

# CS4.301 Data & Applications

Ponnurangam Kumaraguru ("PK")  
#ProfGiri @ IIIT Hyderabad



pk.profgiri



/in/ponguru



@ponguru



Ponnurangam.kumaraguru

# Actors on the Scene: Day-to-Day use of DB

End Users: Casual, naïve, sophisticated, stand-alone users

sophisticated: thoroughly familiarize themselves with all facilities of DBMS, implement their own, complex requirements

stand-alone: maintain personal DB using ready-made programs; TALLY

System analysts & application programmers

determine the requirements of end-users, including naïve, develop specifications for canned transactions

ap implement above specifications as programs, they test – debug – maintain these canned transactions

software developers / engineers play these roles sometimes

# Workers Behind the Scene: Maintain the DB

## DBMS designers & implementers

design and implement the DBMS modules; complex modules like query language processing, interface processing, controlling concurrency, handling data recovery & security

## Tool developers

design & implement tools; optional packages that are often purchased separately; facilitate DB modeling & design, system design, and improved performance

## Operators & maintenance personnel

responsible for running & maintenance of the hardware & software environment for DB

# Advantages of using DBMS approach

## Controlling redundancy

- redundancy in storing the same data multiple times

- e.g. student details in university maintained by acad & finance office separately

- duplication of efforts, storage space, inconsistent data [Jan-19-1998 vs Jan-29-1998]

- ideally student details in only one place, *data normalization*

- keeping all needed data together, *denormalization*

# Advantages of using DBMS approach

Restricting unauthorized access

your grades accessible to only some; my salary and personal details only to some [hopefully 😊]

Providing storage structures and search techniques for efficient query processing

efficiently executing queries & updates; creating *indexes* and maintaining it; *buffering* & *caching* modules

# Advantages of using DBMS approach

## Providing backup & recovery

provide facilities for recovering from hardware & software failures  
complex updates, should not crash; if crash what state to recover

## Providing multiple user interfaces

apps for mobile users; query language for causal; programming  
language for application programmers; forms / command codes  
for parametric; menu & natural language interfaces for  
standalone

# Advantages of using DBMS approach

## Representing Complex relationship among data

DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, and to retrieve / update related data easily & efficiently

## Enforcing integrity constraints

student name: 30 alphabetic characters; record in one file must be related to records in other files [e.g. every SECTION record must be related to a COURSE record] *referential integrity*

uniqueness on data item values [e.g. every COURSE record must have a unique value for COURSE\_NUMBER] *key or uniqueness constraint*

# Advantages of using DBMS approach

Permitting inferencing and actions using rules and triggers

triggers associated with tables; trigger is a rule activated by updates to the table results in performing some addition operations to other tables, sending messages, etc.

stored procedures are invoked appropriately when some conditions are met



# When not to use a DBMS

## Main inhibitors (costs) of using a DBMS:

- High initial investment and possible need for additional hardware.

- Overhead for providing generality, security, concurrency control, recovery, and integrity functions.

## When a DBMS may be unnecessary:

- If the database and applications are simple, well defined, and not expected to change.

- If access to data by multiple users is not required.

## When a DBMS may be infeasible:

- In embedded systems where a general purpose DBMS may not fit in available storage

# Data Models

## 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

# Schemas versus Instances

## Database Schema:

The ***description*** of a 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*

# Example of a Database Schema

## STUDENT

|      |                |       |       |
|------|----------------|-------|-------|
| Name | Student_number | Class | Major |
|------|----------------|-------|-------|

## COURSE

|             |               |              |            |
|-------------|---------------|--------------|------------|
| Course_name | Course_number | Credit_hours | Department |
|-------------|---------------|--------------|------------|

## PREREQUISITE

|               |                     |
|---------------|---------------------|
| Course_number | Prerequisite_number |
|---------------|---------------------|

## SECTION

|                    |               |          |      |            |
|--------------------|---------------|----------|------|------------|
| Section_identifier | Course_number | Semester | Year | Instructor |
|--------------------|---------------|----------|------|------------|

## GRADE\_REPORT

|                |                    |       |
|----------------|--------------------|-------|
| Student_number | Section_identifier | Grade |
|----------------|--------------------|-------|

**Figure 2.1**

Schema diagram for the database in Figure 1.2.

