REVA UNIVERSITY

Bengaluru, India

School of Computer Science and Applications(I MCA-A)

RELATIONAL DATABASE MANAGEMENT SYSTEM

Course Code: M23DE0103

Prof Nagaraj C

TOPICS DISCUSSED IN UNIT - I

- Database System Applications
- Purpose of Database Systems
- View of Data, Database Languages
- Relational Databases
- Databases Design
- Data Storage and Querying
- Transaction Management
- Database Architecture
- Database Users and Administrator
- Structure of Relational Databases
- Database Schema, Keys
- Schema Diagrams
- Relational Query Languages
- Relational Operations

IN TODAY’S CLASS , WE WILL DISCUSS

- Database Design - Overview
- Design Phases

HIGH LEVEL CONCEPTUAL DATA MODELS FOR DATABASE DESIGN

1. The database design begins with the software requirement specifications of the given problem
2. The precise requirement collection is very important to have a good database design.
3. Next step is to create a conceptual schema that describes the data type,relationships and constraints.This is called conceptual design.

Implementation Model consists of two phases

1.Logical database design

2.Physical database design

HIGH LEVEL CONCEPTUAL DATA MODELS FOR DATABASE DESIGN

The database design phases consists of the following steps:

1.Requirement collection and analysis

2.Conceptual Design

3.Logical Design(Implementation of Data Model)

4.Physical Design

HIGH LEVEL CONCEPTUAL DATA MODELS FOR DATABASE DESIGN

1.Requirement collection and analysis:

During this phase, the database designers interview database users. This helps them to understand and document their data requirements.

2.Conceptual Design:

It describes data requirements of the users, entity types, relationships and constraints.These concepts do not include implementation details.

HIGH LEVEL CONCEPTUAL DATA MODELS FOR DATABASE DESIGN

3.Logical Design:

This step involves implementation of data model.Since conceptual schema is transformed from the high level model into the implementation data model is known as Data Model mapping.

This results in a database schema in the implementation data model such as relational or object oriented database model.

4.Physical Design:

Physical database design is the process of describing the implementation of the database on the disk,internal storage structures,access paths,relations ,security issues and constraints.

HIGH LEVEL CONCEPTUAL DATA MODELS FOR DATABASE DESIGN

Overall Data Base Design involves the following steps:

1. Identifying all the required files
2. Identifying the fields of each of these record types
3. Identifying the primary key of each of these files
4. Identifying the relationship between record types.

HIGH LEVEL CONCEPTUAL DATA MODELS FOR DATABASE DESIGN

Example Database Application:

Let us consider an example database application that demonstrates the basic ER model concepts and their use in schema design.

1.First the data requirements for the database are to be listed

2.Then its conceptual schema should be created step by step.

Consider COMPANY Database.This database keeps track of employees,department and projects.

This company can be organized into departments.Each department can have a unique name and a particular employee who is the manager of the department.

HIGH LEVEL CONCEPTUAL DATA MODELS FOR DATABASE DESIGN

Example Database Application:

Department controls a no.of projects,each of which has a unique name,unique number.

The company database will have the following relations with their respective attributes:

Department:DeptNo,DeptName,Dlocation,Manager

Project:Pno,Pname,Plocation

Employee:Eid,Ename,DeptNo,Addr,Sex,Salary,DOB

Dependent:DepID,DepName,sex,DOB,Relnship,Eid

DESIGN PHASES

- Stated in terms of the entity-relationship model, the conceptual schema specifies
  - The entities that are represented in the database.
  - The attributes of the entities.
  - The relationships among the entities and
  - Constraints on the entities and relationships.

DESIGN PHASES

- Typically, the conceptual-design phase results in
    - the creation of an entity-relationship diagram that provides a graphic representation of the schema.
    - A fully developed conceptual schema also indicates the functional requirements of the enterprise.
- In a **specification of functional requirements**, users describe the kinds of operations (or transactions) that will be performed on the data.
- At this stage of conceptual design, the designer can review the schema to ensure it meets functional requirements.

DESIGN PHASES

In designing a database schema, we must ensure that we avoid two major pitfalls:

- **Redundancy**
- **Incompleteness**

Exercise Questions

- Which are the different phases in database design?
- Which are the tasks involved in Database application design?

THE ENTITY-RELATIONSHIP MODEL

- Developed to facilitate database design by allowing specification of an *enterprise schema* that represents the overall logical structure of a database.
- Very popular modeling tool among many such tools available today.
- Many tools are available for data modeling with E-R. All tools have some variations in representation of components.
- The E- R model provides:

  1. An excellent communication tool.
  2. A simple graphical representation of data.

THE ENTITY-RELATIONSHIP MODEL

- The E-R data model employs three basic concepts:
    - Entity Sets
    - Relationship Sets and
    - Attributes
- The E-R model also has an associated diagrammatic representation,

  - **the E-R diagram.**

THE ENTITY-RELATIONSHIP MODEL

- **Entity -** An entity is a “thing” or “object” in the real world that is distinguishable from all other objects.
- **Entity set -** An entity set is a set of entities of the same type that share the same properties, or attributes.
- An entity is represented by a set of **attributes**.
- Each entity has a **value** for each of its attributes.
- A database thus includes a collection of entity sets, each of which contains any number of entities of the same type.

THE ENTITY-RELATIONSHIP MODEL

Entity Set Example:

R19CA001	Abhilash
R19CA002	Abhay
R19CA003	Aradhya
R19CA004	Bharath
R19CA005	Babu
R19CA006	Chethan

THE ENTITY-RELATIONSHIP MODEL

Relationship Sets:

- A **relationship** is an association among several entities.
- A **relationship set** is a set of relationships of the same type.

Formally, it is a mathematical relation on $n \geq 2$ (possibly nondistinct) entity sets.

If $E1, E2, \ldots, En$ are entity sets, then a relationship set R is a subset of

$$\{ (e1, e2, \ldots, en) \mid e1 \in E1, e2 \in E2, \ldots, en \in En \}$$

where $(e1, e2, \ldots, en)$ is a relationship.

THE ENTITY-RELATIONSHIP MODEL

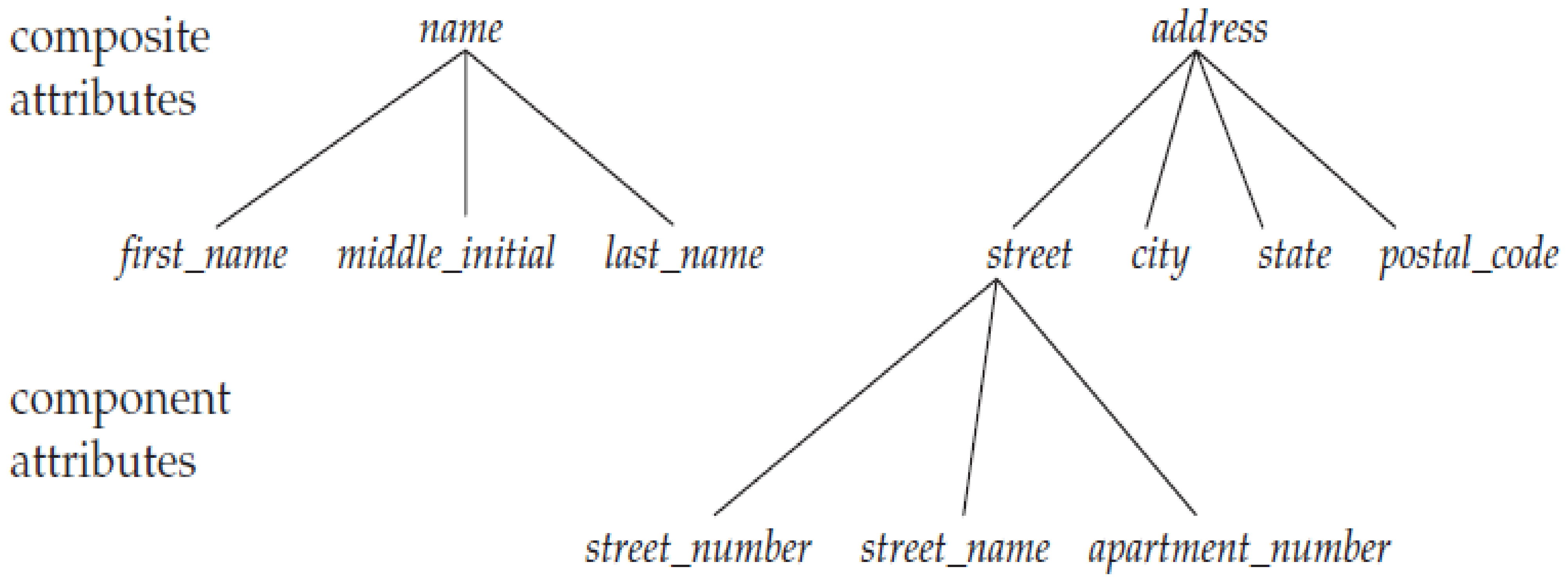
- A **relationship instance** in an E-R schema represents an association between the named entities in the real-world enterprise that is being modeled.
- The function that an entity plays in a relationship is called that entity’s **role**.
- The number of entity sets that participate in a relationship set is the **degree** of the relationship set.

THE ENTITY-RELATIONSHIP MODEL

Attributes

- For each attribute, there is a set of permitted values, called the **domain**, or **value set**, of that attribute.
- An attribute, as used in the E-R model, can be characterized by the following attribute types.
  - **Simple** and **composite** attributes.
- **Simple** Attributes can be divided into subparts.
- **Composite** attributes, on the other hand, can be divided into subparts.

Composite Attributes:



THE ENTITY-RELATIONSHIP MODEL

Single-valued and **multivalued** attributes.

- The attributes have a single value for a particular entity. Such attributes are said to be **single valued**.
- There may be instances where an attribute has a set of values for a specific entity. That type of attribute is said to be **multivalued**.
- **Derived** attribute. The value for this type of attribute can be derived from the values of other related attributes or entities.
- An attribute takes a **null** value when an entity does not have a value for it.
- The *null* value may indicate “not applicable”—that is, that the value does not exist for the entity.

THE ENTITY-RELATIONSHIP MODEL

EXAMPLES

- **Simple attributes**—attributes that cannot be subdivided;

  for example, last name, city, or gender.

- **Composite attributes**—attributes that can be subdivided, into atomic form; **for example,** a full name can be subdivided into the last name, first name, and middle initial.

THE ENTITY-RELATIONSHIP MODEL

EXAMPLES

- **Single-valued attributes**—attributes with a single value;

  for example, Employee ID, Social Security number, or date of birth.

- **Multivalued attributes**—attributes with multiple values;

  for example, degree codes or course registration.

Exercise Questions

- Which are the different types of Attributes?
- Which are the three basic concepts employed in E-R data model?

CONSTRAINTS ON RELATIONSHIP TYPES

- An **E-R** enterprise schema may define certain constraints to which the contents of a database must conform.
- Lets see:
    1. Mapping Cardinalities and
    2. Participation Constraints.

CONSTRAINTS

- **Mapping cardinalities**, or **cardinality ratios**
- express the number of entities to which another entity can be associated via a relationship set.

