Chapter 6
Designing Database
(Logical Data Modeling)

4 Hours ≈ 7-8 marks

Contents

Introduction

- Analysis of information system helps to represent an organization's data graphically using DFDs, Decision Tables, and E-R diagrams
  - Design phase helps to create and manage data files.
  - It also helps to design physical and logical aspects of database (collection of data) and interfaces through which interactions between system and external actors occur.

- In design phase, the System analyst works with programmers to develop a software.
  - Communication occurs through diagrams and documentations.
  - Analyst provide transaction details and how system should perform. Programmers design, code, the computer programs based on analyst's instruction.

- A design phase produces computer programs and system

Database Design

- File and database design occurs in two steps.
  - At first a **logical database design model** is developed, which describes data using a notation that corresponds to a data organization used by a database management system.
  - Afterwards A **physical database design** is developed, that is used to prescribe the technical specifications for computer files and databases in which to store the data ultimately.

- Generally, logical and physical database design are performed in parallel with other systems design steps.
  - Thus, the detailed specifications of data necessary for logical database design are collected simultaneously as the system inputs and outputs are designed.

- Logical database design is driven not only from the previously developed E-R data model for the application but also from form and report layouts.
  - You study data elements on these system inputs and outputs and identify interrelationships among the data.
  - As with conceptual data modeling, the work of all systems development team members is coordinated and shared through the project dictionary or repository.

- The designs for logical databases and system inputs and outputs are then used in physical design activities to specify to computer programmers, database administrators, network managers, and others how to implement the new information system.

Logical Database design

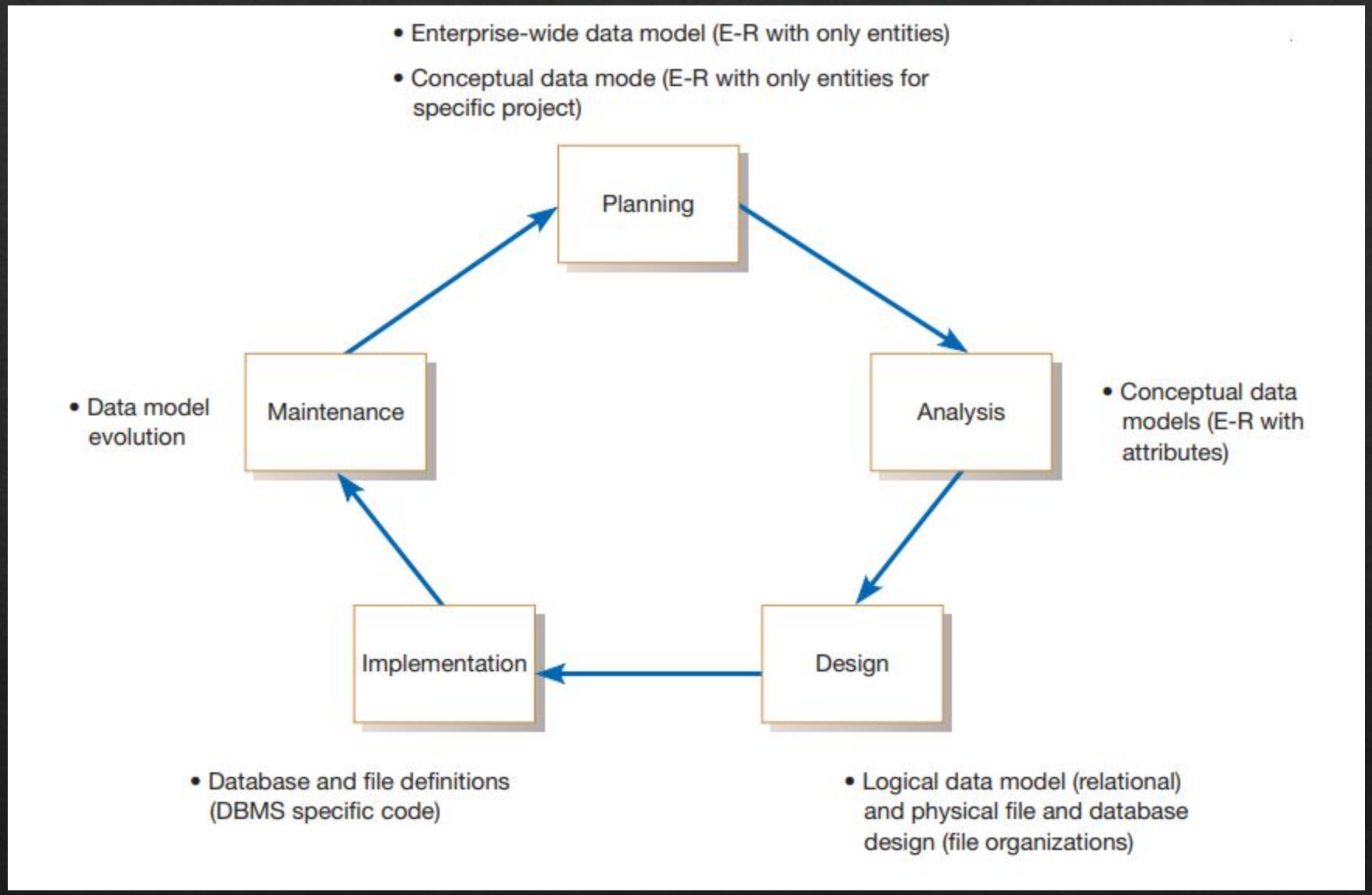
- A logical database design reflects the actual data requirements that exist in the forms (hard copy and computer displays) and reports of an information system.

- Logical database design consists of a process called **normalization**, which is a way to build a data model that has the properties of simplicity, non-redundancy, and minimal maintenance.

- logical database design usually uses a **relational database model**, which represents data in simple tables with common columns to link related tables.

Physical Database Design

- During physical database design, the results of logical database design steps are used.
- You also consider definitions of each attribute; descriptions of where and when data are entered, retrieved, deleted, and updated; expectations for response time and data integrity; and descriptions of the file and database technologies to be used.
- These inputs allow to make key physical database design decisions, such as choosing data types for each attribute, grouping attributes into physical records, and arranging/organizing related records in secondary memory.


• Enterprise-wide data model (E-R with only entities)
• Conceptual data mode (E-R with only entities for specific project)
Planning
Analysis
Design
Implementation
Maintenance
• Data model evolution
• Conceptual data models (E-R with attributes)
• Database and file definitions (DBMS specific code)
• Logical data model (relational) and physical file and database design (file organizations)

Relational Database Model

- A Relational Database Model represents data in the form of related tables, or relations.

