

Database Management System

(18IS5DCDBM)

Unit 1

(Introduction, ER Model)

Mrs. Bhavani K
Assistant Professor
Dept. of ISE, DSCE

Unit 1

- **Introduction**
 - Introduction
 - An Example
 - Characteristics of the Database Approach
 - Advantages of Using the DBMS Approach
 - When Not to Use a DBMS
 - Data Models, Schemas, and Instances
 - Three-Schema Architecture and Data Independence
 - The Database System Environment
- **Entity–Relationship (ER) Model**
 - A Sample Database Application
 - Entity Types, Entity Sets, Attributes, and Keys
 - Relationship Types, Relationship Sets, Roles, and Structural Constraints
 - Weak Entity Types
 - ER Diagrams, Naming Conventions, and Design Issues

Introduction

Introduction

- Storing data and retrieving information has been and is essential for business.
- Data refers to raw facts with implicit meaning
 - Data can be anything such as name of a person, a number, an image, sound etc.
 - Example: 'Monica', 'ISE', 'Student', 'Programming', 'Good'
- When the data is processed and converted into a meaningful and useful form, it is known as information.
 - Example: Monica is ISE student and she is good in programming.
- Fast access to the information is essential for making important decisions.

Basic Definitions

- **Database:**
 - A collection of related data or an organized collection of data.
 - **Data:**
 - Known facts that can be recorded and have an implicit meaning.
 - **Mini-world:**
 - Some part of the real world about which data is stored in a database.
For example, student grades at a university.
 - **Database Management System (DBMS):**
 - A software package/ system that facilitates the creation and maintenance of a computerized database.
 - **Database System:**
 - The DBMS software together with the data itself. Sometimes, the applications are also included.
- Database (DB) + Database Management system(DBMS) = Database System(DS)**

Typical DBMS Functionality

- **Define** a particular database by specifying its data types, structures, and constraints of the data
- **Construct** or Load the initial database by storing the data on a secondary storage medium that is controlled by the DBMS.
- **Manipulating** the database: by
 - Retrieval: Querying, generating reports
 - Modification: Insertions, deletions and updates to its content
 - Accessing the database through Web applications
- **Processing and Sharing** by allowing a set of concurrent users and application programs – yet, keeping all data valid and consistent

Typical DBMS Functionality

- Other features:
 - Protection or Security measures to prevent hardware or software malfunction and unauthorized access
 - Presentation and Visualization of data
 - Maintaining the database and associated programs over the lifetime of the database application (over a long period of time)
 - Called database, software, and system maintenance

Simplified database system environment

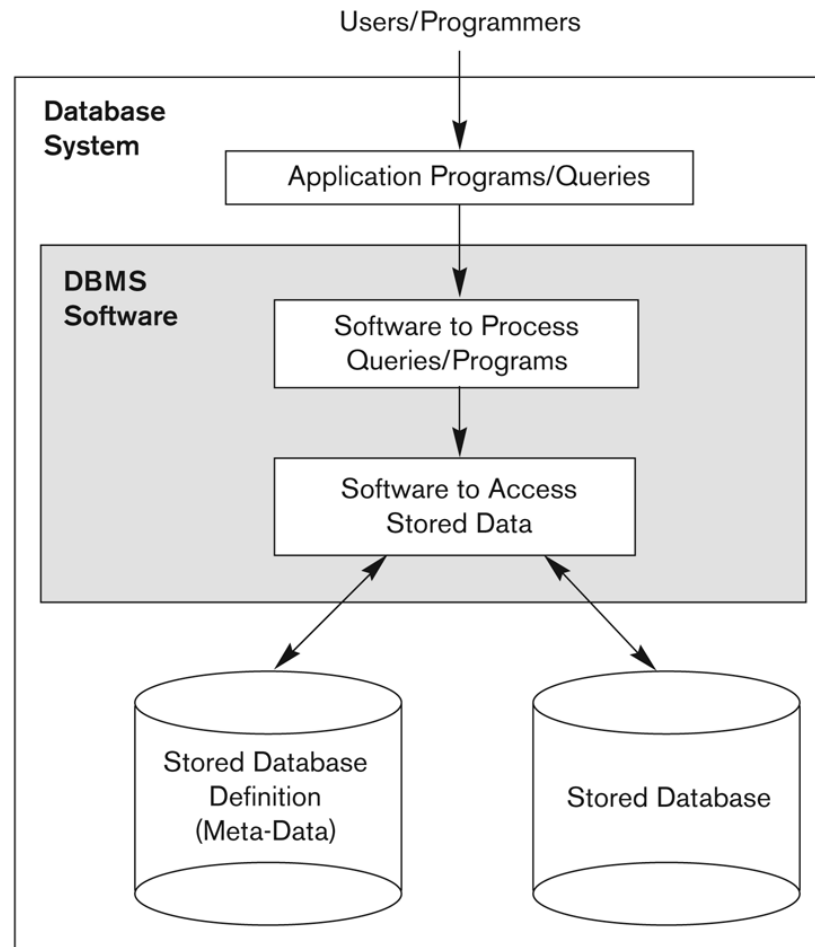


Figure 1.1
A simplified database system environment.

Example of a Database

- **Mini-world for the example:**
 - Part of a UNIVERSITY environment.
- **Some mini-world *entities*:**
 - STUDENTs
 - COURSEs
 - SECTIONs (of COURSEs)
 - (academic) DEPARTMENTs
 - INSTRUCTORs

Example of a Database (with a Conceptual Data Model)

- **Some mini-world *relationships*:**
 - SECTIONs *are of specific* COURSEs
 - STUDENTs *take* SECTIONs
 - COURSEs *have prerequisite* COURSEs
 - INSTRUCTORs *teach* SECTIONs
 - COURSEs *are offered by* DEPARTMENTs
 - STUDENTs *specialize in* DEPARTMENTs

Note: The above entities and relationships are typically expressed in a conceptual data model, such as the ENTITY-RELATIONSHIP data model

Main Characteristics of the Database Approach

- **Self-describing nature of a database system:**
 - A DBMS **catalog** stores the description of a particular database (e.g. data structures, types, and constraints)
 - The description is called **meta-data(i.e. the information contained in the catalog)**.
 - This allows the DBMS software to work with different databases
- **Insulation between programs and data:**
 - Called **program-data independence**.
 - Allows changing data structures and storage organization without having to change the DBMS access programs.

Example of a simplified database catalog

RELATIONS

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

Figure 1.3

An example of a database catalog for the database in Figure 1.2.

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

Main Characteristics of the Database Approach (continued)

- **Data Abstraction:**
 - The characteristic that allows program-data independence and program-operation independence
 - Program-operation independence
 - User application programs can operate on the data by invoking operations through their names and arguments, regardless of how the operations are implemented
 - A **data model** is used to hide storage details and present the users with a conceptual view of the database.
 - Programs refer to the data model constructs rather than data storage details
- **Support of multiple views of the data:**
 - Each user may see a different view of the database, which describes **only** the data of interest to that user.

Main Characteristics of the Database Approach (continued)

- **Sharing of data and multi-user transaction processing:**
 - Allowing a set of **concurrent users** to retrieve from and to update the database.
 - A **Transaction** is
 - An execution program or process that comprise one or more database accesses, such as reading or updating of records.

Sharing of Data and Multi-user Transaction Processing (continued)

- A DBMS must enforce following transaction properties
 - Isolation
 - Each transaction appears to execute in isolation from other transactions.
 - Atomicity
 - Either all the database operations in a transaction are executed or none are.
- *Concurrency control* within the DBMS guarantees that each **transaction** is correctly executed or aborted
- *Recovery* subsystem ensures each completed transaction has its effect permanently recorded in the database
- **OLTP** (Online Transaction Processing) is a major part of database applications. This allows hundreds of concurrent transactions to execute per second.

Differences Between File Processing Approach and Database System Approach

File based system	Database system
1. The data and program are inter-dependent.	1. The data and program are independent of each other.
2. File-based system causes data redundancy. The data may be duplicated in different files	2. Database system control data redundancy.
3. File –based system causes data inconsistency. The data in different files may be different that cause data inconsistency.	3. In database system data is always consistent.
4. The data cannot be shared because data is distributed in different files.	4. In database, data is easily shared because data is stored at one place.
5. In file based system data is widely spread. Due to this reason file based system provides poor security.	5. It provides many methods to maintain data security in the database.

Database Users

- End users
 - Use the database system to realize some goal
- Application developers
 - Write software to allow end users to interface with the database system
- Database Administrator (DBA)
 - To create and implement the technical controls and security to enforce policy decisions in the database system.
- Database systems programmer
 - Writes the database software itself

Advantages of Using the Database Approach

- Controlling redundancy in data storage and in development and maintenance efforts.
- Sharing of data among multiple users.
- Restricting unauthorized access to data.
- Providing persistent storage for program Objects
- Providing Storage Structures for efficient Query Processing
- Providing backup and recovery services.
- Providing multiple interfaces to different classes of users.
- Representing complex relationships among data.
- Enforcing integrity constraints on the database.
- Drawing Inferences and Actions using rules

Disadvantages of using a DBMS

- High initial investment and possible need for additional hardware, software and training.
- The generality that a DBMS provides for defining and processing data.
- Overhead for providing generality, security, concurrency control, recovery, and integrity functions.

