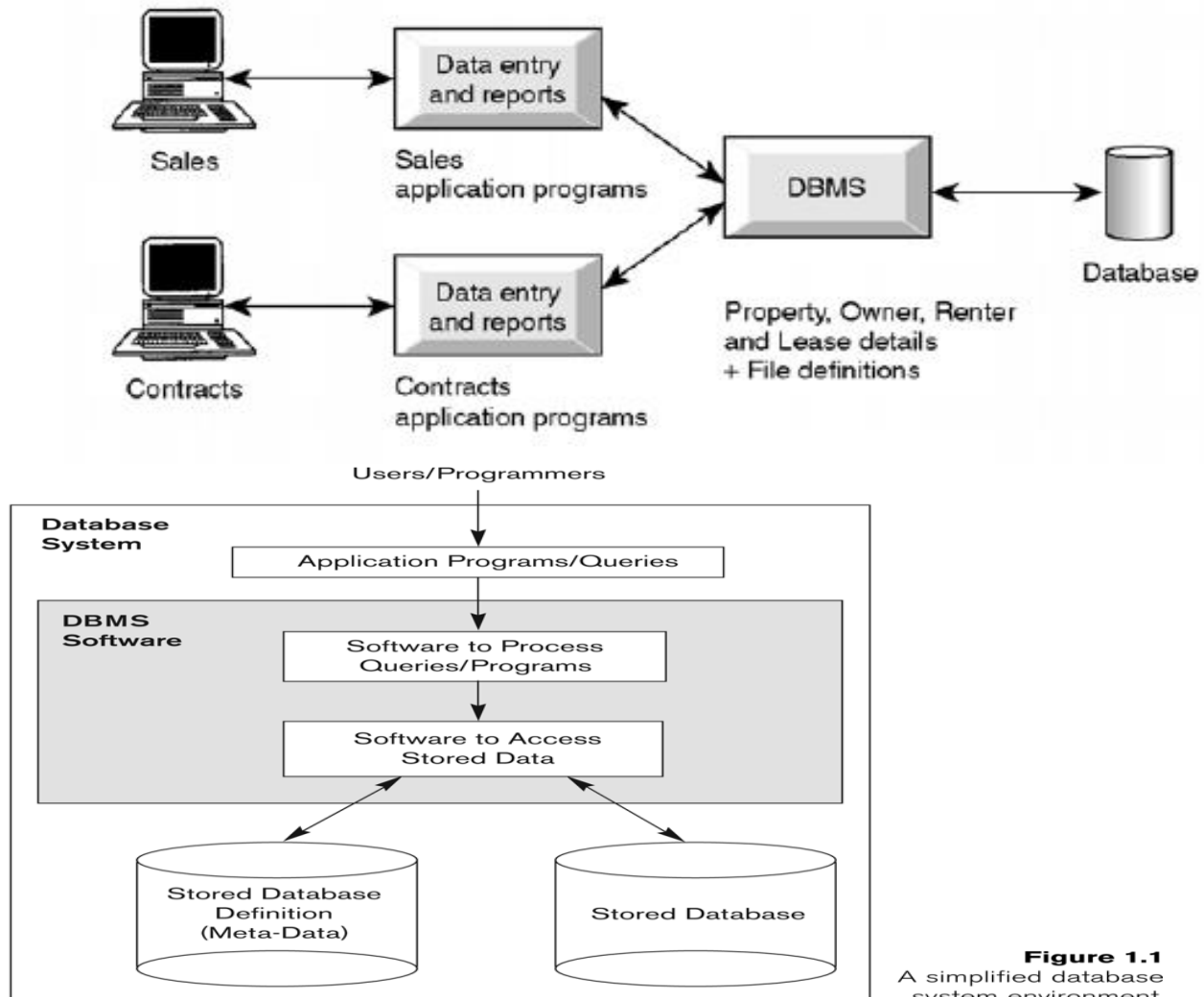


Figure 1.1
A simplified database system environment.

Some Common uses of Databases

In a university, there is a **database**

-  containing information about yourself, the course you are enrolled in, the dormitory you have been given.
-  containing details of Staff who work at the university at personnel, payroll, etc.

When you visit your library

-  There may be a database containing details of the books in the library and details of the users,
-  The database system handles activities such as
 - ✓ Allowing a user to reserve a book
 - ✓ Charging materials to users
 - ✓ Notifying when materials are overdue: Sends out reminders to borrowers who have failed to return books on the due data
-  The system will have a bar code reader to keep track of books and users.

➤ **Limitations and risk of Database Approach**

-  Introduction of new professional and specialized personnel.
-  Complexity in designing and managing data.
-  High cost to be incurred to develop and maintain the system.
-  Complex backup and recovery services from the user's perspective.
-  Reduced performance due to centralization and data independence
-  High impact on the system when failure occurs to the central system.

1.3 USERS AND ACTORS OF THE DATABASE SYSTEM

Users may be divided into those who actually use and control the content (called “**Actors on the Scene**”) and those who enable the database to be developed and the DBMS software to be designed and implemented (called “**Workers Behind the Scene**”).

-  **Database administrators:** responsible for authorizing access to the database, for coordinating and monitoring its use, acquiring software, and hardware resources, controlling its use and monitoring efficiency of operations.

- ✚ **Database Designers:** responsible to define the content, the structure, the constraints, and functions or transactions against the database. They must communicate with the end-users and understand their needs.
- ✚ **Application developer:** are software programmers that designs and implements the application programs to be used as an interface for the database system and for manipulating the data in the database.
- ✚ **End-users:** they use the data for queries, reports and some of them actually update the database content
 - ✚ **Casual:** access database occasionally when needed
 - ✚ **Naïve or Parametric:** they make up a large section of the end-user population. They use previously well-defined functions in the form of “canned transactions” against the database. Examples are bank-tellers or reservation clerks who do this activity for an entire shift of operations.
 - ✚ **Sophisticated:** these include business analysts, scientists, engineers, others thoroughly familiar with the system capabilities. Many use tools in the form of software packages that work closely with the stored database.
 - ✚ **Stand-alone:** mostly maintain personal databases using ready-to-use packaged applications. An example is a tax program user that creates his or her own internal database.

Chapter 2: Database system Concepts & Architecture

2.1 Data models, Schemas and Instances

➤ Data Model:

- ✚ A set of concepts to describe the **structure** of a database, the **operations** for manipulating these structures, and certain **constraints** that the database should obey.

➤ Data Model Structure and Constraints:

- ✚ Constructs are used to define the database structure
- ✚ Constructs typically include **elements** (and their **data types**) as well as groups of elements (e.g. **entity**, **record**, **table**), and **relationships** among such groups
- ✚ Constraints specify some restrictions on valid data; these constraints must be enforced at all times

➤ Data Model Operations:

- ✚ These operations are used for specifying database *retrievals* and *updates* by referring to the constructs of the data model.
- ✚ Operations on the data model may include **basic model operations** (e.g. generic insert, delete, update) and **user-defined operations** (e.g. compute student_gpa, update_inventory)

➤ Schemas versus Instances

Database Schema:

- ✚ The **description** of database.
- ✚ Includes descriptions of the database structure, data types, and the constraints on the database.

Schema Diagram:

- ✚ An **illustrative** display of (most aspects of) a database schema.

Schema Construct:

- ✚ A **component** of the schema or an object within the schema, e.g., STUDENT, COURSE.

Schemas versus Instances

Database State:

- ✚ The actual data stored in a database at a **particular moment in time**. This includes the collection of all the data in the database.
- ✚ Also called database instance (or occurrence or snapshot).
- ✚ The term *instance* is also applied to individual database components, e.g. *record instance*, *table instance*, *entity instance*

➤ Database Schema vs. Database State

➤ Database Schema:

- ✚ The description of database.
- ✚ Includes descriptions of the database structure and the constraints that should hold on the database.

➤ Database State:

- ✚ Refers to the **content** of a database at a moment in time.

Initial Database State:

- ✚ Refers to the database state when it is initially loaded into the system.

Valid State:

- ✚ A state that satisfies the structure and constraints of the database.

Distinction

- ✚ The **database schema** changes very infrequently.
- ✚ The **database state** changes every time the database is updated.

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

2.2 The Database System Architecture

The major purpose of a database system is to provide users with an abstract view of the system which some time referred as **architecture of the system**.

The system hides certain details of how data is stored and maintained

Complexity should be hidden from database users.

There are several levels of abstraction of the database architecture:

(a) Physical Level:

- ✓ How the data are stored.
- _ E.g. index, B-tree, hashing.
- ✓ Lowest level of abstraction.
- ✓ Complex low-level structures described in detail.

(b) Conceptual Level:

- ✓ Next highest level of abstraction.
- ✓ Describes what data are stored.
- ✓ Describes the relationships among data.
- ✓ Database administrator level.

(c) View Level:

- ✓ Highest level.
- ✓ Describes part of the database for a particular group of users.
- ✓ Can be many different views of a database.

E.g. tellers in a bank get a view of customer accounts, but not of payroll data.

Figure 2.2

The three-schema architecture.

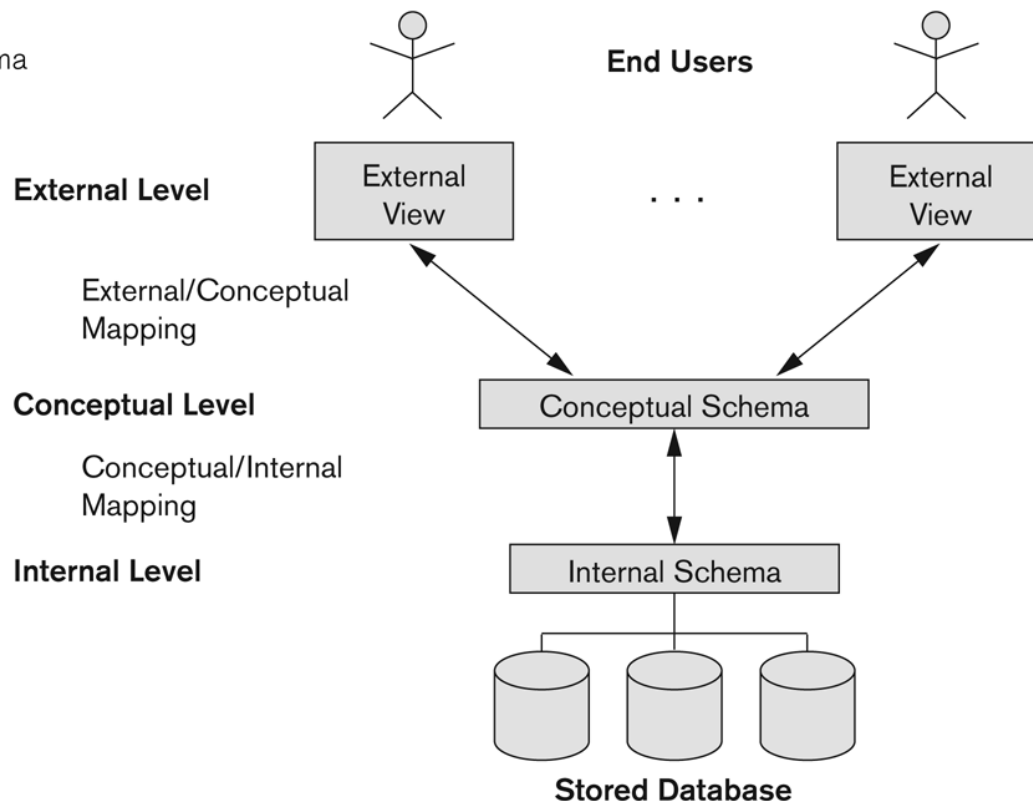


Fig: The three levels of data abstraction of a database system

A database system structure or architecture can have different components at different layers of abstraction.