**COURSE**

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

**SECTION**

| Section_identifier | Course_number | Semester | Year | Instructor |
|--------------------|---------------|----------|------|------------|
| 85                 | MATH2410      | Fall     | 04   | King       |
| 92                 | CS1310        | Fall     | 04   | Anderson   |
| 102                | CS3320        | Spring   | 05   | Knuth      |
| 112                | MATH2410      | Fall     | 05   | Chang      |
| 119                | CS1310        | Fall     | 05   | Anderson   |
| 135                | CS3380        | Fall     | 05   | Stone      |

**GRADE\_REPORT**

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

**PREREQUISITE**

| Course_number | Prerequisite_number |
|---------------|---------------------|
| CS3380        | CS3320              |
| CS3380        | MATH2410            |
| CS3320        | CS1310              |

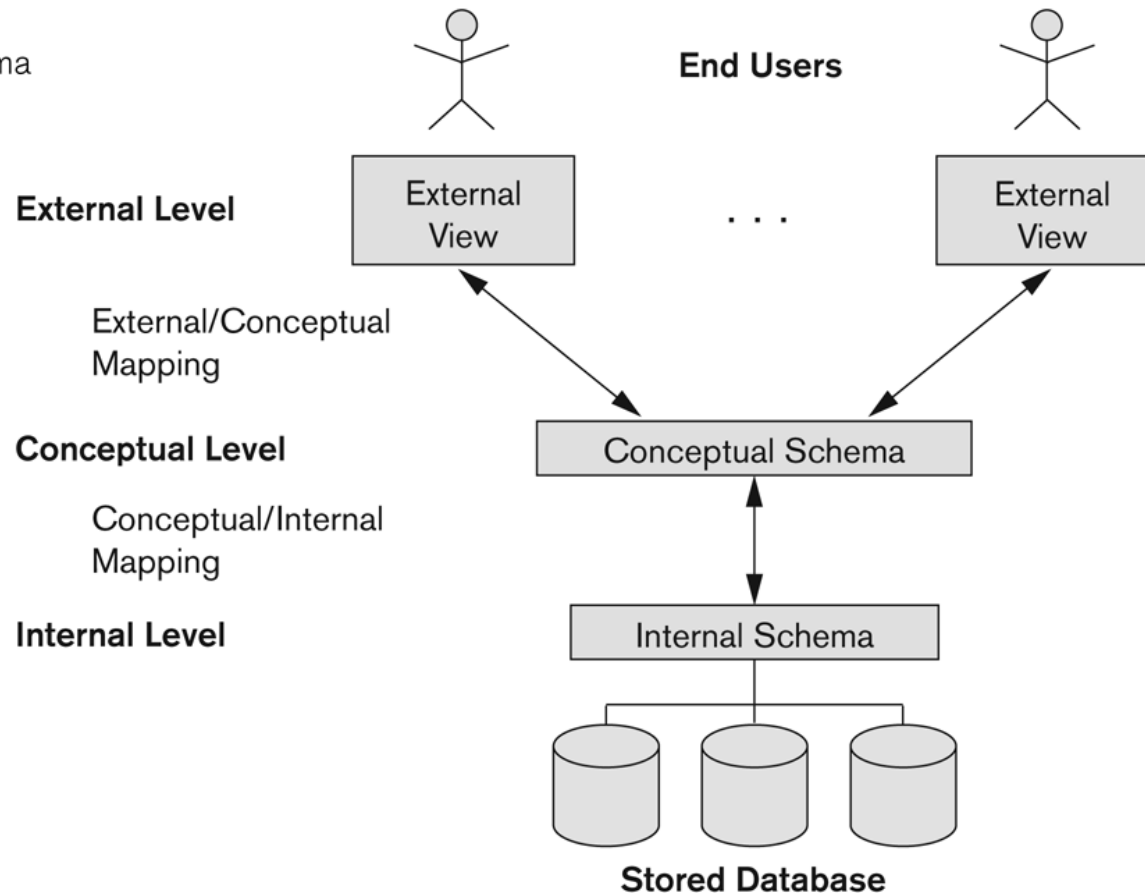
Example of a  
database state

**Figure 1.2**

A database that stores  
student and course  
information.

**Figure 2.2**

The three-schema architecture.