When not to use a DBMS

- **When a DBMS may be unnecessary:**
 - If the database and applications are simple, well defined, and not going to change often.
 - If there are strict real-time requirements that cannot be met because of DBMS overhead.
 - If multiple users access to data is not required
 - If the data manipulation is very much specialized.
- **When no DBMS may suffice:**
 - If the database system is not able to handle the complexity of data because of modeling limitations
 - If the database users need special operations not supported by the DBMS.

Data Models

- **Data Model:** A set of concepts to express the *structure* of a database, and certain *constraints* that the database should obey.
- **Data Model Operations:** Operations for specifying database retrievals and updates by referring to the concepts of the data model.
 - Operations on the data model may include
 - *basic operations*
 - Ex: Insert, Delete, Update
 - *user-defined operations*
 - Ex: Compute_Student_Aggr

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.
- **Implementation (representational)** data models: Provide concepts that fall between the above two, balancing user views with some computer storage details.

Schemas versus Instances

- **Database Schema:** The *description* of a database.
 - Includes descriptions of the database structure and the constraints that should hold on the database.
- **Schema Diagram:** A diagrammatic display of (some aspects of) a database schema.
- **Schema Construct:** A component of the schema or an object within the schema.
 - e.g., STUDENT, COURSE
- **Database Instance:** The actual data stored in a database at a *particular moment in time*.
 - Also called **database state** (or **occurrence**).

Schema Diagram

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
----------------	--------------------	-------

Database Schema Vs. Database State

- **Database State:** Refers to the content of a database at a moment in time.
- **Initial Database State:** Refers to the database when it is loaded
- **Valid State:** A state that satisfies the structure and constraints of the database.
- **Differences**
 - The **database schema** changes *very infrequently*. The **database state** changes *every time the database is updated*.
 - **Schema** is also called **intension**, whereas **state** is called **extension**.

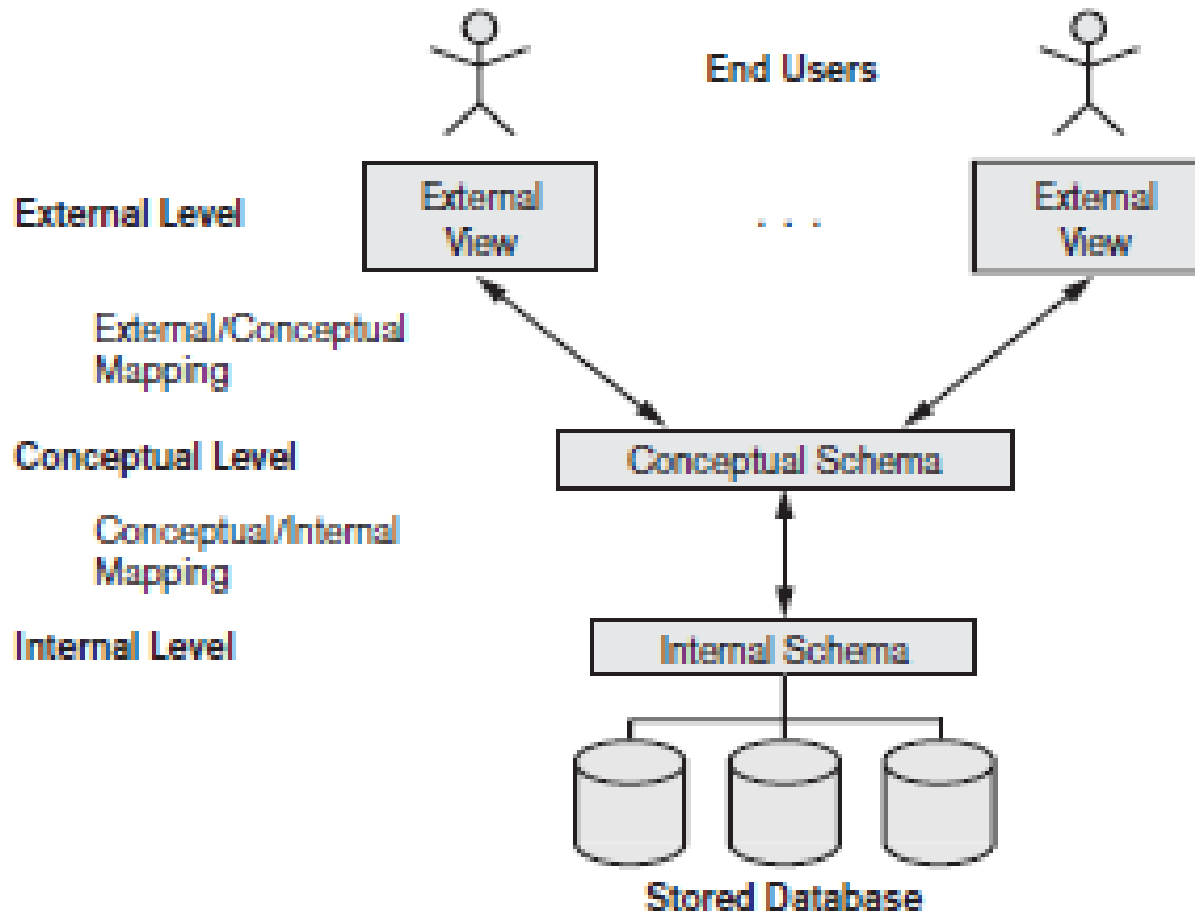
Three-Schema Architecture

- Proposed to support DBMS characteristics of:
 - **Program-data independence.**
 - Support of **multiple views** of the data.
 - Use of a catalog to store the database description

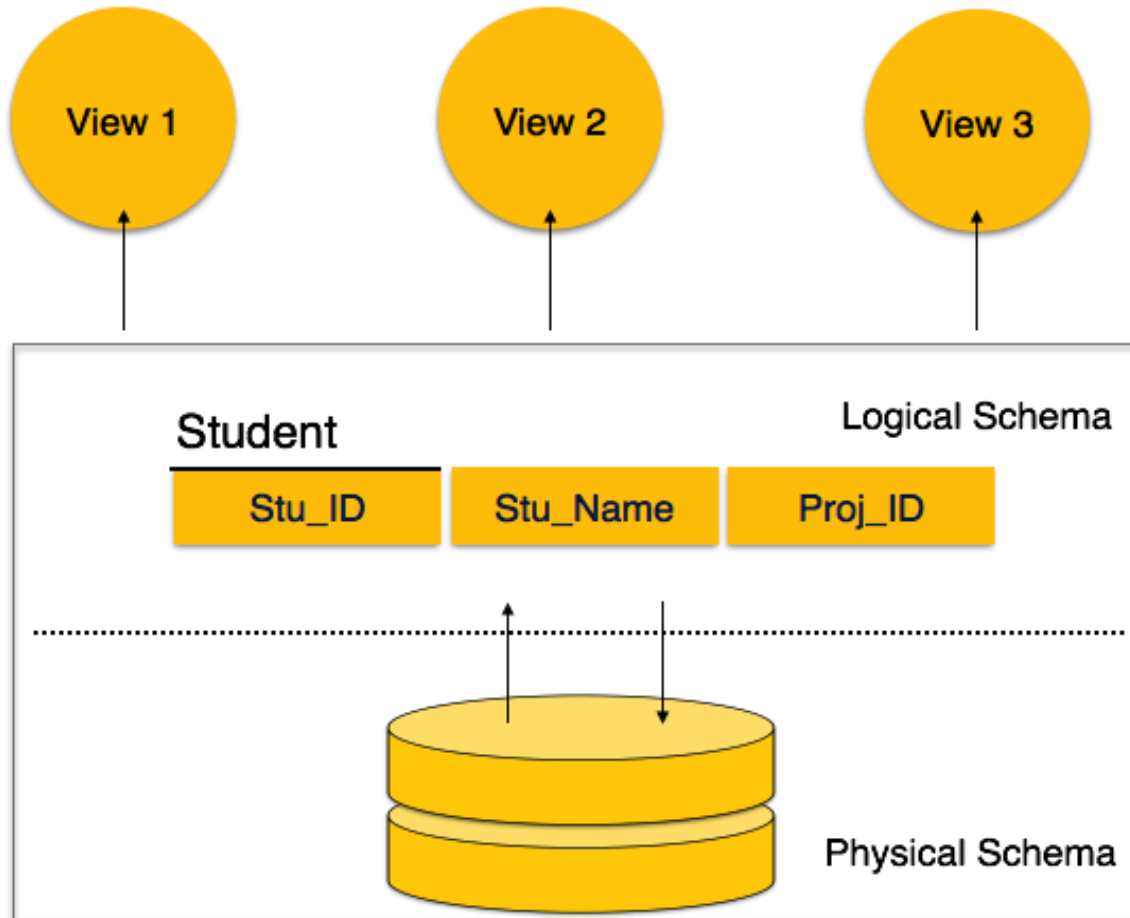
Three-Schema Architecture

- Defines DBMS schemas at *three levels*:
 - **Internal schema** at the internal level to describe physical storage structures and access paths. Typically uses a *physical* data model.
 - **Conceptual schema** at the conceptual level to describe the structure and constraints for the *whole* database for a community of users. Uses a *conceptual* or an *implementation* data model.
 - **External schemas** at the external level to describe the various user views. Usually uses the same data model as the conceptual level.

Three-Schema Architecture



Three-Schema Architecture: Example



Three-Schema Architecture

- **Mappings** among schema levels are needed to transform requests and data. Programs refer to an external schema, and are mapped by the DBMS to the internal schema for execution.