- A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organized into 2-D tables.

- A relation is a named, 2-D table of data.
  - Each relation (or table) consists of a set of named columns and an arbitrary number of unnamed rows.
  - Each column in a relation corresponds to an attribute of that relation. Each row of a relation corresponds to a record that contains data values for an entity.

EMPLOYEE1

Emp_ID	Name	Dept	Salary
100	Margaret Simpson	Marketing	42,000
140	Allen Beeton	Accounting	39,000
110	Chris Lucero	Info Systems	41,500
190	Lorenzo Davis	Finance	38,000
150	Susan Martin	Marketing	38,500

SALES

Customer_ID	Customer_Name	Salesperson	Region
8023	Anderson	Smith	South
9167	Bancroft	Hicks	West
7924	Hobbs	Smith	South
6837	Tucker	Hernandez	East
8596	Eckersley	Hicks	West
7018	Arnold	Faulb	North

Properties of Relational Tables

1. Data is presented as a collection of relations.
2. Each relation is depicted as a table.
3. Columns are attributes that belong to the entity modeled by the table
   - E.g. In a **student** table, you could have name, address, student ID, major, etc.).
4. Each row ("tuple") represents a single entity.
   - E.g. In a student table, John Smith, 14 Oak St, 9002342, Accounting, would represent one student entity.
5. Every table has a set of attributes that taken together as a "key" uniquely identifies each entity.
   - E.g. In the student table, “student ID” would uniquely identify each student – no two students would have the same student ID)

Are all tables relations??

- No!!

- Relational tables have varying characteristics than non-relational tables:
  - Entries in cells are simple.
  - Entries in a given column are from the same set of values.
  - Each row is unique.
  - The sequence of columns can be interchanged without changing the meaning or use of the relation.
  - The rows may be interchanged or stored in any sequences.

Normalization

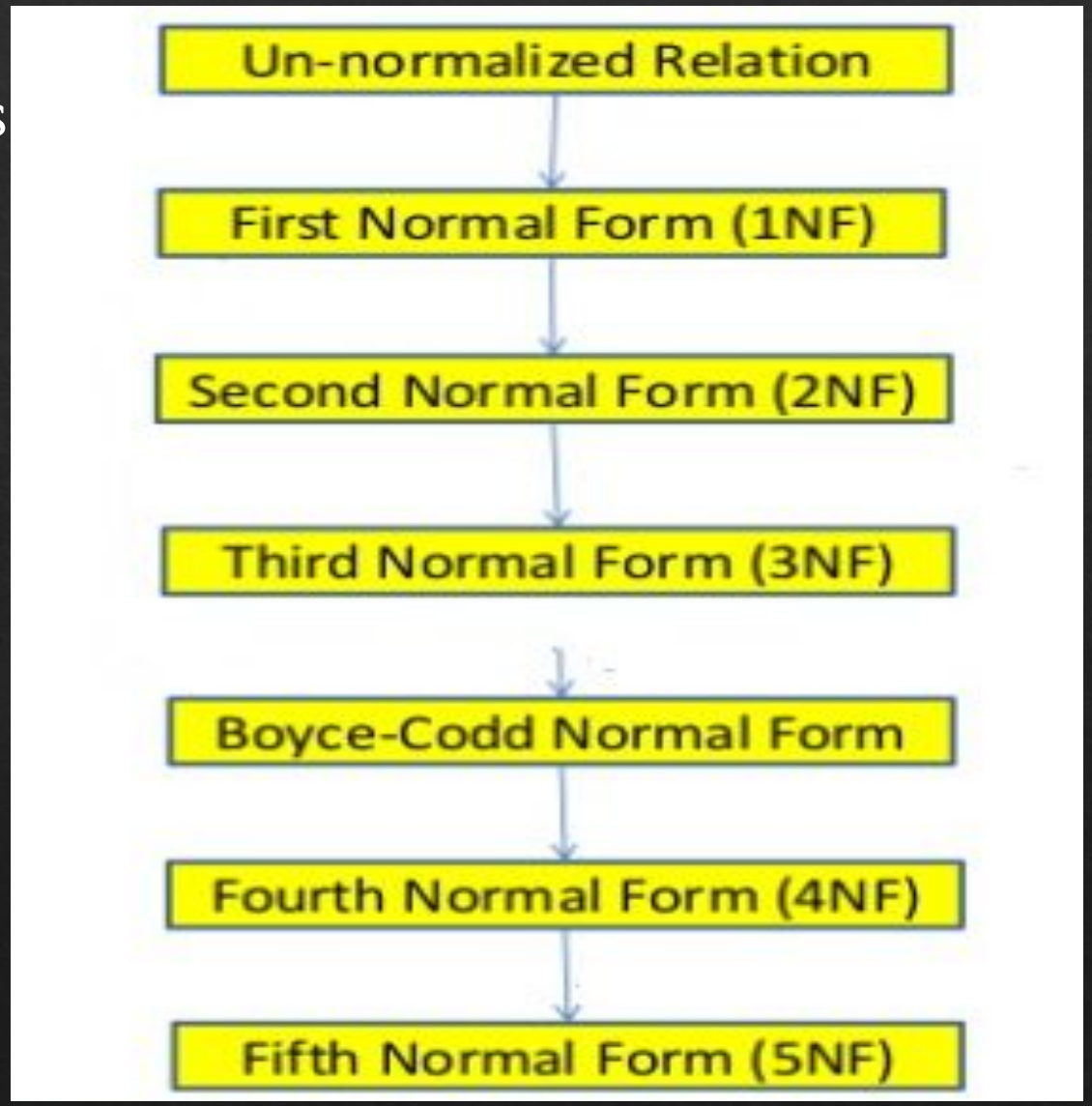
Normalization, Role of Normalization

Normalization:

- Normalization is the process of organizing the data in the database.
- It is the process of organizing the columns (attributes) and tables (relations) of a relational database to reduce data redundancy and improve data integrity.
- It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- Normalization is used for mainly two purposes:
  - Eliminating redundant(useless) data.
  - Ensuring data dependencies i.e data is logically stored.

Normal Forms

- The database community sets the few rules or guidelines for database normalization. These rules are called as **normal forms.**
- Normal form starts from the **1NF till 5NF.**



First Normal Form (1NF)

- As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values.

  i.e. It should hold only atomic (single) values.