CONSTRAINTS

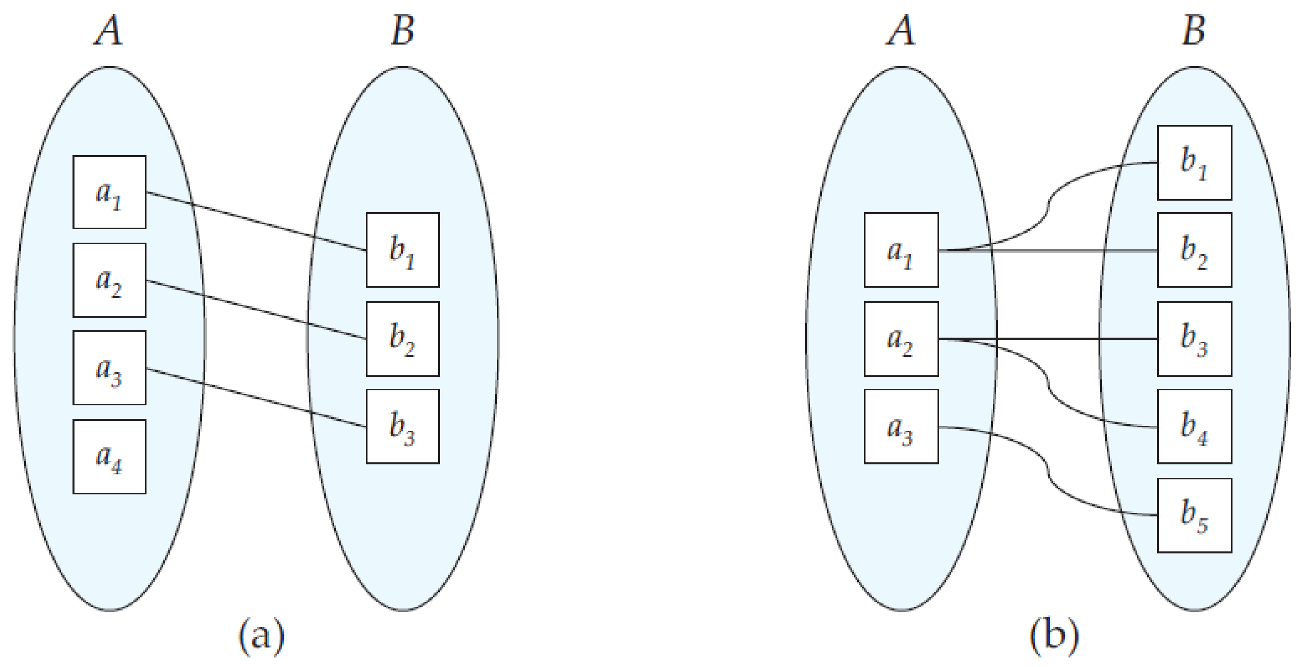
- **Participation Constraints**
- The participation of an entity set *E* in a relationship set *R* is said to be **total** if every entity in *E* participates in at least one relationship in *R*.
- If only some entities in *E* participate in relationships in *R*, the participation of entity set *E* in relationship *R* is said to be **partial**.

RELATIONSHIP BETWEEN ENTITY SET

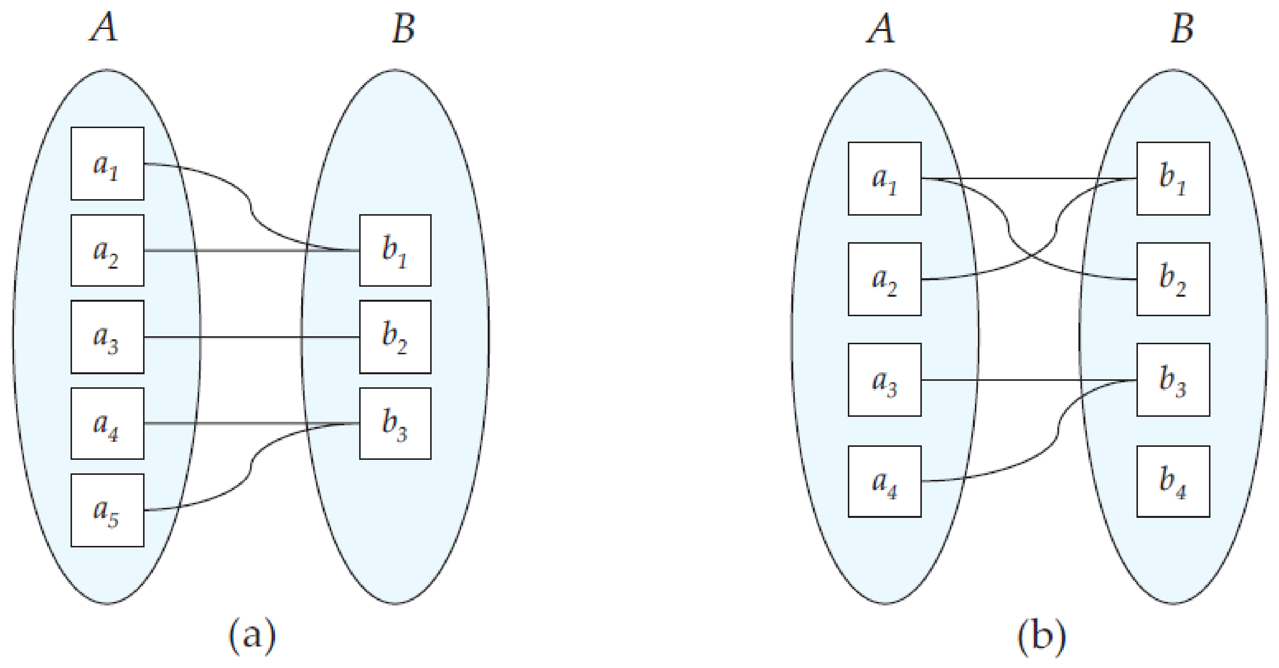
For a binary relationship set *R* between entity sets *A* and *B*, the mapping cardinality must be one of the following:

- **One-to-one**. An entity in *A* is associated with *at most* one entity in *B*, and an entity in *B* is associated with *at most* one entity in *A*.
- **One-to-many**. An entity in *A* is associated with any number (zero or more) of entities in *B*. An entity in *B*, however, can be associated with *at most* one entity in *A*.
- **Many-to-one**. An entity in *A* is associated with *at most* one entity in *B*. An entity in *B*, however, can be associated with any number (zero or more) of entities in *A*.
- **Many-to-many**. An entity in *A* is associated with any number (zero or more) of entities in *B*, and an entity in *B* is associated with any number (zero or more) of entities in *A*.

EXAMPLE:



Mapping cardinalities. (a) One-to-one. (b) One-to-many.



Mapping cardinalities. (a) Many-to-one. (b) Many-to-many.

WHAT ARE KEYS?

- An attribute or set of an attribute which helps you to identify a row(tuple) in a relation(table).
- Allow you to find the relation between two tables.
- Keys help you uniquely identify a row in a table by a combination of one or more columns in that table.

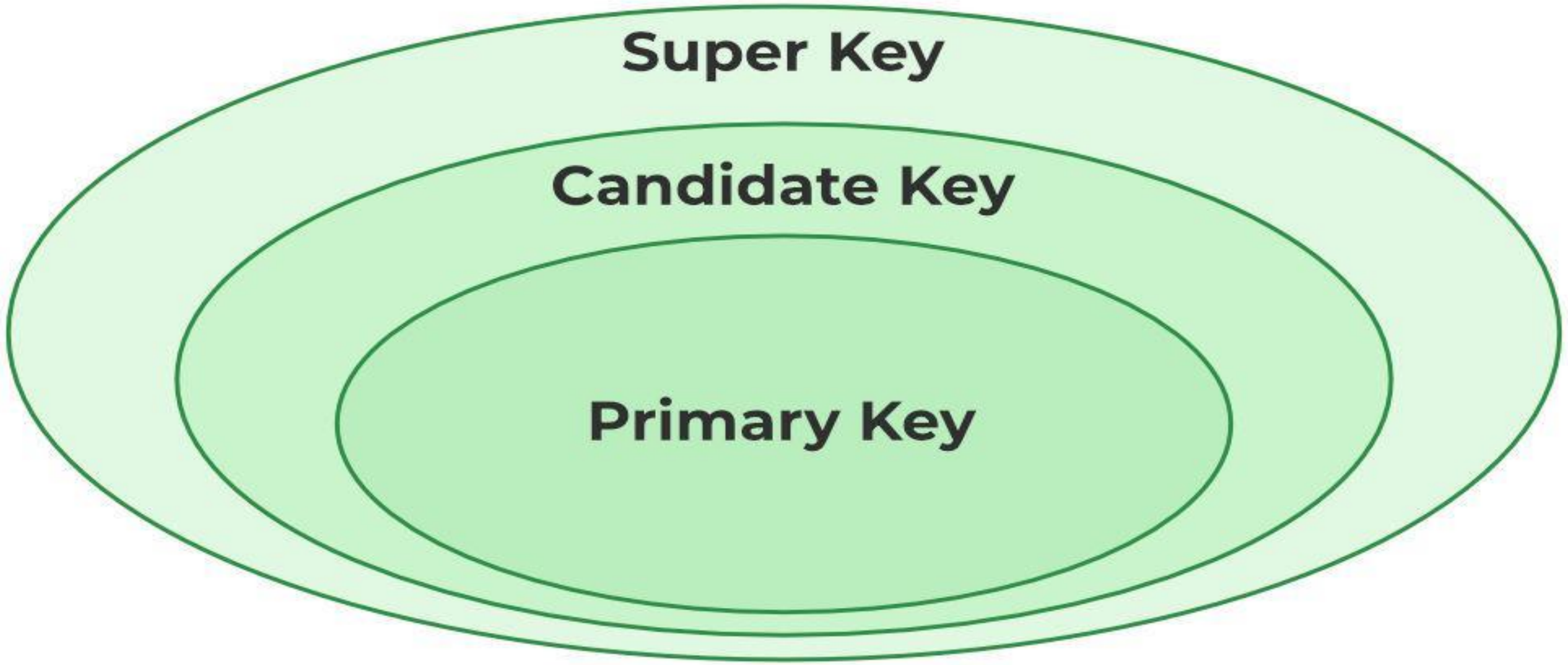
WHY WE NEED A KEY?

- Help you to identify any row of data in a table.
- In a real-world application, a table could contain thousands of records.
- Moreover, the records could be duplicated.
- Keys ensure that you can uniquely identify a table record despite these challenges.
- Allows you to establish a relationship between and identify the relation between tables
- Help you to enforce identity and integrity in the relationship.

VARIOUS KEYS IN DBMS

- Super Key
- Primary Key
- Candidate Key
- Alternate Key
- Foreign Key
- Composite Key
- Surrogate Key

VARIOUS KEYS IN DBMS



VARIOUS KEYS IN RDBMS

Primary Key

There can be more than one candidate key in relation out of which one can be chosen as the primary key. For Example, STUD_NO, as well as STUD_PHONE, are candidate keys for relation STUDENT but STUD_NO can be chosen as the primary key (only one out of many candidate keys).

- A **primary key** is a **unique key**, meaning it can uniquely identify each record (tuple) in a table.
- It must have **unique values** and cannot contain any **duplicate** values.
- A **primary key cannot be NULL**, as it needs to provide a valid, unique identifier for every record.
- A primary key does not have to consist of a single column. In some cases, a **composite primary key** (made of multiple columns) can be used to uniquely identify records in a table.
- Databases typically store rows ordered in memory according to primary key for fast access of records using primary key.

VARIOUS KEYS IN RDBMS

Alternate Key

An alternate key is any candidate key in a table that is **not** chosen as the **primary key**. In other words, all the keys that are not selected as the primary key are considered alternate keys.

•An alternate key is also referred to as a **secondary key** because it can uniquely identify records in a table, just like the primary key.

•An alternate key can consist of **one or more columns** (fields) that can uniquely identify a record, but it is not the primary key

•Eg:- PAN, and AADHAR is Alternate keys

VARIOUS KEYS IN RDBMS

Candidate Key

The minimal set of attributes that can uniquely identify a tuple is known as a candidate key. For Example, STUD_NO in STUDENT relation.

- A candidate key is a minimal super key, meaning it can uniquely identify a record but contains no extra attributes.
- The minimal set of attributes that can uniquely identify a record.
- A candidate key must contain unique values, ensuring that no two rows have the same value in the candidate key’s columns.
- Every table must have at least a single candidate key.
- A table can have multiple candidate keys but only one primary key.

Artificial Key

An **Artificial Key** is artificially created by the system when no suitable **natural key** or **candidate key** is available.

It is usually an **auto-incremented** numeric value (like an ID column) that uniquely identifies each record in a table.

VARIOUS KEYS IN RDBMS

Super Key

The set of one or more attributes (columns) that can uniquely identify a tuple (record) is known as Super Key.

For Example, STUD_NO, (STUD_NO, STUD_NAME), etc.

•A super key is a group of single or multiple keys that uniquely identifies rows in a table. It supports NULL values in rows.

