

Basics of DBMS

A database is a computer based record keeping system whose over all purpose is to record and maintain information.

The Role of DBMS

Real World



(DBMS) Database Management System

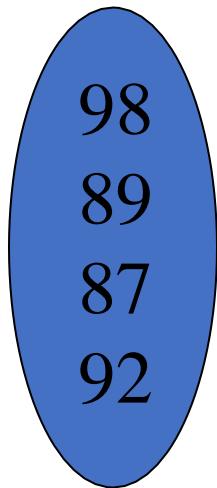
- Term Database requires understanding of **DATA** and **INFORMATION**

What is Data ?

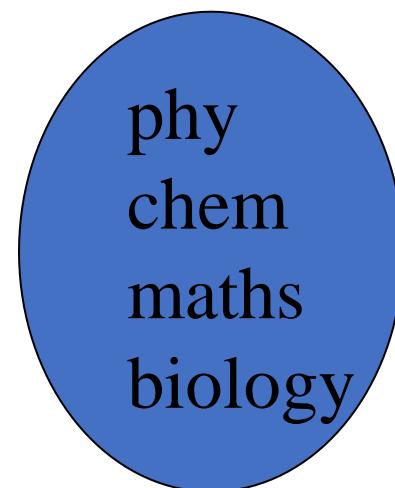
Data is stored facts.

Data may be **numerical data** which may be integers or floating point numbers, and **non-numerical data** such as **characters, date and etc.,**

Eg :



98
89
87
92

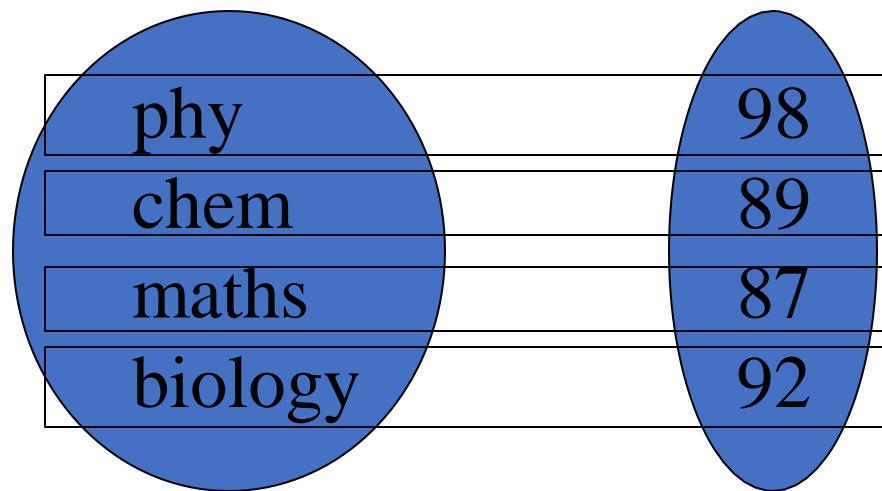


phy
chem
maths
biology

The above numbers may be anything: It may be distance in kms or amount in rupees or no of days or marks in each subject etc.,

What is information ?

Information is RELATED DATA



Difference between Data and Information?

Data

Data is raw fact and figures.

For example: 23 is data.

Data is not significant to a business and of itself.

Data are atomic level pieces of information.

For example in the healthcare industry, much activity surrounds data collection. Nurses collect data every day and sometimes hourly. Examples of data include vital signs, weight, and relevant assessment parameters.

Data does not help in decision making.

3/3/2025

UCS310: DBMS

Information

Information is a processes form of data.

For example: When 23 is stored in row column form as shown below in become information:

Age 23

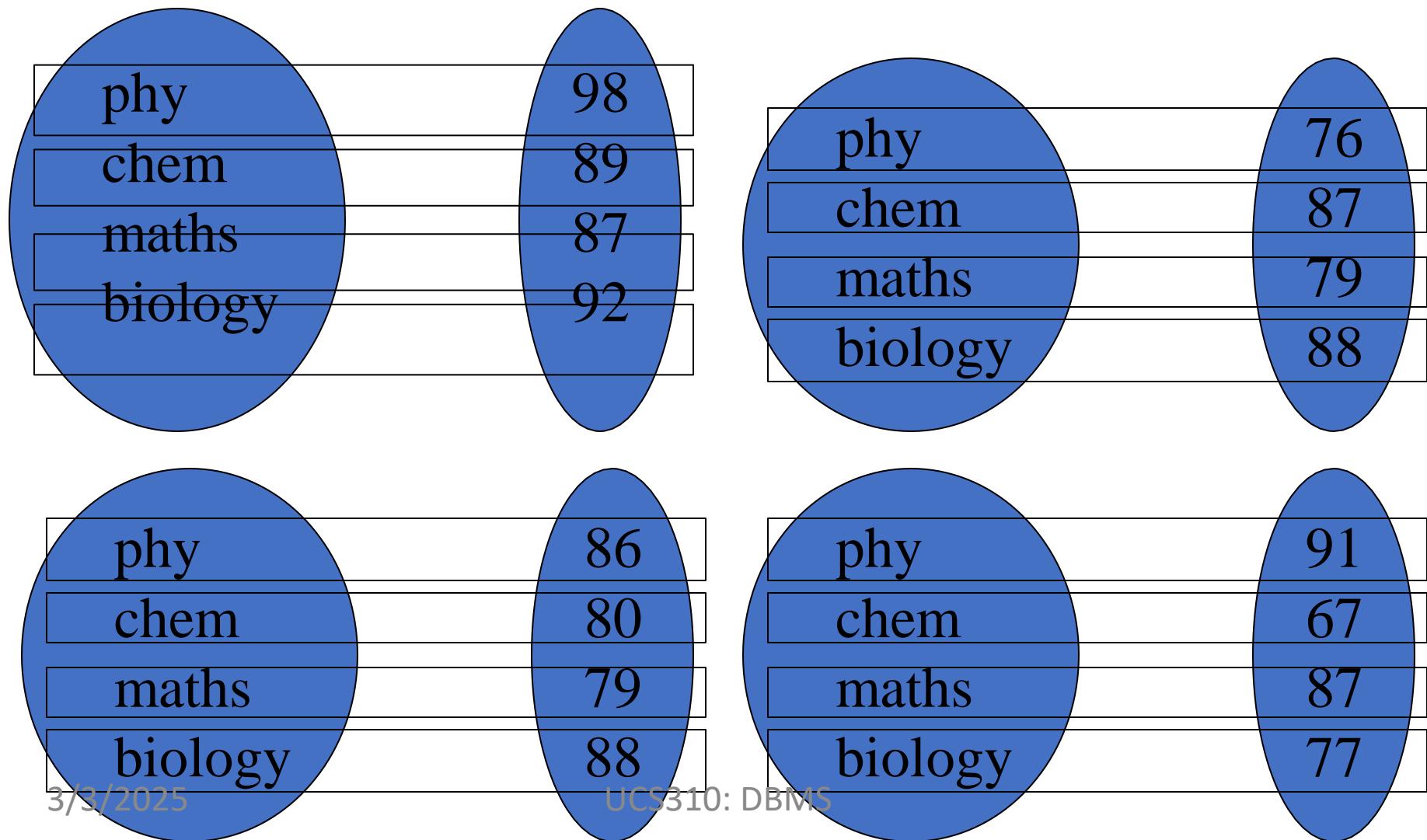
Information is significant to a business and of itself; for example 23 is insignificant for business but age 23 is significant for a business like music.

Information is a collection of data, for example age and 23 collected together to form information.

Information, however, provides answers to questions that guide clinicians to change their practices. For example, the trending of vital signs over time provides a pattern that may lead to certain clinical decisions.

As explained above information helps in decision-making.

DATABASE



What is DBMS ?

(DataBase Management System)

- A collection of programs that enables you to **store, modify, and extract information** from a database.
- The general purpose of a DBMS is to provide for the **definition, storage, and management of data** that can be shared by many users.
- E.g. ORACLE's database management system is patterned on the relational model described by Dr. E. F. Codd.

So.....

- The related information when placed is an organized form makes a database.
- Database is a collection of information organized in such a way that a computer program can quickly select desired pieces of data.
- The organization of data/information is necessary because unorganized information has no meaning.

Operations on Databases

- To add new information
- To view or retrieve the stored information
- To modify or edit the existing information
- To remove or delete the unwanted information
- Arranging the information in a desired order
- etc.

Manual database and its problems

- Wastage of skills and intelligence of human beings on repetitive calculations.
- Error prone.

Database and Computers

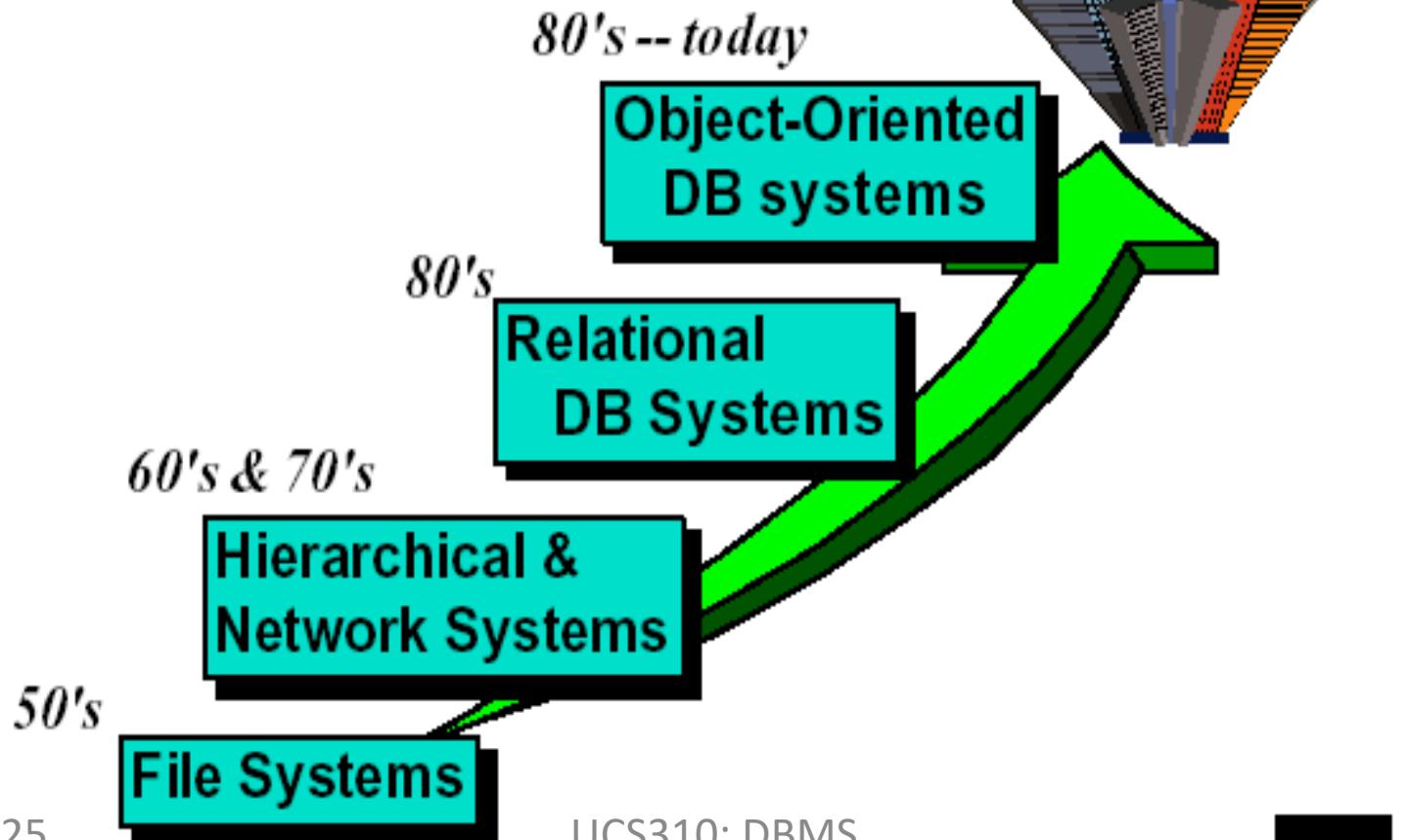
- Large storage capacity
- It has high speed
- Computer is more accurate.

*There are two approaches for storing data in computers such as **File based approach** and*

Database approach.

Evolution of DBMS

Closing the gap



File Based Approach

- File-based systems were an **early attempt to computerize** the manual filing system that we are all familiar with.
- File systems may use a **storage device** such as a hard disk or CD-ROM and involve **maintaining the physical location of the files**.
- Programmers used programming languages such as C, C++ etc. to write applications that **directly accessed flat files** to perform data management services and provide information for users.

File Systems (1950's)

□ Basic constructs:

- ◆ Sequential records
- ◆ A record contains sequential fields

□ Relies on indexes for random access

ISAM (Indexed Sequential Access Method)

VSAM (Virtual Storage Access Method)

□ Basic operations

- ◆ Open, close, or reset a file,
- ◆ Read, write, or delete a record.

'John', 27 Main Street (234)987-2314
'Lisa', 26 Peach Street (345)987-2314
'Peter', 32 Main Street (876)234-9734
'David', 36 Peachtree (885)-349-3418
...
'Linda', 25 Circle Dr. (234)987-2314

Limitations of the File-Based Approach

- Separated and Isolated Data
- Duplication of data
- Data Dependence
- Difficulty in representing data from the user's view
- Data Integrity
- Incompatible file formats
- Data Security

- Uncontrolled data redundancy, data inconsistency**
- Poor data sharing**
- Difficult to keep up with changes**
 - ◆ Record format vs. user requirements
 - ◆ Programs vs. record format
- Low productivity**
- High maintenance cost**

Database Approach

- Database is a collection of information organized in such a way that a computer program can quickly select desired pieces of data.
- A DBMS is the software system that allows users to define, create and maintain a database and provides controlled access to the data.
- A database management system (DBMS) is basically a collection of programs that enables users to store, modify, and extract information from a database as per the requirements.

Examples of Database Applications

The following are main examples of database applications:

- **Computerized library systems**
- **Automated teller machines**
- **Flight reservation systems**
- **Computerized inventory systems**
- **Banking Systems**
- **Smart Hospital Systems**

Example

University Database in File Based System

General Office	Library	Hostel	Account Office
Rollno	Rollno	Rollno	Rollno
Name	Name	Name	Name
Class	Class	Class	Class
Father_Name	Address	Father Name	Address
Date_of_birth	Date of birth	Date of birth	Phone_No
Address	Phone No	Address	Fee
Phone_No	No of books issued	Phone No	Installments
Previous_Record	Fine	Mess_Bill	Discount
Attendance	etc.	RoomNo	Balance
Marks		etc.	Total
Etc.			etc.

Advantages of DBMS

➤ Controlling Redundancy

No duplication

➤ Integrity can be enforced

Integrity of data means that data in database is always accurate, such that incorrect information cannot be stored in database.

➤ Inconsistency can be avoided

When the same data is duplicated and changes are made at one site, which is not propagated to the other site, it gives rise to inconsistency and the two entries regarding the same data will not agree.

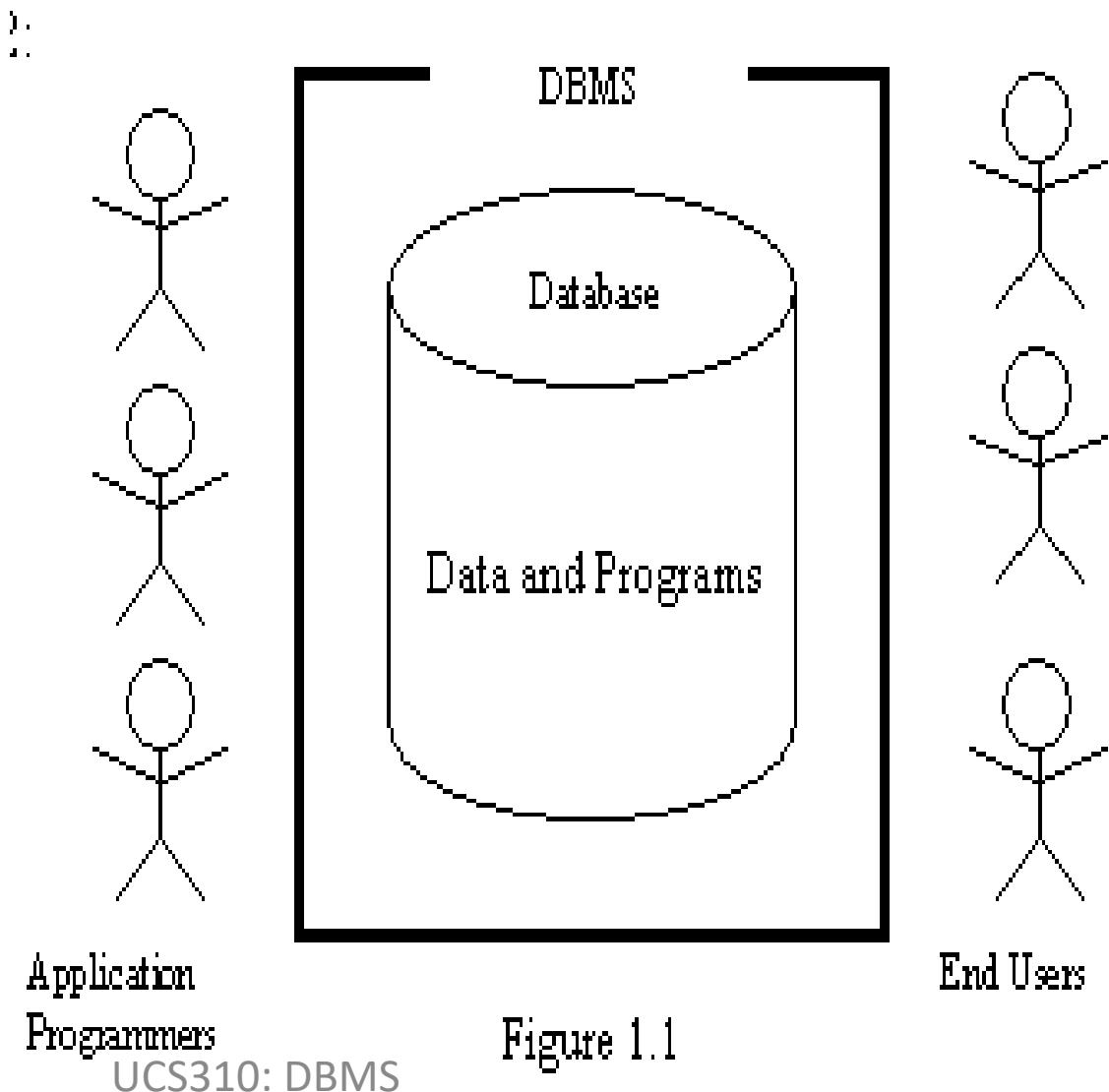
Data consistency can help ensure that data remains uniform across all systems, while Data Integrity can help ensure that data remains accurate, reliable and error-free throughout its life cycle. By focusing on these aspects, organizations can maintain high-quality data that supports informed decision-making.

Advantages of DBMS

- **Data can be shared**
- **Providing Backup and Recovery**
- **Standards can be enforced**
- **Restricting unauthorized access**
- **Solving enterprise requirement than individual requirement**

Components of the DBMS Environment

- Hardware
- Software
- Data
- Users
- Procedures



Disadvantages of DBMS

- **Complexity**
- **Size**
- **Performance**
- **Higher impact of a failure**
- **Cost of DBMS**
- **Additional Hardware costs**
- **Cost of Conversion**

Thank You

Architecture of Database Management System

Early Two Level Architecture

- Early proposal for a standard terminology and general architecture for database systems was produced in 1971 by the DBTG (Data Base Task Group)
- Appointed by the Conference on Data Systems and Languages (CODASYL, 1971).
- Recommend two level approach.

1. Schema
2. Subschema

Schema

4

- A map of the overall logical structure or description of the database .
- It gives the names of the entities, attributes and specifies the relationships among the entities.
- A schema is defined as an outline or a plan that describes the records and relationships existing at the particular level.
- The schema is sometimes called the **intension of the database**, while an instance is called an **extension (or state)** of the database.

Subschema

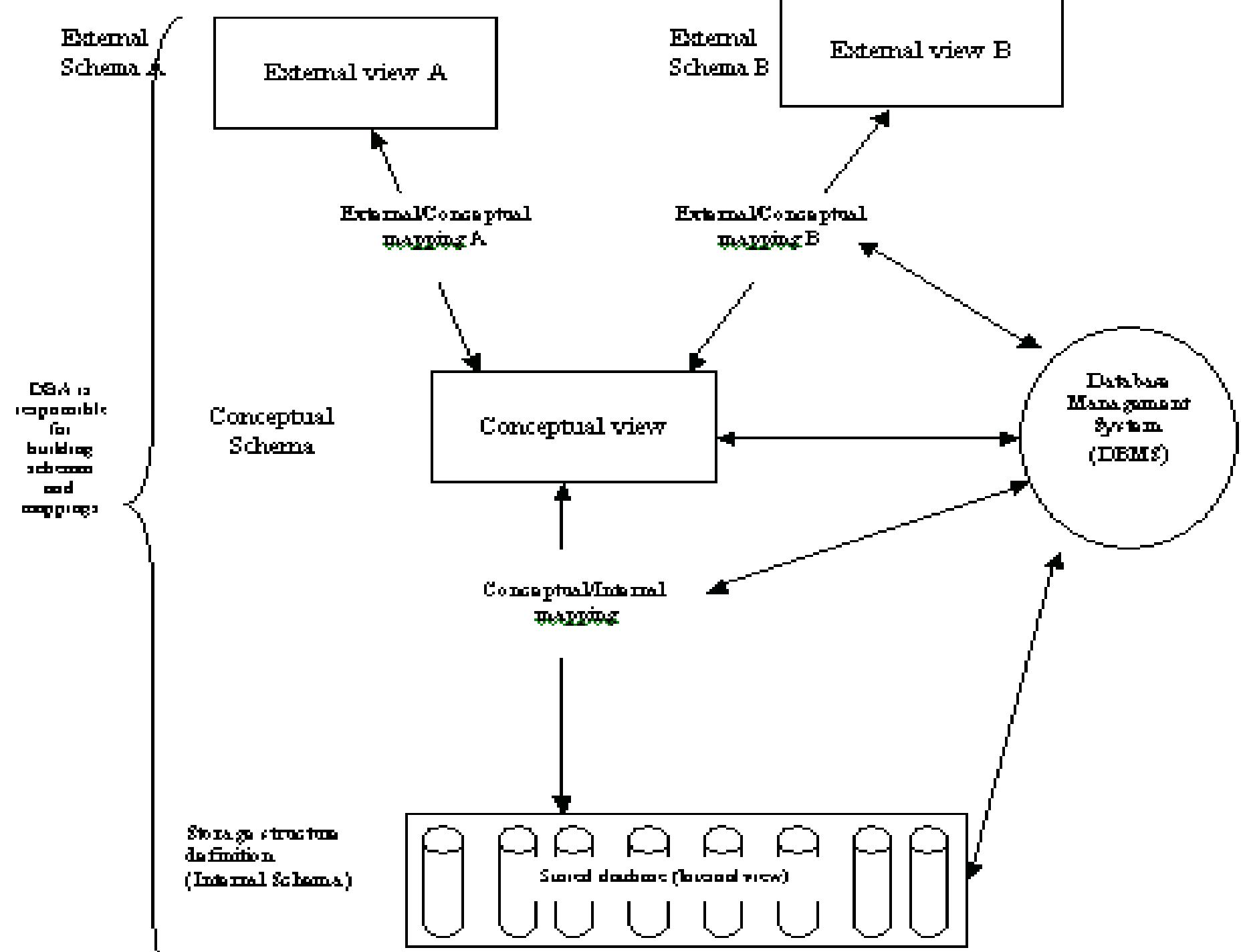
5

- A map of application programmer's view of the data he/she uses.
- It is a subset of the schema having the same properties that a schema has. It allows the user to view only that part of the database that is of interest to him. They are values of a table at a particular time.

Three Level Architecture

There are following three levels or layers of DBMS architecture:

- **External Level**
- **Conceptual Level**
- **Internal Level**



External Level or View level

- It is the users' view of the database. This level describes that part of the database that is relevant to each user.
- For example, one user may view dates in the form (day, month, year), while another may view dates as (year, month, day).

Conceptual Level or Logical level

- It is the community view of the database.
- This level describes **what data is stored in the database** and the relationships among the data.
- It represents:
 - All entities, their attributes, and the relationships among them;
 - The constraints on the data;
 - Security and integrity information.

Internal Level or Storage level or Physical Level

- It is the **physical representation** of the database on the computer.
- This level describes **how the data is stored** in the database.
- The internal level is concerned with such things as:
 - Storage space allocation for data and indexes;
 - Record descriptions for storage (with stored sizes for data items);
 - Record placement;
 - Data compression and data encryption techniques.

Roll Number
Name
Address
Book Number
Date of Issue
Date of Return

Roll Number
Name
Fee

Conceptual Level

Field Name	Data Type
Roll Number	Number(6)
Name	Character(20)
Address	Character(40)
Book Number	Number(6)
Date of Issue	Date
Date of Return	Date
Fee	Number(5 + 2)

Internal Level

Stored Item

Roll Number
Name
Address
Book Number
Date of Issue
Date of Return
Fee

Length

Type=Byte(6), Offset=0
Type=Byte(20), Offset=6
Type=Byte(40), Offset=26
Type=Byte(6), Offset=66
Type=Byte(8), Offset=72
Type=Byte(8), Offset=80
Type=Byte(8), Offset=88

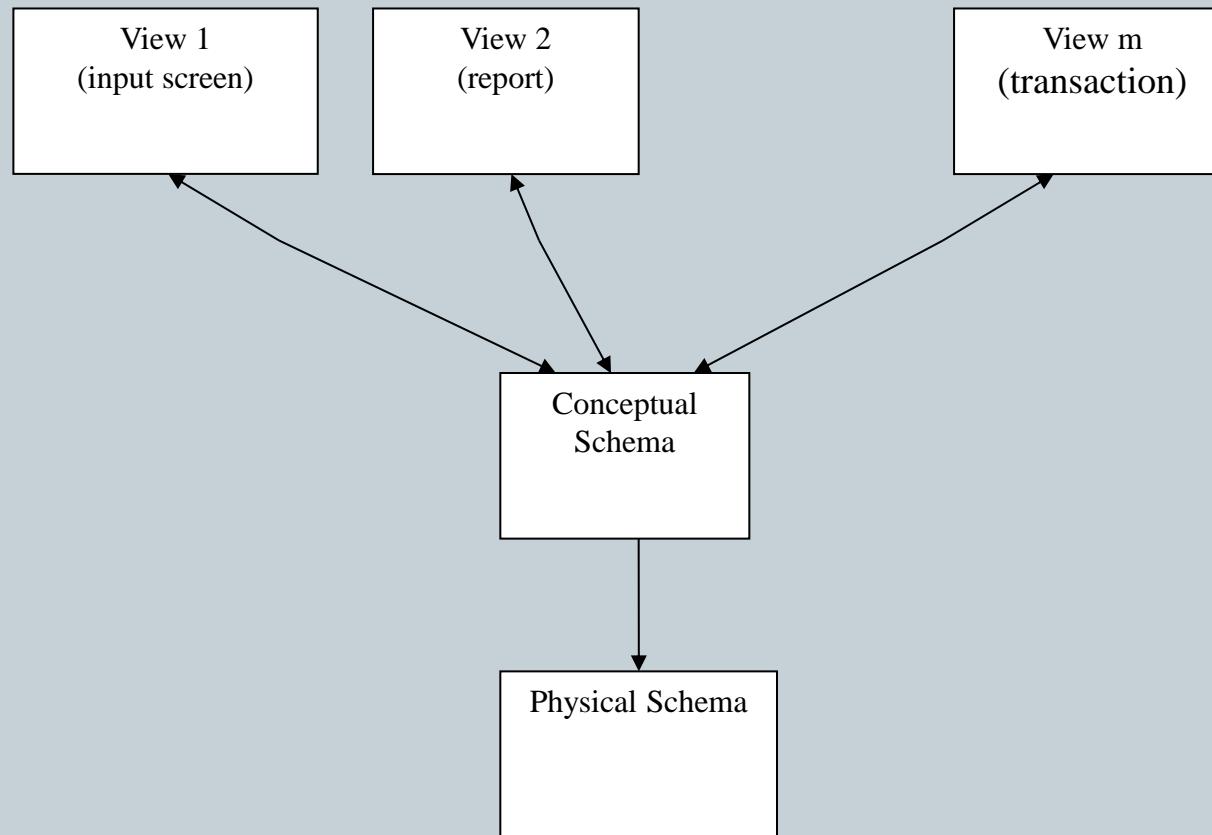
Mapping between Views

External/Conceptual Mapping:

A mapping between the external and conceptual views gives the correspondence among the records and the relationships of the external and conceptual views.

Conceptual/Internal Mapping:

Conceptual schema is related to the internal schema by the conceptual/internal mapping. This enables the DBMS to find the actual record or combination of records in physical storage that constitute a logical record in conceptual schema.



Data Independence-Achievement of Layered Architecture of DBMS

There are two kinds of data independence:

- .. Logical data independence
- .. Physical data independence

Logical data independence

- Logical data independence indicates that the conceptual schema can be changed without affecting the existing external schemas. Ability to modify the conceptual schema without causing application programs to be re-written.
- Modifications at conceptual level are necessary whenever the logical structure of the database is altered.

Physical data independence

- Physical data independence indicates that the physical storage structures or devices could be changed without affecting conceptual schema. Ability to modify the physical schema without causing application programs to be re-written.
- Modifications at physical level are occasionally necessary in order to improve performance.
- **Note:** logical data independence is more difficult to achieve than physical data independence, because application programs are heavily dependent on the logical structure of the data, they access.

Ex: Thapar Course Registration Database

- **Conceptual schema:**

student (st_id: string, name: string, birthdate: date, gender: character, pa_l1: character, pa_l2: character, pa_l3: character, la_l1: character, la_l2: character, la_l3: character, cugp: integer, cup: integer, cgpa: real)

course (courseno: string, cname: string, credits: integer, lect_h: integer, tut_h: integer, pract_h: integer)

instructor (instructor_id: string, name: string, design: string, group: string, specialization: string, chamber_no: string, tel_no: integer)

course_info (course_no: string, enrollment: integer, sections: integer)

st_course_reg (stid: string, course_no: string, section: integer, sem: integer, year: string, grade: string)

inst_course_reg (instructor_id: string, courseno: string, section: integer, sem: integer, year: string)

Ex: Thapar Course Registration Database

- External Schema (View):

course-allotment (instructor_id: string, group: string,
specialization: string,
course_no: string, section: integer)

course-reg (st_id: string, course_no: string,
section: integer)

grades (st_id: string, course_no: string,
grade: string)

Ex: Thapar Course Registration Database

19

- **Physical schema:**

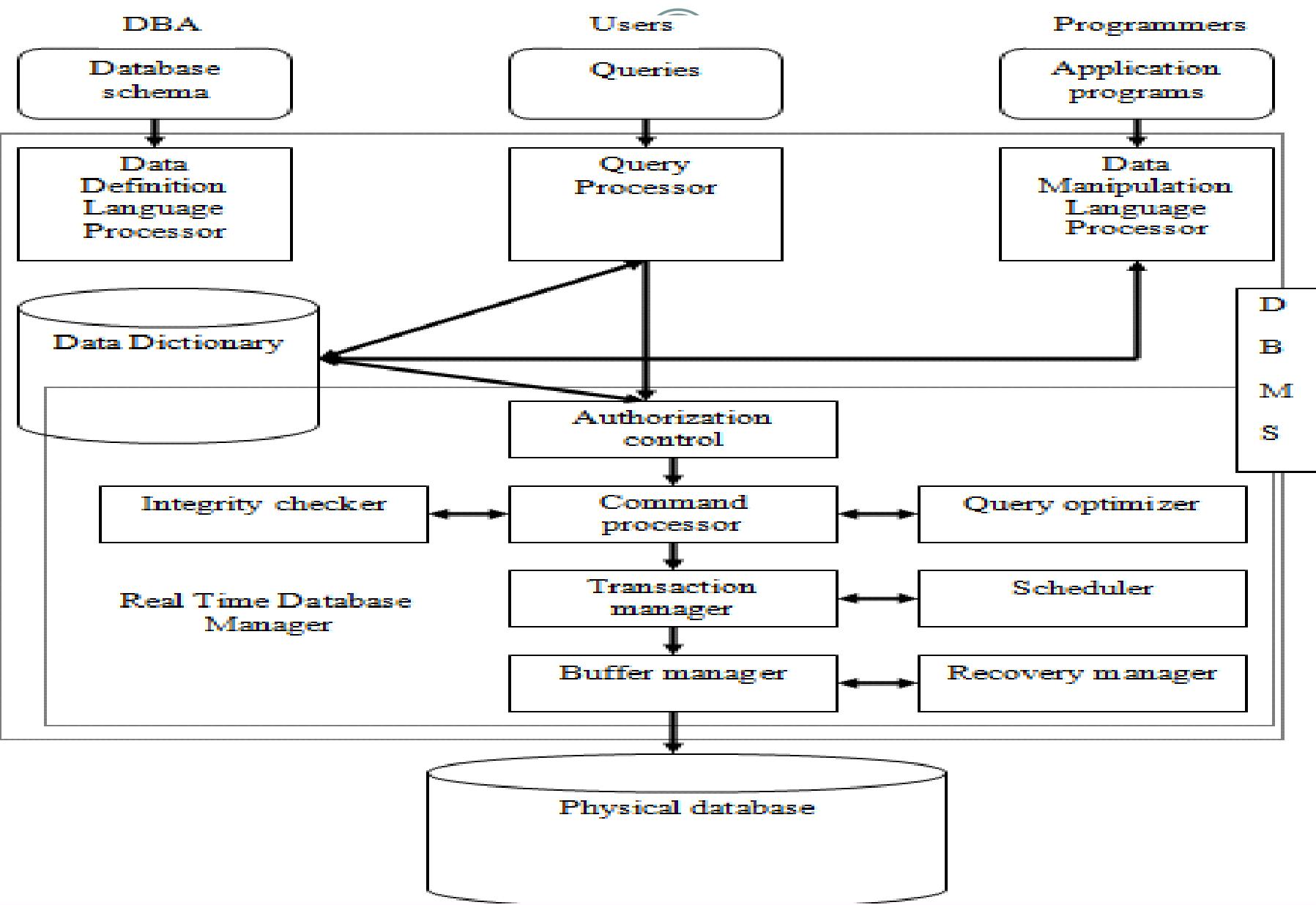
- Relations stored as unordered files.
- Indexes on column(s) of relations (tables).

Metadata or Data Dictionary

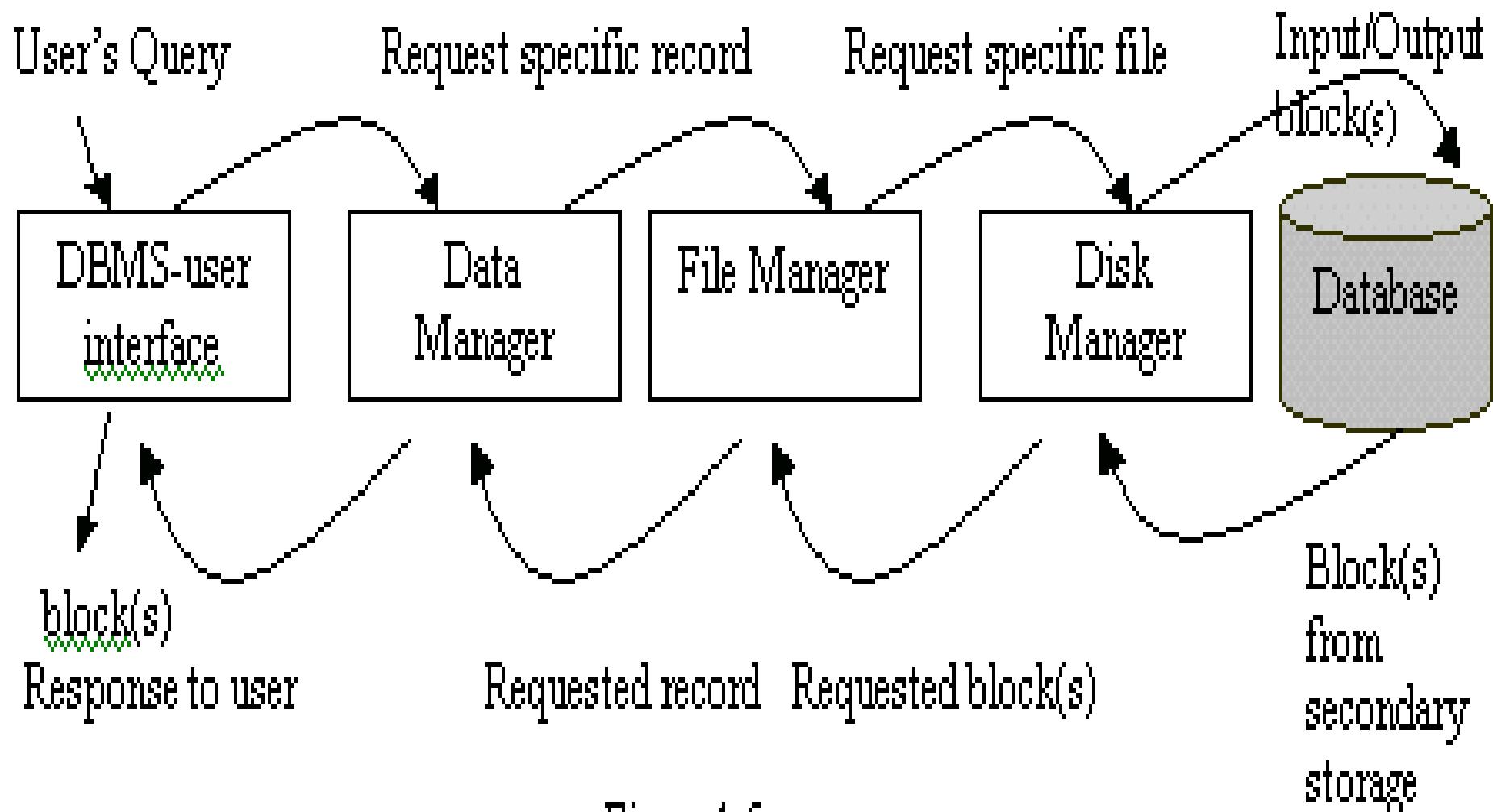
20

- A metadata (also called the data dictionary) is the data about the data.
- Data dictionary is a file that contains meta data. i.e., data about data.
- Data dictionary file is consulted before actual data is read or modified in the database.

Components of a DBMS



The Procedure for Database Access



END

Database Model

Database Models

- **Hierarchical model**
- **Network model**
- **Relational model**

Hierarchical model:

- Hierarchical data model is the oldest type of the data model. It was developed by IBM in 1968. It organizes data in the tree-like structure. Hierarchical model consists of the following :
- It contains nodes which are connected by branches.
- The topmost node is called the root node.
- If there are multiple nodes appear at the top level, then these can be called as root segments.
- Each node has exactly one parent.
- One parent may have many child.

Network model:

It is the advance version of the hierarchical data model. To organize data it uses directed graphs instead of the tree-structure. In this child can have more than one parent. It uses the concept of the two data structures i.e. Records and Sets.

Network model, Cont'd...

- Data in the network model are represented by collection of records, and relationships among data are represented by links, which can be viewed as pointers. The records in the database are organized as collections of arbitrary graphs.
- CODASYL DBTG proposal, new in DDL and DML (Honeywell, Univac, ICL)

Similarity and Difference between Hierarchical and Network Models

The hierarchical model is similar to network model in the sense that data and relationships among data are represented by records and links respectively. The hierarchical model differs from the network model in the sense that records are organized as a collection of trees rather than graphs.

- IMS from IBM or system 2000 from intel.

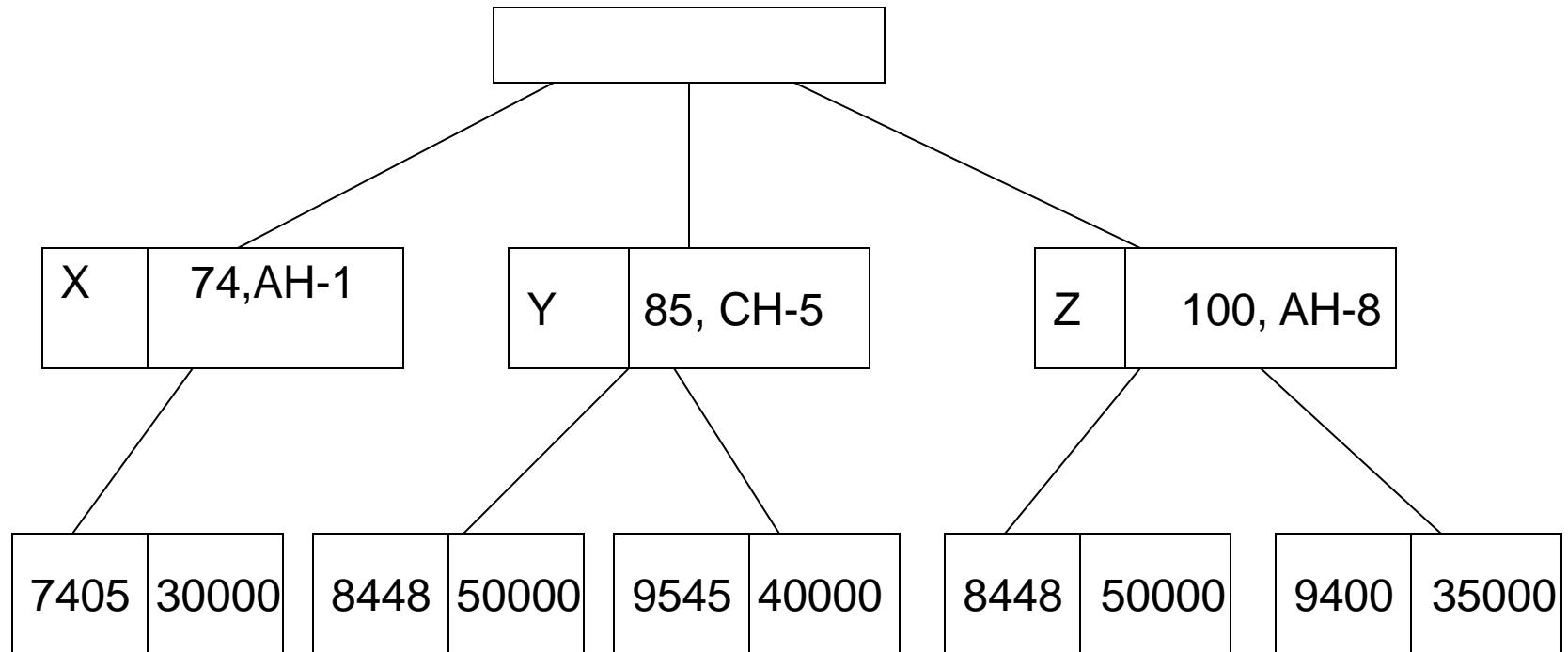
Relational model:

- The relational data model was developed by E.F. Codd in 1970. There are no physical links as they are in the hierarchical data model. Following are the properties of the relational data model :
- Data is represented in the form of table only.
- It deals only with the data not with the physical structure.
- It provides information regarding metadata.
- At the intersection of row and column there will be only one value for the tuple.
- It provides a way to handle the queries with ease.

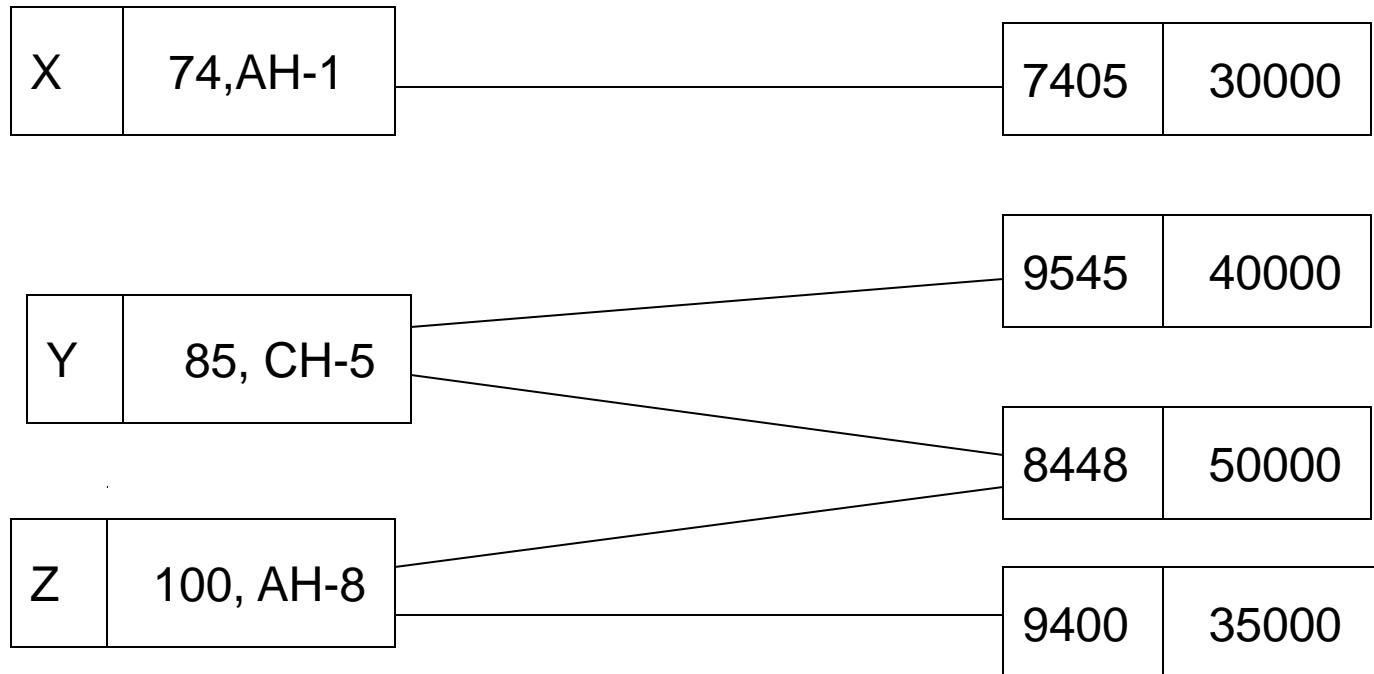
Database Models (Example)

- Let us take the example of Bank accounts, where customer name, address, account number and balance are considered.

Hierarchical model



Network model



Relational model

cust-name	cust-address	ac-no
X	74,AH-1	7405
Y	85, CH-5	9545
Y	85, CH-5	8448
Z	100, AH-8	9400
Z	100, AH-8	8448

ac-no	balance
7405	30000
8448	50000
9400	35000
9545	40000

S. No.	Hierarchical Data Model	Network Data Model	Relational Data Model
1.	In this model, to store data hierarchy method is used. It is the oldest method and not in use today.	It organizes records to one another through links or pointers.	It organizes records in the form of table and relationship between tables are set using common fields.
2.	To organize records, it uses tree structure.	It organizes records in the form of directed graphs.	It organizes records in the form of tables.
3.	It implements 1:1 and 1:n relations.	In addition to 1:1 and 1:n it also implements many to many relationships.	In addition to 1:1 and 1:n it also implements many to many relationships.
4.	Pointers are used to establish relationships among records physically.	A linked list is used to establish a relationship among records physically.	The logical representation is used with rows and columns to depict relationship among records.
5.	Insertion anomaly exists in this model i.e. child node cannot be inserted without the parent node.	There is no insertion anomaly.	There is no insertion anomaly.

6.	<p>Deletion anomaly exists in this model i.e. it is difficult to delete the parent node.</p>	<p>There is no deletion anomaly.</p>	<p>There is no deletion anomaly.</p>
7.	<p>Update leads to inconsistency problems because of the existence of multiple instances of a child record.</p>	<p>No such problem as only one instance of records exist.</p>	<p>Updating a record is easy and simple with the process of normalization, the redundant data gets removed.</p>
8.	<p>This model lacks data independence.</p>	<p>There is partial data independence in this model.</p>	<p>This model provides data independence.</p>
9.	<p>No such facility for querying database is supported.</p>	<p>No such facility for querying database is supported.</p>	<p>SQL-based declarative querying is supported.</p>
10.	<p>It is used to access the data which is complex and asymmetric.</p>	<p>It is used to access the data which is complex and symmetric.</p>	<p>It is used to access the data which is complex and symmetric.</p>

11.	Difficult to design a database because of its complexity.	Difficult to design a database and manipulate a database because of its complexity. Hence, it imposes a burden on the programmer.	It is easy to comprehend due to concealed physical level details from end-users.
12.	It is less flexible.	It is flexible as compared to the hierarchical model.	It is flexible as compared to the hierarchical model.
13.	&XML and XAML use this model.	VAX-DBMS, DMS-1100 of UNIVAC and SUPRADBMS's use this model.	It is mostly used in real world applications. Oracle, SQL.

End

The Relational Data Model

Relational Model Concepts

- The relational model represents the database as a collection of *relations*. Informally, each relation resembles a table of values or, to some extent, a "flat" file of records.
- When a relation is thought of as a **table** of values, each row in the table represents a collection of related data values.

Relational Model Concepts

- In the formal relational model terminology, a row is called a *tuple*, a column header is called an *attribute*, and the table is called a *relation*. The data type describing the types of values that can appear in each column is called a *domain*.

Domains, Attributes, Tuples, and Relations

- A **domain** D is a set of atomic values. By **atomic** we mean that each value in the domain is indivisible as far as the relational model is concerned.
- A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn.

Domains, Attributes, Tuples, and Relations

- Some examples of domains:
 - USA_phone_numbers: The set of 10-digit phone numbers valid in the United States.
 - Local_phone_numbers: The set of 7-digit phone numbers valid within a particular area code in the United States.
 - Social_security_numbers: The set of valid 9-digit social security numbers.
 - Names: The set of names of persons.

Domains, Attributes, Tuples, and Relations

- Some examples of domains (continued):
 - Grade_point_averages: Possible values of computed grade point averages; each must be a real (floating point) number between 0 and 4.
 - Employee_ages: Possible ages of employees of a company; each must be a value between 15 and 80 years old.
 - Academic_department_names: The set of academic department names, such as Computer Science, Economics, and Physics, in a university.
 - Academic_department_codes: The set of academic department codes, such as CS, ECON, and PHYS, in a university.

Domains, Attributes, Tuples, and Relations

- The preceding examples are called *logical* definitions of domains. A **data type** or **format** is also specified for each domain.
 - For example, the data type for the domain USA_phone_numbers can be declared as a character string of the form (ddd)ddd-dddd, where each d is a numeric (decimal) digit and the first three digits form a valid telephone area code.
 - The data type for Employee_ages is an integer number between 15 and 80.
 - For Academic_department_names, the data type is the set of all character strings that represent valid department names.

Domains, Attributes, Tuples, and Relations

- A domain is thus given a name, data type, and format.
 - Additional information for interpreting the values of a domain can also be given; for example, a numeric domain such as Person_weights should have the units of measurement—pounds or kilograms.

Domains, Attributes, Tuples, and Relations

- A **relation schema** R , denoted by $R(A_1, A_2, \dots, A_n)$, is made up of a relation name R and a list of attributes A_1, A_2, \dots, A_n . Each **attribute** A_i is the name of a role played by some domain D in the relation schema R . D is called the **domain** of A_i and is denoted by $\text{dom}(A_i)$. A relation schema is used to *describe* a relation; R is called the **name** of this relation. The **degree** of a relation is the number of attributes n of its relation schema.

Domains, Attributes, Tuples, and Relations

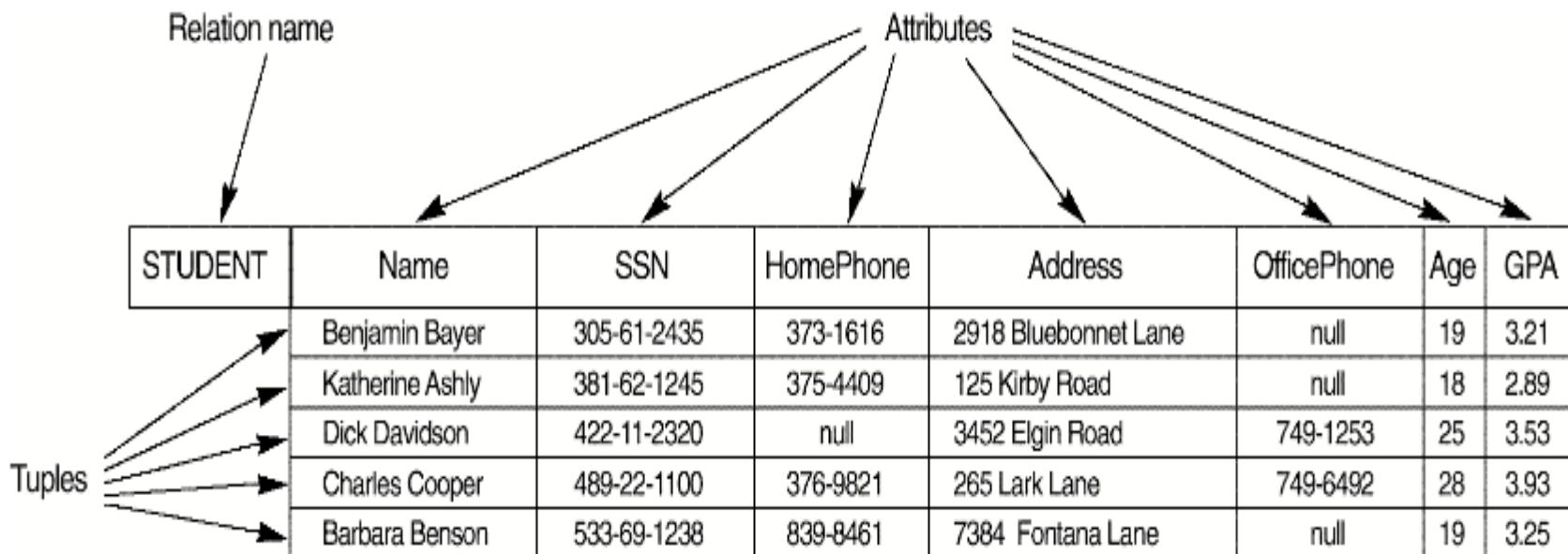
- An example of a relation schema for a relation of degree 7, which describes university students, is the following:
 - STUDENT(Name, SSN, HomePhone, Address, OfficePhone, Age, GPA)
 - For this relation schema, STUDENT is the name of the relation, which has seven attributes. We can specify the following previously defined domains for some of the attributes of the STUDENT relation: $\text{dom}(\text{Name}) = \text{Names}$; $\text{dom}(\text{SSN}) = \text{Social_security_numbers}$; $\text{dom}(\text{HomePhone}) = \text{Local_phone_numbers}$, $\text{dom}(\text{OfficePhone}) = \text{Local_phone_numbers}$, and $\text{dom}(\text{GPA}) = \text{Grade_point_averages}$.