- Example:

Student

Student_name	Age	Subject
Ram	22	DBMS, Math
Gita	21	Math
Gina	20	Microprocessor
Hari	21	DBMS

Student table in 1NF:

Student_name	Age	Subject
Ram	22	DBMS
Ram	22	Math
Gita	21	Math
Gina	20	Microprocessor
Hari	21	DBMS

Second Normal Form (2NF)

- A table is said to be in 2NF if both the following conditions hold:
  - Table is in 1NF (First normal form)
  - No non-prime attribute is dependent on the proper subset of any candidate key of table.

  i.e. The primary key of the table should compose of exactly 1 column.
- An attribute that is not part of any candidate key is known as **non-prime attribute.**

Example of 2NF:

Teacher

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Candidate Keys: {teacher_id, subject}
Non prime attribute: teacher_age

Here, teacher_age depends on composite key (teacher_id, subject)

Teacher table in 2NF:

Teacher_Details

teacher_id	teacher_age
111	38
222	38
333	40

In this table, teacher_age depends on only teacher_id (primary key).

Teacher_Subject

teacher_id	subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

In this table, subject depends on only teacher_id (primary key).

Functional dependency (FD)

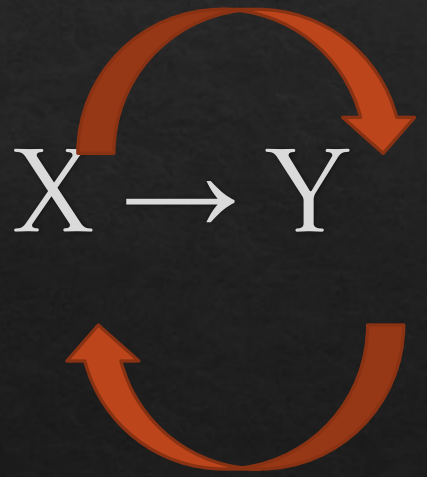
- FD is a relationship between two attributes, typically between the PK and other non-key attributes within a table.
- **Statement**: “An attribute in a relational model table is said to be functionally dependent on another attribute in the table if it can take only one value for a given value of the attribute upon which it is functionally dependent”
- or
- A functional dependency is a property of the semantics of the attributes in a relation.
  - The semantics indicate how attributes relate to one another, and specify the functional dependencies between attributes.
- It helps to maintain the quality of data in the database.
- It plays a vital role to find the difference between good and bad database design.

- A functional dependency is denoted by an arrow "→".
- The functional dependency of X on Y is represented by X → Y.

Here, Y is functionally dependent on X.

X functionally determines Y.

X functionally determines Y.



Y is functionally dependent on X.

- Example:

Employee number	Employee Name	Salary	City
1	Dana	50000	San Francisco
2	Francis	38000	London
3	Andrew	25000	Tokyo

- In this example, if we know the value of Employee number, we can obtain Employee Name, city, salary, etc. By this, we can say that the city, Employee Name, and salary are functionally depended on Employee number.
- We can represent relationships between attributes as:

  Employee number → Employee Name
  Employee number → Salary
  Employee number → City

  OR,

  Employee number → Employee Name, Salary, City

Third Normal Form (3NF)

- A table design is said to be in 3NF if both the following conditions hold:
  - Table must be in 2NF
  - Transitive functional dependency of non-prime attribute on any super key should be removed.
- Example:

Tournament

Tournament	Year	Winner	Winner_DOB
Indiana Invitational	1998	All Fredrickson	1975/5/9
Cleveland	1999	Bob Albertson	1968/9/22
Des Moines	1999	All Fredrickson	1975/5/9
Indiana Invitational	1998	Chip Masterson	1977/7/13

In this table, following transitive functional dependencies are existing:

(Tournment, Year) -> Winner

Winner -> Winner_DOB

Here, transitive dependency is presence.

Tournment table will be in 3NF if we divide Tournament table as below:

Tournament_Details

Tournment	Year	Winner
Indiana Invitational	1998	All Fredrickson
Cleveland	1999	Bob Albertson
Des Moines	1999	All Fredrickson
Indiana Invitational	1998	Chip Masterson

Tournament_Winner

Winner	Winner_DOB
All Fredrickson	1975/5/9
Bob Albertson	1968/9/22
All Fredrickson	1975/5/9
Chip Masterson	1977/7/13

Note: No Transitive functional dependency

Transitive Dependency

- A functional dependency is said to be transitive if it is indirectly formed by two functional dependencies.

- For e.g., X Z is a transitive dependency that occurs when following functional dependencies occur
  1. X Y but Y X doesn't hold true
  2. Y Z

- A transitive dependency can occur in a relation of 3 or more attributes.

- **Example:**

Book

Book_Name	Author	Author_Age
Muna Madan	Laxmi Prasad Devkota	52
Harry Porter	J.K. Rowling	49
Games Of Thrones	George R.R. Martin	66

- **Here,** {Book_Name} □ {Author }
  - **if we know the book's name ,we know the author's name.**
- **But ,** {Author } does not □ {Book_Name}
- **Also ,** {Author } □ {Author_Age}
  - **if we know Author ,we know the Author's age.**
- Therefore as per the rule of transitive dependency,
  - {Book_Name} □ {Author_Age} surely holds, that makes sense also because if we know the book's name ,we can know the author's age too.

Role of Normalization

The following makes Database Normalization a crucial step in database design process :

- **Resolving the database anomalies**
  - The forms of Normalization remove all the Insert, Update and Delete anomalies.
    - **Insertion Anomaly** occurs when you try to insert data in a record that does not exist.
- **Eliminate Redundancy of Data**
- **Data Dependency**
  - The data gets stored in the correct table and ensures normalization.
- **Isolation of Data**
  - A good designed database states that the changes in one table or field do not affect other. This is achieved through Normalization.
- **Data Consistency**
  - While updating if a record is left, it can led to inconsistent data, Normalization resolves it and ensures Data Consistency.
- **Much more flexible DB design.**
- **A better handle on DB security, etc.**

Normalization: Analysis

Advantages

- Greater overall database organization.
- Reduction of redundant data.
- Data consistency within the database.
- A much more flexible database design.
- A better handle on database security.
- Easy to add data
- Data storage requires less space.

Disadvantages

- Difficult to implement (needs expertise)
- Time consuming process
- Requires detailed analysis because poorly normalized database is inefficient for storage.
- Can slow down performance when huge tables with many joins are normalized.
- Results in complex design of database.