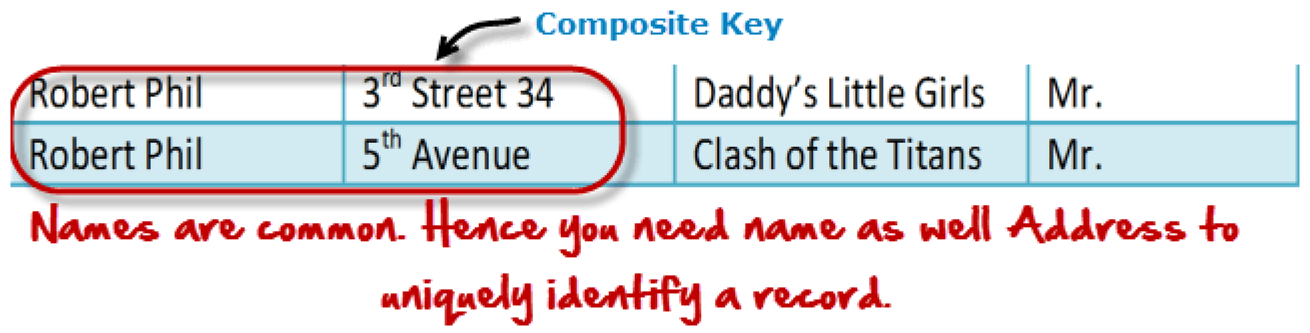
•A super key can contain extra attributes that aren't necessary for uniqueness. For example, if the "STUD_NO" column can uniquely identify a student, adding "SNAME" to it will still form a valid super key, though it's unnecessary.

Composite Key

•A combination of two or more columns that uniquely identifies a row.

•Used when a single column cannot uniquely identify a record.

•Example: (*Course_ID, Student_ID*) in an **Enrollment** table.



Composite Key

Robert Phil	3rd Street 34	Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

Names are common. Hence you need name as well Address to uniquely identify a record.

VARIOUS KEYS IN RDBMS

Surrogate Key

•A system-generated unique key, usually an **auto-increment** field.(In Oracle IDENTITY)
•Used when no natural primary key exists.
•Example: *Order_ID* in an **Orders** table.

Foreign Key

A foreign key is an attribute in one table that refers to the **primary key** in another table. The table that contains the foreign key is called the **referencing table**, and the table that is referenced is called the **referenced table**.

•A **foreign key** in one table points to the **primary key** in another table, establishing a relationship between them.

•It helps **connect two or more tables**, enabling you to create relationships between them. This is essential for maintaining data integrity and preventing data redundancy.

•They act as a cross-reference between the tables.

•For example, DNO is a primary key in the DEPT table and a non-key in EMP

Summary

- Constraints
    - Mapping Cardinalities and
    - Participation Constraints.
- Keys
- Types of Keys

Exercise Questions

- Which are the different types of keys?
- Different mapping cardinalities.

REMOVING REDUNDANT ATTRIBUTES IN ENTITY SETS

- When we design a database using the E-R model, we usually start by identifying those entity sets that should be included.

- Once the entity sets are decided upon, we must choose the appropriate attributes.

- These attributes are supposed to represent the various values we want to capture in the database.

REMOVING REDUNDANT ATTRIBUTES IN ENTITY SETS

- Once the entities and their corresponding attributes are chosen, the relationship sets among the various entities are formed.

- These relationship sets may result in a situation where attributes in the various entity sets are redundant and need to be removed from the original entity sets.

- A good entity-relationship design does not contain redundant attributes.

EXAMPLE:

- **classroom**: with attributes (*building*, *room number*, *capacity*).
- **department**: with attributes (*dept name*, *building*, *budget*).
- **course**: with attributes (*course id*, *title*, *credits*).
- **instructor**: with attributes (*ID*, *name*, *salary*).
- **section:** with attributes (*course id*, *sec id*, *semester*, *year*).
- **student**: with attributes (*ID*, *name*, *tot cred*).
- **time slot**: with attributes (*time slot id*, {(*day*, *start time*, *end time*) }).

The relationship sets in our design are listed below:

- **inst_dept**: relating instructors with departments.
- **stud_dept**: relating students with departments.
- **teaches**: relating instructors with sections.
- **takes**: relating students with sections, with a descriptive attribute *grade*.
- **course_dept**: relating courses with departments.
- **sec_course**: relating sections with courses.
- **sec_class**: relating sections with classrooms.
- **sec_time_slot**: relating sections with time slots.
- **advisor**: relating students with instructors.
- **prereq**: relating courses with prerequisite courses.

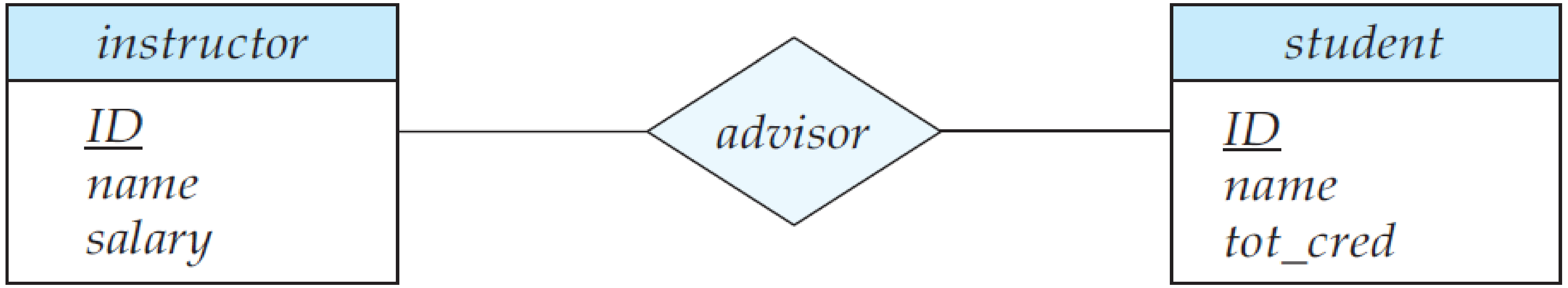
Exercise Questions

1. ....is a ‘thing’ in the real world with an independent existence.
A) Entity
B) Attribute
C) Key
D) Relationship
ANSWER: A

2. The relational model feature is that there
(A) is no need for primary key data.
(B) is much more data independence than some other database models.
(C) are explicit relationships among records.
(D) are tables with many dimensions.
ANSWER: B

ENTITY-RELATIONSHIP DIAGRAMS

- An **E-R diagram** can express the overall logical structure of a database graphically.
- E-R diagrams are simple and clear.

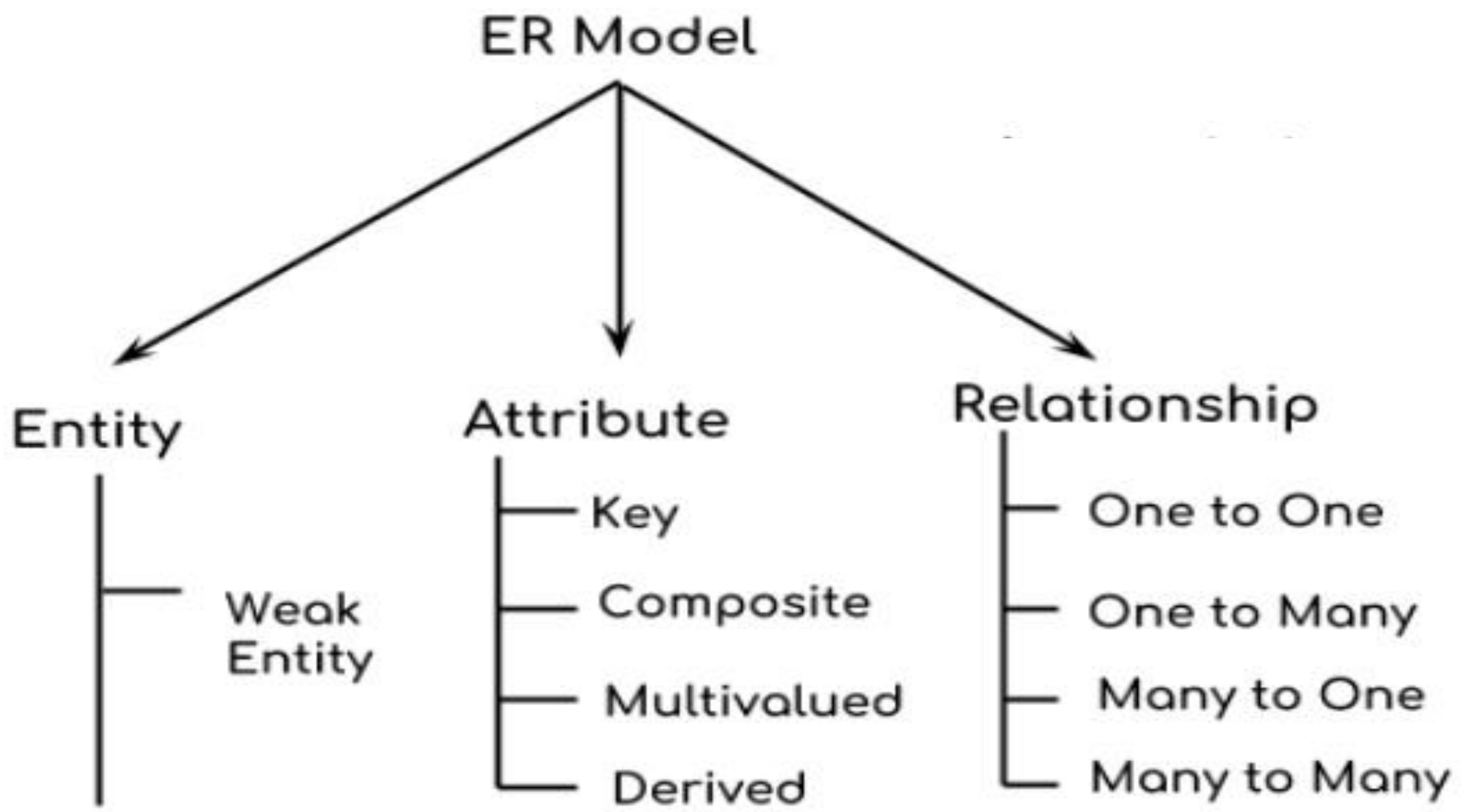


E-R diagram corresponding to instructors and students.

An E-R diagram consists of the following major components:

- **Rectangle**: Represents Entity sets.
- **Ellipses**: Attributes
- **Diamonds**: Relationship Set
- **Lines**: They link attributes to Entity Sets and Entity sets to Relationship Set
- **Double Ellipses:** Multivalued Attributes
- **Dashed Ellipses**: Derived Attributes
- **Double Rectangles**: Weak Entity Sets
- **Double Lines**: Total participation of an entity in a relationship set

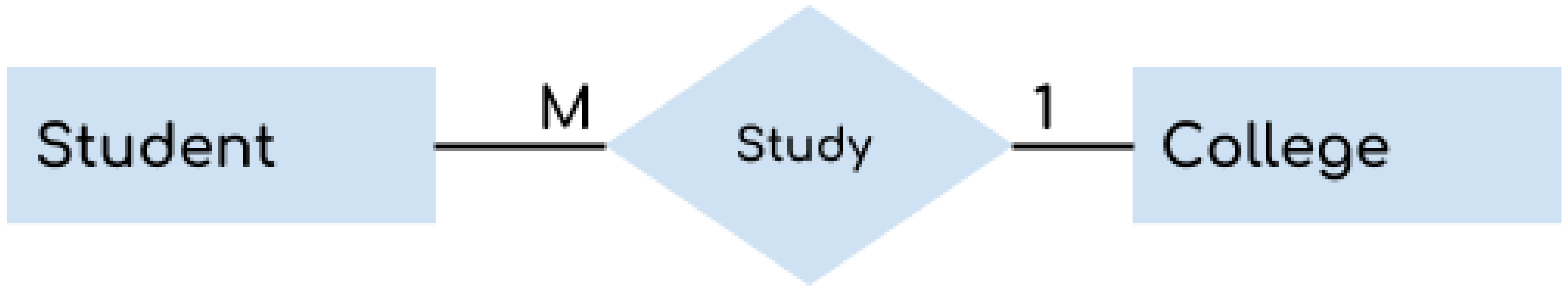
Components of a ER Diagram



Entity

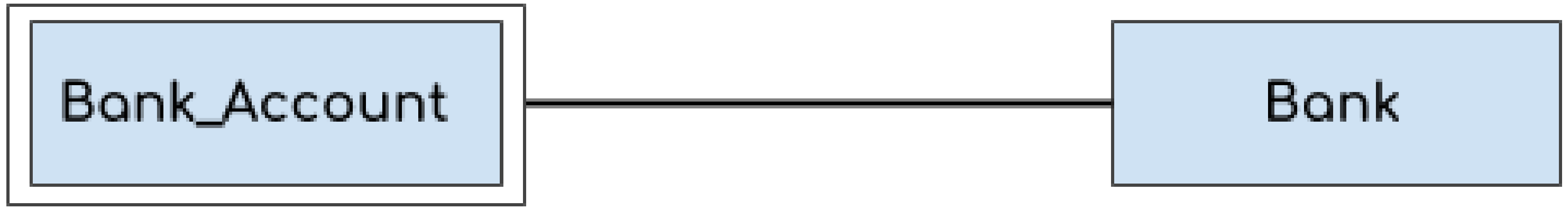
- An object or component of data.
- An entity is represented as rectangle in an ER diagram.