The three-sche  
architecture

# DBMS Languages

Data Definition Language (DDL)

Data Manipulation Language (DML)



# DBMS Languages

## **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

# DBMS Languages

## **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*).

This lecture

# DBMS Interfaces

## 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.

## Mobile Interfaces: interfaces allowing users to perform transactions using mobile apps

# DBMS Programming Language Interfaces

Programmer interfaces for embedding DML in a programming languages:

**Embedded Approach:** e.g. embedded SQL (for C, C++, etc.), SQLJ (for Java)

**Procedure Call Approach:** e.g. JDBC for Java, ODBC (Open Database Connectivity) for other programming languages as API's (application programming interfaces)

**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

**Scripting Languages:** PHP (client-side scripting) and Python (server-side scripting) are used to write database programs.

# User-Friendly DBMS Interfaces

Menu-based (Web-based), popular for browsing on the web

Forms-based, designed for naïve users used to filling in entries on a form

Graphics-based

- Point and Click, Drag and Drop, etc.

- Specifying a query on a schema diagram

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

Natural language: free text as a query

Speech: Input query and Output response

Web Browser with keyword search

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.

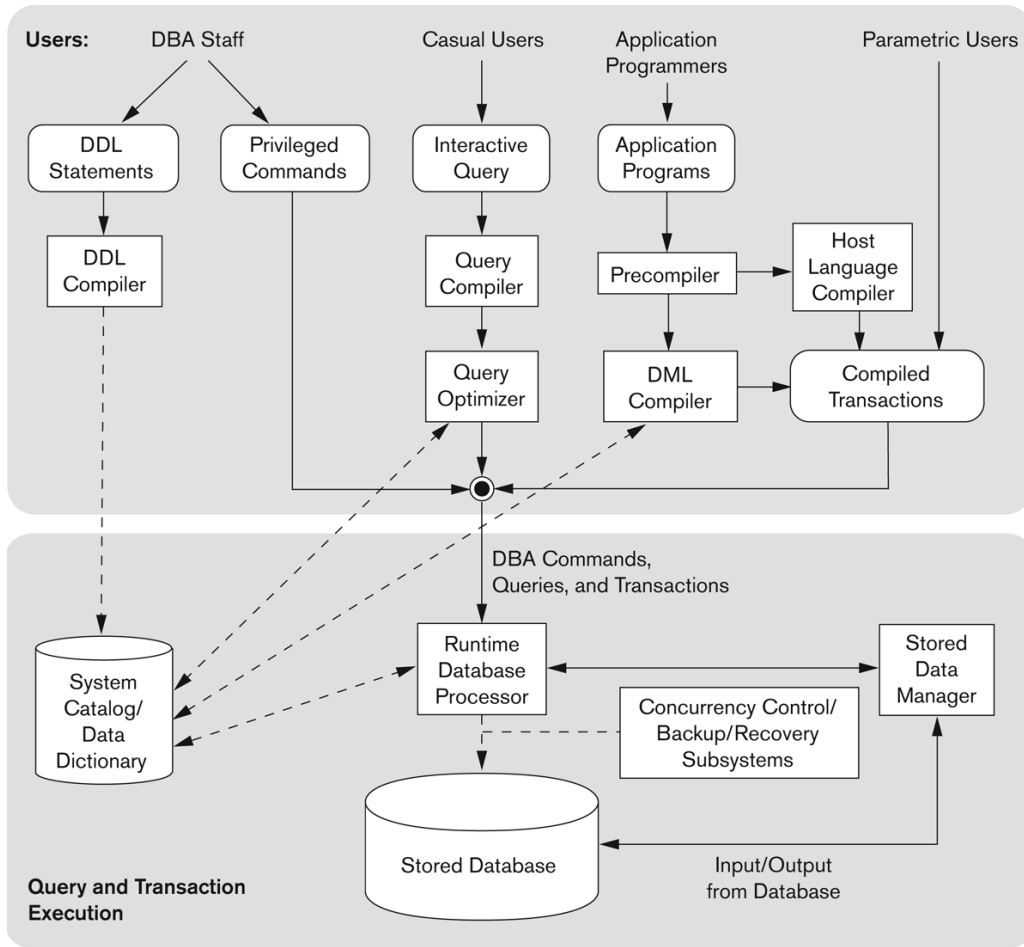
- Reorganizing database file structures.

- Performance monitoring utilities.

- Report generation utilities.

- Other functions, such as sorting, user monitoring, data compression, etc.





