ATME College of Engineering

ATME College of Engineering, Mysuru

Department of Computer Science & Engineering (Data Science)

Database Management System(BCS403)
Module-1

CHAPTER 1: INTRODUCTION TO DATABASES

Outline

- Introduction
  - Basic Definitions
  - Typical DBMS Functionality
  - Example of a Database
- Characteristics of the Database Approach
- Advantages of Using the Database Approach

Introduction

- **Database:** A collection of related data
  - **Data: -** Known facts

    - implicit meaning
  - **Implicit Properties**

    - represents some aspect of the real world, sometimes called the **miniworld**

    - logically coherent collection of data with some inherent meaning

    - designed, built, and populated with data for a specific purpose
  - Database

    - any size and complexity

    - may be generated and maintained manually or it may be computerized

➢ Database Management System (DBMS):

- General-purpose software system that facilitates the processes of
  - Defining
  - constructing
  - manipulating and
  - sharing databases among various users and applications.
- Functions
  - protecting the database: system protection & security protection
  - maintaining it over a long period of time

Database + DBMS software = Database System

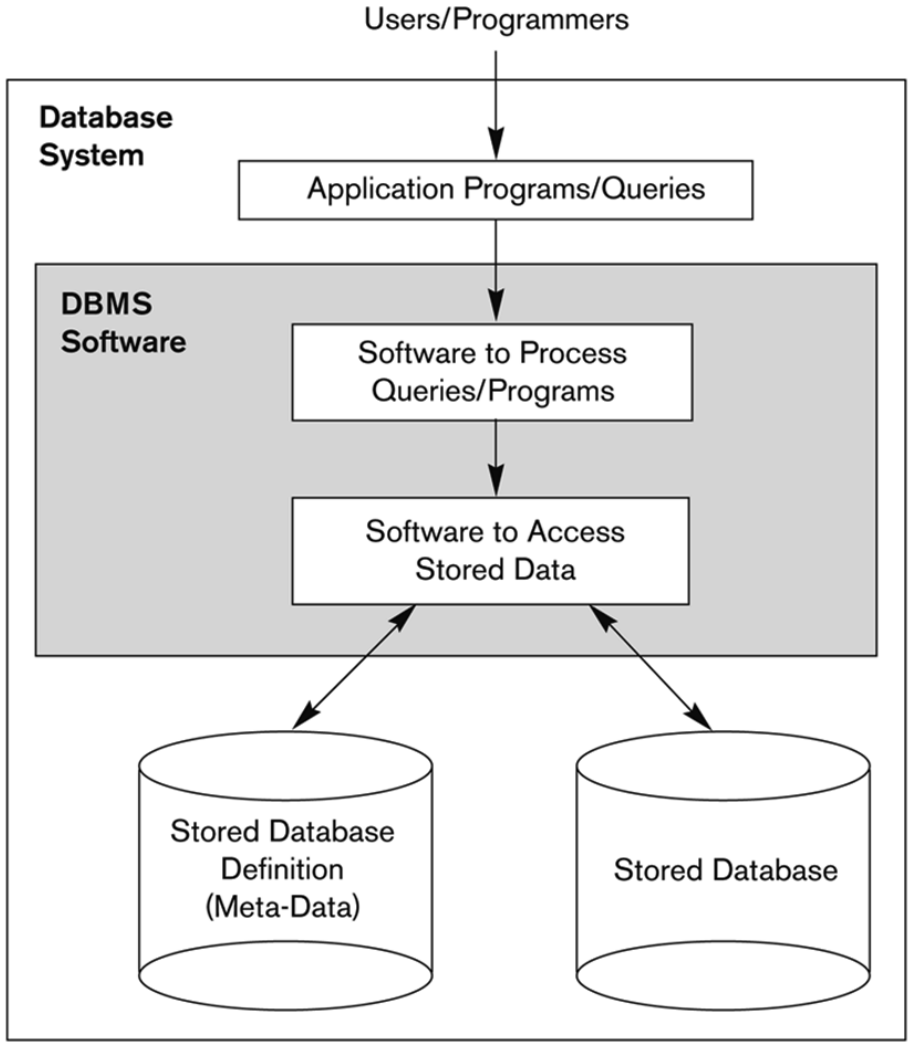


Fig: A Simplified Database System Environment

Example of a Database

- **Mini-world for the example:**
  - **UNIVERSITY** database for maintaining information concerning students, courses, and grades in a university environment
  - **Some mini-world *entities*:**
    1. STUDENT file: record on each student
    2. COURSE file: record on each course
    3. SECTION file: record on each section of a course
    4. GRADE_REPORT file: stores the grades
    5. PREREQUISITE file : stores the prerequisites of each course

Defining a UNIVERSITY database

- Specify data elements to be stored in each record
  - For example:
    - each STUDENT record includes data to represent the student's Name, Student_number, Class, Major
    - each COURSE record includes data to represent the course_name, Course_number, Credit_hours and Department
- Specify a data type for each data element within a record
  - For example:
    - student's Name is a string of alphabetic characters
    - student_number is an integer

STUDENT

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

COURSE

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

SECTION

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

Fig 1.2a : A database that stores student and course information

GRADE_REPORT

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

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Fig 1.2b : A database that stores student and course information

Constructing the UNIVERSITY database

- store data to represent each student, course, section, grade report, and prerequisite as a record in the appropriate file
- records in the various files may be related
  - Example:
    - The record for Smith in the STUDENT file is related to two records in the GRADE_REPORT file that specify Smith's grades in two sections.

Manipulating a UNIVERSITY database

➢involves querying and updating.

- **Examples of queries:**
  - Retrieve the transcript—a list of all courses and grades—of ‘Smith’
  - List the prerequisites of the ‘Database’ course
- **Examples of updates :**
  - Create a new section for the ‘Database’ course for the semester fall 2008
  - Enter a grade of ‘A’ for ‘Smith’ in the ‘discrete mathematical structure’ section

Characteristics of the Database Approach

File Processing v/s DBMS

Redundancy

- 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 files on students and their grades
- Programs to print a student's transcript and to enter new grades are implemented as part of the application
- A second user, the ***accounting office***, may keep track of students' fees and their payments
- Although both users are interested in data about students, each user maintains separate files and programs to manipulate these files because each requires some data not available from the other user's files

- Redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common up-to-date data.
- In the database approach, a single repository maintains data 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, the names or labels of data are defined once, and used repeatedly by queries, transactions, and applications.

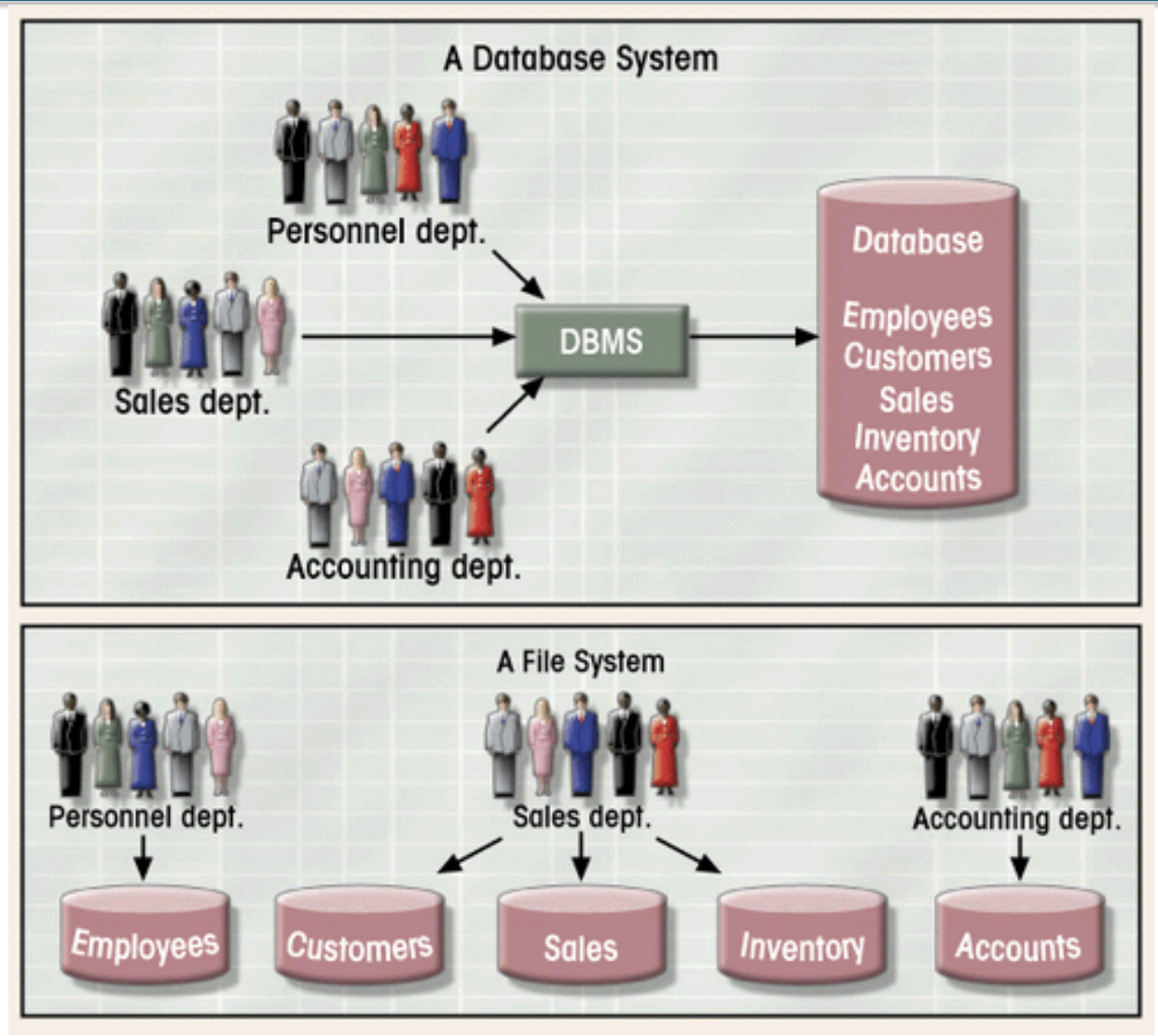
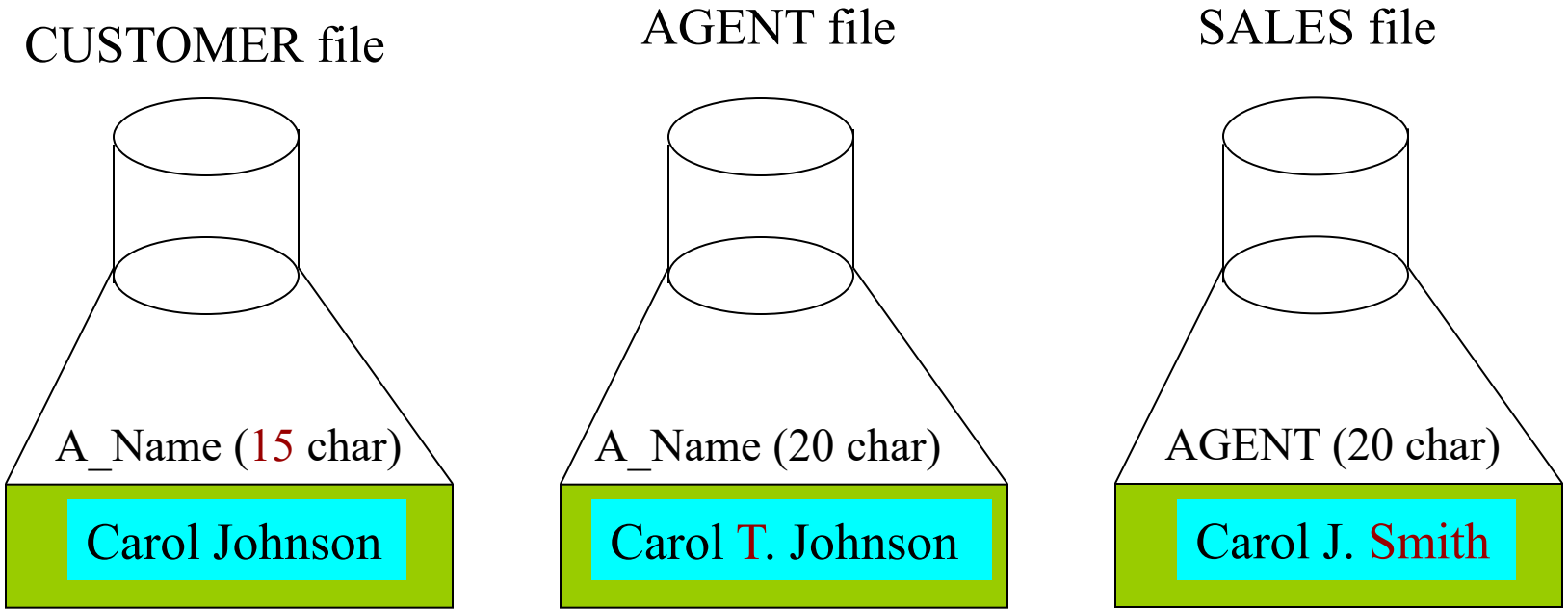