Transforming E-R diagrams into Relations

- Normalization produces a set of well-structured relations that contains all of the data mentioned in system inputs and outputs developed in human interface design.
- To ensure all possible data requirements have been addressed in the design of database, analysts needs to compare the conceptual data model and the normalized relations developed so far.
- This means, the developed E-R diagram must be transformed into relational notation, normalized, and then merged with the existing normalized relations to produce one final, consolidated set of relations.
- Transformation of E-R diagram occurs in 4 steps:

Phases in transformation of E-R Diagram

1. Represent Entities.
   - Each entity in E-R Diagram transforms into a relation.
   - In database terms, an entity changes into a table.

2. Represent Relationship.
   - Representing relationship from E-R diagram depend on its nature.
   - For e.g., usage of foreign key attribute to relate two entities/tables.

3. Normalize Relations.
   - The relations from 1 and 2 need to be normalized for better structuring.

4. Merge the Relations.
   - If any redundant relations occur during 1,2, and 3, they need to be merged and re-normalized.

Relational Model

Definitions and terminology

- **Relational Model (RM)** represents the database as a collection of relations.
- A relational model database is defined as a database that allows you to group its data items into one or more independent tables that can be related to one another by using fields common to each related table.
- The **relational model** for database management is an approach to managing data using a structure and language, where all data is represented in terms of tuples, grouped into relations.
- A database organized in terms of the relational model is a relational database.
- **Some popular Relational Database management systems are:**
  - DB2 and Informix Dynamic Server - IBM
  - Oracle and RDB – Oracle
  - SQL Server and Access - Microsoft

Basic terminology used in relational model

- **Tuples of a Relation:** Each row in data is actually a tuple.
- **Cardinality of a Relation**: The number of tuples in a relation determines its cardinality.
- **Attribute:** Each column in the tuple is called attribute.
- **Degree of relation**: The number of attributes in tuple determines its degree.
- **Domain**: Specifies the type of data represented by the attribute or a domain is all kind of values that an attribute can validly contain.
  - Domains are often confused by data types; data type is physical concept while domain is logical.
  - Number is data type and age is domain.
- **Field:** It stores actual value of an attribute.
- **Keys of a Relation:** it is a set of one or more columns whose combined values are unique. Some different types of keys are:
  - Primary key
  - Foreign Key
- **Relation:** Table with rows and column.

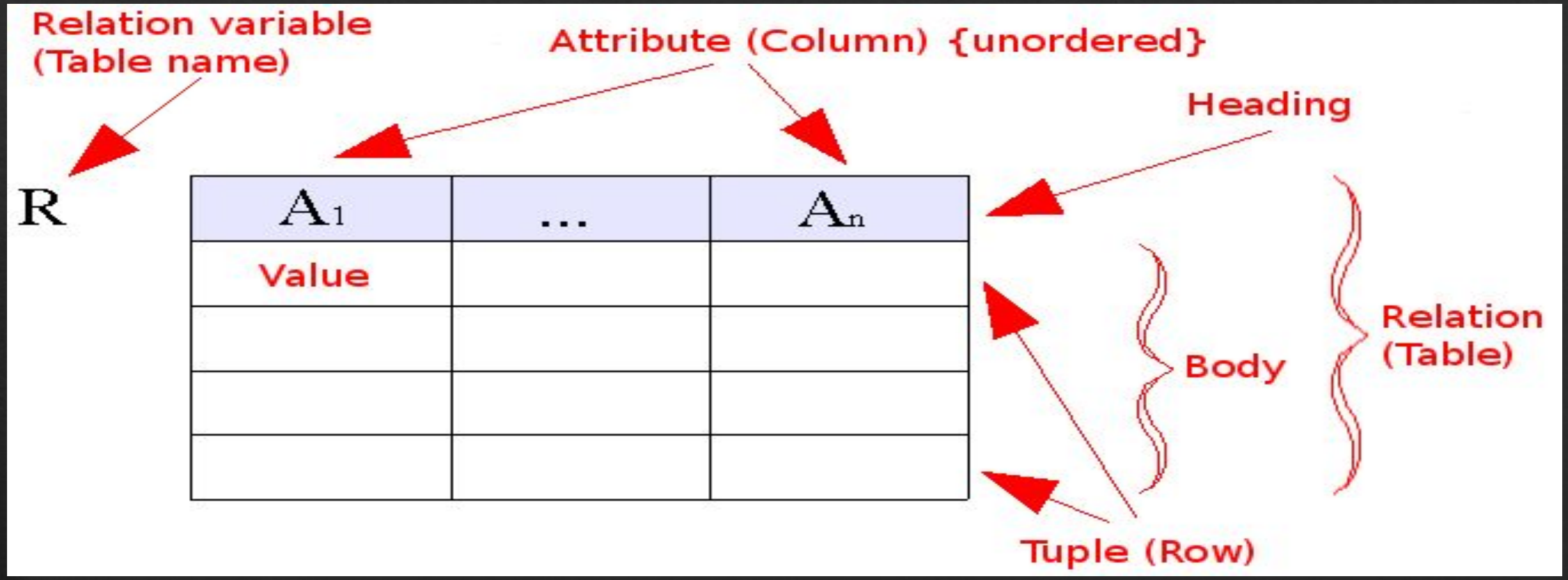
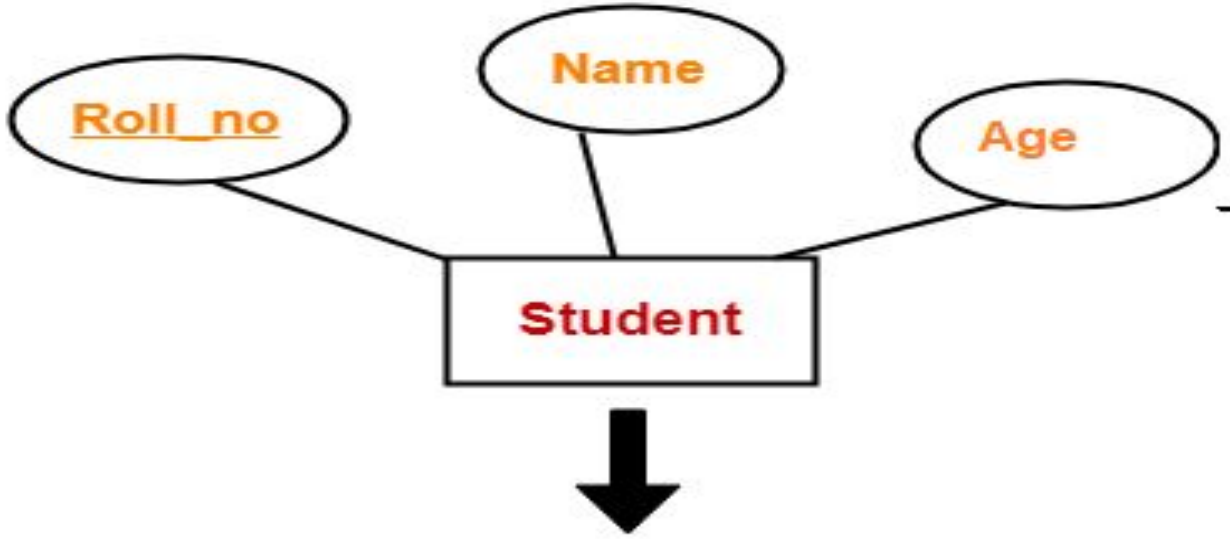