For example:



Weak Entity:

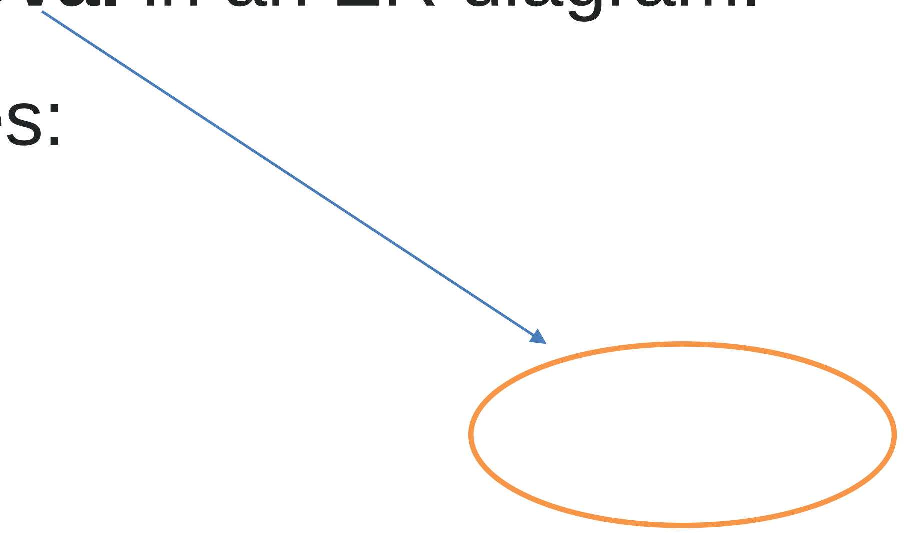
- An entity that cannot be uniquely identified by its own attributes and relies on the relationship with other entity is called **weak entity**.
- The weak entity is represented by a **double rectangle.**
- **For example** – a bank account cannot be uniquely identified without knowing the bank to which the account belongs, so bank account is a weak entity.



ATTRIBUTE

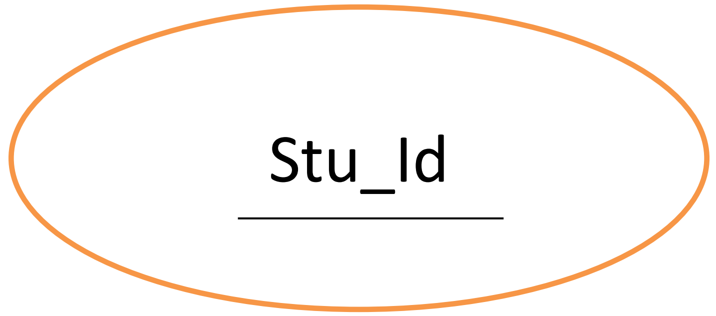
- An attribute describes the property of an entity.
- An attribute is represented as **Oval** in an ER diagram.
- There are four types of attributes:

  1. Key attribute
  2. Composite attribute
  3. Multivalued attribute
  4. Derived attribute



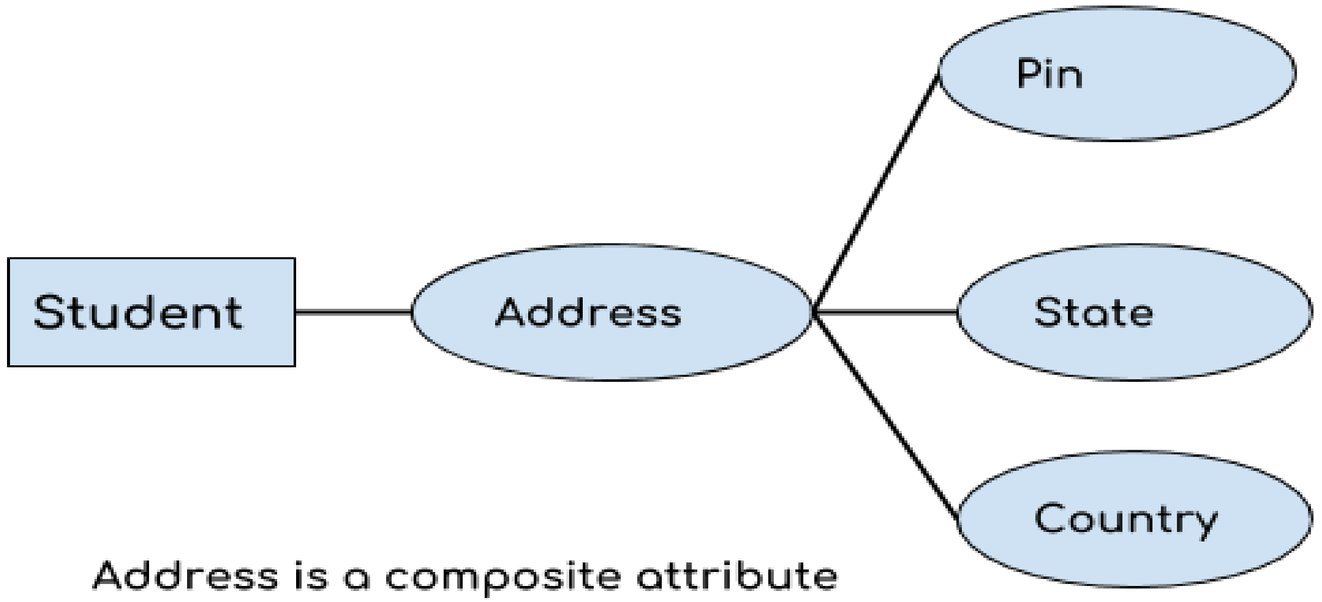
Key attribute:

- A key attribute can uniquely identify an entity from an entity set.
- For example, student roll number can uniquely identify a student from a set of students.
- Represented by oval same as other attributes however the **text of key attribute is underlined**.



Composite attribute:

- An attribute that is a combination of other attributes
- For example, In student entity, the student address is a composite attribute as an address is composed of other attributes such as pin code, state, country.

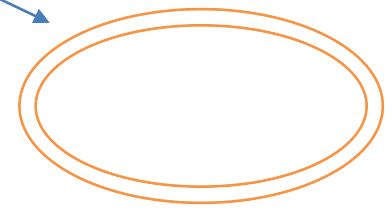


Address is a composite attribute

ATTRIBUTE

Multivalued attribute:

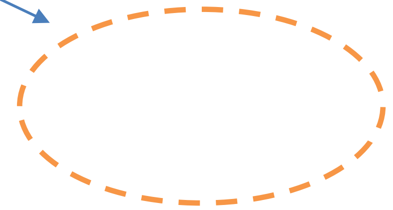
- An attribute that can hold **multiple values** is known as multivalued attribute.
- It is represented with **double ovals** in an ER Diagram.
- For example – A person can have more than one phone numbers so the phone number attribute is multivalued.



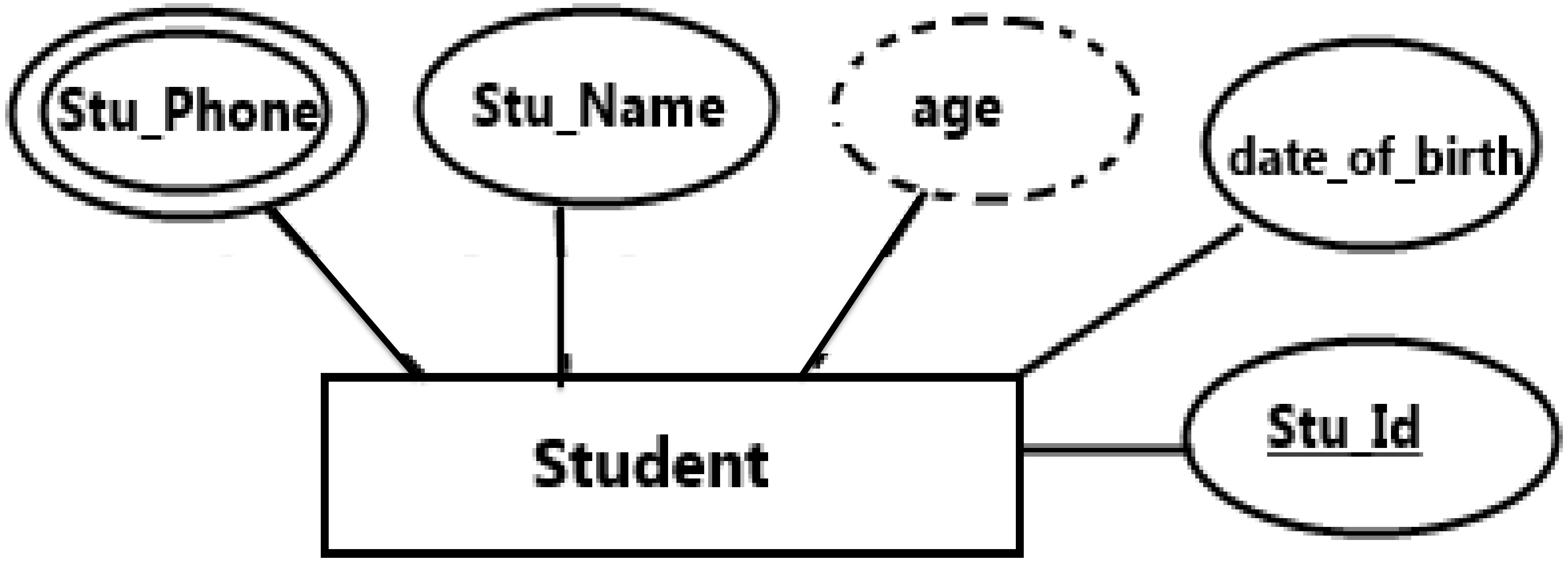
ATTRIBUTE

Derived attribute:

- A derived attribute is one whose value is dynamic and derived from another attribute.
- It is represented by **dashed oval** in an ER Diagram.
- For example – Person age is a derived attribute as it changes over time and can be derived from another attribute (Date of birth).