## Typical DBMS Component Modules

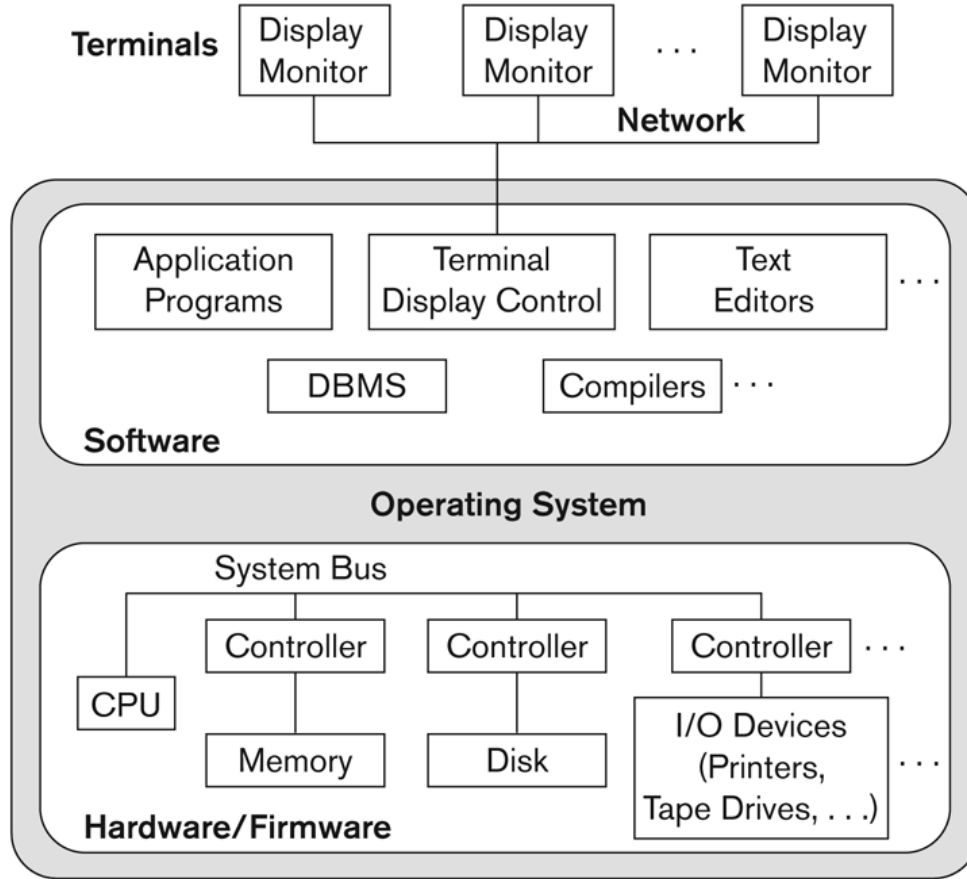
**Figure 2.3**  
Component modules of a DBMS and their interactions.

# 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.



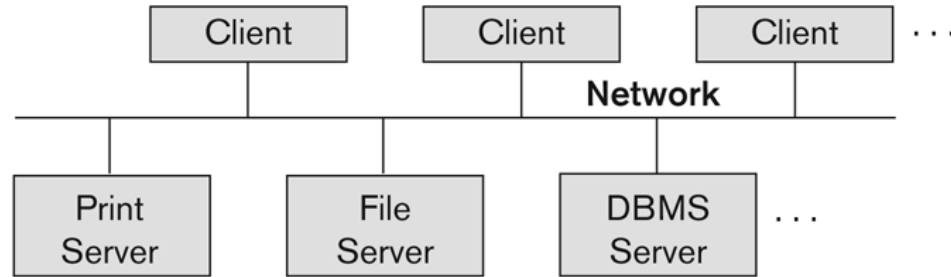
**Figure 2.4**  
A physical centralized  
architecture.

A Physical  
Centralized  
Architecture

# 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 diskless 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

# Two Tier Client-Server Architecture

Client and server must install appropriate client module and server module software for ODBC or JDBC

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.