+ Database Administrator

1. The database administrator is a person having central control over data and programs accessing that data. Duties of the database administrator include:

- **Scheme definition:** the creation of the original database scheme. This involves writing a set of definitions in a DDL (data storage and definition language), compiled by the DDL compiler into a set of tables stored in the data dictionary.

- **Storage structure and access method definition:** writing a set of definitions translated by the data storage and definition language compiler
- **Scheme and physical organization modification:** writing a set of definitions used by the DDL compiler to generate modifications to appropriate internal system tables (e.g. data dictionary). This is done rarely, but sometimes the database scheme or physical organization must be modified.
- **Granting of authorization for data access:** granting different types of authorization for data access to various users
- **Integrity constraint specification:** generating integrity constraints. These are consulted by the database manager module whenever updates occur.

Database Users

The database users fall into several categories:

- ✓ **Application programmers** are computer professionals interacting with the system through DML calls embedded in a program written in a host language (e.g. C, PL/1, Pascal).
 - These programs are called application programs.
 - The DML precompiler converts DML calls (prefaced by a special character like \$, #, etc.) to normal procedure calls in a host language.
 - The host language compiler then generates the object code.
 - Some special types of programming languages combine Pascal-like control structures with control structures for the manipulation of a database.
 - These are sometimes called fourth-generation languages.
 - They often include features to help generate forms and display data.
- ✓ **Sophisticated users** interact with the system without writing programs.
 - They form requests by writing queries in a database query language.
 - These are submitted to a query processor that breaks a DML statement down into instructions for the database manager module.
- ✓ **Specialized users** are sophisticated users writing special database application programs. These may be CADD systems, knowledge-based and expert systems, complex data systems (audio/video), etc.
- ✓ **Naive users** are unsophisticated users who interact with the system by using permanent application programs (e.g. automated teller machine).

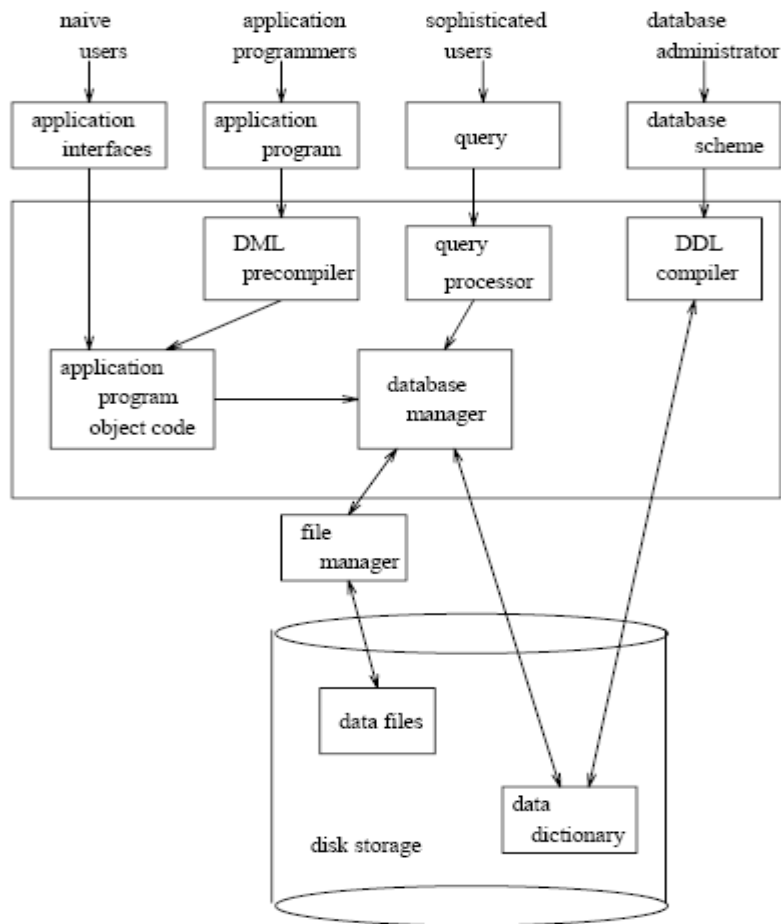


Fig: The database system architecture

Database systems are partitioned into modules for different functions. Some functions (e.g. file systems) may be provided by the operating system.

Components include:

Database Manager

1. The database manager is a program module which provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
2. Databases typically require lots of storage space (gigabytes). This must be stored on disks. Data is moved between disk and main memory (MM) as needed.
3. The goal of the database system is to simplify and facilitate access to data. Performance is important. Views provide simplification.
4. So the database manager module is responsible for

- **Interaction with the file manager**: Storing raw data on disk using the _le system usually provided by a conventional operating system. The database manager must translate DML statements into low-level system commands (for storing, retrieving and updating data in the database).
- **Integrity enforcement**: Checking that updates in the database do not violate consistency constraints (e.g. no bank account balance below \$25)
- **Security enforcement**: Ensuring that users only have access to information they are permitted to see
- **Backup and recovery**: Detecting failures due to power failure, disk crash, software errors, etc., and restoring the database to its state before the failure
- **Concurrency control**: Preserving data consistency when there are concurrent users.

5. Some small database systems may miss some of these features, resulting in simpler database managers. (For example, no concurrency is required on a PC running MS-DOS.) These features are necessary on larger systems.

- ✓ **File manager** manages allocation of disk space and data structures used to represent information on disk.
- ✓ **Database manager**: The interface between low-level data and application programs and queries.
- ✓ **Query processor** translates statements in a query language into low-level instructions the database manager understands. (May also attempt to an equivalent but more efficient form.)
- ✓ **DML precompiles** converts DML statements embedded in an application program to normal procedure calls in a host language. The precompiler interacts with the query processor.
- ✓ **DDL compiler** converts DDL statements to a set of tables containing metadata stored in a data

dictionary.

In addition, several data structures are required for physical system implementation:

- ✓ **Data files**: store the database itself.
- ✓ **Data dictionary**: stores information about the structure of the database. It is used heavily. Great emphasis should be placed on developing a good design and efficient implementation of the dictionary.
- ✓ **Indices**: provide fast access to data items holding particular values.

DBMS Languages

Data Definition Language (DDL)

Data Manipulation Language (DML)

- High-Level or Non-procedural Languages: These include the relational language SQL
 - May be used in a standalone way or may be embedded in a programming language
- Low Level or Procedural Languages:
 - These must be embedded in a programming language

Data Definition Language (DDL):

- Used by the DBA and database designers to specify the conceptual schema of a database.
- In many DBMSs, the DDL is also used to define internal and external schemas (views).
- In some DBMSs, separate **storage definition language (SDL)** and **view definition language (VDL)** are used to define internal and external schemas.
 - SDL is typically realized via DBMS commands provided to the DBA and database designers

Data Manipulation Language (DML):

- Used to specify database retrievals and updates
- DML commands (data sublanguage) can be *embedded* in a general-purpose programming language (host language), such as COBOL, C, C++, or Java.
 - A library of functions can also be provided to access the DBMS from a programming language

Alternatively, stand-alone DML commands can be applied directly (called a *query language*).

High Level or Non-procedural Language:

- For example, the SQL relational language
- Are “set”-oriented and specify what data to retrieve rather than how to retrieve it.
- Also called **declarative** languages.

Low Level or Procedural Language:

- Retrieve data one record-at-a-time;
- Constructs such as looping are needed to retrieve multiple records,

along with positioning pointers.

✚ Stand-alone query language interfaces

- Example: Entering SQL queries at the DBMS interactive SQL interface (e.g. SQL*Plus in ORACLE)

✚ Programmer interfaces for embedding DML in programming languages

✚ User-friendly interfaces

- Menu-based, forms-based, graphics-based, etc.

✚ Programmer interfaces for embedding DML in a programming language:

- **Embedded Approach:** e.g embedded SQL (for C, C++, etc.), SQLJ (for Java)
- **Procedure Call Approach:** e.g. JDBC for Java, ODBC for other programming languages.
- **Database Programming Language Approach:** e.g. ORACLE has PL/SQL, a programming language based on SQL; language incorporates SQL and its data types as integral components

✚ **User-Friendly DBMS Interfaces**

- Menu-based, popular for browsing on the web
- Forms-based, designed for naïve users
- Graphics-based
 - (Point and Click, Drag and Drop, etc.)
- Natural language: requests in written English
- Combinations of the above:
 - For example, both menus and forms used extensively in Web database interfaces

✚ **Other DBMS Interfaces**

- Speech as Input and Output
- Web Browser as an interface
- Parametric interfaces, e.g., bank tellers using function keys.
- Interfaces for the DBA:
 - Creating user accounts, granting authorizations
 - Setting system parameters
 - Changing schemas or access paths

✚ **Database System Utilities**

- To perform certain functions such as:
 - Loading data stored in files into a database. Includes data conversion tools.

- Backing up the database periodically on tape.
- Reorganizing database file structures.
- Report generation utilities.
- Performance monitoring utilities.
- Other functions, such as sorting, user monitoring, data compression, etc.

Other Tools

- Data dictionary / repository:
 - Used to store schema descriptions and other information such as design decisions, application program descriptions, user information, usage standards, etc.
 - **Active data dictionary** is accessed by DBMS software and users/DBA.
 - **Passive data dictionary** is accessed by users/DBA only.

Typical DBMS Component Modules

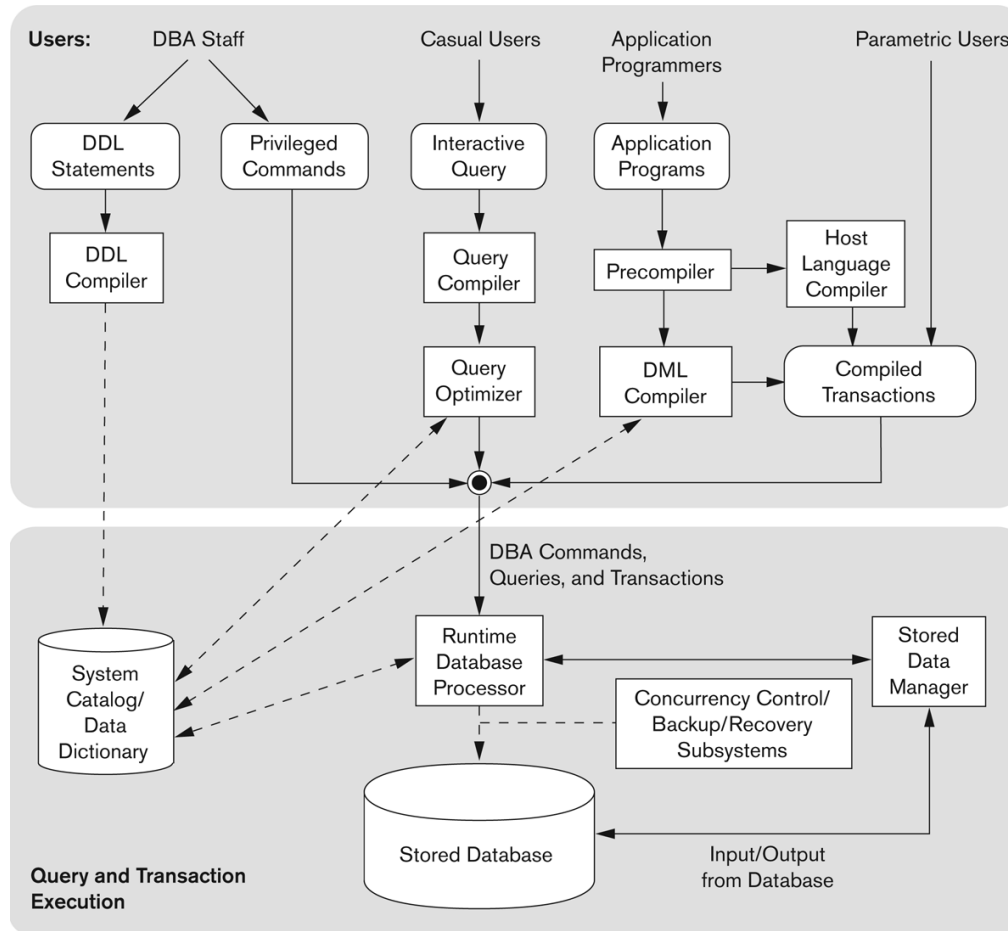


Figure 2.3
Component modules of a DBMS and their interactions.