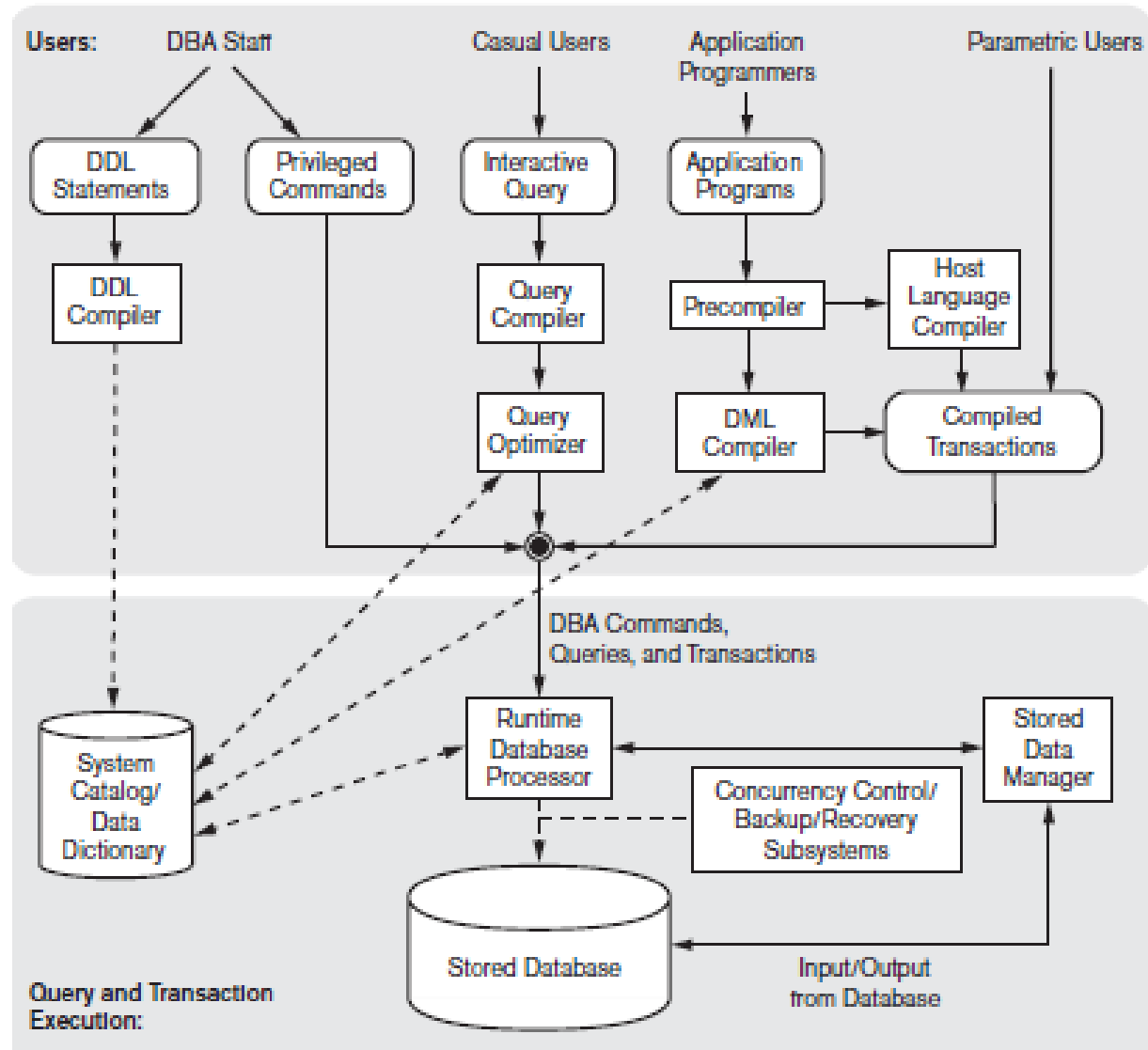
Data Independence

- **Logical Data Independence:** The capacity to change the conceptual schema without having to change the external schemas and their application programs.
- **Physical Data Independence:** The capacity to change the internal schema without having to change the conceptual schema.

Data Independence

- When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed in a DBMS that fully supports data independence. The higher-level schemas themselves are *unchanged*. Hence, the application programs need not be changed since they refer to the external schemas.

The Database System Environment



The Database System Environment

- The figure is divided into two parts:
 - The top part
 - various users of the database environment and their interfaces.
 - The lower part
 - internals of the DBMS responsible for storage of data and processing of transactions.

Entity–Relationship (ER) Model

DATABASE ARCHITECTURE

- External level – concerned with the way individual users see the data
- Conceptual level – a formal description of data of interest / database to the organization
- Internal level – concerned with the way in which the data is actually stored

Example : COMPANY Database

- Requirements of the Company (illustrative):
 - 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.
 - Each department *controls* a number of PROJECTs. Each project has a name, number and is located at a single location.

Example : COMPANY Database contd...

- We store each EMPLOYEE's social security number, address, salary, gender, and birthdate. Each employee *works for* one department but may *work on* several projects. We keep track of the number of hours per week that an employee currently works on each project. We also keep track of the *direct supervisor* of each employee.
- Each employee may *have* a number of DEPENDENTS. For each dependent, we keep track of their name, sex, birthdate, and relationship to employee.

ER Model Concepts

- Entities and Attributes
 - Entities are specific objects or things in the mini-world that are represented in the database.
 - For example the EMPLOYEE Raghu Ram, the Research DEPARTMENT, the ProductX PROJECT
 - Attributes are properties used to describe an entity.
 - For example an EMPLOYEE entity may have a 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='Raghu Ram', SSN='55555', Address='55, KS layout, Bangalore', Gender='M', BirthDate='05-JAN-95'
 - Each attribute has a *value set* (or data type) associated with it
 - e.g. integer, string, char, ...

ENTITY

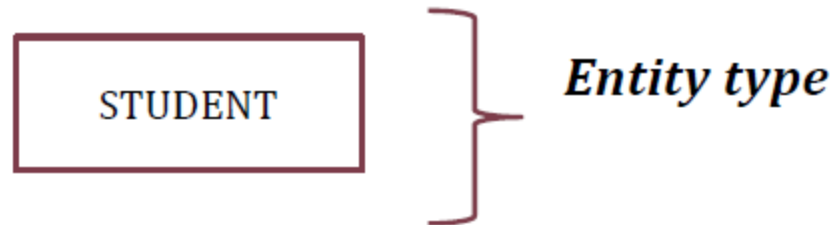
- It is a real world item / concept that can exist on it's own.
- It may be an object with physical existence (person, house) or it may be an object with conceptual existence (company ,job, university course)

ENTITY TYPE

- Entity type defines collection of entities that have same attribute.
- Entity type in a database is defined by it's name and attribute.
- Entity instance is a single occurrence of an entity type.

ENTITY SET

- Collection of entities of a particular entity type in a database at any point of time is called entity set
- Entity set is usually referred to by same name as the entity type



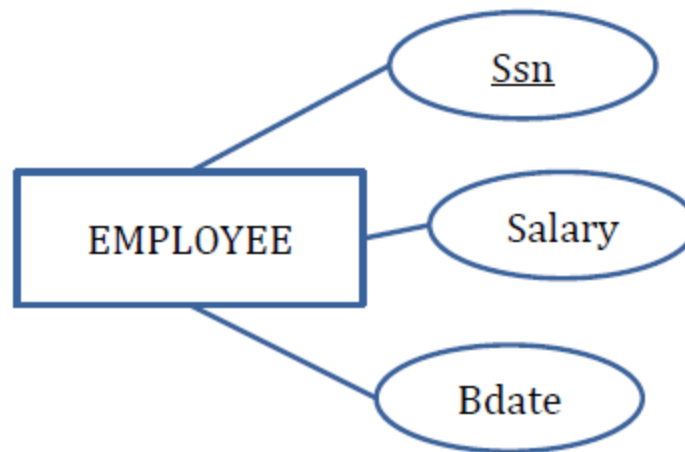
- Entity type is represented as rectangle enclosing the type name which is *singular noun*.

Entity and its Attributes

- An entity is a real-world object either living or non-living thing that has an independent existence.
- Example : College Database
 - » Teacher
 - » Student
 - » Class
 - » Course
- Attributes :
 - properties of an entity that characterize and describe it.
 - Example: Student entity has the attributes USN, Name, Email_id, CGPA
 - Each attribute accepts a value from a set of permitted values called the domain or the value set of the attribute.
 - Example: domain of CGPA – set of numbers between 1 and 10.

ATTRIBUTE

- Attributes are the *properties that describe the entities*.
- Attribute names are enclosed by ovals and connected to their entities by single line.
- Set of attribute values of a given attribute is *the value set or domain*



Types of Attributes

- 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).
 - Composition may form a hierarchy where some components are themselves composite.
- Multi-valued
 - An entity may have multiple values for that attribute.
 - For example, Color of a CAR or PreviousDegrees of a STUDENT. Denoted as {Color} or {PreviousDegrees}.

Types of Attributes contd...

- 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)}.

Types of Attributes contd...