# 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

- Clients contain user interfaces and Web browsers

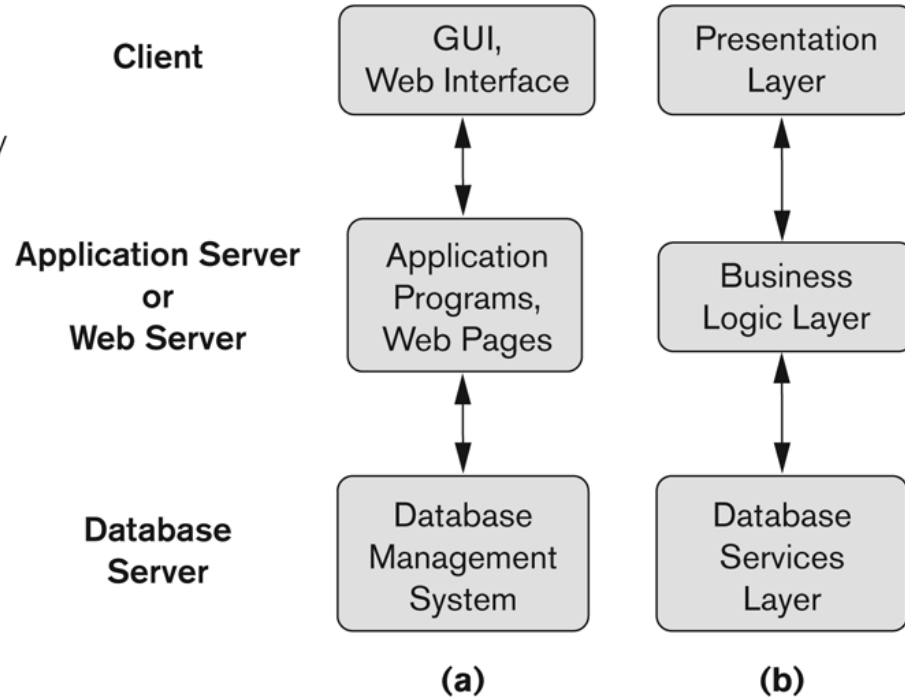
- The client is typically a PC or a mobile device connected to the Web



# Three-tier client-server architecture

**Figure 2.7**

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



# Data Modeling Using the Entity-Relationship (ER) Model

# What we will cover?

Overview of Database Design Process

Example Database Application (COMPANY)

ER Model Concepts

- Entities and Attributes

- Entity Types, Value Sets, and Key Attributes

- Relationships and Relationship Types

- Weak Entity Types

- Roles and Attributes in Relationship Types

ER Diagrams - Notation

ER Diagram for COMPANY Schema

Alternative Notations – UML class diagrams

# Overview of Database Design Process

Two main activities:

- Database design

- Applications design

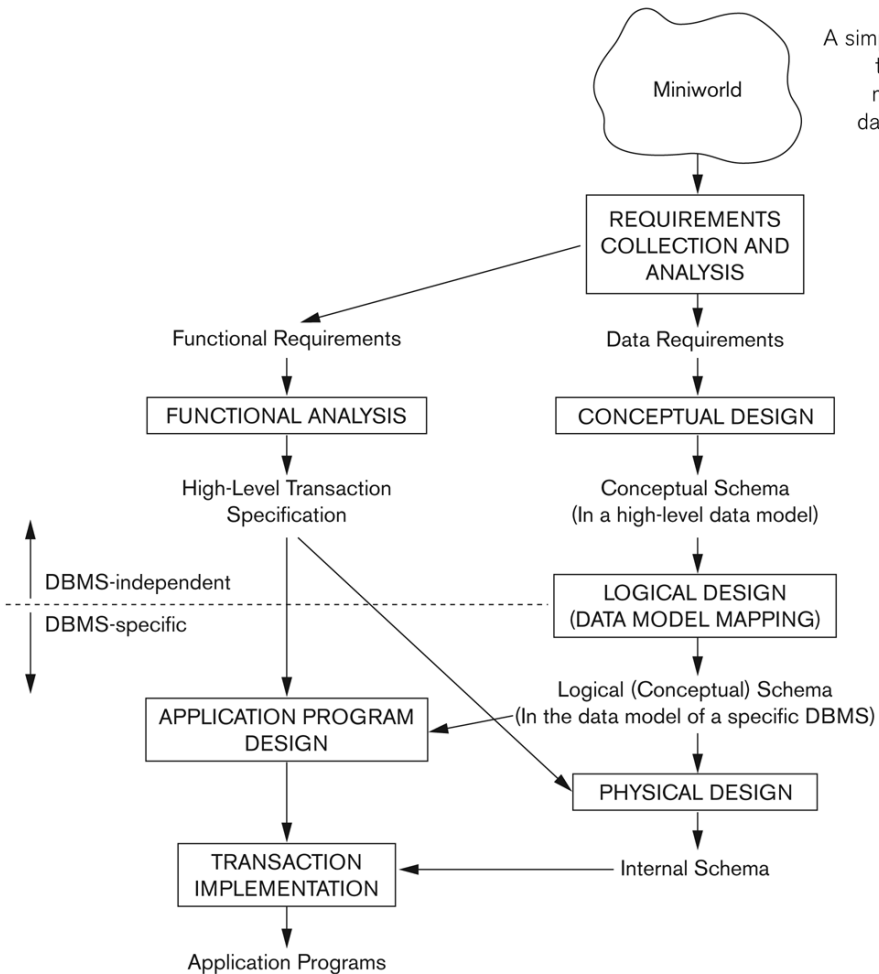
Focus is on conceptual database design

- To design the conceptual schema for a database application

Applications design focuses on the programs and interfaces that access the database

- Generally considered part of software engineering

**Figure 3.1**  
A simplified diagram  
to illustrate the  
main phases of  
database design.



## Overview of Database Design Process

# Methodologies for Conceptual Design

Entity Relationship (ER) Diagrams

Enhanced Entity Relationship (EER) Diagrams

Use of Design Tools in industry for designing and documenting large scale designs

The UML (Unified Modeling Language) Class Diagrams are popular in industry to document conceptual database designs

# Example COMPANY Database

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

The company is organized into DEPARTMENTS. Each department has a name, number and an employee who *manages* the department. We keep track of the start date of the department manager. A department may have several locations.

Each department *controls* a number of PROJECTs. Each project has a unique name, unique number and is located at a single location.

# Example COMPANY Database (Continued)

The database will store each EMPLOYEE's social security number, address, salary, gender, and birthdate.

Each employee *works for* one department but may *work on* several projects.

The DB will keep track of the number of hours per week that an employee currently works on each project.

It is required to keep track of the *direct supervisor* of each employee.

Each employee may *have* a number of DEPENDENTS.

For each dependent, the DB keeps a record of name, gender, birthdate, and relationship to the employee.



# ER Model Concepts

## Entities and Attributes

Entity is a basic concept for the ER model. Entities are specific things or objects in the mini-world that are represented in the database.

For example the EMPLOYEE John Smith, the Research DEPARTMENT, the ProductX PROJECT

Attributes are properties used to describe an entity.

For example an EMPLOYEE entity may have the attributes Name, SSN, Address, gender, BirthDate

A specific entity will have a value for each of its attributes.

For example a specific employee entity may have Name='John Smith', SSN='123456789', Address='731, Fondren, Houston, TX', gender='M', BirthDate='09-JAN-55'

Each attribute has a *value set* (or data type) associated with it – e.g. integer, string, date, enumerated type, ...

# Types of Attributes (1)

## Simple

Each entity has a single atomic value for the attribute. For example, SSN or gender.

## Composite

The attribute may be composed of several components. For example:  
Address(Apt#, House#, Street, City, State, ZipCode, Country), or  
Name(FirstName, MiddleName, LastName).

## Multi-valued

Multiple values for the attribute. For example, Color of a CAR or PreviousDegrees of a STUDENT.

Denoted as {Color} or {PreviousDegrees}.

## Types of Attributes (2)

In general, composite and multi-valued attributes may be nested arbitrarily to any number of levels, although this is rare.

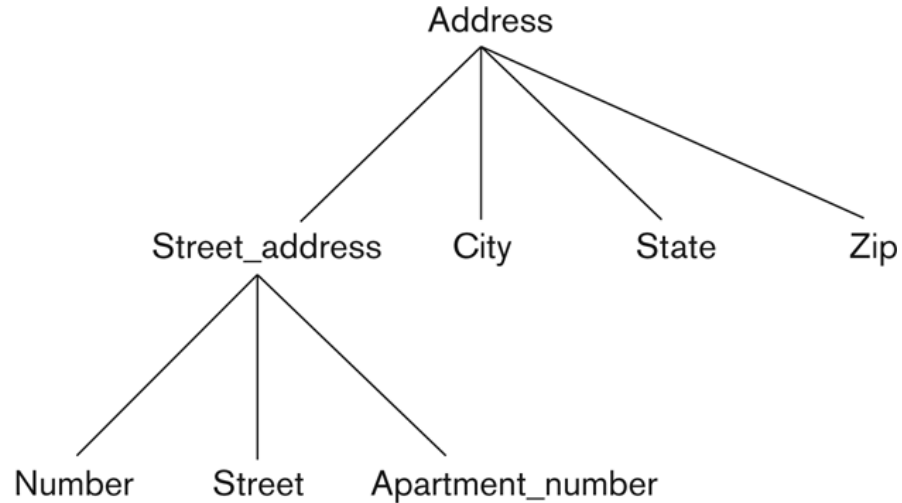
For example, PreviousDegrees of a STUDENT is a composite multi-valued attribute denoted by {PreviousDegrees (College, Year, Degree, Field)}