Domains, Attributes, Tuples, and Relations

- A **relation** (or **relation state**) r of the relation schema $R(A_1, A_2, \dots, A_n)$, also denoted by $r(R)$, is a set of n -tuples $r = \{t_1, t_2, \dots, t_m\}$. Each **n -tuple** t is an ordered list of n values $t = \langle v_1, v_2, \dots, v_n \rangle$, where each value v_i , $1 \leq i \leq n$, is an element of $\text{dom}(A_i)$ or is a special **null** value.
- The i^{th} value in tuple t , which corresponds to the attribute A_i , is referred to as $t[A_i]$. The terms relation **intension** for the schema R and relation **extension** for a relation state $r(R)$ are also commonly used.

Domains, Attributes, Tuples, and Relations

- Example



Domains, Attributes, Tuples, and Relations

- Out of all the possible combinations, a relation state at a given time—the **current relation state**—reflects only the valid tuples that represent a particular state of the real world.
- In general, as the state of the real world changes, so does the relation, by being transformed into another relation state.
- However, the schema R is relatively static and does *not* change except very infrequently—for example, as a result of adding an attribute to represent new information that was not originally stored in the relation.

Domains, Attributes, Tuples, and Relations

- It is possible for several attributes to *have the same domain*. The attributes indicate different **roles**, or interpretations, for the domain. For example, in the STUDENT relation, the same domain Local_phone_numbers plays the role of HomePhone, referring to the "home phone of a student," and the role of OfficePhone, referring to the "office phone of the student."

Characteristics of Relations

- The earlier definition of relations implies certain characteristics that make a relation different from a file or a table.
- These characteristics are

Ordering of Tuples in a Relation

- A *relation* is defined as a set of tuples.
 - Mathematically, elements of a set have *no order* among them; hence tuples in a relation do not have any particular order.
 - However, in a file, records are physically stored on disk so there always is an order among the records. This ordering indicates first, second, i^{th} , and last records in the file. Similarly, when we display a relation as a table, the rows are displayed in a certain order.

Ordering of Tuples in a Relation

- Tuple ordering is not part of a relation definition, because a relation attempts to represent facts at a logical or abstract level.

Ordering of Tuples in a Relation

- No logical difference:

STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21

STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
Tuples	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

Ordering of Tuples in a Relation

- According to the preceding definition of a relation, an n-tuple is an *ordered list* of n values, so the ordering of values in a tuple—and hence of attributes in a relation schema definition—is important. However, at a logical level, the order of attributes and their values are *not* really important as long as the correspondence between attributes and values is maintained.

Ordering of Tuples in a Relation

- An **alternative definition** of a relation can be given, making the ordering of values in a tuple *unnecessary*. In this definition, a relation schema $R = \{A_1, A_2, \dots, A_n\}$ is a *set* of attributes, and a relation $r(R)$ is a finite set of **mappings** $r = \{t_1, t_2, \dots, t_m\}$, where each tuple t_i is a mapping from R to D , and D is the union of the attribute domains; that is, $D = \text{dom}(A_1) \cup \text{dom}(A_2) \cup \dots \cup \text{dom}(A_n)$. In this definition, $t[A_i]$ must be in $\text{dom}(A_i)$ for $1 \leq i \leq n$ for each mapping t in r . Each mapping t_i is called a tuple.

Ordering of Tuples in a Relation

- Thus, there is no logical difference between the following:

$t = < (\text{Name}, \text{Dick Davidson}), (\text{SSN}, 422-11-2320), (\text{HomePhone}, \text{null}), (\text{Address}, 3452 \text{ Elgin Road}), (\text{OfficePhone}, 749-1253), (\text{Age}, 25), (\text{GPA}, 3.53) >$

$t = < (\text{Address}, 3452 \text{ Elgin Road}), (\text{Name}, \text{Dick Davidson}), (\text{SSN}, 422-11-2320), (\text{Age}, 25), (\text{OfficePhone}, 749-1253), (\text{GPA}, 3.53), (\text{HomePhone}, \text{null}) >$

Values in the Tuples

- Each value in a tuple is an **atomic** value; that is, it is not divisible into components within the framework of the basic relational model.
 - Thus, composite and multivalued attributes are not allowed. Much of the theory behind the relational model was developed with this assumption in mind, which is called the **first normal form** assumption.
 - Multivalued attributes must be represented by separate relations, and composite attributes are represented only by their simple component attributes.

Interpretation of a Relation

- The relation schema can be interpreted as a declaration or a type of **assertion**.
 - For example, the schema of the STUDENT relation asserts that, in general, a student entity has a Name, SSN, HomePhone, Address, OfficePhone, Age, and GPA.
- Each tuple in the relation can then be interpreted as a fact or a particular instance of the assertion.
 - For example, the first tuple asserts the fact that there is a STUDENT whose name is Benjamin Bayer, SSN is 305-61-2435, Age is 19, and so on.

Interpretation of a Relation

- Notice that some relations may represent facts about *entities*, whereas other relations may represent facts about *relationships*.
 - For example, a relation schema **MAJORS** (StudentSSN, DepartmentCode) asserts that students major in academic departments; a tuple in this relation relates a student to his or her major department.
- Thus, the relational model represents facts about both entities and relationships *uniformly* as relations.

Relational Model Notation

- The text uses the following notation:
 - A relation schema R of degree n is denoted by $R(A_1, A_2, \dots, A_n)$.
 - An n -tuple t in a relation $r(R)$ is denoted by $t = \langle v_1, v_2, \dots, v_n \rangle$, where v_i is the value corresponding to attribute A_i . The following notation refers to **component values** of tuples:
 - Both $t[A_i]$ and $t.A_i$ refer to the value v_i in t for attribute A_i .
 - Both $t[A_u, A_w, \dots, A_z]$ and $t.(A_u, A_w, \dots, A_z)$, where A_u, A_w, \dots, A_z is a list of attributes from R , refer to the subtuple of values $\langle v_u, v_w, \dots, v_z \rangle$ from t corresponding to the attributes specified in the list.

Relational Model Notation

- The text uses the following notation (cont.):
 - The letters Q, R, S denote relation names.
 - The letters q, r, s denote relation states.
 - The letters t, u, v denote tuples.
 - In general, the name of a relation schema such as **STUDENT** *also indicates* the current set of tuples in that relation—the *current relation state*—whereas **STUDENT**(Name, SSN, . . .) refers *only* to the relation schema.
 - An attribute A can be qualified with the relation name R to which it belongs by using the *dot notation* R.A—for example, **STUDENT**.Name or **STUDENT**.Age. This is because the same name may be used for two attributes in different relations. However, all attribute names *in a particular relation* must be distinct.

Relational Model Notation

- As an example, consider the tuple $t = \langle \text{'Barbara Benson'}, \text{'533-69-1238'}, \text{'839-8461'}, \text{'7384 Fontana Lane'}, \text{null}, 19, 3.25 \rangle$ from the STUDENT relation; we have $t[\text{Name}] = \langle \text{'Barbara Benson'} \rangle$, and $t[\text{SSN, GPA, Age}] = \langle \text{'533-69-1238'}, 3.25, 19 \rangle$.

Thank You

Keys

Keys?

- A DBMS key is an attribute or set of an attribute which helps you to identify a row(tuple) in a relation(table).
- They 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?

- Keys 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.

Keys in Database Management System

- **Super Key**
- **Primary Key**
- **Candidate Key**
- **Alternate Key**
- **Foreign Key**
- **Composite Key**
- **Surrogate Key/ Artificial Key**

Super key?

- A superkey is a group of single or multiple keys which identifies rows in a table.
- A Super key may have additional attributes that are not needed for unique identification.

In this example, EmpSSN, EmpNum and Empname are superkeys.

Example:

EmpSSN	EmpNum	Empname
9812345098	AB05	Shown
9876512345	AB06	Roslyn
199937890	AB07	James

Primary Key?

- A column or group of columns in a table which helps us to uniquely identifies every row in that table is called a primary key.
- This can't be a duplicate. The same value can't appear more than once in the table.
- Rules for defining Primary key:
 - Two rows can't have the same primary key value.
 - It must for every row to have a primary key value.
 - The primary key field cannot be null.
 - The value in a primary key column can carefully be modified or updated, considering that if any foreign key refers to that primary key.

Example:

In the following example, StudID is a Primary Key.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com (mailto:abc@gmail.com)
2	12	Nick	Wright	xyz@gmail.com (mailto:xyz@gmail.com)
3	13	Dana	Natan	mno@yahoo.com (mailto:mno@yahoo.com)

Alternate key?

- All the keys which are not primary key are called an alternate key.
- It is a candidate key which is currently not the primary key.
- However, A table may have single or multiple choices for the primary key.

Example: In this table, **StudID**, **Roll No**, **Email** are qualified to become a primary key. But since **StudID** is the primary key, **Roll No**, **Email** becomes the alternative key.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com (mailto:abc@gmail.com)
2	12	Nick	Wright	xyz@gmail.com (mailto:xyz@gmail.com)
3	13	Dana	Natan	mno@yahoo.com (mailto:mno@yahoo.com)

Candidate Key?

- A super key with no repeated attribute is called candidate key.
 - The Primary key should be selected from the candidate keys. Every table must have at least a single candidate key.
-
- Properties of Candidate key:
 - It must contain unique values
 - Candidate key may have multiple attributes
 - Must not contain null values
 - It should contain minimum fields to ensure uniqueness
 - Uniquely identify each record in a table

Candidate Key?

- Example: In the given table **Stud ID**, **Roll No**, and **email** are candidate keys which help us to uniquely identify the student record in the table.

StudID	Roll No	First Name	Last Name	Email
1	11	Tom	Price	abc@gmail.com (mailto:abc@gmail.com)
2	12	Nick	Wright	xyz@gmail.com (mailto:xyz@gmail.com)
3	13	Dana	Natan	mno@yahoo.com (mailto:mno@yahoo.com)



Foreign key?

- A foreign key is a column which is added to create a relationship with another table.
- Foreign keys help us to maintain data integrity and also allows navigation between two different instances of an entity.
- Every relationship in the model needs to be supported by a foreign key.

Example:

DeptCode	DeptName
001	Science
002	English
005	Computer

Teacher ID	Fname	Lname
B002	David	Warner
B017	Sara	Joseph
B009	Mike	Brunton

In this example, we have two table, **teacher** and **department** in a school. However, there is no way to see which search work in which department.

In this table, adding the foreign key in Deptcode to the Teacher, we can create a relationship between the two tables.

teacher and department tables

Example:

DeptCode	DeptName	
001	Science	
002	English	
005	Computer	
Teacher ID	Fname	Lname
B002	David	Warner
B017	Sara	Joseph
B009	Mike	Brunton

Teacher with DeptCode

Teacher ID	DeptCode	Fname	Lname
B002	002	David	Warner
B017	002	Sara	Joseph
B009	001	Mike	Brunton

This concept is also known as Referential Integrity.

Composite key?

- A key which has multiple attributes to uniquely identify rows in a table is called a composite key.
- It is type of Compound key has many fields which allow you to uniquely recognize a specific record.
- It is possible that each column may be not unique by itself within the database.
- However, when combined with the other column or columns, the combination of composite keys become unique.

Composite or Compound key example

Example:

OrderNo	ProductID	Product Name	Quantity
B005	JAP102459	Mouse	5
B005	DKT321573	USB	10
B005	OMG446789	LCD Monitor	20
B004	DKT321573	USB	15
B002	OMG446789	Laser Printer	3

In this example, OrderNo and ProductID can't be a primary key as it does not uniquely identify a record. However, a compound key of Order ID and Product ID could be used as it uniquely identified each record.

Surrogate Key/Artificial key ?

- An artificial key which aims to uniquely identify each record is called a surrogate key.
- These kind of key are unique because they are created when you don't have any natural primary key.
- They do not lend any meaning to the data in the table.
- Surrogate key is usually an integer.

Student Table

Student_Name	Class
Amit	CSE
Amit	CSE
Rohit	ENC

Student Table with Sr_No as Surrogate Key/Artificial key ?

Sr_No	Student_Name	Class
1	Amit	CSE
2	Amit	CSE
3	Rohit	ENC

Difference Between Primary key & Foreign key

Primary Key	Foreign Key
Helps you to uniquely identify a record in the table.	It is a field in the table that is the primary key of another table.
Primary Key never accept null values.	A foreign key may accept multiple null values.
Primary key is a clustered index and data in the DBMS table are physically organized in the sequence of the clustered index.	A foreign key cannot automatically create an index, clustered or non-clustered. However, you can manually create an index on the foreign key.
You can have the single Primary key in a table.	You can have multiple foreign keys in a table.

Thank You

Relational Integrity constraints or RULES

Concept of Null or Unknown Value

- Null represent a value for an attribute that is currently unknown or it is not applicable for this tuple.
- Null is not same as 0 or zero value or a text string.
- It represent absence of value
- At a particular time the following table might not have the value of age (of Rajesh) and Job (of Raja)

Name	Age	Job
Rajesh	-	Clerk
Raja	23	-
Amit	43	Sales

Difference between Null and Not Applicable

- Blank against Amit might indicate that pension is not applicable to this employee as per firm policy.

Name	Age	Job	Pension_Date
Rajesh	-	Clerk	-
Raja	23	-	01/01/1999
Amit	43	Sales	-

Best way to represent it as :

Name	Age	Job	Pension_Date
Rajesh	null	Clerk	null
Raja	23	null	01/01/1999
Amit	43	Sales	Not Applicable

Issues in basic STUDENT table

Basic table suffers from following anomalies:

- ▶ There is no check on storing duplicate records
- ▶ There is no mandatory column
- ▶ There is no check on validity of data

To ensure all these features, there is a need to apply constraints during creation of table.

Relational Integrity constraints/RULES

Relational Integrity constraints is referred to conditions which must be present for a valid relation. These integrity constraints are derived from the rules that the database represents.

There are many types of integrity constraints. Constraints on the Relational database management system is mostly divided into three main categories are:

- 1. Domain constraints**
- 2. Entity integrity or Key/primary key constraints**
- 3. Referential integrity constraints**

Relational Integrity constraints or RULES

Data Integrity

The following categories of data integrity exist with each RDBMS:

- **Entity Integrity:** There are no duplicate rows in a table.
- **Domain Integrity:** Enforces valid entries for a given column by restricting the type, the format, or the range of values.
- **Referential integrity:** Rows cannot be deleted, which are used by other records.
- **User-Defined Integrity:** Enforces some specific business rules that do not fall into entity, domain or referential integrity.

Levels of Constraints

Column level constraints

Constraints which are applied on single column of table are known as column level constraints.

For example:

Roll_Number is Unique.

Table level constraints

Constraints which are applied on more than one column of a table are known as table level constraints.

For example:

Roll_Number and Class combination is unique in primary School database
[In primary school Roll number of students starts with one in each class]

Types of Constraints

- ▶ Not Null
- ▶ Unique
- ▶ Primary key
- ▶ Check
- ▶ Foreign key
- ▶ Default

NOT NULL Constraint

- By default, a column can hold NULL values.
- If you do not want a column to have a NULL value, then you need to define such a constraint on this column specifying that NULL is now not allowed for that column.
- A NULL is not the same as no data, rather, it represents unknown data.

NOT NULL Constraint

- ▶ Specifies if the column must contain value or might not contain any.
- ▶ By default all columns in a table allow nulls.
- ▶ NOT NULL specifies that all rows in the table to have value for specified column.
- ▶ Syntax:

Columnname datatype (size) [constraint constraintname]
NOT NULL

Example [Not Null: Column level constraint]

```
CREATE TABLE student
(
    Roll_Number Number(4) Not Null,
    Name Char(15), Class Char(10), Marks Number(4),
    DOB Date
);
```

Example

```
CREATE TABLE CUSTOMERS(  
    ID INT                      NOT NULL,  
    NAME VARCHAR (20)            NOT NULL,  
    AGE INT                     NOT NULL,  
    ADDRESS CHAR (25) ,  
    SALARY DECIMAL (18, 2),  
    PRIMARY KEY (ID)  
);
```

If CUSTOMERS table has already been created, then to add a NOT NULL constraint to the SALARY column in

```
ALTER TABLE CUSTOMERS MODIFY SALARY DECIMAL (18, 2) NOT NULL;
```

UNIQUE Constraint

- The **UNIQUE Constraint** prevents two records from having identical values in a column.
- E.g. In the CUSTOMERS table, for example, you might want to prevent two or more people from having an identical age.
 - ▶ Enforce uniqueness of the column or group of columns.
 - ▶ Syntax:
Columnname datatype (size) [constraint constraintname]
UNIQUE

Example [Unique: Column level constraint]

```
CREATE TABLE student  
(  
    Roll_Number Number(4) Unique,  
    Name Char(15), Class Char(10), Marks Number(4),  
    DOB Date  
);
```

OR

```
CREATE TABLE student  
(  
    Roll_Number Number(4) Constraint RNO_Unique Unique,  
    Name Char(15), Class Char(10), Marks Number(4),  
    DOB Date  
);
```

Example

For example, the following SQL query creates a new table called CUSTOMERS and adds five columns.

Here, the AGE column is set to UNIQUE, so that you cannot have two records with the same age.

```
CREATE TABLE CUSTOMERS(  
    ID INT      NOT NULL,  
    NAME VARCHAR (20) NOT NULL,  
    AGE INT      NOT NULL UNIQUE,  
    ADDRESS CHAR (25) ,  
    SALARY DECIMAL (18, 2),  
    PRIMARY KEY (ID)  
);
```

Example [Unique:Table level constraint]

```
CREATE TABLE primary_student  
(  
    Roll_Number Number(4), Name Char(15), Class Char(10),  
    Marks Number(4), DOB Date,  
    Unique(Roll_Number, Class)  
);
```

OR

```
CREATE TABLE primary_student  
(  
    Roll_Number Number(4), Name Char(15), Class Char(10), Marks Number(4),  
    Date, DOB,  
    Constraint RNO_Class_Uncique Unique(Roll_Number, Class)  
);
```

Adding Constraint

```
ALTER TABLE CUSTOMERS  
ADD CONSTRAINT myUniqueConstraint UNIQUE(AGE, SALARY);
```

DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL query.

```
ALTER TABLE CUSTOMERS  
DROP CONSTRAINT myUniqueConstraint;
```

Entity Integrity OR Key/Primary Key constraints

- An attribute that can uniquely identify a tuple in a relation is called the key of the table.
- The value of the attribute for different tuples in the relation has to be unique.
- Entity integrity constraints are rules for primary keys:
 - The primary key cannot have a null value.
 - If the primary key is a composite key, none of the fields in the key can contain a null value.

Entity Integrity OR Key/Primary Key constraints

Example:

- In the given table, CustomerID is a key attribute of Customer Table.
- It is most likely to have a single key for one customer, CustomerID =1 is only for the CustomerName = " Google".

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Primary Key Constraint

- ▶ It is a combination of both the unique key constraint as well as the not null constraint.
- ▶ A primary key is used to identify each row of the table uniquely.
- ▶ A primary key may be either consists of a single field (Simple key) or group of fields (Composite Primary Key).
- ▶ Oracle enforces all PRIMARY KEY constraints using indexes.
- ▶ Syntax:

Columnname datatype (size) [constraint constraintname] PRIMARY KEY

Example [Primary Key: Column level Constraint]

```
CREATE TABLE student
(
    Roll_Number Number(4) Primary Key,
    Name Char(15), Class Char(10), Marks Number(4),
    DOB Date
);
```

OR

```
CREATE TABLE student
(
    Roll_Number Number(4) Constraint RNO_PK Primary Key,
    Name Char(15), Class Char(10), Marks Number(4),
    DOB Date
);
```

Example [Primary Key: Table level Constraint]

```
CREATE TABLE primary_student  
(  
    Roll_Number Number(4), Name Char(15), Class Char(10),  
    Marks Number(4), DOB Date,  
    Primary Key(Roll_Number, Class)  
);
```

OR

```
CREATE TABLE primary_student  
(  
    Roll_Number Number(4), Name Char(15), Class Char(10), Marks Number(4), DOB  
    Date,  
    Constraint RNO_Class_PK Primary Key(Roll_Number, Class)  
);
```

Check Constraint

- ▶ Check constraints allow Oracle to verify the validity of data being entered on a table against a set of constants. These constants act as valid values.
- ▶ The Check constraint consists of the keyword CHECK followed by parenthesized conditions.
- ▶ Check constraints must be specified as a logical expression that evaluates either to TRUE or FALSE.
- ▶ Syntax:
- ▶ Columnname datatype (size) [constraint constraintname] CHECK (logical expression)

Example [Check: Column level constraint]

```
CREATE TABLE student
```

```
(  
    Roll_Number Number(4) , Name Char(15),  
    Class Char(10) Check (class in ('BE','ME','MCA')),  
    Marks Number(4) Check (marks>=0), DOB Date  
)
```

OR

```
CREATE TABLE student
```

```
(  
    Roll_Number Number(4) , Name Char(15),  
    Class Char(10) Constraint class_check Check (class in ('BE','ME','MCA')),  
    Marks Number(4) Check (marks>=0), DOB Date  
)
```

Example [Check: Table level constraint]

```
CREATE TABLE library
(
    Roll_Number Number(4) Primary Key,
    Book_Number Number(10) NOT NULL,
    DOI Date, DOR Date,
    Check (DOR>DOI);
```

OR

```
CREATE TABLE library
(
    Roll_Number Number(4) Primary Key,
    Book_Number Number(10) NOT NULL,
    DOI Date, DOR Date,
    Constraint Date_Check Check (DOR>DOI);
```

Example

For example, the following program creates a new table called CUSTOMERS and adds five columns. Here, we add a CHECK with AGE column, so that you cannot have any CUSTOMER who is below 18 years.

```
CREATE TABLE CUSTOMERS(  
    ID INT      NOT NULL,  
    NAME VARCHAR (20) NOT NULL,  
    AGE INT      NOT NULL CHECK (AGE >= 18),  
    ADDRESS CHAR (25) ,  
    SALARY DECIMAL (18, 2),  
    PRIMARY KEY (ID)  
);
```

If the CUSTOMERS table has already been created, then to add a CHECK constraint to AGE column, you would write a statement like the one given below.

```
ALTER TABLE CUSTOMERS  
    MODIFY AGE INT NOT NULL CHECK (AGE >= 18 );
```

Or

```
ALTER TABLE CUSTOMERS  
    ADD CONSTRAINT myCheckConstraint CHECK(AGE >= 18);
```

DROP a CHECK Constraint

```
ALTER TABLE CUSTOMERS  
    DROP CONSTRAINT myCheckConstraint;
```

Limitations of Check Constraint

Rno	Name	Class	Marks	DOB
1	Ram	BE	60	12-DEC-1980
2	Rajesh	MCA	70	13-JAN-1989
3	Surinder	ME	78	10-JUN-1980

Check class in ('BE','ME','MCA')

Limitations of Check Constraint

Rno	Name	Class	Marks	DOB
1	Ram	BE	60	12-DEC-1980
2	Rajesh	MCA	70	13-JAN-1989
3	Surinder	ME	78	10-JUN-1980
4	Rahat	MBA	76	15-MAY-1980

Check class in ('BE', 'ME', 'MCA')
It will not be allowed...

Limitations of Check Constraint

- ▶ It will not be suitable if check values need to be changed in future.
- ▶ Class can be added or removed in future, for check constraint is not desirable.
- ▶ Apply check constraint only if its value is not going to change in future.
- ▶ For example,
- ▶ Marks number(3) check (marks>=0)
- ▶ Gender
Gender char(10) check (gender in ('male', 'female', 'transgender'));

Referential integrity constraints

- Referential integrity constraints is base on the **concept of Foreign Keys**.
- A foreign key is an important attribute of a relation **which should be referred to in other relationships.**
- Referential integrity constraint **state happens where relation refers to a key attribute of a different or same relation.** However, that **key element must exist in the table.**

Referential Integrity

- Referential Integrity can help to avoid data input errors, voids data inconsistency and quality problems.
- In a relational database, **referential integrity means that a foreign key value cannot be entered in one table unless it matches an existing primary key in another table.**
- You cannot change a primary key that has matching child table records
 - A child table that has a foreign key for a different record
- Referential integrity also can prevent the deletion of a record has a primary key that matches foreign keys in another table.

The diagram illustrates two relational tables: Customer and Billing. The Customer table has columns CustomerID, CustomerName, and Status. It contains three tuples: (1, Google, Active), (2, Amazon, Active), and (3, Apple, Inactive). The Billing table has columns InvoiceNo, CustomerID, and Amount. It contains three tuples: (1, 1, \$100), (2, 1, \$200), and (3, 2, \$150). A dashed arrow points from the CustomerID column in the Customer table to the CustomerID column in the Billing table, indicating a relationship between the two tables.

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

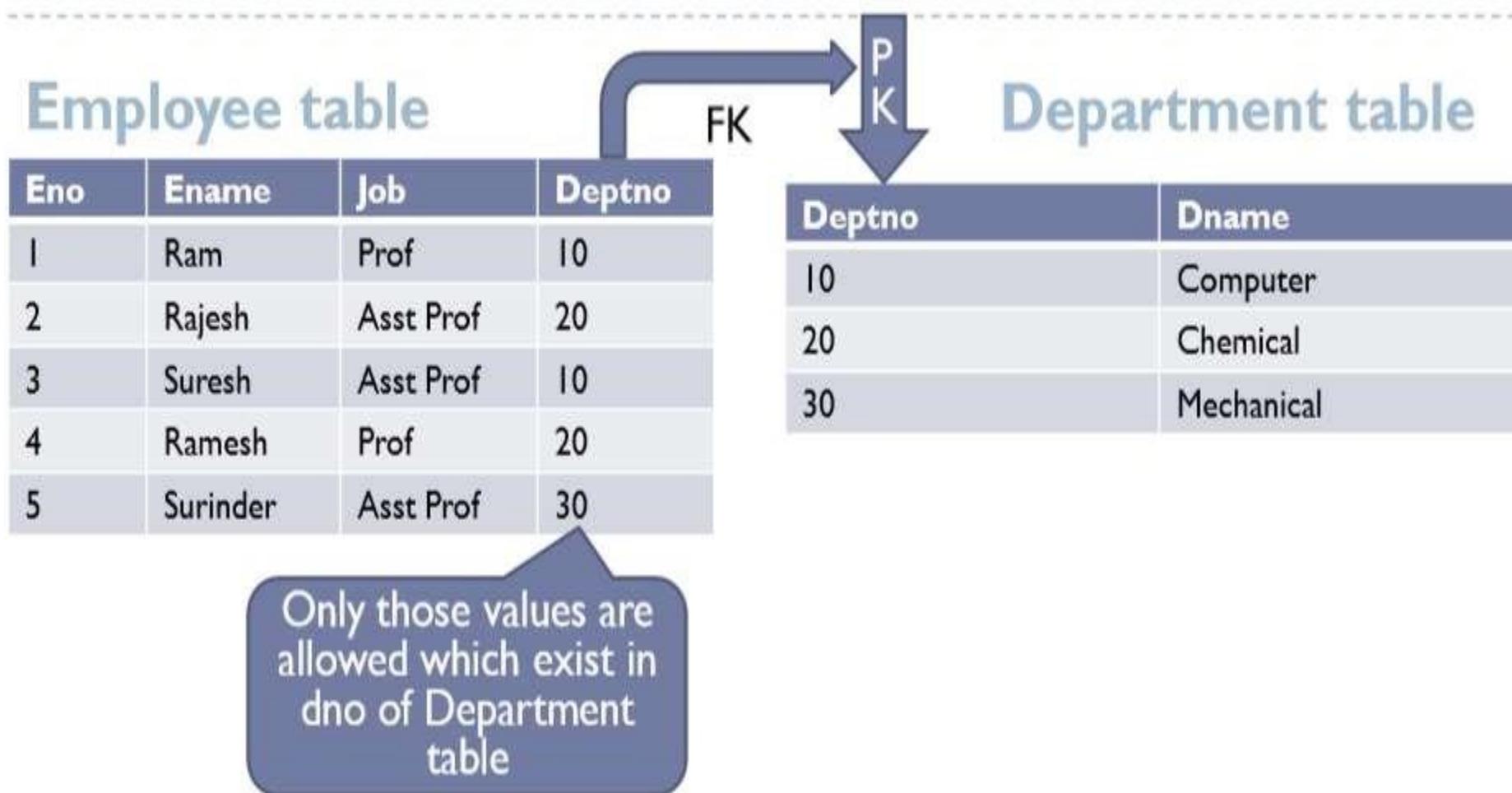
InvoiceNo	CustomerID	Amount
1	1	\$100
2	1	\$200
3	2	\$150

In the above example, we have 2 relations, Customer and Billing.

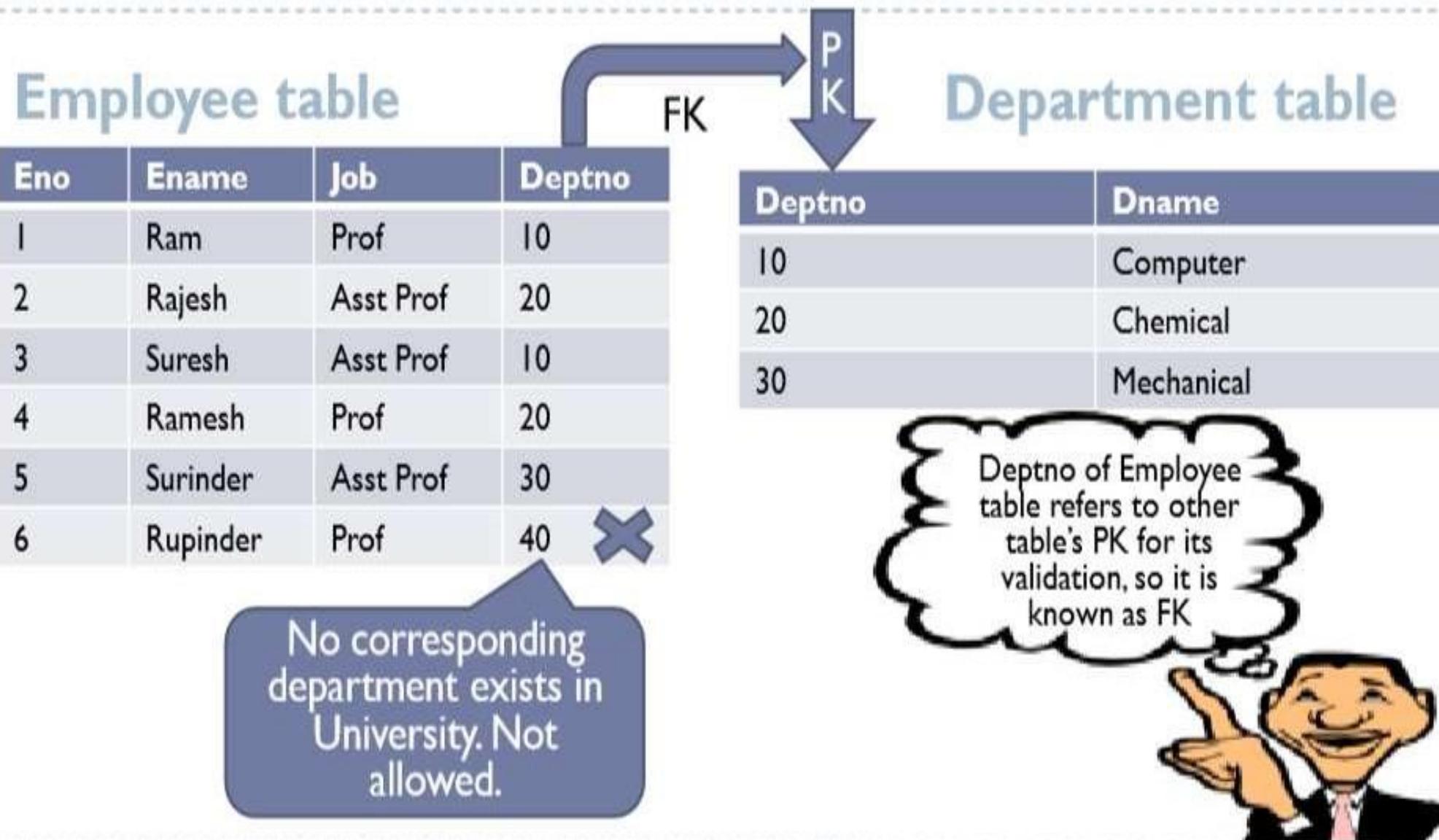
Tuple for CustomerID =1 is referenced twice in the relation Billing. So we know CustomerName=Google has billing amount \$300

Foreign Key Constraint

Concept of Foreign Key: Case Study EMP-DEPT



Concept of Foreign Key: Case Study EMP-DEPT



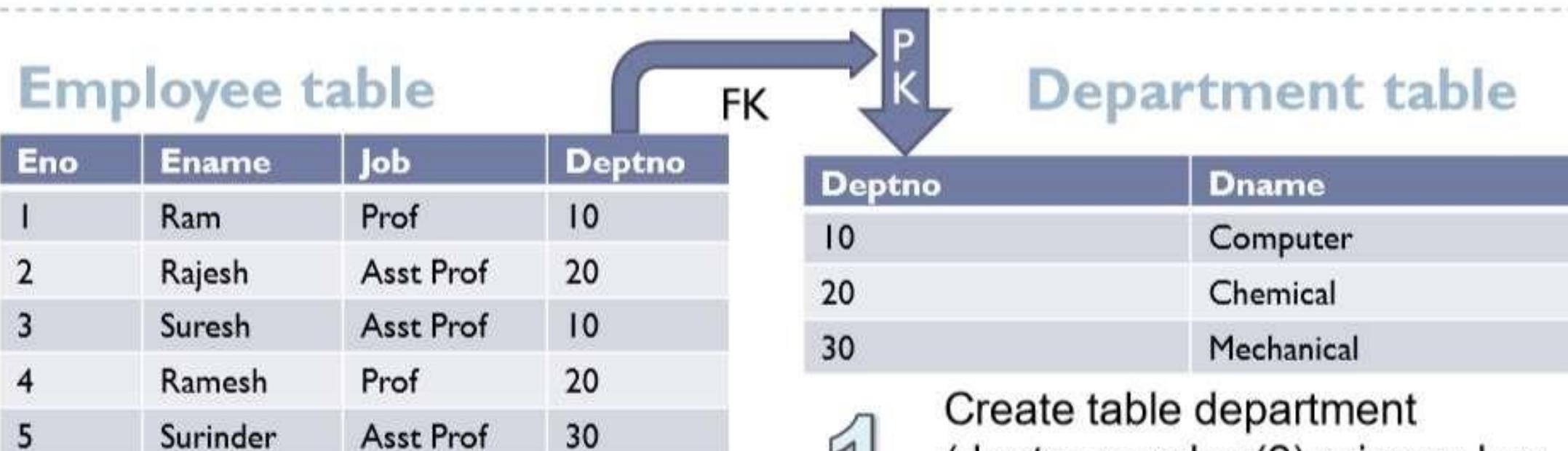
Foreign key Constraint

- ▶ Used to enforce referential integrity of data and establishes a relationship among tables.
- ▶ A foreign key may be a single column or the combinations of columns, which derive their values, based on the primary key of some other table.
- ▶ A table in which foreign key is present is known as child table and to which it refers is known as master or parent table.
- ▶ Its Syntax is:

Columnname datatype (size) references tablename [(columnname)]

[ON DELETE RESTRICT/ON DELETE CASCADE/ON DELETE SET
NULL]

Syntax: Case Study EMP-DEPT

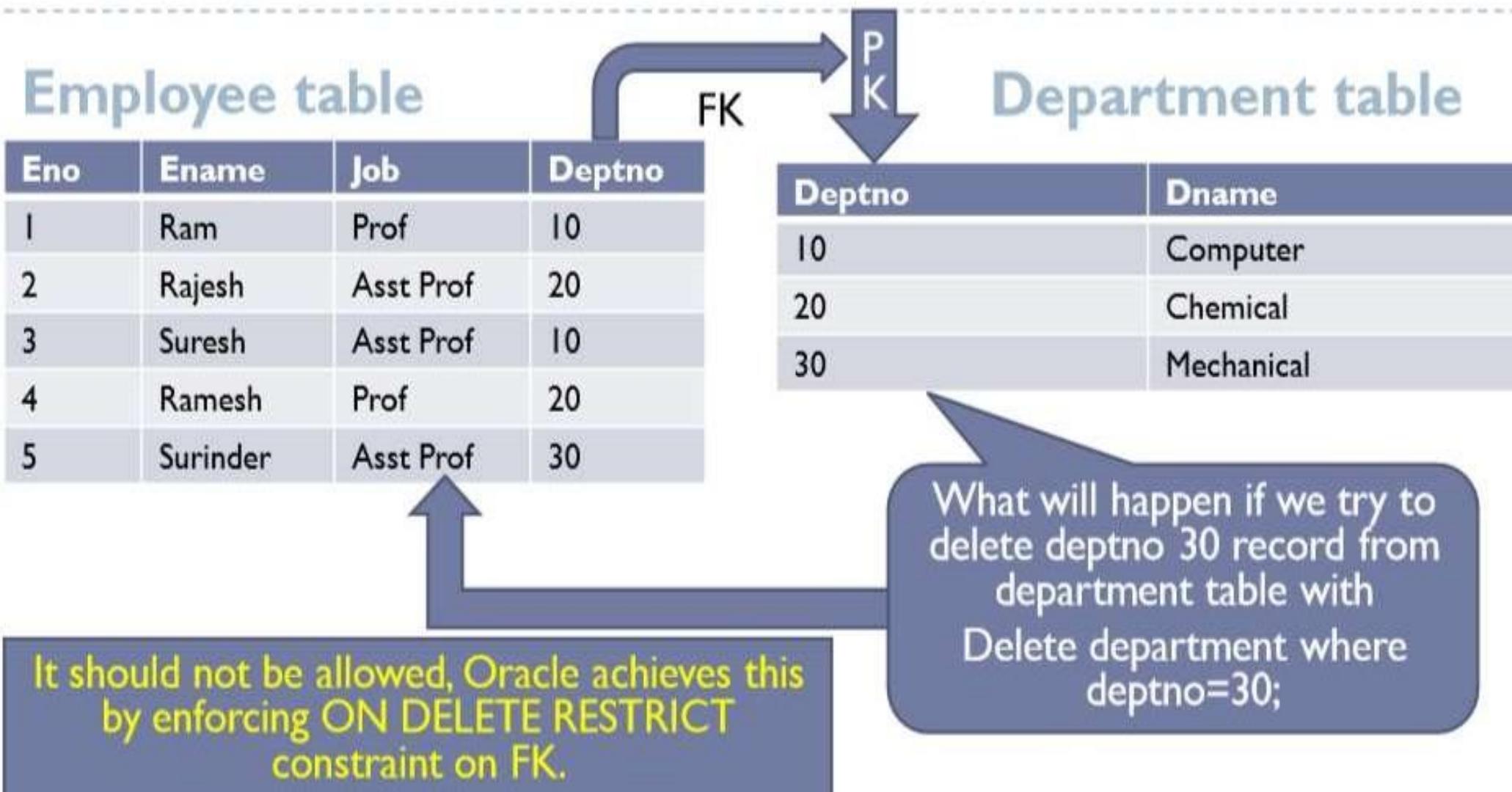


- 2 Create table employee
(eno number(2) primary key,
ename char(20), job char(10),
deptno number(2) references department(deptno));

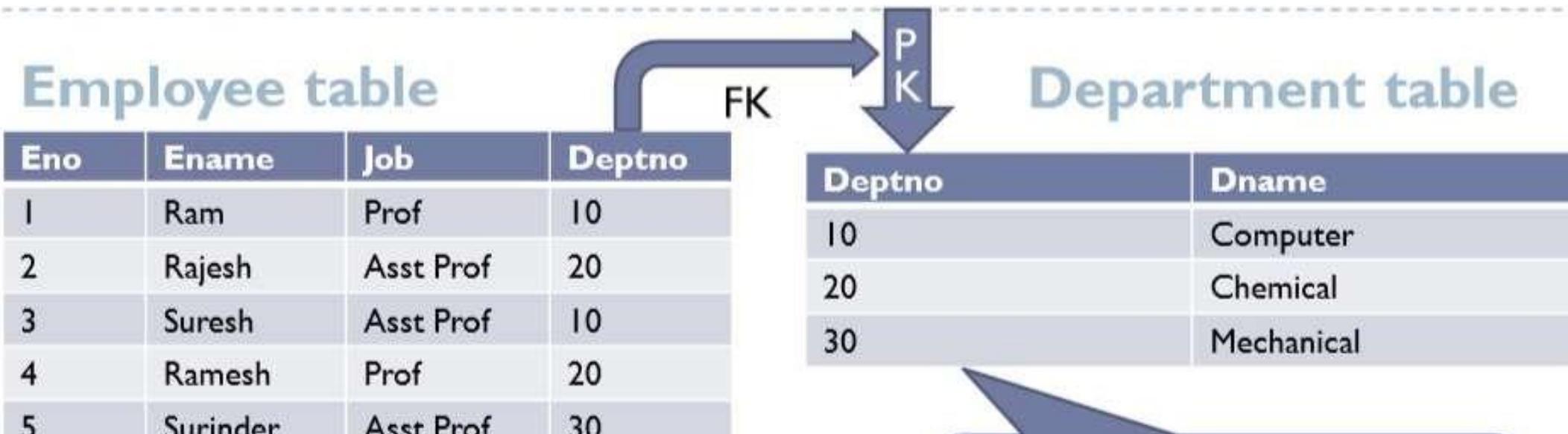
1

Create table department
(deptno number(2) primary key,
dname char(20));

Concept of Foreign Key: Case Study EMP-DEPT



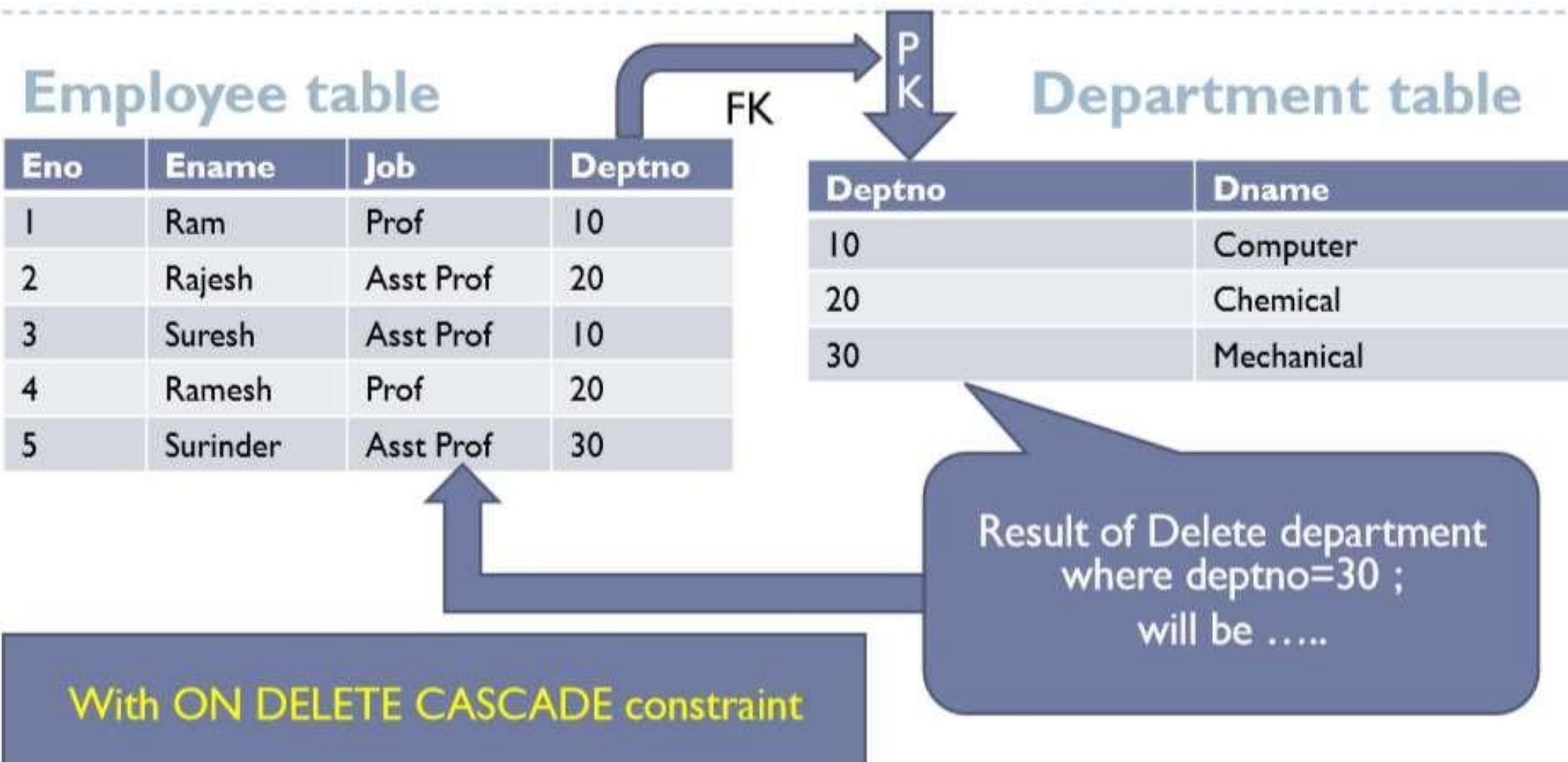
Concept of Foreign Key: Case Study EMP-DEPT



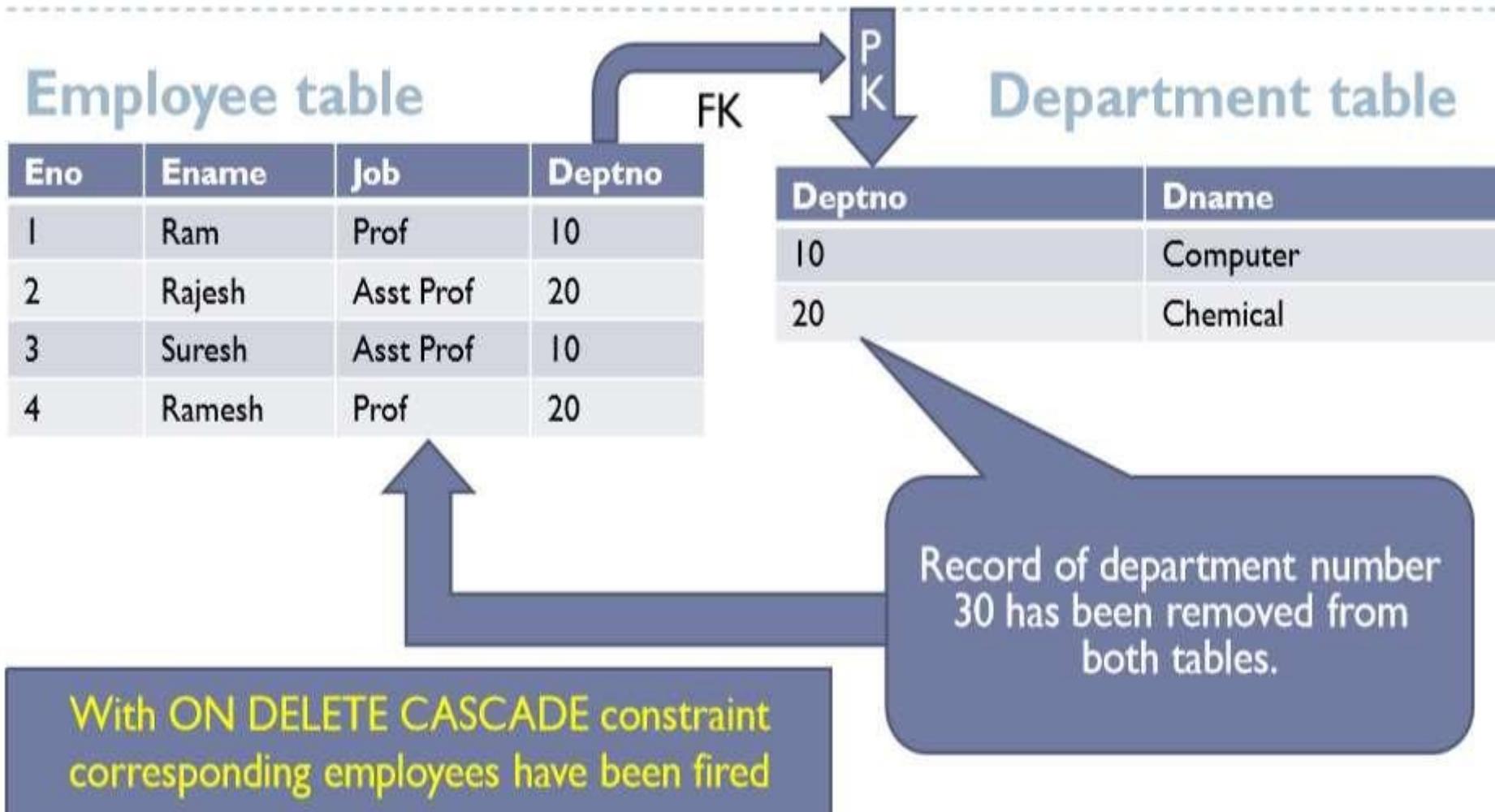
What will be the solution if we have to delete department number 30 from department table

Solution: Apply ON DELETE CASCADE constraint during creation of table so that department and corresponding employees records removed together.