Fig 1.3: Database System vs. File System

File System: Problem Case



- inconsistent field name, field size
- inconsistent data values
- data duplication

- The main characteristics of the database approach versus the file-processing approach are the following:
  - Self-describing nature of a database system
  - Insulation between programs and data, and data abstraction
  - Support of multiple views of the data
  - Sharing of data and multiuser transaction processing

Self-Describing Nature of a Database System

- Database system contains not only the database itself but also a complete definition or description of the database structure and constraints
- This definition is stored in the DBMS catalog
- The information stored in the catalog is called **meta-data**
- The catalog is used by the DBMS software and also by database users who need information about the database structure
- In traditional file processing, data definition is typically part of the application programs themselves.
- Hence, these programs are constrained to work with only one specific database, whose structure is declared in the application programs.

RELATIONS

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

COLUMNS

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

Note: Major_type is defined as an enumerated type with all known majors.
XXXXNNNN is used to define a type with four alpha characters followed by four digits.

Insulation between Programs and Data, and Data Abstraction

Program-data independence

- In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access that file.
- By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs.

Program-operation independence

- Users can define operations on data as part of the database definitions.
- User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented.

Data abstraction

- A DBMS provides users with a **conceptual representation** of data that does not include many of the details of how the data is stored or how the operations are implemented

Support of Multiple Views of the Data

- A database 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 that is derived from the database files but is not explicitly stored
- For example, one user of the database may be interested only in accessing and printing the transcript of each student; the view for this user is shown in Figure below

TRANSCRIPT

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

(a)

Sharing of Data and Multiuser Transaction Processing

- A multiuser DBMS must allow multiple users to access the database at the same time.
- The DBMS must include **concurrency control** software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.
- For example, when several reservation agents try to assign a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one agent at a time for assignment to a passenger.

Advantages of Using the DBMS Approach

1. Controlling Redundancy
2. Restricting Unauthorized Access
3. Providing Persistent Storage for Program Objects
4. Providing Storage Structures and Search Techniques for Efficient Query Processing
5. Providing Backup and Recovery
6. Providing Multiple User Interfaces
7. Representing Complex Relationships among Data
8. Enforcing Integrity Constraints
9. Permitting Inferencing and Actions Using Rules
10. Additional Implications of Using the Database Approach

Advantages of Using the DBMS Approach

1. Controlling Redundancy

- Redundancy leads to several problems- duplication, wastage of storage and data may become inconsistent.
- In the database approach, the views of different user groups are integrated during database design.
- Database design that stores each logical data item in *only one place* in the database. This is known as **data normalization**, and it ensures consistency and saves storage space

- It is sometimes necessary to use **controlled redundancy** to improve the performance of queries
- For example, we may store Student_name and Course_number redundantly in a GRADE_REPORT file because whenever we retrieve a GRADE_REPORT record, we want to retrieve the student name and course number along with the grade, student number, and section identifier.
- By placing all the data together, we do not have to search multiple files to collect this data. This is known as **denormalization**.

2.Restricting Unauthorized Access

- When multiple users share a large database, it is likely that most users will not be authorized to access all information in the database.
- For example, financial data is often considered confidential, and only authorized persons are allowed to access such data.
- A DBMS should provide a **security and authorization subsystem,** which the DBA uses to create accounts and to specify account restrictions
- DBMS should enforce these restrictions automatically

3. Providing Persistent Storage for Program Objects

- object-oriented database systems make it easier for complex runtime objects to be saved in secondary storage so as to survive beyond program termination and to be retrievable at a later time.
- Object-oriented database systems are compatible with programming languages such as C++ and Java, and the DBMS software automatically performs any necessary conversions.

4. Providing Storage Structures and Search Techniques for Efficient Query Processing

- The DBMS maintains indexes that are utilized to improve the execution time of queries and updates.
- DBMS has a **buffering** or **caching** module that maintains parts of the database in main memory buffers.
- The **query processing and optimization** module is responsible for choosing an efficient query execution plan for each query submitted to the system

5. Providing Backup and Recovery

- The **backup and recovery subsystem** of the DBMS is responsible for recovery.
- ensures that recovery is possible in the case of a system crash during execution of one or more transactions.
- Disk backup is also necessary in case of a catastrophic disk failure.

6 Providing Multiple User Interfaces

- Because many types of users with varying levels of technical knowledge use 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 and command codes for parametric users
  - menu-driven interfaces and natural language interfaces for standalone users.

7. Representing Complex Relationships among Data

- A database may include numerous varieties of data that are interrelated in many ways.
- For example each section record is related to one course record and to a number of GRADE_REPORT records—one for each student who completed that section.
- A 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 and update related data easily and efficiently.

8.Enforcing Integrity Constraints

- Most database applications are such that the semantics of the data require that it satisfy certain restrictions in order to make sense.
- The simplest type of integrity constraint involves specifying a data type for each data item.
  - For example, in student table we specified that the value of Name must be a string of no more than 30 alphabetic characters.
- More complex type of constraint is **referential integrity** involves specifying that a record in one file must be related to records in other files
  - For example, in university database, we can specify that every section record must be related to a course record.

- Another type of constraint specifies uniqueness on data item values, such as every course record must have a unique value for Course_number. This is known as a **key** or **uniqueness** constraint.
- It is the responsibility of the database designers to identify integrity constraints during database design.

9.Permitting Inferencing and Actions Using Rules

- In a deductive database system, one may specify declarative rules that allow the database to infer new data.
- For example figure out which students are on academic probation.
- These can be specified declaratively as rules, which when compiled and maintained by the DBMS
- In today's relational database systems, it is possible to associate triggers with tables.
- A trigger is a form of a rule activated by updates to the table
- Stored procedures
  - More involved procedures to enforce rules

10 Additional Implications of Using the Database Approach

➢ Potential for Enforcing Standards

- permits the DBA to define and enforce standards among database users in a large organizatio which facilitates communication and cooperation among various departments, projects, an users within the organization.

- Standards can be defined for names and formats of data elements, display formats, repor structures and so on.

➢ Reduced Application Development Time

- once a database is up and running, substantially less time is generally required to create new applications using DBMS facilities

➢Flexibility

- It may be necessary to change the structure of a database as requirements change.

- DBMSs allow changes to the structure of the database without affecting the stored data and the existing application programs.

➢Availability of Up-to-Date Information

- DBMS makes the database available to all users

- availability of up-to-date information is essential for many transaction-processing applications, such as reservation systems or banking databases

➢ Economies of Scale

- DBMS approach permits consolidation of data and applications, to overlap between activities of data-processing in different projects or departments

- enables the whole organization to invest in more powerful processors, storage devices, or communication gear, rather than having each department purchase its equipment.

-reduces overall costs of operation and management.

Question Bank

1. Define database and briefly explain the implicit properties of the database.
2. Discuss the main Characteristics of the database approach and how does it differ from Traditional file systems?
3. Define Database Management System. Briefly discuss the advantages of using the DBMS.

Database system concepts

- Data Models, Schemas, and Instances
- Three-Schema Architecture and Data Independence
- Database Languages and Interfaces
- The Database System Environment

Data Models, Schemas, and Instances

- **Data abstraction**
  - Suppression of details of data organization and storage
  - Highlighting of the essential features for an improved understanding of data
- **Data model**
  - Collection of concepts that describe the structure of a database
  - Provides means to achieve data abstraction
- **Basic operations**
  - Specify retrievals and updates on the database
- **Dynamic aspect or behavior of a database application**
  - Allows the database designer to specify a set of valid operations allowed on database objects.
- **Examples**
  - **Basic data model operations**: insert, delete, modify, or retrieve any kind of object
  - **User defined operation** : COMPUTE_GPA, which can be applied to a STUDENT object

Categories of Data Models

- categorize according to the types of concepts they use to describe the database structure:

1. High-level or conceptual data models
2. Low-level or physical data models
3. Representational or implementation data models

1. High-level or conceptual data models

- Close to the way many users perceive data
- use concepts such as
  - Entity
  - Attribute
  - Relationship
  - Entity-Relationship model

2. Low-level or physical data models

- Describe the details of how data is stored on computer storage media
- Access path : Structure that makes the search for particular database records efficient
  - Index is an example of an access path which allows direct access to data using an index term or a keyword

3. Representational data models

- Easily understood by end users
- most frequently used in traditional commercial DBMSs.
  - Relational data model
  - Network and hierarchical models
- Record-based data models.
  - Object data model

Schemas, Instances, and Database State

- **Database schema**
  - Description of a database
  - specified during database design and is not expected to change frequently
- **Schema diagram**
  - Displays selected aspects of 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

- **Schema construct**
  - Each object in the schema
  - ex: student or course
- **Database state or snapshot**
  - Data in database at a particular moment in time
  - also called the current set of occurrences or instances in the database
  - For example:
  - The STUDENT construct will contain the set of individual student entities (records) as its instances. Every time we insert or delete a record or change the value of a data item in a record, we change one state of the database into another state

Important- distinction between database schema and database state

- **Define a new database**
  - Specify database schema to the DBMS
  - At this point, the corresponding database state is the empty state with no data.
- **Initial state**
  - Populated or loaded with the initial data
- Every time an update operation is applied to the database, we get another database state.
- At any point in time, the database has a **current state**.