SINGLE VALUED ATTRIBUTE <ul style="list-style-type: none">• Attributes having single value for particular entity.<ul style="list-style-type: none">• Ex - Age	MULTI VALUED ATTRIBUTE <ul style="list-style-type: none">• Attribute having set of values• Denoted by double circled oval• Ex: Phone-number, Collegedegree
DERIVED ATTRIBUTE <ul style="list-style-type: none">• Attribute values are derived from another attribute.• Denoted by dotted oval<ul style="list-style-type: none">• Ex - Age	STORED ATTRIBUTE <ul style="list-style-type: none">• Attributes from which the values of other attributes are derived<ul style="list-style-type: none">• Ex: Bdate

Entity Types and Key Attributes

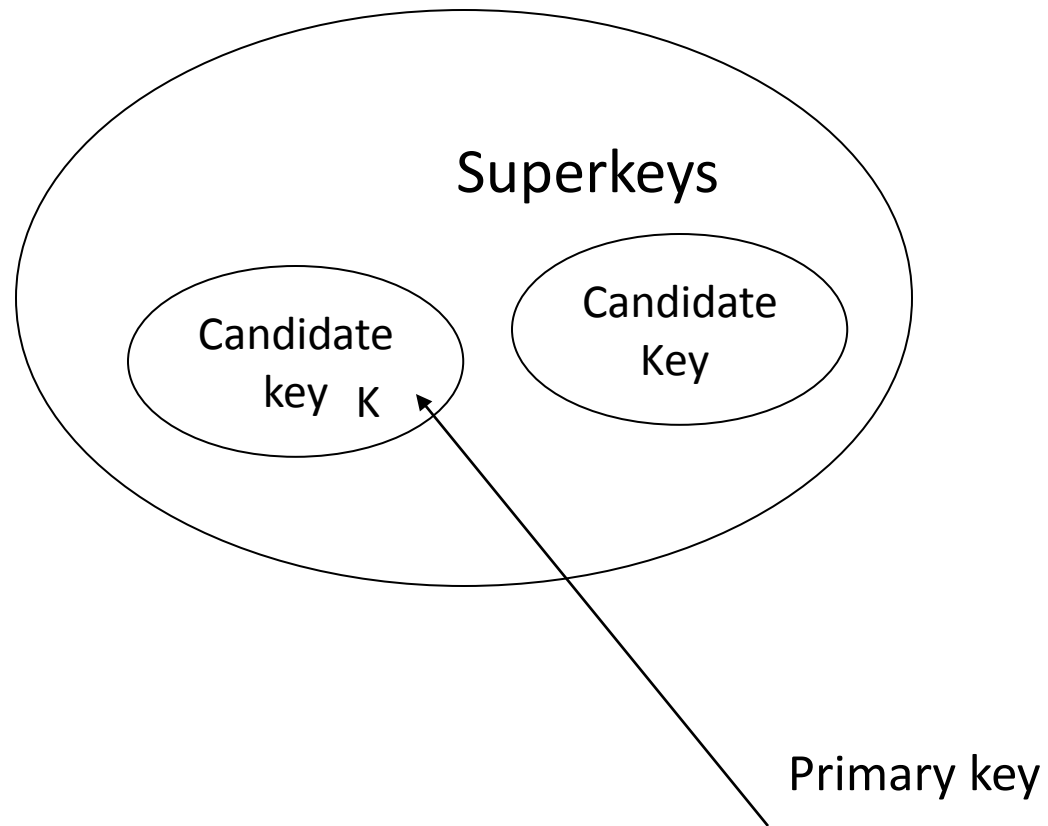
- Entities with the same basic attributes are grouped or typed into an entity type.
 - For example, the EMPLOYEE entity type or the PROJECT entity type.
- An attribute of an entity type for which each entity must have a unique value is called a key attribute of the entity type.
 - For example, SSN of EMPLOYEE.
- A key attribute may be composite.
 - For example, VehicleTagNumber is a key of the CAR entity type with components (Number, State).
- An entity type may have more than one key.
 - For example, the CAR entity type may have two keys:
 - VehicleIdentificationNumber (popularly called VIN) and
 - VehicleTagNumber (Number, State), also known as license_plate number.

IDENTIFIER ATTRIBUTE OR KEY ATTRIBUTE

<i>Key</i>	<ul style="list-style-type: none">• Data item that allows us to uniquely identify individual occurrences or an entity type.
<i>Superkey</i>	<ul style="list-style-type: none">• Attribute or set of attributes that uniquely identify a tuple.
<i>Candidate key</i>	<ul style="list-style-type: none">• Minimal super key with the property of irreducability and uniqueness
<i>Primary key</i>	<ul style="list-style-type: none">• An entity type may have one or more possible candidate keys, the one which is selected as primary key.
<i>Composite key</i>	<ul style="list-style-type: none">• candidate key that consisting of two or more attributes
<i>Foreign key</i>	<ul style="list-style-type: none">• An attribute or set of attribute that matches the candidate key or other or same relation

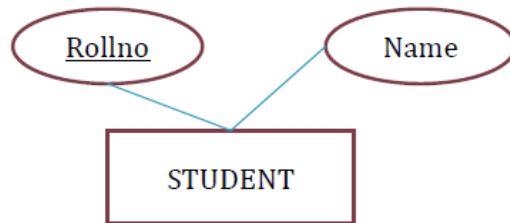
- The name of each primary key attribute is underlined.

Keys

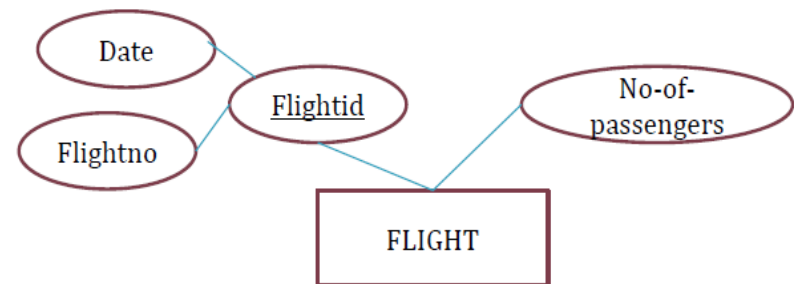


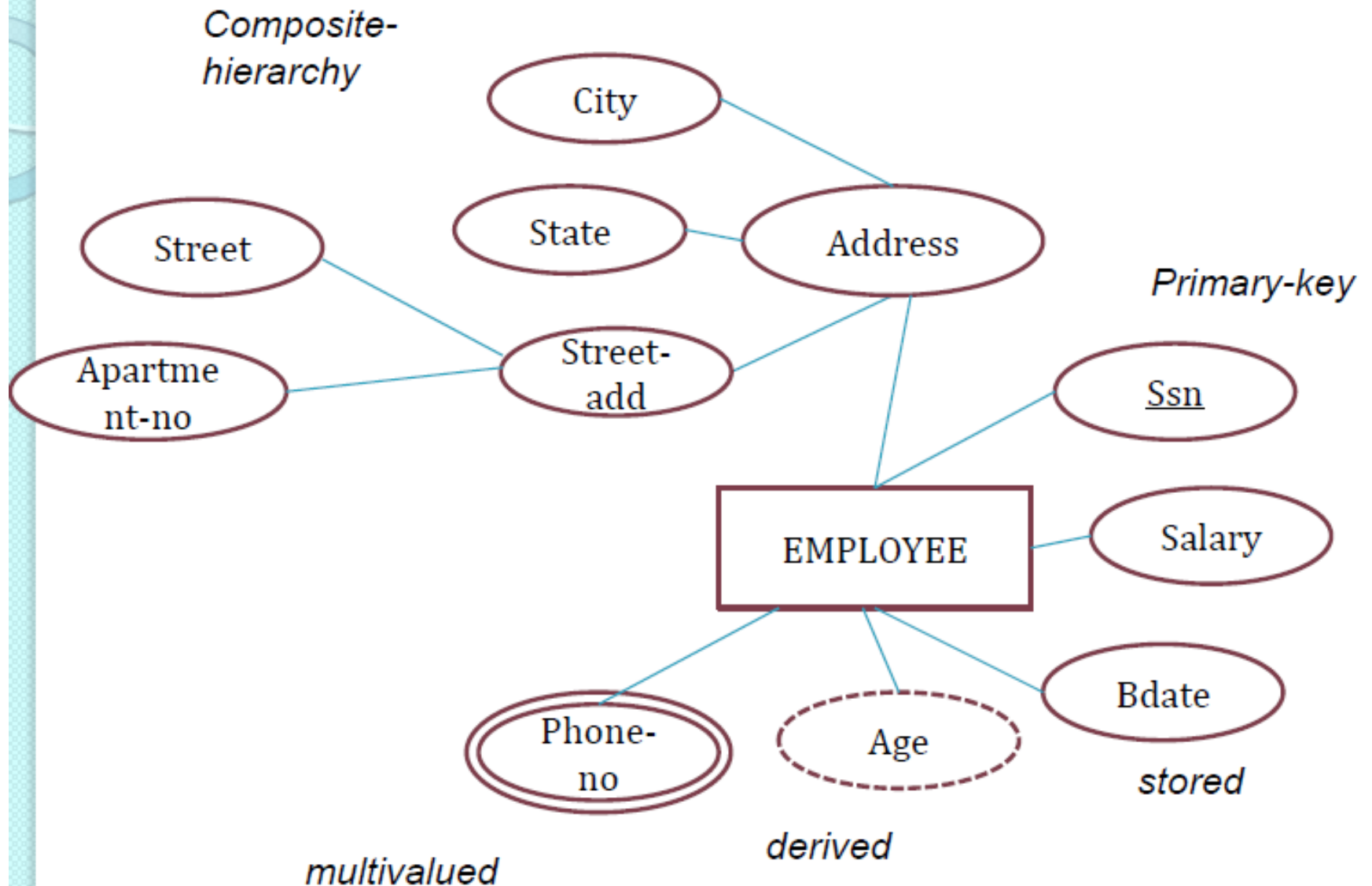
KEY ATTRIBUTE

SIMPLE KEY ATTRIBUTE



COMPOSITE KEY ATTRIBUTE





RELATIONSHIP

- The association among entities is called a **relationship**.
- For example,
 - an employee **Works-For** a department
 - a student **enrolls** in a course
 - Here, Works-For and Enrolls are called relationships.
- **Relationship Set**
 - A set of relationships of similar type is called a relationship set.
 - Like entities, a relationship too can have attributes.