Centralized and Client-Server DBMS Architectures

Centralized DBMS:

- Combines everything into single system including- DBMS software, hardware, application programs, and user interface processing software.
- User can still connect through a remote terminal – however, all processing is done at centralized site.

A Physical Centralized Architecture

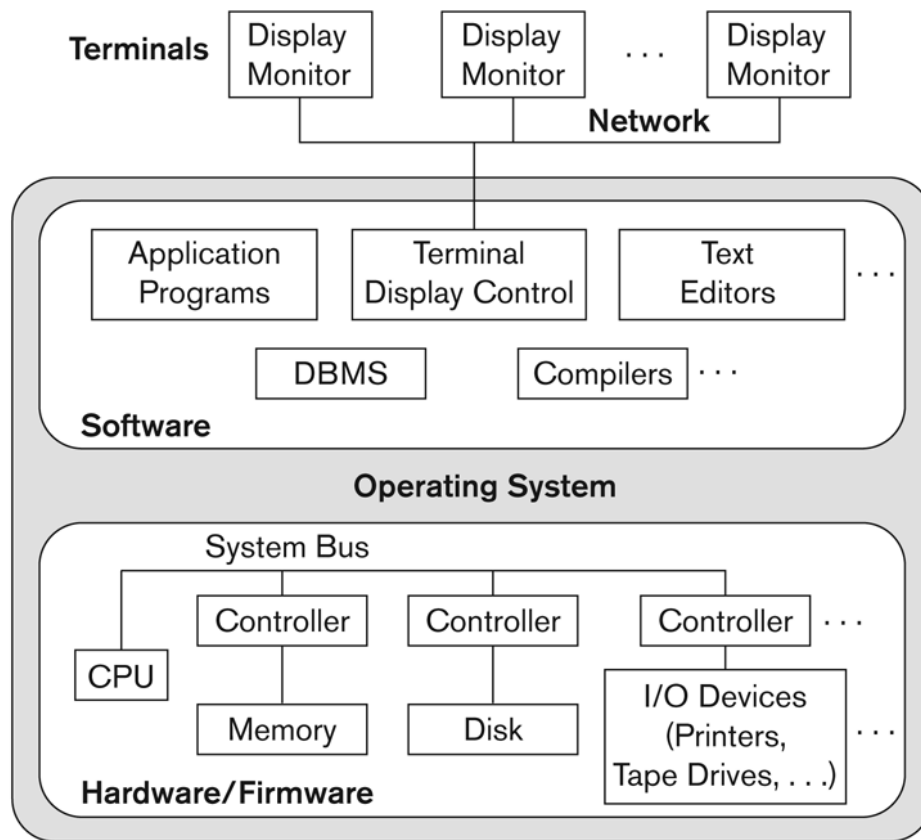


Figure 2.4
A physical centralized architecture.

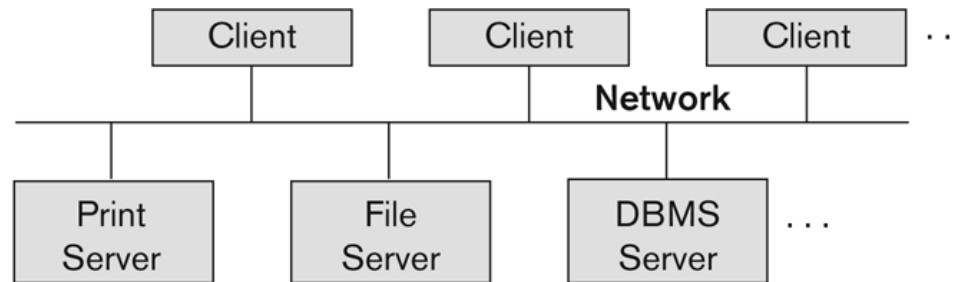
✚ Basic 2-tier Client-Server Architectures

- Specialized Servers with Specialized functions
 - Print server
 - File server
 - DBMS server
 - Web server
 - Email server
- Clients can access the specialized servers as needed

Logical two-tier client server architecture

Figure 2.5

Logical two-tier client/server architecture.



Clients:

- Provide appropriate interfaces through a client software module to access and utilize the various server resources.
- Clients may be disk less machines or PCs or Workstations with disks with only the client software installed.
- Connected to the servers via some form of a network.
 - (LAN: local area network, wireless network, etc.)

DBMS Server:

- Provides database query and transaction services to the clients
- Relational DBMS servers are often called SQL servers, query servers, or transaction servers
- Applications running on clients utilize an Application Program Interface (**API**) to access server databases via standard interface such as:
 - ODBC: Open Database Connectivity standard
 - JDBC: for Java programming access
- Client and server must install appropriate client module and server module software for ODBC or JDBC

Two Tier Client-Server Architecture:

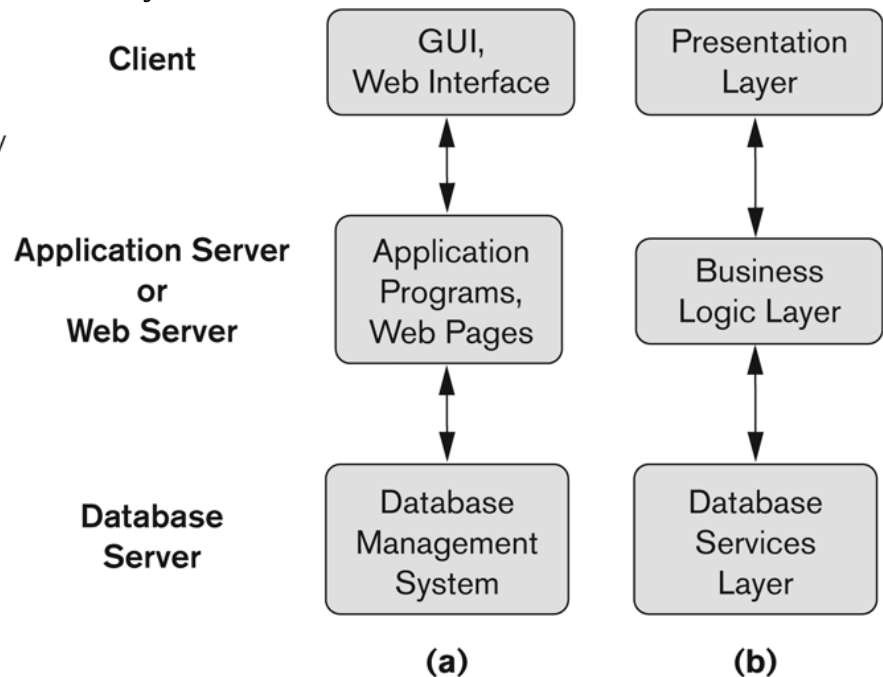
- A client program may connect to several DBMSs, sometimes called the data sources.
- In general, data sources can be files or other non-DBMS software that manages data.
- Other variations of clients are possible: e.g., in some object DBMSs, more functionality is transferred to clients including data dictionary functions, optimization and recovery across multiple servers, etc.

✚ Three Tier Client-Server Architecture:

- Common for Web applications
- Intermediate Layer called Application Server or Web Server:
 - Stores the web connectivity software and the business logic part of the application used to access the corresponding data from the database server
 - Acts like a conduit for sending partially processed data between the database server and the client.
- Three-tier Architecture Can Enhance Security:
 - Database server only accessible via middle tier
 - Clients cannot directly access database server

Figure 2.7

Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.



✚ Classification of DBMSs

- Based on the data model used
 - Traditional: Relational, Network, Hierarchical.
 - Emerging: Object-oriented, Object-relational.
- Other classifications
 - Single-user (typically used with personal computers) vs. multi-user (most DBMSs).
 - Centralized (uses a single computer with one database) vs. distributed (uses multiple computers, multiple databases)

Variations of Distributed DBMSs (DDBMSs):

- Homogeneous DDBMS
- Heterogeneous DDBMS

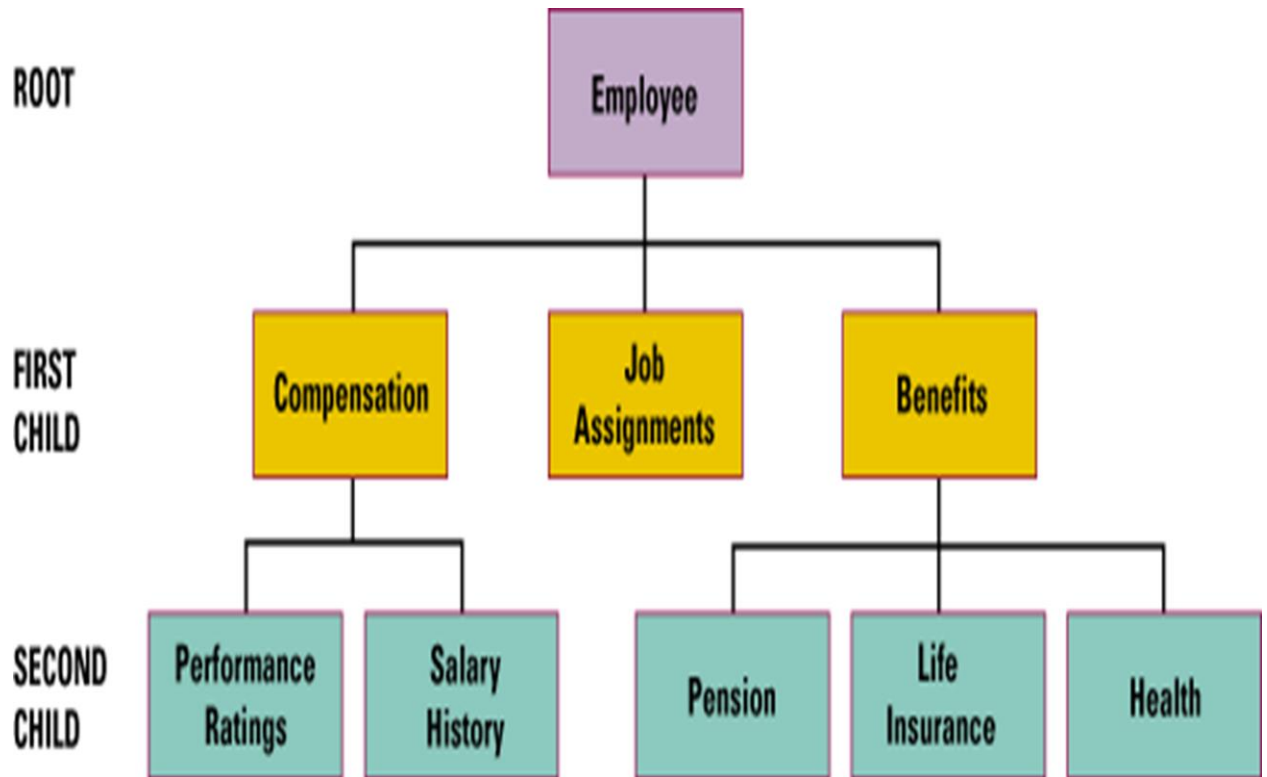
- Federated or Multi database Systems
- Distributed Database Systems have now come to be known as client-server-based database systems because:
 - They do not support a totally distributed environment, but rather a set of database servers supporting a set of clients.