- **Valid state**
  - Satisfies the structure and constraints specified in the schema
- **Schema evolution**
  - Changes applied to schema as application requirements change

The Three-Schema Architecture

- **Goal**
  - to separate the user applications from the physical database.

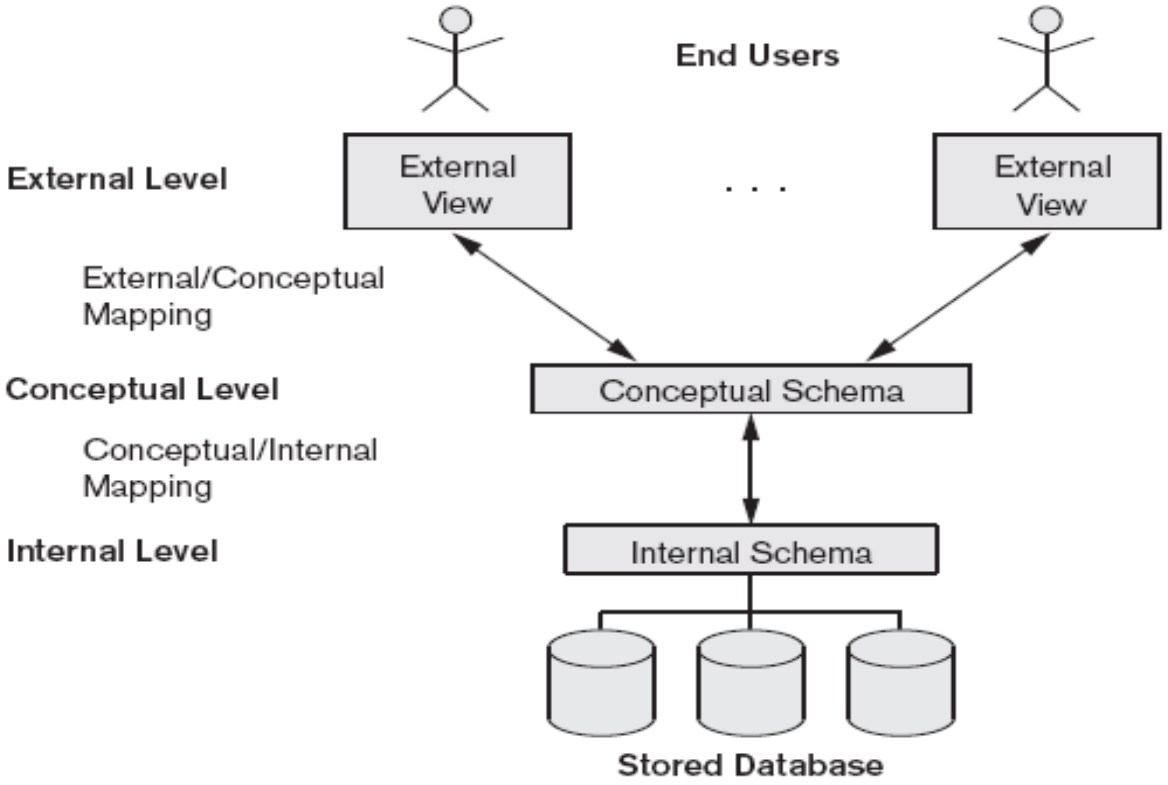


Figure 2.1: The three-schema architecture.

➢schemas can be defined at three levels:

- **Internal level –internal schema**
  - Describes physical storage structure of the database
  - uses a physical data model and describes the complete details of data storage and access paths for the database.
- **Conceptual level – conceptual schema**
  - Describes structure of the whole database for a community of users
  - hides the details of physical storage structures
  - concentrates on describing entities, data types, relationships, user operations, and constraints

- **External or view level –external schema or user views**
  - Describes part of the database that a particular user group is interested in

➢ The three schemas are only descriptions of data; the stored data that actually exists is at the physical level only

➢ In a DBMS based on the three-schema architecture, each user group refers to its own external schema.

➢ Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database.

➢ If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user's external view.

Mappings : The processes of transforming requests and results between levels.

- **An example of three levels**

Customer_Loan Cust_ID: 101 Loan_No:1011 Amount_in_Dollors:8755.00	**External**
CREATE TABLE Customer_Loan(Cust_ID NUMBER(4), Loan_No NUMBER(4), Amount_in_Dollars NUMBER(7,2))	**Conceptual**
Cust_ID TYPE = BYTE(4), OFFSET = 0 Loan_No TYPE = BYTE(4), OFFSET = 4 Amount_in_Dollars TYPE = BYTE(7), OFFSET = 8	**Internal**

Data Independence

– capacity to change the schema at one level of a database system without having to change the schema at the next higher level

- **Types:**

1. **Logical data independence**

– 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 to change constraints.

2. Physical data independence

- capacity to change the internal schema without having to change the conceptual schema
- Changes to the internal schema may be needed because some physical files were reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update.

Database Languages and Interfaces

➢DBMS Languages

- Once the design of a database is completed and a DBMS is chosen to implement the database the first step is to specify conceptual and internal schemas for the database and any mappings between the two

- **Data definition language (DDL)**
  - Defines both schemas
  - DDL compiler to process DDL statements in order to identify descriptions of the schema constructs and to store the schema description in the DBMS catalog

- **Storage definition language (SDL)**
  - Specifies the internal schema
- **View definition language (VDL)**
  - specify user views and their mappings to the conceptual schema
- **Data manipulation language (DML)**
  - retrieval, insertion, deletion, and modification of the data
  - Types:
    - **High-level** or **nonprocedural DML**
      - to specify complex database operations concisely
      - DML statements either to be entered interactively from a display monitor / terminal or to be embedded in a general-purpose programming language.

- **Low level or procedural DML**
  - must be embedded in a general-purpose programming language
  - retrieves individual records or objects from the database and processes each separately
  - also called **record-at-a-time** DMLs

➢**set-at-a-time** or **set-oriented** DMLs

- Highlevel DMLs, such as SQL, can specify and retrieve many records in a single DMLstatement
- a high-level DML often specifies which data to retrieve rather than how to retrieve it

- In most DBMSs the DDL is used to define both conceptual and external schemas.
- In relational DBMSs, SQL is used in the role of VDL to define user or application views as results of predefined queries
- **host language**
  - DML commands embedded in a general-purpose programming language

➢DBMS Interfaces

- **Menu-based interfaces for Web clients or browsing**

- Present the user with lists of options (called menus) that lead the user through the formulation of a request.
- no need to memorize the specific commands and syntax of a query language
- Pull-down menus are a very popular technique in Web-based user interfaces.

- **Apps for mobile devices**

- Present mobile user with apps to access their data
- for example, banking, reservations and insurance companies provide apps that allow users to access their data through a mobile phone
- built-in programmed interfaces that allow users to login using their account name and password
- Provide a limited menu of options for mobile access to user data
- Options for paying bills, making reservations

- **Forms-based interfaces**
  - displays a form to each user
  - usually designed and programmed for naive users as interface to canned transactions
  - Many DBMSs have **forms specification languages**, which are special languages that help programmers specify such forms.
- **Graphical user interfaces**
  - displays a schema to the user in diagrammatic form
  - The user then can specify a query by manipulating the diagram
  - GUIs utilize both menus and forms.
  - use a pointing device such as a mouse, to select certain parts of the displayed schema diagram

- **Natural Language Interfaces**
  - accept requests written in English or some other language and attempt to understand them
  - has its own schema, which is similar to the database conceptual schema, as well as a dictionary of important words.
  - refers to the words in its schema, as well as to the set of standard words in its dictionary, to interpret the request.
  - If the interpretation is successful, the interface generates a high-level query corresponding to the natural language request and submits it to the DBMS for processing otherwise, a dialogue is started with the user to clarify the request.
- **Keyword-based database search**
  - Similar to web search engines which accepts strings of natural language words and matches them with documents at specific sites
  - Use predefined indexes on words and use ranking functions to retrieve and present resulting documents

Speech Input and Output

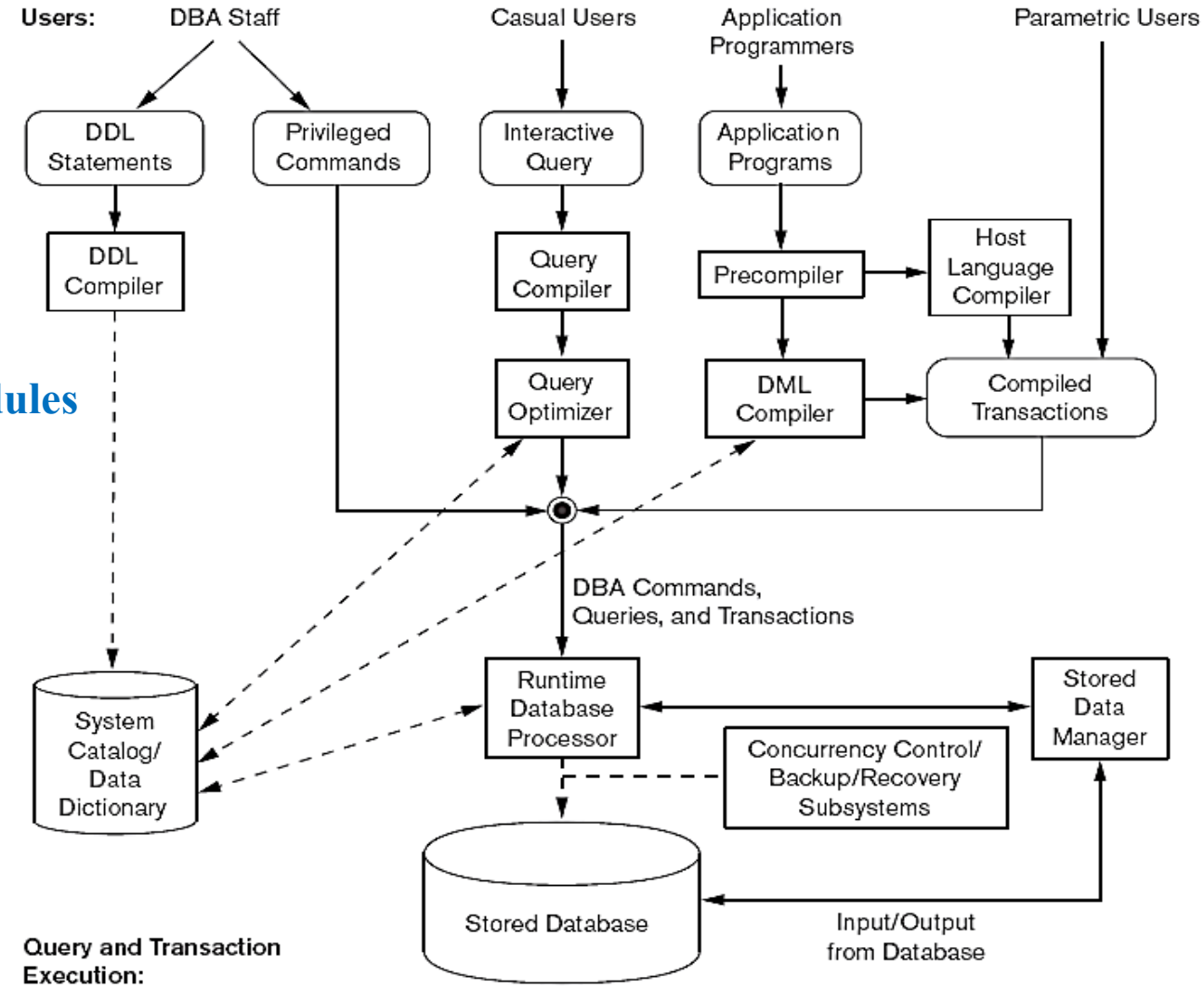
- Applications with limited vocabularies such as inquiries for telephone directory, flight arrival/departure, and credit card account information are allowing speech for input and output to enable customers to access this information.
- The speech input is detected using a library of predefined words and used to set up the parameters that are supplied to the queries.