RELATIONSHIP contd..

- The relationship is often denoted by diamond symbol and are usually verbs.
- Each relationship instance in relationship set WORKS_FOR associates one EMPLOYEE entity and one DEPARTMENT entity.



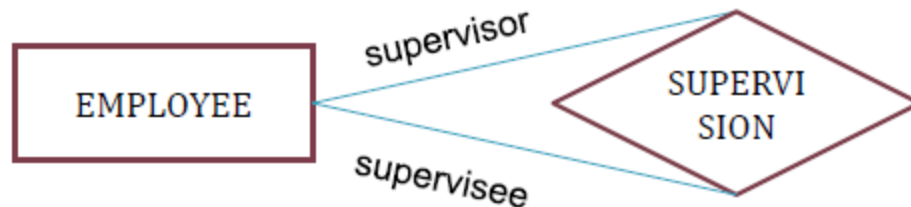
DEGREE OF A RELATIONSHIP

It is the number of participating entity types in a relationship

- If there are two entity types participating in a relationship, it is a binary relationship type
- If there are three entity types participating in a relationship, it is a ternary relationship type
- It is possible to have a n-array relationship

Unary relationships

- It is a relationship where the same entity participates more than once in different roles.
- In the example below , employees are supervised by employees (manager).



Unary relationships are also known as a **recursive relationship**.

ROLE NAME

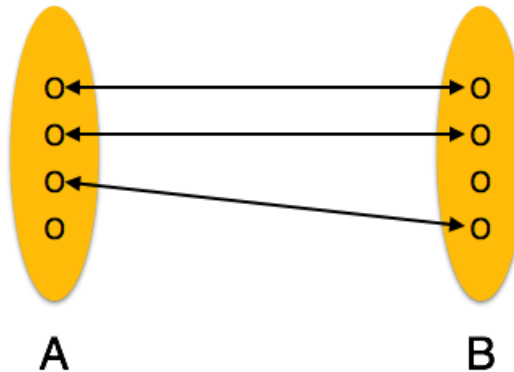
- Some entities participate more than once in a relationship type in different roles.
- **Role name represents a task that a participating entity from the entity type plays in the relationship.**
- Ex: Employee plays the role of supervisor as well as supervisee.
- If the participating entity types are distinct then there is no need for role name else role name is a must.

CARDINALITY CONSTRAINTS

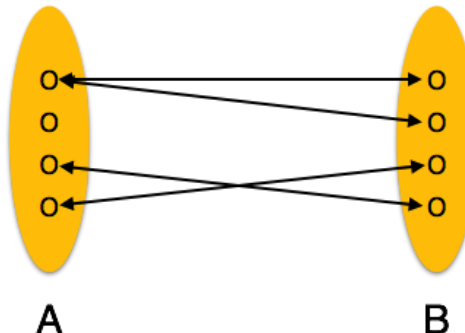
- **Cardinality** defines the number of entities in one entity set, that can be associated with the number of entities of other set via relationship.
- If we have two entity types A and B, the cardinality constraint specifies the number of instances of entity B that can (or must) be associated with entity A.
- Four possible categories are:
 - *One to one (1:1) relationship*
 - *One to many (1:m) relationship*
 - *Many to one (m:1) relationship*
 - *Many to many (m:n) relationship*

Mapping Cardinalities

- **One-to-one** – One entity from entity set A can be associated with at most one entity of entity set B and vice versa.

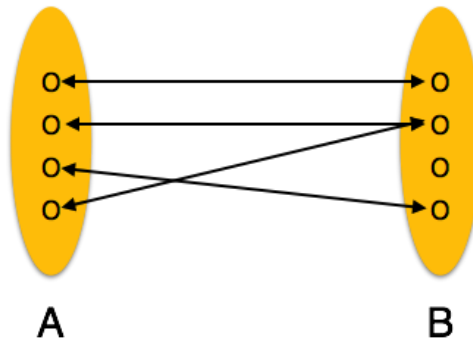


- **One-to-many** – One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.

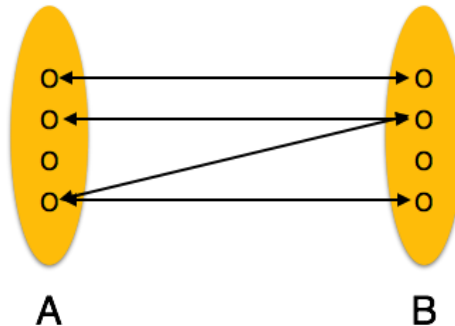


Mapping Cardinalities contd...

- **Many-to-one** – More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.



- **Many-to-many** – One entity from A can be associated with more than one entity from B and vice versa.



CARDINALITY CONSTRAINTS contd...

- one-to-one



- one to many

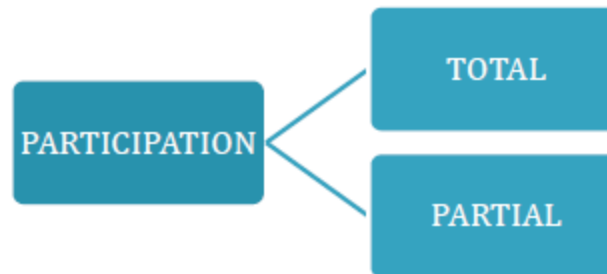


- many-to-many



PARITICIPATION CONSTRAINTS

- ***Specifies if existence of an entity depends on it being related to another entity via relationship.***
- Specifies minimum number of relationship instances each entity can participate in .
- Two type of the participation are : Total and Partial



PARITICIPATION CONSTRAINTS contd...

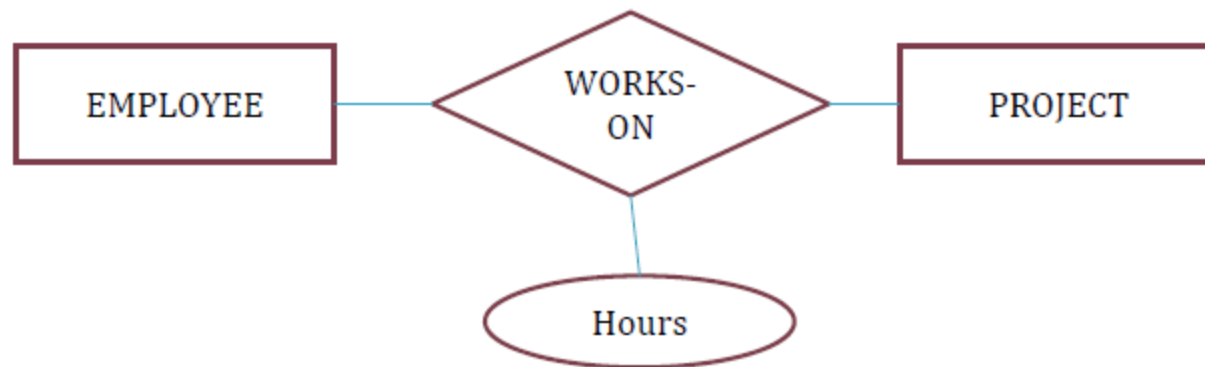
- Ex: if company policy says that every employee must work for the department then participation of employee in work-for is **total** (Each entity is involved in the relationship).



- Every entity in total set of employee must be related to a department via WORKS-FOR
- But we can't say that every employee must MANAGE a department . Hence relationship is **partial** (Not all entities are involved in the relationship).
- Total participation is indicated by double line and partial participation by single line.

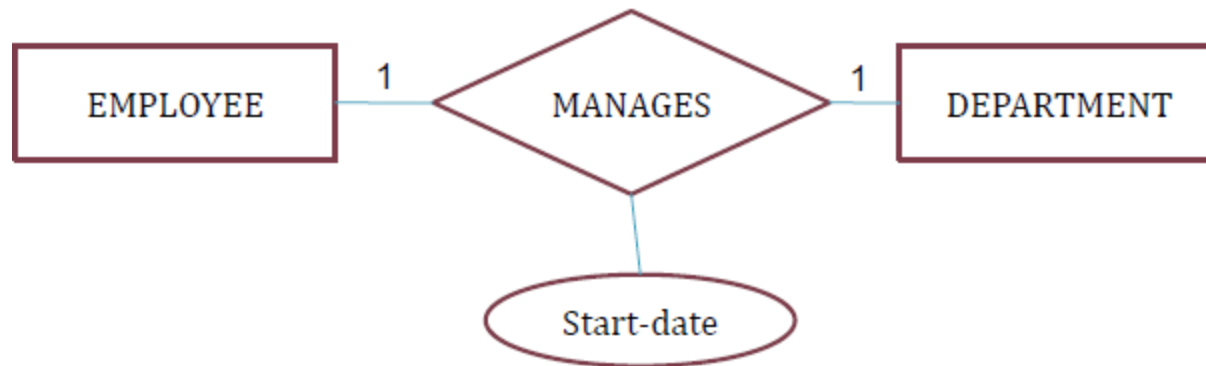
ATTRIBUTE OF RELATIONSHIP TYPE

- Relationship can also have attributes
- Ex: Hours for WORKS-ON relationship between EMPLOYEE and PROJECT



ATTRIBUTE OF RELATIONSHIP TYPE contd..