Cost considerations for DBMSs:

- Cost Range: from free open-source systems to configurations costing millions of dollars
- Examples of free relational DBMSs: MySQL, PostgreSQL, others
- Commercial DBMS offer additional specialized modules, e.g. time-series module, spatial data module, document module, XML module
- These offer additional specialized functionality when purchased separately
- Sometimes called cartridges (e.g., in Oracle) or blades Different licensing options: site license, maximum number of concurrent users (seat license), single user, etc.

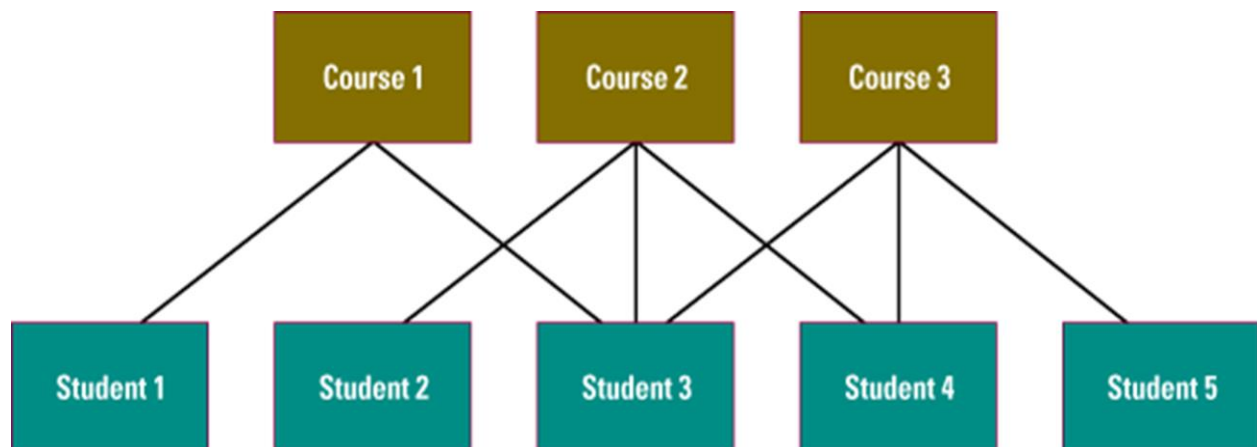
Categories of Data Models

- ✚ **Conceptual (high-level, semantic) data models:**
 - ✚ Provide concepts that are close to the way many users perceive data.
 - (Also called *entity-based* or *object-based* data models.)
- ✚ **Physical (low-level, internal) data models:**
 - ✚ Provide concepts that describe details of how data is stored in the computer. These are usually specified in an ad-hoc manner through DBMS design and administration manuals
- ✚ **Implementation (representational) data models:**
 - ✚ Provide concepts that fall between the above two, used by many commercial DBMS implementations (e.g. relational data models used in many commercial systems).
 - Within the history of database systems, we have: -
 - ➔ **The first-generation data models**
 - Hierarchical Model
 - Network Model
 - ➔ **The second-generation data models**
 - Relational Model
 - ➔ **The third-generation data models**
 - Object Oriented Data Models
- ✚ **Hierarchical Data Model**
 - The simplest data model
 - Record type is referred to as node or segment
 - The top node is the root node
 - Nodes are arranged in a hierarchical structure as sort of upside-down tree
 - A parent node can have more than one child node
 - A child node can only have one parent node
 - The relationship between parent and child is one-to-many Relation is established by creating physical link between stored records



✚ Network Data Model

- Allows record types to have more than one parent unlike hierarchical model
- It doesn't allow many to many relationships between entities
- Like hierarchical model network model is a collection of physically linked records.



✚ Relational Data Model

- Viewed as a collection of tables called “**Relations**” equivalent to collection of record types
- **Relation: Two-dimensional table**
- Stores information or data in the form of tables --> **rows and columns**

- A **row** of the table is called **tuple**--> equivalent to **record**
- A **column** of a table is called **attribute**--> equivalent to **fields**
- Data value is the value of the Attribute.

Alternative terminologies		
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

- The rows represent records (collections of information about separate items)
- The columns represent fields (particular attributes of a record)

A Relational model

customer-id	customer-name	customer-street	customer-city
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

account-number	balance
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table

customer-id	account-number
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

10

The Object-Oriented Model

The object-oriented model is based on a collection of objects, like the E-R model.

- ✓ An object contains values stored in instance variables within the object.
- ✓ _ Unlike the record-oriented models, these values are themselves objects.
- ✓ Thus, objects contain objects to an arbitrarily deep level of nesting.
- ✓ An object also contains bodies of code that operate on the the object.
- ✓ These bodies of code are called methods.

_ Objects that contain the same types of values and the same methods are grouped into classes.

- ✓ A class may be viewed as a type definition for objects.
- _ Analogy: the programming language concept of an abstract data type.
 - ✓ The only way in which one object can access the data of another object is by invoking the method of that other object.
- _ This is called sending a message to the object.
 - ✓ Internal parts of the object, the instance variables and method code, are not visible externally.
 - ✓ Result is two levels of data abstraction.

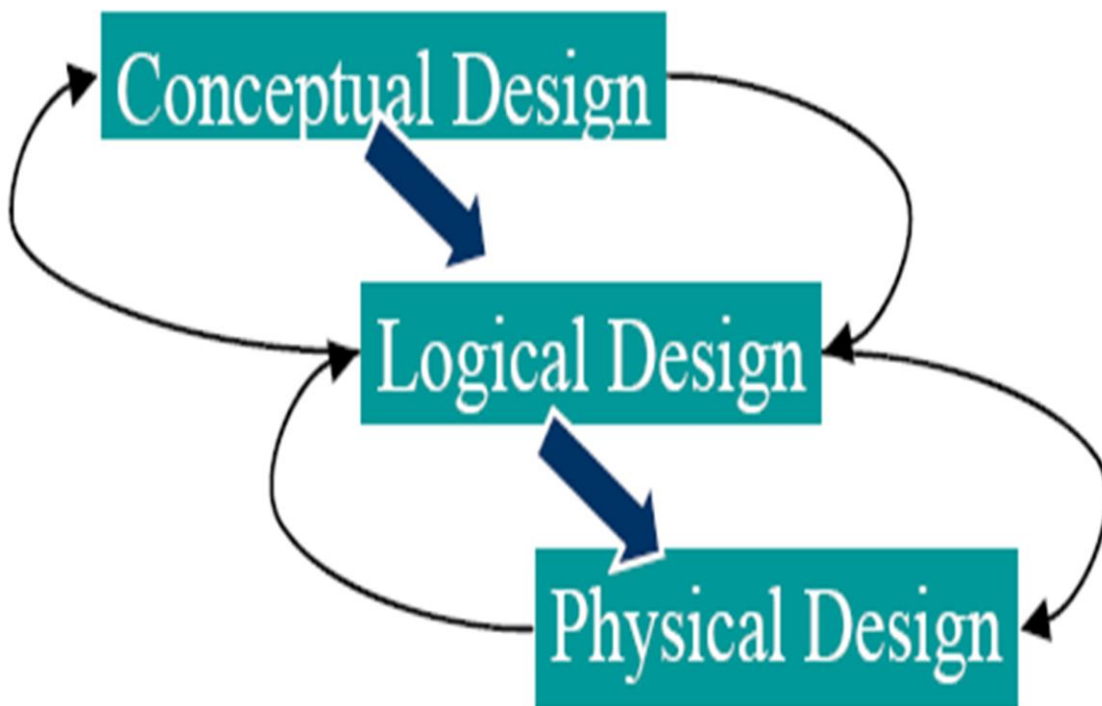
For example, consider an object representing a bank account.

- ✓ The object contains instance variables number and balance.
- ✓ The object contains a method pay-interest which adds interest to the balance.
- ✓ Under most data models, changing the interest rate entails changing code in application programs.
- ✓ In the object-oriented model, this only entails a change within the pay-interest method.

CHAPTER FOUR: Database Modelling using ERD

- In developing a good database design/model, one should answer such questions as:
 - ✚ What are the relevant Entities or the Organization?
 - ✚ What are the important features of each Entity?
 - ✚ What are the important Relationships?
 - ✚ What are the important queries from the user?
 - ✚ What are the other requirements of the Organization and the Users ?

The Three levels of Database Design are the following:



✚ **Conceptual Database Design**

- ✚ Is the process of constructing a model of the information used in an enterprise, *independent of any physical considerations*.
- ✚ It is the source of information for the **logical design phase**.
- ✚ Mostly uses an **Entity Relationship Model** to describe the data at this level.
- ✚ After the completion of Conceptual Design one has to go for refinement of the **schema**, which is verification of **Entities**, **Attributes**, and **Relationships**.

Relationships

- Represents the association that exist between entities
- Represented by Diamond shape.

Constraints

- Represent the constraint in the data.
- Before working on the **conceptual design** of the database, one has to answer the following basic questions: -
 -  What are the **entities** and **relationships** in the enterprise?
 -  What information about these entities and relationships should we store in the database?
 -  What are the integrity constraints that hold? Constraints on each data with respect to update, retrieval and store.
 -  Represent this information pictorially in **ER diagrams**,
 - then map **ER diagram** into a **relational schema**.

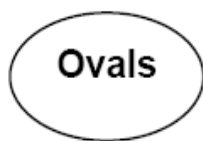
Developing an E-R Diagram

-  Designing conceptual model for the database is not a linear **process** but an **iterative activity** where the design is refined again and again.
-  To identify: -
 - ✓ Entities,
 - ✓ Attributes,
 - ✓ Relationships, and
 - ✓ Constraints on the data,
- There are different set of methods used during the **analysis phase**.
- These include information gathered by...
 -  Interviewing end users individually and in a group
 -  Questionnaire survey
 -  Direct observation
 -  Examining different documents
- The basic E-R model is graphically depicted and presented for review.
- The process is repeated until the end users and designers agree that the E-R diagram is a fair representation of the organization's activities and functions.
- Checking for Redundant Relationships in the ER Diagram.
- Relationships between entities indicate access from one entity to another -
- it is therefore possible to access one entity occurrence from another entity occurrence even if there are other entities and relationships that separate them -
- This is often referred to as **Navigation** of the ER diagram.

- **Validating** an ER Model against requirement of the user.
- Graphical Representation in E-R Diagram**
- **Entity** is represented by a **RECTANGLE** containing the name of the entity.



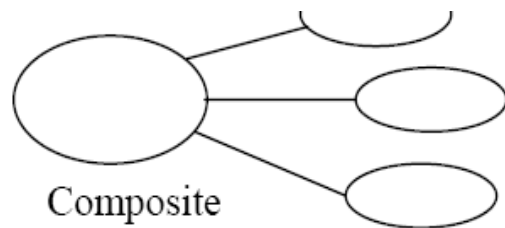
- Connected entities are called **relationship participants**.
- **Attributes** are represented by **OVALS** and are connected to the entity by a line.



Attribute



Multi-valued
Attribute



Composite
Attribute

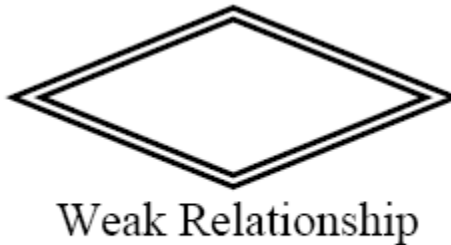
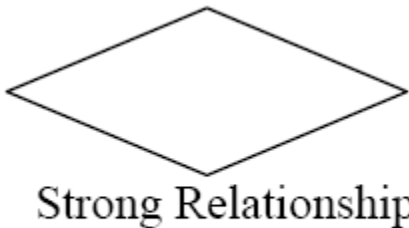
A **derived attribute** is indicated by a **DOTTED LINE**. (...)