- **Interfaces for Parametric Users**
  - Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly
  - special interface for each known class of naive users
  - a small set of abbreviated commands is included, with the goal of minimizing the number of keystrokes required for each request.

- **Interfaces for the DBA**
  - Most database systems contain privileged commands that can be used only by the DBA staff
  - These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, and reorganizing the storage structures of a database.

The Database System Environment

DBMS Component Modules



- The top part of the figure refers to the various users of the database environment and their interfaces.
- The lower part shows the internals of the DBMS responsible for storage of data and processing of transactions.
- **Top part of the figure**:
  - **Interfaces-**for the DBA staff, casual users , application programmers and parametric users
  - **DDL compiler-**processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog.
  - **Interactive query -**interface-Casual users
  - **Query compiler-** validates for correctness of the query syntax, the f files and data elements & compiles them into an internal form

- **Query optimizer** - rearrangement and possible reordering of operations, elimination of redundancies, and use of correct algorithms and indexes during execution. Makes calls on the runtime processor.
- **Precompiler -** extracts DML commands from an application program and sends to the DML compiler for compilation into object code for database access
- **host language compiler -** rest of the program
- The object codes for the DML commands and the rest of the program are linked, forming a **canned transaction** whose executable code includes calls to the runtime database processor.

➢ **lower part of figure:**

- **Runtime database processor**
  - executes with runtime parameters :

    - privileged command - executable query plan - canned transactions
  - works with the **system catalog and stored data manager**
  - management of buffers in the main memory
- **stored data manager** uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory.
- **concurrency control** and **backup and recovery systems** integrated into the working of the runtime database processor for purposes of transaction management.

Database System Utilities

- Database utilities help the DBA to manage the database system
- **functions**
  - **Loading**
    - used to load existing data files—such as text files or sequential files into the database
  - **Backup**
    - creates a backup copy of the database by dumping the entire database onto tape or other mass storage medium
    - used to restore the database in case of catastrophic disk failure
    - Incremental backups are also often used, where only changes since the previous backup are recorded
    - Incremental backup is more complex, but saves storage space

- **Database storage reorganization**
  - used to reorganize a set of database files into different file organizations, and create new access paths to improve performance.
- **Performance monitoring**
  - monitors database usage and provides statistics to the DBA
  - DBA uses the statistics in making decisions such as whether or not to reorganize files or whether to add or drop indexes to improve performance.

Tools, Application Environments, and Communications Facilities

➢ Tools

- **CASE**
  - used in the design phase of database systems
- **data dictionary**
  - stores other information, such as design decisions, usage standards, application program descriptions, and user information

➢ Application development environments

- PowerBuilder (Sybase) or JBuilder (Borland)
- provide an environment for developing database applications including database design, GUI development, querying and updating, and application program development.

- **communications software**
  - allow users at locations remote from the database system site to access the database through computer terminals, workstations, or personal computers
  - integrated DBMS and data communications system is called a DB/DC system

Question Bank

1. Describe the three-schema architecture. Why do we need mappings between schema levels? How do different schema definition languages support this architecture?
2. Differentiate between

   i) logical data independence and physical data independence

   ii) procedural and nonprocedural DMLs

2. Discuss the various database languages.
3. Discuss the different types of user-friendly interfaces and the types of users who typically use each.
4. Explain the component modules of DBMS and their interaction with the help of a diagram.
5. Discuss some types of database utilities and tools and their functions.

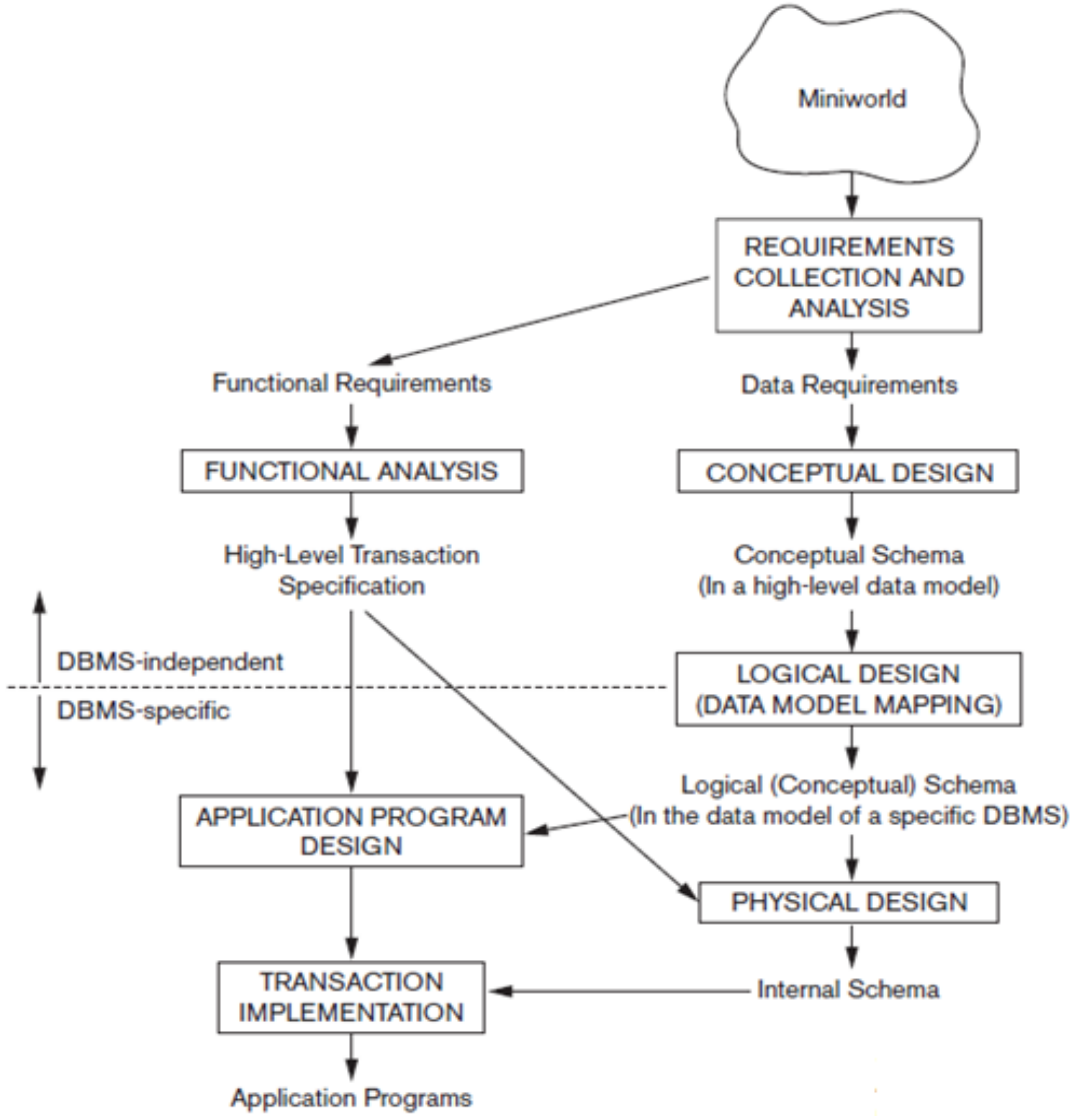
DATA MODELLING USING ENTITIES – RELATIONSHIP(ER) MODEL

- Conceptual modeling is a very important phase in designing a successful database application

Chapter Outline

- Using High-Level Conceptual Data Models for Database Design
- Entity Types, Entity Sets, Attributes, and Keys
- A Sample Database Application
- Relationship Types, Relationship Sets, Roles, and Structural Constraints
- Weak Entity Types
- ER Diagrams

Using High-Level Conceptual Data Models for Database Design



Entity Types, Entity Sets, Attributes, and Keys

Entity

- a thing in the real world with an independent existence.
- may be an object with a physical existence - for example, a particular person, car, house, or employee or
- may be an object with a conceptual existence - for instance, a company, a job, or a university course

Attributes

- Particular properties that describe entity
- For example, an EMPLOYEE entity may be described by the employee's name, age, address, salary, and job

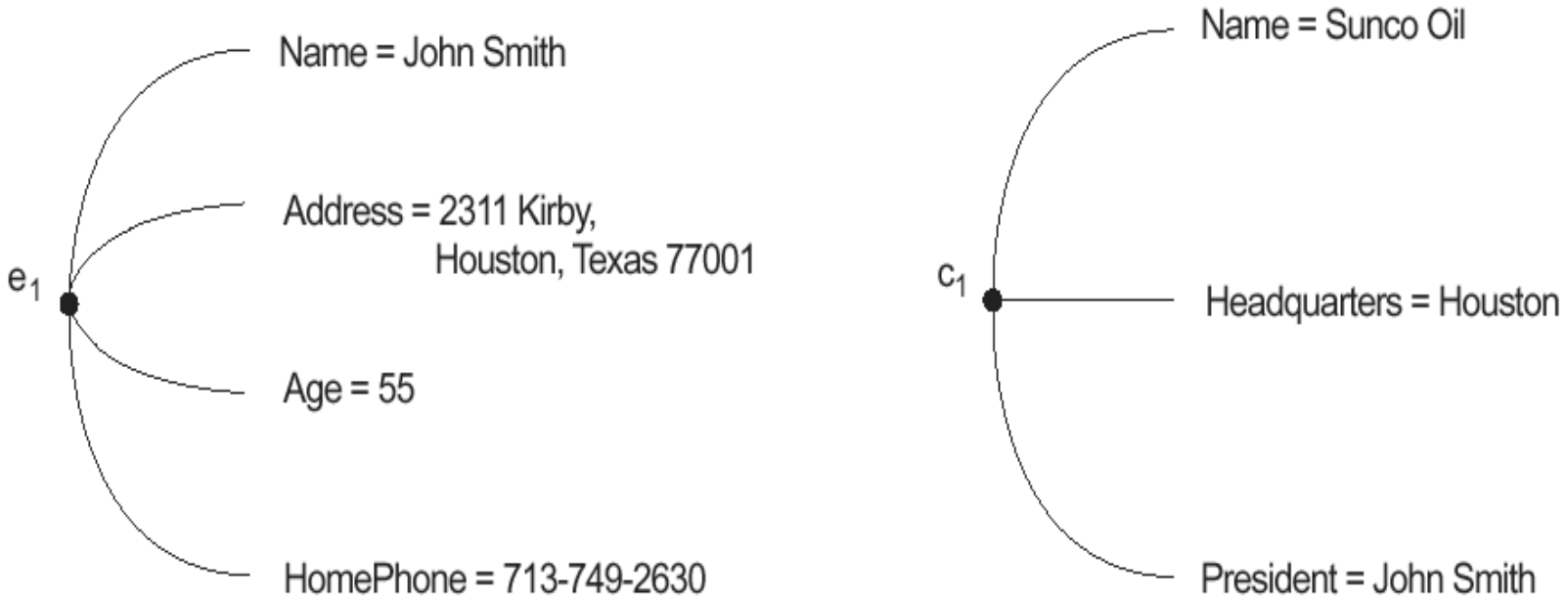


Figure : Two entities, EMPLOYEE e1, and COMPANY c1, and their attributes

➢ Types of attributes:

- Composite versus Simple (Atomic) Attributes
- Single-valued versus multivalued
- Stored versus derived
- NULL values
- Complex attributes

Composite versus Simple (Atomic) Attributes

Composite Attributes

- can be divided into smaller subparts, which represent more basic attributes with independent meanings.
- For example, the Address attribute of the EMPLOYEE entity can be subdivided into Street_address, City, State, and Zip
- can form a hierarchy
- For example, Street_address can be further subdivided into three simple component attributes: Number, Street, and Apartment_number

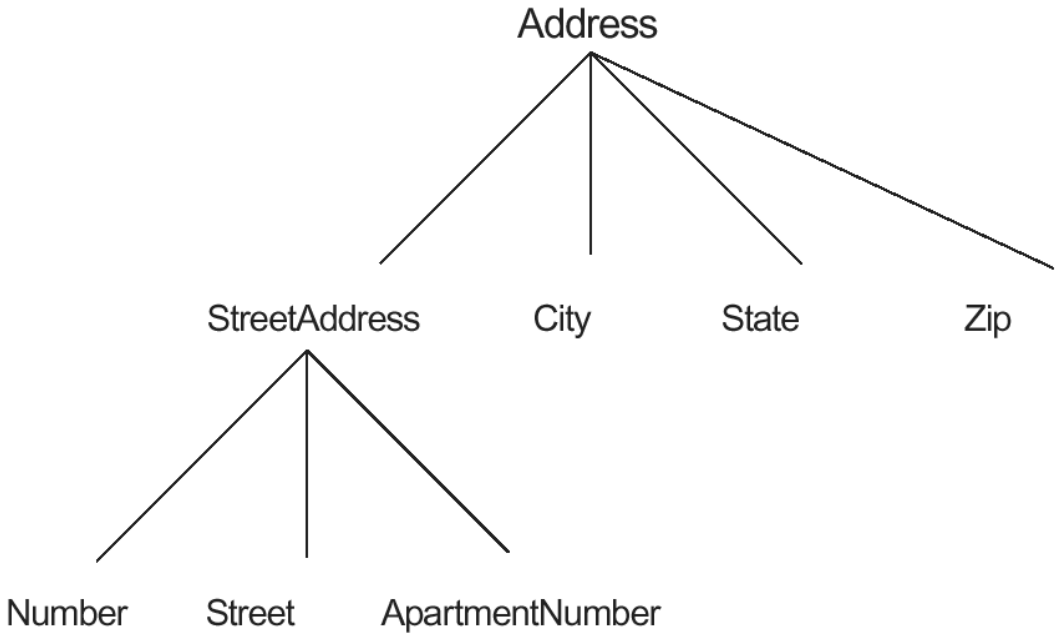


Figure: A hierarchy of composite attributes.

- The value of a composite attribute is the concatenation of the values of its component simple attributes.

- **Simple or Atomic attributes**
  - Attributes that are not divisible
  - Example: SSN

Single-Valued versus Multivalued Attributes

- **Single-Valued**
  - attributes that have a single value for a particular entity
  - For example: age attribute of a person
- **Multivalued**
  - attributes that can have a set of values
  - For example: college degree of a person

Stored versus Derived Attributes

- **Stored**
  - An attribute, which cannot be derived from other attribute
  - For example, BirthDate of an employee
- **Derived**
  - Attributes derived from other stored attribute
  - For example age from Date of Birth and Today's date

Null Value Attribute(Optional Attribute)

- attribute may not have a value in it and can be left blank
- For example, the Apartment_number attribute of an address applies only to addresses that are in apartment buildings and not to other types of residences,such as single-family homes.

Complex Attributes

- If an attribute for an entity, is built using composite and multivalued attributes, then these attributes are called complex attributes
- For example, a person can have more than one residence and each residence can have multiple phones, an addressphone for a person entity can be specified as –

{ Addressphone (phone {(Area Code, Phone Number)},

Address(Sector Address (Sector Number,House Number),

City, State, Pin))

}

Here {} are used to enclose multivalued attributes and () are used to enclose composite attributes with comma separating individual attributes

Entity Types

- a collection (or set) of entities that have the same attributes
- Each entity type in the database is described by its name and attributes

Entity set

- collection of all entities of a particular entity type in the database at any point in time
- the entity set is usually referred to using the same name as the entity type

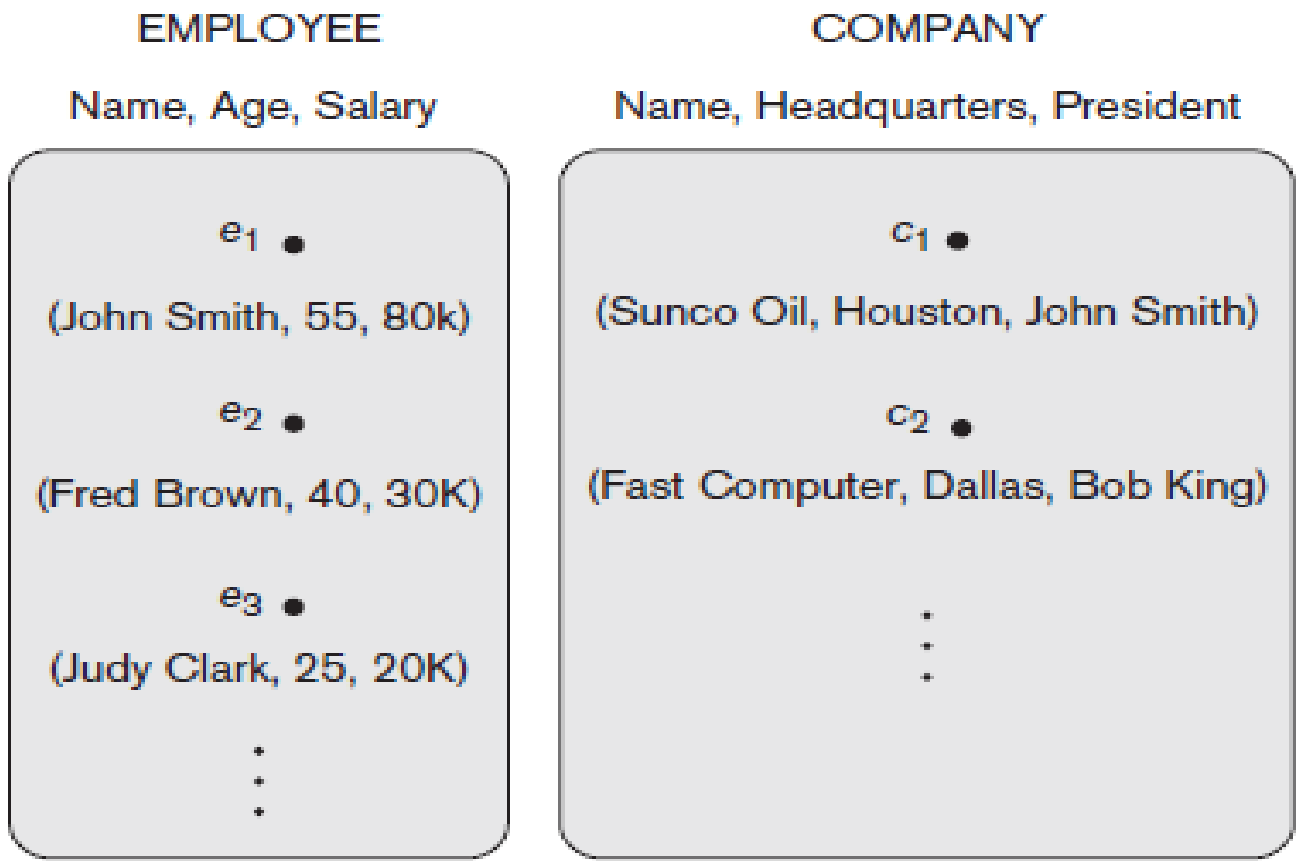


Figure: Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

- 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 the **extension** of the entity type.

- An **entity type** is represented in ER diagrams a **rectangular box** enclosing the entity type name
- **Attribute names** are enclosed in **ovals** and are attached to their entity type by straight lines
- **Composite attributes** are attached to their component attributes by straight lines.
- **Multivalued attributes** are displayed in **double ovals**

Key Attributes of an Entity Type

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

- For example, the Name attribute is a key of the COMPANY entity because no two companies are allowed to have the same name.
- In ER diagrammatic notation, each key attribute has its name underlined inside the oval
- Some entity types have more than one key attribute.
- For example, each of the Vehicle_id and Registration attributes of the entity type CAR is a key in its own right

➢ Example:

The CAR entity type with two key attributes, Registration and Vehicle_id.

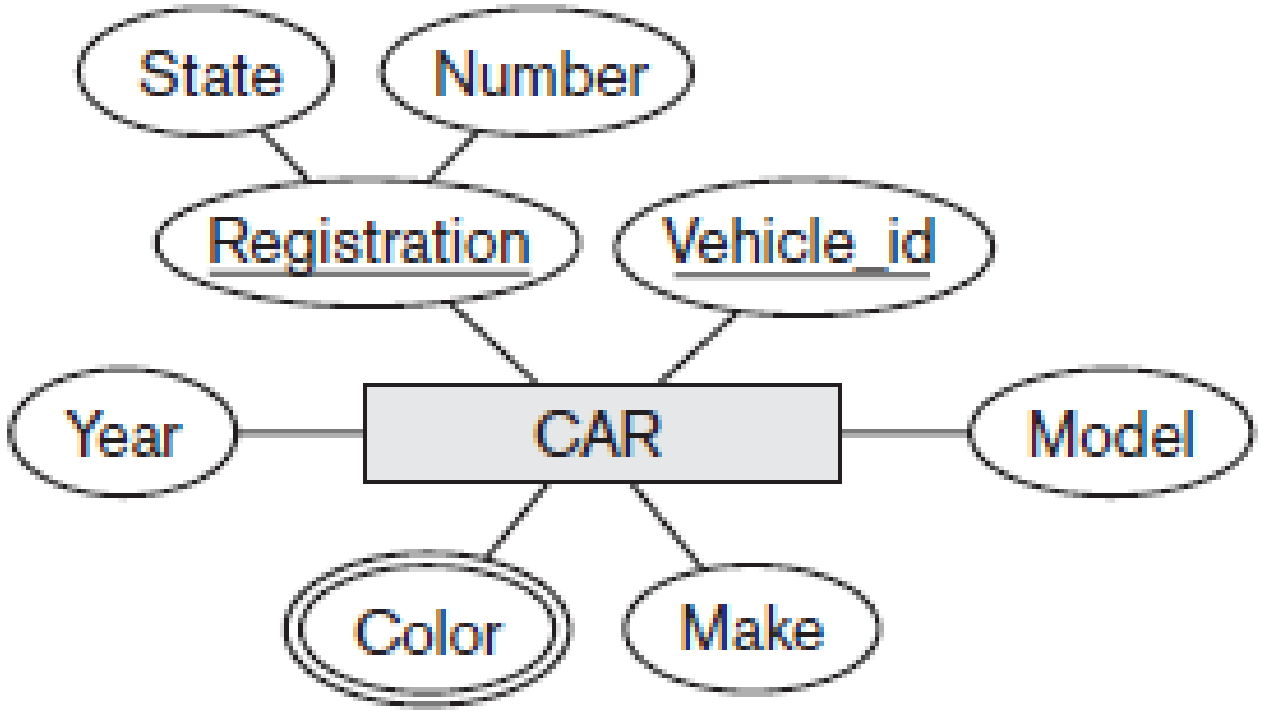


Figure : ER diagram notation.