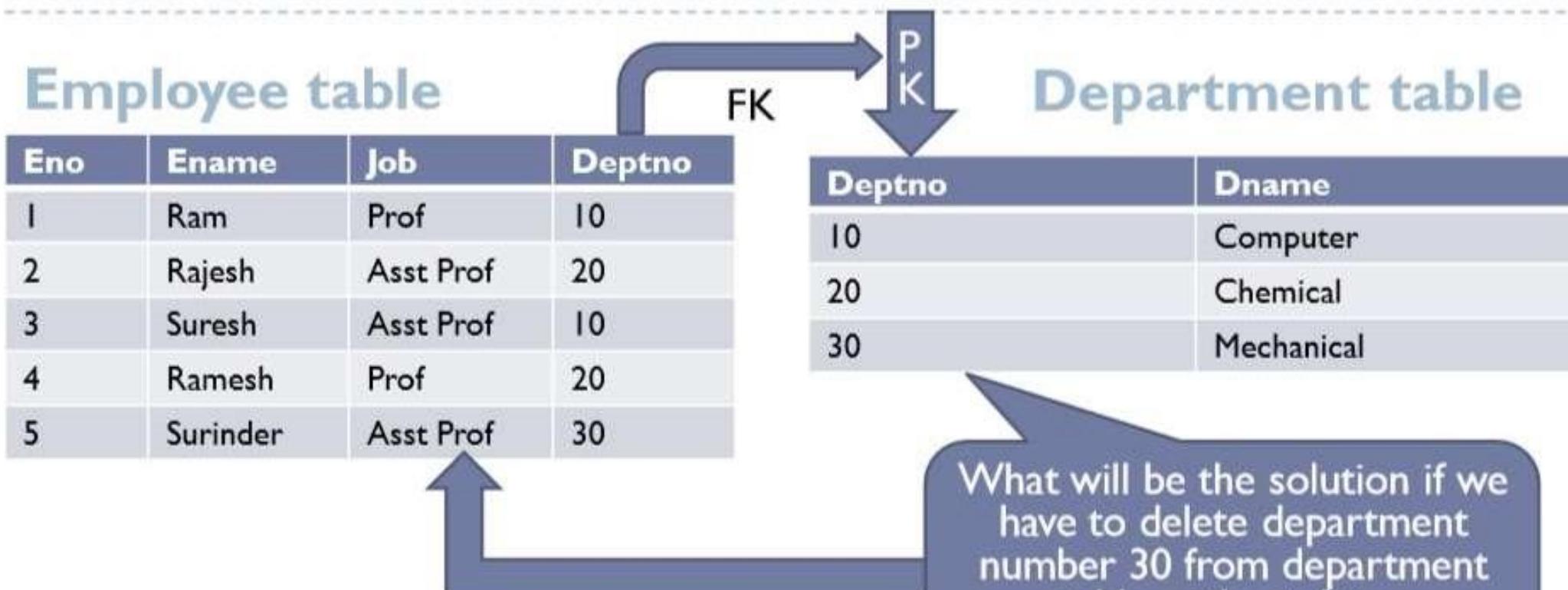
Concept of Foreign Key: Case Study EMP-DEPT



Concept of Foreign Key: Case Study EMP-DEPT



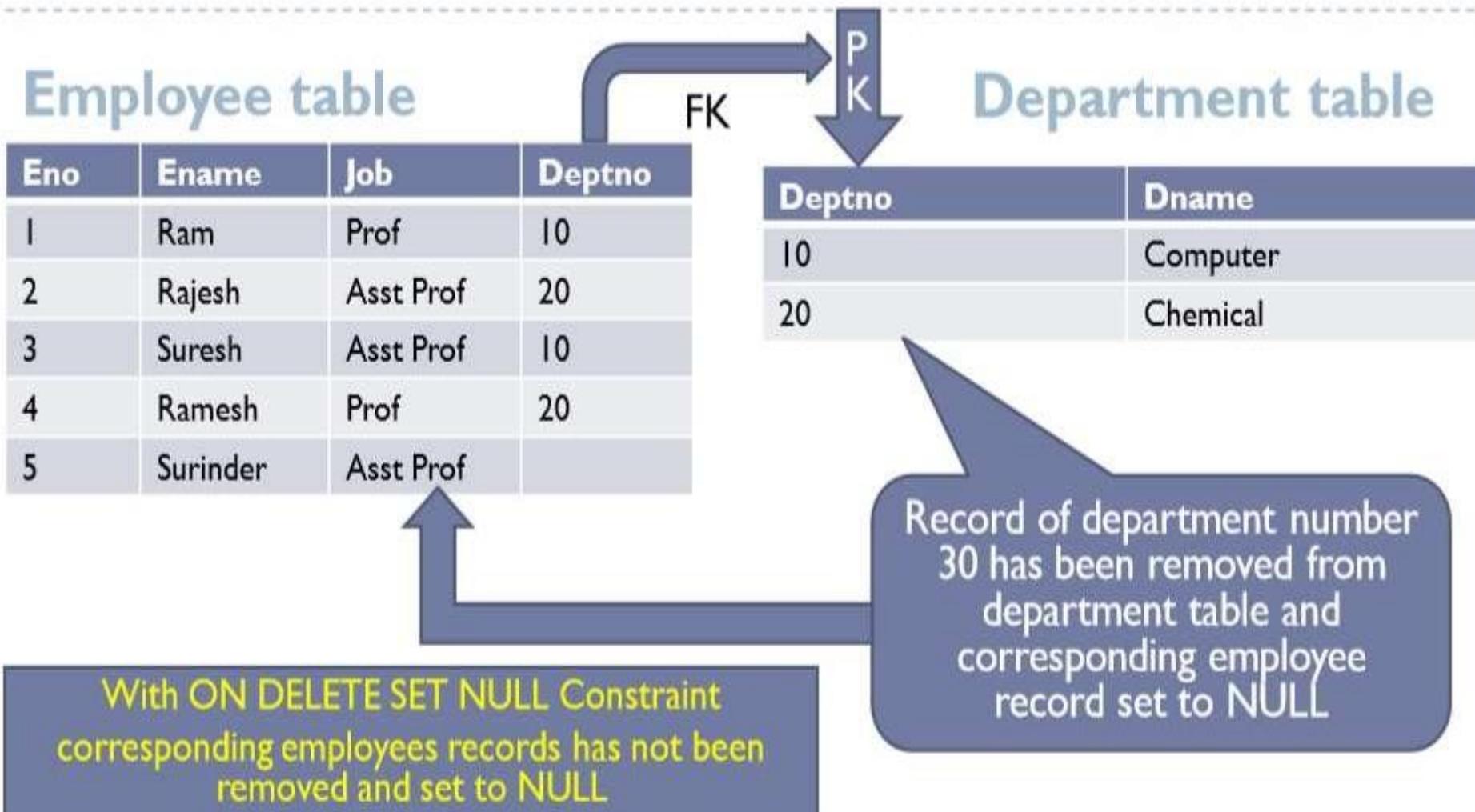
Concept of Foreign Key: Case Study EMP-DEPT



Solution: Apply ON DELETE SET NULL
Corresponding employees records will not be removed and their dno will be set to NULL

What will be the solution if we have to delete department number 30 from department table, without firing corresponding employees?

Concept of Foreign Key: Case Study EMP-DEPT



Creating Master and Child tables with ON DELETE RESTRICT (By default)

- ▶ Parent Table
- ▶ SQL>CREATE TABLE department
 - (DEPTNO NUMBER(2) PRIMARY KEY,
 - DNAME CHAR(10));
- ▶ Child table
- ▶ SQL>CREATE TABLE employee(
 - empno NUMBER(4) PRIMARY KEY,
 - ename CHAR(20), job CHAR(10),
 - deptno NUMBER(2) REFERENCES department(deptno)
ON DELETE RESTRICT);

Creating Master and Child tables with ON DELETE CASCADE

- ▶ Parent Table
- ▶ SQL>CREATE TABLE department
 - (DEPTNO NUMBER(2) PRIMARY KEY,
 - DNAME CHAR(10));
- ▶ Child table
- ▶ SQL>CREATE TABLE employee(
 - empno NUMBER(4) PRIMARY KEY,
 - ename CHAR(20), job CHAR(10),
 - deptno NUMBER(2) REFERENCES department(deptno)
 - ON DELETE CASCADE);

Creating Master and Child tables with ON DELETE SET NULL

- ▶ Parent Table
- ▶ SQL>CREATE TABLE department
(DNO NUMBER(2) PRIMARY KEY,
DNAME CHAR(10));
- ▶ Child table
- ▶ SQL>CREATE TABLE employee(
empno NUMBER(4) PRIMARY KEY,
ename CHAR(20), job CHAR(10),
dno NUMBER(2) REFERENCES department(deptno)
ON DELETE SET NULL);

DEFAULT constraint

The **DEFAULT constraint** provides a default value to a column when the **INSERT INTO** statement does not provide a specific value.

- ▶ The default value constraint allows the user to insert the values in the columns where the user do not want to insert the value.
- ▶ This is not actually a constraint and used to insert default value if value is missing for this column.
- ▶ The datatype of the default value should match the datatype of the column.
- ▶ Syntax:
- ▶ Columnname datatype (size) default value

Example [Default: Column level constraint]

```
CREATE TABLE student
(
    Roll_Number Number(4) Primary key,
    Name Char(15),
    Class Char(10) Check (class in ('BE','ME','MCA')),
    Marks Number(4) default 60,
    DOB Date
);
```

DEFAULT constraint

Example

For example, the following SQL creates a new table called CUSTOMERS and adds five columns.

Here, the **SALARY column is set to 5000.00 by default**, so in case the INSERT INTO statement does not provide a value for this column, then by default this column would be set to 5000.00.

```
CREATE TABLE CUSTOMERS(  
    ID INT      NOT NULL,  
    NAME VARCHAR (20) NOT NULL,  
    AGE INT      NOT NULL,  
    ADDRESS CHAR (25) ,  
    SALARY DECIMAL (18, 2) DEFAULT 5000.00,  
    PRIMARY KEY (ID)  
);
```

if the CUSTOMERS table has already been created, then to add a DEFAULT constraint to the SALARY column, you would write a query like the one which is shown in the code block below.

```
ALTER TABLE CUSTOMERS  
MODIFY SALARY DECIMAL (18, 2) DEFAULT 5000.00;
```

Drop Default Constraint

To drop a DEFAULT constraint, use the following SQL query.

```
ALTER TABLE CUSTOMERS  
ALTER COLUMN SALARY DROP DEFAULT;
```

USER_CONSTRAINTS and USER_CONS_COLUMNS tables to list constraints details

- ▶ To see the name of the constraints on a table you can query **USER_CONSTRAINTS** table (maintains internally by Oracle).

To query it:

- ▶ **SELECT * FROM USER_CONSTRAINTS WHERE
TABLE_NAME='NAME_OF_TABLE';**

For example:

- ▶ **SELECT * FROM USER_CONSTRAINTS WHERE
TABLE_NAME='STUDENT';**
- ▶ **SELECT * FROM USER_CONS_COLUMNS WHERE
TABLE_NAME='STUDENT';**

Viewing Constraints

Query the `USER_CONSTRAINTS` table to view all constraint definitions and names.

```
SELECT    constraint_name, constraint_type,  
          search_condition  
FROM      user_constraints  
WHERE     table_name = 'EMPLOYEES';
```

CONSTRAINT_NAME	C	SEARCH_CONDITION
EMP_LAST_NAME_NN	C	"LAST_NAME" IS NOT NULL
EMP_EMAIL_NN	C	"EMAIL" IS NOT NULL
EMP_HIRE_DATE_NN	C	"HIRE_DATE" IS NOT NULL
EMP_JOB_NN	C	"JOB_ID" IS NOT NULL
EMP_SALARY_MIN	C	salary > 0
EMP_EMAIL_UK	U	
...		

Viewing the Columns Associated with Constraints

View the columns associated with the constraint names in the `USER_CONS_COLUMNS` view.

```
SELECT      constraint_name, column_name
FROM        user_cons_columns
WHERE       table_name = 'EMPLOYEES';
```

CONSTRAINT_NAME	COLUMN_NAME
EMP_DEPT_FK	DEPARTMENT_ID
EMP_EMAIL_NN	EMAIL
EMP_EMAIL_UK	EMAIL
EMP_EMP_ID_PK	EMPLOYEE_ID
EMP_HIRE_DATE_NN	HIRE_DATE
EMP_JOB_FK	JOB_ID
EMP_JOB_NN	JOB_ID

Adding a Constraint Syntax

Use the **ALTER TABLE** statement to:

- **Add or drop a constraint, but not modify its structure**
- **Enable or disable constraints**
- **Add a NOT NULL constraint by using the MODIFY clause**

```
ALTER TABLE table
ADD [CONSTRAINT constraint] type (column);
```

Adding a Constraint

Add a FOREIGN KEY constraint to the EMPLOYEES table indicating that a manager must already exist as a valid employee in the EMPLOYEES table.

```
ALTER TABLE      employees
ADD CONSTRAINT  emp_manager_fk
    FOREIGN KEY(manager_id)
    REFERENCES employees(employee_id) ;
```

Table altered.

Cascading Constraints

- The **CASCADE CONSTRAINTS** clause is used along with the **DROP COLUMN** clause.
- The **CASCADE CONSTRAINTS** clause drops all referential integrity constraints that refer to the primary and unique keys defined on the dropped columns.
- The **CASCADE CONSTRAINTS** clause also drops all multicolumn constraints defined on the dropped columns.

Disabling Constraints

- Execute the **DISABLE** clause of the **ALTER TABLE** statement to deactivate an integrity constraint.
- Apply the **CASCADE** option to disable dependent integrity constraints.

```
ALTER TABLE employees  
DISABLE CONSTRAINT emp_emp_id_pk CASCADE;  
Table altered.
```

Enabling Constraints

- Activate an integrity constraint currently disabled in the table definition by using the ENABLE clause.

```
ALTER TABLE employees  
ENABLE CONSTRAINT emp_emp_id_pk;  
Table altered.
```

- A UNIQUE or PRIMARY KEY index is automatically created if you enable a UNIQUE key or PRIMARY KEY constraint.

Thank You

Structured Query Language (SQL)

Data Types: [Click here to Read Oracle's Data Types](#)

Introduction to SQL

What is SQL?

- SQL stands for Structured Query Language.
- SQL is used to create, remove, alter the database and database objects in a database management system and to store, retrieve, update the data in a database.
- SQL is a standard language for creating, accessing, manipulating database management system.

Introduction to SQL

Concept of SQL

- The user specifies a certain condition.
- The program will go through all the records in the database file and select those records that satisfy the condition (searching).
- Statistical information of the data.
- The result of the query will then be stored in form of a table.

Introduction to SQL

- **Different Types of SQL Commands:**
- DDL (Data Definition Language).
- DML (Data Manipulation Language).
- DCL (Data Control Language).
- TCL (Transaction Control Language).
- DQL (Data Query Language).

DDL: Data Definition Language

All DDL commands are auto-committed. That means it saves all the changes permanently in the database.

Command	Description
Create	to create new table or database
Alter	for alteration
Truncate	delete data from table
Drop	to drop a table
Rename	to rename a table

DML: Data Manipulation Language

DML commands are not auto-committed. It means changes are not permanent to database, they can be rolled back.

Command	Description
Insert	to insert a new row
Update	to update existing row
Delete	to delete a row
Merge	merging two rows or two tables

TCL: Transaction Control Language

These commands are to keep a check on other commands and their effect on the database. These commands can annul changes made by other commands by rolling back to original state. It can also make changes permanent.

.

Command	Description
Commit	to permanently save
Rollback	to undo change
Savepoint	to save temporarily

DCL : Data Control Language

Data control language provides command to grant and take back authority.

Command	Description
Grant	grant permission of right
Revoke	take back permission.

Data Definition Language

- DDL consists of the SQL commands that can be used to define the database schema.
- Deals with descriptions of the database schema
- Used to create and modify the structure of database objects in the database.
- List of Commands: Create, Drop, and Alter

Data Definition Language

Create: It is used to create a table

Syntax: Create table tablename (column1 data_type (size), column2 data_type (size)...);

Query:

```
Create table Customer (CustomerId number,  
CustomerName char(20), ContactNo number, address  
varchar (20), city char(20), Postalcode varchar(20),  
country char (20);
```

Data Definition Language

Alter command

alter command is used for alteration of table structures.

There are various uses of *alter* command, such as,

- * to add a column to existing table
- * to rename any existing column
- * to change datatype of any column or to modify its size.
- * *alter* is also used to drop a column.

Data Definition Language

Alter command

To Add Column to existing Table

Using alter command we can add a column to an existing table. Following is the Syntax,

```
alter table table-name add(column-name datatype);
```

Here is an Example for this,

```
alter table Students add (address varchar(50));
```

This command will add a new column *address* to the **Students** table

Data Definition Language

Alter command

To Add Multiple Column to existing Table

Using alter command we can even add multiple columns to an existing table. Following is the Syntax,

```
alter table table-name add(column-name1datatype1, column-name2datatype2);
```

Here is an Example for this,

```
alter table students add (dob number(10), city varchar(10));
```

Data Definition Language

Alter command

To Add column with Default Value

Alter command can add a new column to an existing table with default values. Following is the Syntax.

```
alter table table-name add(column-name1 datatype1 default data);
```

```
alter table students add (branch varchar(10) default 'COE');
```

Data Definition Language

Alter command

To Modify an existing Column

alter command is used to modify data type of an existing column . Following is the Syntax,

```
alter table table-name modify(column-name datatype);
```

Here is an Example for this,

```
alter table students modify (address char(30));
```

The above command will modify *address* column of the **Students**

table

Data Definition Language

Alter command

To Rename a column

Using alter command you can rename an existing column.

Following is the Syntax,

alter table *table-name* rename column old-column-name to column-name;

Here is an Example for this,

alter table Student rename address to Location;

The above command will rename *address* column to *Location*.

Data Definition Language

Alter command

To Drop a Column

alter command is also used to drop columns also.
Following is the Syntax,

```
alter table table-name drop(column-name);
```

Here is an Example for this,

```
alter table Students drop (address);
```

Data Definition Language

Truncate command

Truncate command removes all records from a table. But this command will not destroy the table's structure. When we apply truncate command on a table its Primary key is initialized. Following is its Syntax,

```
truncate table table-name
```

Here is an Example explaining it.

```
truncate table students;
```

The above query will delete all the records of **students** table.

Data Definition Language

Drop command

Drop query completely removes a table from database. This command will also destroy the table structure.

Following is its Syntax,

```
drop table table-name;
```

;

Here is an Example explaining it.

```
drop table students;
```

The above query will delete the **students** table completely. It can also be used on Databases. For Example, to drop a database,

```
drop database Test;
```

The above query will drop a database named **Test** from the system

Data Definition Language

Rename query

Rename command is used to rename a table. Following is its Syntax,

```
rename old_table_name to new_table_name;
```

Here is an Example explaining it.

```
rename students to people;
```

The above query will rename **Students** table to **People**.

Data Manipulation Language

- The SQL commands that deals with the manipulation of data present in the database.
- Insert, select, select distinct, update are DML commands.
- Insert: Insert is used to insert values in the created table

Syntax: Insert into tablename values (column1, column2,...column);

Query: Insert into customer values(1, ‘Aman Sharma’, ‘Obere Str. 57’,’Berlin’,12209,’Germany');

Data Manipulation Language

1) INSERT command

Insert command is used to insert data into a table. Following is its general syntax,

```
INSERT into table-namevalues(data1,data2,...)
```

Consider a table **Student** with following fields.

Students(Roll, Name, Age)

```
insert into Students values(1, 'mani',22);
insert into Students values(2, 'mahi',24);

insert into Students values(3, 'appy',20);
```

Data Manipulation Language

1) INSERT command

Example to Insert NULL value to a column

The statements below will insert NULL value into **age** column of the Student table.

```
insert into students values(4,'padm',null);
```

ID	NAME	AGE
1	Mani	22
2	Mahi	24
3	Appy	20
4	Padm	-

Example to Insert Default value to a column

```
insert into students values(5,'rahul', default);
```

Suppose the **age** column of student table has default value of 18, then for the age field 18 value will be added in new record.

Data Manipulation Language

2) UPDATE command

Update command is used to update a row of a table. Following is its general syntax,

```
UPDATEtable-name set column-name = value where condition;
```

Lets see an example,

```
update students set age=22 where id=4;
```

Example to Update multiple columns

```
update students set name='pri',age=25 where id=5;
```

Data Manipulation Language

3) Delete command

Delete command is used to delete data from a table. Delete command can also be used with condition to delete a particular row. Following is its general syntax,

```
DELETE from table-name;
```

Example to Delete all Records from a Table

```
delete from students;
```

The above command will delete all the records from **Students** table.

Example to Delete a particular Record from a Table

```
delete from students where id=5;
```

The above command will delete the record where id is 5 from **Students** table.

Data Query Language

SQL Query: SELECT

- ✓ Select query is used to retrieve data from a tables.
- ✓ It is the most used SQL query.
- ✓ We can retrieve complete tables, or partial by mentioning conditions using WHERE clause.

Basic structure of an SQL query

General Structure	Data Definition Language (DDL): Create, Drop, Alter Data Manipulation Language (DML): Select, Select Distinct, Insert, Update, Delete
Comparison	IN, BETWEEN, LIKE "% _"
Grouping	GROUP BY, HAVING, COUNT(), SUM(), AVG(), MAX(), MIN()
Display Order	ORDER BY, ASC / DESC
Logical Operators	AND, OR, NOT
Output	INTO TABLE / CURSOR TO FILE [ADDITIVE], TO PRINTER, TO SCREEN
Union	UNION

Example: North wind Sample Dataset (Customer Table)

CustomerID	CustomerName	Contact Name	Address	City	PostalCode	Country
1	Aman	Sharma	Obere Str. 57	Berlin	12209	Germany
2	Rohan	Garg	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Mohan	Goel	Mataderos 2312	México D.F.	05023	Mexico
4	Sohan	wadhwa	120 Hanover Sq.	London	WA1 1DP	UK
5	Kawal	Sahni	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Aarohan	Verma	125 Hanover Sq.	London	WA1 1DP	UK

The table above contains six records and seven columns (CustomerID, CustomerName, ContactName, Address, City, PostalCode, and Country).

General Structure: Select Statement

Select: It is used to extract data from a database.

Syntax: SELECT [* / DISTINCT] expr1 [col1], expr2 [col2] FROM *tablename* WHERE *condition*;

- * will extract all columns of table including duplicates values.
- DISTINCT will extract only the value which are unique.
- expr1, expr2 can be a column name or any expression of a function.
- col1 and col2 are the particular column names you want to extract.
- tablename is the name of the table from which you want to extract the values.
- Where clause is used where you want to extract columns of some specific value.

Examples: Select Statement

Note: We are using the Customer Table for all the examples.

eg 1. Select all values from customer

Query: Select * from Customer;

Output:

CustomerID	CustomerName	Contact Name	Address	City	PostalCode	Country
1	Aman	Sharma	Obere Str. 57	Berlin	12209	Germany
2	Rohan	Garg	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Mohan	Goel	Mataderos 2312	México D.F.	05023	Mexico
4	Sohan	wadhwa	120 Hanover Sq.	London	WA1 1DP	UK
5	Kawal	Sahni	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Aarohan	Verma	125 Hanover Sq.	London	WA1 1DP	UK

Examples: Select Statement

Note: We are using the Customer Table for all the examples.

eg 2. Extract distinct city from customer table.

Query: Select DISTINCT City from Customer;

Output: The customer database consists of 6 rows in which 4 cities are distinct. Therefore four cities are extracted as shown below.

City
Berlin
México D.F.
Luleå
London

Examples: Select Statement

Note: We are using the Customer Table for all the examples.

eg 3. Extract customerName and Contact Name of Germany People from customer table.

Query: Select customerName, ContactName from Customer where country = ‘Germany’;

Output:

CustomerName	Contact Name	Country
Aman	Sharma	Germany

Examples: Select Statement

Note: We are using the Customer Table for all the examples.

eg 4. Extract all values of customer id 1 from customer table.

Query: Select * from Customer where id = 1;

Output:

CustomerID	CustomerName	Contact Name	Address	City	PostalCode	Country
1	Aman	Sharma	Obere Str. 57	Berlin	12209	Germany

Examples: Select Statement

Note: We are using the Customer Table for all the examples.

eg 5. Extract Customer name, Contact name and postal code of Mexico D.F city of Mexico People.

Query: Select CustomerName, ContactName, PostalCode from
Customer where City='Mexico D.F.' AND Country='Mexico';

Output:

CustomerName	Contact Name	PostalCode
Rohan	Garg	05021
Mohan	Goel	05023

Update Statement

Update Command is used to update the records in the table

Syntax: UPDATE table_name SET column1 = value1, column2 = value2...., columnN = valueN WHERE [condition];

Query: Update Customer set CustomerName='abc', where City='Berlin.' AND Country='Germany';

Output:

CustomerID	CustomerName	Contact Name	Address	City	PostalCode	Country
1	abc	Sharma	Obere Str. 57	Berlin	12209	Germany

Comparison

1. IN: The **IN** command allows you to specify multiple values in a WHERE clause. The IN operator is a shorthand for **multiple OR conditions**.

Syntax: Select col1 from tablename Where IN (value1, value2, value3);

2. Between: The **BETWEEN** command is used to select values within a given range. The values can be numbers, text, or dates.

Syntax: Select col1 from tablename Where condition BETWEEN Value1 AND Value2;

3. Like: The **LIKE** command is used in a WHERE clause to search for a specified pattern in a column.

Syntax: Select col1 from tablename Where col like ‘val1’;

Comparison (IN)

Note: We are using the Customer Table for all the examples.

eg 1. Extract all values from customer table of id 1, 4 and 5.

Query: Select * from Customer where id IN (1,4,5);

Output:

CustomerID	CustomerName	Contact Name	Address	City	PostalCode	Country
1	Aman	Sharma	Obere Str. 57	Berlin	12209	Germany
4	Sohan	wadhwa	120 Hanover Sq.	London	WA1 1DP	UK
5	Kawal	Sahni	Berguvsvägen 8	Luleå	S-958 22	Sweden

Comparison (Not IN)

Note: We are using the Customer Table for all the examples.

eg 2. Extract all values from customer table who are not from London and Berlin.

Query: Select * from Customer where city NOT IN ('London', 'Berlin');

Output:

CustomerID	CustomerName	Contact Name	Address	City	PostalCode	Country
2	Rohan	Garg	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Mohan	Goel	Mataderos 2312	México D.F.	05023	Mexico
5	Kawal	Sahni	Berguvsvägen 8	Luleå	S-958 22	Sweden

Comparison (Between)

Note: We are using the Customer Table for all the examples.

eg 3. Extract all values from customer table between 2 to 5.

Query: Select * from Customer where CustomerId Between 2 and 5;

Output:

CustomerID	CustomerName	Contact Name	Address	City	PostalCode	Country
3	Mohan	Goel	Mataderos 2312	México D.F.	05023	Mexico
4	Sohan	wadhwa	120 Hanover Sq.	London	WA1 1DP	UK

Comparison (Like)

Note: We are using the Customer Table for all the examples.

eg 4. Extract all values from customer table with a CustomerName ending with n.

Query: Select * from Customer where CustomerName LIKE '%n'

Output:

CustomerID	CustomerName	Contact Name	Address	City	PostalCode	Country
1	Aman	Sharma	Obere Str. 57	Berlin	12209	Germany
2	Rohan	Garg	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Mohan	Goel	Mataderos 2312	México D.F.	05023	Mexico
4	Sohan	wadhwa	120 Hanover Sq.	London	WA1 1DP	UK
6	Aarohan	Verma	125 Hanover Sq.	London	WA1 1DP	UK

Comparison (LIKE)

Note: We are using the Customer Table for all the examples.

eg 5. Extract all values from customer table with a CustomerName starting with a.

Query: Select * from Customer where CustomerName LIKE ‘a%’

Output:

CustomerID	CustomerName	Contact Name	Address	City	PostalCode	Country
1	Aman	Sharma	Obere Str. 57	Berlin	12209	Germany
6	Aarohan	Verma	125 Hanover Sq.	London	WA1 1DP	UK

The following SQL (LIKE)

Note: We are using the Customer Table for all the examples.

eg 6. Selects all customers with a CustomerName starting with a.

Query: Select * from Customer where CustomerName LIKE ‘a%’

Output:

CustomerID	CustomerName	Contact Name	Address	City	PostalCode	Country
1	Aman	Sharma	Obere Str. 57	Berlin	12209	Germany
6	Aarohan	Verma	125 Hanover Sq.	London	WA1 1DP	UK

The following SQL (LIKE)

Note: We are using the Customer Table for all the examples.

eg 6. Selects all customers with a CustomerName that have “ha” in any position.

Query: Select * from Customer where CustomerName LIKE ‘%ha%’

Output:

CustomerID	CustomerName	Contact Name	Address	City	PostalCode	Country
2	Rohan	Garg	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Mohan	Goel	Mataderos 2312	México D.F.	05023	Mexico
4	Sohan	wadhwa	120 Hanover Sq.	London	WA1 1DP	UK
6	Aarohan	Verma	125 Hanover Sq.	London	WA1 1DP	UK

Logical Operators AND, OR, NOT

- The AND, OR and NOT operators are used to filter records based on more than one condition:
- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.
- The NOT operator displays a record if the condition(s) is NOT TRUE.

AND

Syntax: SELECT *column1, column2, ...*
FROM *table_name*
WHERE *condition1 AND condition2 AND condition3 ...;*
Query: Select DeptId and Salary from employee where
name='Anjali' and city='Bangalore'
Output:

DeptId	Salary
1014	58000

OR

Syntax: SELECT *column1, column2, ...*

FROM *table_name*

WHERE *condition1 OR condition2 OR condition3 ...;*

Query: Select DeptId and Salary from employee where name='Anjali' or city='Bangalore'

Output:

DeptId	Salary
1014	58000
1003	34000
1014	45000

NOT

Syntax: SELECT *column1, column2, ...*

FROM *table_name* WHERE NOT *condition1*;

Query: Select DeptId and Salary from employee where city<>‘Bangalore’

Output:

Dept_Id	Salary
1014	29001
1023	54000
1003	67000
1023	56000
1003	29000
1249	87000
1003	64000

Set Operations in SQL

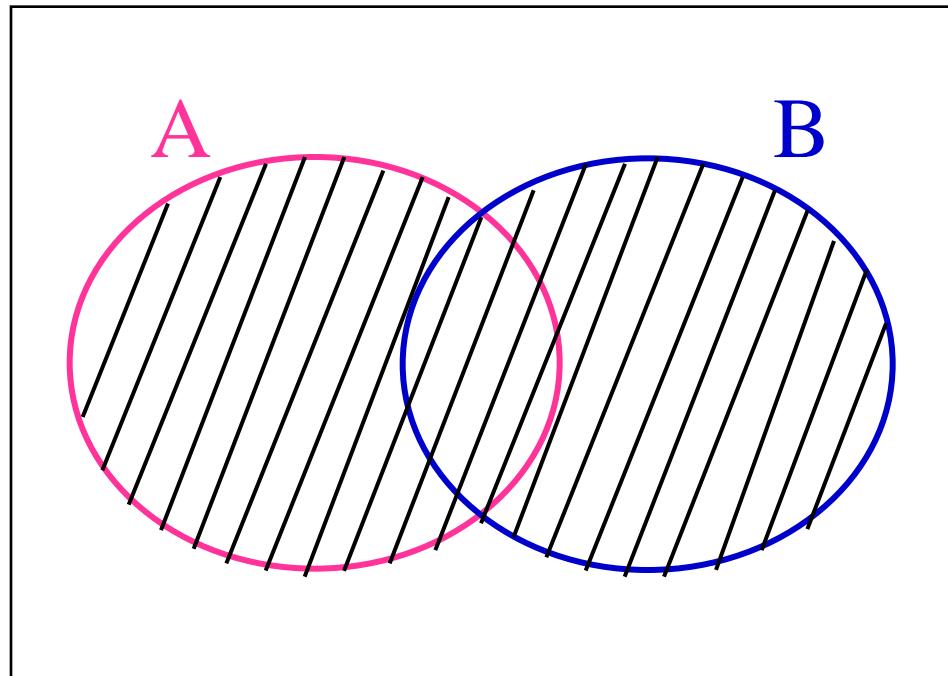
SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.

- UNION
- UNION ALL
- INTERSECT
- MINUS

Union Operation in SQL

UNION is used to combine the results of two or more **SELECT** statements. However, it will eliminate duplicate rows from its result set. In case of union, **number of columns and datatype** must be same in both the tables, on which UNION operation is being applied.

The ***union*** of A and B ($A \cup B$)



A table containing all the rows from **A** and **B**.

Union

Syntax: SELECT FROM WHERE

UNION

SELECT FROM WHERE;

Table1

ID	Name
1	abhi
2	adam

Table2

ID	Name
2	adam
3	chester

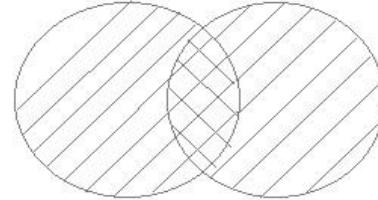
```
SELECT * FROM Table1 UNION SELECT *  
FROM Table2;
```

OUTPUT

ID	NAME
1	abhi
2	adam
3	chester

Union All

This operation is similar to Union. But it also shows the duplicate rows.



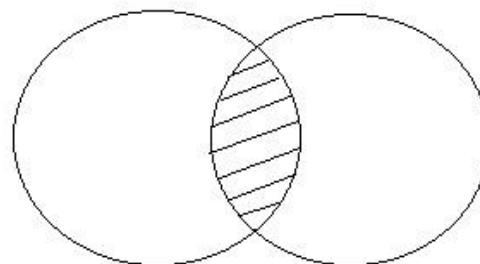
```
SELECT * FROM Table1 UNION ALL SELECT * FROM Table2;
```

OUTPUT

	ID	NAME
	1	abhi
	2	adam
	2	adam
	3	chester

Intersection

Intersect operation is used to combine two **SELECT** statements, but it only returns the records which are common from both **SELECT** statements. In case of **Intersect** the **number of columns and datatype must be same**.



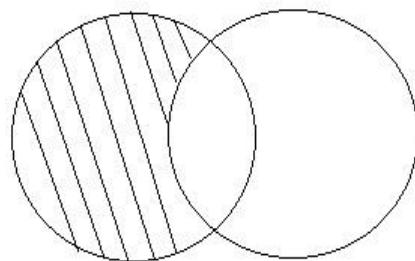
```
SELECT * FROM Table1 INTERECT SELECT * FROM  
Table2;
```

OUTPUT

ID	NAME
2	adam

Difference or Minus

The Minus operation combines results of two **SELECT** statements and return only those in the final result, which belongs to the first set of the result.



```
SELECT * FROM Table1 MINUS SELECT * FROM  
Table2;
```

OUTPUT

ID	NAME
1	abhi

More Examples

Customer

first_name	last_name
Stephen	Jones
Mark	Smith
Denise	King
Paula	Johnson
Richard	Archer

Employee

first_name	last_name
Christina	Jones
Michael	McDonald
Paula	Johnson
Stephen	Jones
Richard	Smith

Union

eg 1. Combine both customer and employee table.

Query:

```
Select first_name, last_name from Customer
```

```
UNION
```

```
Select first_name, last_name from Employee
```

Union

Output:

First_name	last_name
Stephen	Jones
Mark	Smith
Denise	King
Paula	Johnson
Richard	Archer
Christina	Jones
Michael	McDonald
Richard	Smith

Union

Example 2. Combine both customer and employee table in ascending order.

Query:

```
Select first_name, last_name from Customer
```

```
UNION
```

```
Select first_name, last_name from Employee Order
```

Intersection

SELECT FROM *table1* ;

Intersect

SELECT *col* FROM *table2*

eg 2. Find common first_name and last_name from both customer and employee table.

Query:

Select first_name, last_name from customer

Intersect

Select first_name, last_name from customer;

Intersection

Output:

FIRST_NAME	LAST_NAME
Paula	johson
Stephen	jones

You can also use IN to find intersection

Difference of Tables

SELECT FROM *table1* ;

SELECT *col* FROM *table2*)

eg 3. Find first name and last name from customer which are not in employee.

Query:

Select * from customer

MINUS

Select * from employee;

Difference of Tables

Output:

FIRST_NAME	LAST_NAME
Denise	king
Mark	Smith
Richard	Archer

You can also use NOT IN to find difference.

Grouping

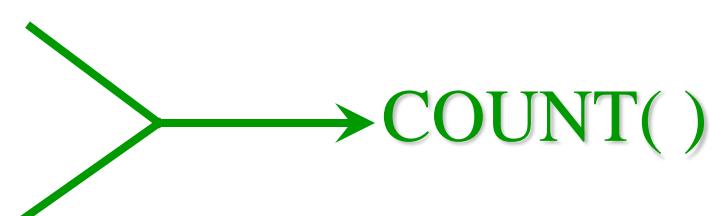
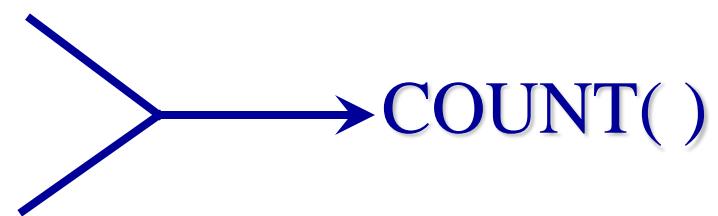
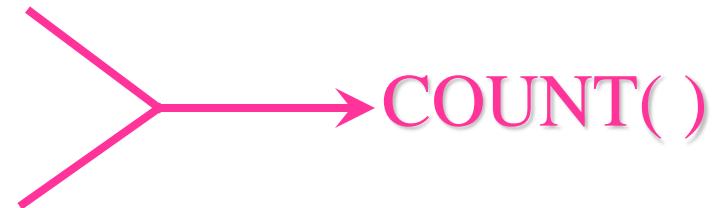
eg. 11 List the number of students of each class.

↓ Group By Class

class

		1A		
		1A		
		1A		
		1B		
		1C		
		1C		
		1C		

Student



Grouping

eg. 11 List the number of students of each class.

```
SELECT class, COUNT(*) FROM student  
GROUP BY class;
```

Result

class	cnt
1A	10
1B	9
1C	9
2A	8
2B	8
2C	6

Grouping

eg. 12 List the average Math test score of each class.

↓ Group By Class

class

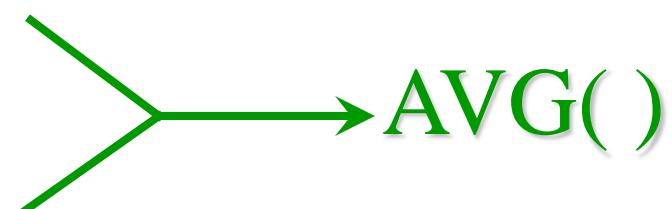
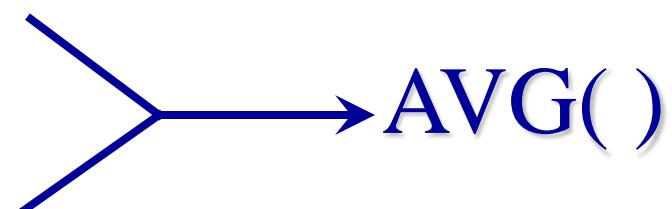
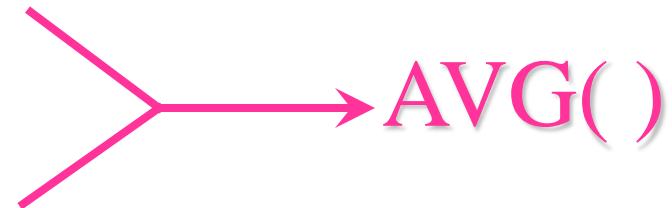
1A

1B

1C

		1A		
		1A		
		1A		
		1B		
		1C		
		1C		
		1C		

Student



Grouping

eg. 12 List the average Math test score of each class.

```
SELECT class, AVG(mtest) FROM student  
GROUP BY class;
```

Result

class	avg_mtest
1A	85.90
1B	70.33
1C	37.89
2A	89.38
2B	53.13
2C	32.67

Grouping, Cont'd...

The **GROUP BY** command is used to group the result set (used with aggregate functions: COUNT, MAX, MIN, SUM, AVG).

Syntax: `SELECT FROM WHERE condition`

`GROUP BY groupexpr [HAVING requirement];`

- ***groupexpr*** specifies the related rows to be grouped as one entry.
Usually it is a column.
- **WHERE *condition*** specifies the condition of individual rows before the rows are group. **HAVING *requirement*** specifies the condition involving the whole group.

Grouping

Note: Consider the following table for further queries .

ID	Dept_Id	Name	City	Salary
1	1014	Ajay	Bangalore	58000
2	1014	Harry	Delhi	29001
3	1003	Anjali	Bangalore	34000
4	1023	Rahul	Delhi	54000
5	1003	Anisha	Chennai	67000
6	1023	Rajni	Mumbai	56000
7	1003	Zaid	Mumbai	29000
8	1249	Bib	NY	87000
9	1014	Raj	Bangalore	45000
10	1003	Arya	Chennai	64000

Grouping (AVG)

eg 1. find the average salary for each department's employees

Query: Select Dept_Id, AVG (Salary) from Employee Group BY
Dept_ID

Output:

Dept_ID	AVG (Salary)
1014	44000.3333
1003	48500.0000
1023	55000.0000
1249	87000.0000

Grouping (Count)

eg 2. List the number of employees from each city

Query: Select City, COUNT(City) from Employee Group BY City;

Output:

City	Count (City)
Bangalore	3
Delhi	2
Chennai	2
Mumbai	2
NY	1

Grouping with Order By

eg 3. List the number of employees from each city in ascending order

Query: Select City, COUNT(City) from Employee Group BY City
ORDER By COUNT(City);

Output:

City	Count (City)
NY	1
Delhi	2
Chennai	2
Mumbai	2
Bangalore	3

Grouping with Order By

eg 4. List the number of employees from each city in descending order

Query: Select City, COUNT(City) from Employee Group BY City
ORDER By COUNT(City) DESC;

Output:

City	Count (City)
Bangalore	3
Delhi	2
Chennai	2
Mumbai	2
NY	1

Grouping (MAX)

eg 5. Find the maximum salary for each department with “Dept_ID” greater than 1003.

Query: Select Dept_ID, MAX(Salary) from Employee where Dept_ID>1003 Group BY Dept_ID;

Output:

Dept_ID	MAX (Salary)
1014	58000
1023	56000
1249	87000

Grouping (MIN)

eg 6. Find the minimum salary for each department with “Dept_ID” greater than 1003.

Query: Select Dept_ID, MIN(Salary) from Employee where Dept_ID>1003 Group BY Dept_ID;

Output:

Dept_ID	MAX (Salary)
1014	29001
1023	54000
1249	87000

Grouping (SUM)

eg 7. Find the total salary of all the employees:

Query: Select Count(Dept_ID), SUM(Salary) from Employee.

Output:

(Count) Dept_ID	SUM (Salary)
10	5230001

Grouping (Count)

eg 8. Find the number of employees from each city in all the departments.

Query: Select City, Dept_ID, Count (*) from Employee Group By City, Dept_ID Order By Count (*);

Output:

City	Dept_ID	Count (*)
Delhi	1014	1
Bangalore	1003	1
Delhi	1023	1
Mumbai	1023	1
Mumbai	1003	1
NY	1249	1
Bangalore	1014	2
Chennai	1003	2

Grouping with Having

eg 9. Find the cities with multiple employees.

Query: Select City, Count (City) from Employee Group By City
Having Count (City)>1 Order By Count (City);

Output:

City	Count (City)
Delhi	2
Chennai	2
Mumbai	2
Bangalore	3

END

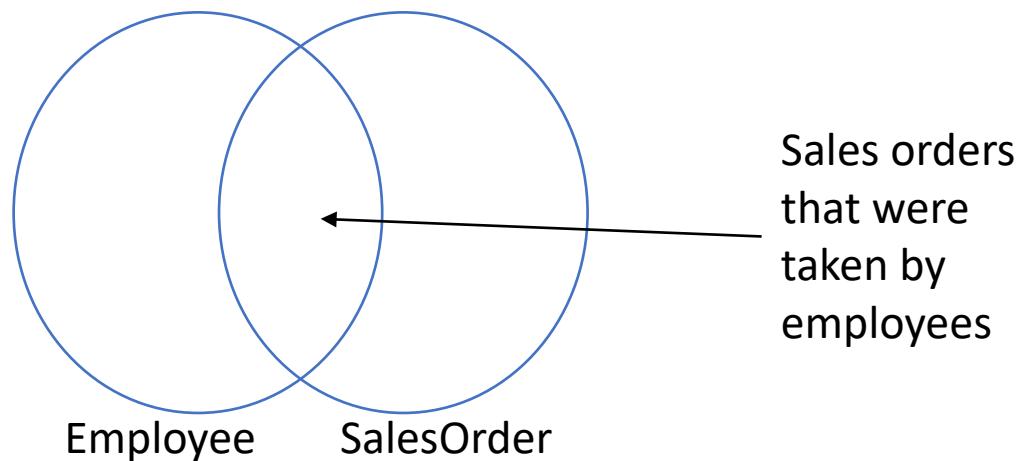
Joins in SQL

Joins on Multiple Tables:

- SQL provides a convenient operation to retrieve information from multiple tables.
- This operation is called **join**.
- The join operation will **combine** the tables into one large table with all possible combinations (Math: Cartesian Product), and then it will filter the rows of this combined table to yield useful information.

Join Concepts

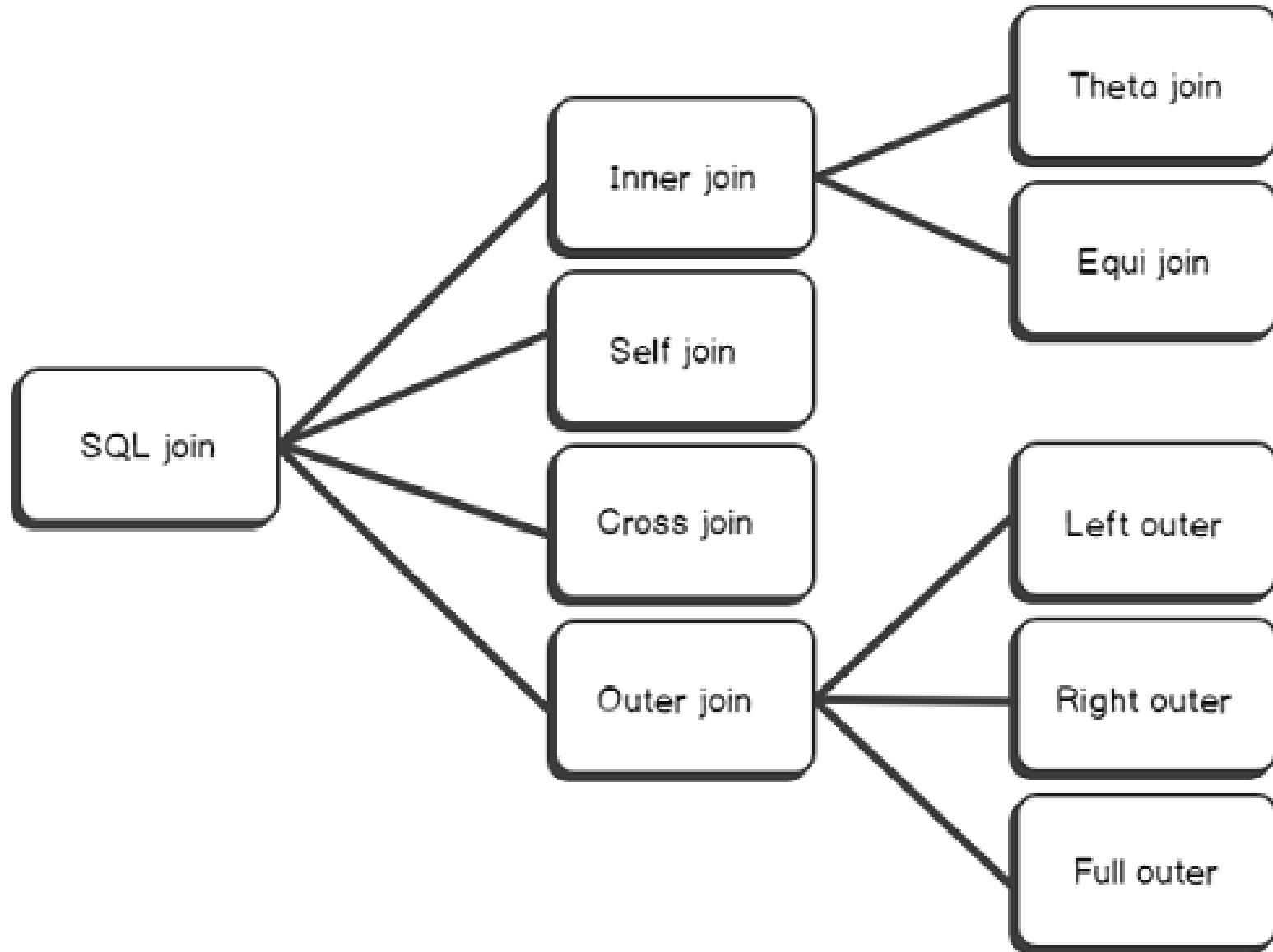
- Combine rows from multiple tables by specifying matching criteria
 - Usually based on primary key – foreign key relationships
 - For example, return rows that combine data from the **Employee** and **SalesOrder** tables by matching the **Employee.EmployeeID** primary key to the **SalesOrder.EmployeeID** foreign key
- It helps to think of the tables as sets in a Venn diagram



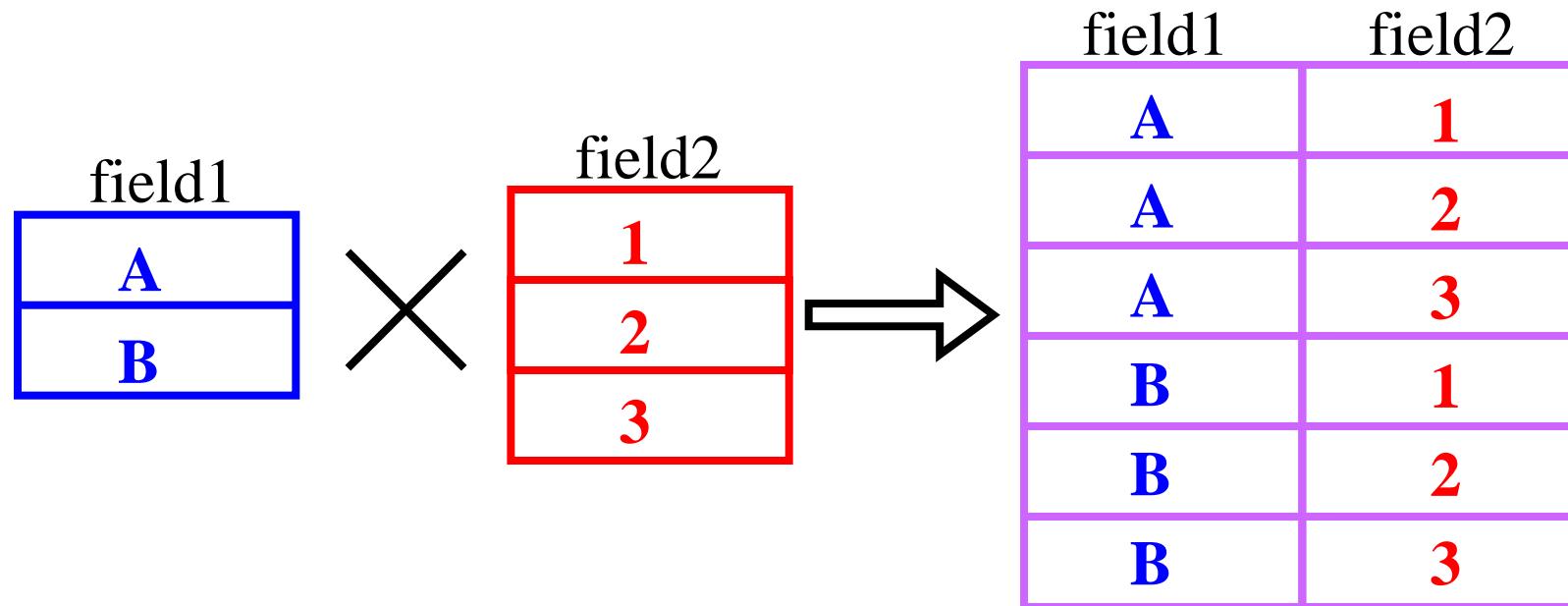
Different type of Joins

1. Cross Join
2. Inner Join
 - i. Equi/Equality join ('=')
 - ii. Natural join
 - iii. Theta join (or non-equi join (<, <=, >, >=, <>))
3. Outer Join
 - i. Left outer join
 - ii. Right outer join
 - iii. Full outer join
4. Self Join
5. Semi Join

Different type of Joins



Multiple Tables:



What kind of Join is this?

```
SELECT *  
FROM Students S ?? Enrolled E;
```

S	S.name	S.sid
Jones	11111	
Smith	22222	

E	E.sid	E.classid
11111	History105	
11111	DataScience194	
22222	French150	

S.name	S.sid	E.sid	E.classid
Jones	11111	11111	History105
Jones	11111	11111	DataScience194
Jones	11111	22222	French150
Smith	22222	11111	History105
Smith	22222	11111	DataScience194
Smith	22222	22222	French150

The **CROSS JOIN** is used to generate a paired combination of each row of the first table with each row of the second table. This **join** type is also known as **cartesian join**.

```
SELECT *  
FROM Students S CROSS JOIN Enrolled E;
```

S	S.name	S.sid
Jones	11111	
Smith	22222	

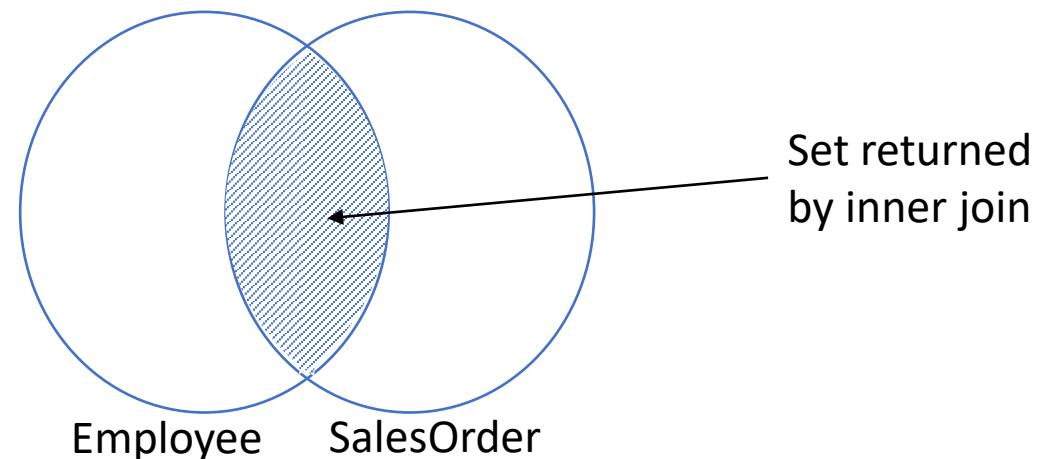
E	E.sid	E.classid
11111	History105	
11111	DataScience194	
22222	French150	

S.name	S.sid	E.sid	E.classid
Jones	11111	11111	History105
Jones	11111	11111	DataScience194
Jones	11111	22222	French150
Smith	22222	11111	History105
Smith	22222	11111	DataScience194
Smith	22222	22222	French150

Inner Joins

- Return only rows where a match is found in both input tables
- Match rows based on attributes supplied in predicate
- If join predicate operator is $=$, then it known as equi-join

```
SELECT emp.FirstName, ord.Amount  
FROM Employee emp  
[INNER] JOIN SalesOrder ord  
ON emp.EmployeeID = ord.EmployeeID
```



SQL Inner Joins (Equi join)

```
SELECT S.name, E.classid  
FROM Students S (INNER) JOIN Enrolled E  
ON S.sid=E.sid
```

S	S.name	S.sid
Jones	11111	
Smith	22222	
Brown	33333	

E	E.sid	E.classid
11111	History105	
11111	DataScience194	
22222	French150	
NULL	English10	

	S.name	E.classid
Jones	History105	
Jones	DataScience194	
Smith	French150	

Note the previous version of this query (with no join keyword) is an “Implicit join”

SQL Inner Joins (Equi join)

```
SELECT S.name, E.classid  
FROM Students S (INNER) JOIN Enrolled E  
ON S.sid=E.sid
```

S	S.name	S.sid
Jones	11111	
Smith	22222	
Brown	33333	

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150

E	E.sid	E.classid
11111	History105	
11111	DataScience194	
22222	French150	
NULL	English10	

Unmatched keys

Joins and Inference

- Chaining relations together is the basic inference method in relational DBs. It produces new relations (effectively new facts) from the data:

```
SELECT S.name, M.mortality  
FROM Students S, Mortality M  
WHERE S.Race=M.Race
```

S

Name	Race
Socrates	Man
Thor	God
Barney	Dinosaur
Blarney stone	Stone

M

Race	Mortality
Man	Mortal
God	Immortal
Dinosaur	Mortal
Stone	Non-living

Joins and Inference

- Chaining relations together is the basic inference method in relational DBs. It produces new relations (effectively new facts) from the data:

```
SELECT S.name, M.mortality  
FROM Students S, Mortality M  
WHERE S.Race=M.Race
```

Name	Mortality
Socrates	Mortal
Thor	Immortal
Barney	Mortal
Blarney stone	Non-living

Natural Join

Natural Join joins two tables based on same attribute name and datatypes. The resulting table will contain all the attributes of both the table but keep only one copy of each common column.