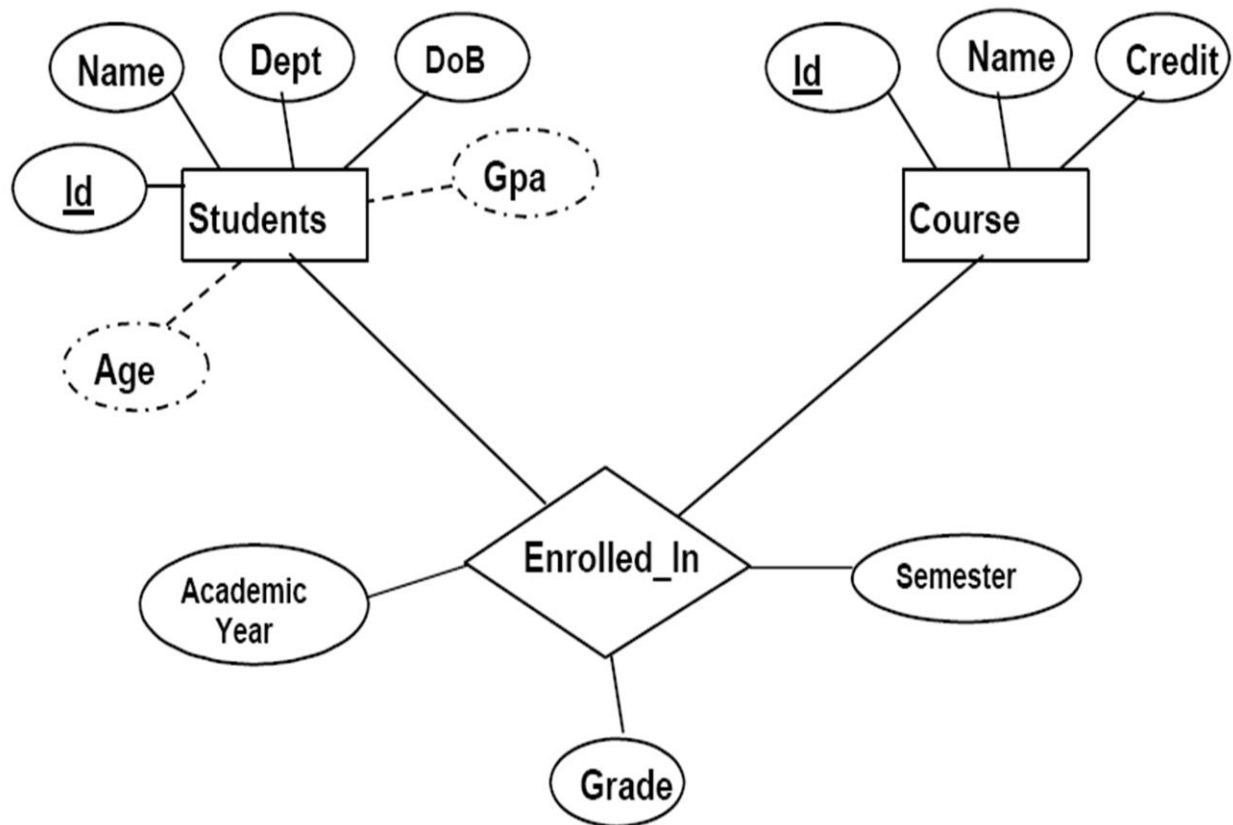
- **PRIMARY KEYS** are underlined.



- **Relationships** are represented by **DIAMOND** shaped symbols.
 - **Weak Relationship** is a relationship between *Weak* and *Strong Entities*.
 - **Strong Relationship** is a relationship between *two strong Entities*.



- **Example 1:** Build an **ER Diagram** for the following information:
 - A **student record management system** will have the following two basic data object categories with their own features or properties:
 - Students will have an Id, Name, Dept, Age, GPA and Course will have an Id, Name, Credit Hours\2019SZA\M2.....4.
 - Whenever a student enrol in a course in a specific Academic Year and Semester, the Student will have a grade for the course.



- **Exercise:** Build an **ER Diagram** for the following information:

✚ A **Personnel record management system** will have the following two basic data object categories with their own features or properties:

✚ **Employee** will have an Id, Name, DoB, Age, Tel and

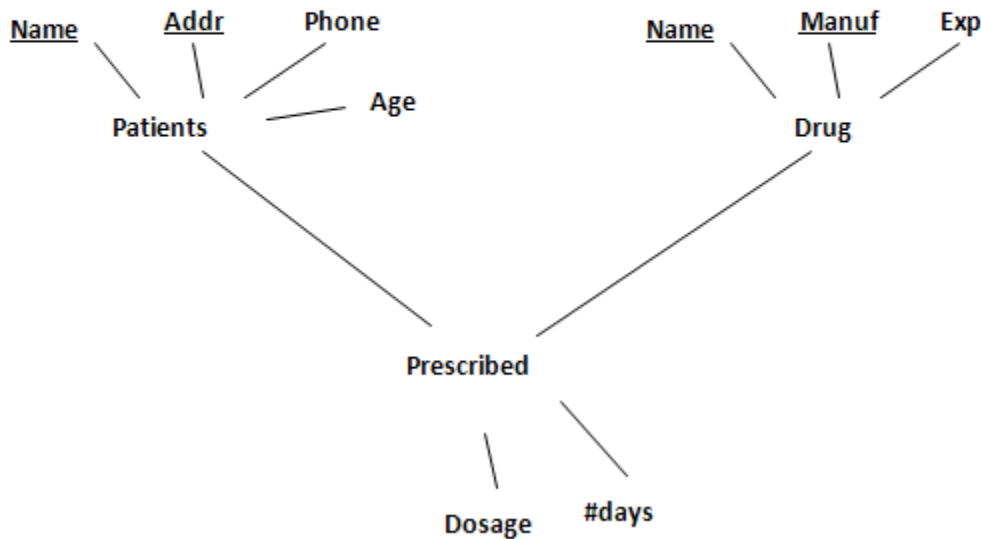
✚ **Department** will have an Id, Name, Location

Whenever an Employee is assigned in one Department, the duration of his stay in the respective department should be registered.

Example 2

- Build an ER Diagram for the following information:
 - ✓ Patients
 - Name, Address, Phone #, Age
 - ✓ Drugs
 - Name, Manufacturer , Expiration Date
 - ✓ Patients are prescribed drugs
 - Dosage, # Days

Example 2 Answer

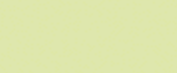


✚ Structural Constraints on Relationship

➤ *Constraints on Relationship /Multiplicity/ Cardinality Ratio Constraints.*

- ✚ Specifies the number of relationships instances an entity can participate in.
- ✚ Specifies lower and upper bounds on the number of relationships each entity can participate in.
- ✚ Displayed on the diamonds.

Symbols

	Chen	Crow's Foot	Rein85	IDEF1X
Entity				
Relationship line				
Relationship				
Option symbol				
One (1) symbol	1			
Many (M) symbol	M			
Composite entity				
Weak entity				

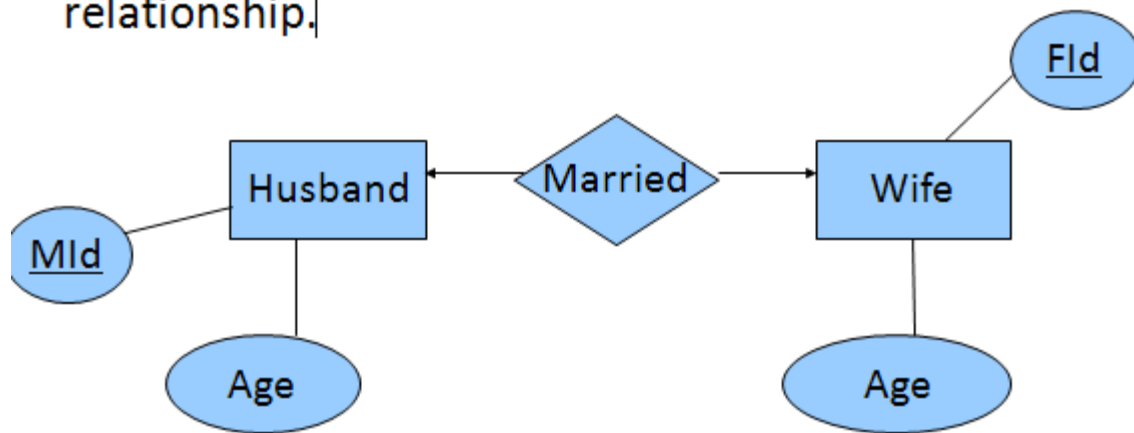
Relationship classifications

- ✓ **1:1** One-to-one relationship
- ✓ **1:M** One-to-many relationship
- ✓ **M:N** many-to-many relationship

✚ **One-to-one relationship:**

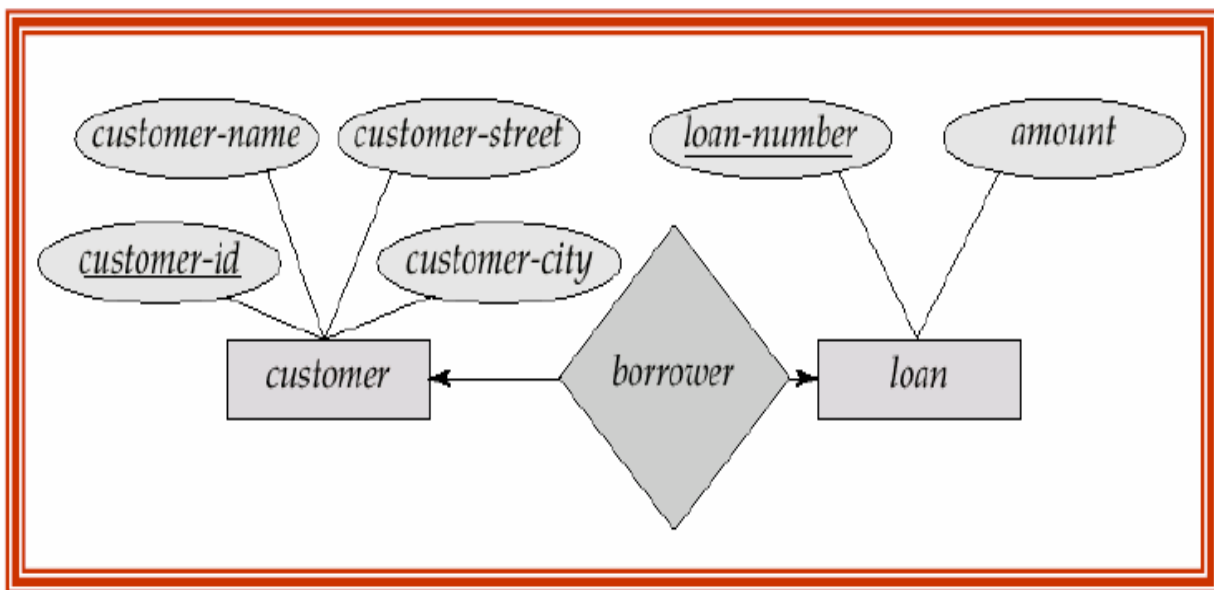
- ✓ If the relationship between A and B satisfies the one-to-one mapping constraint from A to B, then:
 - ✓ an entity in A is related to at most one entity in B,
 - ✓ an entity in B is related to at most one entity in A.
- ✓ For a one-to-one relationship between two entity sets E and F , $key(E)$ and $key(F)$ are both keys for the relationship set.