➢ Value Sets (Domains) of Attributes

- set of values that may be assigned to that attribute for each individual entity
- For ex: if the range of ages allowed for employees is between 18 and 70, we can specify the value set of the Age attribute of EMPLOYEE to be the set of integer numbers between 18 and 70
- Value sets are not displayed in ER diagrams, and are specified using the basic data types available in most programming languages, such as integer, string, Boolean,

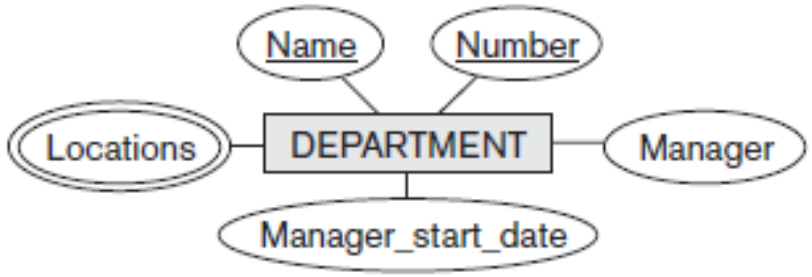
A Sample Database Application

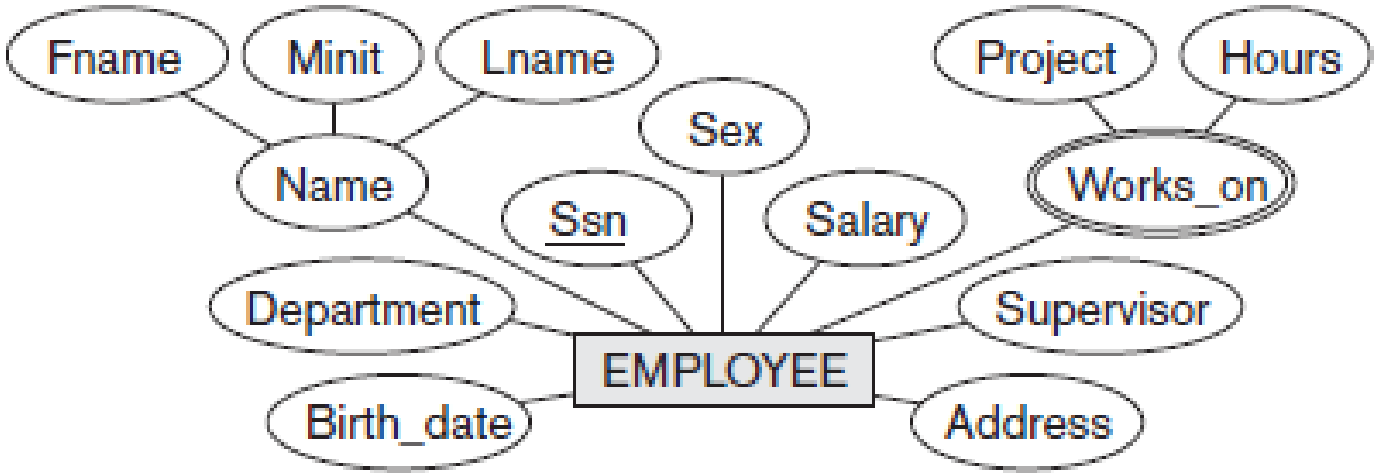
Miniworld : **COMPANY** database keeps track of a company's

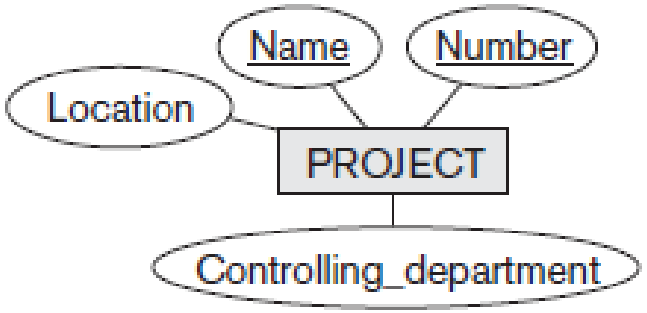
employees, departments, and projects.

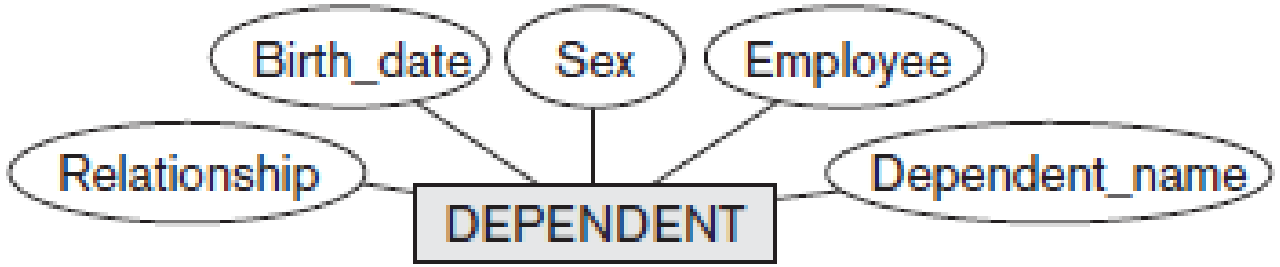
- After the requirements collection and analysis phase, the database designers provide the following description of the *miniworld:*
  - The company is organized into departments.
  - Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
  - A department controls a number of projects, each of which has a unique name, a unique number, and a single location.

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


Name
Number
Locations
DEPARTMENT
Manager
Manager_start_date


Fname
Minit
Lname
Name
Sex
Ssn
Salary
Project
Hours
Works_on
Department
EMPLOYEE
Supervisor
Birth_date
Address


Name
Number
Location
PROJECT
Controlling_department


Birth_date
Sex
Employee
Relationship
DEPENDENT
Dependent_name

Relationship Types, Relationship Sets, Roles, and Structural Constraints

Relationship

- An association among two or more entities
- Each relationship has a name

  ex: Teacher teaches Student

- For example
  - The attribute Manager of DEPARTMENT refers to an employee who manages the department
  - The attribute Controlling_department of PROJECT refers to the department that controls the project
- In the ER model, these references should not be represented as attributes but as **relationships**

- A **relationship type R** among n entity types E1, E2, ..., En defines a set of associations
- Each of the entity types E_1, E_2, ..., E_n is said to participate in the relationship type R
- similarly, each of the individual entities e_1, e_2, ..., e_n is said to participate in the relationship instance $r_i = (e_1, e_2, ..., e_n)$
- Informally, each relationship instance r_i in R is an association of entities, where the association includes exactly one entity from each participating entity type.

➢ Example: consider a relationship type WORKS_FOR between the two entity types EMPLOYEE and DEPARTMENT, which associates each employee with the department for which the employee works in the corresponding entity set.

➢ Each relationship instance in the relationship set WORKS_FOR associates one EMPLOYEE entity and one DEPARTMENT entity.

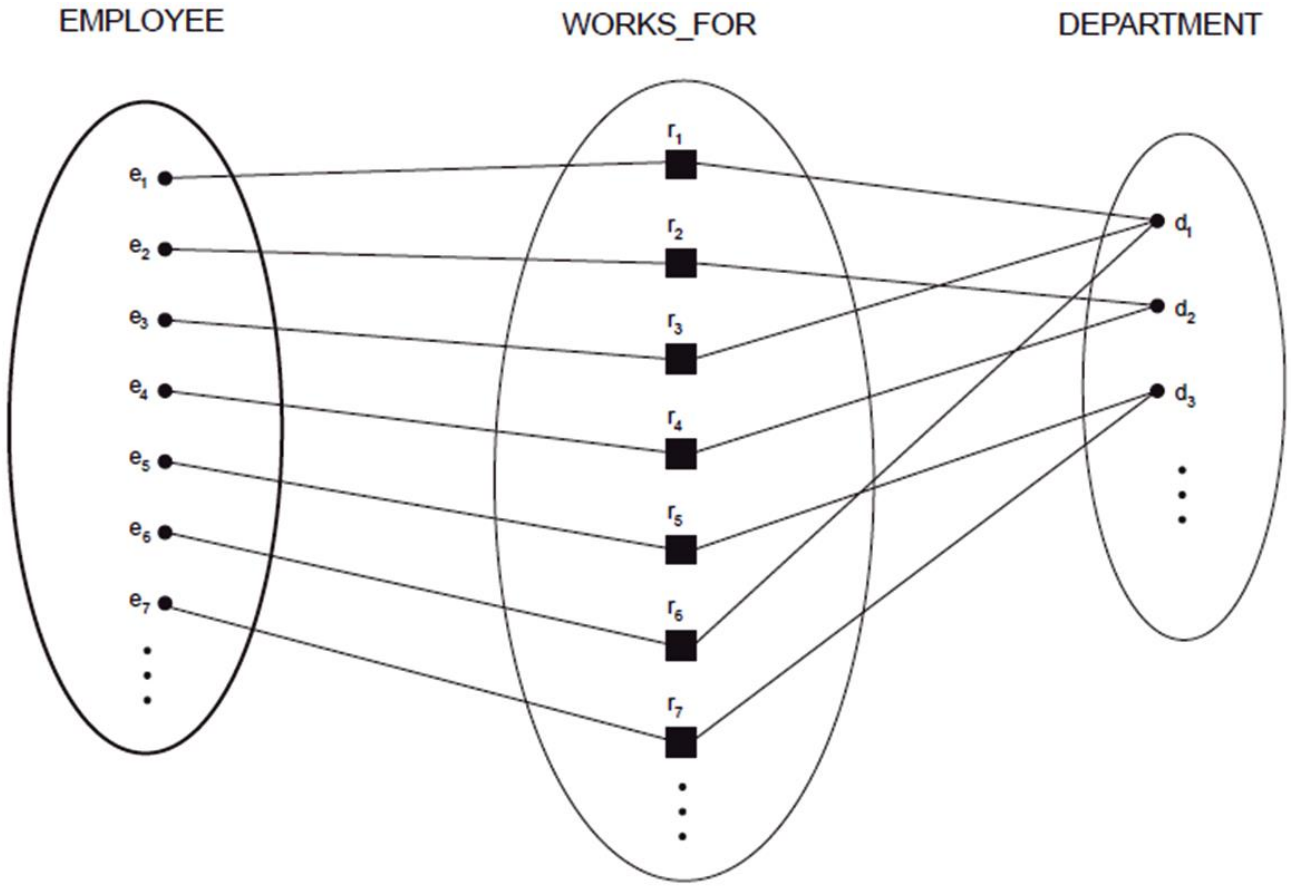


Figure : Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

- employees e_1 , e_3, and e_6 work for department d_1
- employees e_2 and e_4 work for department d_2 and
- employees e_5 and e_7 work for department d_3
- In ER diagrams, relationship types are displayed as diamond-shaped boxes, which are connected by straight lines to the rectangular boxes representing the participating entity types.
- The relationship name is displayed in the diamond-shaped box