ATTRIBUTE



E-R diagram with multivalued and derived attributes:

Relationship

- Represented by diamond shape in ER diagram
- Shows the relationship among entities.
- Four types -

  1. One to One

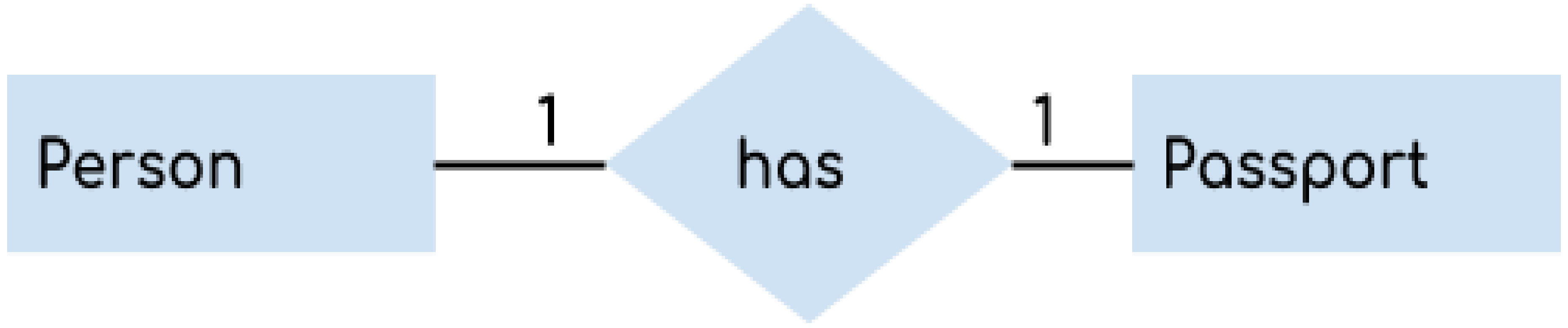
  2. One to Many

  3. Many to One

  4. Many to Many

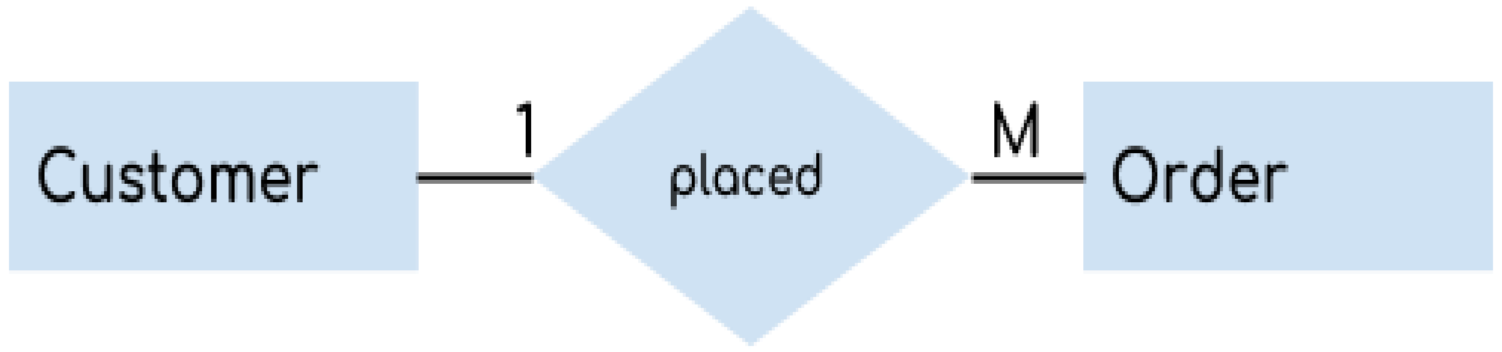
One to One Relationship

- When a single instance of an entity is associated with a single instance of another entity then it is called one to one relationship.
- For example, a person has only one passport and a passport is given to one person.



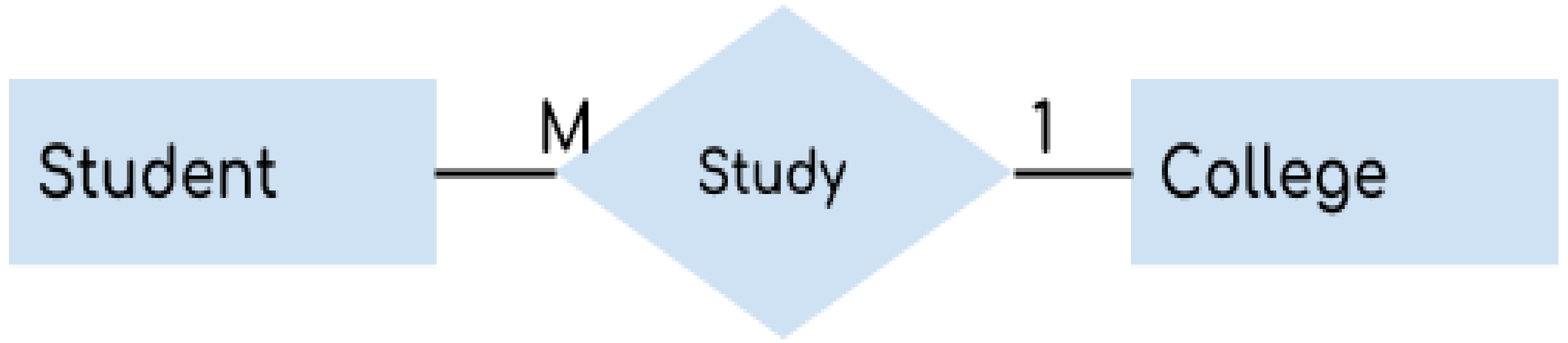
One to Many Relationship

- When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationship.
- **For example** – a customer can place many orders but a order cannot be placed by many customers.



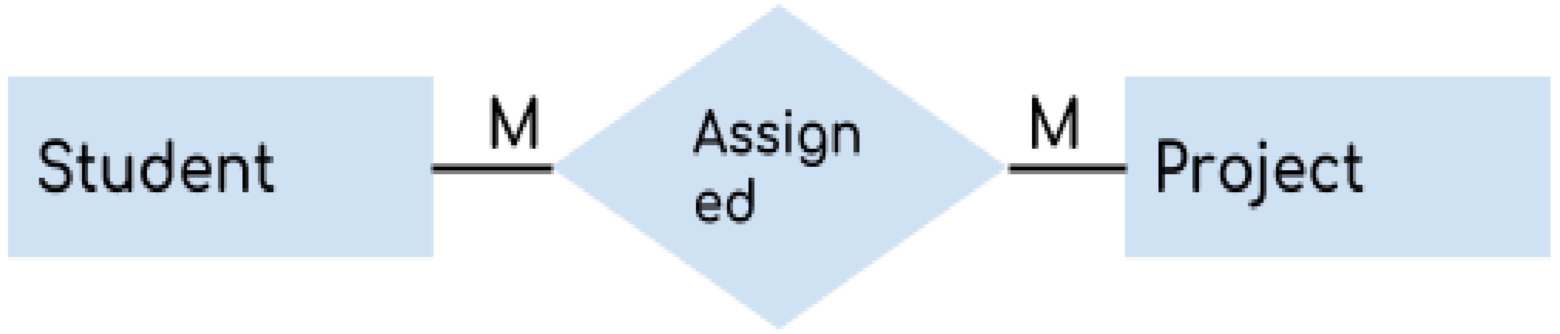
Many to One Relationship

- When more than one instances of an entity is associated with a single instance of another entity then it is called many to one relationship.
- **For example** – many students can study in a single college but a student cannot study in many colleges at the same time.



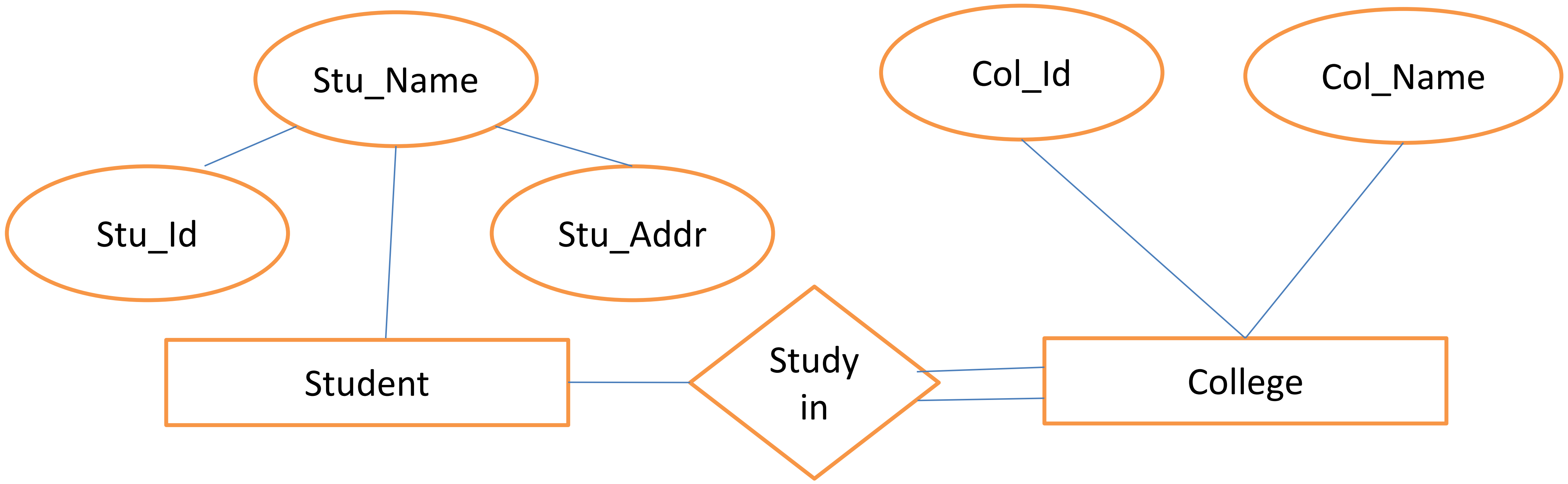
Many to Many Relationship

- When more than one instances of an entity is associated with more than one instances of another entity then it is called many to many relationship.
- **For example**, a can be assigned to many projects and a project can be assigned to many students.

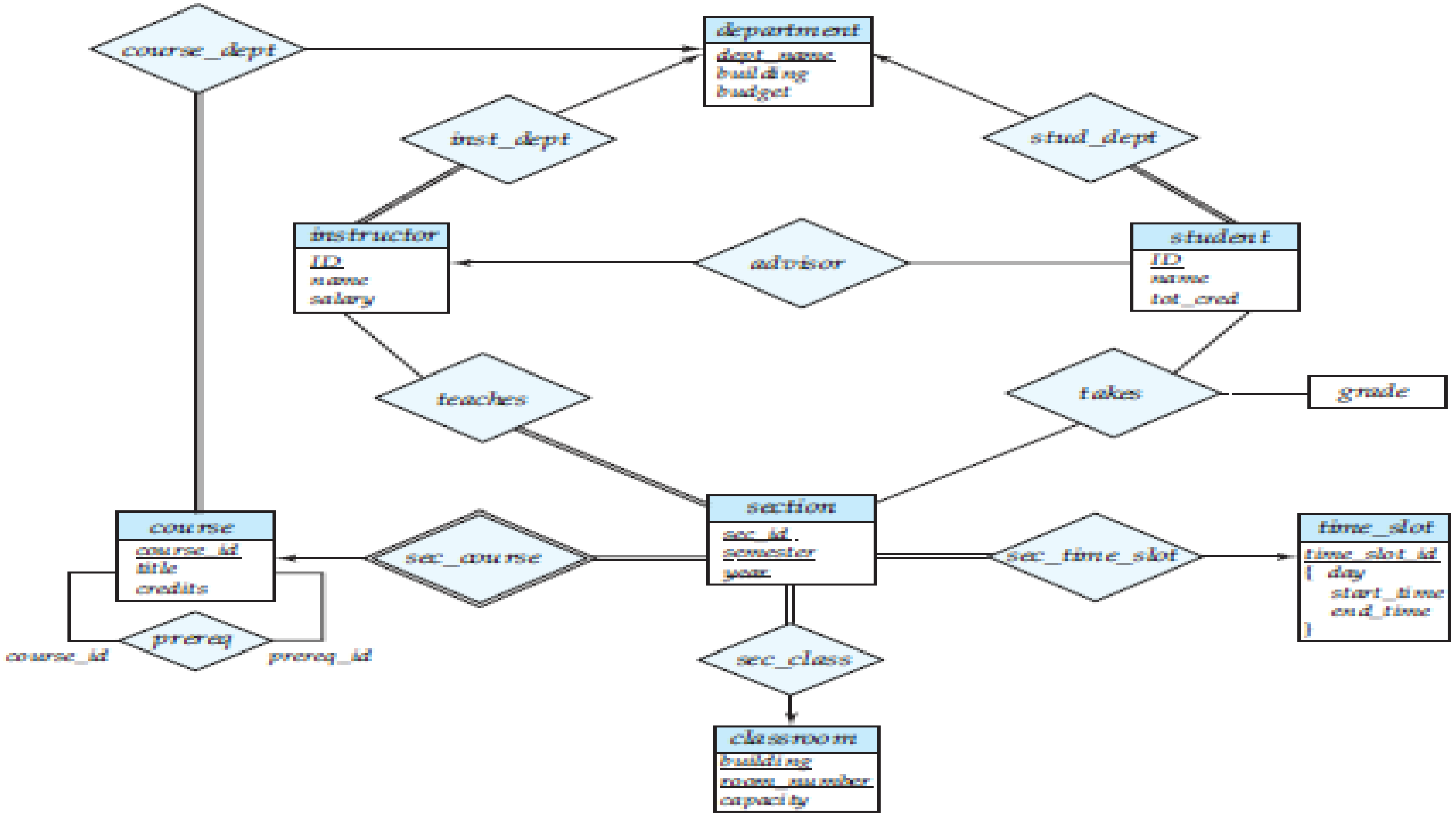


Total Participation of an Entity set

- A Total participation of an entity set represents that each entity in entity set must have at least one relationship in a relationship set.
- **For example**: In the below diagram each college must have at-least one associated Student.