Example1:- Husband-wife (marriage) is a **one-to-one** relationship.



Example 2: - A **customer** is associated with at most one loan via the **relationship borrower**.

– A **loan** is associated with at most one customer via **borrower**.

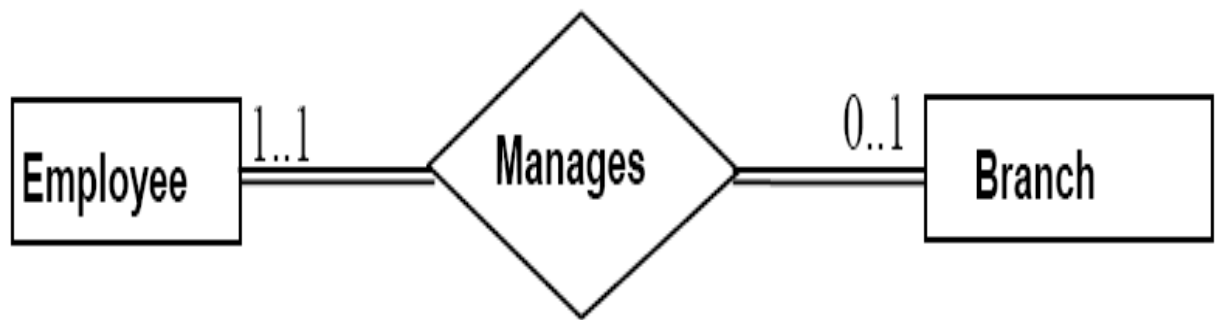


E.g.: Relationship **Manages** between **Staff** and **branch**.

- The **multiplicity** of the relationship is:

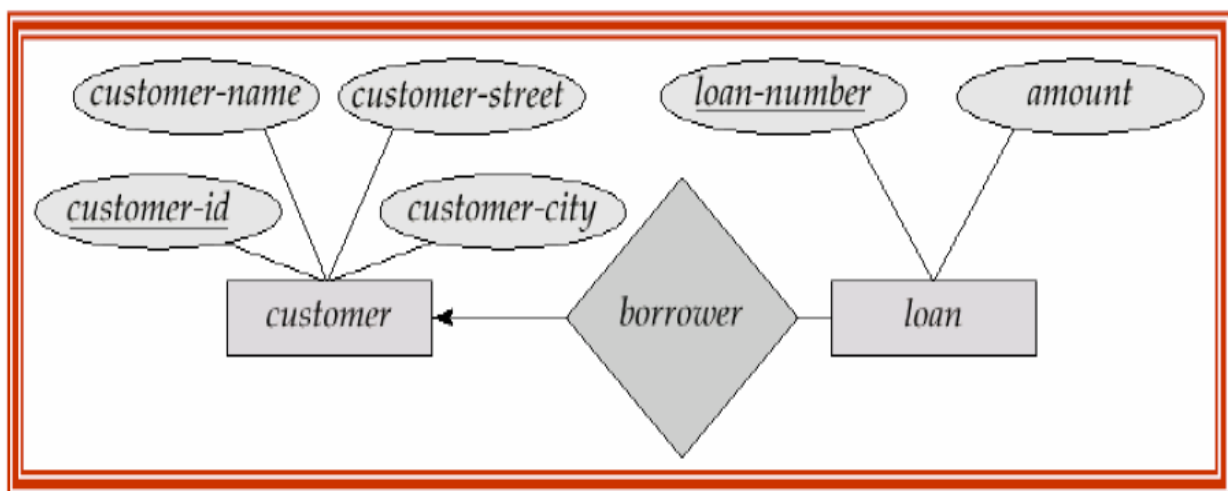
– One **branch** can only have one **manager**

One **employee** could **manage** either one or no **branches**



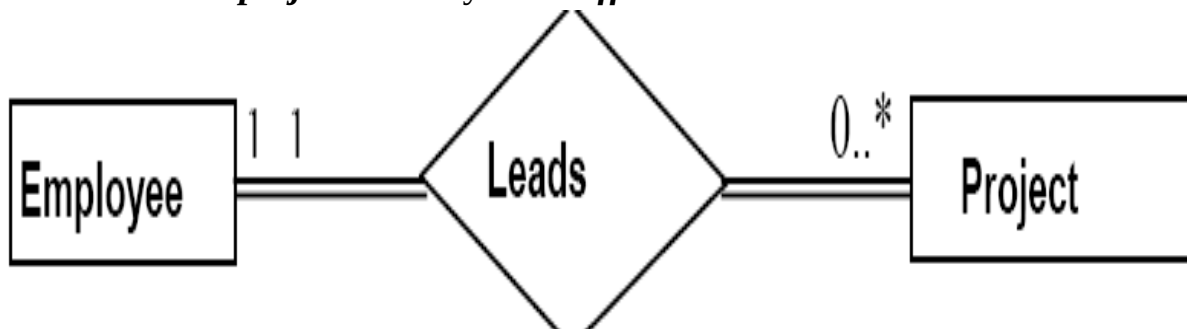
✚ One-To-Many Relationships

- In the one-to-many relationship: -
 - ✓ a **loan** is associated with at most **one customer** via **borrower**.
 - ✓ a **customer** is associated with **several (including 0)** loans via **borrower**.



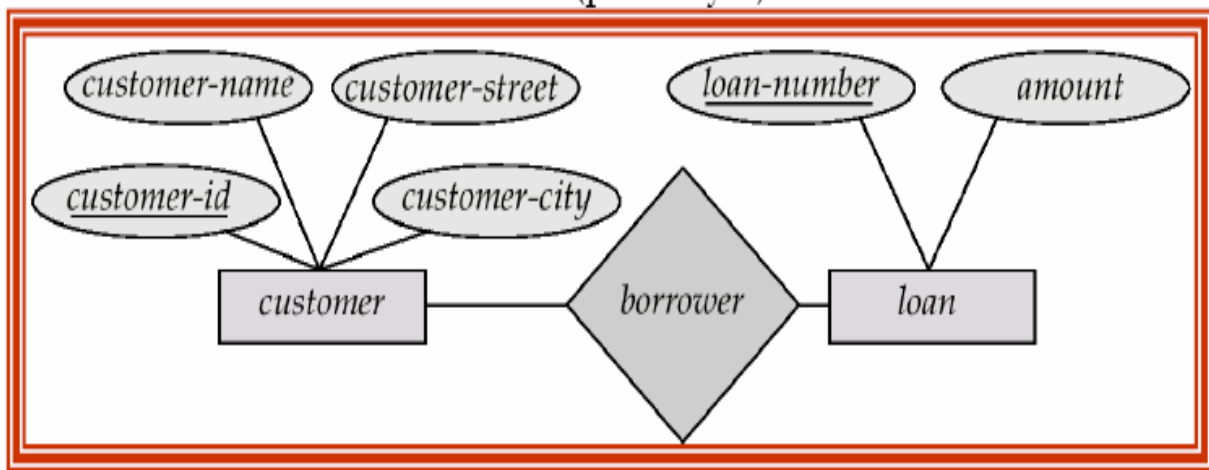
✚ E.g.: Relationship **Leads** between **staff** and **project**.

- ✚ The **multiplicity** of the relationship: -
 - One **staff** may **Lead** zero or more **project(s)**.
 - One **project** is led by one **staff**.

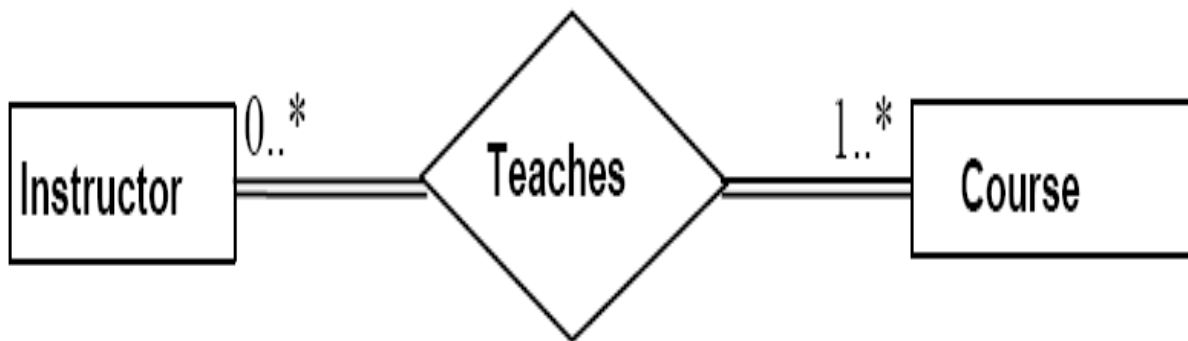


➤ Many-To-Many Relationship

- ✚ A **customer** is associated with several (possibly 0) **loans** via **borrower**.
- ✚ A **loan** is associated with several (possibly 0) **customers** via **borrower**.



- ✓ E.g.: Relationship **Teaches** between *instructor* and *course*.
- ✓ The **multiplicity** of the relationship: -
 - ✚ One **Instructor** Teaches One or more **Course(s)**.
 - ✚ One **course** is taught by Zero or more **Instructor(s)**.



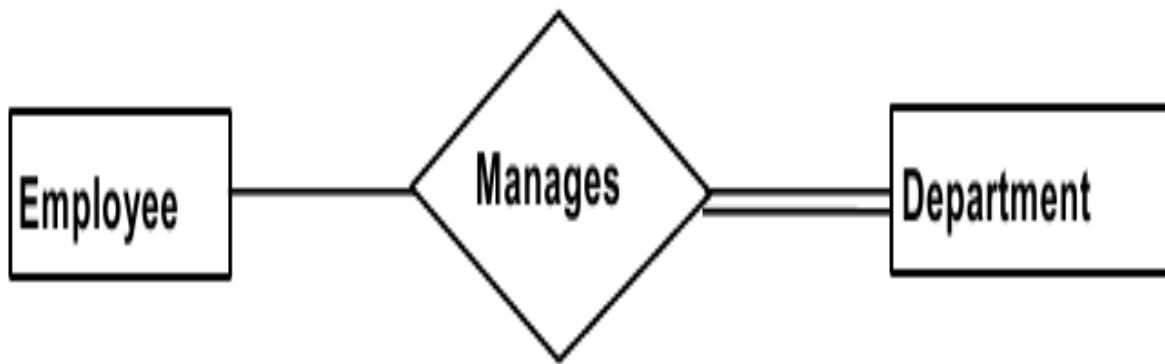
✚ Participation constraint of a relationship

- ✚ Specifies whether the existence of an entity depends on its being related to another entity via the relationship type.
- ✚ There are two distinct participation constraints with this respect, namely: -
 - ✚ **Total Participation and**
 - ✚ **Partial Participation**

Total participation constraints require the participation of every entity in the relationship.

- ✓ It is displayed by a double line.
- ✓ **Partial** participation constraints (displayed by a single line).
- ✓ Participation of EMPLOYEE in “manages” relationship with DEPARTMENT, is **partial participation** since not all employees are managers.
- ✓ Participation of DEPARTMENT in “Manages” relationship with

EMPLOYEE is **total** since every department should have a manager.



MODELING WITH THE RELATIONAL DATA MODEL & NORMALIZATION

Logical Database Design

- ✚ Is the process of constructing a model of the information used in an enterprise based on a specific data model.
- ✚ (e.g. relational, hierarchical or network or object),
- ✚ but independent of a particular DBMS and other physical considerations.
- ✚ Normalization process:
 - ✓ Collection of Rules to be maintained
 - ✓ Discover new entities in the process
 - ✓ Revise attributes based on the rules and the discovered Entities.