Student

Roll_No	Name
1	A
2	B
3	C

Mark

Roll_No	Marks
2	70
3	50
NULL	85

Select * from Student natural join Mark;

Roll_No	Name	Marks
2	B	70
3	C	50

Difference between Equi join and Natural join

```
SELECT * FROM student S INNER JOIN Marks M ON S.Roll_No = M.Roll_No;
```

Roll_No	Name	Roll_No	Marks
2	B	2	70
3	C	3	50

1.

Natural Join joins two tables based on same attribute name and datatypes.

2.

In Natural Join, The resulting table will contain all the attributes of both the tables but keep only one copy of each common column

3.

In Natural Join, If there is no condition specifies then it returns the rows based on the common column

4.

SYNTAX:

SELECT *

FROM table1 NATURAL JOIN table2;

Inner Join joins two table on the basis of the column which is explicitly specified in the ON clause.

In Inner Join, The resulting table will contain all the attribute of both the tables including duplicate columns also

In Inner Join, only those records will return which exists in both the tables

SYNTAX:

SELECT *

**FROM table1 INNER JOIN table2 ON
table1.Column_Name = table2.Column_Name;**

What kind of Join is this?

```
SELECT *
  FROM Students S, Enrolled E
 WHERE S.sid <= E.sid
```

S	S.name	S.sid
Jones	11111	
Smith	22222	

E	E.sid	E.classid
11111	History105	
11111	DataScience194	
22222	French150	

S.name	S.sid	E.sid	E.classid
Jones	11111	11111	History105
Jones	11111	11111	DataScience194
Jones	11111	22222	French150
Smith	22222	22222	French150

Theta Joins (<, <=, >, >=, <>)

```
SELECT *
  FROM Students S, Enrolled E
 WHERE S.sid <= E.sid
```

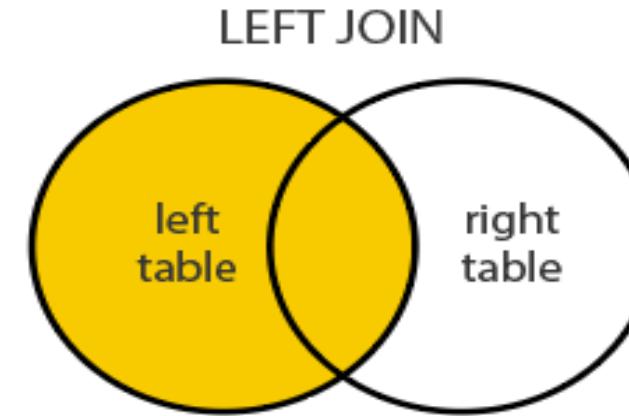
S	S.name	S.sid
Jones	11111	
Smith	22222	

E	E.sid	E.classid
11111	History105	
11111	DataScience194	
22222	French150	

S.name	S.sid	E.sid	E.classid
Jones	11111	11111	History105
Jones	11111	11111	DataScience194
Jones	11111	22222	French150
Smith	22222	22222	French150

Left Outer Join

A LEFT JOIN performs a join starting with the first (left-most) table. Then, any matched records from the second table (right-most) will be included. LEFT JOIN and LEFT OUTER JOIN are the same. The result is 0 records from the right side, if there is no match.



SELECT column-names

FROM table-name1 LEFT OUTER JOIN table-name2

ON column-name1 = column-name2

What kind of Join is this?

```
SELECT S.name, E.classid  
FROM Students S ?? Enrolled E  
ON S.sid=E.sid
```

S	S.name	S.sid
Jones	11111	
Smith	22222	
Brown	33333	

E	E.sid	E.classid
11111	History105	
11111	DataScience194	
22222	French150	
NULL	English10	

	S.name	E.classid
Jones	History105	
Jones	DataScience194	
Smith	French150	
Brown	NULL	

Query: List all students name and their classid irrespective whether they enrolled for that class or not.

```
SELECT S.name, E.classid  
FROM Students S LEFT OUTER JOIN Enrolled E  
ON S.sid=E.sid
```

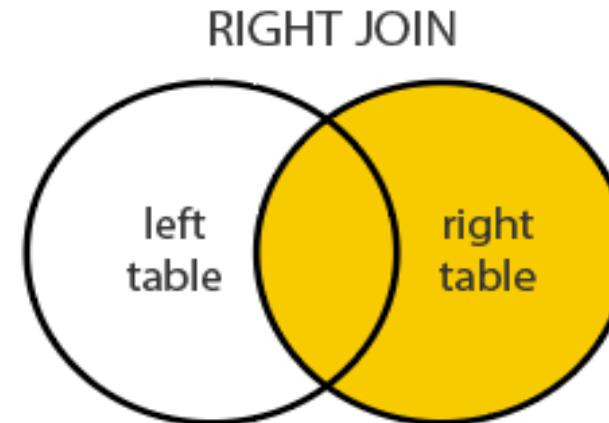
S	S.name	S.sid
Jones	11111	
Smith	22222	
Brown	33333	

E	E.sid	E.classid
11111	History105	
11111	DataScience194	
22222	French150	
NULL	English10	

	S.name	E.classid
Jones	History105	
Jones	DataScience194	
Smith	French150	
Brown	NULL	

Right outer join

A RIGHT JOIN performs a join starting with the second (right-most) table and then any matching first (left-most) table records. RIGHT JOIN and RIGHT OUTER JOIN are the same. The result is 0 records from the left side, if there is no match.



SELECT column-names

**FROM table-name1 RIGHT OUTER JOIN table-name2
ON column-name1 = column-name2**

What kind of Join is this?

```
SELECT S.name, E.classid  
FROM Students S ?? Enrolled E  
ON S.sid=E.sid
```

S	S.name	S.sid
Jones	11111	
Smith	22222	
Brown	33333	

E	E.sid	E.classid
11111	History105	
11111	DataScience194	
22222	French150	
-	English10	

	S.name	E.classid
Jones	History105	
Jones	DataScience194	
Smith	French150	
NULL	English10	

Query: List all students name and their classid irrespective whether for a class any student enrolled or not.

```
SELECT S.name, E.classid  
FROM Students S RIGHT OUTER JOIN Enrolled E  
ON S.sid=E.sid
```

S	S.name	S.sid
Jones	11111	
Smith	22222	
Brown	33333	

E	E.sid	E.classid
11111	History105	
11111	DataScience194	
22222	French150	
NULL	English10	

	S.name	E.classid
Jones	History105	
Jones	DataScience194	
Smith	French150	
NULL	English10	

SQL Joins

```
SELECT S.name, E.classid  
FROM Students S ? JOIN Enrolled E  
ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

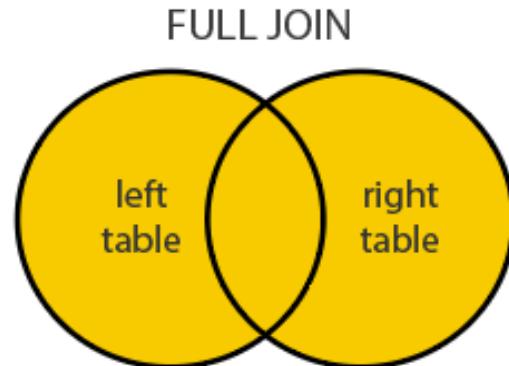
E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
NULL	English10

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
NULL	English10
Brown	NULL

Full outer join

FULL JOIN returns all matching records from both tables whether the other table matches or not. Be aware that a FULL JOIN can potentially return very large datasets. These two: FULL JOIN and FULL OUTER JOIN are the same.



SELECT column-names

FROM table-name1 FULL OUTER JOIN table-name2

ON column-name1 = column-name2

Query: List all students name and their classid irrespective whether a student enrolled for a class or not, and irrespective whether for a class any student enrolled or not.

```
SELECT S.name, E.classid  
FROM Students S FULL OUTER JOIN Enrolled E  
ON S.sid=E.sid
```

S	S.name	S.sid
Jones	11111	
Smith	22222	
Brown	33333	

	S.name	E.classid
Jones		History105
Jones		DataScience194
Smith		French150
NULL		English10
Brown		NULL

E	E.sid	E.classid
11111		History105
11111		DataScience194
22222		French150
NULL		English10

Self Join

- A self JOIN occurs when a table takes a 'selfie', that is, it JOINS with itself. A self JOIN is a regular join but the table that it joins to is itself.
- **Joining** a table with itself means that each row of the table is combined with itself and with every other row of the table. SELF JOINs are also useful for comparisons within a table.

SELECT column-names

FROM table-name T1 JOIN table-name T2

WHERE condition

<u>Emp-id</u>	<u>Name</u>	<u>Manager-id</u>
111	Ravi	NULL
112	Rakul	111
113	Pal	111
114	Yuvi	113
115	Juri	112

Query: Return all employees and the name of the employee's manager

```
SQL> SELECT e1.Name as Employee, e2.Name as Manager
      FROM Employee e1 JOIN Employee e2
      ON e1.Manager-id = e2.Employee-id;
```

Output:

	<u>Employee</u>	<u>Manager</u>
	Ravi	NULL
	Rakul	Ravi
	Pal	Ravi
	Yuvi	Pal
	Juri	Rakul

Semi-Join

- Semi join is a type of join whose result-set contains only the columns from one of the “semi-joined” tables. Each row from the first table(left table if Left Semi Join) will be returned maximum once, if matched in the second table. The duplicate rows from the first table will be returned, if matched once in the second Table. A distinct row from the first Table A will be returned no matter how many times matched in a second Table B.
- **The essential differences between a semi join and a regular join when we join two tables A and B are:**
 - Semi join either returns each row from Table A, or it does not. No row duplication can occur.
 - Regular join returns duplicates rows if there are multiple matches on the join predicate.
 - Semi join returns only columns from Table A.
 - Regular join may return columns from either (or both) join inputs.

What kind of Join is this?

```
SELECT S.name, E.classid  
FROM Students S ?? Enrolled E  
ON S.sid=E.sid
```

S	S.name	S.sid
Jones	11111	
Smith	22222	
Brown	33333	

E	E.sid	E.classid
11111	History105	
11111	DataScience194	
22222	French150	
NULL	English10	

	S.name	E.classid
Jones		History105
Smith		French150

SQL Joins