EXAMPLE


course_dept
department
dept_name
building
budget
inst_dept
stud_dept
instructor
ID
name
salary
advisor
student
ID
name
tot_cred
teaches
takes
grade
course
course_id
title
credits
sec_course
section
sec_id
semester
year
sec_time_slot
time_slot
time_slot_id
{ day
start_time
end_time
}
prereq
course_id
prereq_id
sec_class
classroom
building
room_number
capacity

Exercise Questions

E-R model uses this symbol to represent weak entity set ?
(A) Dotted rectangle.
(B) Diamond
(C) Doubly outlined rectangle
(D) None of these
ANSWER: C

In E-R Diagram relationship type is represented by
(A) Ellipse
(B) Dashed ellipse
(C) Rectangle
(D) Diamond
ANSWER: D

Reduction to Relational Schemas

- Both the E-R model and the relational database model are abstract, logical representations of real-world enterprises.
- Because the two models employ similar design principles, we can convert an E-R design into a relational design.

- **Representation of Strong Entity Sets with Simple Attributes**
- **Representation of Strong Entity Sets with Complex Attributes**
- **Representation of Weak Entity Sets**
- **Representation of Relationship Sets**
    - **Redundancy of Schemas**
    - **Combination of Schemas**

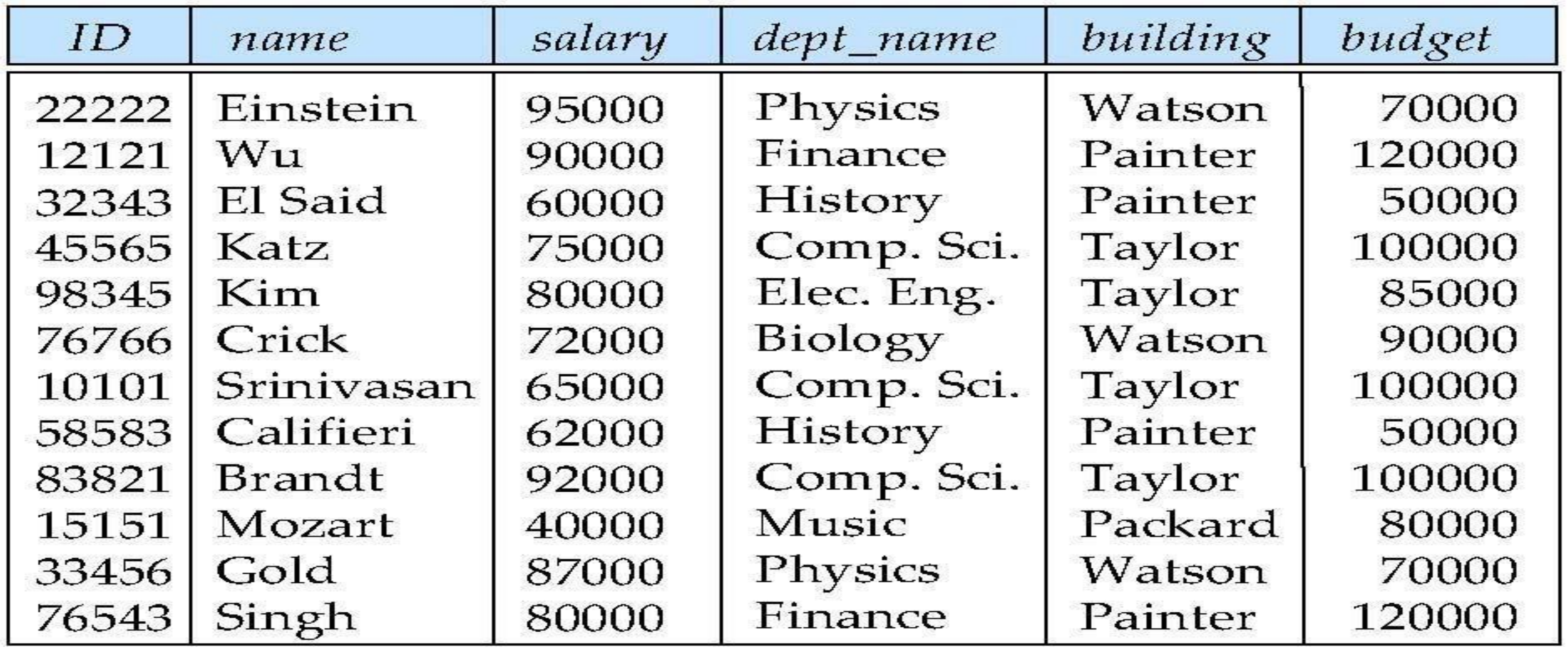
Relational Database Design

- The goal of Relational Database Design is to generate a set of relation schemas that allows us to store information without unnecessary redundancy, yet also allows us to retrieve information easily.
- This is accomplished by designing schemas that are in an appropriate ***normal form***.

Features of Good Relational Designs

Combine Schemas?

- Suppose we combine *instructor* and *department* into *inst_dept*
  - *(No connection to relationship set inst_dept)*
- Result is possible repetition of information



ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Features of Good Relational Designs

A Combined Schema without Repetition

Consider combining relations

- *sec_class(sec_id, building, room_number)* and
- *section(course_id, sec_id, semester, year)*

into one relation

- *section(course_id, sec_id, semester, year, building, room_number)*

❖No repetition in this case

Features of Good Relational Designs

What About Smaller Schemas?

- ❖Suppose we had started with *inst_dept.* How would we know to split up (decompose) it into *instructor* and *department*?
- ❖Write a rule —if there were a schema (*dept_name, building, budget*), then *dept_name* would be a candidate key‖
- ❖Denote as a functional dependency:

  dept_name → *building, budget*

Features of Good Relational Designs

What About Smaller Schemas?

- ❖In *inst_dept*, because *dept_name* is not a candidate key, the building and budget of a department may have to be repeated.

  This indicates the need to decompose *inst_dept*

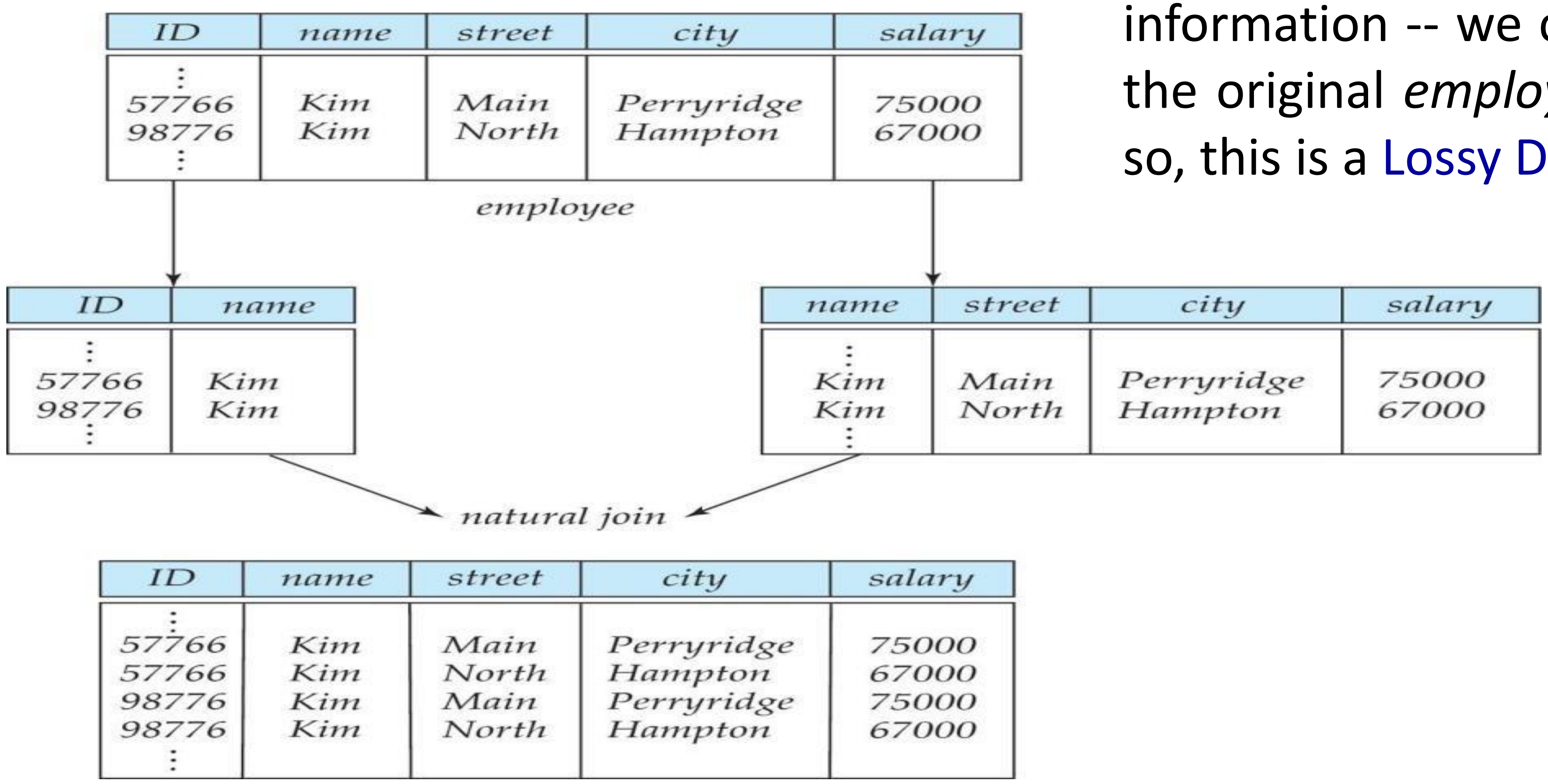
- ❖Not all decompositions are good. Suppose we decompose

  employee(ID, name, street, city, salary) into

  employee1 (*ID*, *name*)

  employee2 (*name*, *street, city, salary*)

A Lossy Decomposition



ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

employee

ID	name
⋮	
57766	Kim
98776	Kim
⋮	

name	street	city	salary
⋮			
Kim	Main	Perryridge	75000
Kim	North	Hampton	67000
⋮			

natural join

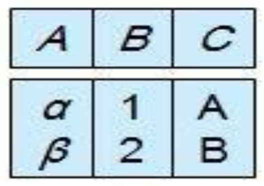
ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
57766	Kim	North	Hampton	67000
98776	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

The figure shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is a Lossy Decomposition.