Converting ER Diagram to Relational Tables

- ✚ Three basic rules to convert **ER** into **tables** or **relations**: -
- ✚ **Rule 1: Entity Names** will automatically be **table names**.
- ✚ **Rule 2: Mapping of attributes:** **attributes** will be **columns** of the respective tables.
 - ✚ **Atomic or single**-valued or derived or stored attributes will be columns.
 - ✚ **Composite attributes:** the parent attribute will be ignored and the decomposed attributes (child attributes) will be columns of the table.
 - ✚ **Multi-valued attributes:** will be mapped to a new table where the primary key of the main table will be posted for cross referencing.
- ✚ **Rule 3: Relationships:** **relationship** will be mapped by using a **foreign key** attribute.
- ✚ **Foreign key** is a **primary** or **candidate key** of one relation used to create association between tables.
- **For a relationship with One-to-One Cardinality:** post the **primary** or **candidate key** of one of the tables into the other as a **foreign key**.
- ✓ In cases where one entity is having **partial participation** on the relationship, it is recommended to post the candidate key of the partial participants to the total participant so as to save some memory location due to null values on the foreign key attribute.
- ✓ E.g.: for a relationship between **Employee** and **Department** where employee manages a department, the cardinality is one-to-one.
- ✓ Here the **PK** of the **Employee** can be posted to the **Department** or
- ✓ the **PK** of the **Department** can be posted to the **Employee**.
- ✓ But the **Employee** is having partial participation on the relationship

"**Manages**" as not all employees are managers of departments.

- ✓ it is recommended to post the primary key of the employee to the Department table as a foreign key.

✚ **For a relationship with One-to-Many Cardinality**:- Post the primary key or candidate key from the “one” side as a foreign key attribute to the “many” side.

- ✓ E.g.: For a relationship called “**Belongs To**” between
- ✓ **Employee** (Many) and **Department** (One) the primary or candidate key of the one side which is Department should be posted to the many side which is Employee table.

✚ **For a relationship with Many-to-Many Cardinality**: create a new table (which is the associative entity). and

- ✓ post primary key or candidate key from the participant entities as foreign key attributes in the new table along with some additional attributes (if applicable).

The same approach should be used for relationships with degree greater than binary.

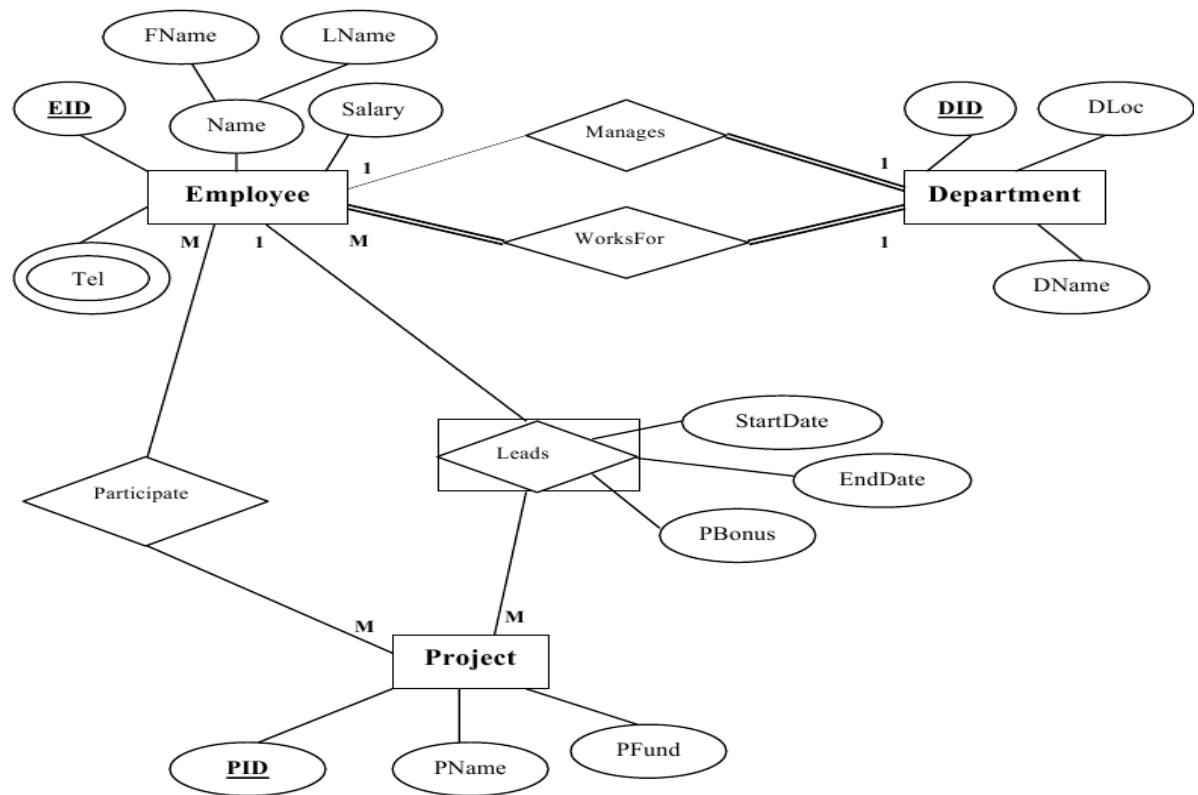
✚ **For a relationship having Associative Entity property**: in cases where the relationship has its own attributes (associative entity),

✚ create a new table for the associative entity and post primary key or candidate key from the participating entities as foreign key attributes in the new table.

- ✓ **Example**: to illustrate the major rules in mapping **ER** to **relational schema**:
- ✓ **Employee**, **Department** and **Project** information.
- ✓ **Employee**: Eid, Name, Salary, Tel.
- ✓ **Department**: Did, Dname, Dloc.
- ✓ **Project**: Pid, Pname, PFund

Employee works for department.

- ✓ where an **employee** might be *assigned to manage* a **department**.
- ✓ **Employee** might *participate* on different **projects** within the organization.
 - ✓ An **employee** might as well be *assigned to lead* a **project** where the *starting* and *ending date* of his/her project leadership and *bonus* will be registered.



- ✓ After we have drawn the **ER diagram**, the next step is to map the **ER** into **relational schema** so as the rules of the relational data model can be tested for each relational schema.
 - ✓ The mapping can be done for the **entities** followed by **relationships** based on the rule of mapping.
 - ✓ The mapping has been done as follows.
 - ➔ **Mapping EMPLOYEE Entity:**
 - ✓ There will be **Employee** table with **EID**, **Salary**, **FName** and **LName** being the columns.
 - ✓ The composite attribute **Name** will be ignored as its decomposed attributes (**FName** and **LName**) are columns in the Employee Table.
- The **Tel** attribute will be a new table as it is **multi-valued**.

Employee

<u>EID</u>	FName	LName	Salary
------------	-------	-------	--------

Telephone

<u>EID</u>	<u>Tel</u>
------------	------------

→ Mapping DEPARTMENT Entity:

- There will be *Department* table with **DID**, DName, and DLoc being the columns.

Department

<u>DID</u>	DName	DLoc
------------	-------	------

→ Mapping PROJECT Entity:

- There will be *Project* table with **PID**, PName, and PFund being the columns.

Project

<u>PID</u>	PName	PFund
------------	-------	-------

→ Mapping the MANAGES Relationship:

- As the relationship is having one-to-one cardinality, the PK or CK of one of the table can be posted into the other.
- But based on the recommendation, the Pk or CK of the partial participant (**Employee**) should be posted to the total participants (**Department**).
- This will require adding the PK of Employee (EID) in the Department Table as a foreign key.
- We can give the foreign key another name which is MEID to mean "managers employee id".
- this will affect the degree of the Department table.

Department

<u>DID</u>	DName	DLoc	MEID
------------	-------	------	------

✚ Mapping the PARTICIPATES Relationship:

- ✓ As the relationship is having many-to-many cardinality, we need to create a new table and post the PK or CK of the **Employee** and **Project** table into the new table.
- ✓ We can give a descriptive new name for the new table like
- ✓ **Emp_Partc_Project** to mean "Employee participate in a project".

Emp_Partc_Project	
<u>EID</u>	<u>PID</u>

✚ Mapping the LEADS Relationship:

- ✓ As the relationship is **associative entity**, we are supposed to create a table for the associative entity where the PK of **Employee** and **Project** tables will be posted in the new table as a foreign key.
- ✓ The new table will have the attributes of the associative entity as columns.
- ✓ We can give a descriptive new name for the new table like **Emp_Lead_Project** to mean "Employee participate in a project".

Emp_Lead_Project

<u>EID</u>	<u>PID</u>	PBonus	StartDate	EndDate
------------	------------	--------	-----------	---------

- At the end of the mapping, we will have the following relational schema (tables) for the logical design phase.

Project

<u>PID</u>	PName	PFund
------------	-------	-------

Telephone

<u>EID</u>	<u>Tel</u>
------------	------------

Department

<u>DID</u>	DName	DLoc	MEID
------------	-------	------	------

Employee

<u>EID</u>	FName	LName	Salary	EDID
------------	-------	-------	--------	------

Emp_Partc_Project

<u>EID</u>	<u>PID</u>
------------	------------

Emp_Lead_Project

<u>EID</u>	<u>PID</u>	PBonus	StartDate	EndDate
------------	------------	--------	-----------	---------

Normalization

- ✓ **Database normalization** is a series of steps followed to obtain a database design that allows: -
 - ✓ for consistent data storage and efficient access of data in a relational database.
- These steps reduce: -
 - ✓ data redundancy and
 - ✓ the risk of data becoming inconsistent.
-  **NORMALIZATION** is the process of identifying the logical associations between data items and designing a database that will represent such associations but without suffering the **update anomalies** which are;

1. Insertion Anomalies
2. Deletion Anomalies
3. Modification Anomalies

-  **Normalization** may reduce system performance since data will be cross referenced from many tables.
-  Thus, **denormalization** is sometimes used to improve performance, at the cost of reduced consistency guarantees.
-  All the normalization rules will eventually remove the **update anomalies** that may exist during data manipulation after the implementation.
-  The **update anomalies** are;

- ✚ The type of problems that could occur in insufficiently normalized table is called **update anomalies** which includes;

➤ **Insertion anomalies**

- An "insertion anomaly" is a failure to place information about a new database entry into all the places in the database.
- In a properly normalized database, information about a new entry needs to be inserted into only one place in the database;

➤ **Deletion anomalies**

- ✚ A "deletion anomaly" is a failure to remove information about an existing database entry when it is time to remove that entry.

In a properly normalized database, information about an old, to-be-gotten-rid-of entry needs to be deleted from only one place in the database;

➤ **Modification anomalies**

- ✚ A modification of a database involves changing some value of the attribute of a table.

- In a properly normalized database table, what ever information is modified by the user, the change will be effected and used accordingly.

✚ ***Thus, the purpose of normalization is to reduce the chances for anomalies to occur in a database.***

- Example of problems related with anomalies.

EmpID	FName	LName	SkillID	Skill	SkillType	School	SchoolAdd	Skill Level
12	Abebe	Mekuria	2	SQL	Database	AAU	Sidist_Kilo	5
16	Lemma	Alemu	5	C++	Programming	Unity	Gerji	6
28	Chane	Kebede	2	SQL	Database	AAU	Sidist_Kilo	10
25	Abera	Taye	6	VB6	Programming	Helico	Piazza	8
65	Almaz	Belay	2	SQL	Database	Helico	Piazza	9
24	Dereje	Tamiru	8	Oracle	Database	Unity	Gerji	5
51	Selam	Belay	4	Prolog	Programming	Jimma	Jimma City	8
94	Alem	Kebede	3	Cisco	Networking	AAU	Sidist_Kilo	7
18	Girma	Dereje	1	IP	Programming	Jimma	Jimma City	4
13	Yared	Gizaw	7	Java	Programming	AAU	Sidist_Kilo	6

+ Deletion Anomalies:

- ✓ If employee with **ID 16** is deleted then information about skill **C++** and **the type of skill** is deleted from the database.
- ✓ Then we will not have any information about **C++** and its **skill type**.