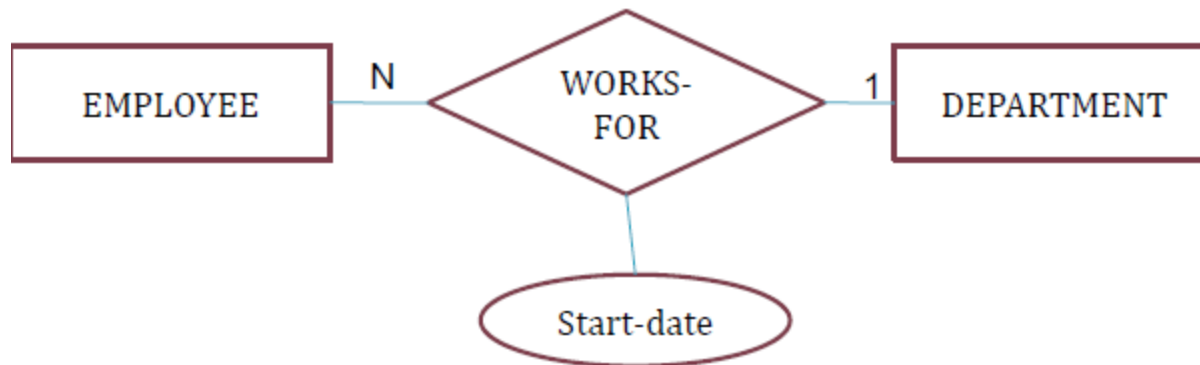
- Attributes of 1:1 relationship can be migrated to one of the participating entity types.
- Ex: Start-date attributes of MANAGES can be attribute of either DEPARTMENT or EMPLOYEE though conceptually it belongs to manages.



Every DEPARTMENT /EMPLOYEE entity participate in atmost one relationship instance. So value of the Start-date can be determined separately either by participating DEPARTMENT entity or participating EMPLOYEE entity.

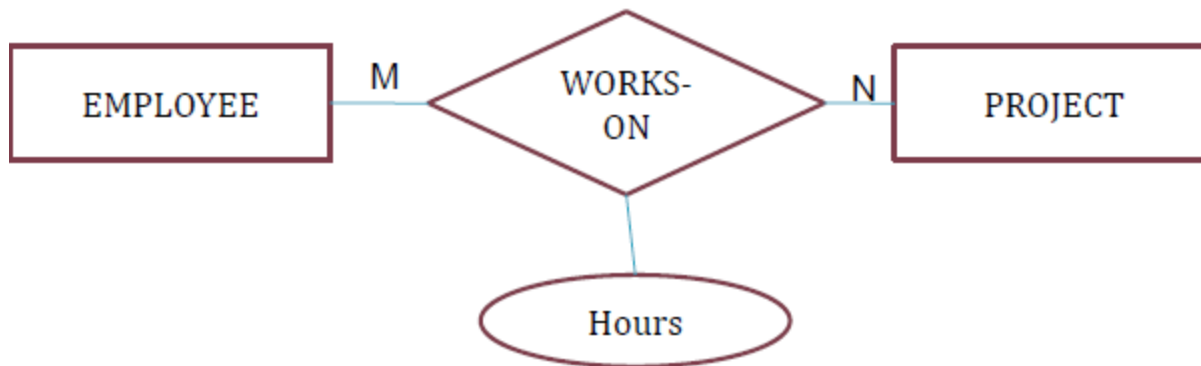
ATTRIBUTE OF RELATIONSHIP TYPE contd..

- For 1:N relationship a relationship attribute can be migrated **only to** entity type on N-side of relationship
- Start-date attribute here can be added only to employee.



ATTRIBUTE OF RELATIONSHIP TYPE contd..

- For M:N relationship types some attribute are determined by the combination of the participating entities, not by a single entity.
- Such attribute ***must be*** specified as the relationship attributes
- Ex: No. of hours an employee works on is department is determent is determined by the EMPLOYEE-PROJECT combination.

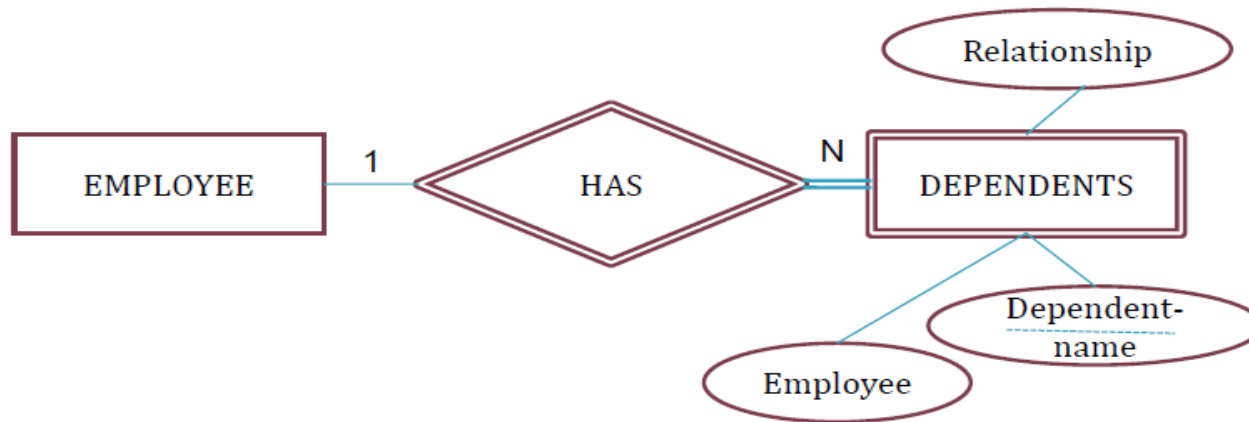


WEAK ENTITY AND STRONG ENTITY

- Entity type that doesn't have a key attribute on its own is called weak entity and the Regular entity types that have key value is called strong entities.
- Entity belonging to weak entity type is identified by being related to another entity type.
- We call this entity type as identifying or owner entity type (Parent/Dominant Entity type)
- The relationship that connects owner entity type to weak entity is called Identifying relationship.

WEAK ENTITY AND STRONG ENTITY contd...

- Weak entities have always a total participating constraint because they cannot be identified without an owner entity.














- Partial key attribute is denoted by dotted line.

Weak entity type normally has ***partial key(discriminator)***

Partial key is an attribute or set of attributes that can uniquely identify weak entities that are related to some owner entity

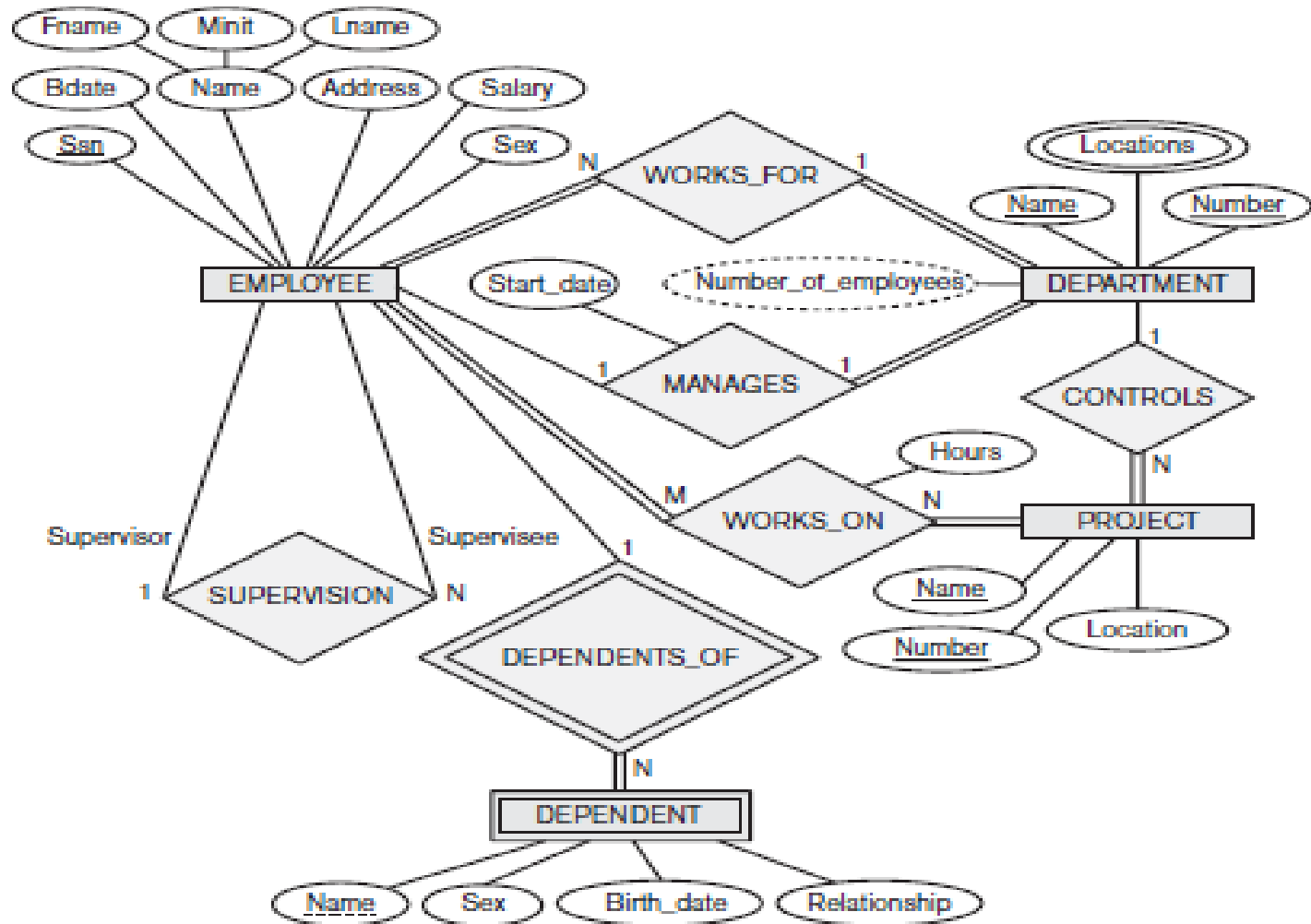
WEAK ENTITY AND STRONG ENTITY contd...