```
SELECT S.name, E.classid  
FROM Students S LEFT SEMI JOIN Enrolled E  
ON S.sid=E.sid
```

S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
NULL	English10

S.name	E.classid
Jones	History105
Smith	French150

END

Entity-Relationship Model

E-R Model

Basic concept of E-R diagram

Basic concepts

- What is Database Design?
 - Database Design is a collection of processes that facilitate the designing, development, implementation and maintenance of enterprise database management systems.
- What is E-R diagram?
 - E-R diagram: (Entity-Relationship diagram)
 - The ER data model was developed to facilitate database design by allowing specification of an **enterprise schema** that represents the overall logical structure of a database.
 - The ER data model employs three basic concepts:
 - entity sets,
 - relationship sets,
 - attributes.
 - The ER model also has an associated diagrammatic representation, the **ER diagram**, which can express the overall logical structure of a database graphically.

E-R diagram

Entity Name

Entity

- An **entity** is an object that exists and is distinguishable from other objects.
- An entity is a **person**, a **place** or an **object**.
- An entity is represented by a **rectangle** which contains the name of an entity.
- Entities of a **college database** are:
 - Student
 - Professor/Faculty
 - Course
 - Department
 - Result
 - Class
 - Subject

Course

Faculty

Student

Exercise:

List the different entities of bank database.

List the different entities of hospital database.

Entity:

- **Tangible Entity:** Tangible Entities are those entities which exist in the real world physically that can be touched, seen, or measured.
 - Example: Person, car, buildings etc.
- **Intangible Entity:** Intangible Entities are those entities which exist only logically and have no physical existence.
 - Example: Bank Account, login information etc.

Entity Set

- An **entity set** is a set of entities of the same type that share the same properties.
- Example:
 - All persons having an account in a bank
 - All the students studying in a college
 - All the professors working in a college
 - Set of all accounts in a bank

E-R diagram

Attributes

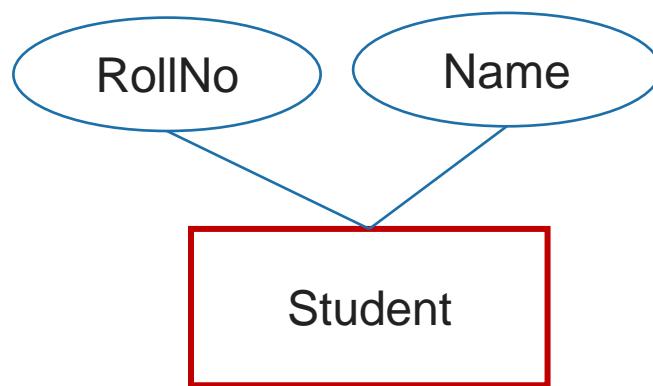
- Attribute is **properties** or details about an entity.



- An attribute is represented by an **oval** containing name of an attribute.

- Attributes of Student are:

- Roll No
- Student Name
- Branch
- Semester
- Address
- Mobile No
- Age
- SPI
- Backlogs



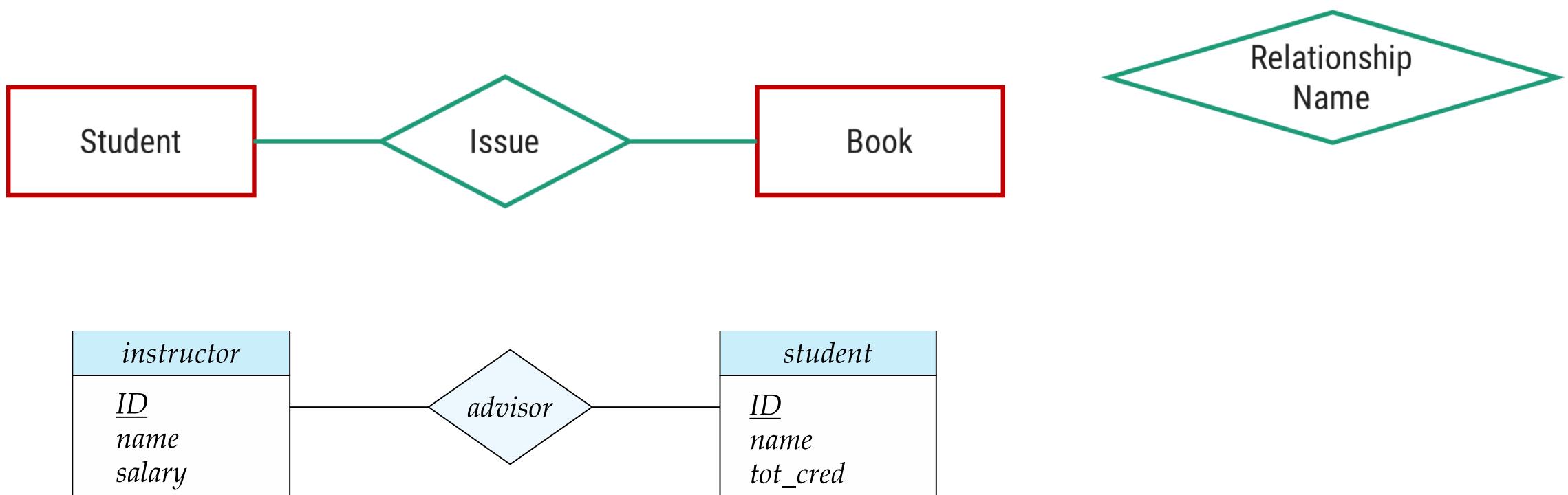
Exercise Write down the different attributes of Faculty entity.

Exercise Write down the different attributes of Account entity.

E-R diagram

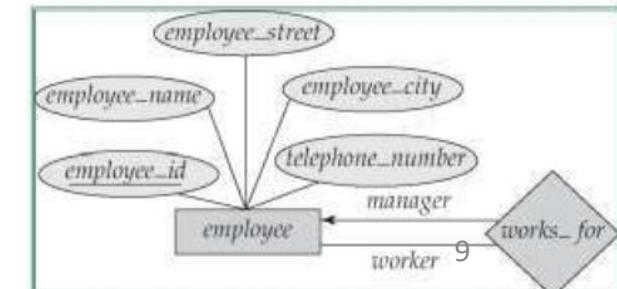
Relationship

- Relationship is an **association** (connection) between several entities.
- It should be placed between **two entities** and a line connecting it to an entity.
- A relationship is represented by a **diamond** containing relationship's name.



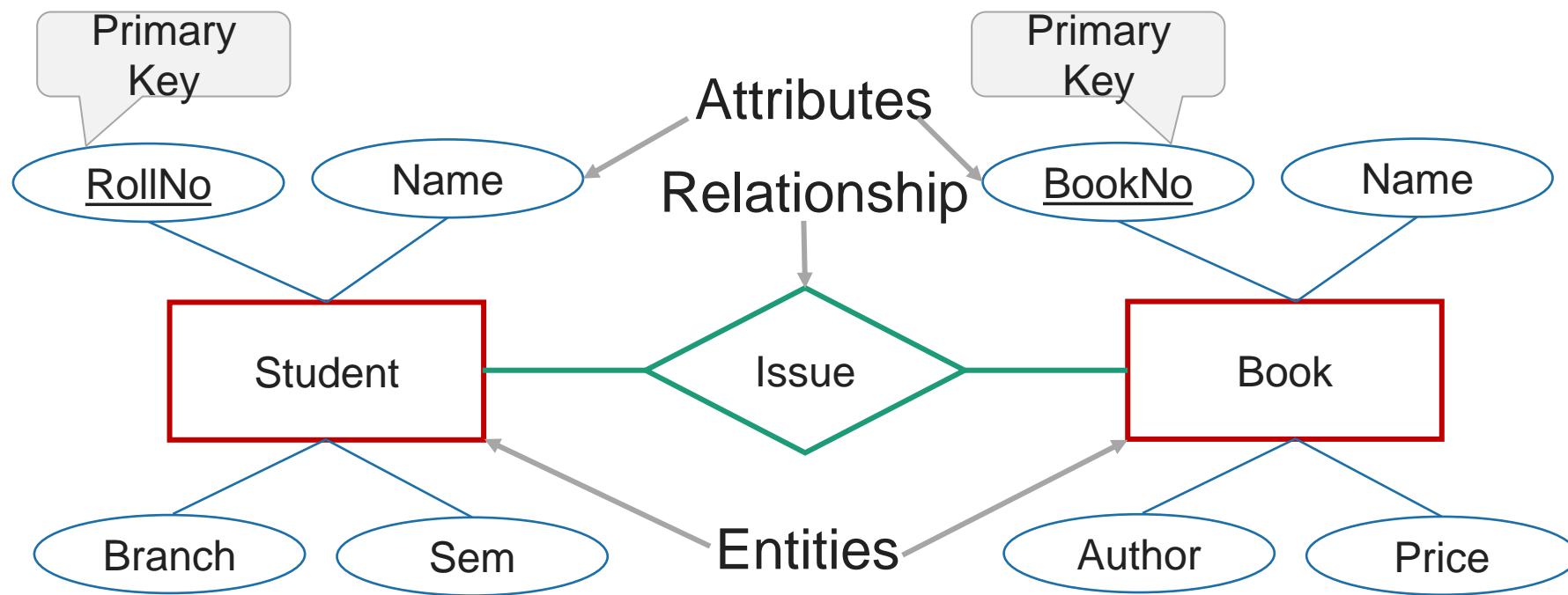
Degree of a Relationship

- A Relationship represents an association between two or more entities.
- Relationships are classified in terms of degree, connectivity, cardinality, and existence.
- The degree of a relationship is the number of entities associated with the relationship.
- The **n-ary relationship** is the general form for degree n.
- Special cases are the **binary**, and **ternary**, where the degree is 2, and 3, respectively
- A **recursive binary relationship** occurs when an entity is related to itself.
 - Example:
 - "some employees are married to other employees".
 - "Employee work_for a manager who is also an employee."



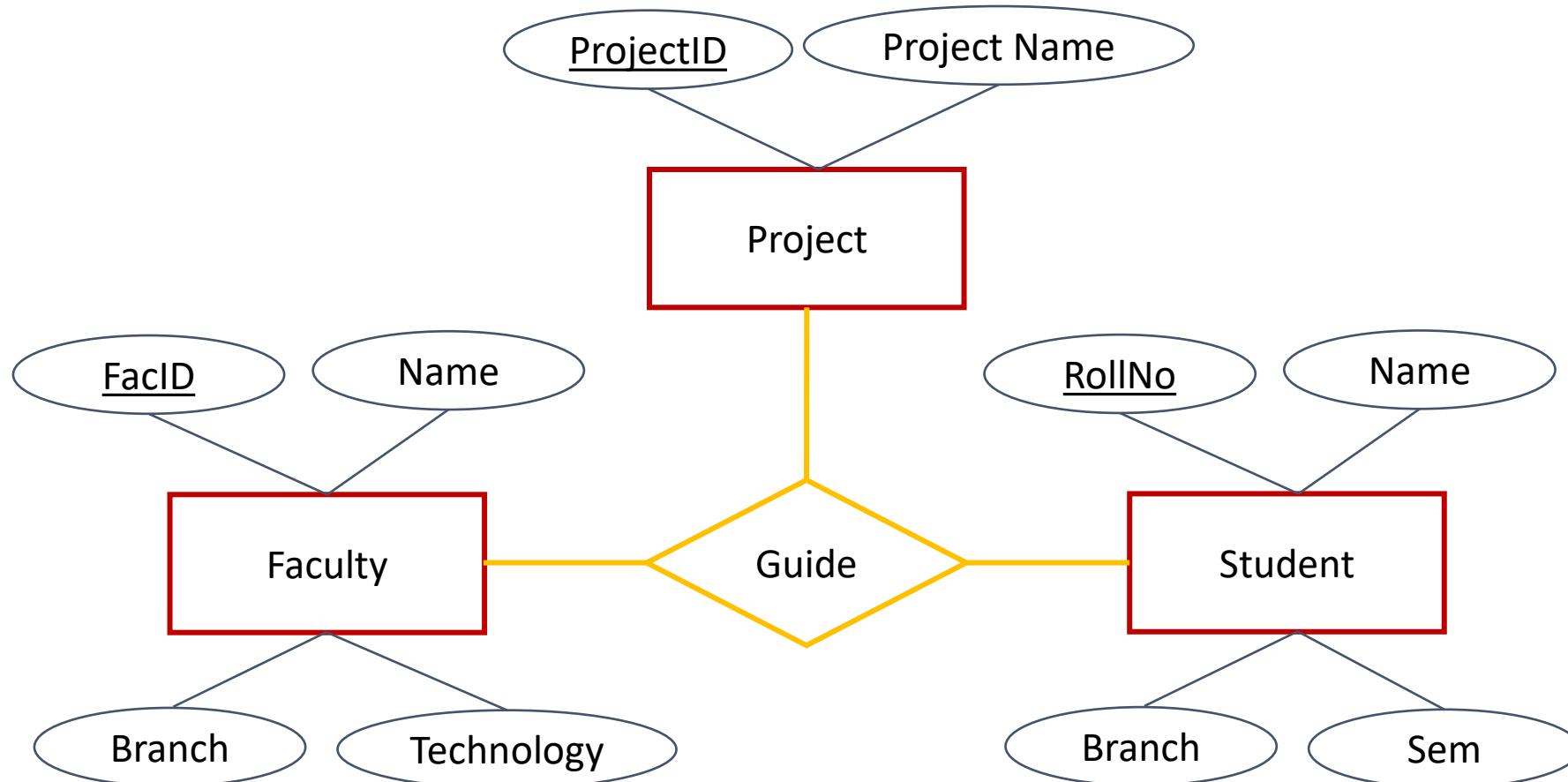
E-R diagram

E-R Diagram of a Library System



E-R diagram

Ternary Relationship



E-R diagram

Types of Attributes

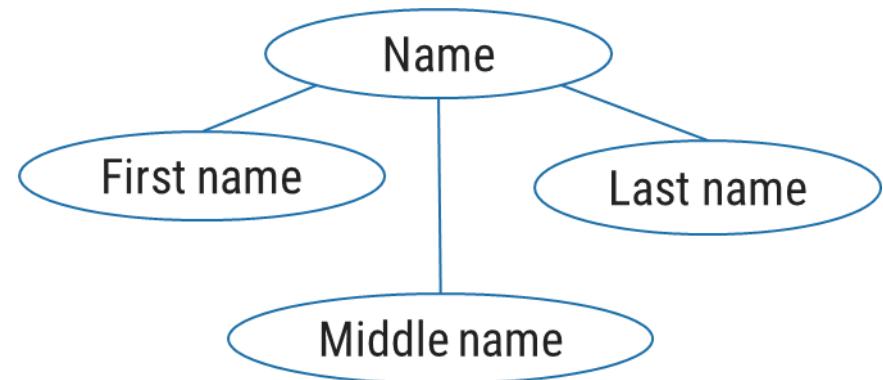
- **Simple Attribute**

- Cannot be divided into subparts
- E.g. RollNo, Age



- **Composite Attribute**

- Can be divided into subparts
- **Name**
 - (first name, middle name, last name)
- **Address**
 - (street, road, city)



E-R diagram

Types of Attributes

- **Single-valued Attribute**

- Has single value
- E.g. RollNo, Age



- **Multi-valued Attribute**

- Has multiple (more than one) value
- **PhoneNo**
- (person may have multiple phone nos)
- **Email_ID**
- (person may have multiple emails)



E-R diagram

Types of Attributes

- **Stored Attribute**

- Its value is stored physically in the database
- E.g. Birthdate



- **Derived Attribute**

- Its value is derived or calculated from other attributes
- Age
- (can be calculated using current date and birthdate)

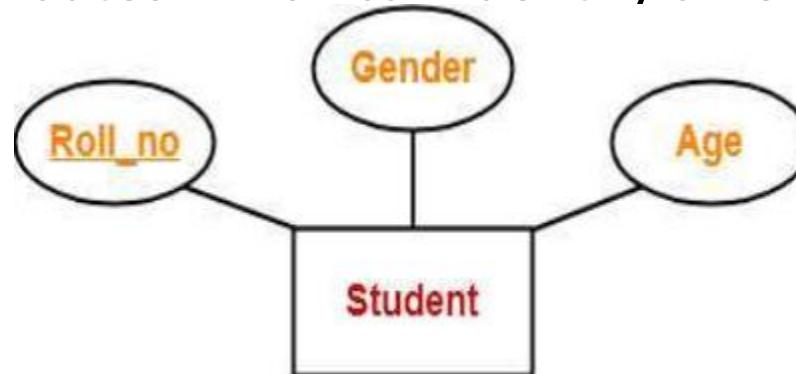


E-R diagram

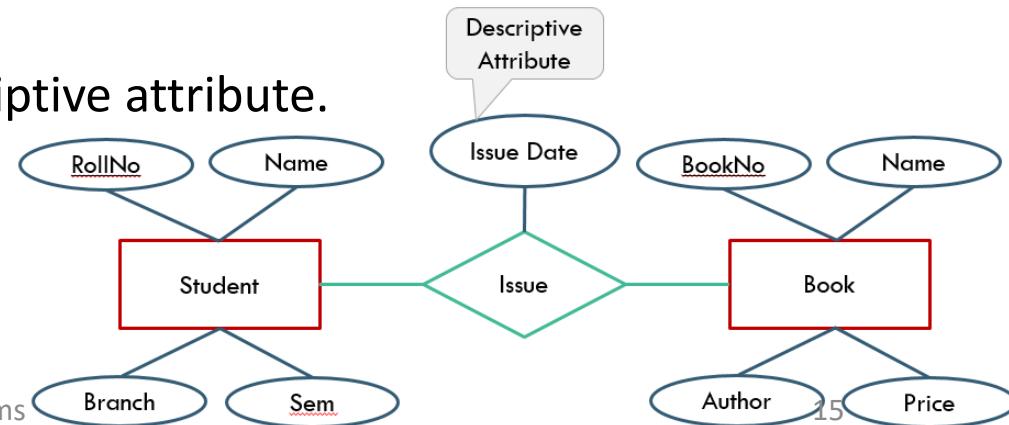
Types of Attributes

- **Key Attribute:**

- Key attributes are those attributes which can identify an entity uniquely in an entity set.
- Example: Roll_no.

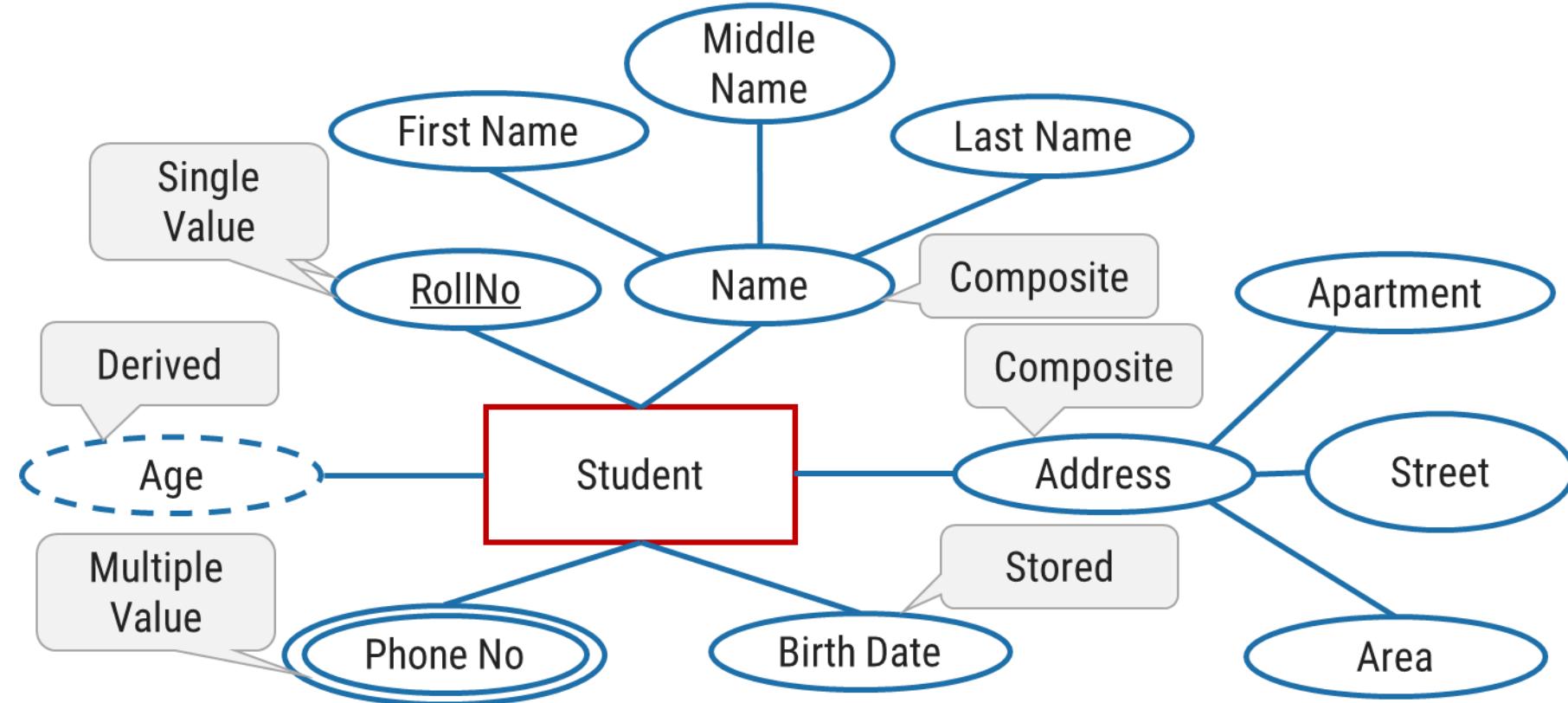


- **Descriptive Attribute:** Attributes of the relationship is called descriptive attribute.



E-R diagram

Entity with all types of Attributes



Exercise

➤ Draw an E-R diagram of following pair of entities

- Customer & Account
- Customer & Loan
- Doctor & Patient
- Student & Project
- Student & Teacher

➤ Note: Take four attributes per entity with one primary key attribute.

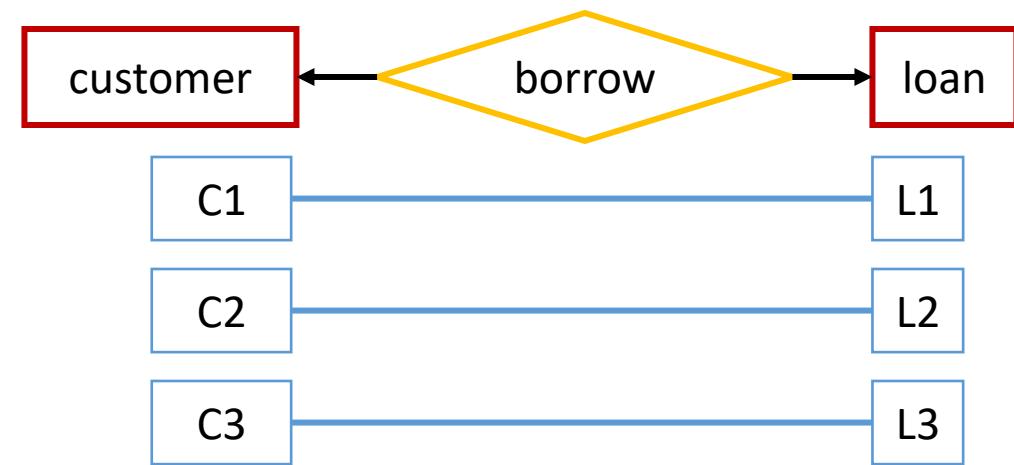
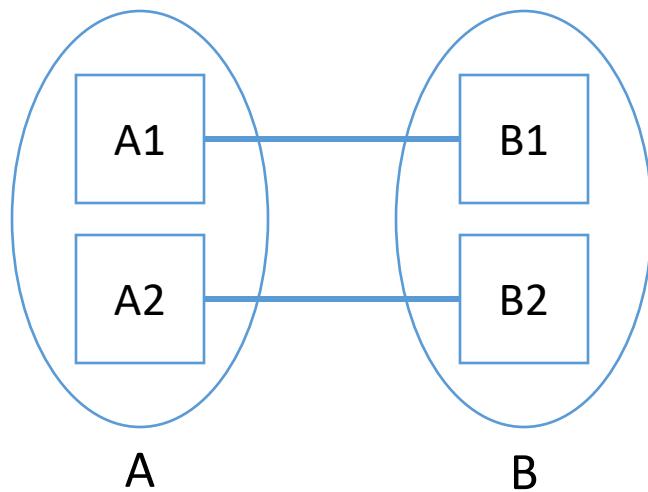
Keep proper relationship between two entities.

Mapping Cardinality (Cardinality Constraints)

- It represents the **number of entities of another entity set** which are **connected to an entity** using a relationship set.
- It is most **useful in describing binary relationship sets**.
- For a binary relationship set the mapping cardinality must be one of the following types:
 - One to One
 - One to Many
 - Many to One
 - Many to Many

One-to-One relationship (1 – 1)

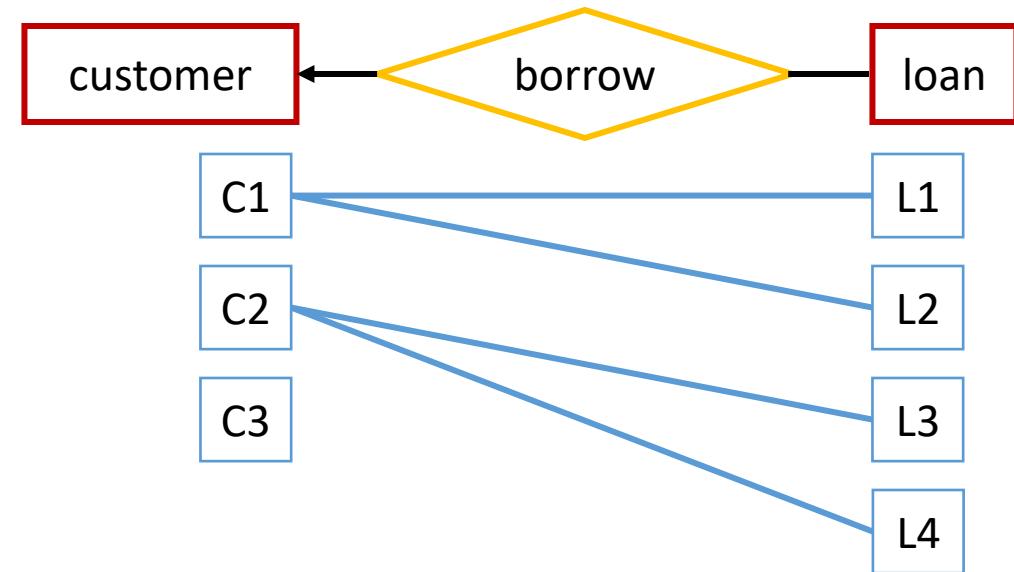
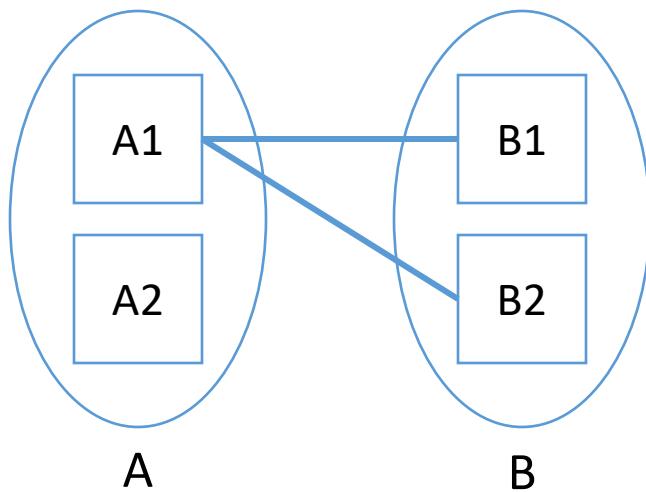
- An entity in A is associated with only one entity in B and an entity in B is associated with only one entity in A.



- Example: A **customer** is connected with **only one loan** using the relationship **borrower** and a **loan** is connected with **only one customer** using **borrower**.

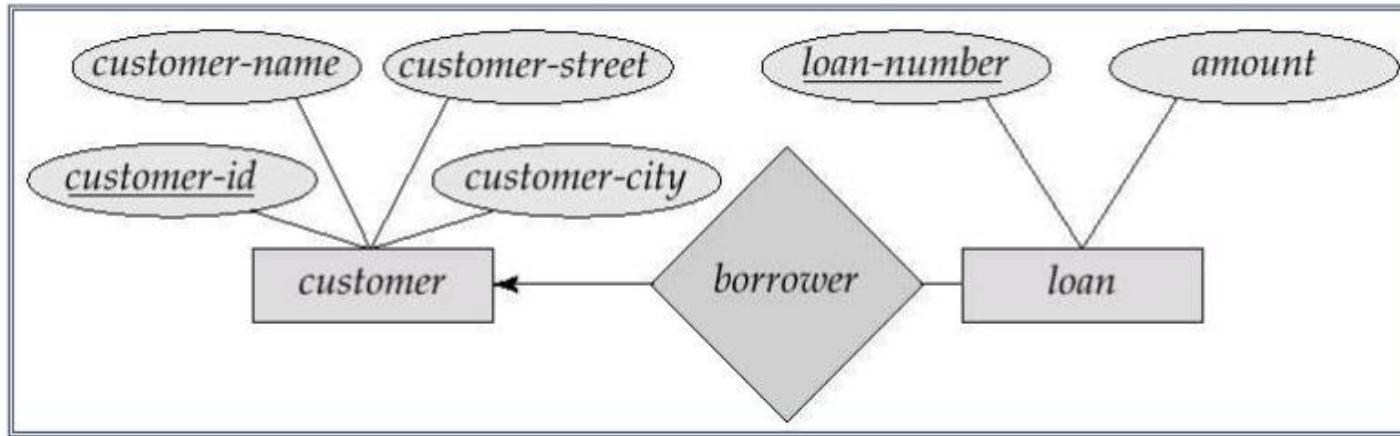
One-to-Many relationship (1 – N)

- An entity in **A** is associated with more than one entities in **B** and an entity in **B** is associated with only one entity in **A**.



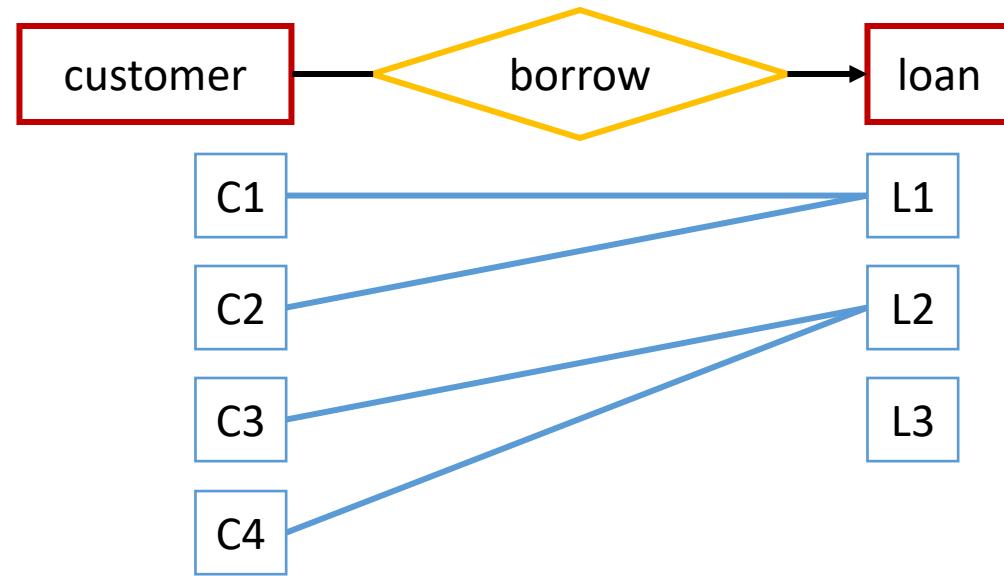
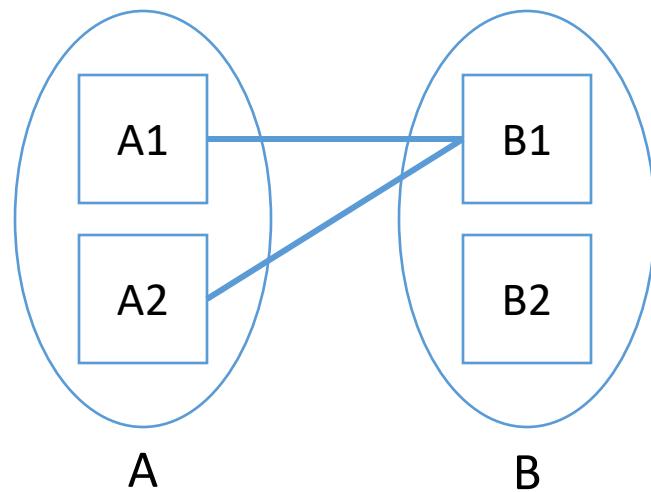
- Example: A **loan** is connected with only one **customer** using **borrower** and a **customer** is connected with more than one **loans** using **borrower**.

One-to-Many relationship (1 – N)



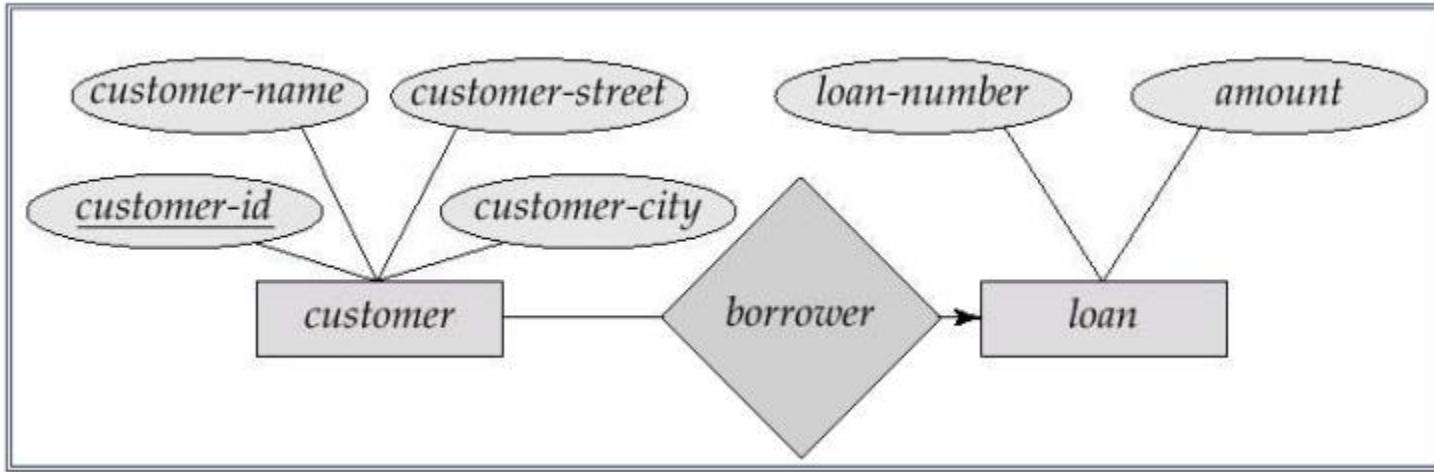
Many-to-One relationship (N – 1)

- An entity in A is associated with only one entity in B and an entity in B is associated with more than one entities in A.



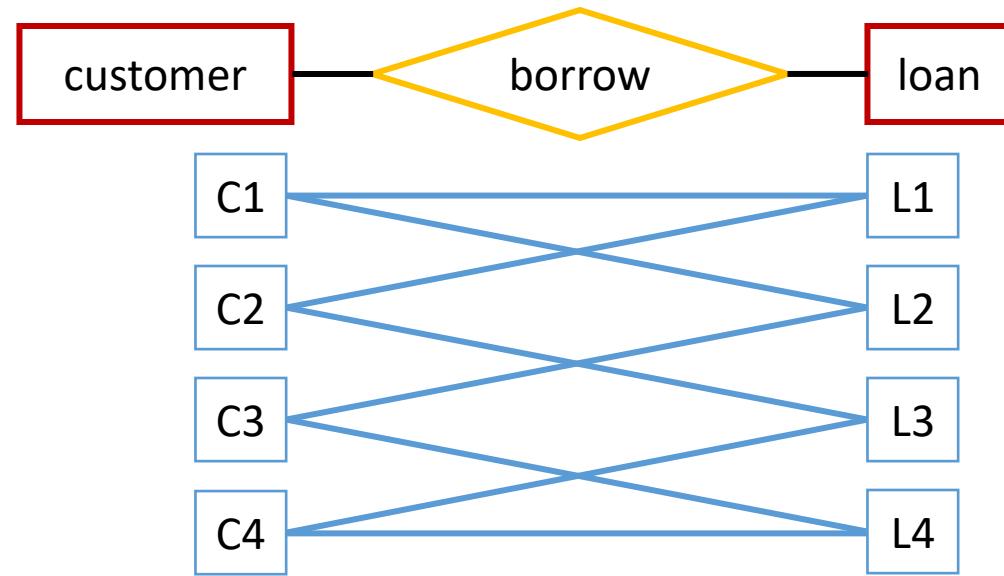
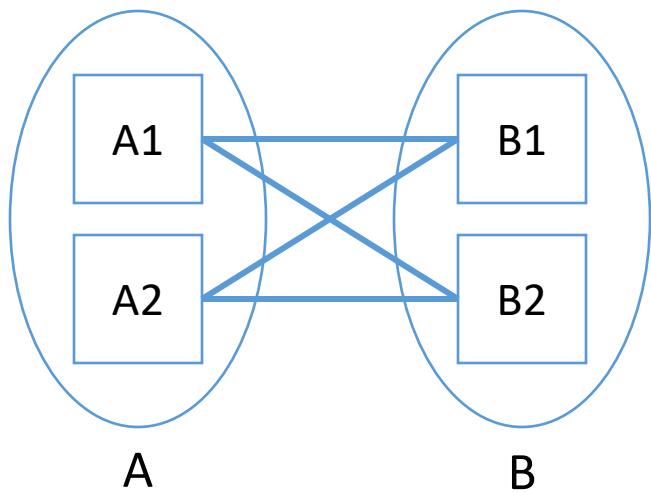
- Example: A loan is connected with more than one customer using borrower and a customer is connected with only one loan using borrower.

Many-to-One relationship (N – 1)



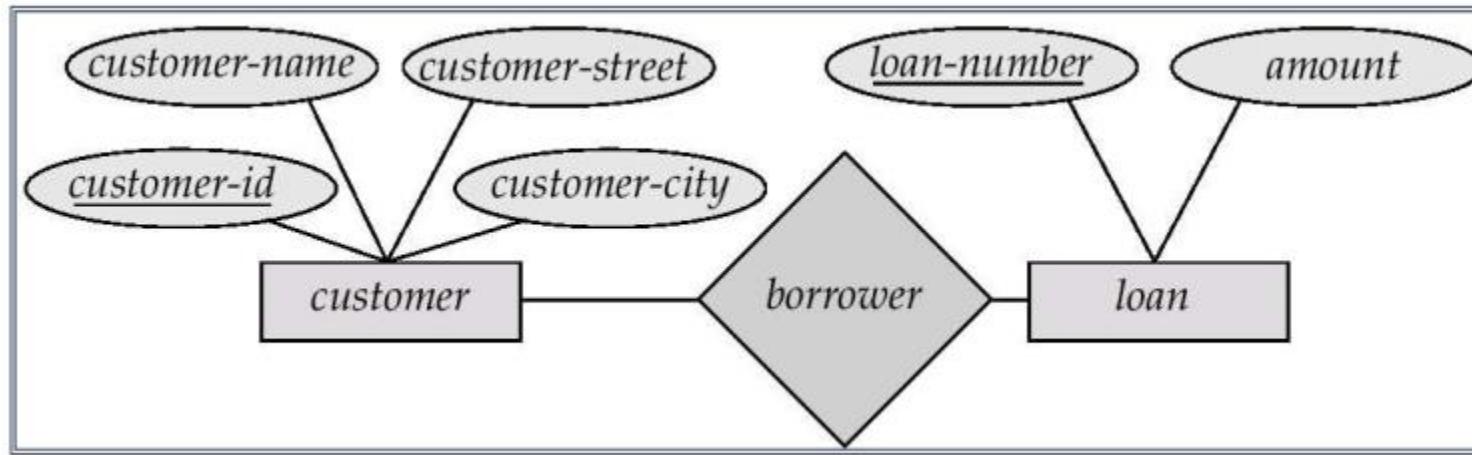
Many-to-Many relationship (N – N)

- An entity in A is associated with more than one entities in B and an entity in B is associated with more than one entities in A.



- Example: A customer is connected with more than one loan using borrower and a loan is connected with more than one customer using borrower.

Many-to-Many relationship (N – N)

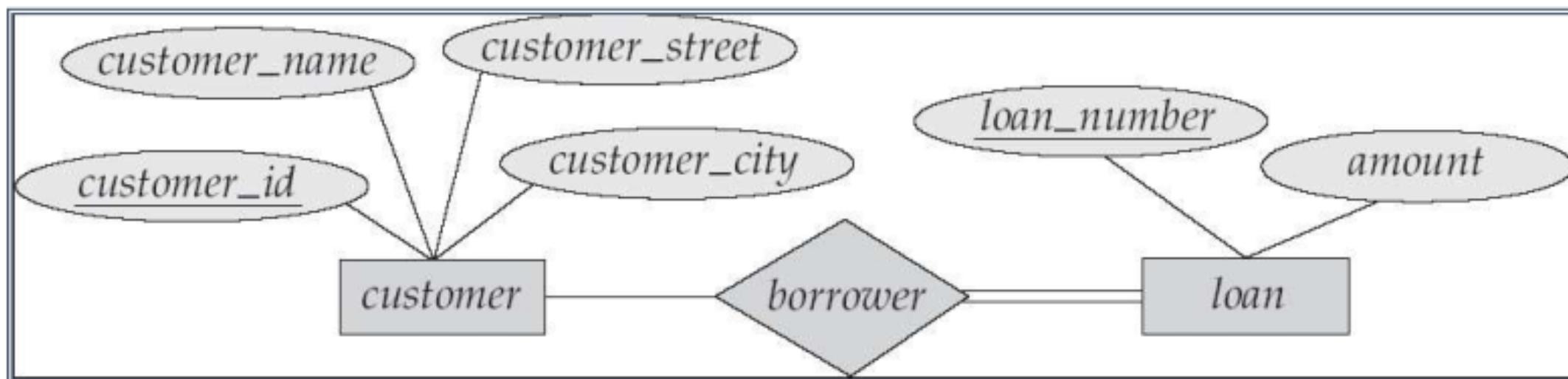


Mapping Cardinality (Cardinality Constraints) [Exercise]

- Draw an E-R diagram and specify which type of mapping cardinality will be there in the following examples:
 - Each customer has only one account in the bank and each account is held by only one customer. [single account]
 - Each customer has only one account in the bank but an account can be held by more than one customer. [joint account]
 - A customer may have more than one account in the bank but each account is held by only one customer. [multiple accounts]
 - A customer may have more than one account in the bank and each account is held by more than one customer. [join account as well as multiple accounts]
 - A student can work in more than one project and a project can be done by more than one student.
 - A student can issue more than one book but a book is issued to only one student.
 - A subject is taught by more than one faculty and a faculty can teach more than one subject.

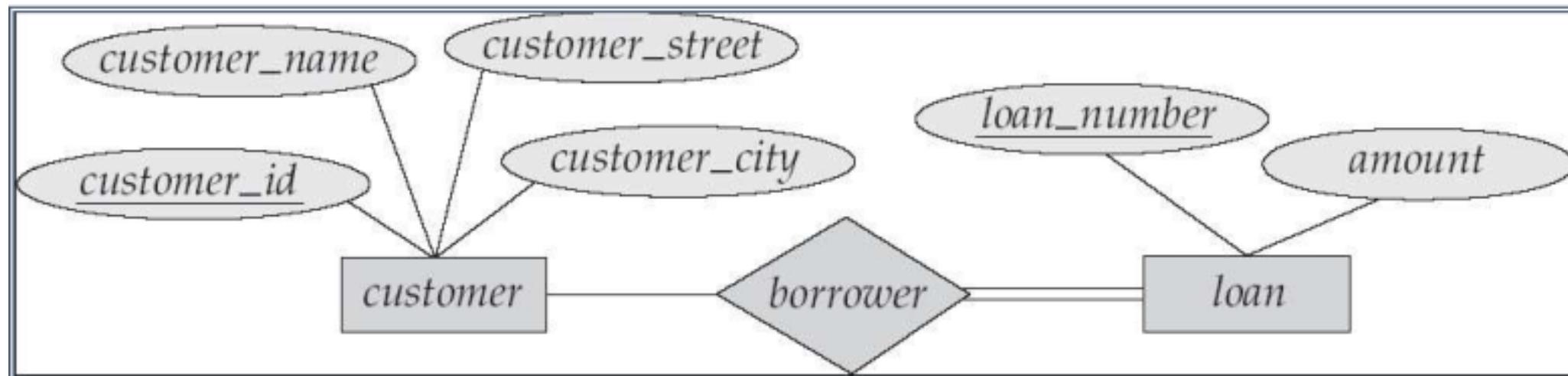
Participation of an Entity Set in a Relationship set

- **Total participation** (indicated by double line): every entity in the entity set participates in at least one relationship in the relationship set
 - E.g. participation of loan in borrower is total
 - ▶ every loan must have a customer associated to it via borrower



Participation of an Entity Set in a Relationship set

- **Partial participation** (indicated by single line): some entities in the entity set may not participate in any relationship in the relationship set
 - Example: participation of customer in borrower is partial

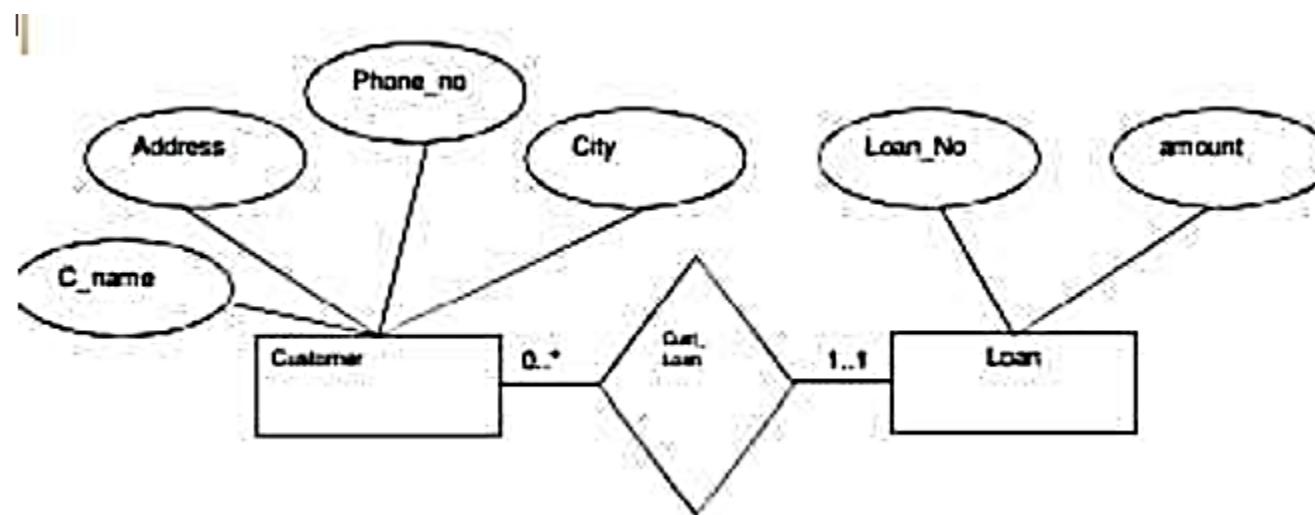


Representation of Cardinality in ER

- The cardinality of a relationship is the actual number of related occurrences for each of the two entities.
- E-R diagrams also provide a way to indicate cardinality of the relationship.
- An **edge** between an entity set and a relationship set can have an associated minimum and maximum cardinality, shown in the form **I..h**, where **I** is the minimum and **h** the maximum cardinality.
- A **minimum value of 1** indicates **total participation** of the entity in the relationship.
- A **maximum value of 1** indicates that the entity participates in at most one relationship, while a **maximum value *** indicates no limit.
- The label **1..*** or **1..1** on an edge is equivalent to a double line.

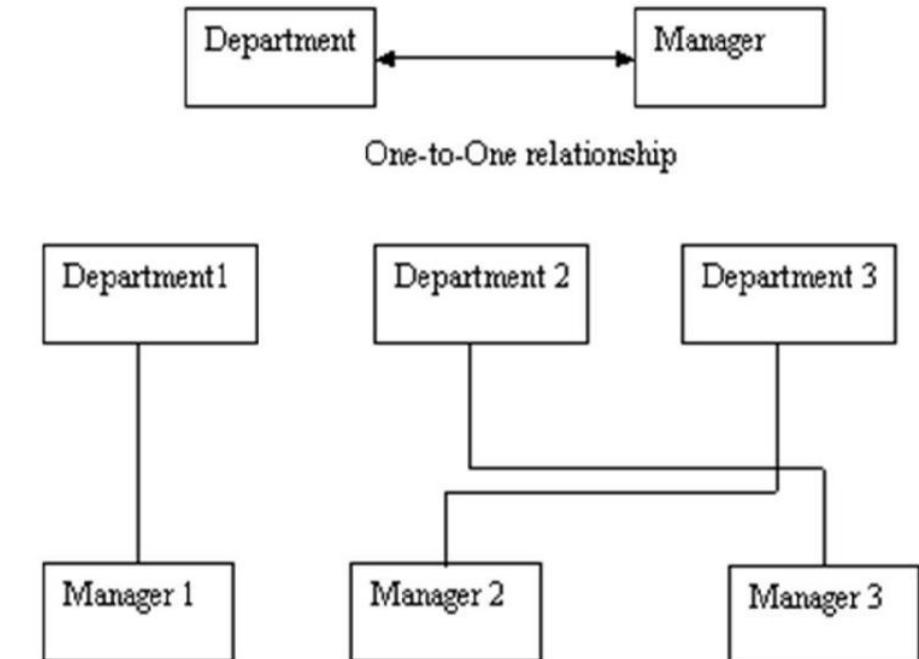
Representation of Cardinality in ER

- The edge between **Loan** and **Cust_Loan** has a cardinality constraint of **1..1**, meaning the minimum and the maximum cardinality are both 1.
- That is, each loan must have exactly one associated customer.
- The limit **0..*** on the edge from **Customer** to **Cust_Loan** indicates that a customer can have zero or more loans.
- Thus, the relationship **Cust_Loan** is **one to many from customer to loan**, and further the **participation of loan in Cust_Loan is total**.

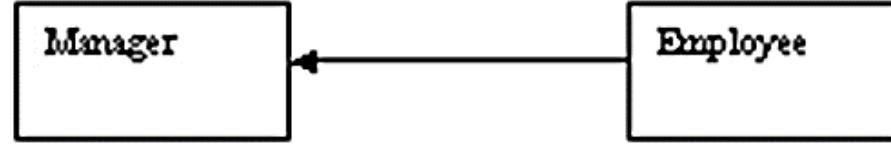


Direction Symbol

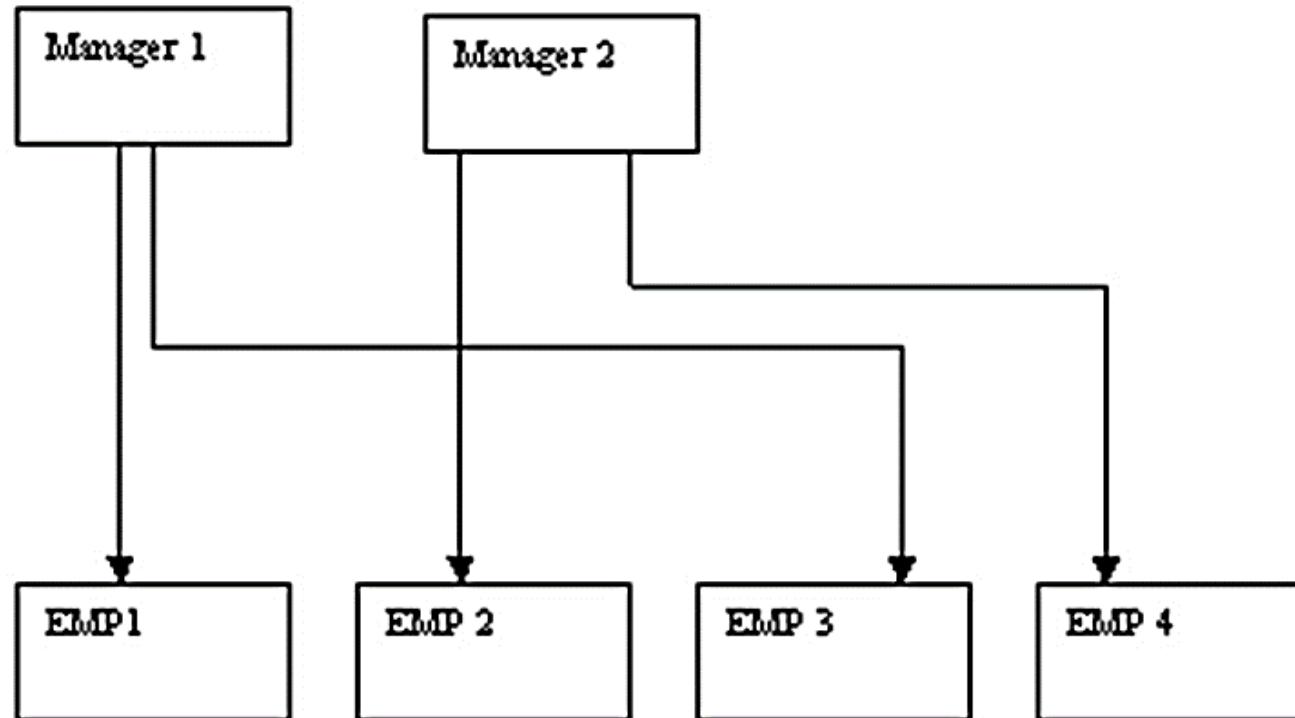
- The direction of a relationship indicates the originating entity of a relationship.
- The entity from which a relationship originates is the parent entity; the entity where the relationship terminates is the child entity.
- The type of the relation is determined by the direction of line connecting relationship component and the entity.
- To distinguish different types of relation, we draw either a directed line or an undirected line between the relationship set and the entity set.
- Directed line is used to indicate one occurrence and undirected line is used to indicate many occurrences in a relation as shown in next case.



Direction Symbol

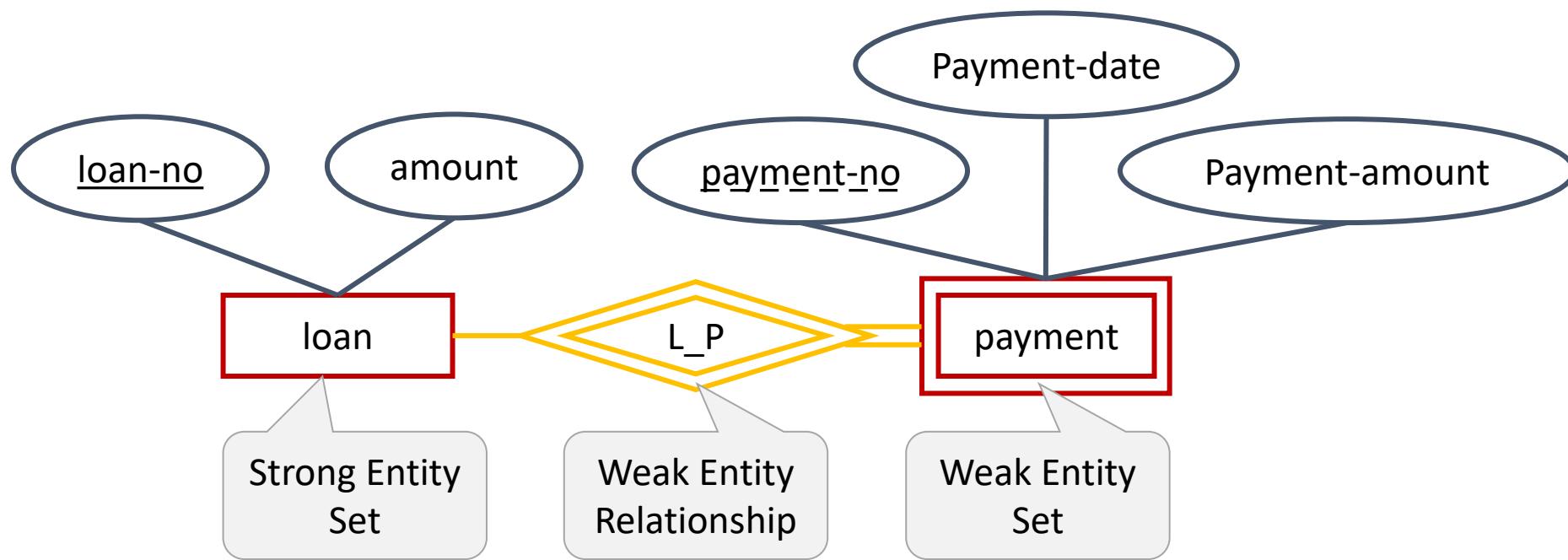


One-to-Many relationship



Weak Entity Set

- The entity set which does not have sufficient attributes to form a primary key is called as **weak entity set**. An entity set that has a primary key is called as **Strong entity set**.



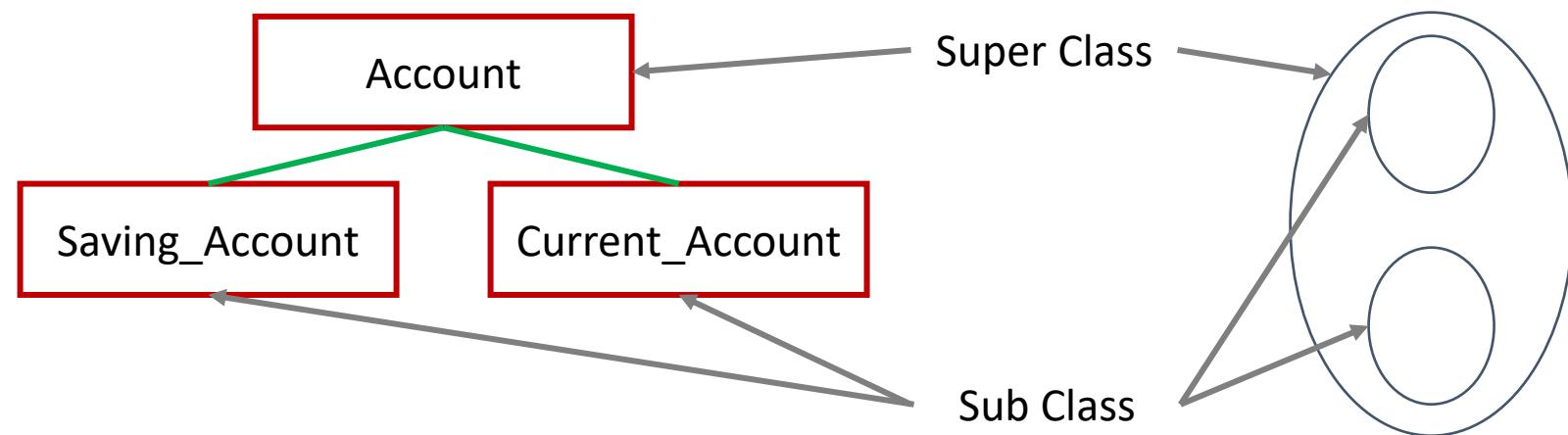
- Weak entity set is indicated by double rectangle.
- Weak entity relationship set is indicated by double diamond.

Weak Entity Set

- A member of a strong entity set is called dominant entity and member of weak entity set is called as subordinate entity.
- A weak entity set does not have a primary key but we need a means of distinguishing among all those entries in the entity set that depend on one particular strong entity set.
- i.e. the **existence of a weak entity set depends on the existence of a strong entity set.**
- The **primary key** of a weak entity set is created by **combining the primary key of the strong entity set** on which the weak entity set existence depends and **the weak entity set's discriminator**.
- For example, payment_no is acts as discriminator for payment entity set. It is also called as the Partial key of the entity set.

Superclass and Subclass

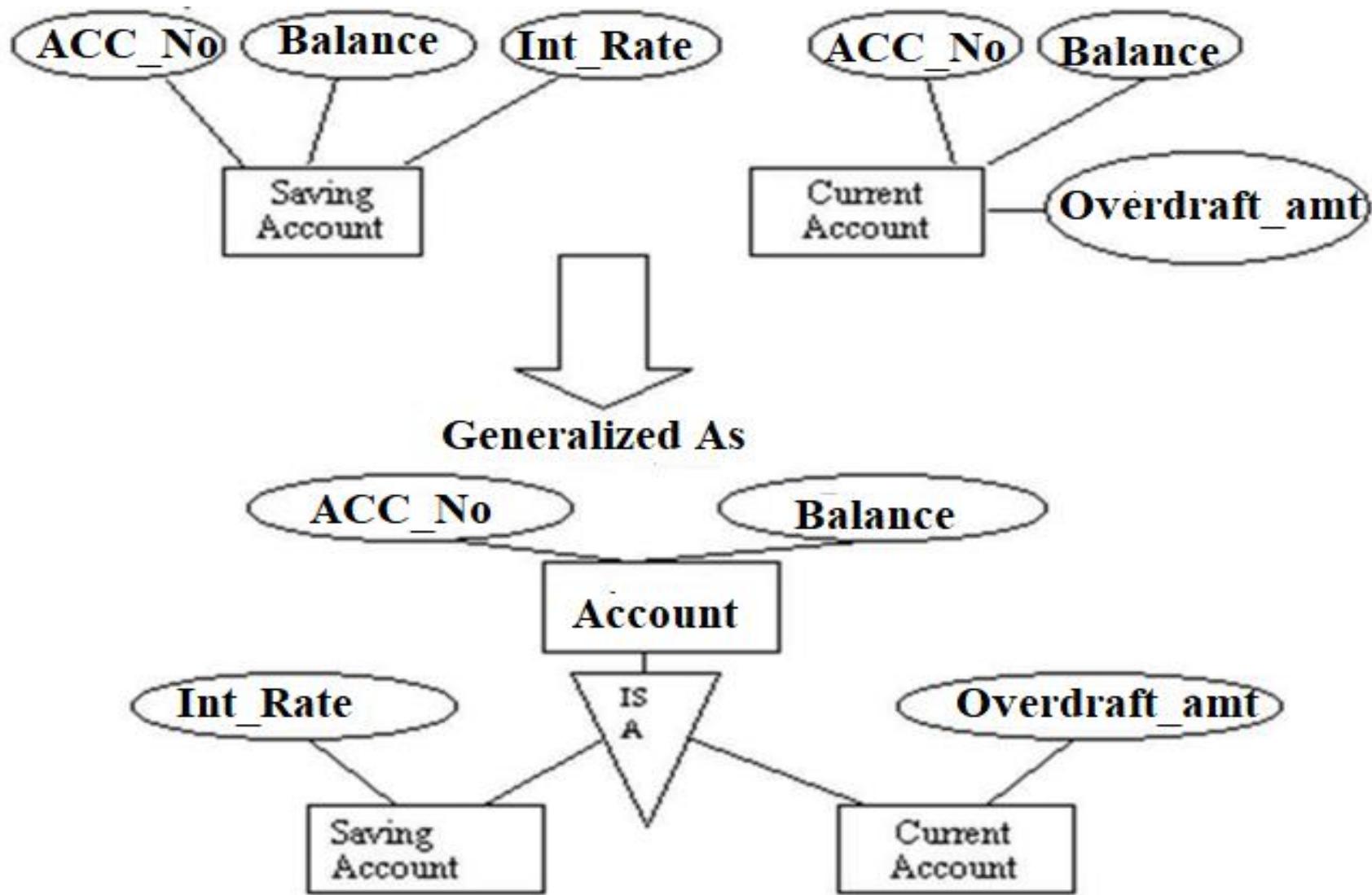
Super Class	Sub Class
A superclass is an entity from which another entities can be derived.	A subclass is an entity that is derived from another entity.
E.g, an entity account has two subsets saving_account and current_account So an account is superclass.	E.g, saving_account and current_account entities are derived from entity account. So saving_account and current_account are subclass.



Generalization

- A generalization hierarchy is a form of abstraction that specifies that two or more entities that share common attributes can be generalized into a higher-level entity type called a super type or generic entity.
- The lower level of entities becomes the subtype, or categories, to the super type. Subtypes are dependent entities.
- Generalization is used to emphasize the similarities among lower-level entity sets and to hide differences.
- It makes ER diagram simpler because shared attributes are not repeated.
- Generalization is denoted through a triangle component labeled ‘IS_A’,
- Bottom Up Approach.

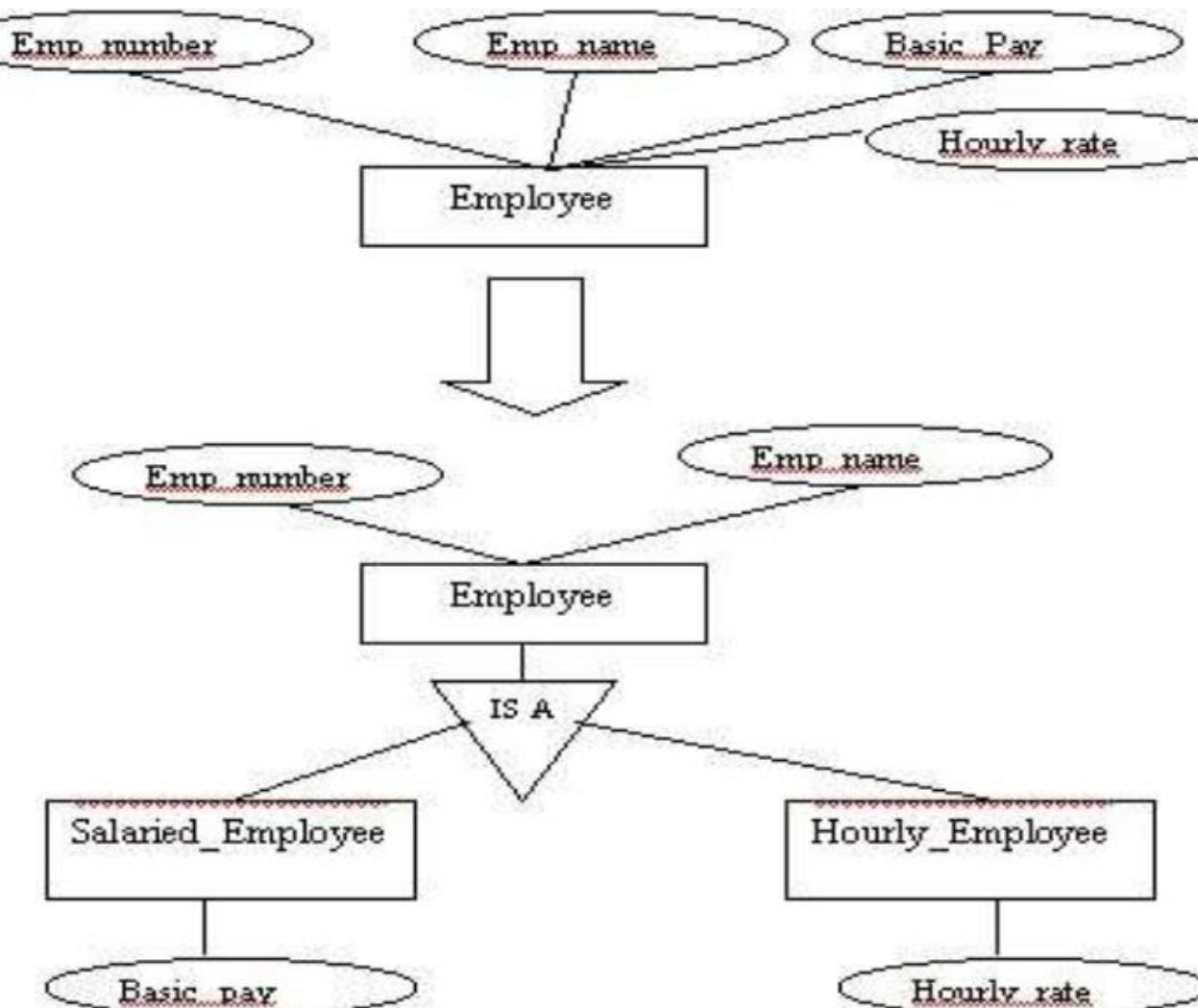
Generalization



Specialization

- Specialization is the process of taking subsets of a higher-level entity set to form lower level entity sets.
- It is a process of defining a set of subclasses of an entity type, which is called as super class of the specialization.
- The process of defining subclass is based on the basis of some distinguish characteristics of the entities in the super class.
- Top Down Approach
- For example,
 - specialization of the Employee entity type may yield the set of subclasses namely Salaried_Employee and Hourly_Employee on the method of pay

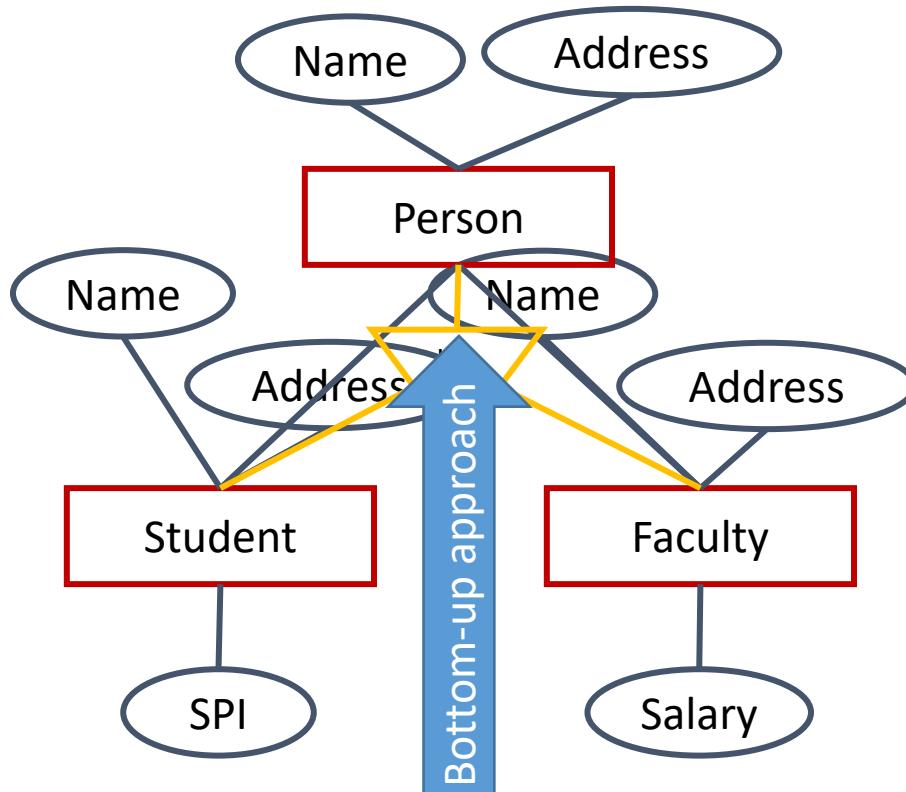
Specialization



Generalization and Specialization

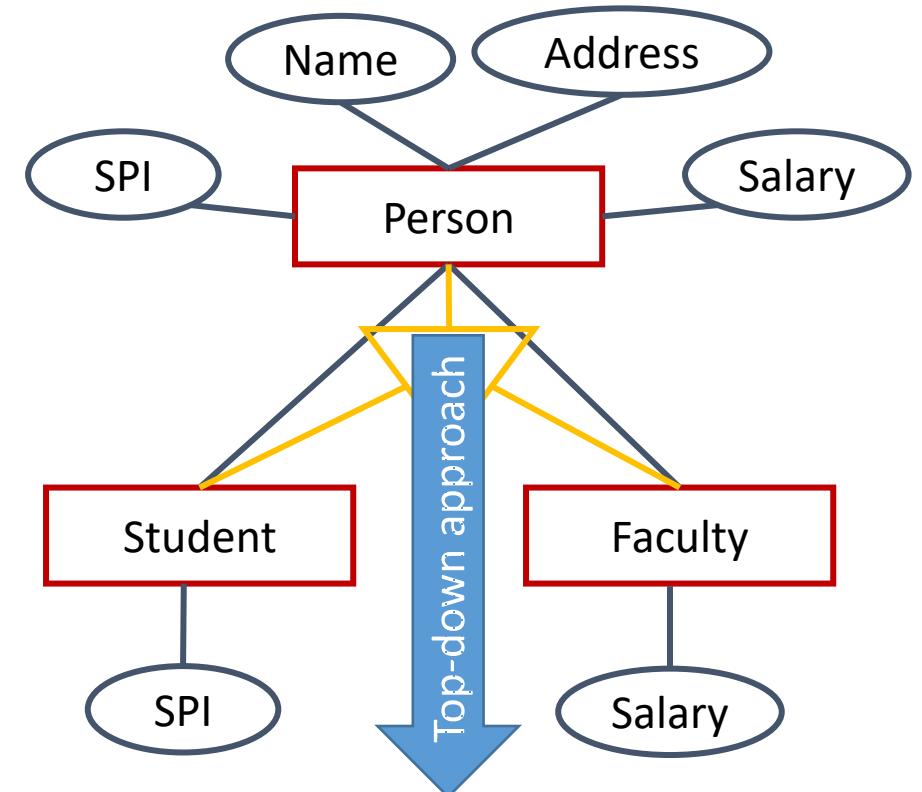
Generalization

It extracts the common features of multiple entities to form a new entity.



Specialization

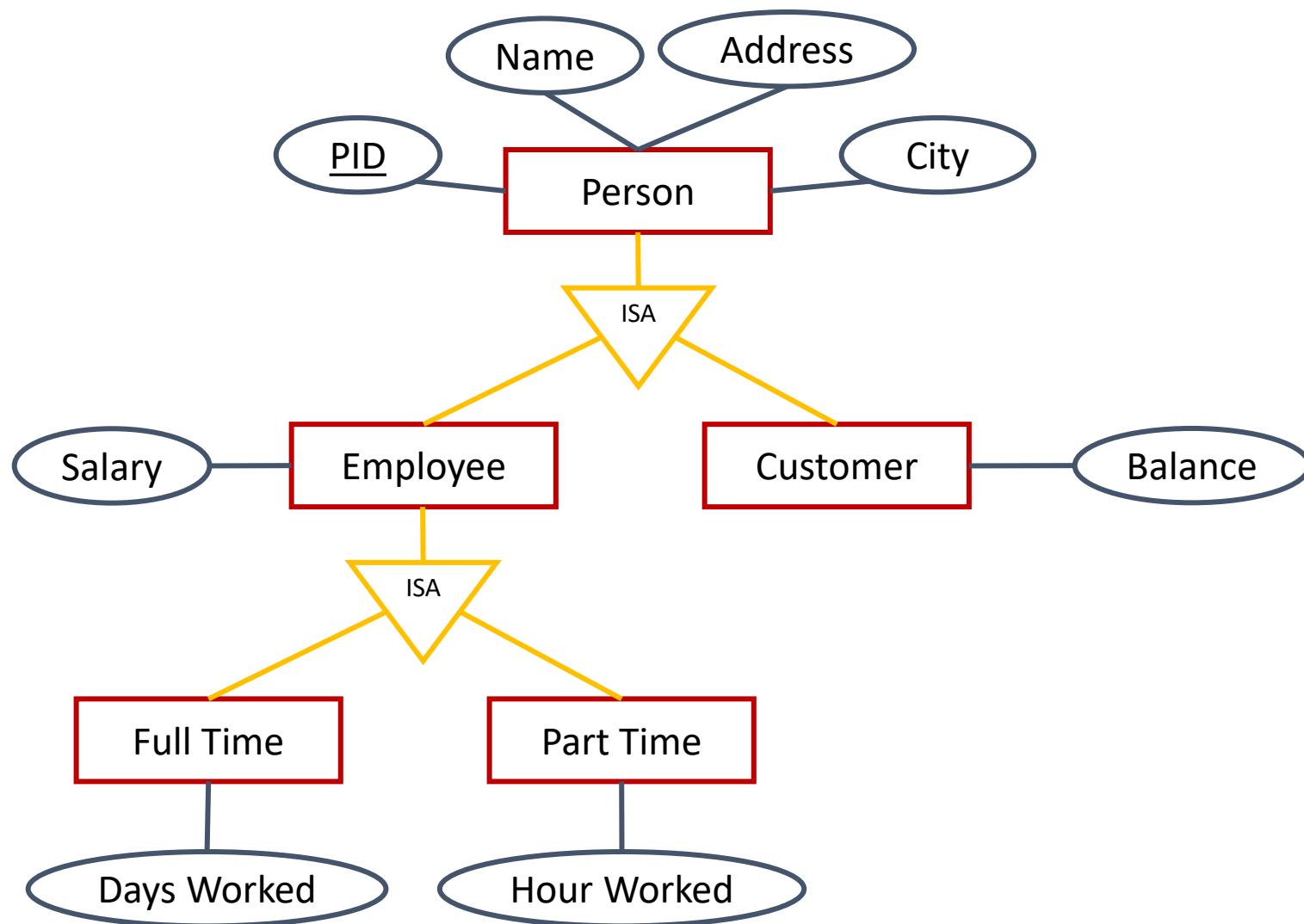
It splits an entity to form multiple new entities that inherit some feature of the splitting entity.



Generalization v/s Specialization

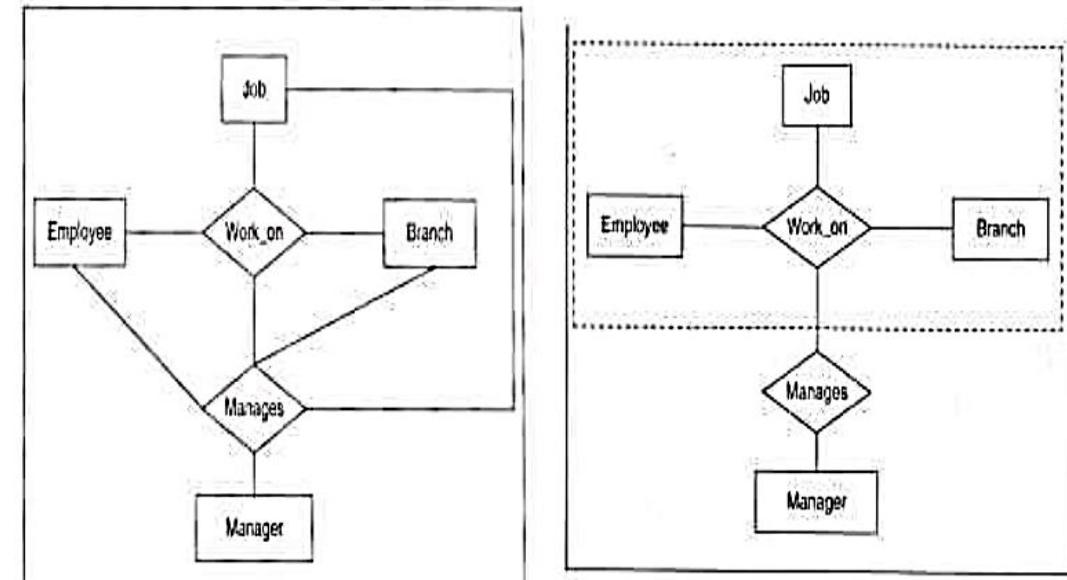
Generalization	Specialization
The process of creation of group from various entities is called generalization.	The process of creation of sub-groups within an entity is called specialization.
It is Bottom-up approach.	It is Top-down approach.
The process of taking the union of two or more lower level entity sets to produce a higher level entity set.	The process of taking a sub set of higher level entity set to form a lower level entity set.
It starts from the number of entity sets and creates high level entity set using some common features.	It starts from a single entity set and creates different low level entity sets using some different features.

Generalization & Specialization example



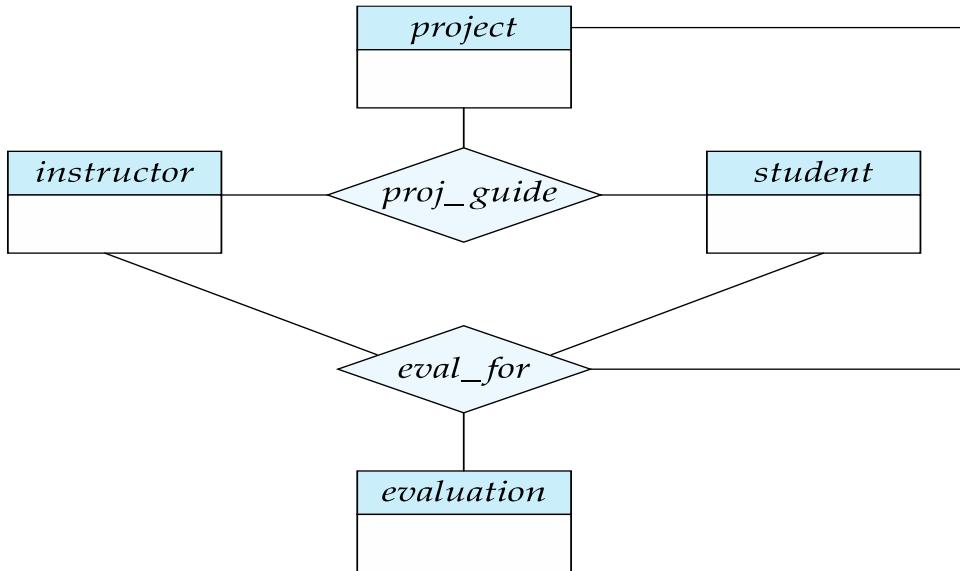
Aggregation

- One limitation of the E-R model is that it cannot express **relationships among relationships**.
- The best way to model a situation like this is by the use of **aggregation**.
- Thus, the relationship set **work_on** relating the entity sets **Employee**, **Branch** and **Job** is a higher-level entity set.
- Such an entity set is treated in the same manner, as is any other entity set.
- We can then create a binary relationship **Manages** between **work_on** and **Manager** to represent who manages what tasks.



Aggregation: Example 2

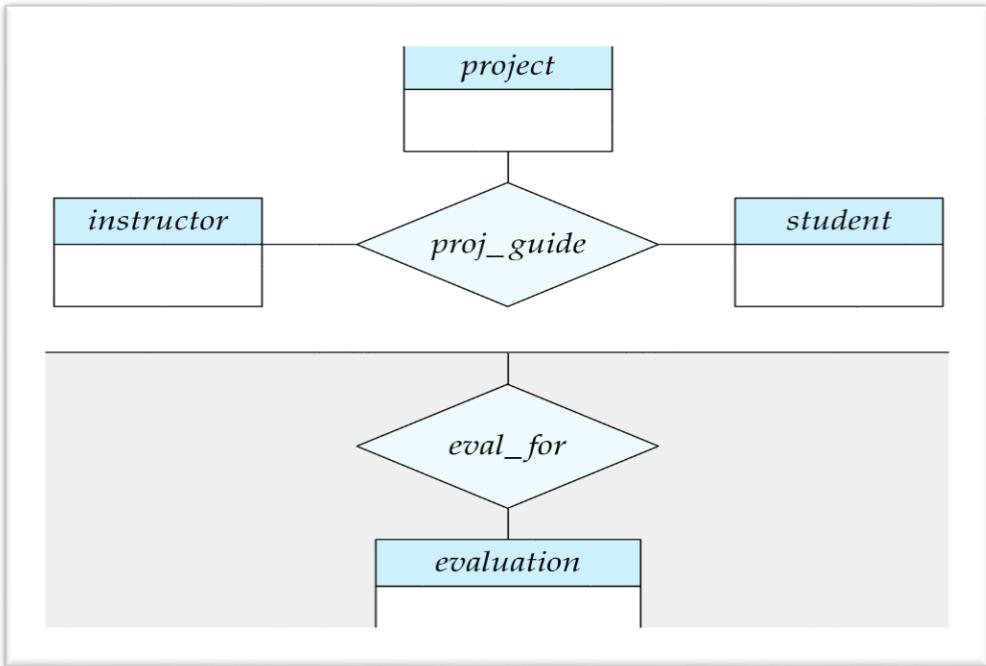
- Consider the ternary relationship ***proj_guide***, which we saw earlier
- Suppose we want to record **evaluations** of a **student** by a **guide** on a **project**



- Relationship sets ***eval_for*** and ***proj_guide*** represent overlapping information
 - Every ***eval_for*** relationship corresponds to a ***proj_guide*** relationship
 - However, some ***proj_guide*** relationships may not correspond to any ***eval_for*** relationships
 - So we can't discard the ***proj_guide*** relationship
- Eliminate this redundancy via aggregation**
 - Treat relationship as an abstract entity
 - Allows relationships between relationships
 - Abstraction of relationship into new entity

Aggregation: Example 2

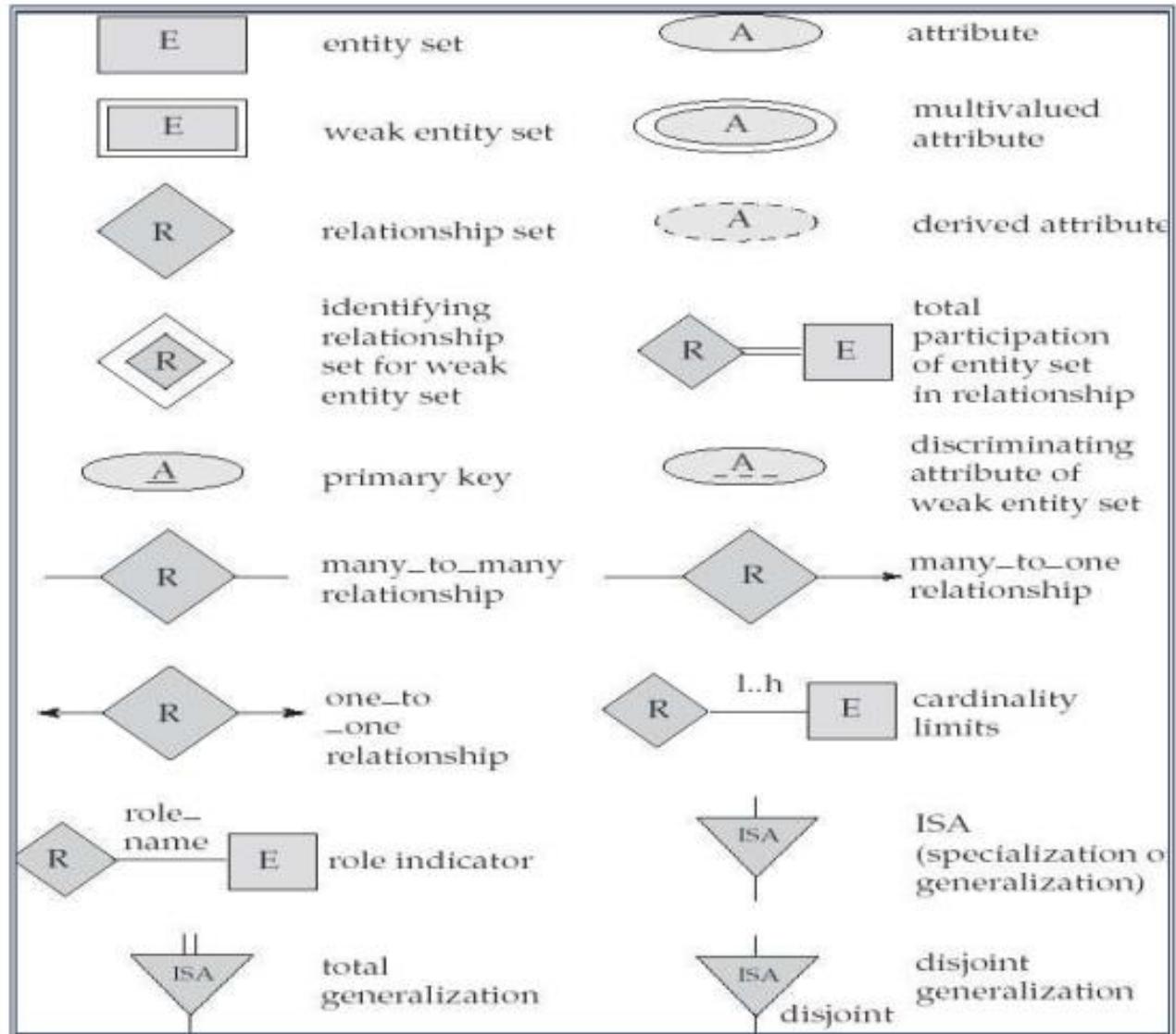
- Eliminate this redundancy via *aggregation* without introducing redundancy, the following diagram represents:
 - A student is guided by a particular instructor on a particular project
 - A student, instructor, project combination may have an associated evaluation



Reduction to Relational Schemas

- To represent aggregation, create a schema containing
 - Primary key of the aggregated relationship,
 - The primary key of the associated entity set
 - Any descriptive attributes
- In our example:
 - The schema *eval_for* is:
 $\text{eval_for} (s_ID, \text{project_id}, i_ID, \text{evaluation_id})$
 - The schema *proj_guide* is redundant.

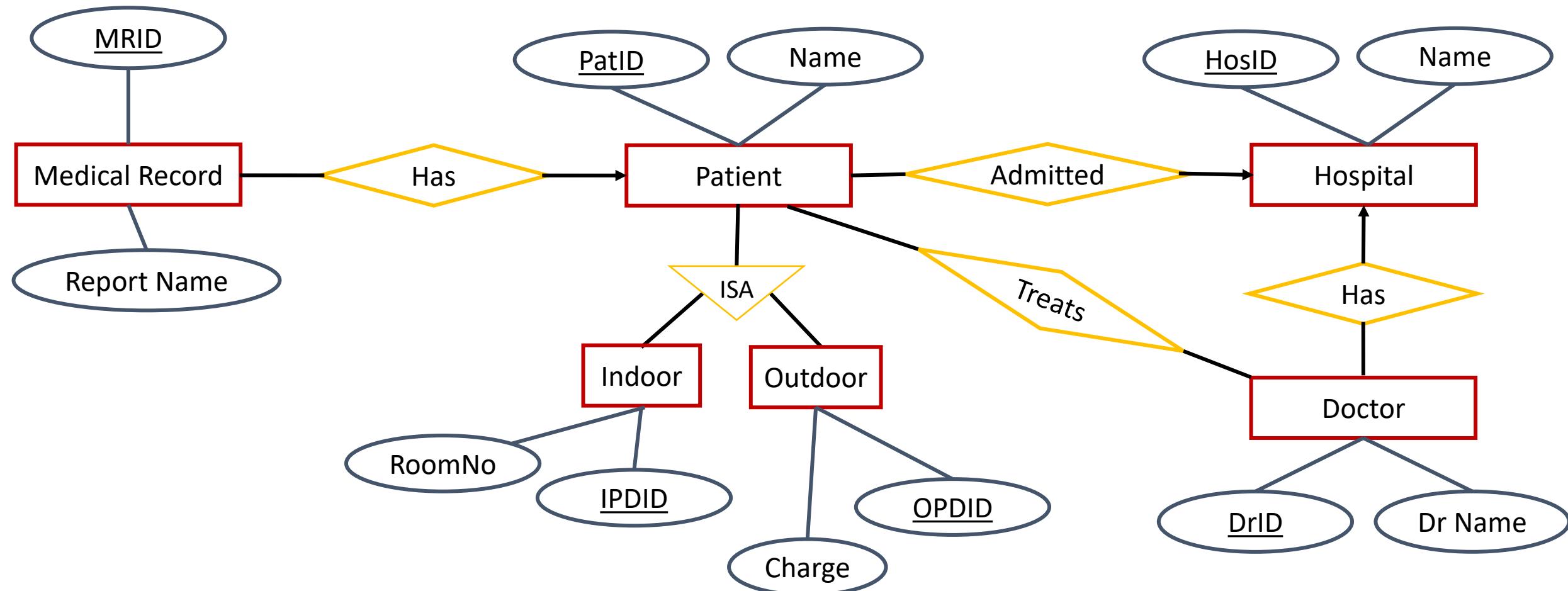
E-R Notation



Steps to design E-R diagram

1. Identify the Entities.
2. Find relationships among these entities.
3. Identify the key attributes for every Entity.
4. Identify other relevant attributes
5. Draw the complete e-r diagram with all attributes including the primary key

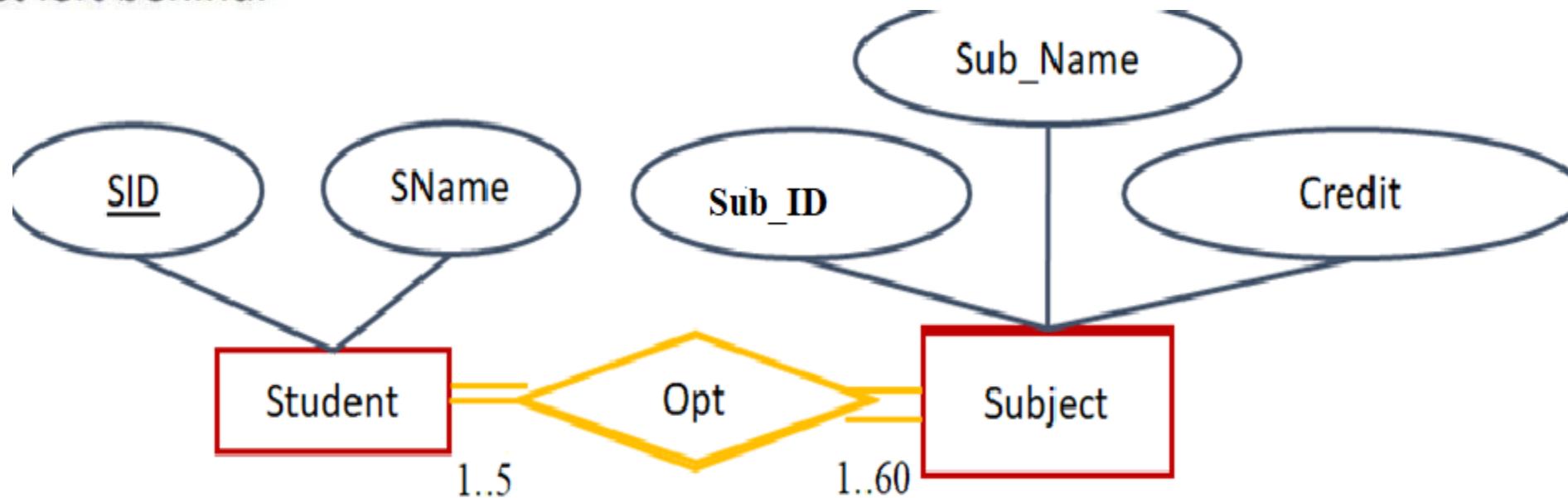
E-R diagram of Hospital Management System



Question: Draw ER Diagram

One student can take max of 5 subjects where as one subject can be taken by max of 60 students .

Student must take subject and there should be no subject left behind.



Case Study 1: University Management System

Consider, a **university** contains many **departments**. Each department can offer any number of **courses**. Many **teachers** can work in a department. A teacher can work only in one department. For each department there is a Head. A teacher can be head of only one department. Each teacher can take any number of courses. A **student** can enroll for any number of courses. Each course can have any number of students.

Step1: Identify the Entities as

University,
Department,
Course,
Teacher,
Student

Note: Consider University as single instance

Step 3: Key Attributes:

D_no,
C_code,
Roll_no,
t_code

Step 2: Relationship:

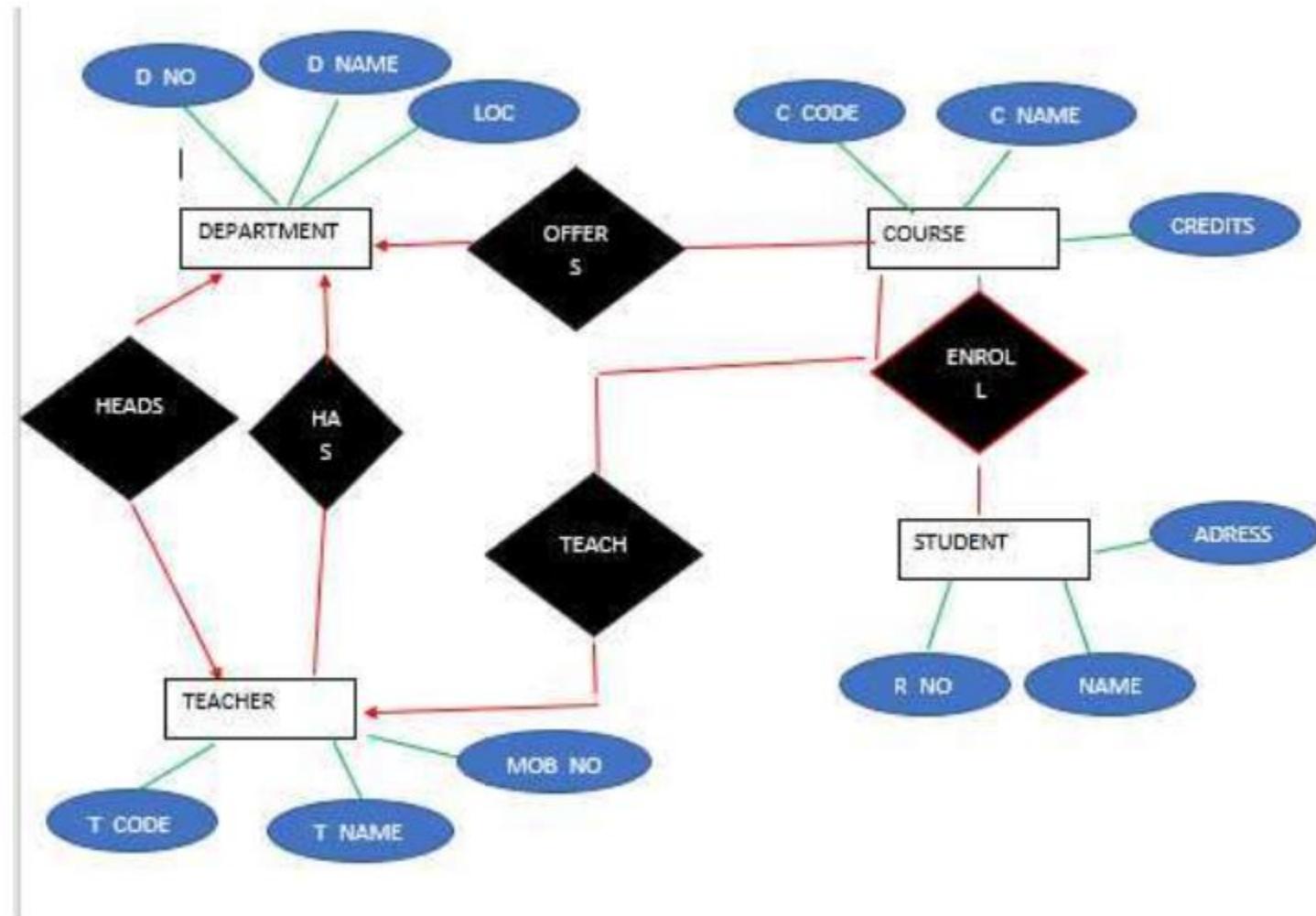
- Department and course (1:M)
- Department & teacher(1:M)
- Department Head and teacher(1:1)
- Teacher and course(1:M)
- Student & course(M:M)

Step 4: Relevant Attributes:

d_name, loc, c_name, credits, t_name, mob_no, name, address etc.

Case Study 1: University Management System, Cont'd...

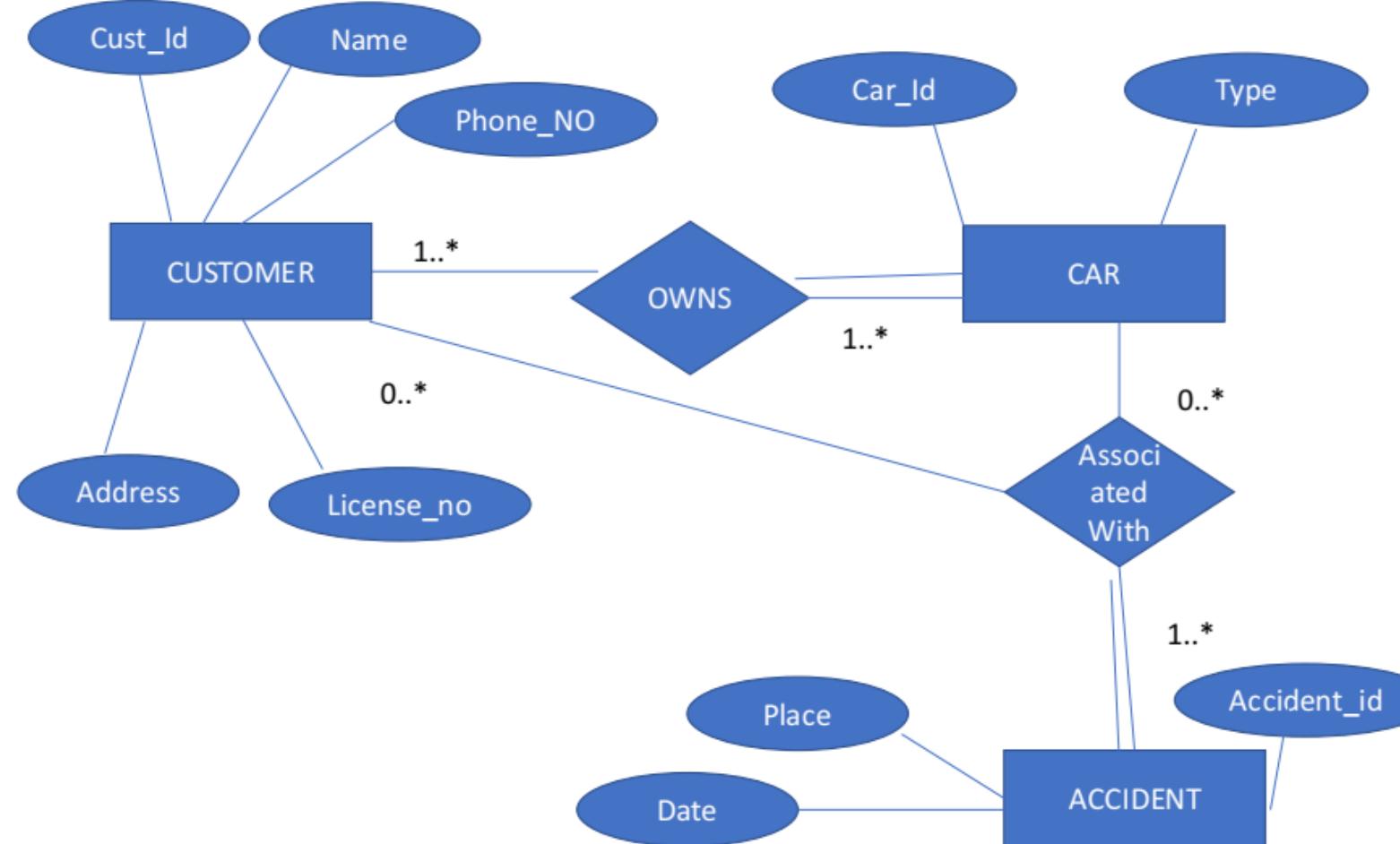
Solution:



Case Study 2: Car Insurance System

- Construct an ER diagram for a car insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.

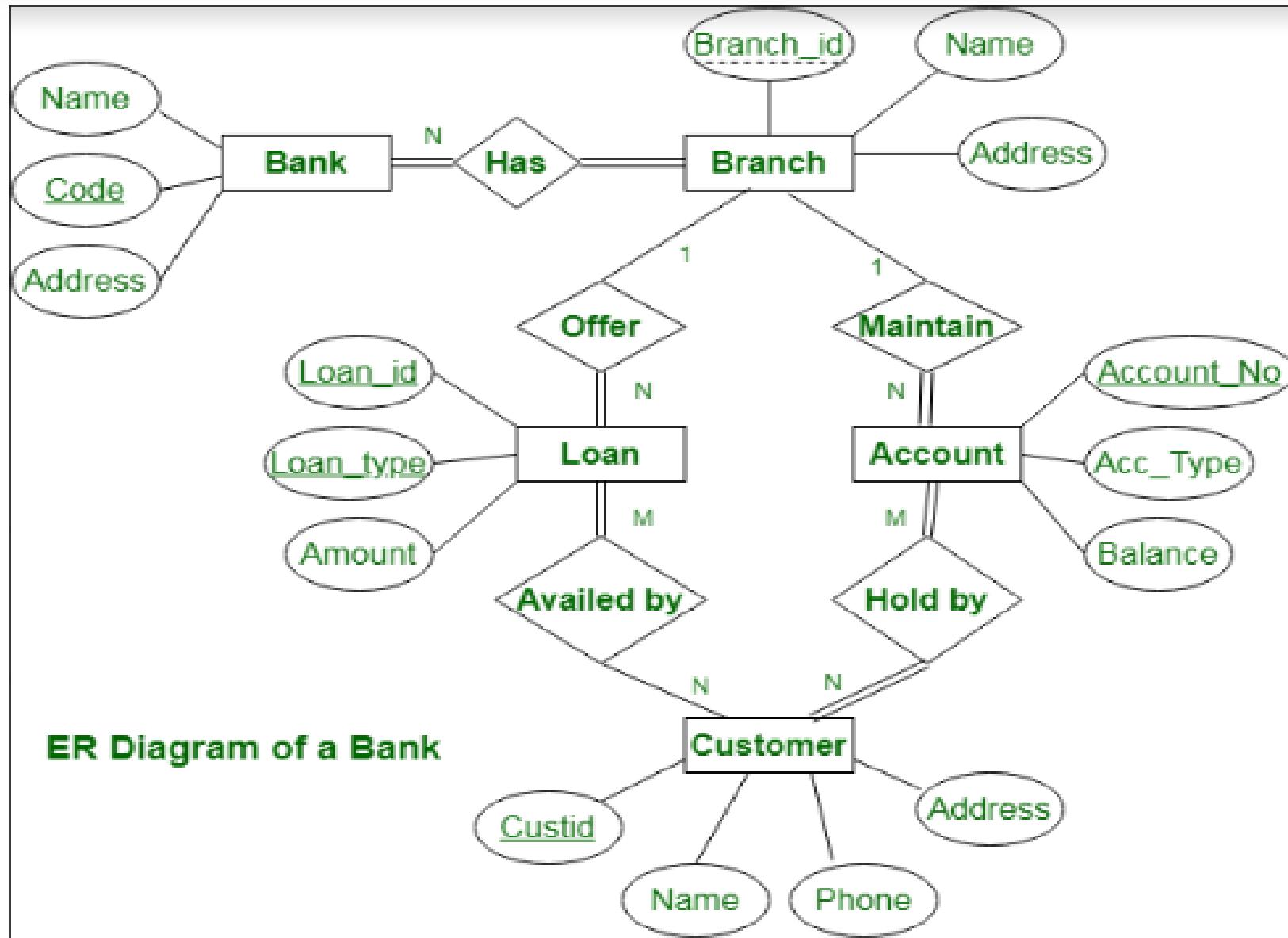
Solution:



Case Study 3: Banking Management System

- Banks are identified by a name, code, address of main office.
- Banks have branches.
- Branches are identified by a branch_no., branch_name, address.
- Customers are identified by name, cust-id, phone number, address.
- Customer can have one or more accounts.
- Accounts are identified by account_no., acc_type, balance.
- Customer can avail loans.
- Loans are identified by loan_id, loan_type and amount.
- Account and loans are related to bank's branch.

Banking Management System, Cont'd...



Case Study 4: The Library Management System

- The system keeps track of the staff with a single point authentication system comprising
 - login Id
 - password
- Staff maintains the book catalog with its ISBN, Book title, price(in INR), category(novel, general, story), edition, author Number and details.
- A publisher has publisher Id, Year when the book was published, and name of the book.
- Readers are registered with their user_id, email, name (first name, last name), Phone no (multiple entries allowed), communication address.
- The staff keeps track of readers.
- Readers can return/reserve books that stamps with issue date and return date. If not returned within the prescribed time period, it may have a due date too.
- Staff also generate reports that has readers id, registration no of report, book no and return/issue info.

The Library Management System, Cont'd...

- **Entities and their Attributes –**

- **Book Entity :**

- It has authno, isbn number, title, edition, category, price.
- ISBN is the Primary Key for Book Entity.

- **Reader Entity :**

- It has UserId, Email, address, phone no, name.