+ Insertion Anomalies:

- ✓ What if we have a new employee with a skill called **Pascal**? We cannot decide whether **Pascal** is allowed as a value for **skill** and
- ✓ we have no clue about the **type of skill** that **Pascal** should be categorized as.

+ Modification Anomalies:

- ✓ What if the address for **Helico** is changed from **Piazza** to **Mexico**?
- ✓ We need to look for every occurrence of **Helico** and change the value of **School_Add** from **Piazza** to **Mexico**, which is prone to error.

+ Functional Dependency (FD)

- ✓ Two data items **A** and **B** are said to be in dependent relationship if certain values of data item **B** always appears with certain values of data item **A**.
 - ✓ The notation is: $A \rightarrow B$ which is read as; **B** is functionally dependent on **A**.
- $A \rightarrow B$ holds if whenever a given tuple has the same value for **A**, it must have the same value for **B**.

Example

<i>Dinner Course</i>	<i>Type of Wine</i>
Meat	Red
Fish	White
Cheese	Rose

- ✓ Since the **type of Wine** served depends on the type of **Dinner**, we say **Wine** is functionally dependent on **Dinner**.

Dinner $\rightarrow$ Wine

+ Partial Dependency

- ✓ If an attribute which is not a member of the primary key is dependent on some part of the primary key (if we have composite primary key) then that attribute is partially functionally dependent on the primary key.
- ✓ Example:
- ✓ Let $\{A,B\}$ is the Primary Key and **C** is no key attribute.

- ✓ Then if $\{A,B\} \rightarrow C$ and $B \twoheadrightarrow C$
- ✓ Then C is partially functionally dependent on $\{A,B\}$

✚ Full Dependency

- ✓ If an attribute which is not a member of the primary key is not dependent on some part of the primary key but the whole key (if we have composite primary key) then that attribute is fully functionally dependent on the primary key.
- ✓ Let $\{A,B\}$ is the Primary Key and C is no key attribute
- ✓ Then if $\{A,B\} \rightarrow C$ and $B \rightarrow C$ and $A \rightarrow C$ does hold
- ✓ Then C Fully functionally dependent on $\{A,B\}$.

✚ Transitive Dependency

- ✓ In mathematics and logic, a transitive relationship is a relationship of the following form: "If A implies B, and if B implies C, then A implies C."
- ✓ Example:
- ✓ If *Mr X* is a **Human**, and if *every Human* is an **Animal**, then *Mr X* must be an **Animal**.
- ✓ Generalized way of describing transitive dependency is that:

If A functionally governs B, AND
If B functionally governs C

$$\{(A \rightarrow B) \text{ AND } (B \rightarrow C)\} \implies A \rightarrow C \text{ provided that } B \not\rightarrow A \text{ and } C \not\rightarrow A$$

✚ Steps in Normalization

- ✓ We have various levels or steps in normalization called **Normal Forms**.
- ✓ Normalization towards a logical design consists of the following steps:
- ✓ **Unnormalized Form:**

- Identify all data elements

✚ First Normal Form:

- Find the **key** with which you can find **all** data.

✚ Second Normal Form:

- ✓ Remove part-key dependencies. Make all data dependent **on the whole key**.

✚ Third Normal Form

- Remove non-key dependencies. Make all data dependent on **nothing but the key**.

First Normal Form (1NF)

- ✓ Requires that all column values in a table are **atomic**.
(e.g., a number is an atomic value, while a list or a set is not).
- ✓ We have two ways of achieving this: -
 1. Putting each repeating group into a separate table and connecting them with a **primary key-foreign key relationship**.
 2. Moving this repeating groups to a new row by repeating the common attributes.
 - If so then Find the key with which you can find all data.

Definition

- ➔ A table (relation) is in **1NF** If: -
- ✓ There are no duplicated rows in the table. Unique identifier
 - ✓ Each cell is single-valued (i.e., there are no repeating groups).
 - ✓ Entries in a column (attribute, field) are of the same kind.

Example for First Normal form (1NF)

UNNORMALIZED

<i>EmpID</i>	<i>FirstName</i>	<i>LastName</i>	<i>Skill</i>	<i>SkillType</i>	<i>School</i>	<i>SchoolAdd</i>	<i>SkillLevel</i>
12	Abebe	Mekuria	SQL, VB6	Database, Programming	AAU, Helico	Sidist_Kilo Piazza	5 8
16	Lemma	Alemu	C++ IP	Programming Programming	Unity Jimma	Gerji Jimma City	6 4
28	Chane	Kebede	SQL	Database	AAU	Sidist_Kilo	10
65	Almaz	Belay	SQL Prolog Java	Database Programming Programming	Helico Jimma AAU	Piazza Jimma City Sidist_Kilo	9 8 6
24	Dereje	Tamiru	Oracle	Database	Unity	Gerji	5
94	Alem	Kebede	Cisco	Networking	AAU	Sidist_Kilo	7

FIRST NORMAL FORM (1NF)

- ✓ Remove all repeating groups.
- ✓ Distribute the **multi-valued** attributes into different rows and
- ✓ identify a unique identifier for the relation so that is can be said is a relation in relational database.

<u>EmpID</u>	<u>FirstName</u>	<u>LastName</u>	<u>SkillID</u>	<u>Skill</u>	<u>SkillType</u>	<u>School</u>	<u>SchoolAdd</u>	<u>SkillLevel</u>
12	Abebe	Mekuria	1	SQL	Database	AAU	Sidist_Kilo	5
12	Abebe	Mekuria	3	VB6	Programming	Helico	Piazza	8
16	Lemma	Alemu	2	C++	Programming	Unity	Gerji	6
16	Lemma	Alemu	7	IP	Programming	Jimma	Jimma City	4
28	Chane	Kebede	1	SQL	Database	AAU	Sidist_Kilo	10
65	Almaz	Belay	1	SQL	Database	Helico	Piazza	9
65	Almaz	Belay	5	Prolog	Programming	Jimma	Jimma City	8
65	Almaz	Belay	8	Java	Programming	AAU	Sidist_Kilo	6
24	Dereje	Tamiru	4	Oracle	Database	Unity	Gerji	5
94	Alem	Kebede	6	Cisco	Networking	AAU	Sidist_Kilo	7

✚ **Second Normal form 2NF**

- ✓ No partial dependency of a non-key attribute on part of the primary key.
- ✓ This will result in a set of relations with a level of **Second Normal Form**.
- ✓ Any table that is in **1NF** and has a single-attribute (i.e., a non-composite) key is automatically also in **2NF**.
- ✓ **Definition:** a table (relation) is in **2NF** If: -
- ✓ It is in **1NF** and
- ✓ If all **non-key** attributes are dependent on the **entire primary key**. i.e. no partial dependency.

Example for 2NF:

EMP_PROJ

<u>EmpID</u>	<u>EmpName</u>	<u>ProjNo</u>	<u>ProjName</u>	<u>ProjLoc</u>	<u>ProjFund</u>	<u>ProjMangID</u>	<u>Incentive</u>
--------------	----------------	---------------	-----------------	----------------	-----------------	-------------------	------------------

EMP_PROJ rearranged

<u>EmpID</u>	<u>ProjNo</u>	<u>EmpName</u>	<u>ProjName</u>	<u>ProjLoc</u>	<u>ProjFund</u>	<u>ProjMangID</u>	<u>Incentive</u>
--------------	---------------	----------------	-----------------	----------------	-----------------	-------------------	------------------

- ✓ **Business rule:** Whenever an **employee** participates in a **project**, he/she will be entitled for an **incentive**.
- ✓ This schema is in its **1NF** since we don't have any repeating groups or attributes with **multi-valued** property.
- ✓ To convert it to a **2NF** we need to remove all partial dependencies of non-key attributes on part of the primary key.
- ✓ **{EmpID, ProjNo} --> EmpName, ProjName, ProjLoc, ProjFund, ProjMangID, Incentive.**
- ✓ But in addition to this we have the following dependencies:

FD1: {EmpID}--> EmpName

FD2: {ProjNo}--> ProjName, ProjLoc, ProjFund, ProjMangID

FD3: {EmpID, ProjNo} --> Incentive

✓ some non-key attributes are partially dependent on some part of the primary key.

✓ This can be witnessed by analyzing the first two functional dependencies (FD1 and FD2).

- ✓ Thus, each Functional Dependencies, with their dependent attributes should be moved to a new relation where the Determinant will be the Primary Key for each.

EMPLOYEE

<u>EmpID</u>	EmpName
--------------	---------

PROJECT

<u>ProjNo</u>	ProjName	ProjLoc	ProjFund	ProjMangID
---------------	----------	---------	----------	------------

EMP_PROJ

<u>EmpID</u>	<u>ProjNo</u>	Incentive
--------------	---------------	-----------

✚ Third Normal Form (3NF)

- ✓ Eliminate Columns Dependent on another non-Primary Key.
- ✓ If attributes do not contribute to a description of the key, remove them to a separate table.
- ✓ This level avoids update and delete anomalies.
- ✓ **Definition:**
- ✓ a Table (Relation) is in **3NF** If :-
- ✓ It is in **2NF** and
- ✓ There are no transitive dependencies between a primary key and

non-primary key attributes.

Example for (3NF)

- ✓ **Assumption:** Students of same batch (same year) live in one building or dormitory.

STUDENT

<u>StudID</u>	Stud_F_Name	Stud_L_Name	Dept	Year	Dormitory
125/97	Abebe	Mekuria	Info Sc	1	401
654/95	Lemma	Alemu	Geog	3	403
842/95	Chane	Kebede	CompSc	3	403
165/97	Alem	Kebede	InfoSc	1	401
985/95	Almaz	Belay	Geog	3	403

This schema is in its **2NF** since the primary key is a single attribute.

- ✓ Let's take **StudID**, **Year** and **Dormitory** and see the dependencies.
- ✓ **StudID --> Year AND Year → Dormitory**
- ✓ And **Year** cannot determine **StudID** and
- ✓ **Dormitory** cannot determine **StudID** Then transitively **StudID--> Dormitory**
- ✓ To convert it to a **3NF** we need to remove all transitive dependencies of non-key attributes on another non-key attribute.
- ✓ The non-primary key attributes, dependent on each other will be moved to another table and linked with the main table using Candidate Key- Foreign Key relationship.

STUDENT

<u>StudID</u>	Stud F_Name	Stud L_Name	Dept	Year
125/97	Abebe	Mekuria	Info Sc	1
654/95	Lemma	Alemu	Geog	3
842/95	Chane	Kebede	CompSc	3
165/97	Alem	Kebede	InfoSc	1
985/95	Almaz	Belay	Geog	3

DORM

<u>Year</u>	Dormitory
1	401
3	403

- ✓ Generally, even though there are other four additional levels of Normalization, a table is said to be **normalized** if it reaches **3NF**.
- ✓ A database with all tables in **3NF** is said to be **Normalized Database**.
- ✓ Mnemonic for remembering the rationale for **normalization** up to **3NF** could be the following: -
 - ✚ **No Repeating or Redundancy**: no repeating fields in the table.
 - ✚ **The Fields Depend Upon the Key**: the table should solely depend on the key.
 - ✚ **The Whole Key**: no partial key dependency.
 - ✚ **And Nothing But The Key**: no inter data dependency.