- An attribute that exists in several entity types may be elevated or promoted to an independent entity.
- Ex: Suppose several entity types in an UNIVERSITY DATABASE such as student , instructor & course each has an attribute named department . Then we can better keep an entity named department with a single attribute department name and relate it to 3 entity types student , instructor & course via appropriate relationship.

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

ER Diagram Notations

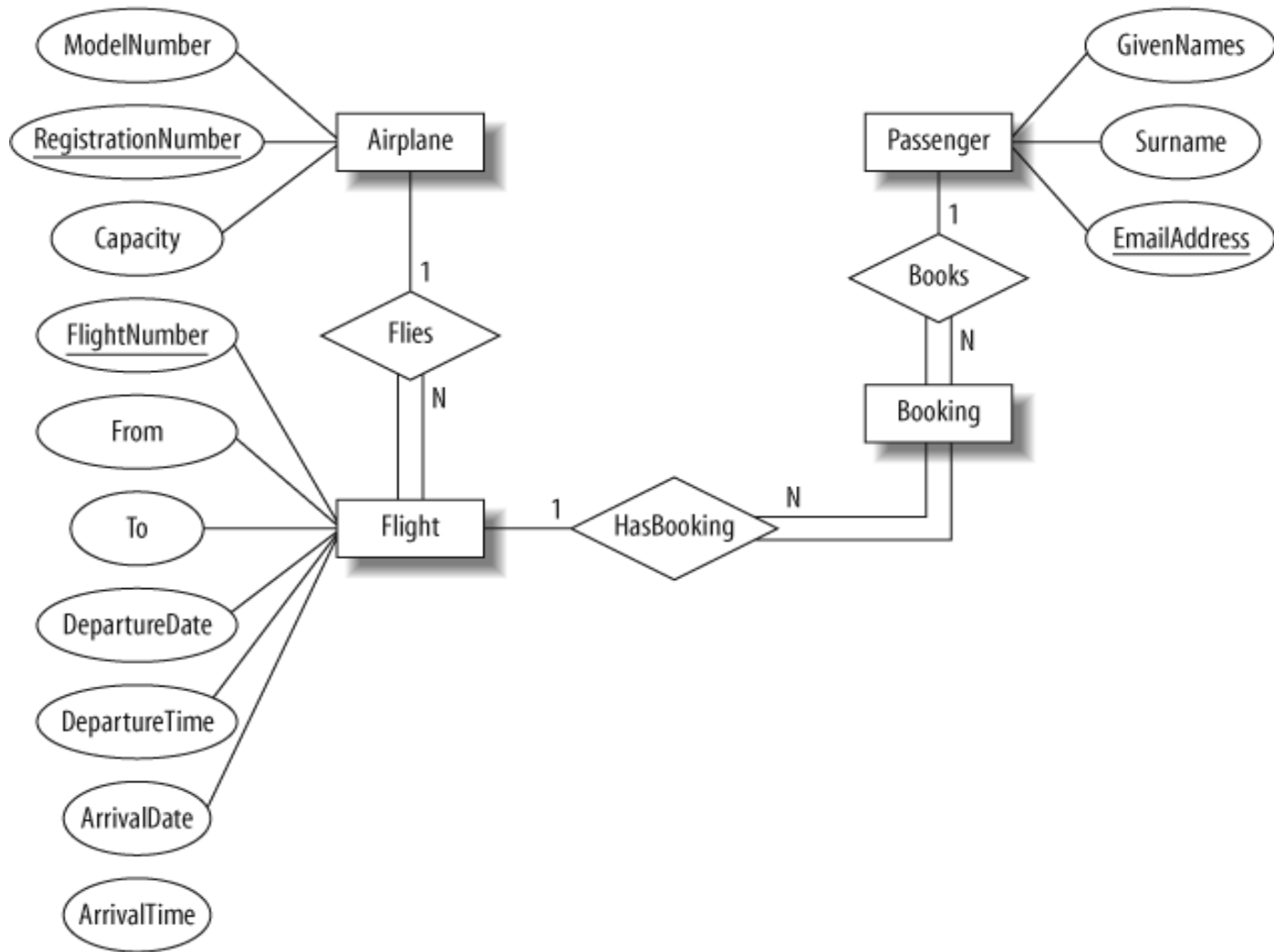
An ER schema diagram for the COMPANY database



The Flight Database

- The flight database stores details about an airline's fleet, flights, and seat bookings. Again, it's a hugely simplified version of what a real airline would use, but the principles are the same.
- Consider the following requirements list:
 - The airline has one or more airplanes.
 - An airplane has a model number, a unique registration number, and the capacity to take one or more passengers.
 - An airplane flight has a unique flight number, a departure airport, a destination airport, a departure date and time, and an arrival date and time.
 - Each flight is carried out by a single airplane.
 - A passenger has given names, a surname, and a unique email address.
 - A passenger can book a seat on a flight.