- Name is composite attribute of firstname and lastname.
- Phone no is multi valued attribute.
- UserId is the Primary Key for Readers entity.

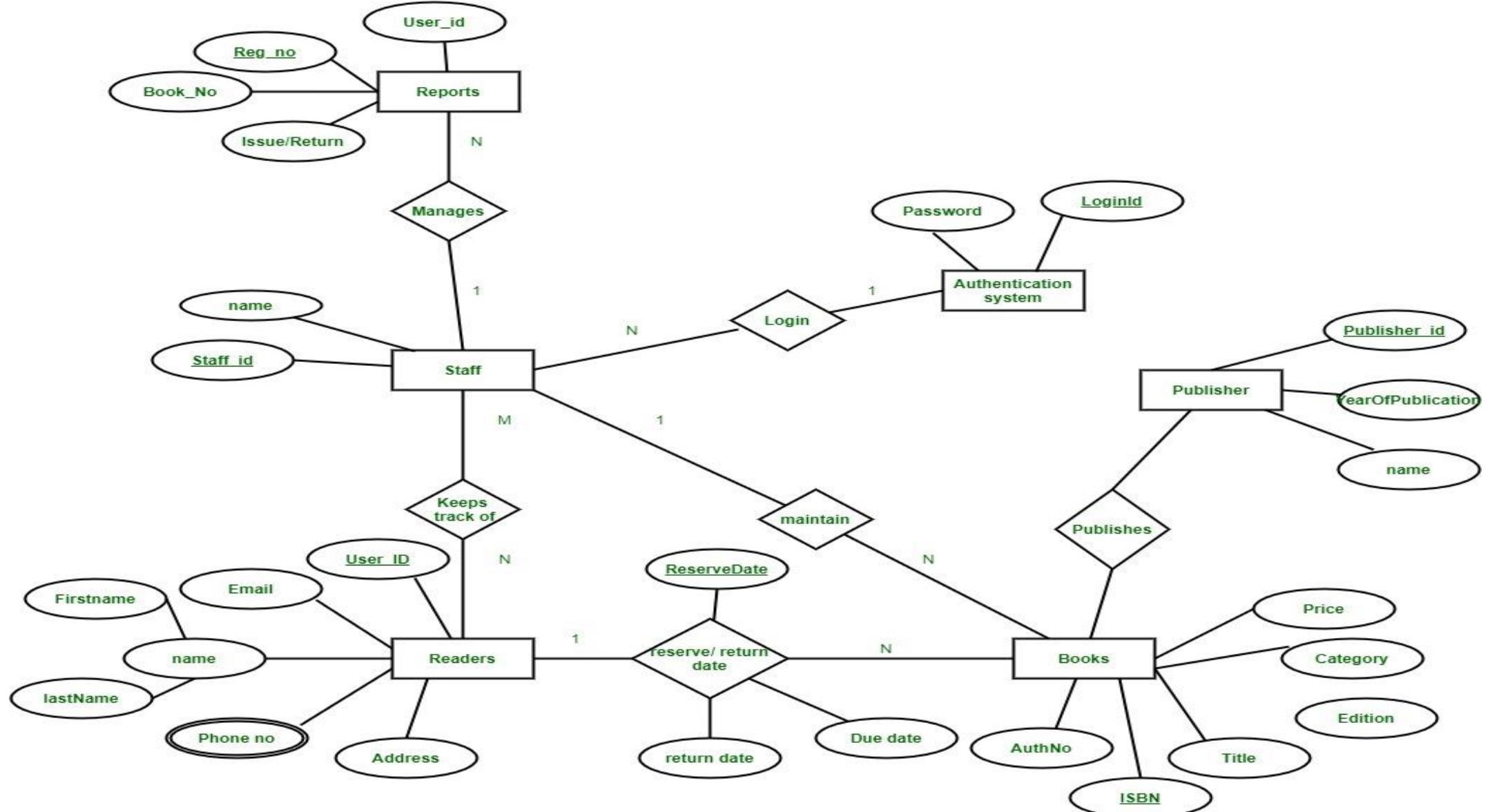
- **Publisher Entity :**

- It has PublisherId, Year of publication, name.
- PublisherID is the Primary Key.

The Library Management System, Cont'd...

- **Authentication System Entity :**
- It has LoginId and password with LoginID as Primary Key.
- **Reports Entity :**
- It has UserId, Reg_no, Book_no, Issue/Return date. Reg_no is the Primary Key of reports entity.
- Staff Entity :** It has name and staff_id with staff_id as Primary Key.
- **Reserve/Return Relationship Set :** It has three attributes: Reserve date, Due date, Return date.
- **Relationships between Entities** – A reader can reserve N books but one book can be reserved by only one reader. The relationship 1:N.A publisher can publish many books but a book is published by only one publisher. The relationship 1:N.Staff keeps track of readers. The relationship is M:N.Staff maintains multiple reports. The relationship 1:N.Staff maintains multiple Books. The relationship 1:N.Authentication system provides login to multiple staffs. The relation is 1:N.

The Library Management System, Cont'd...



Thank You

ER-to-Table

MAPPING ER DIAGRAMS TO RELATIONAL DATA MODEL

- Conceptual ER-models allow you to more accurately represent the subject area than logical models (relational, network, etc.).
- But, there are no DBMSs that support ER models i.e. Need to convert ER diagrams to an implementation schema.
- So, ER diagram is converted into the tables in Relational Data Model (RDM or RM).
- The **ER to RDM** mapping method is based on the formation of a set of initial relation tables from ER-diagrams (initial logical model) and based on the factors:
 - **atomic and multivalued** of attribute,
 - **cardinality** (max=one-many) **and**
 - **obligation** (min=optional-mandatory) of relationship.

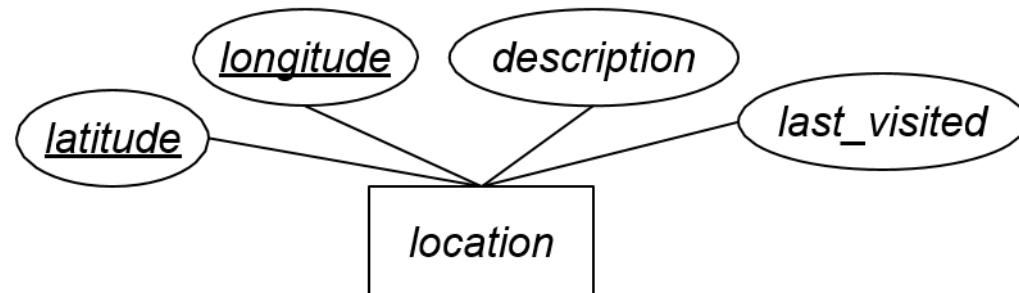
ER to RDM MAPPING BRIEFLY

- A **table** is created for each entity:
 - Each simple entity attribute corresponds to **a current table** column, derived entity attribute removed from **a current table**.
 - Each element of composite attribute corresponds to **a current table** column.
 - Each multivalued attribute **+1 separate table** will require.
- For Binary Relationship With **Weak Entity Set**:
 - **2 tables** for 2 entities will be required.
- For a relationship **without cardinality** ratios description:
 - **3 tables** will be required.
- For a relationship of cardinality type many-to-many (**m:n**) relationships between two entity sets:
 - **3 tables** will be required (two tables for entity sets and one for relationship)

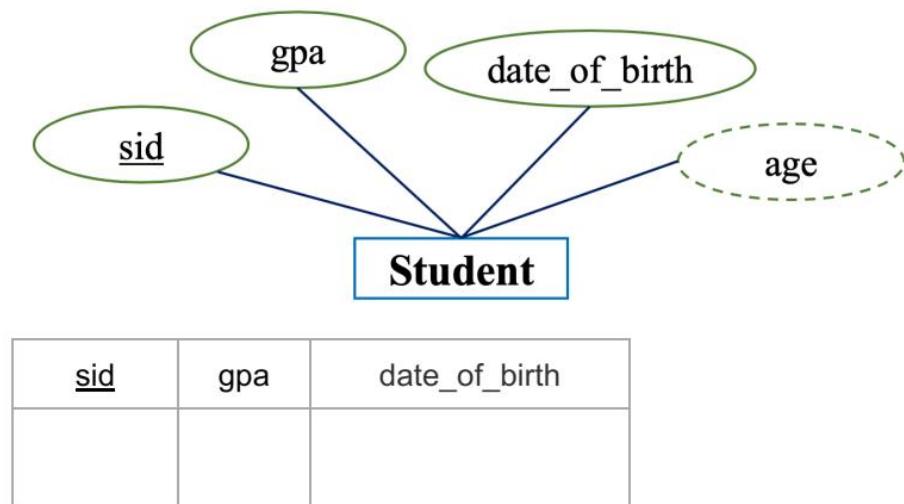
ER to Table

For Strong Entity Set With Only Simple Attributes

- A **strong entity set with only simple attributes** will require **only 1 table** in relational model.
- Attributes of the table will be the attributes of the entity set.
- **Derived attributes** removed from the table.
- The **primary key** of the table will be the key attribute of the entity set.



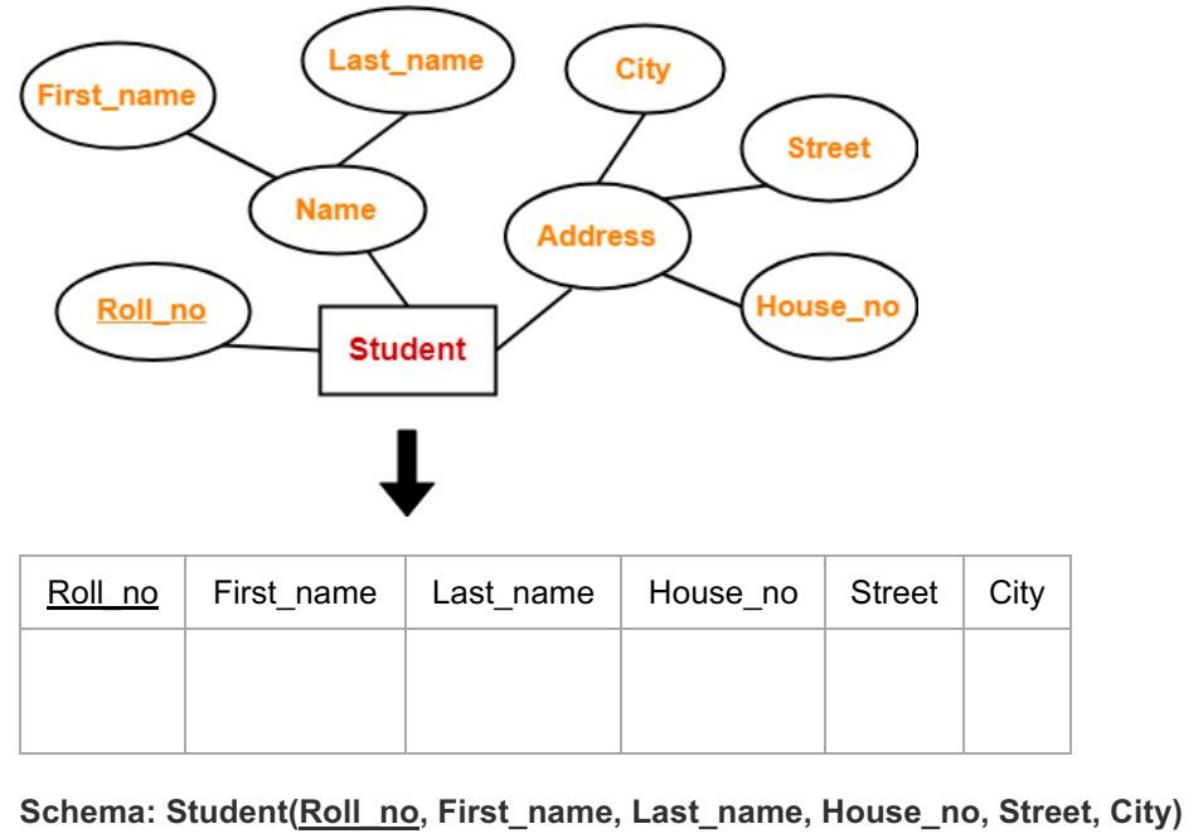
location(latitude, longitude, description, last_visited)



Schema: Student(sid, gpa, date_of_birth)

ER to Table: For Strong Entity Set With Composite Attributes

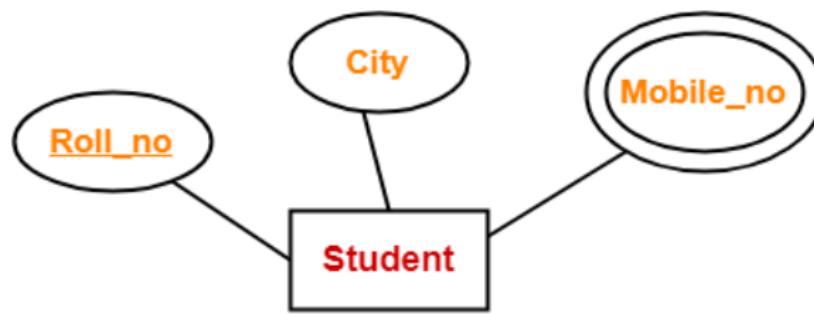
- A **strong entity set with any number of composite attributes** will require **only 1 table** in relational model.
- While conversion, simple attributes of the composite attributes are taken into account and not the composite attribute itself.



ER to Table:

For Strong Entity Set With Multivalued Attributes

- A **strong entity set with any number of multivalued attributes** will require **2 tables** in relational model.
- One table will contain all the simple attributes with the primary key.
- Other table will contain the primary key and all the multivalued attributes

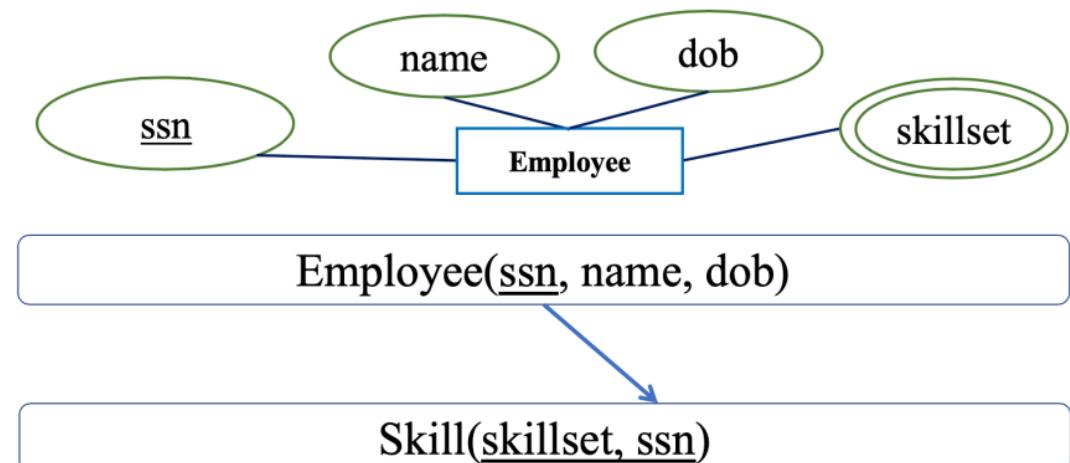


<u>Roll_no</u>	City
PK	

Student(Roll_no, City)
Students_Mobile(Roll_No, Mobile_No)

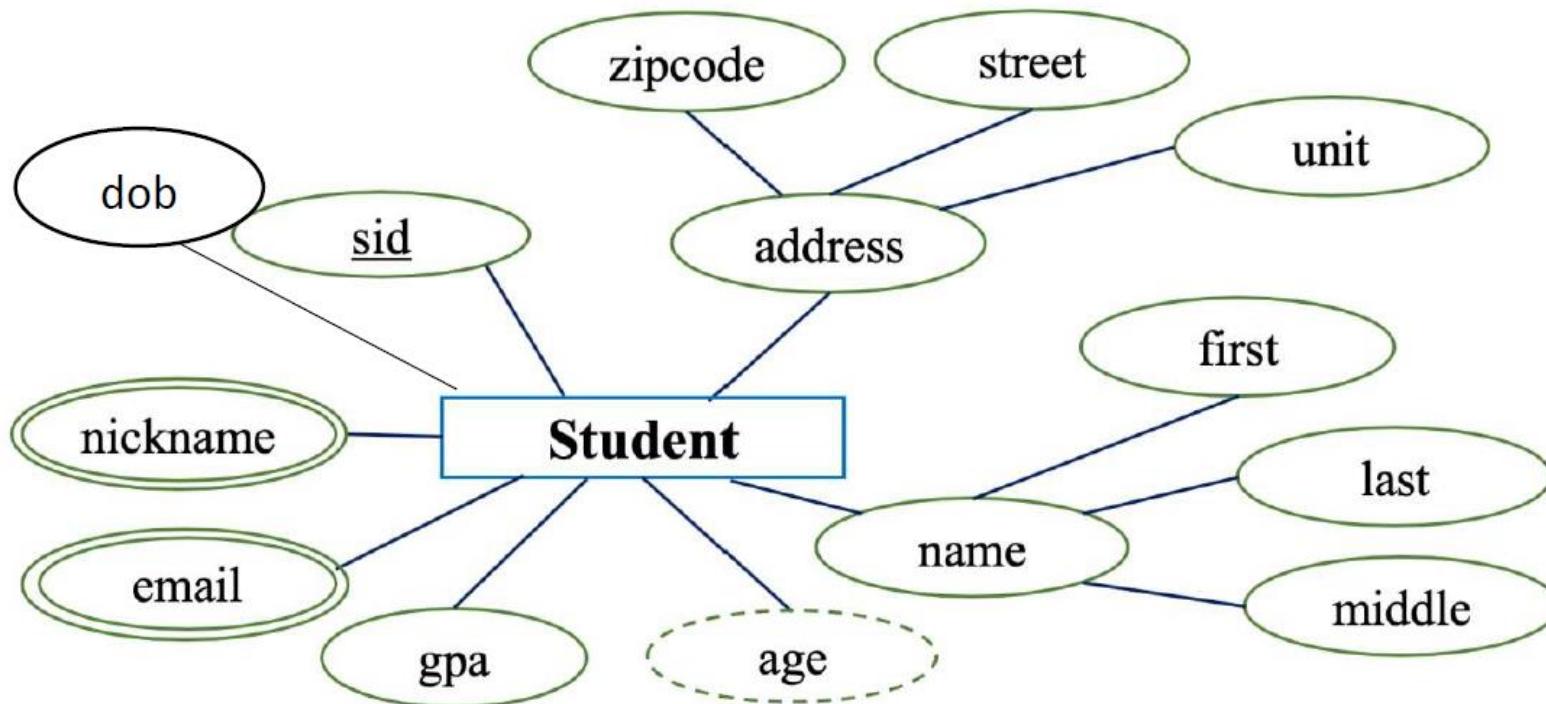
<u>Roll_no</u>	<u>Mobile_no</u>
PK,FK	PK

03-03-2025



Question:

Build A Relational Schema From The Following ER Diagram



Solution

Student(sid, zipcodeAddr, streetAddr, unitAddr, firstName, middleName, lastName, gpa, dob)

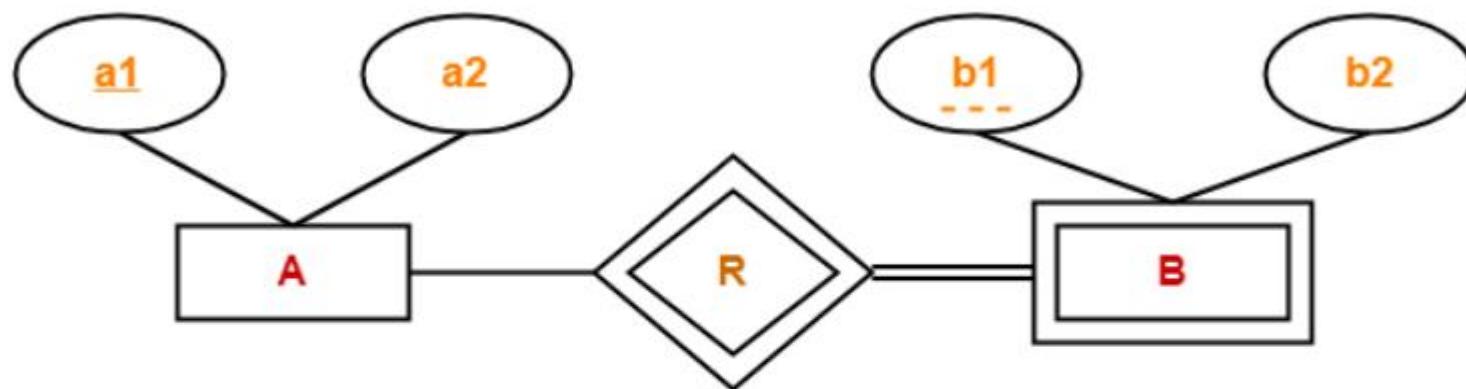
Nickname(sid, nickname)

Email(sid, email)

ER to Table: For Binary Relationship With Weak Entity Set

- Weak entity set always appears in association with identifying relationship with total participation constraint.

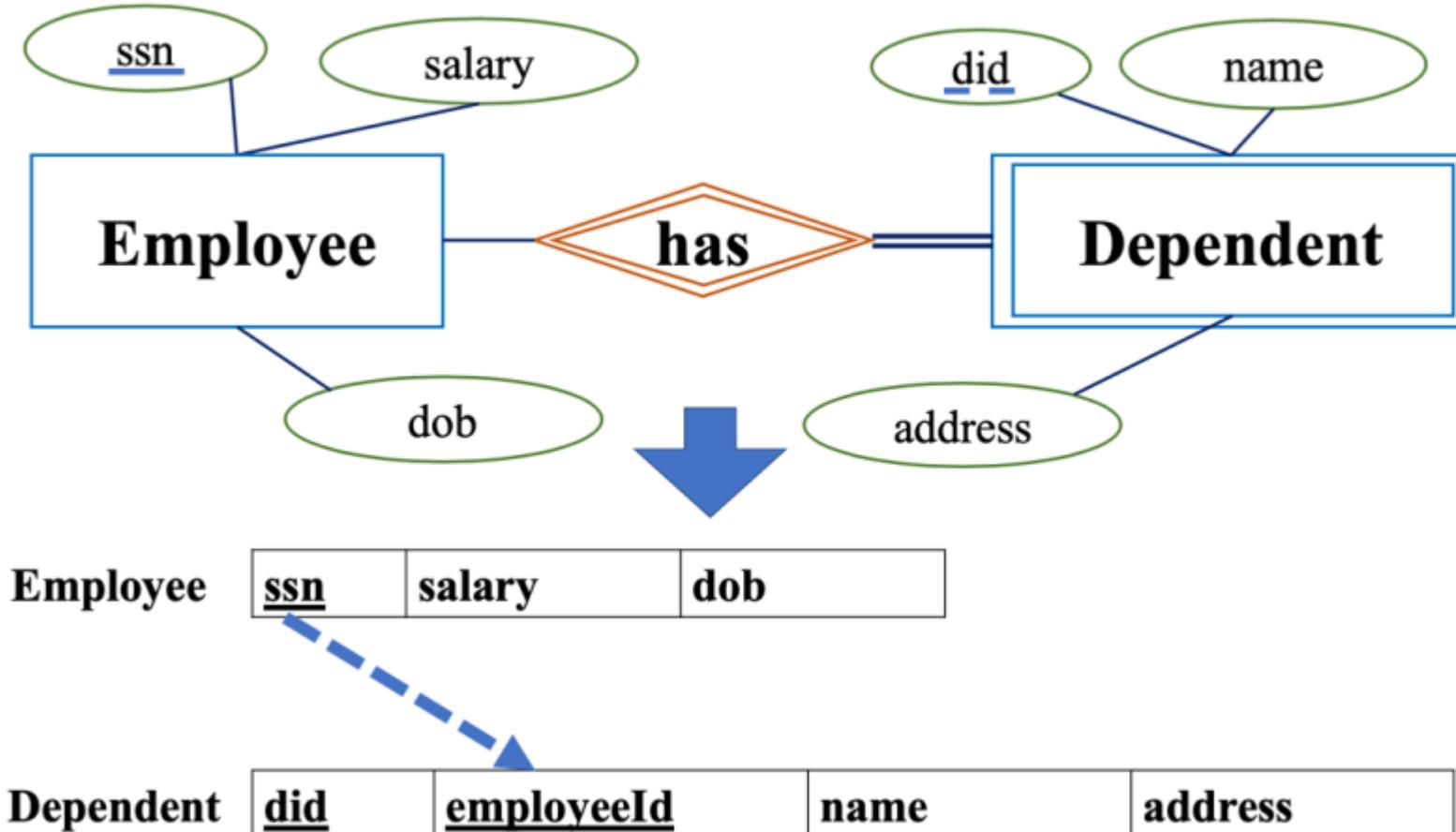
Examples



Here, **2 tables** will be required

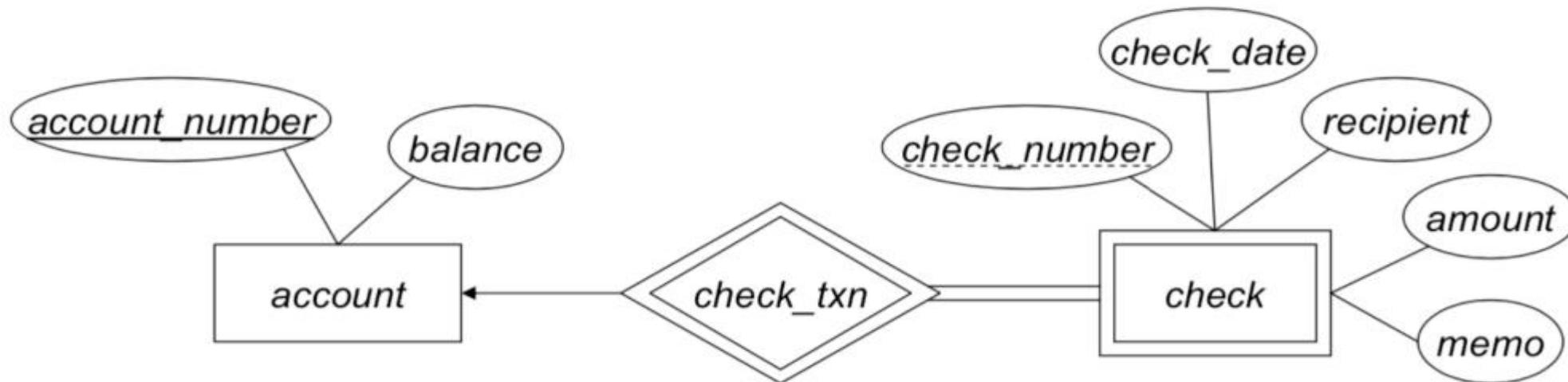
1. A(a₁, a₂)
2. BR(a₁, b₁, b₂)

ER to Table: For Binary Relationship With Weak Entity Set



Question:

Build the Relational Schema From The Following ER Diagram



Solution

account(account number, balance)

check schema:

Discriminator is check_number

Primary key for check is: (account_number, check_number)

check(account number, check number , check_date, recipient, amount, memo)

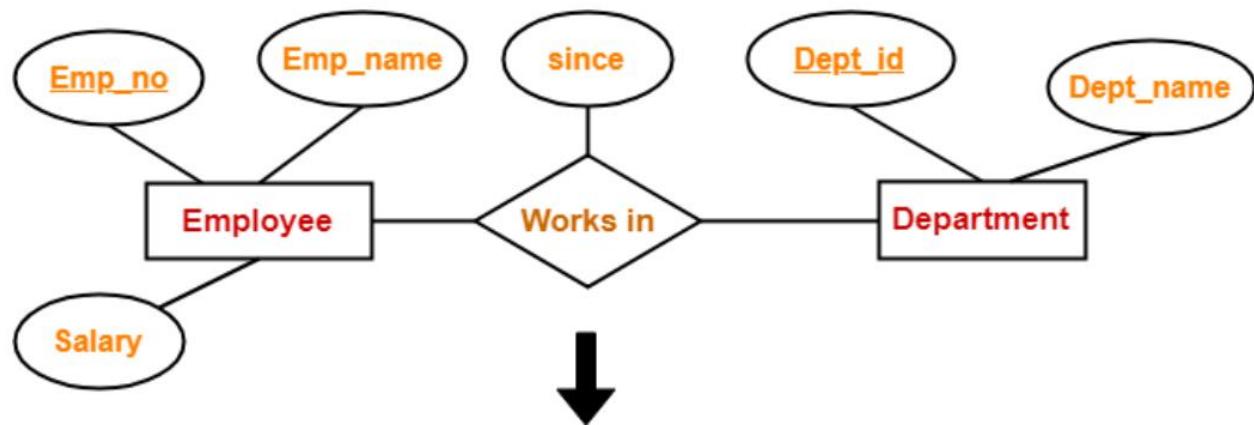
ER to Table:

Translating Relationship Set Without Cardinality Ratios Description Into A Table

- A relationship between 2 relations will require **3 tables** in the relational model.
- It will be treated as many-to-many relationship.
- Separate tables will be created for entity sets and relationships.

Attributes of the relationship table are:

- Primary key attributes of the participating entity sets (also become composite primary key in this table).
- Its own attributes if any.



If we consider the overall ER diagram, 3 tables will be required

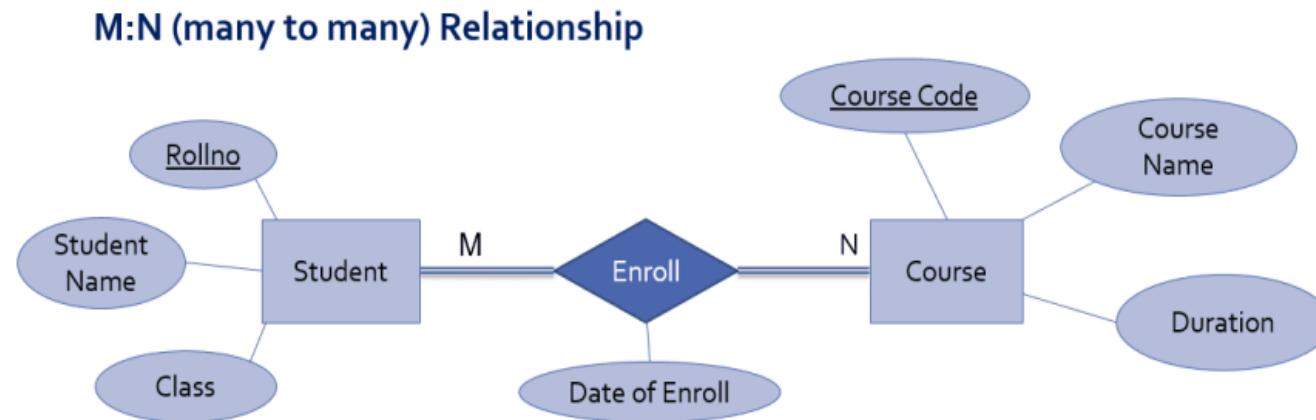
- One table for the entity set “Employee”
- One table for the entity set “Department”
- One table for the relationship set “WorksIn”

<u>Emp_no</u>	<u>Dept_id</u>	since

Schema: WorksIn(Emp_no, Dept_id, since)

ER to Table: For Binary Relationship With Cardinality Ratio M:N

Consider same relationship set enrolled exist between entity sets student and course, which means multiple students can enroll in multiple courses



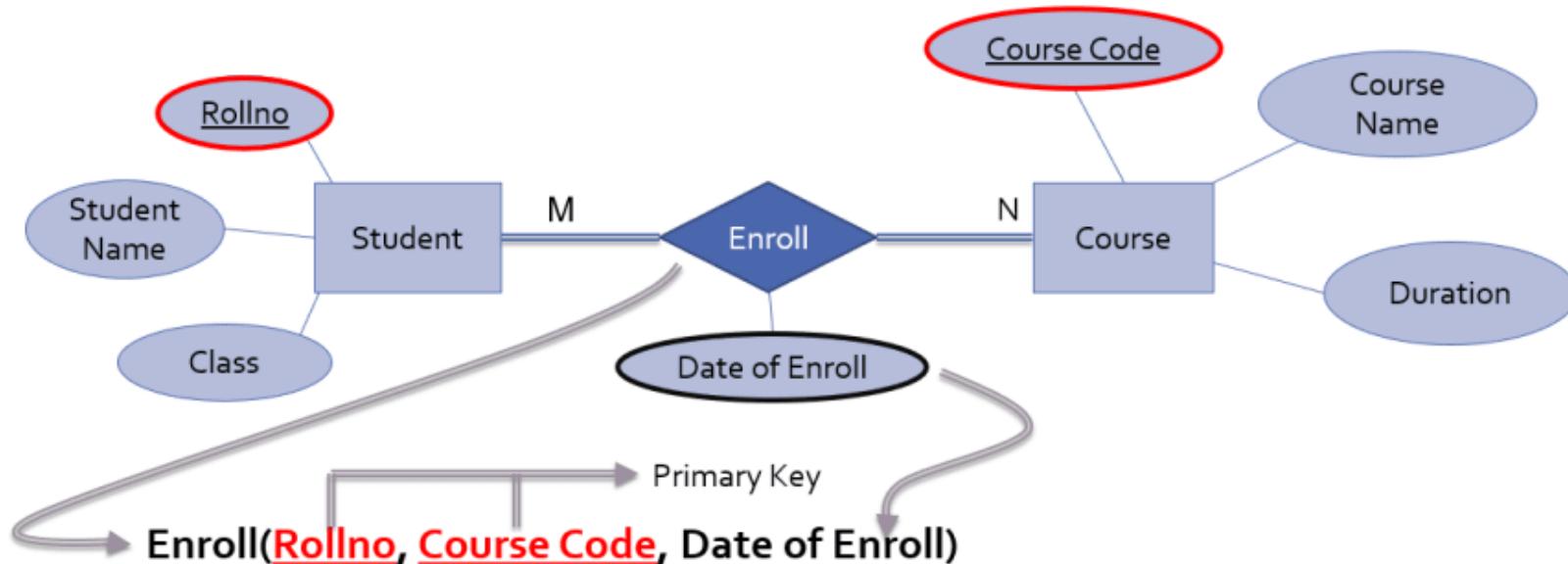
To convert this Relationship set into relational schema:

1. Relationship set is mapped as separate relation
2. Key attributes of participating entity sets are mapped as primary key for that relation
3. Attribute of relationship set becomes simple attributes for that relation
4. And separate relation is created for other participating entities

ER to Table:

For Binary Relationship With Cardinality Ratio M:N

Consider same relationship set enrolled exist between entity sets student and course, which means **multiple students can enroll in multiple courses**

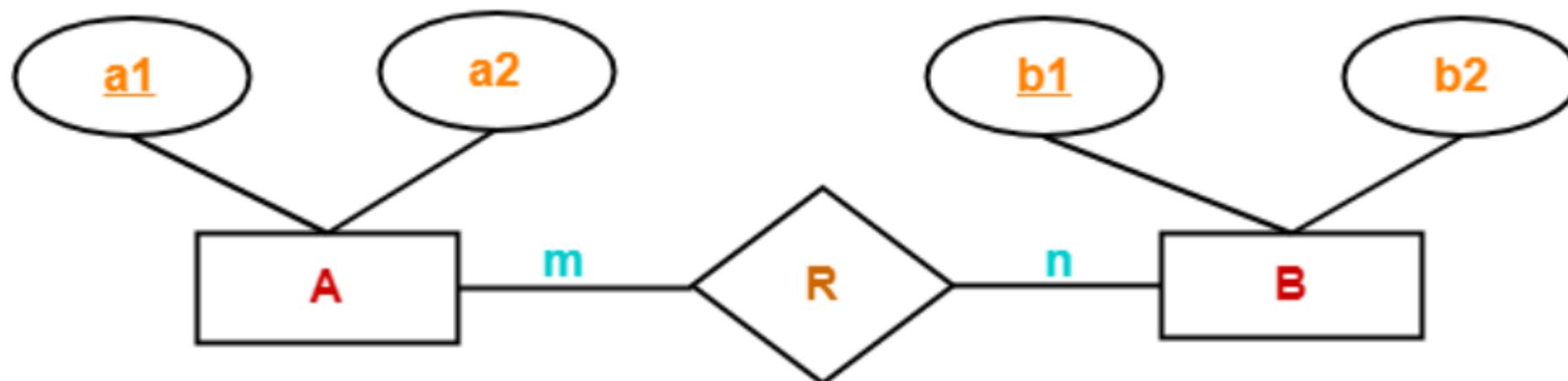


Student(Rollno, Student Name, Class)

Course(Course Code, Course Name, Duration)

ER to Table: For Binary Relationship With Cardinality Ratio M:N

Examples

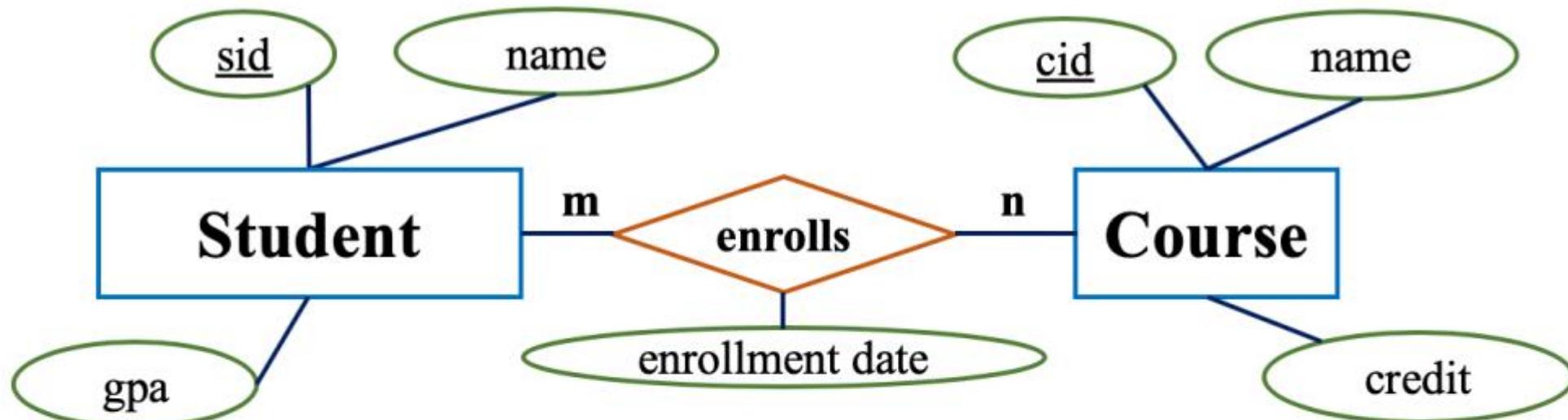


Here, **3 tables** will be required

1. A(a1, a2)
2. R(a1, b1)
3. B(b1, b2)

ER to Table:

For Binary Relationship With Cardinality Ratio M:N



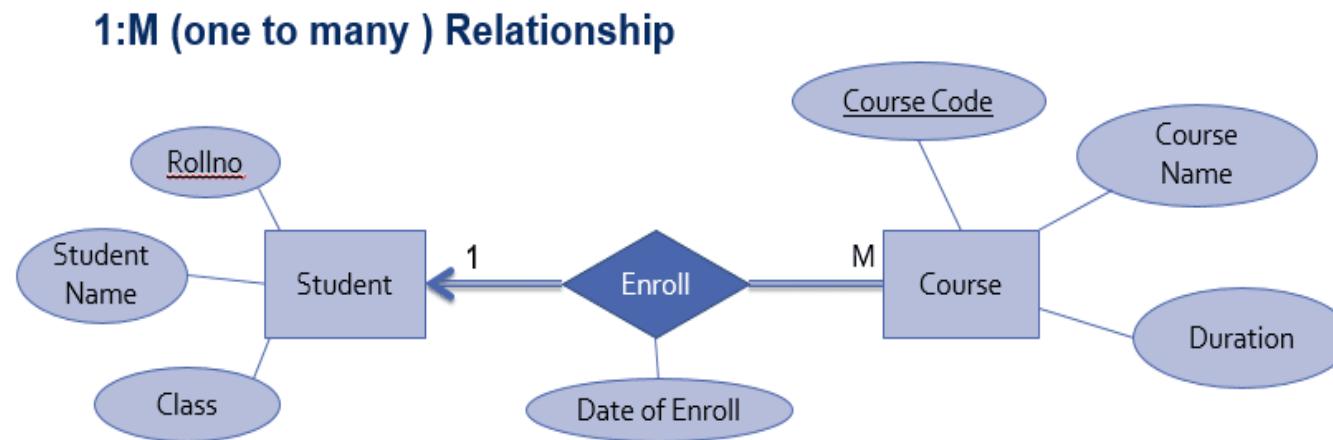
Student	<u>sid</u>	name	gpa

course	<u>cid</u>	name	credit

Enrolls	<u>sid</u>	<u>cid</u>	enrollmentDate

ER to Table: For Binary Relationships With Cardinality Ratio 1:M

Consider 1:M relationship set enrolled exist between entity sets student and course as follow,

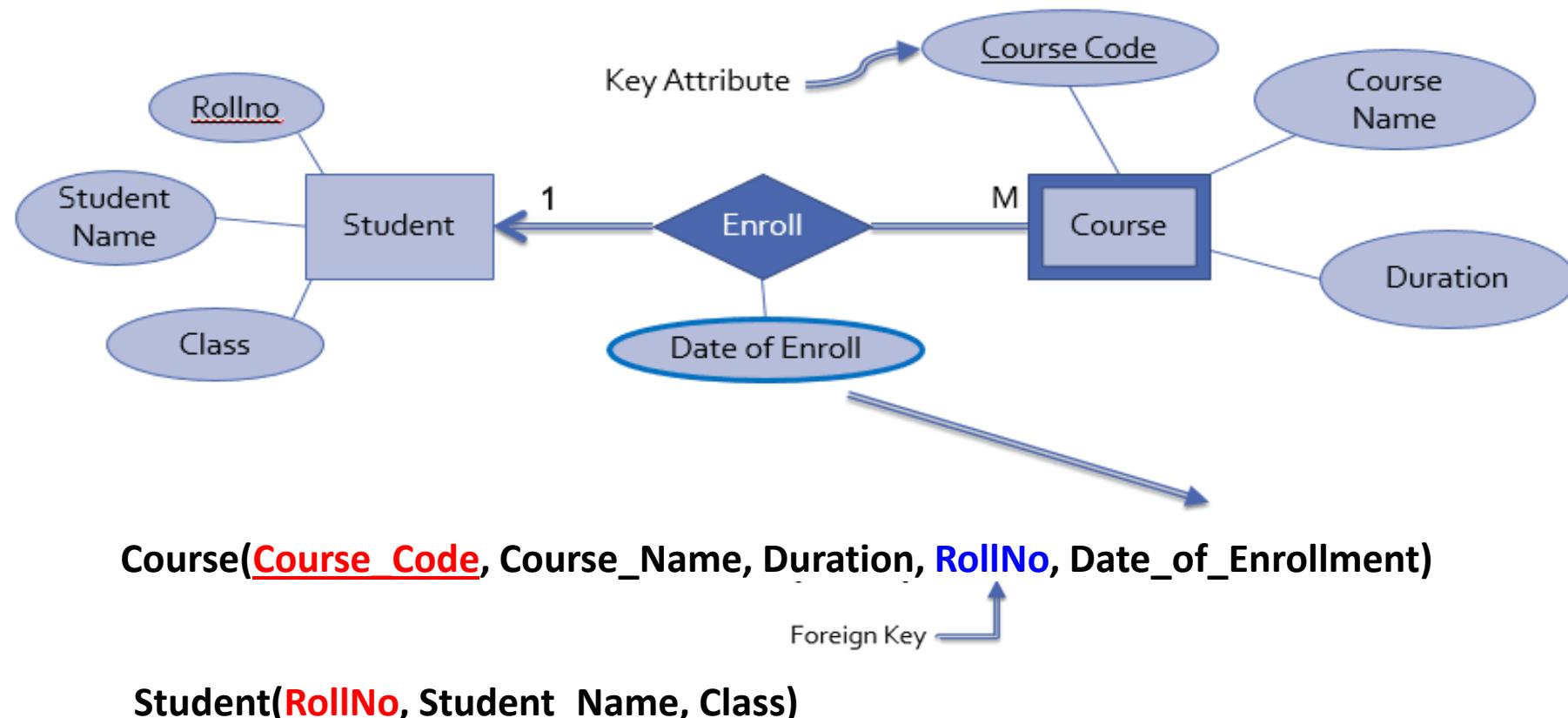


Here **Enroll** is 1:M relationship exist between entity set **student** and **course** which means that **one student can enroll in multiple courses**. In this case, to convert this relationship into relational schema:

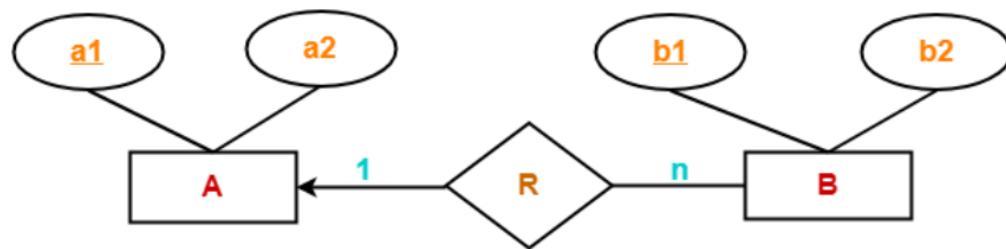
1. Separate relation is created for all participating entity sets (**student** and **course**)
2. Key attribute of one's side entity set (Student) is mapped as foreign key in many's side relation (Course).
3. All attributes of relationship set are mapped as attributes for relation of one's side entity set (student)

ER to Table: For Binary Relationships With Cardinality Ratio 1:M

Consider 1:M relationship set enrolled exist between entity sets student and course as follow,



ER to Table: For Binary Relationships With Cardinality Ratio 1:M

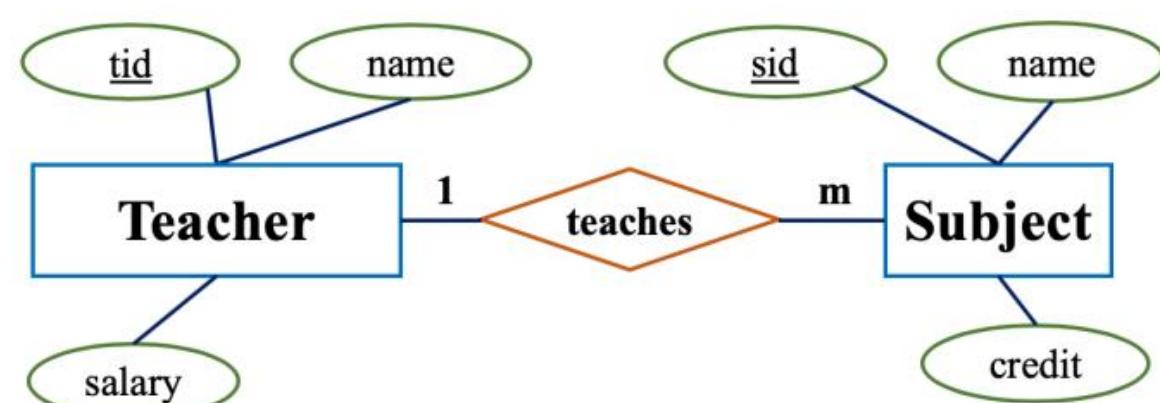


Here, 2 tables will be required

1. A(a₁, a₂)
2. BR(b₁, b₂, a₁)

NOTE. Here, combined table will be drawn for the entity set B and relationship set R.

Question: Convert the following Schema to Relations:

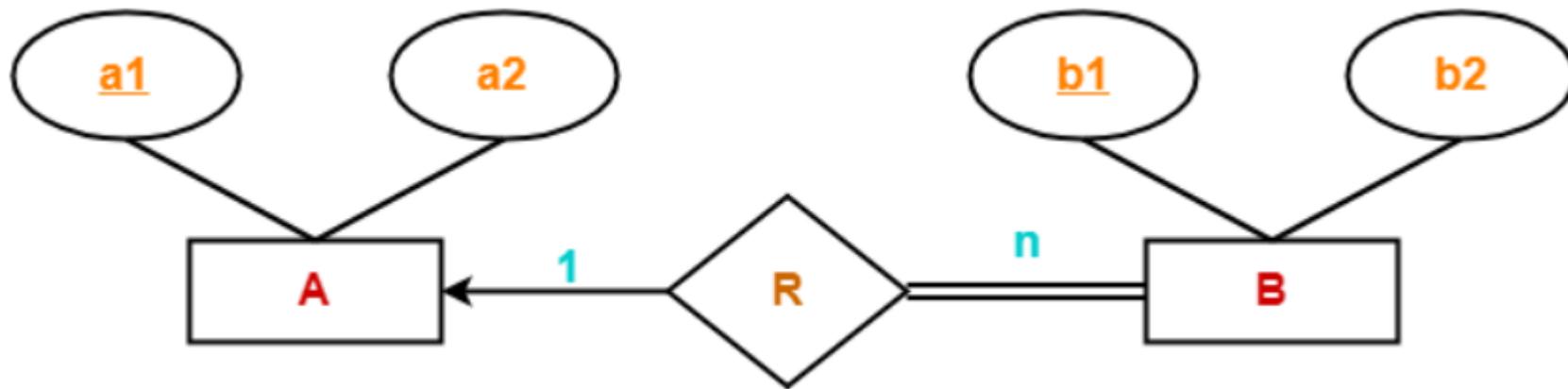


Teacher	tid	name	salary	
Subject	sid	name	credit	teacherId

ER to Table:

For Binary Relationships With Cardinality Ratio 1:M with Total Participation Constraints

Foreign Key with NOT NULL constraints will be applied.



A(a1, a2)

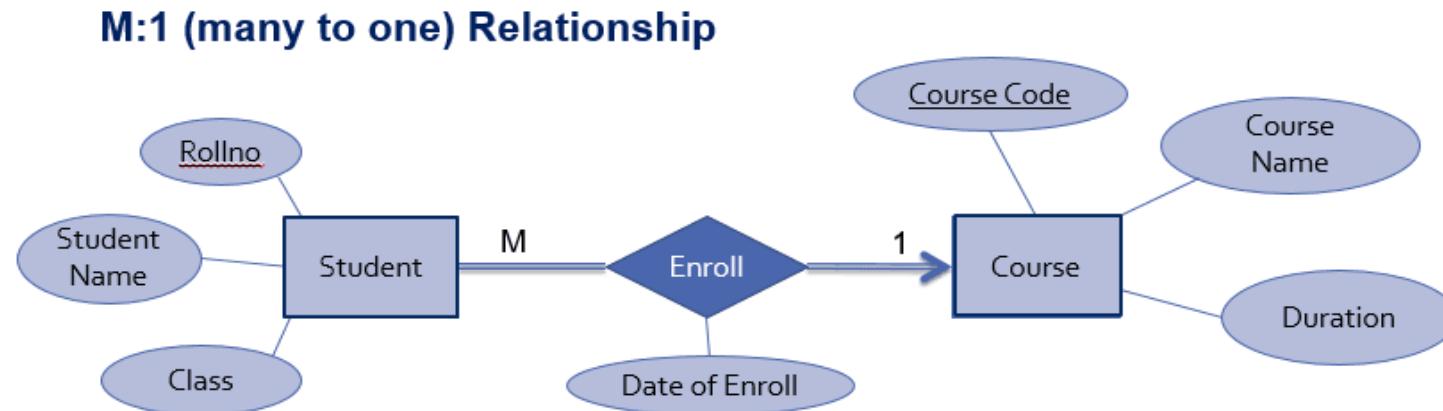
B(b1, b2, a1)

a1 as NOT NULL.

ER to Table:

For Binary Relationships With Cardinality Ratio M:1

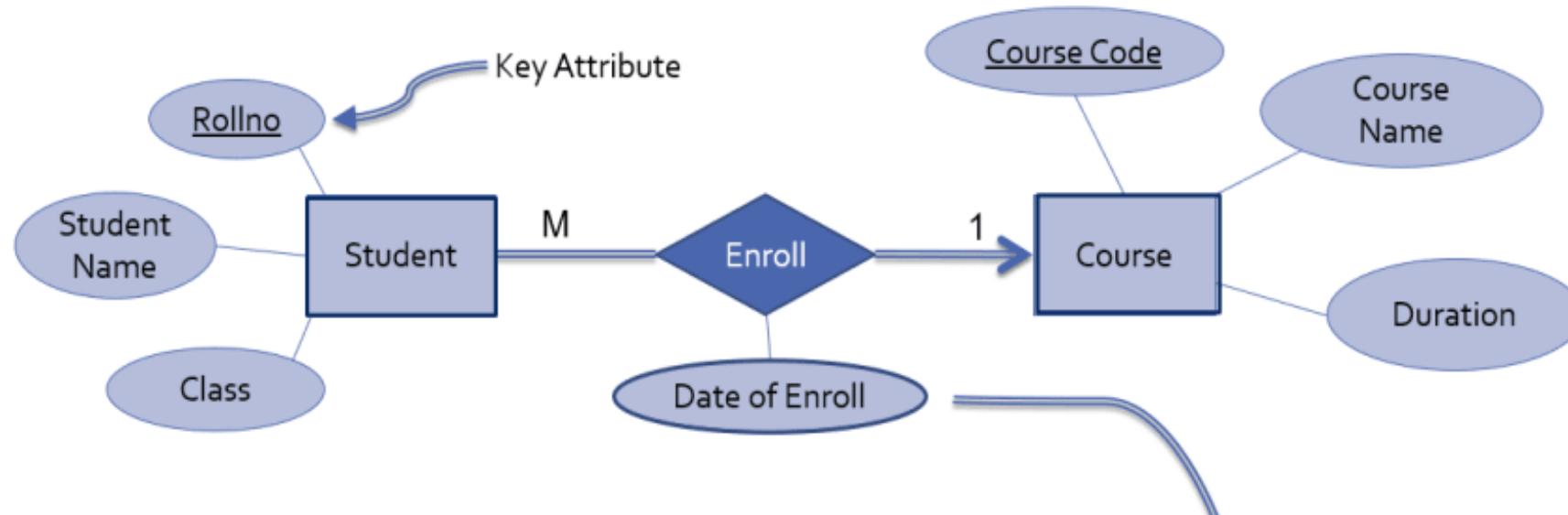
Consider same relationship set enroll exist between entity sets student and course. But here student is many side entities set while course is one side entity set. Which means **many students can enroll in one course.**



To convert this relationship set into relational schema:

1. Separate relation is created for all participating entity sets.
2. Key attribute of one's side entity set (Course) is mapped as foreign key in Many's side relation (Student)
3. All attributes of relationship set are mapped as attributes for one's side relation course.

ER to Table: For Binary Relationships With Cardinality Ratio M:1

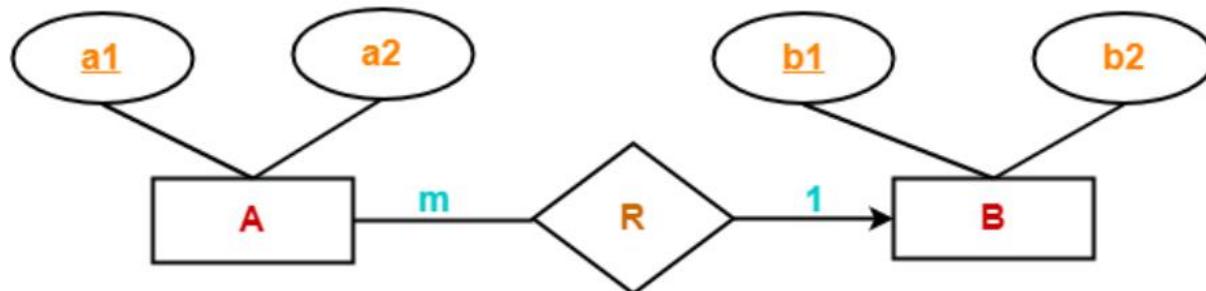


Student(RollNo, Student_Name, Class, Course_Code, Date_of_Enrollment)

Foreign Key

Course(Course_Code, Course_Name, Duration)

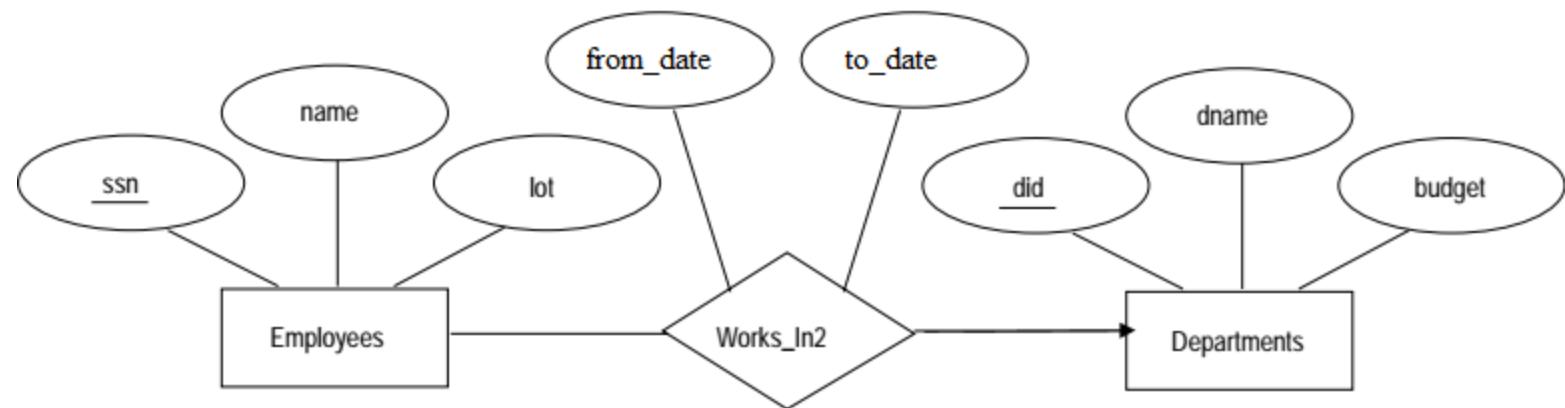
ER to Table: For Binary Relationships With Cardinality Ratio M:1



Here, 2 tables will be required

1. AR(a1, a2, b1)
2. B(b1, b2)

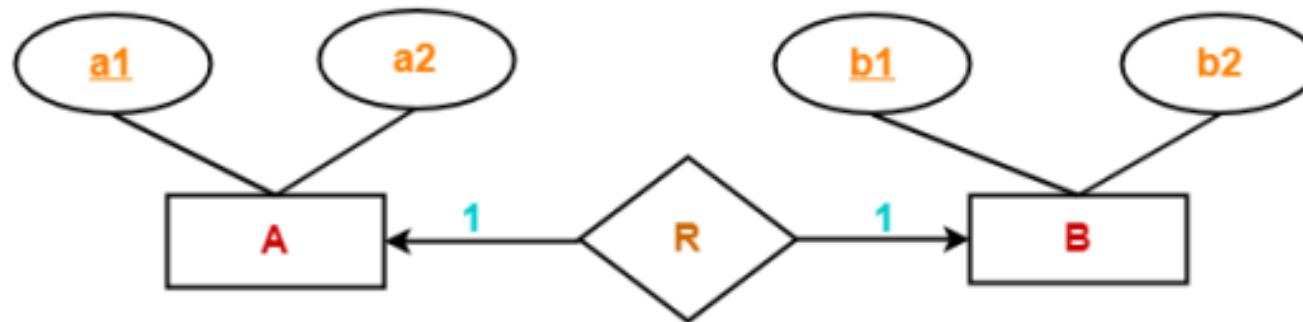
Question: Convert the following Schema to Relations:



Employees(ssn, name, lot, **did**, from_date, to_date)
Departments(**did**, dname, budget)

ER to Table: For Binary Relationships With Cardinality Ratio 1:1

Case 1: Binary Relationship with 1:1 cardinality and partial participation of both entities



Here, **2 tables** will be required. Either combine 'R' with 'A' or 'B' including FK as UNIQUE.

1. AR(a1, a2, b1)
2. B(b1, b2)

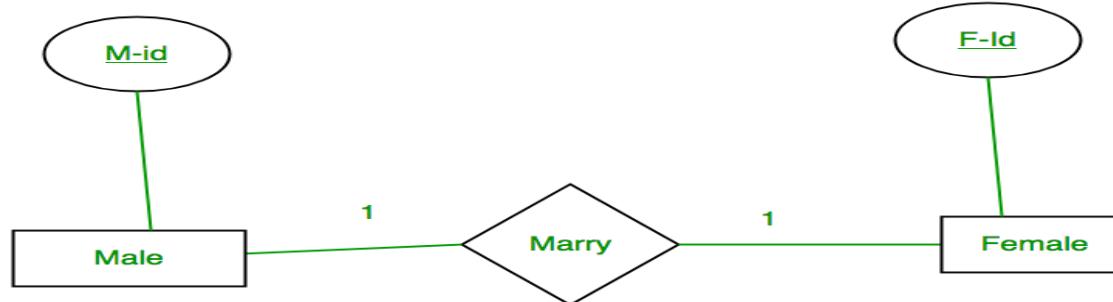
OR

1. A(a1, a2)
2. BR(a1, b1, b2)

ER to Table: For Binary Relationships With Cardinality Ratio 1:1

Case 1: Binary Relationship with 1:1 cardinality and partial participation of both entities

Example: A male marries 0 or 1 female and vice versa as well. So, it is 1:1 cardinality with partial participation constraint from both.



- Binary relationship with 1:1 cardinality will have 2 table if partial participation of both entities in the relationship.
- Here, we can use **PK of one table as FK in another table including FK as UNIQUE**.

Male(M_ID, Other attributes, F_ID)
Female(F_ID, other attributes)

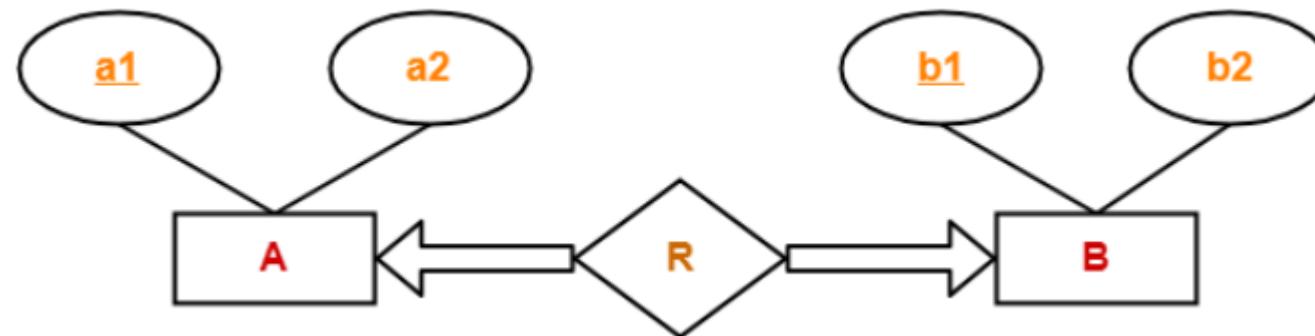
(OR)

Male(M_ID, Other attributes)
Female(F_ID, other attributes, M_ID)

ER to Table: For Binary Relationships With Cardinality Ratio 1:1

Case 2: Binary Relationship with 1:1 cardinality and total participation of both entities

If there is total participation for both entity sets in a binary relationship with cardinality 1:1 then the binary relationship is represented using only single table including UNIQUE and NOT NULL Constraints



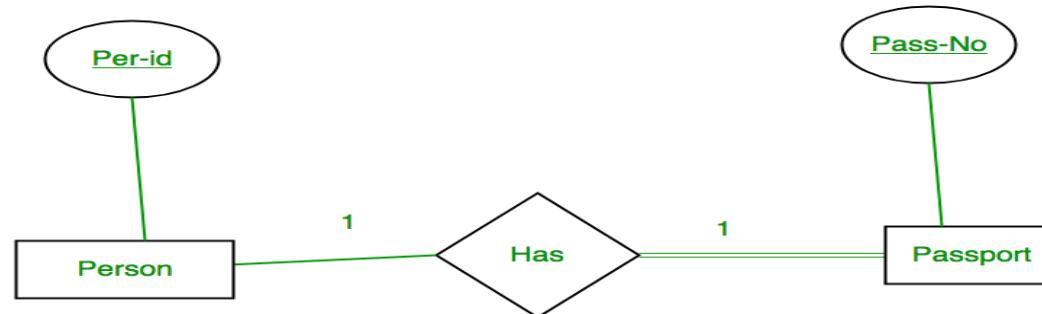
Here, Only one table is required

- ARB(a1, a2, b1, b2)

ER to Table: For Binary Relationships With Cardinality Ratio 1:1

Case 3: Binary Relationship with 1:1 cardinality with total participation of an entity set.

Example: A person has 0 or 1 passport number and Passport is always owned by 1 person. So it is 1:1 cardinality with full participation constraint from Passport.



Person(Per_ID, Other Person Attributes, Pass_No, Other Passport Attributes, Pass_No should be Unique.)

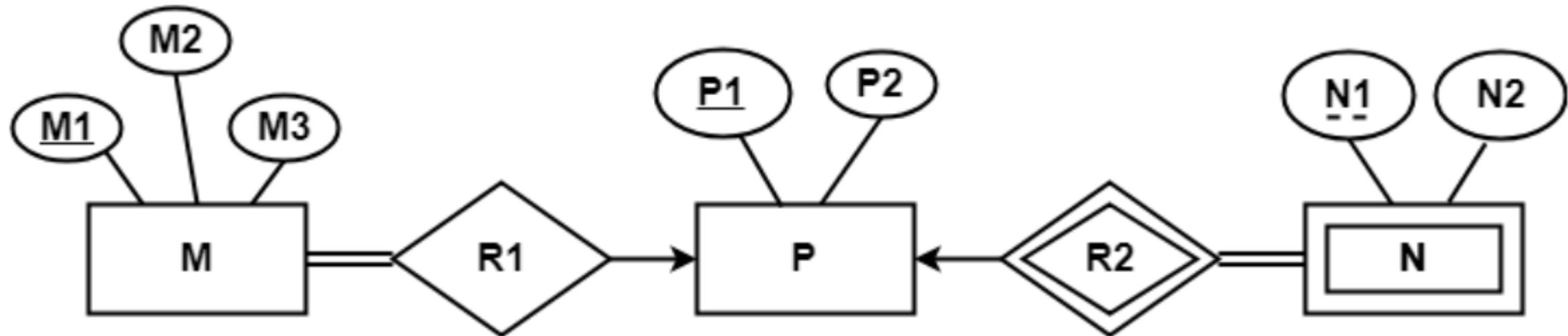
While determining the minimum number of tables required for binary relationships with given cardinality ratios, following thumb rules must be kept in mind-

- For binary relationship with cardinality ratio **$m : n$** ,
 - Separate and individual tables will be drawn for each entity set and relationship (i.e. **three tables** will be required).
- For binary relationship with cardinality ratio either **$m : 1$ or $1 : n$** ,
 - Always remember “many side will consume the relationship” i.e. a combined table will be drawn for many side entity set and relationship set (i.e. **Two tables** will be required).
- For binary relationship with cardinality ratio **$1 : 1$** ,
 - **Two tables** will be required. You can combine the relationship set with any one of the entity sets.

Practice Problem Based On Converting ER Diagram To Tables

PROBLEM-01:

Find the minimum number of tables required for the following ER diagram in relational model



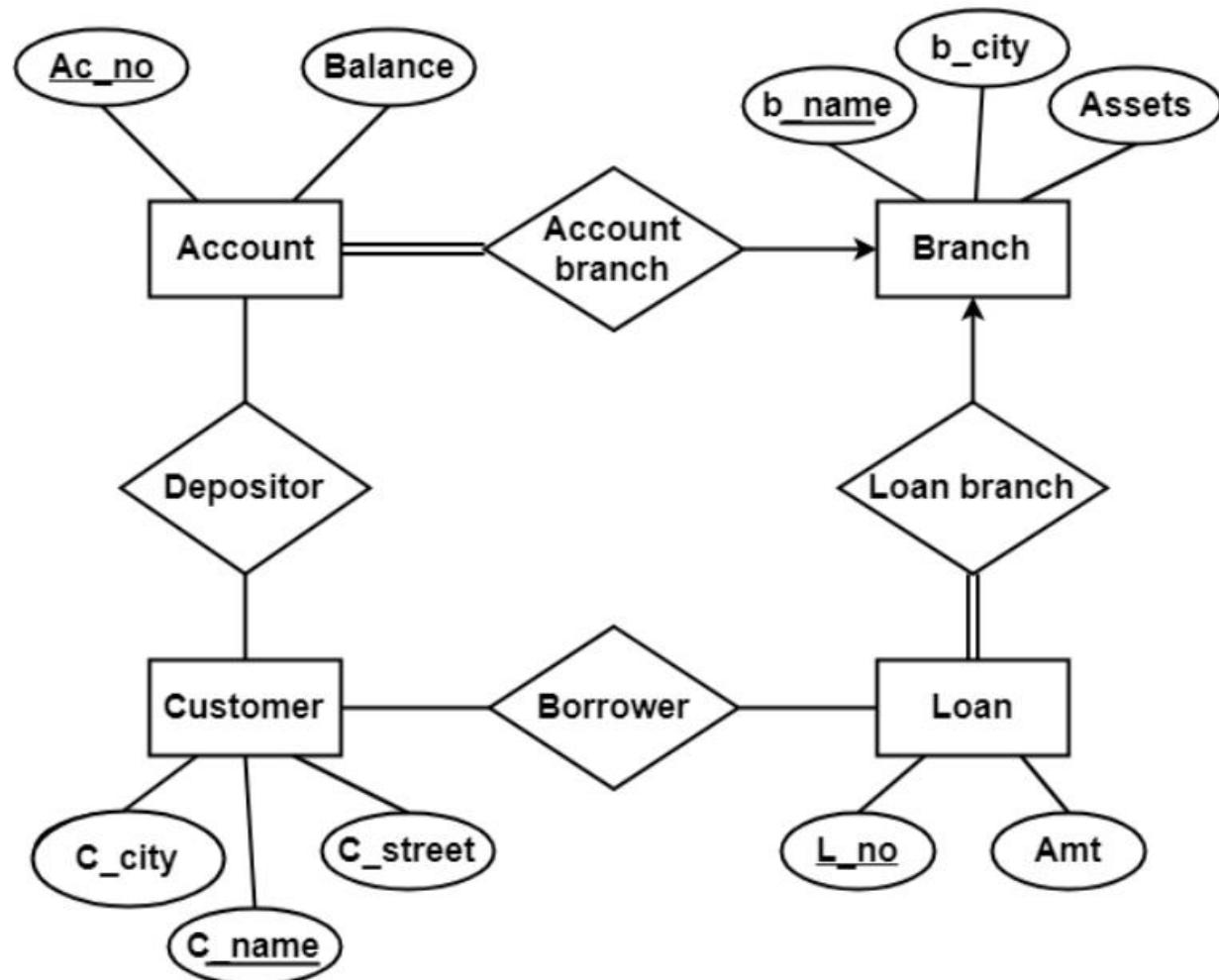
SOLUTION:

Applying the rules, minimum 3 tables will be required

- $\text{MR1}(\underline{\text{M1}}, \text{M2}, \text{M3}, \text{P1})$
- $\text{P}(\underline{\text{P1}}, \text{P2})$
- $\text{NR2}(\underline{\text{P1}}, \underline{\text{N1}}, \text{N2})$

Practice Problem Based On Converting ER Diagram To Tables

PROBLEM-02: Find the minimum number of tables required to represent the given ER diagram in relational model



SOLUTION

Account (Ac_no, Balance, b_name)
Branch (b_name, b_city, Assets)
Loan(L_no, Amt, b_name)
Borrower(C_name, L_no)
Customer(C_name, C_street, C_city)
Depositor(C_name, Ac_no)

Thank You

Normalization

Topics to be Covered

- ❑ Functional Dependency
- ❑ Armstrong's Axioms
- ❑ Closure of a set of FDs
- ❑ Canonical Cover
- ❑ Closure of a set of attributes to decide the key
- ❑ Anomaly detection
- ❑ Decomposition
- ❑ Normalization and Normal Forms

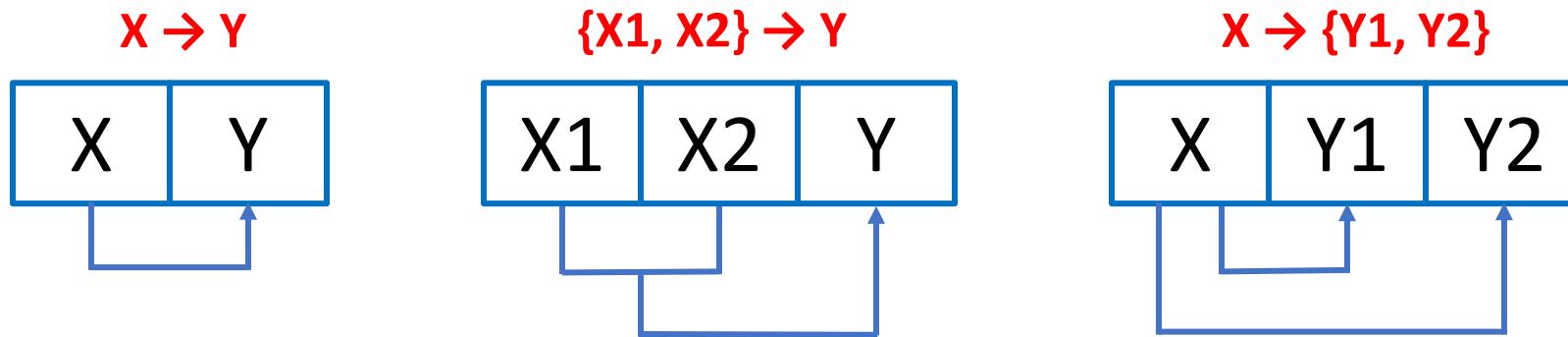
Functional Dependency (FD) and its types

What is Functional Dependency (FD)?

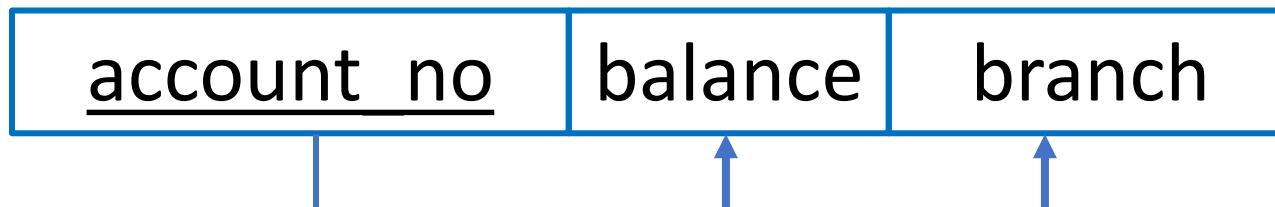
- Let R be a relation schema having n attributes A₁, A₂, A₃,..., A_n.
- Let attributes X and Y are two subsets of attributes of relation R.
- If the **values of the X component of a tuple uniquely** (or functionally) **determine the values of the Y component**, then there is a **functional dependency from X to Y**. This is denoted by **X → Y**.
- (i.e RollNo → Name, SPI, BL).
- It is referred as: **Y is functionally dependent on the X or X functionally determines Y**.

Student			
RollNo	Name	SPI	BL
101	Raju	8	0
102	Mitesh	7	1
103	Jay	7	0

Diagrammatic representation of (FD)



- Example
- Consider the relation Account(account_no, balance, branch).
- account_no can determine balance and branch.
- So, there is a functional dependency from account_no to balance and branch.
- This can be denoted by $\text{account_no} \rightarrow \{\text{balance}, \text{branch}\}$.



Types of Functional Dependency (FD)

ti

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

- **Full Functional Dependency**

- In a relation, the attribute B is fully functional dependent on A if **B is functionally dependent on A, but not on any proper subset of A.**
- Eg. $\{ \text{Roll_No}, \text{Semester}, \text{Department_Name} \} \rightarrow \text{SPI}$
- We **need all three {Roll_No, Semester, Department_Name} to find SPI.**

Types of Functional Dependency (FD)



THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

• Partial Functional Dependency

- In a relation, the attribute B is partial functional dependent on A if **B is functionally dependent on A as well as on any proper subset of A.**
- If there is some attribute that can be removed from A and the still dependency holds then it is partial functional dependency.
- Eg. $\{\text{Enrollment_No}, \text{Department_Name}\} \rightarrow \text{SPI}$
- **Enrollment_No is sufficient to find SPI**, Department_Name is not required to find SPI.

Types of Functional Dependency (FD)

- **Transitive Functional Dependency**

- In a relation, if attribute(s) **A → B and B → C, then A → C (means C is transitively depends on A via B).**

Sub_Fac		
Subject	Faculty	Age
DS	Shah	35
DBMS	Patel	32
DF	Shah	35

- Eg. Subject → Faculty & Faculty → Age then Subject → Age
- Therefore as per the rule of transitive dependency: **Subject → Age** should hold, that makes sense because if we know the subject name we can know the faculty's age.

Types of Functional Dependency (FD)

ti

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

- **Trivial Functional Dependency**

- $X \rightarrow Y$ is trivial FD if **Y is a subset of X**
- Eg. $\{\text{Roll_No}, \text{Department_Name}, \text{Semester}\} \rightarrow \text{Roll_No}$

- **Nontrivial Functional Dependency**

- $X \rightarrow Y$ is nontrivial FD if **Y is not a subset of X**
- Eg. $\{\text{Roll_No}, \text{Department_Name}, \text{Semester}\} \rightarrow \text{Student_Name}$

Armstrong's axioms OR Inference rules

- Armstrong's axioms are a set of rules used to infer (derive) all the functional dependencies on a relational database.

Reflexivity

- If B is a subset of A
- then $A \rightarrow B$

Augmentation

- If $A \rightarrow B$
- then $AC \rightarrow BC$

Self-determination

- If $A \rightarrow A$

Transitivity

- If $A \rightarrow B$ and $B \rightarrow C$
- then $A \rightarrow C$

Pseudo Transitivity

- If $A \rightarrow B$ and $BD \rightarrow C$
- then $AD \rightarrow C$

Decomposition

- If $A \rightarrow BC$
- then $A \rightarrow B$ and $A \rightarrow C$

Union

- If $A \rightarrow B$ and $A \rightarrow C$
- then $A \rightarrow BC$

Composition

- If $A \rightarrow B$ and $C \rightarrow D$
- then $AC \rightarrow BD$

Closure of a set of FDs

What is closure of a set of FDs?

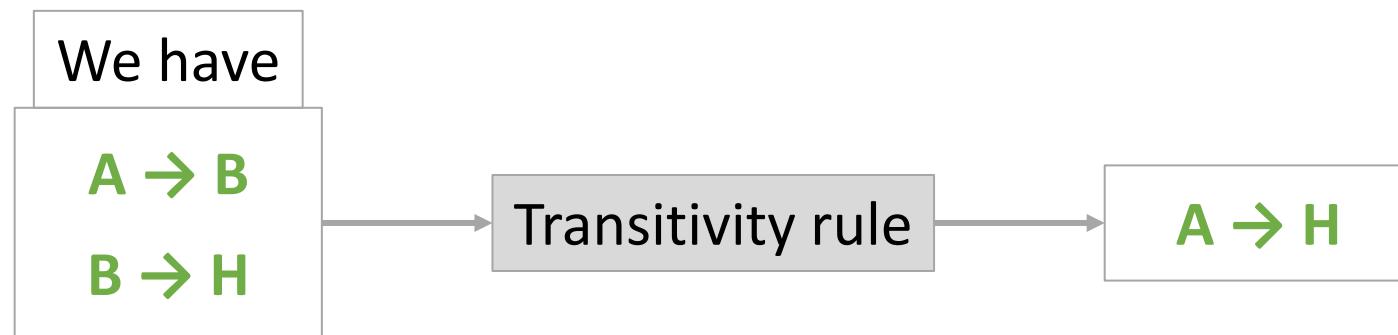


THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

- Given a set F set of functional dependencies, there are certain other **functional dependencies that are logically implied by F .**
- E.g.: $F = \{A \rightarrow B \text{ and } B \rightarrow C\}$, then we can infer that $A \rightarrow C$ (by transitivity rule)
- The set of **functional dependencies (FDs) that is logically implied by F** is called the **closure of F** . It is denoted by F^+ .

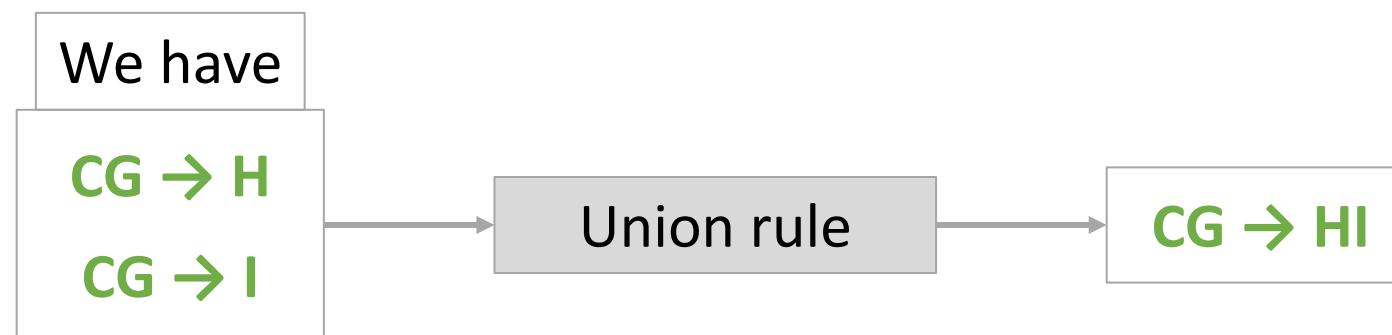
Closure of a set of FDs [Example]

- ▶ Suppose we are given a relation schema $R(A,B,C,G,H,I)$ and the set of functional dependencies are:
 - $F = (\underline{A \rightarrow B}, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, \underline{B \rightarrow H})$
- The functional dependency $A \rightarrow H$ is logically implied.



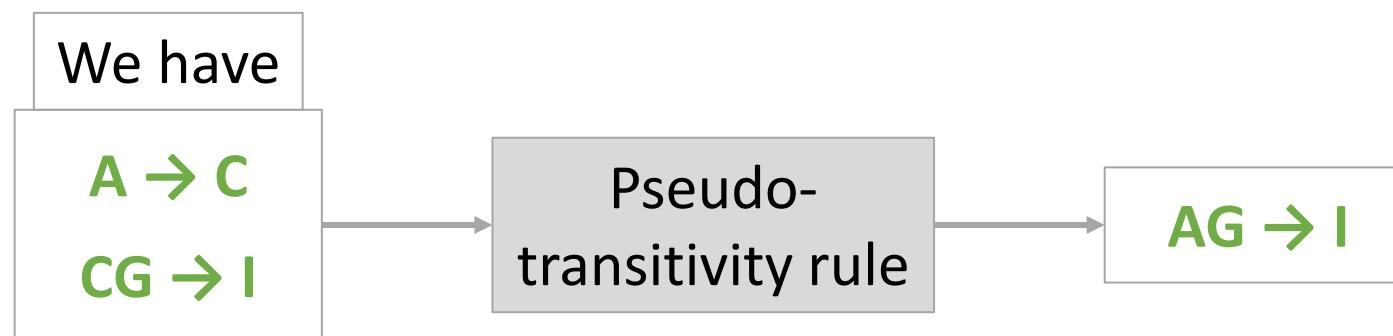
Closure of a set of FDs [Example]

- ▶ Suppose we are given a relation schema $R(A,B,C,G,H,I)$ and the set of functional dependencies are:
 - $F = (A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H)$
- The functional dependency $CG \rightarrow HI$ is logically implied.



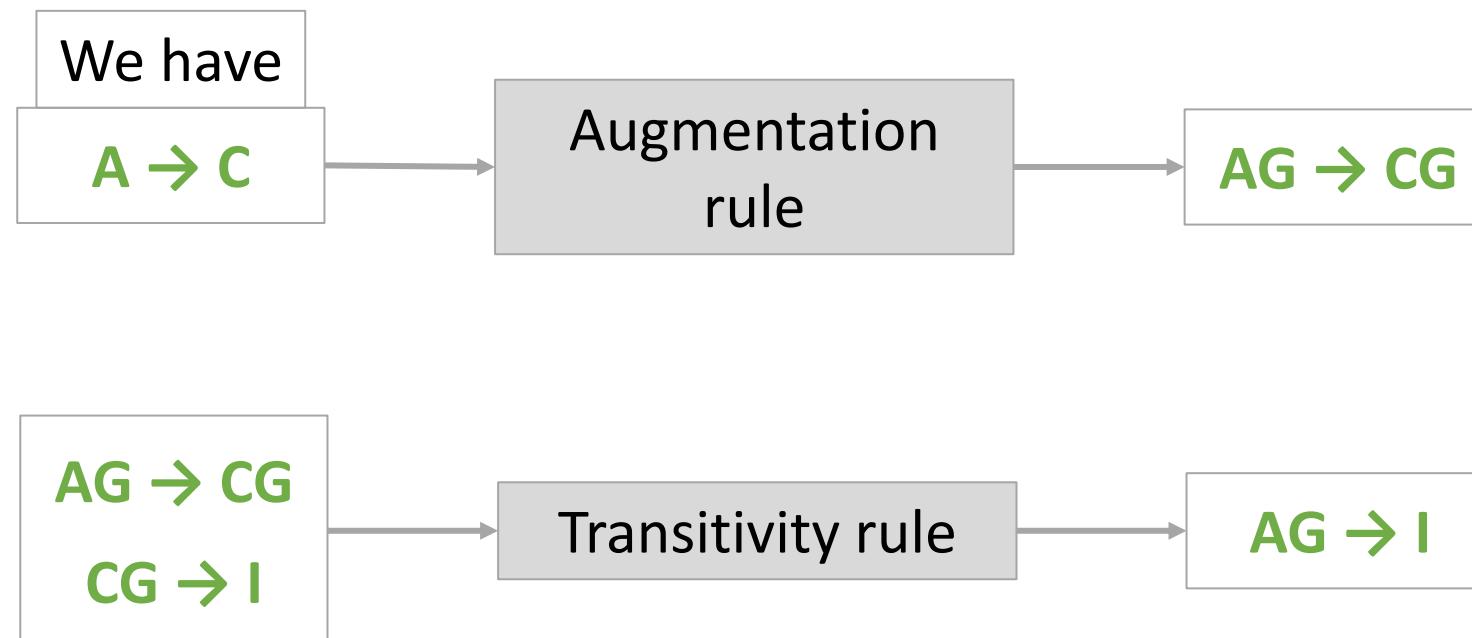
Closure of a set of FDs [Example]

- ▶ Suppose we are given a relation schema $R(A,B,C,G,H,I)$ and the set of functional dependencies are:
 - $F = (A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H)$
- The functional dependency $AG \rightarrow I$ is logically implied.



Closure of a set of FDs [Example]

- ▶ Suppose we are given a relation schema $R(A,B,C,G,H,I)$ and the set of functional dependencies are:
 - $F = (A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H)$
- The functional dependency $AG \rightarrow I$ is logically implied.



Closure of a set of FDs [Example]

- ▶ Suppose we are given a relation schema $R(A,B,C,G,H,I)$ and the set of functional dependencies are:
 - ↳ $F = (A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H)$
- Find out the closure of F .

Several members of F^+ are

$F^+ = (A \rightarrow H, CG \rightarrow HI, AG \rightarrow I)$

Closure of a set of FDs [Example]

- ▶ Compute the closure of the following set F of functional dependencies FDs for relational schema $R = (A, B, C, D, E, F)$:
 - $F = (A \rightarrow B, A \rightarrow C, CD \rightarrow E, CD \rightarrow F, B \rightarrow E)$
- Find out the closure of F.

$A \rightarrow B \text{ & } A \rightarrow C$	Union Rule	$A \rightarrow BC$
$CD \rightarrow E \text{ & } CD \rightarrow F$	Union Rule	$CD \rightarrow EF$
$A \rightarrow B \text{ & } B \rightarrow E$	Transitivity Rule	$A \rightarrow E$
$A \rightarrow C \text{ & } CD \rightarrow E$	Pseudo-transitivity Rule	$AD \rightarrow E$
$A \rightarrow C \text{ & } CD \rightarrow F$	Pseudo-transitivity Rule	$AD \rightarrow F$

$$F^+ = (A \rightarrow BC, CD \rightarrow EF, A \rightarrow E, AD \rightarrow E, AD \rightarrow F)$$

Closure of a set of FDs [Example]

- ▶ Compute the closure of the following set F of functional dependencies FDs for relational schema $R = (A, B, C, D, E)$:
 - $F = (AB \rightarrow C, D \rightarrow AC, D \rightarrow E)$
- Find out the closure of F.

$D \rightarrow AC$	Decomposition Rule	$D \rightarrow A \text{ & } D \rightarrow$
$D \rightarrow AC \text{ & } D \rightarrow E$	Union Rule	$D \rightarrow ACE$

$$F^+ = (D \rightarrow A, D \rightarrow C, D \rightarrow ACE)$$

Closure of attribute sets to decide the Key

What is a closure of attribute sets?

- Given a set of attributes α , the closure of α under F is the **set of attributes that are functionally determined by α under F .**
- It is denoted by α^+ .

What is a closure of attribute sets?



THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

- Given a set of attributes α , the closure of α under F is the **set of attributes that are functionally determined by α under F** .
- It is denoted by α^+ .

Algorithm

→ Algorithm to compute α^+ , the closure of α under F

→ Steps

1. $\text{result} = \alpha$
2. $\text{while } (\text{changes to result}) \text{ do}$
 - for each $\beta \rightarrow \gamma$ in F do
 - begin
 - if $\beta \subseteq \text{result}$ then $\text{result} = \text{result} \cup \gamma$
 - else $\text{result} = \text{result}$
 - end

Closure of attribute sets [Example]

- Consider the relation schema $R = (A, B, C, G, H, I)$.
- For this relation, a set of functional dependencies F can be given as
$$F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$$
- Find out the closure of $(AG)^+$.

Algorithm

→ Algorithm to compute α^+ , the closure of α under F

→ Steps

1. $\text{result} = \alpha$
2. $\text{while } (\text{changes to result}) \text{ do}$
 - for each $\beta \rightarrow \gamma$ in F do
 - begin
 - if $\beta \subseteq \text{result}$ then $\text{result} = \text{result} \cup \gamma$
 - else $\text{result} = \text{result}$

end

Step 1.

$$\text{result} = \alpha \Rightarrow \text{result} = AG$$

$A \rightarrow B$	$A \subseteq AG$	$\text{result} = ABG$
$A \rightarrow C$	$A \subseteq ABG$	$\text{result} = ABCG$
$CG \rightarrow H$	$CG \subseteq ABCG$	$\text{result} = ABCGH$
$CG \rightarrow I$	$CG \subseteq ABCGH$	$\text{result} = ABCGHI$
$B \rightarrow H$	$B \subseteq ABCGHI$	$\text{result} = ABCGHI$

$$AG^+ = ABCGHI$$

Closure of attribute sets [Exercise]



- Given functional dependencies (FDs) for relational schema $R = (A, B, C, D, E)$:
- $F = \{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$
 - Find Closure for A
 - Find Closure for CD
 - Find Closure for B
 - Find Closure for BC
 - Find Closure for E

Answer

$A^+ = ABCDE$

$CD^+ = ABCDE$

$B^+ = BD$

$BC^+ = ABCDE$

$E^+ = ABCDE$

X	Y	Z
1	1	1
2	1	2
2	1	3
3	1	1

Which of the following is Functional Dependency?

1. $X \rightarrow Y$
2. $XY \rightarrow Z$
3. $YZ \rightarrow X$

R(A,B,C,D,E,G)
{
A \rightarrow B
B \rightarrow C
D \rightarrow E
E \rightarrow G
}

The following functional dependencies are given:

$$AB \rightarrow CD, AF \rightarrow D, DE \rightarrow F, C \rightarrow G, F \rightarrow E, G \rightarrow A.$$

Which one of the following options is false?

- (A) $\{CF\}^+ = \{ACDEFG\}$ (C) $\{AF\}^+ = \{ACDEFG\}$
(B) $\{BG\}^+ = \{ABCDG\}$ (D) $\{AB\}^+ = \{ABCDG\}$

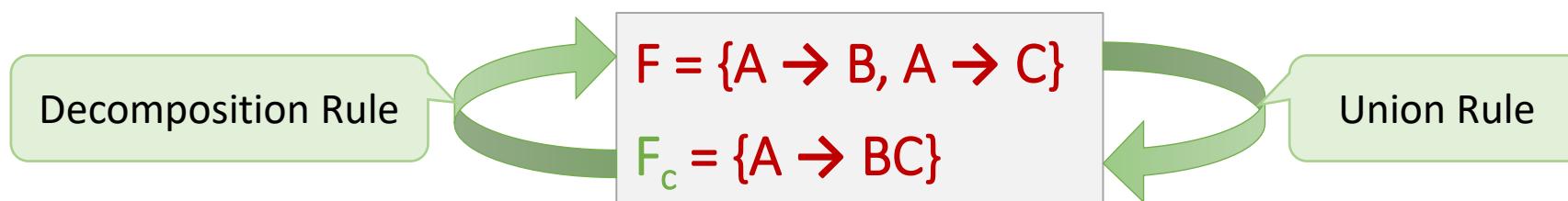
Canonical cover

What is extraneous attributes?

- Let us consider a relation R with schema $R = (A, B, C)$ and set of functional dependencies FDs $F = \{ AB \rightarrow C, A \rightarrow C \}$.
- In $AB \rightarrow C$, **B is extraneous attribute**. The reason is, there is another FD $A \rightarrow C$, which means when **A alone can determine C**, the use of B is unnecessary (extra).
- An attribute of a functional dependency is said to be extraneous if we can **remove it without changing the closure of the set of functional dependencies**.

What is canonical cover?

- A canonical cover of F is a **minimal set of functional dependencies** equivalent to F , having **no redundant dependencies or redundant parts of dependencies**.
- It is denoted by F_c
- A canonical cover for F is a set of dependencies F_c such that
 - F logically implies all dependencies in F_c and
 - F_c logically implies all dependencies in F and
 - **No** functional dependency in F_c contains an **extraneous attribute** and
 - Each **left side** of functional dependency in F_c is **unique**.



Algorithm to find canonical cover

- *Repeat*

- Use the **union rule** to replace any dependencies in $F \alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1\beta_2$
- Find a functional dependency $\alpha \rightarrow \beta$ with an **extraneous attribute** either in α or in β

/* Note: test for extraneous attributes done using F_c , not F */

- If an **extraneous attribute is found, delete it** from $\alpha \rightarrow \beta$

- *until* F does not change

/* Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied */

Canonical cover [Example]

- ▶ Consider the relation schema $R = (A, B, C)$ with FDs

$$F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$$

- ▶ Find canonical cover.

- Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$ (Union Rule)
 - Set is $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$
- A is extraneous in $AB \rightarrow C$
 - Check if the result of deleting A from $AB \rightarrow C$ is implied by the other dependencies
 - Yes: in fact, $B \rightarrow C$ is already present
 - Set is $\{A \rightarrow BC, B \rightarrow C\}$
- C is extraneous in $A \rightarrow BC$
 - Check if $A \rightarrow C$ is logically implied by $A \rightarrow B$ and the other dependencies
 - Yes: using transitivity on $A \rightarrow B$ and $B \rightarrow C$.
 - The canonical cover is: $A \rightarrow B, B \rightarrow C$

Canonical cover [Example]

- ▶ Consider the relation schema $R = (A, B, C, D, E, F)$ with FDs

$$F = \{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$$

- ▶ Find canonical cover.

- The left side of each FD in F is unique.
- Also none of the attributes in the left side or right side of any of the FDs is extraneous.
- Therefore the canonical cover F_c is equal to F .
- $F_c = \{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$

What is an anomaly in database design?

- Anomalies are **problems that can occur in poorly planned, un-normalized database** where all the data are stored in one table.
- There are three types of anomalies that can arise in the database because of redundancy are
 - Insert anomaly
 - Delete anomaly
 - Update / Modification anomaly

Insert anomaly

- Consider a relation Emp_Dept(EID, Ename, City, DID, Dname, Manager) EID as a primary key

Emp_Dept					
EID	Ename	City	DID	Dname	Manager
1	Raj	Rajkot	1	CE	Shah
2	Meet	Surat	1	CE	Shah
NULL	NULL	NULL	2	IT	NULL

An insert anomaly occurs when certain attributes cannot be inserted into the database without the presence of another attribute.

Want to insert new department detail
(IT)

- Suppose a **new department (IT) has been started** by the organization but **initially there is no employee appointed** for that department.
- We **want to insert that department detail** in Emp_Dept table.
- But the **tuple for this department cannot be inserted** into this table as the **EID will have NULL value, which is not allowed because EID is primary key**.
- This kind of problem in the relation where some tuple cannot be inserted is known as **insert anomaly**.

Delete anomaly

- Consider a relation Emp_Dept(EID, Ename, City, DID, Dname, Manager) EID as a primary key

Emp_Dept					
EID	Ename	City	DID	Dname	Manager
1	Raj	Rajkot	1	CE	Shah
2	Meet	Surat	1	CE	Shah
3	Jay	Baroda	2	IT	Dave

A delete anomaly exists when **certain attributes are lost because of the deletion of another attribute.**

Want to delete (Jay)
employee's detail

- Now consider **there is only one employee in some department (IT) and that employee leaves the organization.**
- So we **need to delete tuple of that employee (Jay).**
- But in addition to that **information about the department also deleted.**
- This kind of problem in the relation where deletion of some tuples can lead to loss of some other data not intended to be removed is known as delete anomaly.

Update anomaly

- Consider a relation Emp_Dept(EID, Ename, City, Dname, Manager) EID as a primary key

Emp_Dept				
EID	Ename	City	Dname	Manager
1	Raj	Rajkot	CE	Sah
2	Meet	Surat	CE	Shah
3	Jay	Baroda	IT	Dave
4	Hari	Rajkot	IT	Dave

An update anomaly exists **when one or more records (instance) of duplicated data is updated, but not all.**

Want to update manager of CE department

- Suppose the **manager of a (CE) department has changed**, this requires that the **Manager in all the tuples corresponding to that department must be changed** to reflect the new status.
- If we **fail to update all the tuples of given department**, then **two different records of employee working in the same department might show different Manager lead to inconsistency** in the database.

How to deal with insert, delete and update anomaly

Emp_Dept					
EID	Ename	City	DID	Dname	Manager
1	Raj	Rajkot	1	CE	Shah
2	Meet	Surat	1	CE	Shah
3	Jay	Baroda	2	IT	Dave
NULL	NULL	NULL	3	EC	NULL

Emp			
EID	Ename	City	DID
1	Raj	Rajkot	1
2	Meet	Surat	1
3	Jay	Baroda	2

Dept		
DID	Dname	Manager
1	CE	Shah
2	IT	Dave
3	EC	NULL

Such type of anomalies in the database design can be solved by using **normalization**.

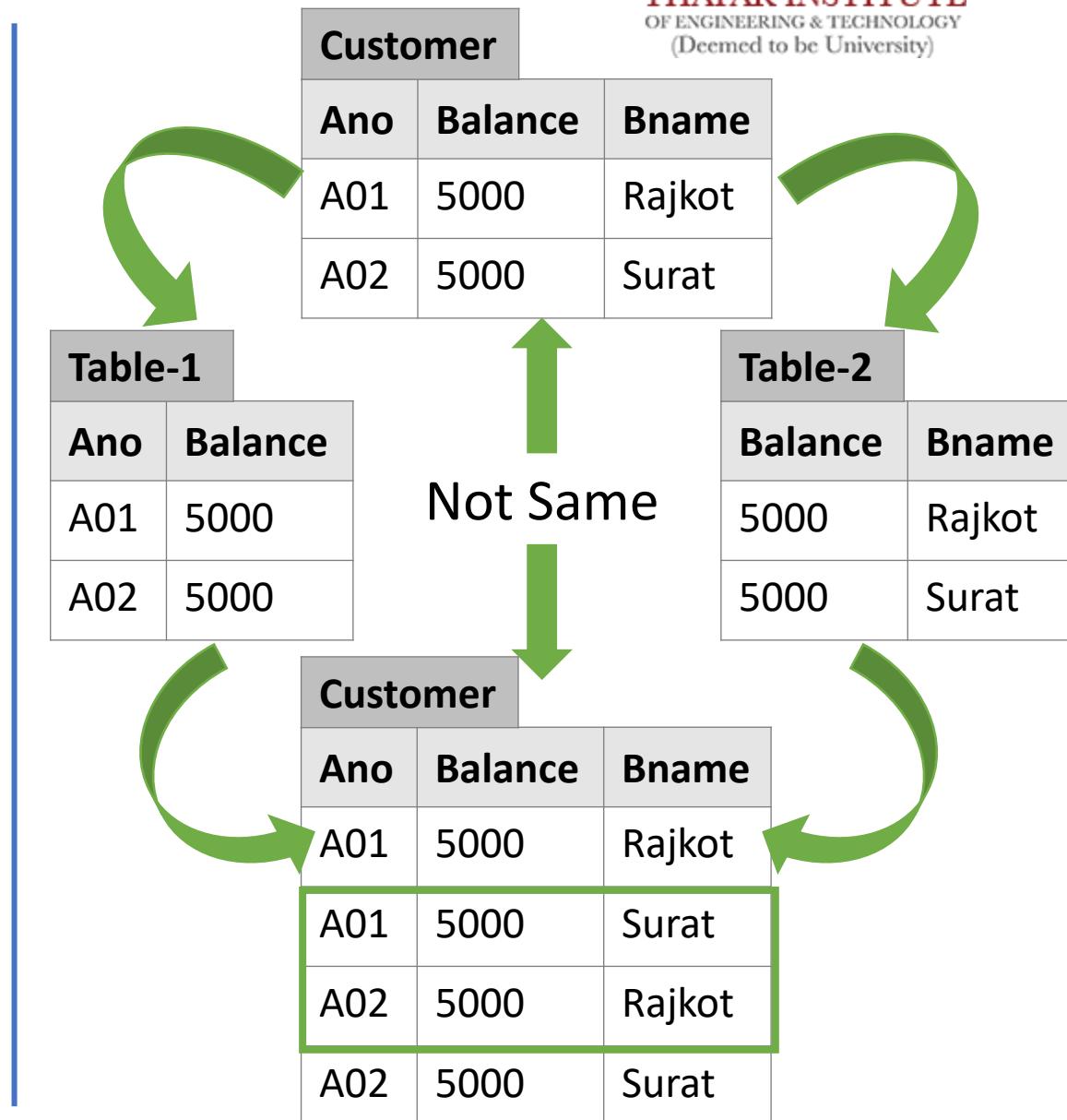
Decomposition

What is decomposition?

- Decomposition is the **process of breaking down given relation** into **two or more relations**.
- Relation R is replaced by two or more relations in such a way that:
 - Each new relation contains a **subset** of the **attributes of R**
 - Together, they all **include all tuples** and **attributes of R**
- **Types of decomposition**
 - **Lossy decomposition**
 - **Lossless decomposition (non-loss decomposition)**

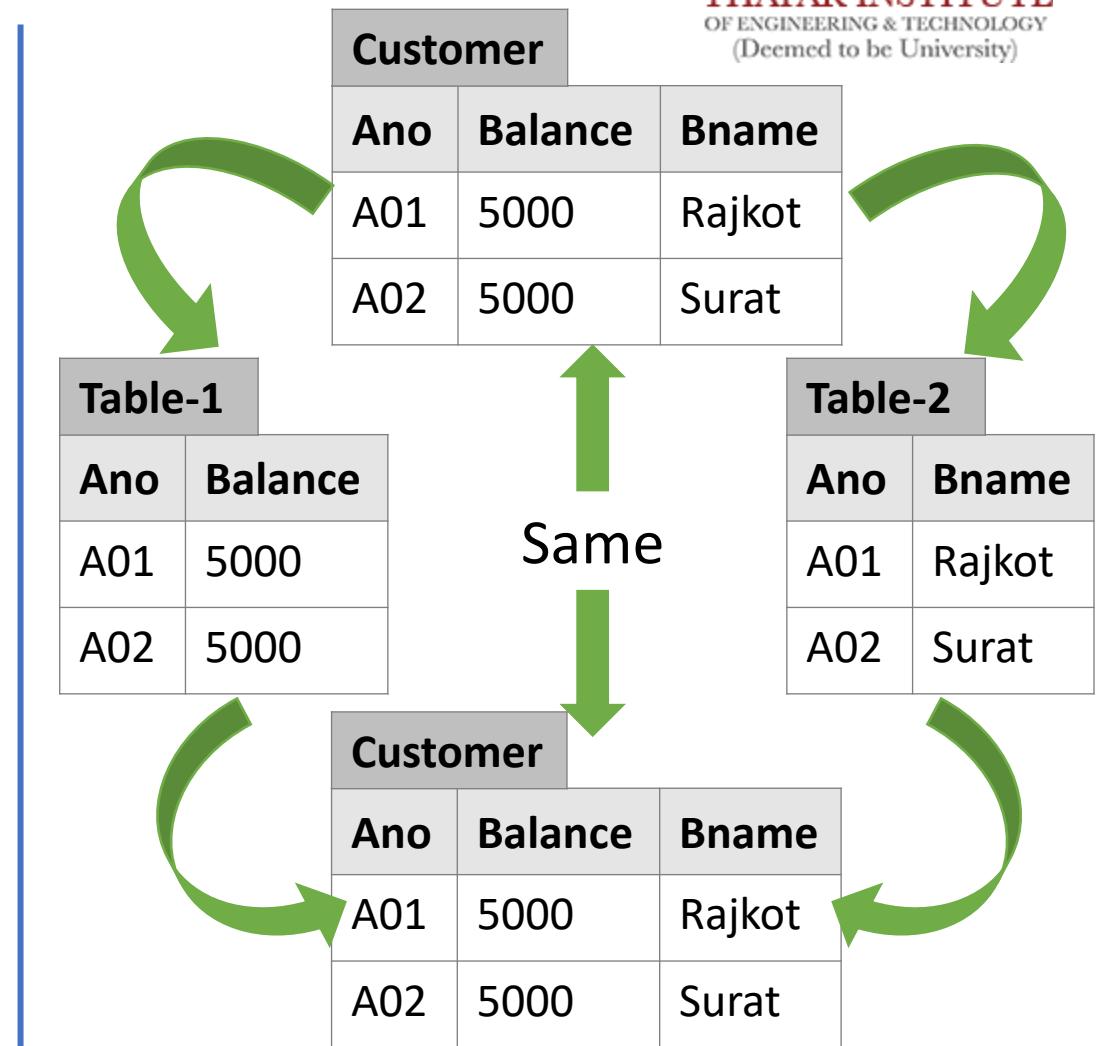
Lossy decomposition

- The decomposition of relation R into R₁ and R₂ is lossy when the join of R₁ and R₂ does not yield the same relation as in R.
- This is also referred as **lossy-join decomposition**.
- The **disadvantage** of such kind of decomposition is that **some information is lost during retrieval of original relation**.
- From practical point of view, **decomposition should not be lossy decomposition**.



Lossless decomposition

- The decomposition of relation R into R₁ and R₂ is lossless when the **join** of R₁ and R₂ produces the same relation as in R.
- This is also referred as a **non-additive (non-loss) decomposition**.
- All decompositions must be lossless.



Normalization and normal forms

What is normalization?



- Normalization is the **process of removing redundant data** from tables **to improve data integrity, scalability and storage efficiency**.
 - data integrity (completeness, accuracy and consistency of data)
 - scalability (ability of a system to continue to function well in a growing amount of work)
 - storage efficiency (ability to store and manage data that consumes the least amount of space)
- What we do in normalization?
 - Normalization generally involves **splitting an existing table into multiple (more than one) tables**, which can be **re-joined or linked** each time a query is issued (executed).

How many normal forms are there?



THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

- Normal forms:

- 1NF (First normal form)
- 2NF (Second normal form)
- 3NF (Third normal form)
- BCNF (Boyce–Codd normal form)
- 4NF (Forth normal form)
- 5NF (Fifth normal form)

As we move from 1NF to 5NF **number of tables** and **complexity increases** but **redundancy decreases**.

Normal forms

1NF (First Normal Form)

1NF (First Normal Form)



- Conditions for 1NF

Each **cells of a table should contain a single value.**

- A relation R is in first normal form (1NF) if and only if it **does not contain any composite attribute or multi-valued attributes or their combinations.**

OR

- A relation R is in first normal form (1NF) if and only if **all underlying domains contain atomic values only.**

1NF (First Normal Form) [Example - Composite attribute]



THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Customer		
CID	Name	Address
C01	Raju	Jamnagar Road, Rajkot
C02	Mitesh	Nehru Road, Jamnagar
C03	Jay	C.G Road, Ahmedabad

- In customer relation **address is composite attribute** which is further divided into sub-attributes as “Road” and “City”.
- So customer relation is not in 1NF.

- **Problem:** It is **difficult to retrieve the list of customers living in 'Jamnagar' city** from customer table.
- The reason is that **address attribute is composite attribute** which **contains road name as well as city name in single cell**.
- It is possible that **city name word is also there in road name**.
- In our example, 'Jamnagar' word occurs in both records, in first record it is a part of road name and in second one it is the name of city.

1NF (First Normal Form) [Example - Composite attribute]

Customer		
<u>CID</u>	Name	Address
C01	Raju	Jamnagar Road, Rajkot
C02	Mitesh	Nehru Road, Jamnagar
C03	Jay	C.G Road, Ahmedabad



Customer			
<u>CID</u>	Name	Road	City
C01	Raju	Jamnagar Road	Rajkot
C02	Mitesh	Nehru Road	Jamnagar
C03	Jay	C.G Road	Ahmedabad

- **Solution:** Divide composite attributes into number of sub-attributes and insert value in proper sub-attribute.

Exercise

Convert below relation into 1NF (First Normal Form)

Person		
<u>PID</u>	Full_Name	City
P01	Raju Maheshbhai Patel	Rajkot

1NF (First Normal Form) [Example - Multivalued attribute]

Student		
Rno	Name	FailedinSubjects
101	Raju	DS, DBMs
102	Mitesh	DBMS, DS
103	Jay	DS, DBMS, DE
104	Jeet	DBMS, DE, DS
105	Harsh	DE, DBMS, DS
106	Neel	DE, DBMS

- In student relation **FailedinSubjects attribute is a multi-valued attribute** which can store more than one values.
- So above relation is not in 1NF.

- **Problem:** It is difficult to retrieve the **list of students failed in 'DBMS' as well as 'DS' but not in other subjects** from student table.
- The reason is that FailedinSubjects attribute is multi-valued attribute so it contains more than one value.

1NF (First Normal Form) [Example - Multivalued attribute]

Student		
<u>Rno</u>	Name	FailedinSubjects
101	Raju	DS, DBMs
102	Mitesh	DBMS, DS
103	Jay	DS, DBMS, DE
104	Jeet	DBMS, DE, DS
105	Harsh	DE, DBMS, DS
106	Neel	DE, DBMS



Student	
<u>Rno</u>	Name
101	Raju
102	Mitesh
103	Jay
104	Jeet
105	Harsh
106	Neel

Result		
<u>RID</u>	Rno	Subject
1	101	DS
2	101	DBMS
3	102	DBMS
4	102	DS
5	103	DS
...

- **Solution:** Split the table into two tables in such as way that
 - the **first table contains all attributes except multi-valued attribute** with same primary key and
 - **second table contains multi-valued attribute** and **place a primary key** in it.
 - **insert the primary key of first table in the second table as a foreign key.**

Normal forms

2NF (Second Normal Form)

2NF (Second Normal Form)



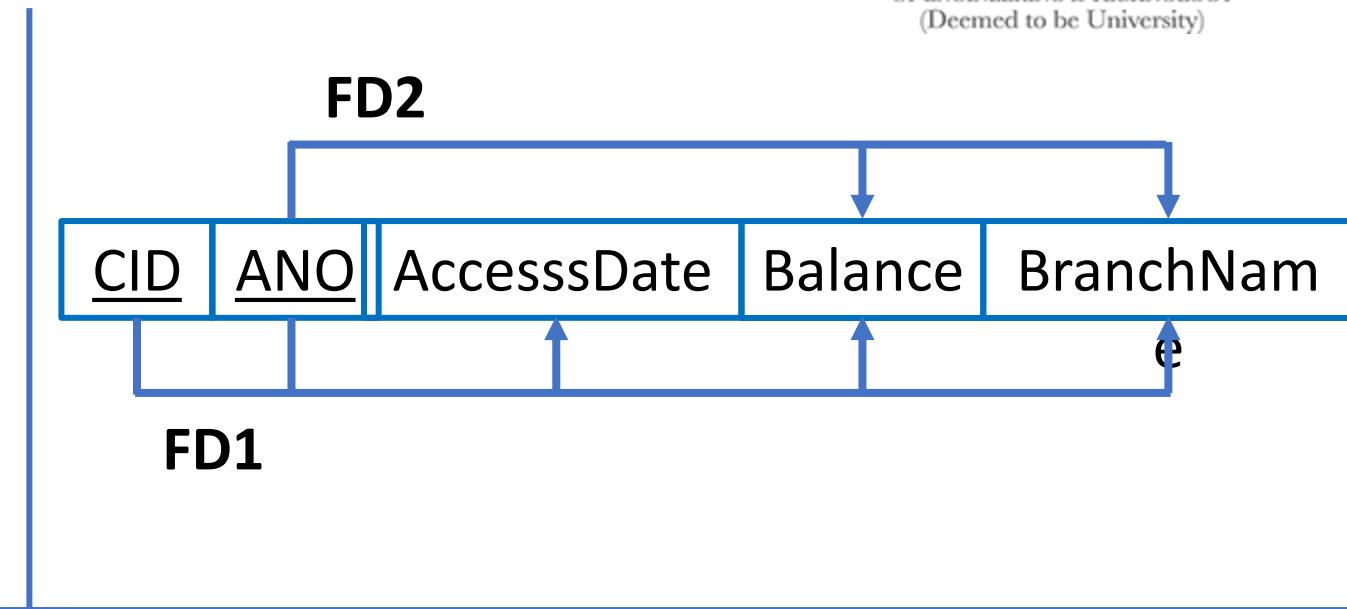
- Conditions for 2NF

It is **in 1NF** and each **table should contain a single primary key**.

- A relation R is in second normal form (2NF)
 - if and only if it is in **1NF** and
 - **every non-primary key attribute is fully dependent on the primary key**
- A relation R is in second normal form (2NF)
 - if and only if it is in **1NF** and
 - **no non-primary key attribute is partially dependent on the primary key**

2NF (Second Normal Form) [Example]

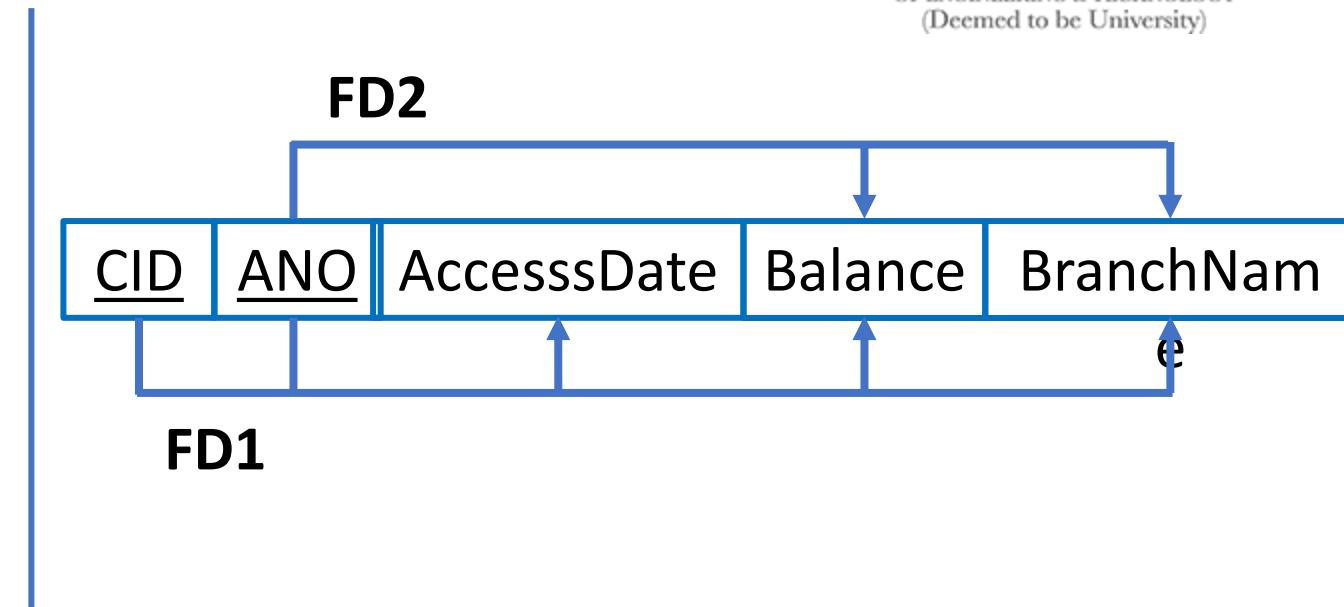
Customer				
CID	ANO	AccessDate	Balance	BranchName
C01	A01	01-01-2017	50000	Rajkot
C02	A01	01-03-2017	50000	Rajkot
C01	A02	01-05-2017	25000	Surat
C03	A02	01-07-2017	25000	Surat



- **FD1:** {CID, ANO} → {AccesssDate, Balance, BranchName}
- **FD2:** ANO → {Balance, BranchName}
- **Balance and BranchName are partial dependent on primary key (CID + ANO).**
So customer relation is not in 2NF.

2NF (Second Normal Form) [Example]

Customer				
CID	ANO	AccessDate	Balance	BranchName
C01	A01	01-01-2017	50000	Rajkot
C02	A01	01-03-2017	50000	Rajkot
C01	A02	01-05-2017	25000	Surat
C03	A02	01-07-2017	25000	Surat



- **Problem:** For example, in case of a joint account multiple (more than one) customers have common (one) accounts.
- If an account '**A01**' is operated jointly by two customers says '**C01**' and '**C02**' then **data** values for attributes **Balance** and **BranchName** will be **duplicated** in **two different tuples** of customers '**C01**' and '**C02**'.

2NF (Second Normal Form) [Example]

Customer				
<u>CID</u>	<u>ANO</u>	AccessDate	Balance	BranchName
C01	A01	01-01-2017	50000	Rajkot
C02	A01	01-03-2017	50000	Rajkot
C01	A02	01-05-2017	25000	Surat
C03	A02	01-07-2017	25000	Surat

Table-1		
<u>ANO</u>	Balance	BranchName
A01	50000	Rajkot
A02	25000	Surat



Table-2		
<u>CID</u>	<u>ANO</u>	AccessDate
C01	A01	01-01-2017
C02	A01	01-03-2017
C01	A02	01-05-2017
C03	A02	01-07-2017

- **Solution: Decompose relation** in such a way that **resultant relations do not have any partial FD**.
 - Remove **partial dependent attributes** from the relation that violates 2NF.
 - Place them in **separate relation** along with the **prime attribute on which they are fully dependent**.
 - The **primary key of new relation** will be the **attribute on which it is fully dependent**.
 - Keep **other attributes same** as in that table with the **same primary key**.

Normal forms

3NF (Third Normal Form)

3NF (Third Normal Form)



- Conditions for 3NF

It is in **2NF** and there is no transitive dependency.

(Transitive dependency???) $A \rightarrow B$ & $B \rightarrow C$ then $A \rightarrow C$

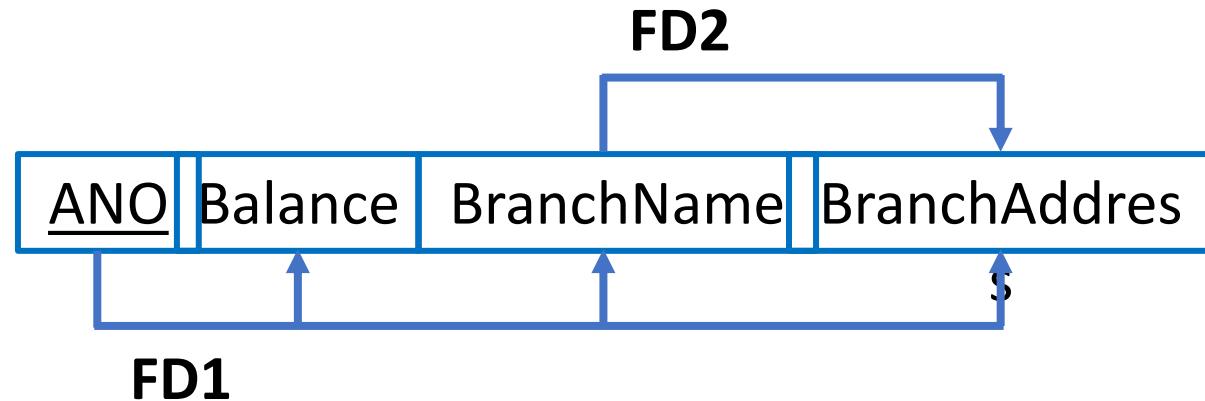
- A relation R is in third normal form (3NF)
 - if and only if it is in **2NF** and
 - **every non-key attribute is non-transitively dependent on the primary key**

OR

- A relation R is in third normal form (3NF)
 - if and only if it is in **2NF** and
 - **no non-key attribute is transitively dependent on the primary key**

3NF (Third Normal Form) [Example]

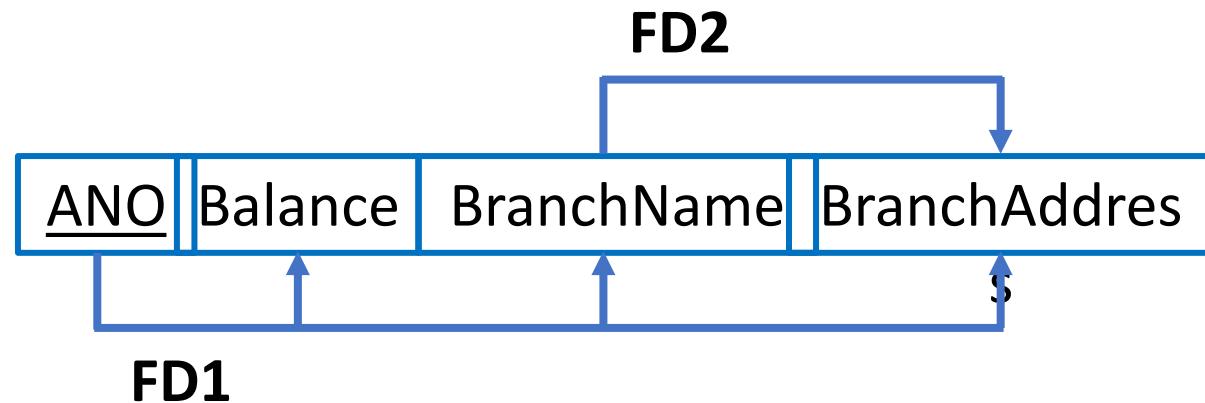
Customer			
<u>ANO</u>	Balance	BranchName	BranchAddress
A01	50000	Rajkot	Kalawad road
A02	40000	Rajkot	Kalawad Road
A03	35000	Surat	C.G Road
A04	25000	Surat	C.G Road



- **FD1:** ANO → {Balance, BranchName, BranchAddress}
- **FD2:** BranchName → BranchAddress
- So ANO → BranchAddress (Using Transitivity rule)
- **BranchAddress is transitive depend on primary key (ANO).** So customer relation is not in 3NF.

3NF (Third Normal Form) [Example]

Customer			
<u>ANO</u>	Balance	BranchName	BranchAddress
A01	50000	Rajkot	Kalawad road
A02	40000	Rajkot	Kalawad Road
A03	35000	Surat	C.G Road
A04	25000	Surat	C.G Road



- **Problem:** In this relation, **branch address will be stored repeatedly** for each account of the same branch which **occupies more space**.

3NF (Third Normal Form) [Example]

Customer			
<u>ANO</u>	Balance	BranchName	BranchAddress
A01	50000	Rajkot	Kalawad road
A02	40000	Rajkot	Kalawad Road
A03	35000	Surat	C.G Road
A04	25000	Surat	C.G Road



Table-1	
<u>BranchName</u>	BranchAddress
Rajkot	Kalawad road
Surat	C.G Road

Table-2		
<u>ANO</u>	Balance	BranchName
A01	50000	Rajkot
A02	40000	Rajkot
A03	35000	Surat
A04	25000	Surat

- Solution: Decompose relation in such a way that resultant relations do not have any transitive FD.**
 - Remove transitive dependent attributes from the relation that violates 3NF.
 - Place them in a new relation along with the non-prime attributes due to which transitive dependency occurred.
 - The primary key of the new relation will be non-prime attributes due to which transitive dependency occurred.
 - Keep other attributes same as in the table with same primary key and add prime attributes of other relation into it as a foreign key.

Normal forms

BCNF (Boyce-Codd Normal Form)

BCNF (Boyce-Codd Normal Form)

- Conditions for BCNF

BCNF is **based on the concept of a determinant.**

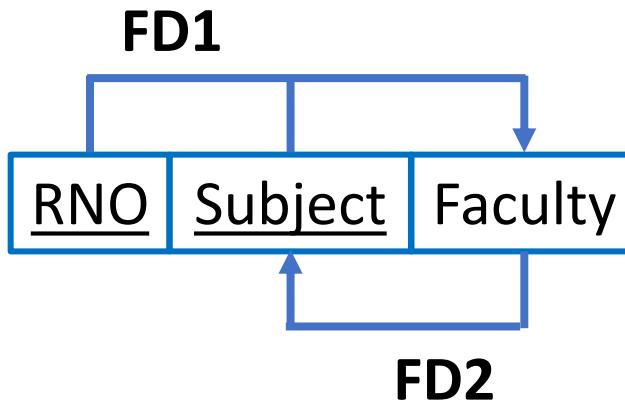


It is in **3NF** and **every determinant should be primary key.**

- A relation R is in Boyce-Codd normal form (BCNF)
 - if and only if it is in 3NF and
 - for every functional dependency $X \rightarrow Y$, X should be the primary key/candidate key of the table.
OR
- A relation R is in Boyce-Codd normal form (BCNF)
 - if and only if it is in 3NF and
 - every prime key attribute is non-transitively dependent on the primary key
OR
- A relation R is in Boyce-Codd normal form (BCNF)
 - if and only if it is in 3NF and
 - no prime key attribute is transitively dependent on the primary key

BCNF (Boyce-Codd Normal Form) [Example]

Student		
RNO	Subject	Faculty
101	DS	Patel
102	DBMS	Shah
103	DS	Jadeja
104	DBMS	Dave
105	DBMS	Shah
102	DS	Patel
101	DBMS	Dave
105	DS	Jadeja



- **FD1:** RNO, Subject → Faculty
- **FD2:** Faculty → Subject
- So {RNO, Subject} → Subject (Transitivity rule)

In FD2, **determinant is Faculty which is not a primary key**. So student table is not in BCNF.

Problem: In this relation **one student can learn more than one subject with different faculty** then **records will be stored repeatedly for each student, language and faculty combination** which **occupies more space**.