Example of Lossless-Join Decomposition

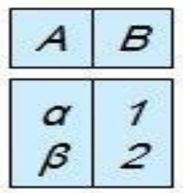
- **Lossless join decomposition**
- **Decomposition of *R = (A, B, C)***

R_1 = (A, B) R_2 = (B, C)



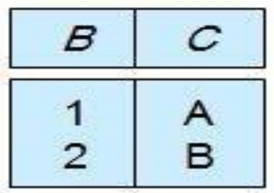
A	*B*	*C*
α	1	A
β	2	B

r



A	*B*
α	1
β	2

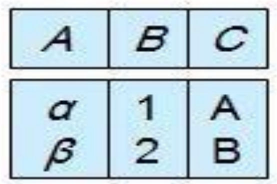
$\Pi_{A,B}(r)$



B	*C*
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A\ (r) \bowtie \Pi_B\ (r)$



A	*B*	*C*
α	1	A
β	2	B

DEPENDENCY

- You learned that the primary key in a table identifies an entity.
- Every table in the database should have a primary key, which uniquely identifies an entity.
- For example, PartNo is a primary key in the PARTS table, and DeptNo is a primary key in the DEPARTMENT table.
- You should define a primary key for all tables for integrity of data.

DEPENDENCY

- Each table has other columns that do not make up the primary key for the table.
- Such columns are called the non-key columns. The nonkey columns are functionally dependent on the primary key column.
- For example, PartDesc and Cost in the PARTS table are dependent on the primary key PartNo, and DeptName is dependent on the primary key DeptNo in the DEPARTMENT table.

INVOICE

Inv No	InvDate	CustN	ItemNo	CustName	ItemName	ItemPrice	Qty
1001	04/14/20	212	1	Mutthu	Screw	Rs. 2.25	5
1001	04/14/20	212	3	Pandian	Bolt	Rs.3.99	5
1001	04/14/20	212	5	Ramesh	Washer	Rs.1.99	9
1002	04/17/20	225	1	Suresh	Screw	Rs.2.25	2
1002	04/17/20	225	2	Ravi	Nut	Rs.5.00	3
1003	04/17/20	239	1	Kapur	Screw	Rs.2.25	7
1003	04/17/20	239	2	Kapur	Nut	Rs.5.00	1
1004	04/18/20	211	4	Kohli	Hammer	Rs.9.99	5

INVOICE

- The INVOICE table in does not have any single column that can uniquely identify an entity. The first choice would be InvNo.
- It is not a unique value in the table, however, because an invoice may contain more than one item and there may be more than one entry for an invoice.
- CustNo cannot be the primary key, because there can be many invoices for a customer and CustNo does not identify an invoice.

INVOICE

- ItemNo cannot be the primary key either, because an item may appear in more than one invoice and ItemNo does not describe an invoice.
- **The table has a composite primary key, which consists of InvNo and ItemNo.**
- InvNo and ItemNo together make up unique values for each row. All other columns that do not constitute the primary key are nonkey columns, and they are dependent on the primary key.

Functional Dependency

There are three types of dependencies in a table:

- ***Total or full dependency*:** A nonkey column dependent on all primary key columns shows total dependency.
- ***Partial dependency*:** In partial dependency, a nonkey column is dependent on part of the primary key.
- ***Transitive dependency*:** In transitive dependency, a nonkey column is dependent on another nonkey column.

Functional Dependency

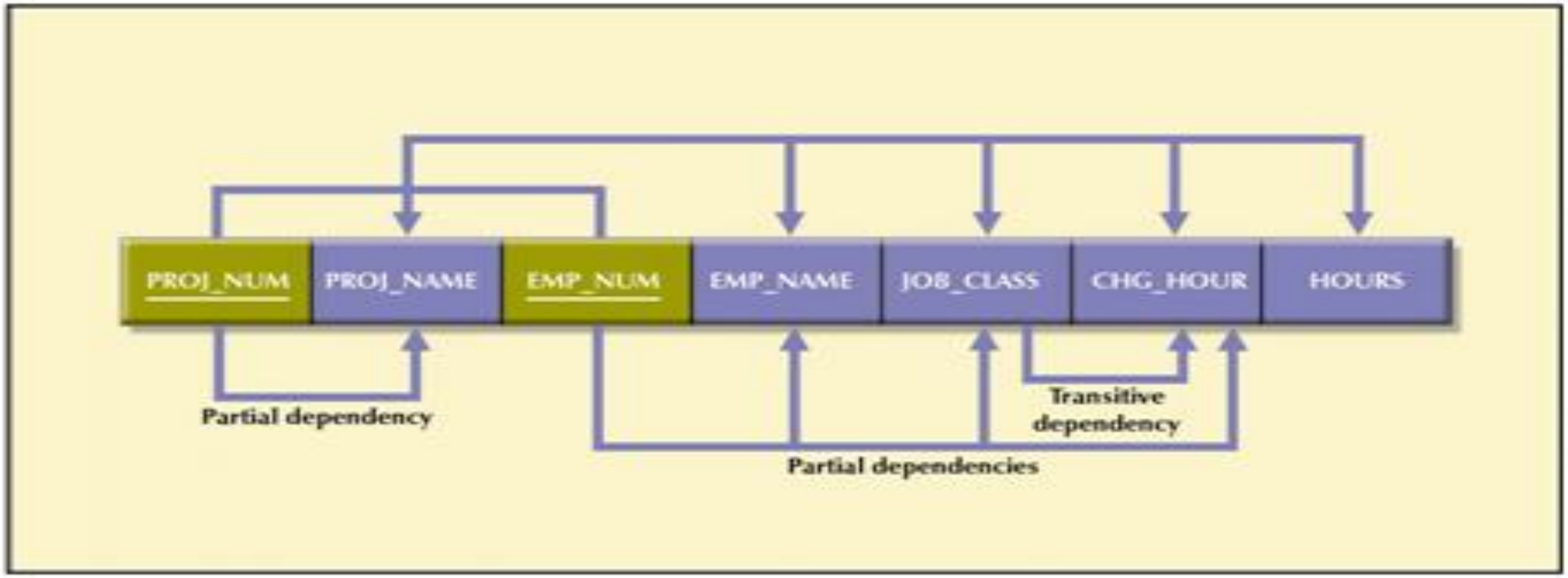
- For example, in the INVOICE table, ItemName and ItemPrice are nonkey columns that are dependent only on a part of the primary key column ItemNo. They are not dependent on the InvNo column. It is said to have **Partial dependency.**
- Similarly, the nonkey column InvDate is dependent only on InvNo. They are partially dependent on the primary key columns.

Functional Dependency

- The nonkey column CustName is not dependent on any primary key column but is dependent on another nonkey column, CustNo. It is said to have **transitive dependency**.
- The nonkey column Qty is dependent on both InvNo and ItemNo, so it is said to have **full dependency**.

Functional Dependency

- A relationship between attributes in which one or more attributes determines the value of another attribute(s) in the same table
  - The Proj_Num “determines” the Proj_Name



Exercise Questions

- What is Functional Dependency?
- What are its Types?

Anomalies

- Database anomaly is normally the flaw in databases which occurs because of poor planning and storing everything in a flat database.
- Generally this is removed by the process of normalization which is performed by splitting/joining of tables.

Anomalies

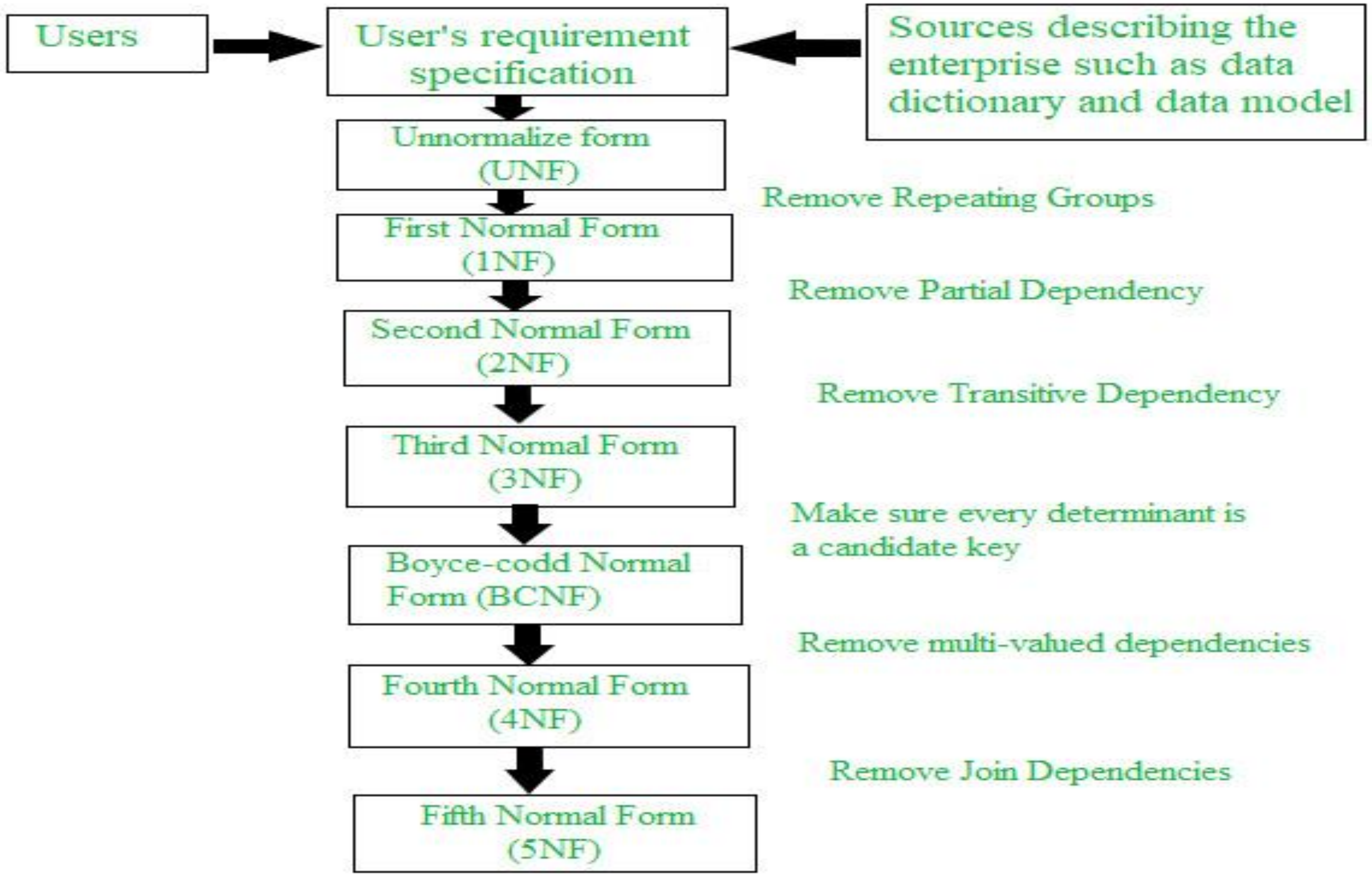
The general **definition** of an **anomaly** is something that you don't expect.

- An **insertion anomaly** occurs when the information about an entity cannot be inserted unless the information about another entity is known.
- A **deletion anomaly** results when the deletion of information about one entity leads to the deletion of information about another entity.
- An **update anomaly** can occurs when **redundant data** in a poorly normalized database causes **inconsistent or partial updates**, leading to **data integrity issues**.This happens when multiple occurrences of the same data exist in a table, and an update to one instance does not automatically update all related instances.

Normalization IN RDBMS

- **Normalization** is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization rules divides larger tables into smaller tables and links them using relationships.
- The purpose of Normalization in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.
- The inventor of the relational model Edgar F Codd proposed the theory of normalization of data with the introduction of the First Normal Form, and he continued to extend theory with Second and Third Normal Form.
- Later he joined Raymond F. Boyce to develop the theory of Boyce-Codd Normal Form.