Figure: Relational Model Concept

Rules for converting ER-model to Relational model:

ER-model	Relational model
1. Entity	1. Relation (table)
2. Simple attributes	2. Columns (fields)
3. Key attributes	3. Primary key
4. Value set	4. Domain
5. 1:1 or 1:N relationship	5. Foreign key
6. M:N relationship	6. A relation and two foreign keys



ER-model

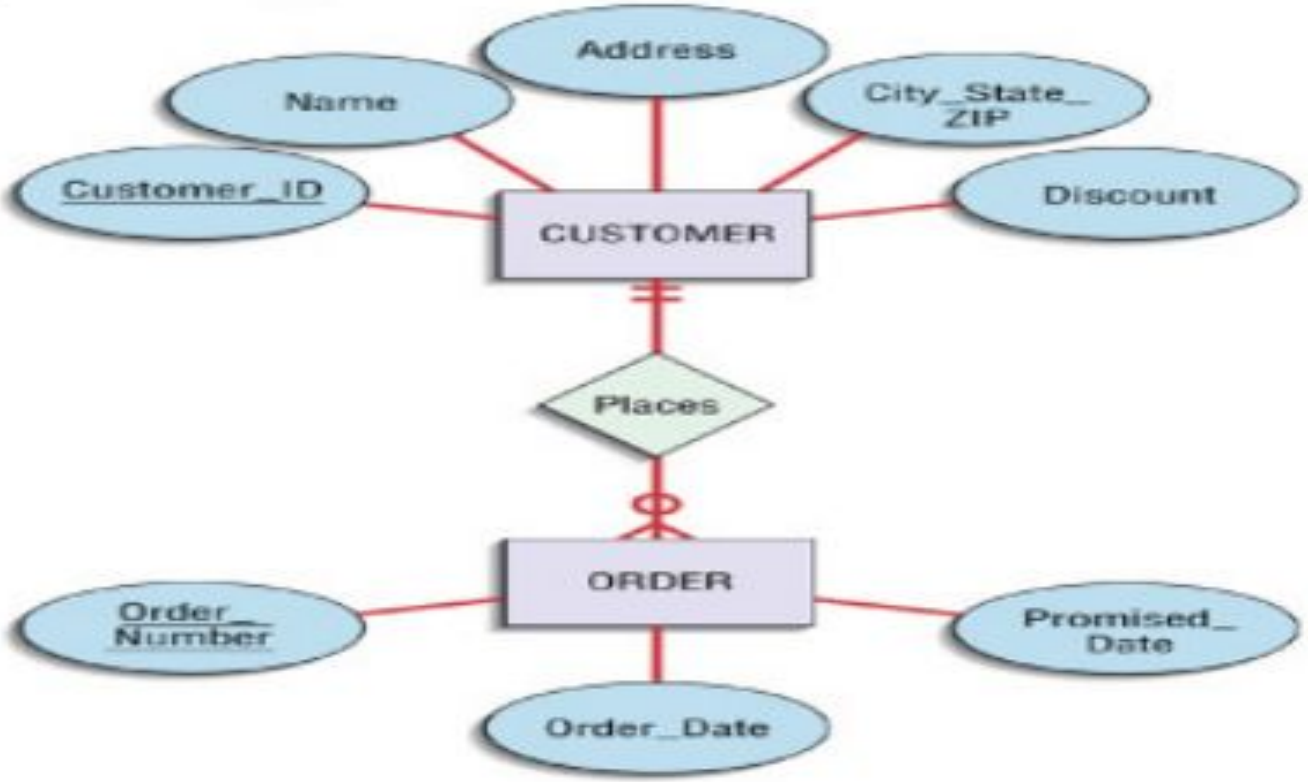
Student

Roll_no	Name	Age
1	Ram	20
2	Rita	22

Relational model

OR Schema : Student (Roll_no , Name ,Age)

Figure: Example of ER-model to Relational-model



CUSTOMER

Customer_ID	Name	Address	City_State_ZIP	Discount
1273	Contemporary Designs	123 Oak St.	Austin, TX 28384	5%
6390	Casual Corner	18 Hoosier Dr.	Bloomington, IN 45821	3%

ORDER

Order_Number	Order_Date	Promised_Date	Customer_ID
57194	3/15/0X	3/28/0X	6390
63725	3/17/0X	4/01/0X	1273
80149	3/14/0X	3/24/0X	6390

Assignment

1. Design ERD for student management system and convert to the Relational model.
2. Design ERD for Bank management system and convert to the Relational model.

Merging Relations

- As part of the logical database design, normalized relations likely have been created from a number of separate E-R diagrams and various user interfaces.
- Some of the relations may be redundant—they may refer to the same entities.
  - If so, those relations must be merged to remove the redundancy.
- Because of this, normalized relations obtained from user interfaces and transformed ER-diagrams are compared.
  - During such comparison, redundant relations are removed by merging.
  - This process is called **view integration.**

- E.g. a 3NF relation obtained by transforming an ER-Diagram yields
  employee1(emp_id, name, address)

- Another relation obtained from other interfaces or reports yields
  employee2(emp_id, name, salary, years_of_service)

- Then merging these two relations, we get,
  employee(emp_id, name, address, salary, years_of_service)

- Note: The attributes that appear in both the relations appear only once in the merged relation.

Problems in View Integration

1. Synonyms
   - In some situations, two or more attributes may have different names but the same meaning, as when they describe the same characteristic of an entity.
   - These attributes are synonyms.
   - E.g. relation1 may have “employee_no.” while relation2 may have “employee_id”

Solution to synonyms

- Seek agreement from users on a single standardized name for the attribute and eliminate the other synonym.