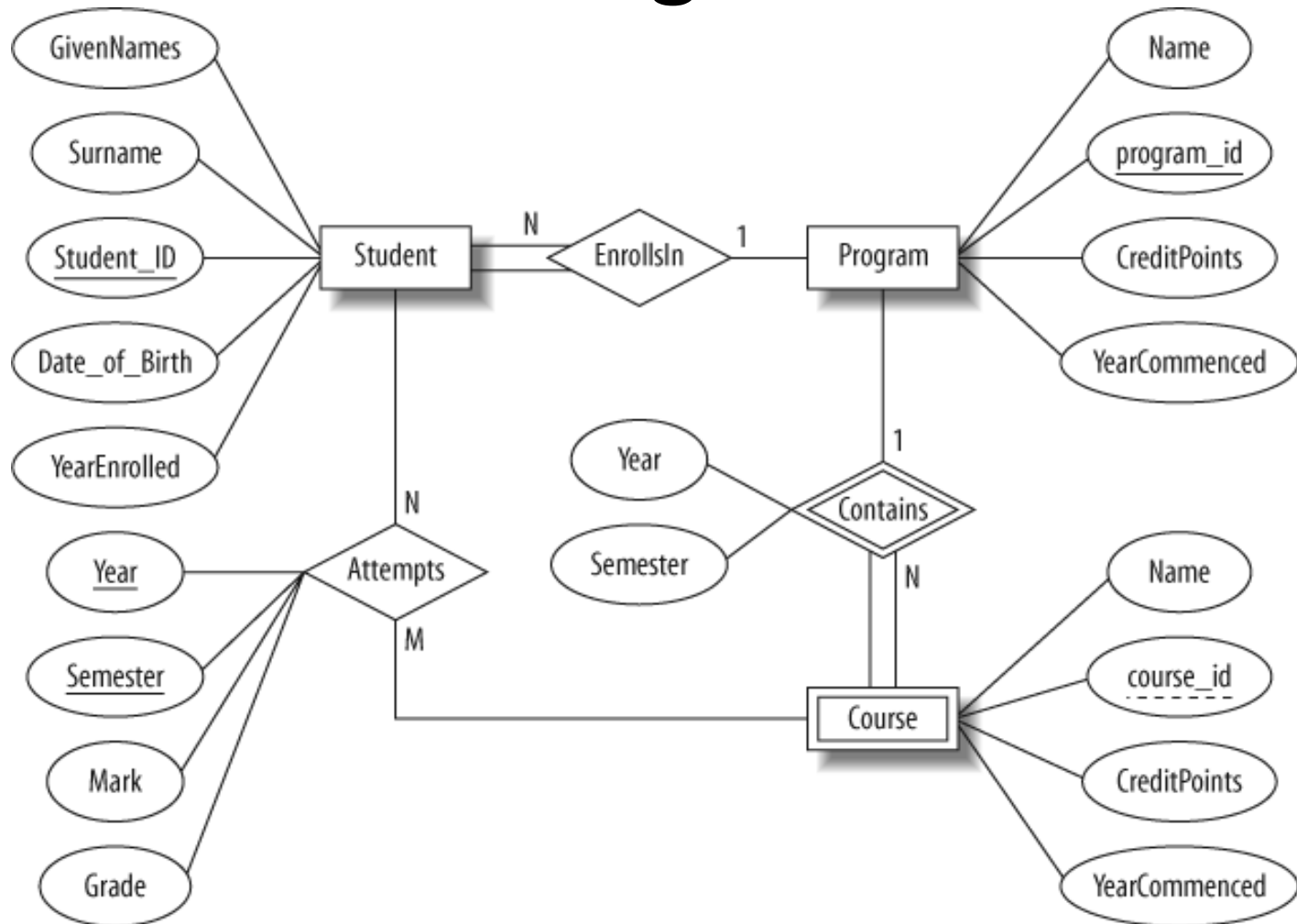
ER diagram



The University Database

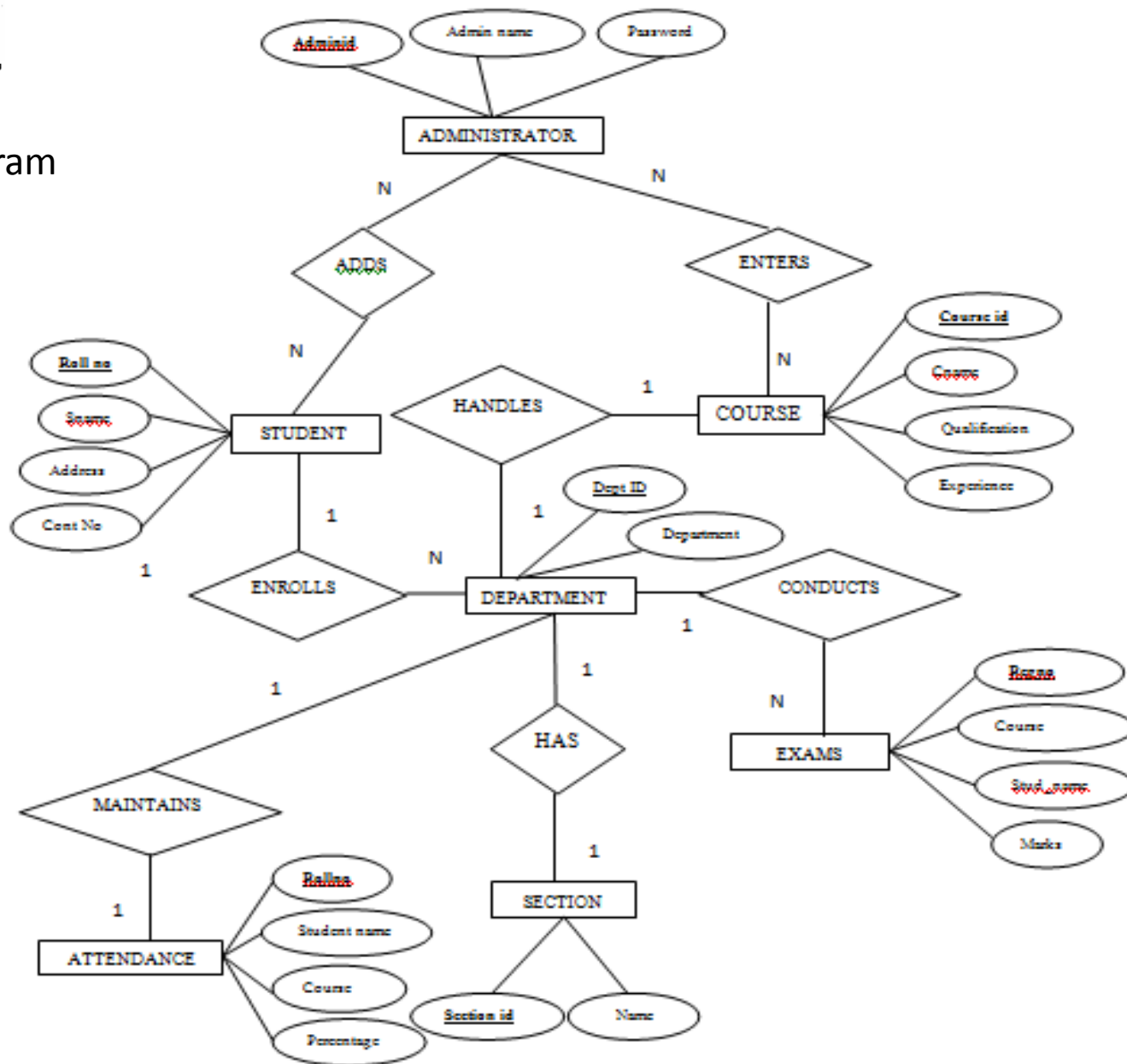
- The university database stores details about university students, courses, the semester a student took a particular course (and his mark and grade if he completed it), and what degree program each student is enrolled in.
- Consider the following requirements list:
 - The university offers one or more programs.
 - A program is made up of one or more courses.
 - A student must enroll in a program.
 - A student takes the courses that are part of her program.
 - A program has a name, a program identifier, the total credit points required to graduate, and the year it commenced.
 - A course has a name, a course identifier, a credit point value, and the year it commenced.
 - Students have one or more given names, a surname, a student identifier, a date of birth, and the year they first enrolled.
 - When a student takes a course, the year and semester he attempted it are recorded. When he finishes the course, a grade (such as A or B) and a mark (such as 60 percent) are recorded.
 - Each course in a program is sequenced into a year (for example, year 1) and a semester (for example, semester 1).

ER diagram



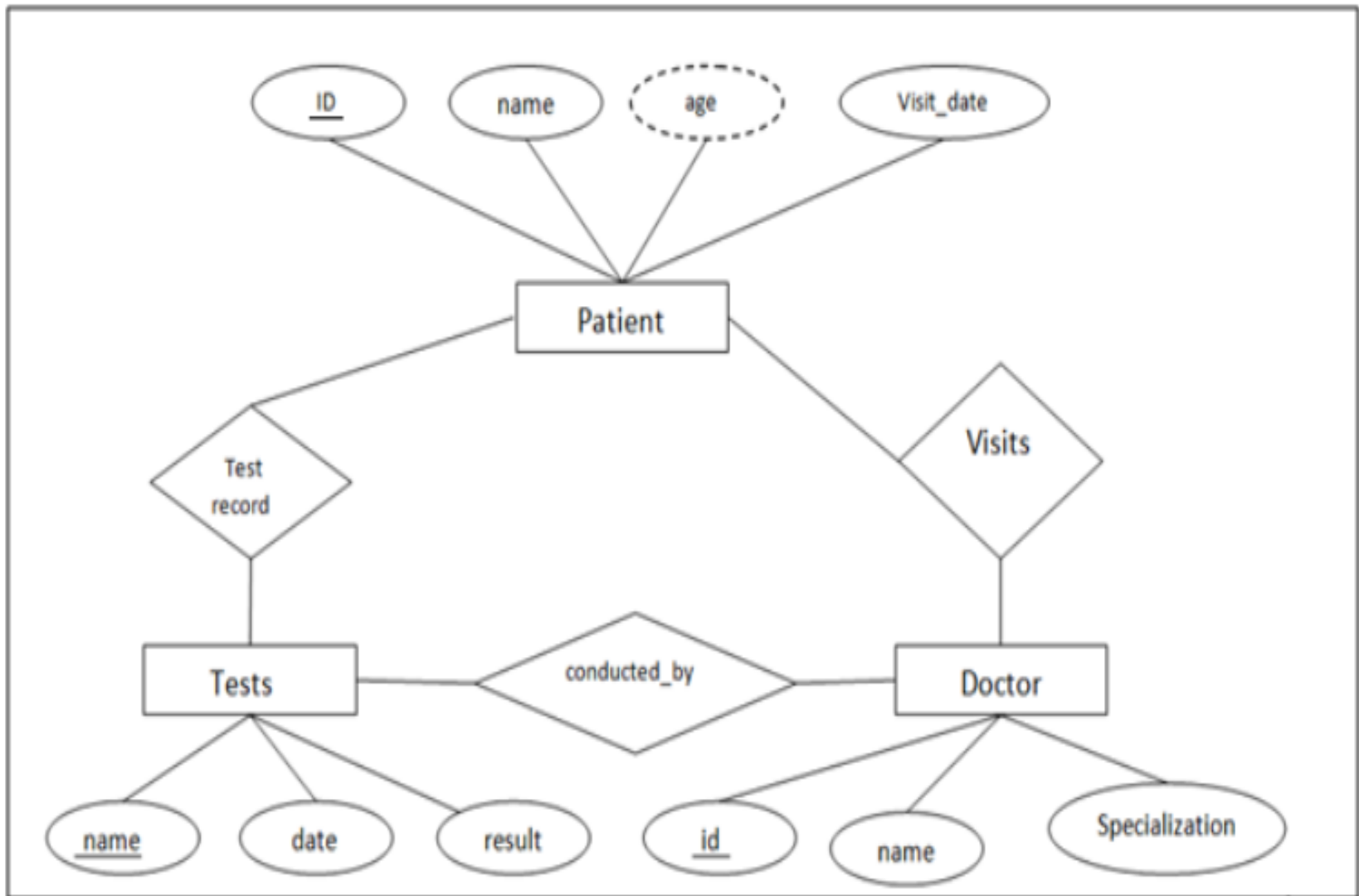
Student Management System

ER diagram



Hospital Database

ER diagram



Thank you