Normalization IN RDBMS


Users
User's requirement specification
Sources describing the enterprise such as data dictionary and data model
Unnormalize form (UNF)
Remove Repeating Groups
First Normal Form (1NF)
Remove Partial Dependency
Second Normal Form (2NF)
Remove Transitive Dependency
Third Normal Form (3NF)
Make sure every determinant is a candidate key
Boyce-codd Normal Form (BCNF)
Remove multi-valued dependencies
Fourth Normal Form (4NF)
Remove Join Dependencies
Fifth Normal Form (5NF)

Normalization

ADVANTAGES OF NORMAL FORM

- **Reduced data redundancy:** Normalization helps to eliminate duplicate data in tables, reducing the amount of storage space needed and improving database efficiency.
- **Improved data consistency:** Normalization ensures that data is stored in a consistent and organized manner, reducing the risk of data inconsistencies and errors.
- **Simplified database design:** Normalization provides guidelines for organizing tables and data relationships, making it easier to design and maintain a database.
- **Improved query performance:** Normalized tables are typically easier to search and retrieve data from, resulting in faster query performance.
- **Easier database maintenance:** Normalization reduces the complexity of a database by breaking it down into smaller, more manageable tables, making it easier to add, modify, and delete data.

Normalization

1ST NORMAL FORM (1NF) – ELIMINATE REPEATING GROUPS

A table is said to be in **first normal form**, or can be labeled INF, if the following conditions exist:

1. All **columns contain atomic (single) value**.
2. Each column contains values of **a single type**.
3. There are **no duplicate rows**.

Example of a Non-1NF Table

Student_ID	Name	Subjects
101	Hashvik	Math, Science
102	Jashvika	English, History

Converted to 1NF

Student_ID	Name	Subject
101	Hashvik	Math
101	Hashvik	Science
102	Jashvika	English
102	Jashvika	History

❌ **Problem**: The **Subjects** column has multiple values.

✅ **Solution**: Each **cell contains a single value**, removing **repeating groups**.

Normalization

SECOND NORMAL FORM (2NF)– REMOVE PARTIAL DEPENDENCY

- A table is said to be in second normal form, or 2NF, if the following requirements are satisfied:
  - All 1NF requirements are fulfilled.
  - All non-key attributes are fully dependent on the primary key.There is no partial dependency.

Normalization

EXAMPLE OF A NON-2NF TABLE

Student_ID	Subject	Teacher	Teacher_Age
101	Math	Mr. A	45
101	Science	Mrs. B	38
102	English	Mr. C	50

✗ **Problem**: **Teacher_Age** depends only on **Teacher**, not **Student_ID + Subject**.

CONVERTED TO 2NF (SPLITTING INTO TWO TABLES)

Student_ID	Subject	Teacher
101	Math	Mr. A
101	Science	Mrs. B
102	English	Mr. C

T_ID	Teacher	Teacher_Age
1	Mr. A	45
2	Mrs. B	38
3	Mr. C	50

✓ **Solution**: Now, **Teacher_Age** depends only on **Teacher ID**, avoiding **partial dependency**.

3RD NORMAL FORM (3NF) – REMOVE TRANSITIVE DEPENDENCY

A table is in **3NF** if:

1. It is **already in 2NF**.
2. **There is no transitive dependency** (i.e., non-key attributes should not be dependent on another non-key attributes, Its only on the primary key).

Example of a Non-3NF Table

Student_ID	Name	City	Zip_Code	State
101	Aju	Coimbatore	641018	TN
102	Dheeraj	Bangalore	560001	KA

❌ **Problem**: **City and State depend on Zip_Code, not Student_ID** (transitive dependency).

3RD NORMAL FORM (3NF) – REMOVE TRANSITIVE DEPENDENCY

Converted to 3NF

Students Table

Student_ID	Name	Zip_Code
101	Aju	641018
102	Dheeraj	560001

Zip_Codes Table

Zip_Code	City	State
641018	Coimbatore	TN
560001	Bangalore	KA

✓ **Solution**: Now, **City and State depend only on Zip_Code**, removing **transitive dependency**.

Normal Form	Condition	Example Fix
1NF	No repeating groups	Break multi-value columns into separate rows
2NF	No partial dependency	Move columns that don’t fully depend on the primary key to another table
3NF	No transitive dependency	Move indirectly related columns to a new table

WHAT IS FUNCTIONAL DEPENDENCY IN DBMS?

Functional dependency in DBMS is an important concept that describes the relationship between attributes (columns) in a table. It shows that the value of one attribute determines the other.

It is denoted as $X \rightarrow Y$, where the attribute set on the left side of the arrow, X is called Determinant, and Y is called the Dependent.

Example:

Suppose we have a student table with attributes: Stu_Id, Stu_Name, Stu_Age. Here Stu_Id attribute uniquely identifies the Stu_Name attribute of student table because if we know the student id we can tell the student name associated with it. This is known as functional dependency and can be written as Stu_Id→Stu_Name or in words we can say Stu_Name is functionally dependent on Stu_Id.

Functional Dependency

There are three types of dependencies in a table:

- ***Total or full dependency:*** A nonkey column dependent on all primary key columns shows total dependency.
- ***Partial dependency:*** In partial dependency, a nonkey column is dependent on part of the primary key.
- ***Transitive dependency:*** In transitive dependency, a nonkey column is dependent on another nonkey column.

Roll_no	Name	dept_name	dept_building
42	RAVI	CO	A4
43	RAM	IT	A3
44	SSS	CO	A4
45	xyz	IT	A3
46	mno	EC	B2
47	jkl	ME	B2

From the above table we can conclude some valid functional dependencies:

- roll_no → { name, dept_name, dept_building }→ Here, roll_no can determine values of fields name, dept_name and dept_building, hence a valid Functional dependency
- roll_no → dept_name , Since, roll_no can determine whole set of {name, dept_name, dept_building}, it can determine its subset dept_name also.
- dept_name → dept_building , Dept_name can identify the dept_building accurately, since departments with different dept_name will also have a different dept_building
- More valid functional dependencies: roll_no → name, {roll_no, name} ⇢ {dept_name, dept_building}, etc.

1. **Here are some invalid functional dependencies:**

- name → dept_name   Students with the same name can have different dept_name, hence this is not a valid functional dependency.
- dept_building → dept_name   There can be multiple departments in the same building. Example, in the above table departments ME and EC are in the same building B2, hence dept_building → dept_name is an invalid functional dependency.

Normalization

BOYCE CODD NORMAL FORM (BCNF)

- It is an advance version of 3NF that’s why it is also referred as 3.5NF.
- BCNF is stricter than 3NF.
- A table complies with BCNF if it is in 3NF and for every functional dependency

  X->Y, X should be the super key of the table.

Converted to BCNF

Non-BCNF Table

$S,C \rightarrow T$

$S,T \rightarrow C$ $T \rightarrow C$

Student	Course	Tutor
101	JAVA	Sandeep
101	Python	Sarathi
102	JAVA	Sandeep
102	Python	Sathvik

Student Table

Student	Tutor
101	Sandeep
101	Sarathi
102	Sandeep
102	Sathvik

Course Table

Course	Tutor
JAVA	Sandeep
Python	Sarathi
Python	Sathvik

MULTI-VALUED DEPENDENCY (MVD)

A **multi-valued dependency** exists when one attribute in a table can have multiple values **independently** of another attribute.

This can lead to unnecessary duplication and redundancy.

4TH NORMAL FORM (4NF)

What is 4th Normal Form (4NF)?

The **4th Normal Form (4NF)** is an advanced level of database normalization that deals with **multi-valued dependencies**. It ensures that a database table does not contain more than one **independent multi-valued dependency**.

A table is in **4NF** if:

1. It is already in **3rd Normal Form (3NF)**.
2. It has **no multi-valued dependencies**, meaning that if one attribute in a table has multiple values that are independent of another attribute, those values should be stored in separate tables.

Example of a Table NOT in 4NF:

Consider a university database storing **professors, courses they teach, and the committees they are part of**.

Professor	Course	Committee
Dr. Smith	DBMS	Research Board
Dr. Smith	DBMS	Ethics Board
Dr. Smith	AI	Research Board
Dr. Smith	AI	Ethics Board
Dr. Jones	ML	Finance Board
Dr. Jones	ML	Ethics Board

Problem:

- A professor can teach **multiple courses**.
- A professor can be part of **multiple committees**.
- These two facts are **independent** but stored together, causing **data redundancy**.

HOW TO CONVERT TO 4NF?

To remove the **multi-valued dependency**, we divide the table into two **separate** tables.

Table 1: Professor-Course

Professor	Course
Dr. Smith	DBMS
Dr. Smith	AI
Dr. Jones	ML

Table 2: Professor-Committee

Professor	Committee
Dr. Smith	Research Board
Dr. Smith	Ethics Board
Dr. Jones	Finance Board
Dr. Jones	Ethics Board

INFORMAL DESIGN GUIDELINES FOR RELATIONAL SCHEMA AND THEIR SIGNIFICANCE

When designing a **relational database schema**, informal design guidelines help ensure **efficiency, consistency, and data integrity**. These guidelines prevent issues such as **data redundancy, anomalies, excessive NULL values, and poor organization**.

1. Avoid Redundant Data (Minimize Redundancy)

Guideline:

Ensure that data is **not repeated unnecessarily** in a table to **reduce storage costs** and prevent inconsistencies.

Significance:

- ✓ Reduces **storage requirements**.
- ✓ Prevents **data inconsistency** during updates.
- ✓ Improves **database performance**.

EXAMPLE:

1. Consider the following unnormalized table:

Emp_ID	Emp_Name	Dept_ID	Dept_Name	Dept_Location
101	Raghu	D01	HR	Mangalore
102	Varun	D02	IT	Chennai
103	Tejas	D03	testing	Mangalore

Issue:

- The **Dept_Name and Dept_Location** are **repeated** for every employee in the same department.
- Updating a department's location requires **updating multiple rows**, leading to **update anomalies**.

Solution (Normalization - Splitting Tables):

Employee Table

Emp_ID	Emp_Name	Dept_ID
101	Raghu	D01
102	Varun	D02
103	Tejas	D01

Department Table

Dept_ID	Dept_Name	Dept_Location
D01	HR	Mangalore
D02	IT	Chennai

💡 **Now, the department details are stored separately, eliminating redundancy.**

2. REDUCE UPDATE, INSERTION, AND DELETION ANOMALIES

A well-structured schema should prevent **anomalies** that occur during updates, insertions, or deletions.

Significance:

✓ Prevents **loss of critical data** during deletion.
✓ Ensures **consistency in data modification**.
✓ Simplifies **data management**.

Example (Deletion Anomaly):

If an **EMPLOYEE** and **DEPARTMENT** are stored in the same table, deleting the last employee in a department **removes the department information** entirely.

💡 **Solution:** Separating related entities into different tables prevents accidental data loss.

3. AVOID EXCESSIVE NULL VALUES

A schema should avoid having **too many NULL values**, as they waste space and create confusion.

Significance:

- ✓ Reduces **wasted storage space**.
- ✓ Prevents **misinterpretation of NULL values**.
- ✓ Improves **query performance**.

Example:

Consider a STUDENT table where some students have multiple contact numbers, while others do not.

Student_ID	Name	Phone_1	Phone_2
201	Alex	9876543210	NULL
202	Ben	NULL	NULL
203	Cara	9234567890	9123456789

Issue:

- Many NULL values exist because **not all students have a second phone number**.

Solution (Separate Table for Phone Numbers):

Student Table

Student_ID	Name
201	Alex
202	Ben
203	Cara

Student_Phone Table

Student_ID	Phone_Number
201	9876543210
203	9234567890
203	9123456789

💡 **This eliminates excessive NULL values while allowing multiple phone numbers per student.**

4. ENSURE SCHEMA IS EASY TO UNDERSTAND & USE

The schema should be **simple, meaningful, and easy to interpret**.

Significance:

✅ Helps developers and users **easily understand** the database structure.
✅ Ensures **clear relationships** between tables.
✅ Reduces **complexity in queries**.

Example (Good vs. Bad Naming Conventions)

❌ **Bad Schema:**

TBL_01(E_ID, E_NM, D_ID, D_NM, D_LOC)

✅ **Good Schema:**

1. EMPLOYEE(Emp_ID, Emp_Name, Dept_ID)
2. DEPARTMENT(Dept_ID, Dept_Name, Dept_Location)

💡 **Using meaningful names improves readability and maintainability.**

REVA UNIVERSITY

THANK YOU