OR

- Choose the third, yet common name, for both the attributes to replace

2. Homonyms

- A single attribute name used for 2 or more different attributes which have different characteristics.
- A homonym represents different data, depending on how it is used.

- E.g., the term ***account*** may represent bank's saving account, current account, loan account, or some other type of account.

Solution to homonyms

- Create different attribute names for each relation to resolve conflict.

OR

- Choose the third, yet common name, for both the attributes to replace synonyms.

3.Dependencies between non-keys

- While merging 3NF relation ,we may encounter such difficulty;

  Example: student1(studID , majorSubject)

  student2(studID , advisor)

- After merging,
  - Student(studID , majorSubject , advisor)
- Let us suppose if each majorSubject has exactly one advisor then advisor is functionally dependent on majorSubject.
- This cause transitive dependency to be existed in the relation Student .
- Hence the relation is in 2NF but not in 3NF.
- The solution is
  - Student(studID, majorSubject)
  - MajorSub(majorSubject,advisor)

4.class/subclass relationship

- Example:

  Patient1(patId , name , address , date_treated)

  Patient2(patId , name ,room_no)

- In above relations , one thing is very much clear that room_no is associated with the admitted patient and date_treated means the patient is out patient.

- After Merging,

  Patient(patId , name , address)

  InPatient(patId ,date_treated)

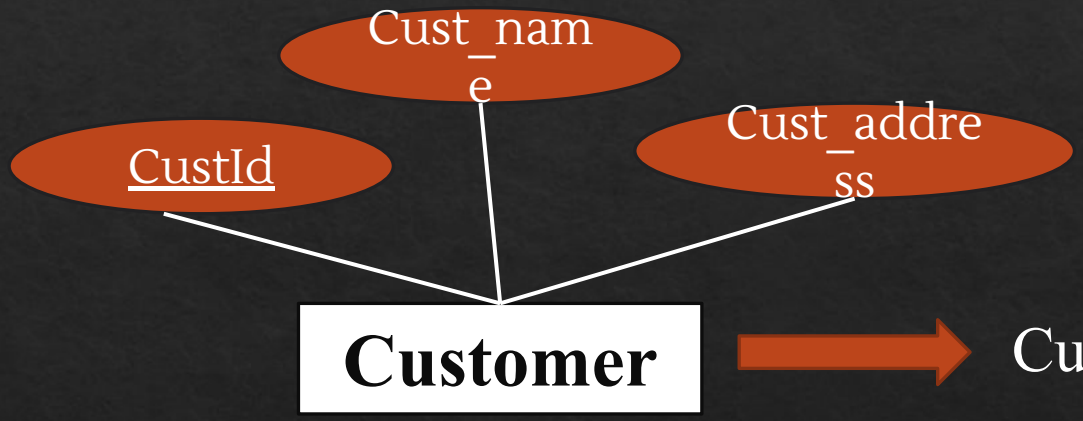
  OutPatient(patId, room_no)

End of the Chapter

- Some more examples:

1.Represent Entities

- The entities of ER is turned into relation/table.
- Each attribute turns into attribute column of relation.
- The primary key attribute of an entity becomes the primary key of the relation which is underlined. It can be composite if required but can never be NULL.
- Example:



Customer(CustId , name , address)

CustId	Cust_name	Cust_address

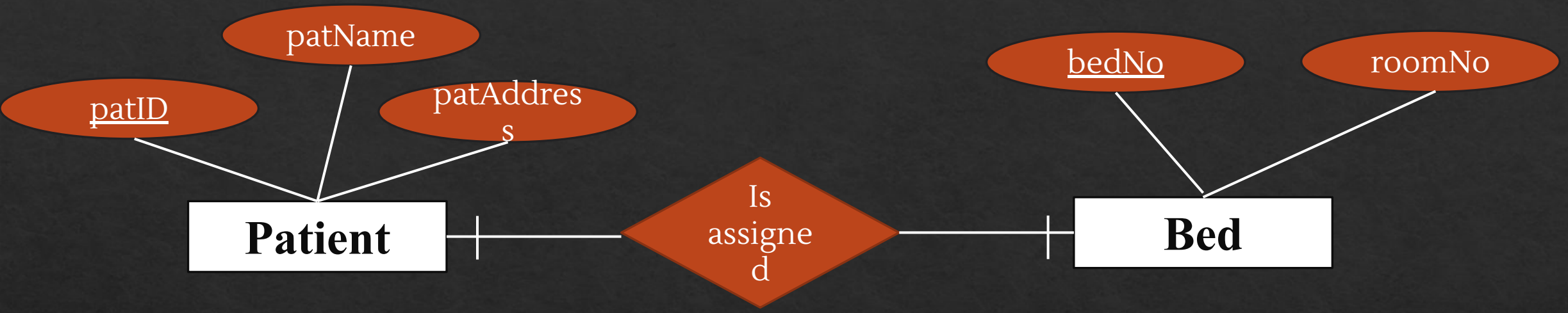
2.Represent relationship

- Representing relationship into relations depend on degree and cardinalities which are described below:

a. **1:1 relationship:**

- **Suppose there are two entities A and B which undergo into one to one relationship . Then we can perform any of the following tasks:**
  - **Place Primary key of A as a Foreign key of B.**
  - **Place primary key of B a Foreign Key of A.**
  - **Both of above**

Example

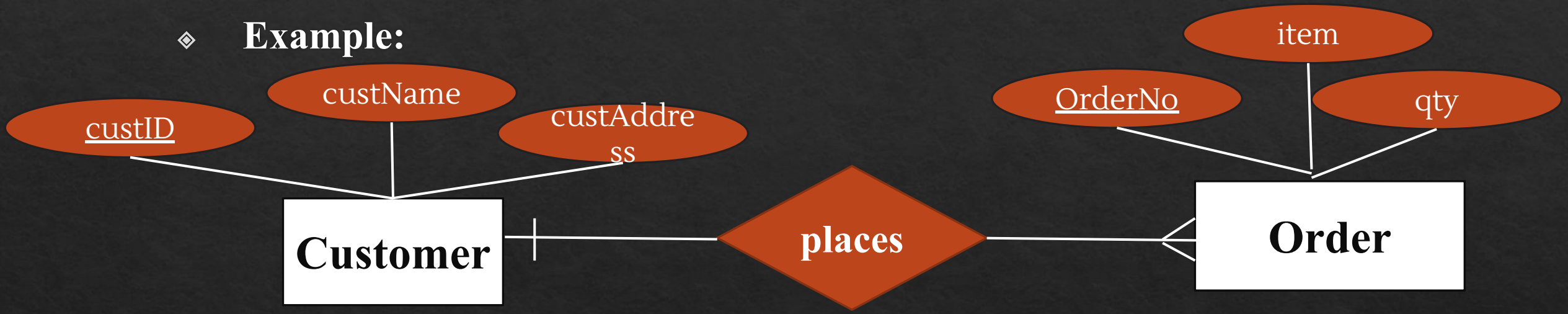


- Patient(patID , patName)
- Bed(bedNo ,patID , roomNo)