Multiple PreviousDegrees values can exist

Each has four subcomponent attributes:

College, Year, Degree, Field

# Example of a composite attribute



**Figure 3.4**

A hierarchy of composite attributes.

# Entity Types and Key Attributes (1)

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

For example, the entity type EMPLOYEE and PROJECT.

An attribute of an entity type for which each entity must have a unique value is called a key attribute of the entity type.

For example, SSN of EMPLOYEE.

# Entity Types and Key Attributes (2)

A key attribute may be composite.

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

An entity type may have more than one key.

The CAR entity type may have two keys:

VehicleIdentificationNumber (popularly called VIN)

VehicleTagNumber (Number, State), aka license plate number.

Each key is underlined

# Entity Set

Each entity type will have a collection of entities stored in the database

Called the **entity set** or sometimes **entity collection**

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

However, entity type and entity set may be given different names

Entity set is the current *state* of the entities of that type that are stored in the database

# Value Sets (Domains) of Attributes

Each simple attribute is associated with a value set

E.g., Lastname has a value which is a character string of upto 15 characters, say

Date has a value consisting of MM-DD-YYYY where each letter is an integer

A **value set** specifies the set of values associated with an attribute



# Attributes and Value Sets

Value sets are similar to data types in most programming languages – e.g., integer, character (n), real, bit

Mathematically, an attribute  $A$  for an entity type  $E$  whose value set is  $V$  is defined as a function

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

Where  $P(V)$  indicates a power set (which means all possible subsets) of  $V$ . The above definition covers simple and multivalued attributes.

We refer to the value of attribute  $A$  for entity  $e$  as  $A(e)$  – Name (Employee)

# Displaying an Entity type

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

Attributes are displayed in ovals


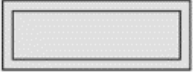
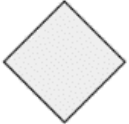




- Each attribute is connected to its entity type

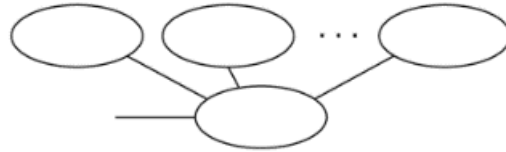
- Components of a composite attribute are connected to the oval representing the composite attribute

- Each key attribute is underlined

- Multivalued attributes displayed in double ovals

**Figure 3.14**  
Summary of the  
notation for ER  
diagrams.

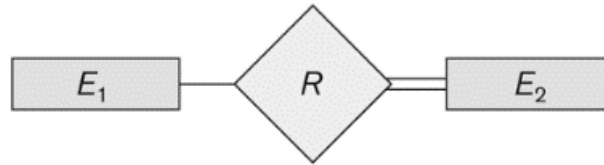
| Symbol  | Meaning                  |
|---|--------------------------|
|  | Entity                   |
|  | Weak Entity              |
|  | Relationship             |
|  | Identifying Relationship |
|  | Attribute                |
|  | Key Attribute            |
|  | Multivalued Attribute    |



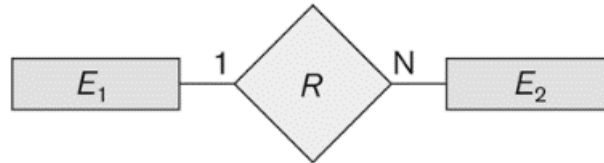
Composite Attribute



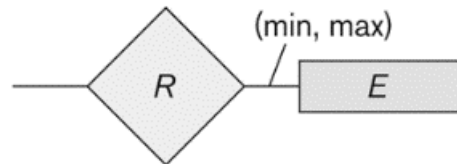
Derived Attribute



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  $R$

 pk.profgiri

 Ponnurangam.kumaraguru

 /in/ponguru

 ponguru

 pk.guru@iiit.ac.in

Thank you  
for attending  
the class!!!