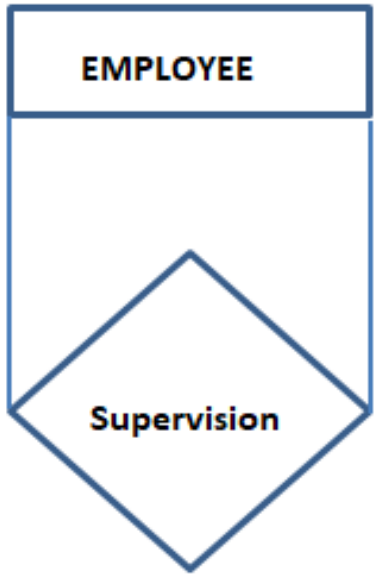
Relationship Degree, Role Names, and Recursive Relationships

➢Degree of a Relationship Type

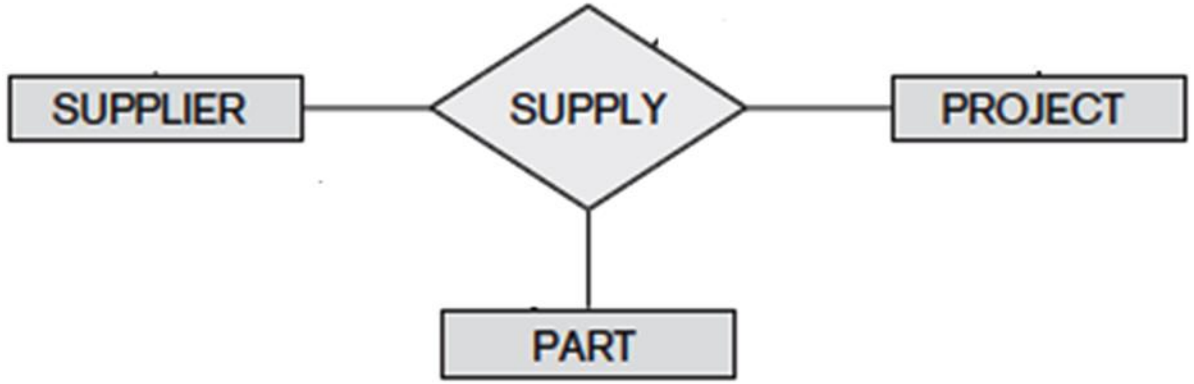
- Denotes the number of participating entity types.
- can be

1. Unary

ex:



-Binary



-Ternary

Role Names and Recursive Relationships

- The **role name** signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means.
- For example, in the WORKS_FOR relationship type, EMPLOYEE plays the role of employee or worker and DEPARTMENT plays the role of department or employer.
- Role names are not technically necessary in relationship types where all the participating entity types are distinct, since each participating entity type name can be used as the role name.

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

- The SUPERVISION relationship type relates an employee to a

  supervisor, where both employee and supervisor entities are members of the same EMPLOYEE entity set

- Hence, the EMPLOYEE entity type participates twice in SUPERVISION: once in the role of supervisor (or boss), and once in the role of supervisee (or subordinate)

- Each relationship instance r_i in SUPERVISION associates two employee entities e_j and e_k , one of which plays the role of supervisor and the other the role of supervisee.

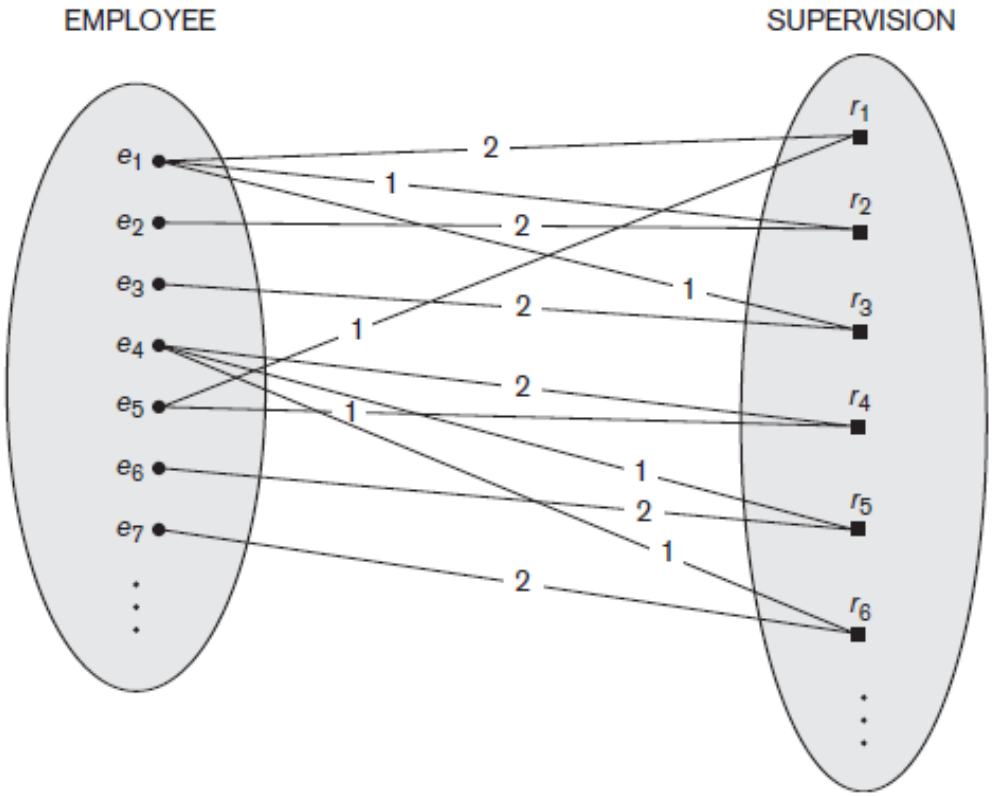


Figure : A recursive relationship SUPERVISION between EMPLOYEE in the supervisor role (1) and EMPLOYEE in the subordinate role (2).

Constraints on Binary Relationship Types

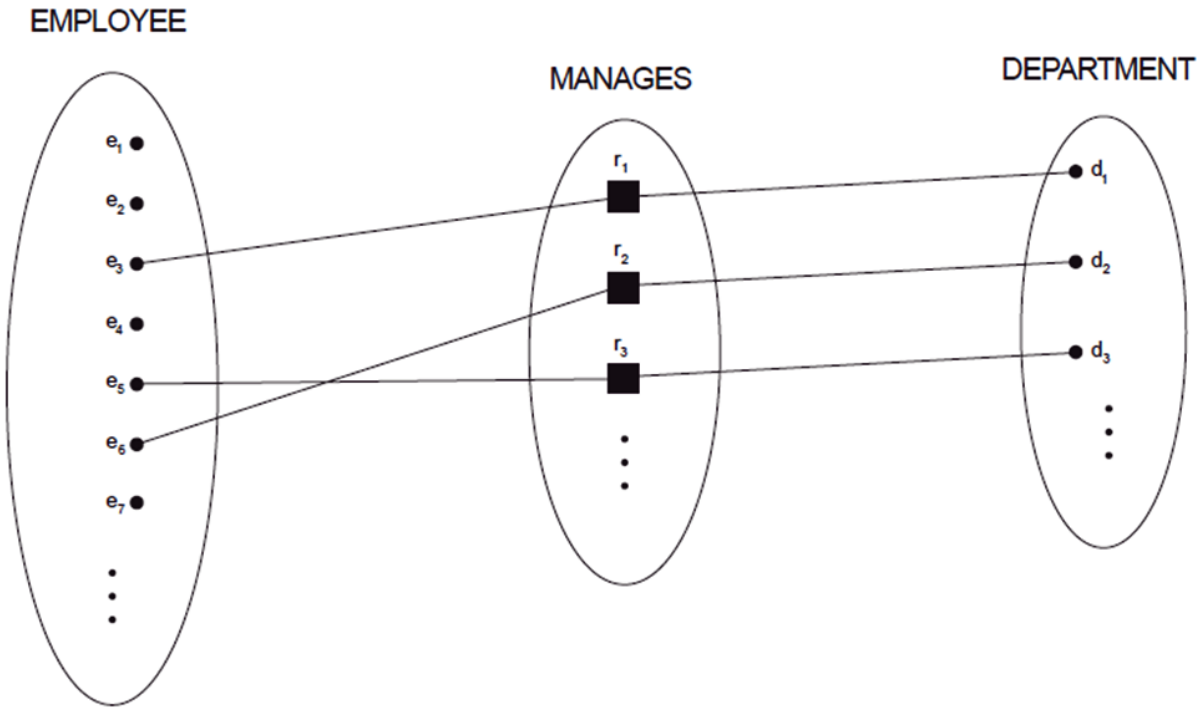
- Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set.
- These constraints are determined from the miniworld situation that the relationships represent.
- Two main types of binary relationship constraints:
  - cardinality ratio and
  - participation.

Cardinality Ratios for Binary Relationships:

- specifies the maximum number of relationship instances that an entity can participate in
- For example, 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.

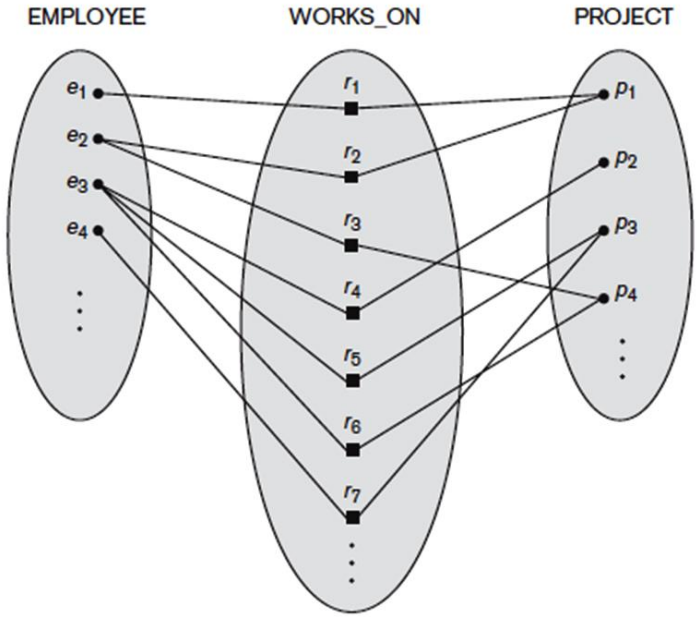
Example of a 1:1 binary relationship

- MANAGES which relates a department entity to the employee who manages that department
- This represents the miniworld constraints that—at any point in time—an employee can manage one department only and a department can have one manager only



Example of a M:N binary relationship

- The relationship type WORKS_ON is of cardinality ratio M:N, because the mini-world rule is that an employee can work on several projects and a project can have several employees.