  Or,
- Patient(patID , patName , bedNo)
- Bed(bedNo , roomNo)

b. 1:N relationship:

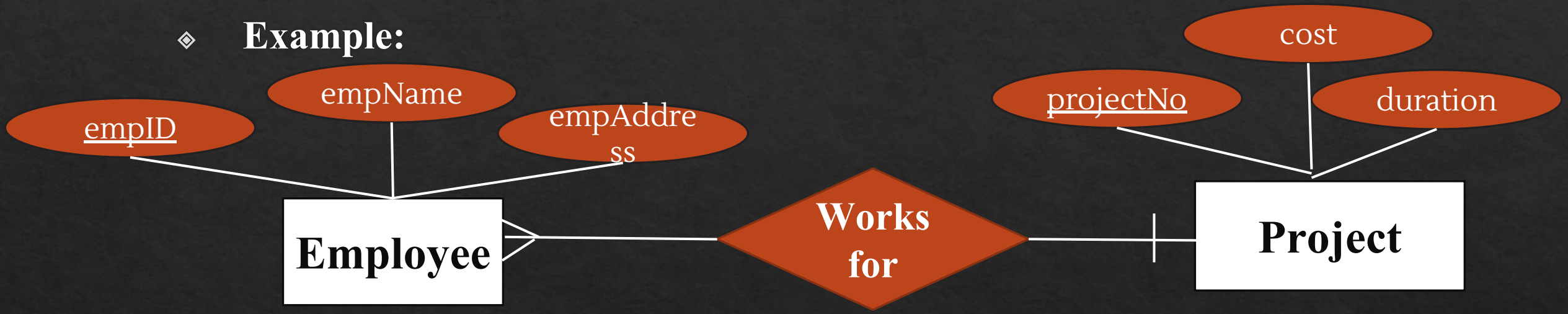
- **We represent binary one to many relationship by adding Primary Key of entity on the "one" side of the relationship as a Foreign Key in the "many" side of the relationship.**
- **Example:**



- Customer(custID , custName , custAddress)
- Order(orderNo ,custID , item , qty)

c. N:1 relationship:

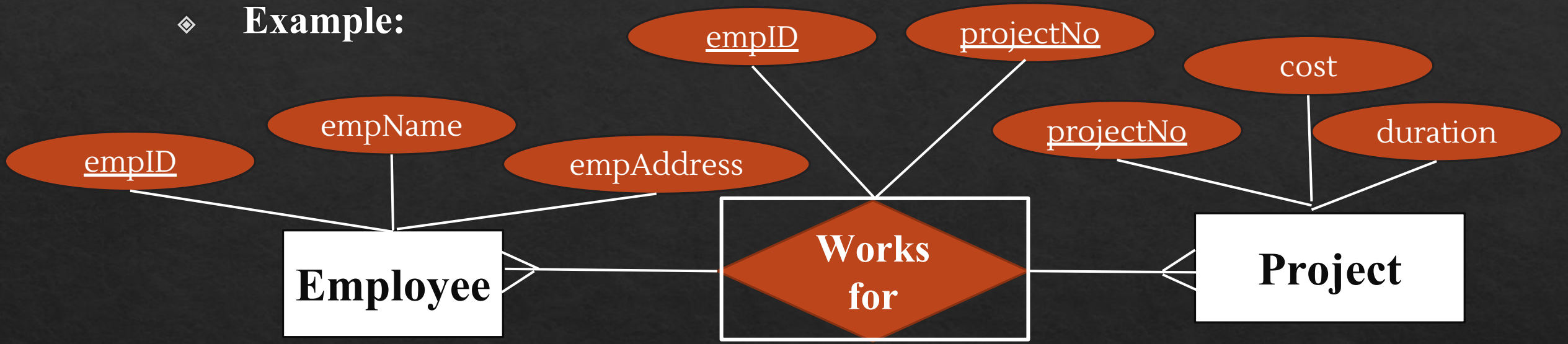
- **We represent binary many to one relationship by adding Primary Key of entity on the "one" side of the relationship as a Foreign Key in the "many" side of the relationship.**
- **Example:**



- Employee(empID ,projectNo, empName , empAddress)
- Project(projectNo , cost , duration)

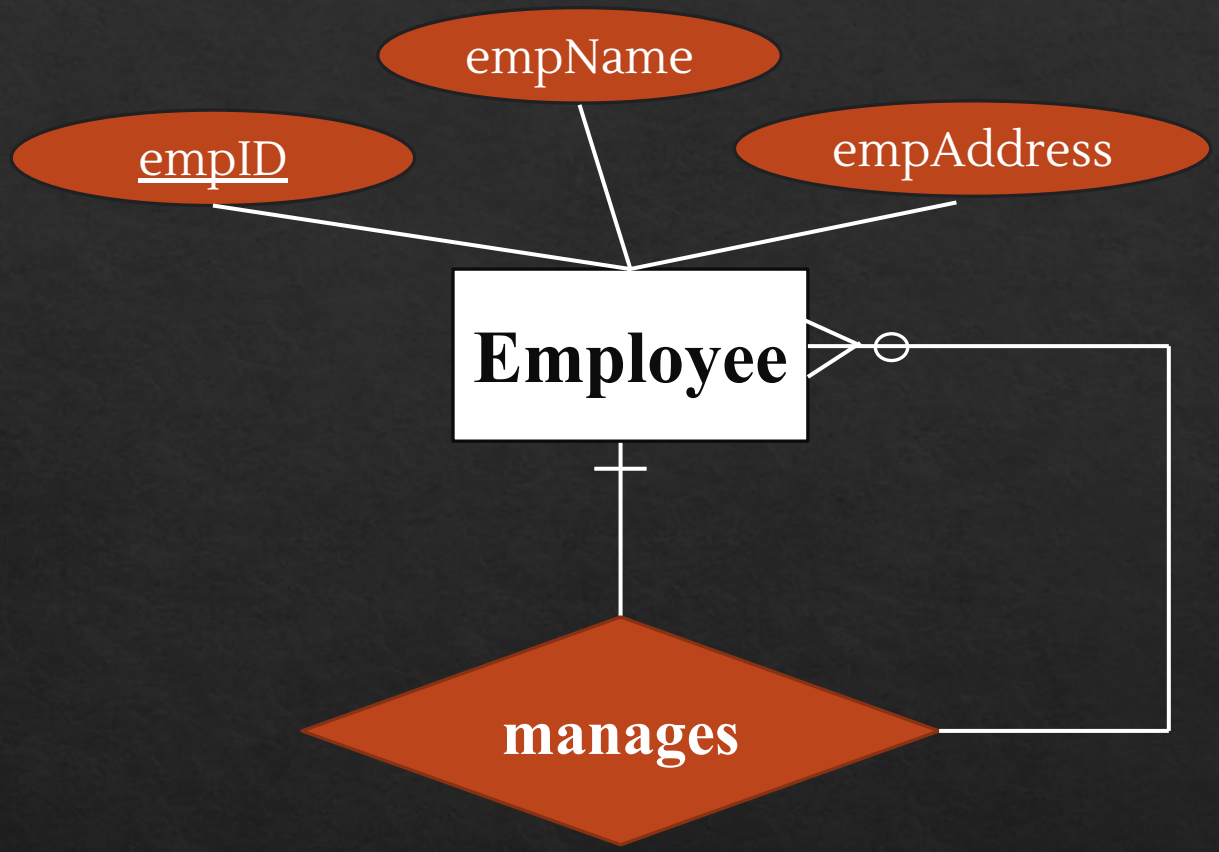
d. M:N relationship:

- **In case of binary Many to Many relationship, there exist associative entity that takes Primary keys of the associated entities as a composite key.**
- **Example:**



- Employee(empID , empName , empAddress)
- Project(projectNo , cost , duration)
- Worksfor(empID , projectNo)

e. Unary 1:N Relationship

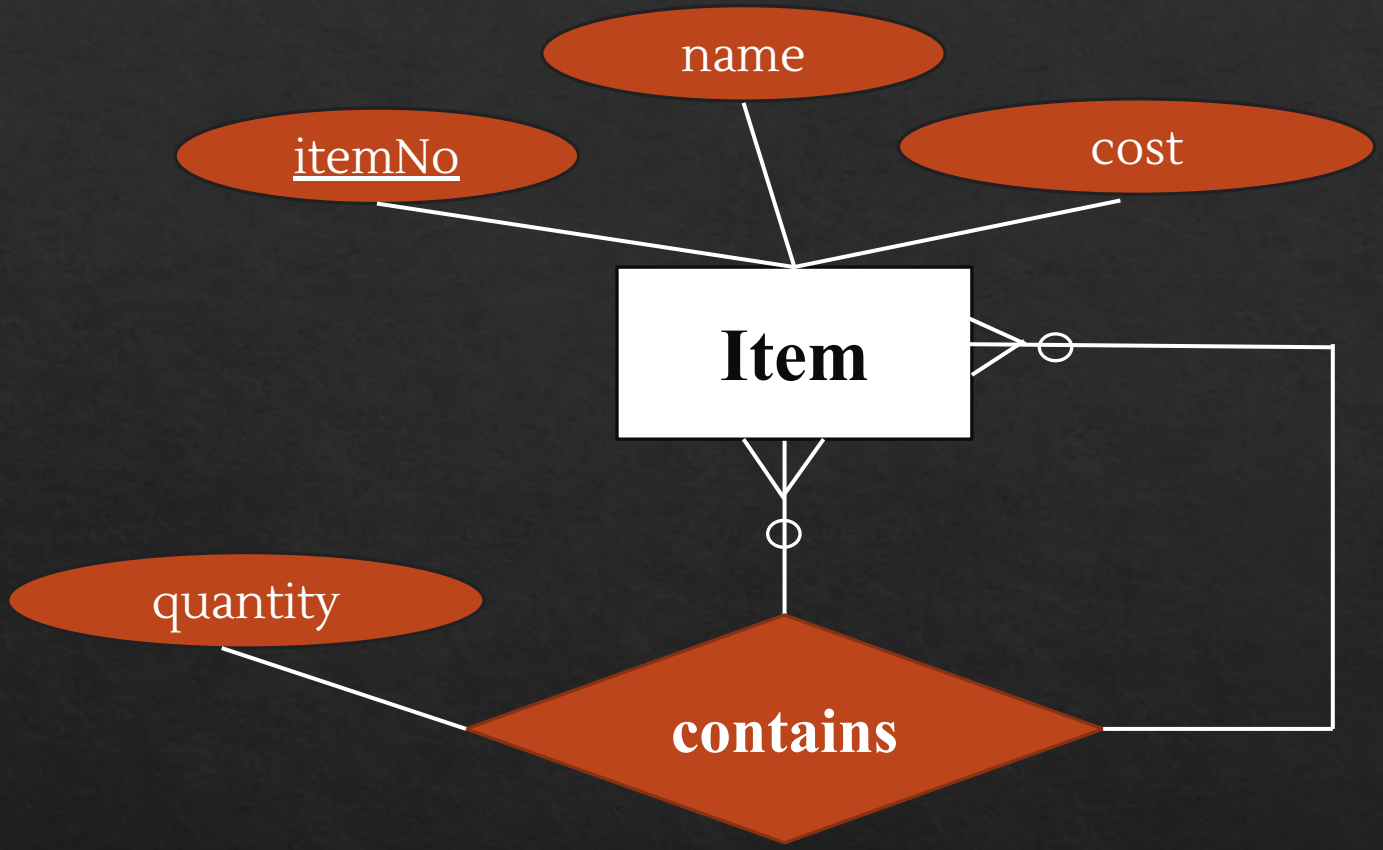


A Recursive Foreign Key is a foreign key that references the primary key values of that same relation. In above example , managerId takes value from Primary key of Employee i.e. empId.

- Employee(empId , empName , empAddress , managerId)

Recursive Foreign Key

f. Unary M:N Relationship



- Item(<u>itemNo</u> , name , cost)
- ItemBill(<u>itemNo</u> , <u>componentNo</u> , Quantity)