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

Participation Constraints and Existence Dependencies.

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

  called the **minimum cardinality constraint**

- There are two types of participation constraints:
  - Total
  - 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 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

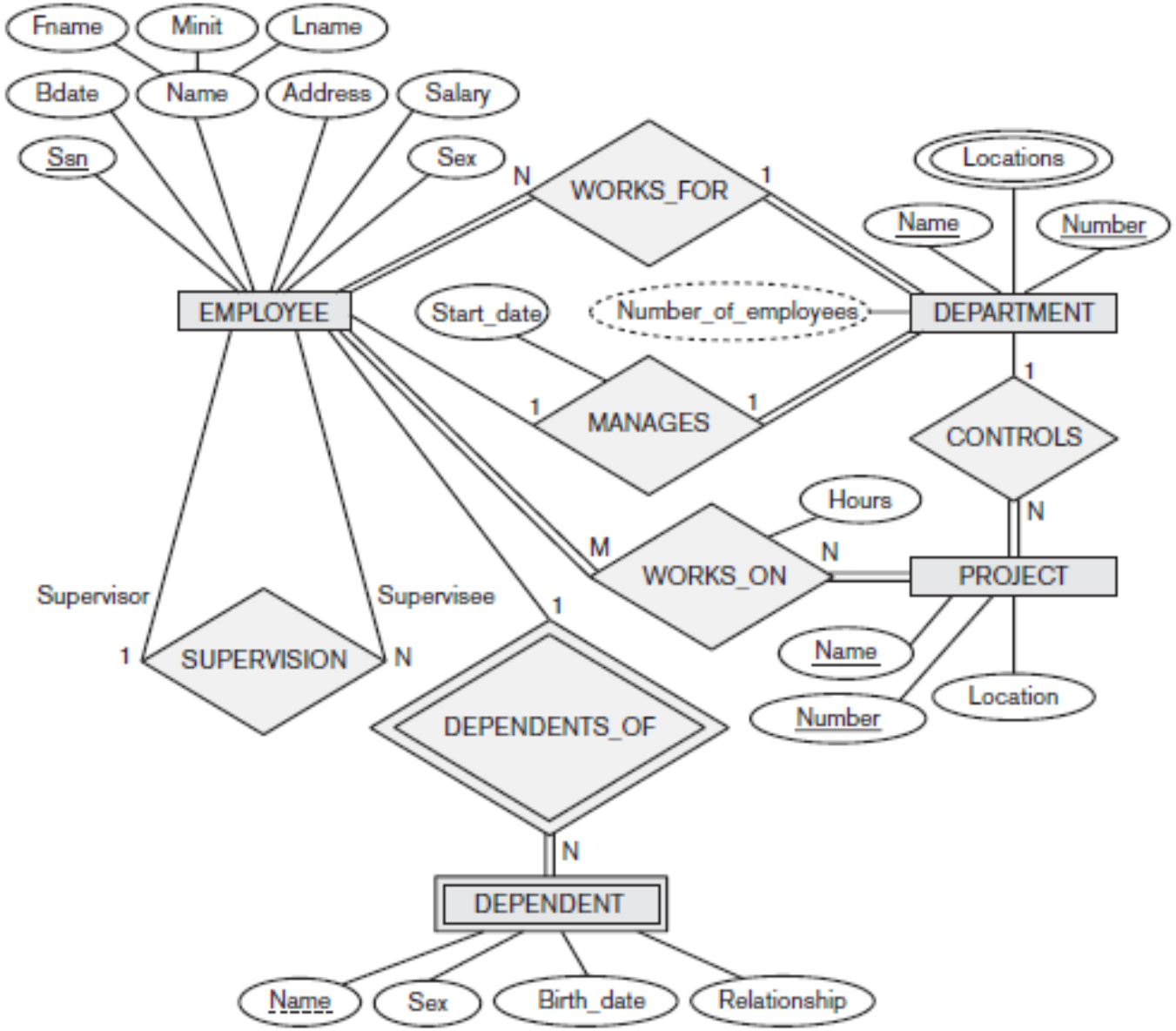
- we do not expect every employee to manage a department
- so the participation of EMPLOYEE in the MANAGES relationship type is partial, meaning that some or part of the set of employee entities are related to some department entity via MANAGES, but not necessarily all.

➢In ER diagrams, **total participation** is displayed as a **double line** connecting the participating entity type to the relationship, whereas **partial participation** is represented by a **single line**

➢**cardinality ratio + participation constraints**

=

structural constraints of a relationship type.


Fname
Minit
Lname
Name
Bdate
Address
Salary
Ssn
Sex
N
WORKS_FOR
1
Locations
Name
Number
EMPLOYEE
Start_date
Number_of_employees
DEPARTMENT
1
MANAGES
1
1
CONTROLS
Hours
N
M
WORKS_ON
N
PROJECT
Supervisor
Supervisee
1
1
SUPERVISION
N
Name
Location
Number
DEPENDENTS_OF
N
DEPENDENT
Name
Sex
Birth_date
Relationship

Attributes of Relationship Types

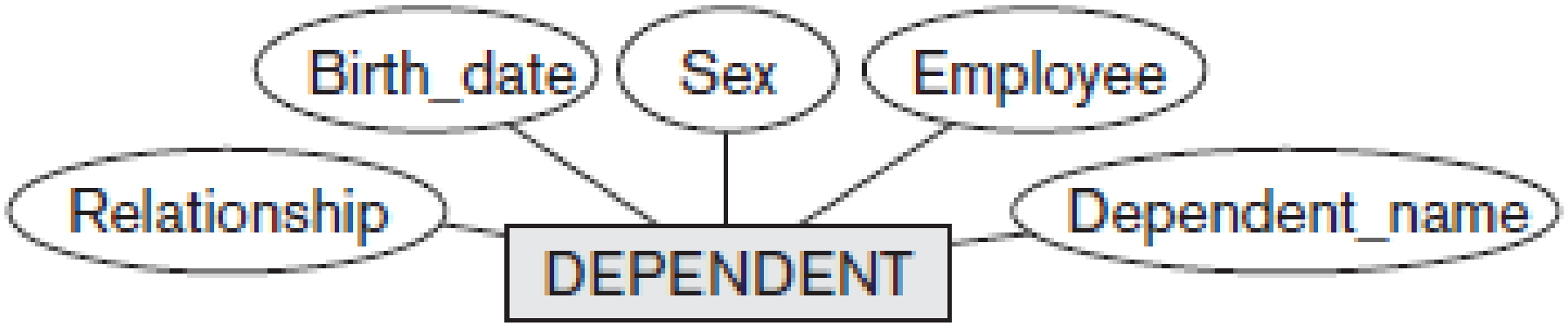
- Relationship types can also have attributes, similar to those of entity types.
- For example, to record the number of hours per week that an employee works on a particular project, we can include an attribute Hours for the WORKS_ON relationship type
- Another example is to include the date on which a manager started managing a department via an attribute Start_date for the MANAGES relationship type
- Attributes of 1:1 relationship types can be migrated to one of the participating entity types

- For a 1:N relationship type, a relationship attribute can be migrated *only* to the entity type on the N-side of the relationship.
- For M:N relationship types, some attributes may be determined by the combination of participating entities in a relationship instance, not by any single entity. Such attributes must be specified as relationship attributes.

.

Weak Entity Types

- Entity types that do not have key attributes of their own are called **weak 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
- We call this other entity type the **identifying** or **owner entity type**
- we call the relationship type that relates a weak entity type to its owner the **identifying relationship** of the weak entity type


Birth_date
Sex
Employee
Relationship
DEPENDENT
Dependent_name

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

- A weak entity type always has a total participation constraint (existence dependency) with respect to its identifying relationship because a weak entity cannot be identified without an owner entity.
- A weak entity type normally has a **partial key**, which is the attribute that can uniquely identify weak entities that are related to the same owner entity.
- In our example, if we assume that no two dependents of the same employee ever have the same first name, the attribute Name of DEPENDENT is the partial key.
- In ER diagrams, both a weak entity type and its identifying relationship are distinguished by surrounding their boxes and diamonds with double lines.
- The partial key attribute is underlined with a dashed or dotted line.

ER Diagrams, Naming Conventions, and Design Issues

- Summary of Notation for ER Diagrams
- Proper Naming of Schema Constructs
- Design Choices for ER Conceptual Design
- Alternative Notations for ER Diagrams

Example: A UNIVERSITY Database

- The university is organized into colleges (COLLEGE), and each college has unique name (CName), a main office (COffice) and phone (CPhone), and particular faculty member who is dean of the college. Each college admini ters a number of academic departments (DEPT). Each department has unique name (DName), a unique code number (DCode), a main offi (DOffice) and phone (DPhone), and a particular faculty member who chai the department. We keep track of the start date (CStartDate) when that fa ulty member began chairing the department.

- A department offers a number of courses (COURSE), each of which has a unique course name (CoName), a unique code number (CCode), a course level (Level: this can be coded as 1 for freshman level, 2 for sophomore, 3 for junior, 4 for senior, 5 for MS level, and 6 for PhD level), a course credit hours (Credits), and a course description (CDesc). The database also keeps track of instructors (INSTRUCTOR); and each instructor has a unique identifier (Id), name (IName), office (IOffice), phone (IPhone), and rank (Rank); in addition, each instructor works for one primary academic department.

- The database will keep student data (STUDENT) and stores each student's name (SName, composed of first name (FName), middle name (MName), last name (LName)), student id (Sid, unique for every student), address (Addr), phone (Phone), major code (Major), and date of birth (DoB). A student is assigned to one primary academic department.

1. Explain with a neat diagram, the phases of database design.
2. Explain the following terms with an example for each:

   i) Entity ii) attribute iii) entity type iv)entity set

   v) key attribute vi) value set v)degree of a relationship type

   vi) role names vii) cardinality ratio

   viii) participation constraints
3. Explain the different types of attributes that occur in an ER model with an example.
4. Explain recursive relationship type with an example.
5. What is a weak entity type? Explain the role of partial key in the design of weak entity type.
6. List and explain symbols used in ER diagram and their meaning.

7. Discuss the naming conventions used for ER schema diagrams.

8. Design an ER diagram for an insurance company. Assume suitable entity types like CUSTOMER, AGENT, BRANCH, POLICY, PAYEMENT and the relationship between them.

9. Design an ER - diagram for the Movie - database considering the following requirements:

i) Each Movie is identifies by its title and year of release, it has length in minutes and can have zero of more quotes, language.

ii) Production companies are identified by Name, they have address, and each production company can produce one or more movies.

iii) Actors are identified by Name and Date of Birth, they can act in one or more movies and each actor has a role in a movie.

iv) Directors are identified by Name and Date of Birth, so each Director can direct one or more movie and each movie can be directed by one or more Directors.

v) Each movie belongs to anyone category like Horror, action, Drama, etc.

10. Draw a ER-Diagram of movie database. Assume your own entities(minimum 4) attributes and relationships.
11. Draw an ER diagram for BANK database schema with atleast five entity types. Also specify primary key and structural constraints

College database

- Each college contains many departments
- Each department can offer any number of courses.
- For each departments there is a head
- Many instructors can work in a department
- An instructors can work only in one department
- An instructor can take any number of courses
- A student can enroll for any number of courses


THANK YOU