- Here, one faculty teaches only one subject, but a subject may be taught by more than one faculty.
- A student can learn a subject from only one faculty.

BCNF (Boyce-Codd Normal Form) [Example]

Student		
<u>RNO</u>	<u>Subject</u>	<u>Faculty</u>
101	DS	Patel
102	DBMS	Shah
103	DS	Jadeja
104	DBMS	Dave
105	DBMS	Shah
102	DS	Patel
101	DBMS	Dave
105	DS	Jadeja



Table-1	
<u>Faculty</u>	<u>Subject</u>
Patel	DS
Shah	DBMS
Jadeja	DS
Dave	DBMS

Table-2	
<u>RNO</u>	<u>Faculty</u>
101	Patel
102	Shah
103	Jadeja
104	Dave
105	Shah
102	Patel
101	Dave
105	Jadeja

- **Solution:** Decompose relation in such a way that resultant relations do not have any transitive FD.
 - Remove transitive dependent prime attribute from relation that violates BCNF.
 - Place them in separate new relation along with the non-prime attribute due to which transitive dependency occurred.
 - The primary key of new relation will be this non-prime attribute due to which transitive dependency occurred.
 - Keep other attributes same as in that table with same primary key and add a prime attribute of other relation into it as a foreign key.

Normal forms

4NF (Forth Normal Form)

Multivalued dependency (MVD)



- For a dependency $X \rightarrow Y$, if **for a single value of X, multiple values of Y exists**, then the **table may have multi-valued dependency**.

Student		
<u>RNO</u>	<u>Subject</u>	<u>Faculty</u>
101	DS	Patel
101	DBMS	Patel
101	DS	Shah
101	DBMS	Shah

- Multivalued dependency (MVD) is denoted by $\rightarrow\rightarrow$
- Multivalued dependency (MVD) is represented as $X \rightarrow\rightarrow Y$

Conditions for MVD :

Any attribute say a multiple define another attribute b ; if any legal relation $r(R)$, for all pairs of tuples t_1 and t_2 in r , such that,

$$t_1[a] = t_2[a]$$

Then there exists t_3 and t_4 in r such that.

$$t_1[a] = t_2[a] = t_3[a] = t_4[a]$$

$$t_1[b] = t_3[b];$$

$$t_2[b] = t_4[b];$$

$$t_1 = t_4; t_2 = t_3$$

Then multivalued (MVD) dependency exists.

Multivalued Dependencies:

- Suppose that we have a relation with attributes *course*, *teacher*, and *book*, which we denote as **CTB**. The meaning of a tuple is that teacher **T** can teach course **C**, and book **B** is a recommended text for the course.
- There are no FDs; the key is **CTB**. However, the recommended texts for a course are independent of the instructor. The instance shown in Table:

<i>course</i>	<i>teacher</i>	<i>book</i>
Physics101	Green	Mechanics
Physics101	Green	Optics
Physics101	Brown	Mechanics
Physics101	Brown	Optics
Math301	Green	Mechanics
Math301	Green	Vectors
Math301	Green	Geometry

BCNF Relation with Redundancy That Is Revealed by MVDs

Multivalued Dependencies:

There are three points to note here:

- The relation schema CTB is in BCNF; thus we would not consider decomposing it further if we looked only at the FDs that hold over CTB .
- There is redundancy. The fact that Green can teach Physics101 is recorded once per recommended text for the course. Similarly, the fact that Optics is a text for Physics101 is recorded once per potential teacher.
- The redundancy can be eliminated by decomposing CTB into CT and CB .

course	teacher	book
Physics101	Green	Mechanics
Physics101	Green	Optics
Physics101	Brown	Mechanics
Physics101	Brown	Optics
Math301	Green	Mechanics
Math301	Green	Vectors
Math301	Green	Geometry

BCNF Relation with Redundancy That Is Revealed by MVDs

- The redundancy in this example is due to the constraint that the text books for a course are independent of the instructors, which cannot be expressed in terms of FDs.
- This constraint is an example of a ***multivalued dependency***, or MVD.

Note:

- *The total number of attributes should be more than two.*
- *If there exists 3 attributes, then 2 attributes must be independent of each other.*

Multivalued Dependencies:

■ *Persons(Man, Phones, Dog_Like)*

Person :			Meaning of the tuples
Man(M)	Phones(P)	Dogs_Like(D)	
M1	P1/P2	D1/D2	M1 have phones P1 and P2, and likes the dogs D1 and D2.
M2	P3	D2	M2 have phones P3, and likes the dog D2.
Key : MPD			

There are no non trivial FDs because all attributes are combined forming Candidate Key i.e. MDP. In the above relation, two multivalued dependencies exists –

- **Man →→ Phones**
- **Man →→ Dogs_Like**

A man's phone are independent of the dogs they like. But after converting the above relation in Single Valued Attribute, each of a man's phones appears with each of the dogs they like in all combinations.

Post 1NF Normalization

Man(M)	Phones(P)	Dogs_Likes(D)
M1	P1	D1
M1	P2	D2
M2	P3	D2
M1	P1	D2
M1	P2	D1

Multivalued Dependencies:

- Let R be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The **multivalued dependency**

$$\alpha \rightarrow\!\!\!\rightarrow \beta$$

holds on R if in any legal relation $r(R)$, for all pairs of tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in r such that:

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$

$$t_3[\beta] = t_1[\beta]$$

$$t_3[R - \beta] = t_2[R - \beta]$$

$$t_4[\beta] = t_2[\beta]$$

$$t_4[R - \beta] = t_1[R - \beta]$$

Example: A relation of university courses, the books recommended for the course, and the lecturers who will be teaching the course:

- $\text{course} \rightarrow\!\!\!\rightarrow \text{book}$
- $\text{course} \rightarrow\!\!\!\rightarrow \text{lecturer}$

Test: $\text{course} \rightarrow\!\!\!\rightarrow \text{book}$

<u>Course</u>	<u>Book</u>	<u>Lecturer</u>	<u>Tuples</u>
AHA	Silberschatz	John D	t1
AHA	Nederpelt	William M	t2
AHA	Silberschatz	William M	t3
AHA	Nederpelt	John D	t4
AHA	Silberschatz	Christian G	
AHA	Nederpelt	Christian G	
OSO	Silberschatz	John D	
OSO	Silberschatz	William M	

4NF (Forth Normal Form)

- Conditions for 4NF
- A relation R is in fourth normal form (4NF)
 - if and only if it is in BCNF and
 - has no multivalued dependencies

Student		
<u>RNO</u>	<u>Subject</u>	<u>Faculty</u>
101	DS	Patel
101	DBMS	Patel
101	DS	Shah
101	DBMS	Shah



Subject	
<u>RNO</u>	<u>Subject</u>
101	DS
101	DBMS

Faculty	
<u>RNO</u>	<u>Faculty</u>
101	Patel
101	Shah

Consider R(X,Y,Z)

Suppose that X →→ Y and by symmetry X →→ Z

Then, decomposition D = (XY, XZ) of R should be lossless

- Above student table has multivalued dependency. So student table is not in 4NF.

Functional dependency & Multivalued dependency

- A table can have both functional dependency as well as multi-valued dependency together.

- RNO → Address
- RNO →→ Subject
- RNO →→ Faculty

Student			
<u>RNO</u>	<u>Address</u>	<u>Subject</u>	<u>Faculty</u>
101	C. G. Road, Rajkot	DS	Patel
101	C. G. Road, Rajkot	DBMS	Patel
101	C. G. Road, Rajkot	DS	Shah
101	C. G. Road, Rajkot	DBMS	Shah



Subject	
<u>RNO</u>	<u>Subject</u>
101	DS
101	DBMS

Faculty	
<u>RNO</u>	<u>Faculty</u>
101	Patel
101	Shah

Address	
<u>RNO</u>	<u>Address</u>
101	C. G. Road, Rajkot

Normal forms

5NF (Fifth Normal Form)

Join Dependency- 5NF



THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

A join dependency is a further generalization of MVDs. A **join dependency** (JD) $\bowtie \{R_1, \dots, R_n\}$ is said to hold over a relation R if R_1, \dots, R_n is a lossless-join decomposition of R .

An MVD $X \rightarrow\rightarrow Y$ over a relation R can be expressed as the join dependency $\bowtie \{XY, X(R-Y)\}$. As an example, in the *CTB* relation, the MVD $C \rightarrow\rightarrow T$ can be expressed as the join dependency $\bowtie \{CT, CB\}$.

Unlike FDs and MVDs, there is no set of sound and complete inference rules for JDs.

5NF (Fifth Normal Form)



- Conditions for 5NF
- A relation R is in fifth normal form (5NF)
 - if and only if it is in **4NF** and
 - it **cannot have a lossless decomposition in to any number of smaller tables (relations).**

Student_Result				
RID	RNO	Name	Subject	Result
1	101	Raj	DBMS	Pass
2	101	Raj	DS	Pass
3	101	Raj	DF	Pass
4	102	Meet	DBMS	Pass
5	102	Meet	DS	Fail
6	102	Meet	DF	Pass
7	103	Suresh	DBMS	Fail
8	103	Suresh	DS	Pass

Student_Result relation is **further decomposed** into sub-relations. So the above relation is **not in 5NF.**

5NF (Fifth Normal Form)

- Conditions for 5NF
- A relation R is in fifth normal form (5NF)
 - if and only if it is in **4NF** and
 - it **cannot have a lossless decomposition in to any number of smaller tables** (relations).

Student_Result				
<u>RID</u>	RNO	Name	Subject	Result
1	101	Raj	DBMS	Pass
2	101	Raj	DS	Pass
3	101	Raj	DF	Pass
4	102	Meet	DBMS	Pass
5	102	Meet	DS	Fail
6	102	Meet	DF	Pass
7	103	Suresh	DBMS	Fail
8	103	Suresh	DS	Pass

Student	
<u>RNO</u>	Name
101	Raj
102	Meet
103	Suresh

Subject	
<u>SID</u>	Name
1	DBMS
2	DS
3	DF

Result			
<u>RID</u>	RNO	SID	Result
1	101	1	Pass
2	101	2	Pass
3	101	3	Pass
4	102	1	Pass
5	102	2	Fail
6	102	3	Pass
7	103	1	Fail
8	103	2	Pass



None of the above relations can be further decomposed into sub-relations. So the above database is in 5NF.

Example

- A software contract and consultancy firm maintains details of all the various projects in which its employees are currently involved. These details comprise: Employee Number, Employee Name, Date of Birth, Department Code, Department Name, Project Code, Project Description, Project Supervisor.
- Assume the following:
 - Each employee number is unique.
 - Each department has a single department code.
 - Each project has a single code and supervisor.
 - Each employee may work on one or more projects.
 - Employee names need not necessarily be unique.
 - Project Code, Project Description and Project Supervisor are repeating fields.
 - Normalize this data to Third Normal Form.

How to normalize database?

- A software contract and consultancy firm maintains details of all the various projects in which its employees are currently involved. These details comprise: Employee Number, Employee Name, Date of Birth, Department Code, Department Name, Project Code, Project Description, Project Supervisor.

UNF

Employee Number	Employee Name	Date of Birth	Department Code	Department Name	Project Code	Project Description	Project Supervisor
1	Raj	1-1-85	1	CE	1	IOT	Patel
2	Meet	4-4-86	2	EC	2	PHP	Shah
3	Suresh	2-2-85	1	CE	1	IOT	Patel
1	Raj	1-1-85	1	CE	2	PHP	Shah

How to normalize database?



THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

UNF

Employee Number	Employee Name	Date of Birth	Department Code	Department Name	Project Code	Project Description	Project Supervisor
1	Raj	1-1-85	1	CE	1	IOT	Patel
2	Meet	4-4-86	2	EC	2	PHP	Shah
3	Suresh	2-2-85	1	CE	1	IOT	Patel
1	Raj	1-1-85	1	CE	2	PHP	Shah

1NF

Employee Number	Employee Name	Date of Birth	Department Code	Department Name
1	Raj	1-1-85	1	CE
2	Meet	4-4-86	2	EC
3	Suresh	2-2-85	1	CE

Employee Number	Project Code	Project Description	Project Supervisor
1	1	IOT	Patel
2	2	PHP	Shah
3	1	IOT	Patel
1	2	PHP	Shah

How to normalize database?



1NF

<u>Employee Number</u>	Employee Name	Date of Birth	Department Code	Department Name
1	Raj	1-1-85	1	CE
2	Meet	4-4-86	2	EC
3	Suresh	2-2-85	1	CE

<u>Employee Number</u>	<u>Project Code</u>	Project Description	Project Supervisor
1	1	IOT	Patel
2	2	PHP	Shah
3	1	IOT	Patel
1	2	PHP	Shah

2NF

<u>Employee Number</u>	Employee Name	Date of Birth	Department Code	Department Name
1	Raj	1-1-85	1	CE
2	Meet	4-4-86	2	EC
3	Suresh	2-2-85	1	CE

<u>Project Code</u>	Project Description	Project Supervisor
1	IOT	Patel
2	PHP	Shah

<u>Employee Number</u>	<u>Project Code</u>
1	1
2	2
3	1
1	2

How to normalize database?



THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

3NF

<u>Employee Number</u>	Employee Name	Date of Birth	Department Code
1	Raj	1-1-85	1
2	Meet	4-4-86	2
3	Suresh	2-2-85	1

<u>Department Code</u>	Department Name
1	CE
2	EC

<u>Project Code</u>	Project Description	Project Supervisor
1	IOT	Patel
2	PHP	Shah

<u>Employee Number</u>	<u>Project Code</u>
1	1
2	2
3	1
1	2

Drawbacks of Normalization and Denormalization

- Normalization may actually slow down the system performance with its frequently occurring table join operations.
- A normalize database requires much more CPU utilization, Memory and I/O to process transactions and database queries.
- Hence, to increase the redundancy in order to reduce the number of joins, we need **denormalization** at run time.
- The process of taking a normalized schema and making it non-normalized is called **denormalization**. The designers use it to tune the system performance for supporting time-critical operations.

Thank You

Functional Dependencies

Equivalence of FDs

Decomposition

Normalization

FUNCTIONAL DEPENDENCIES

- *Functional dependencies (FDs) are a fundamental concept in relational database theory, used to express the relationship between attributes in a database schema. functional dependency occurs when the value of one attribute (or a combination of attributes) uniquely determines the value of another attribute.*
- *If A and B are attributes (or sets of attributes) of a relation R, we say that A functionally determines B, denoted as:*
- $A \rightarrow B$
- *It is referred as: B is functionally dependent on the A or A functionally determines B.*
- *This means that for any two tuples in R if the tuples agree on the value of A, they must also agree on the value of B.*

BENEFITS OF FUNCTIONAL DEPENDENCIES

Benefits of Functional Dependencies (FDs) in Databases: Functional dependencies (FDs) play a crucial role in database normalization and design. Their benefits include:

1. *Helps in Normalization*

FDs help identify redundant data and guide the process of decomposition to achieve higher normal forms (e.g., BCNF, 3NF), ensuring minimal redundancy.

2. *Ensures Data Consistency*

If a functional dependency exists (e.g., $A \rightarrow BA$ \to $BA \rightarrow B$), it ensures that for every unique value of AAA, the value of BBB remains consistent throughout the database.

3. *Reduces Anomalies*

FDs help eliminate *insertion, deletion, and update anomalies* by structuring the database properly.

4. *Enhances Query Optimization*

Query performance can improve because FDs allow the query optimizer to rewrite queries more efficiently and reduce the search space.

FUNCTIONAL DEPENDENCIES

- In any relation, a functional dependency $\alpha \rightarrow \beta$ holds if Two tuples having same value of attribute α also have same value for attribute β .
- If α and β are the two sets of attributes in a relational table R where-

$$\alpha \subseteq R$$

$$\beta \subseteq R$$

- Then, for a functional dependency to exist from α to β ,

If $t1[\alpha] = t2[\alpha]$, then $t1[\beta] = t2[\beta]$

α	β
$t1[\alpha]$	$t1[\beta]$
$t2[\alpha]$	$t2[\beta]$
.....

FUNCTIONAL DEPENDENCIES

The left side of the FD is called the determinant, and the right side is the dependent.

Eg: SIN determines Name, Address and Birthdate. Given SIN, we can determine any of the other attributes within the table.

SIN —————> **Name, Address, Birthdate**

For example, SIN and Course determine the date completed (DateCompleted). This must also work for a composite PK.

SIN, Course → DateCompleted

The example indicates that ISBN determines Title.

ISBN → Title

roll_no	name	dept_name	dept_building
42	abc	CO	A4
43	pqr	IT	A3
44	xyz	CO	A4
45	xyz	IT	A3
46	mno	EC	B2
47	jkł	ME	B2

- $\text{roll_no} \rightarrow \{\text{name}, \text{dept_name}, \text{dept_building}\}$, → Here, roll_no can determine values of fields name, dept_name and dept_building, hence a valid Functional dependency
- $\text{roll_no} \rightarrow \text{dept_name}$, roll_no can determine whole set of {name, dept_name, dept_building}, it can determine its subset dept_name also.
- $\text{dept_name} \rightarrow \text{dept_building}$, 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:
 $\text{roll_no} \rightarrow \text{name}$,
 $\{\text{roll_no}, \text{name}\} \rightarrow \{\text{dept_name}, \text{dept_building}\}$,

Here are some valid/invalid functional dependencies:

- $\text{name} \rightarrow \text{dept_name}$

Students with the same name can have different dept_name, hence this is not a valid functional dependency.

- $\text{dept_building} \rightarrow \text{dept_name}$

There can be multiple departments in the same building, For example, in the above table departments ME and EC are in the same building B2, hence $\text{dept_building} \rightarrow \text{dept_name}$ is an invalid functional dependency. More invalid functional dependencies:

- $\text{name} \rightarrow \text{roll_no},$
- $\{\text{name}, \text{dept_name}\} \rightarrow \text{roll_no}, \checkmark$
- $\text{dept_building} \rightarrow \text{roll_no}, \text{etc.}$

roll_no	name	dept_name	dept_building
42	abc	CO	A4
43	pqr	IT	A3
44	xyz	CO	A4
45	xyz	IT	A3
46	mno	EC	B2
47	JKL	ME	B2

Functional Dependencies

Consider the given tables

A	B
1	1
2	4
3	9
4	16
2	4
7	9

A	B
1	1
2	4
3	9
4	16
3	10
7	9

Table illustrates $A \rightarrow B$

For two different A we can have same B value but for Same A cant have two different values

A	B	C	D	E
a1	b1	c1	d1	e1
a2	b1	C2	d2	e1
a3	b2	C1	d1	e1
a4	b2	C2	d2	e1
a5	b3	C3	d1	e1

Table R

- $A \rightarrow B : YES$
- $A \rightarrow C : YES$
- $D \rightarrow C : NO$
- $C \rightarrow B : NO$
- $E \rightarrow B : NO$
- $B \rightarrow C : NO$

Functional Dependencies

X	Y	Z
1	1	1
2	1	2
2	1	3
3	1	1

Which of the following is Functional Dependency?

1. $X \rightarrow Y$
2. $XY \rightarrow Z$
3. $YZ \rightarrow X$

Functional Dependencies

Note: Data depends on dependency not dependency depends on data a,b are satisfied

a) $A \rightarrow BC$

b) $DE \rightarrow C$

c) $C \rightarrow DE$

d) $BC \rightarrow A$

	A	B	C	D	E
a	2	3	4	5	
2	a	3	4	5	
a	2	3	6	5	
a	2	3	6	6	

A: Y

B: Y

C: N

D: Y

FUNCTIONAL DEPENDENCIES

Rule-01:

A functional dependency $X \rightarrow Y$ will always hold if all the values of X are unique (different) irrespective of the values of Y .

Example- Consider the following table-

The following functional dependencies will always hold since all the values of attribute 'A' are unique-

- $A \rightarrow B$
- $A \rightarrow BC$
- $A \rightarrow CD$
- $A \rightarrow BCD$
- $A \rightarrow DE$
- $A \rightarrow BCDE$

A \rightarrow Any combination of attributes A, B, C, D, E

A	B	C	D	E
5	4	3	2	2
8	5	3	2	1
1	9	3	3	5
4	7	3	3	8

In general, we can say following functional dependency will always hold-

FUNCTIONAL DEPENDENCIES

Rule-02:

A functional dependency $X \rightarrow Y$ will always hold if all the values of Y are same irrespective of the values of X .

Example- Consider the following table-

The following functional dependencies will always hold since all the values of attribute 'C' are same-

- $A \rightarrow C$
- $AB \rightarrow C$
- $ABDE \rightarrow C$
- $DE \rightarrow C$
- $AE \rightarrow C$

Any combination of attributes $A, B, C, D, E \rightarrow C$

A	B	C	D	E
5	4	3	2	2
8	5	3	2	1
1	9	3	3	5
4	7	3	3	8

FUNCTIONAL DEPENDENCIES

Rule-03:

For a functional dependency $X \rightarrow Y$ to hold, if two tuples in the table agree on the value of attribute X, then they must also agree on the value of attribute Y.

Rule-04:

For a functional dependency $X \rightarrow Y$, violation will occur only when for two or more same values of X, the corresponding Y values are different.

A	B	C	D	E
5	4	3	2	2
8	5	3	2	1
1	9	3	3	5
4	7	3	3	8

Full/Partial Functional Dependency

Full Functional Dependency (FFD)

- An attribute *fully depends* on the *entire key* and not just a part of it.
- If any part of the key is removed, the dependency *no longer holds*.

Example:

Consider a relation $R(\text{Student_ID}, \text{Course_ID}, \text{Marks})$ with the primary key: $(\text{Student_ID}, \text{Course_ID})$.

Functional Dependency:

$(\text{Student_ID}, \text{Course_ID}) \rightarrow \text{Marks}$

Student_ID	Course_ID	Student_Name	Marks
101	CSE101	Alice	85
101	MTH102	Alice	90
102	CSE101	Bob	78
102	PHY103	Bob	88

Here, Marks fully depends on both Student_ID and Course_ID.

- If we remove Course_ID, we cannot uniquely determine Marks anymore.
- This is a Full Functional Dependency.

Full/Partial Functional Dependency

Partial Functional Dependency (PFD):

- An attribute depends on only a part of the primary key, not the whole key.
- This creates redundancy and violates 2NF.

Example:

Consider a relation $R(\text{Student_ID}, \text{Course_ID}, \text{Student_Name}, \text{Marks})$ with the primary key: $(\text{Student_ID}, \text{Course_ID})$.

Functional Dependencies:

$(\text{Student_ID}, \text{Course_ID}) \rightarrow \text{Marks}$  (Full Dependency)

$\text{Student_ID} \rightarrow \text{Student_Name}$  (Partial Dependency)

Student_ID	Course_ID	Student_Name	Marks
101	CSE101	Alice	85
101	MTH102	Alice	90
102	CSE101	Bob	78
102	PHY103	Bob	88

- Here, Student_Name depends only on Student_ID , not on Course_ID . This means we are storing Student_Name multiple times if the same student is enrolled in multiple courses.

Full/Partial Functional Dependency

A	B	C
1	α	X
2	β	Y
1	γ	Z
3	γ	Z
4	γ	Z
5	γ	Z

$$\begin{array}{l} AB \rightarrow C (F) \\ B \rightarrow C (P) \end{array}$$

Types of Functional Dependency (FD)

- **Transitive Functional Dependency**

- In a relation, if attribute(s) **A → B and B → C, then A → C** (means C is transitively depends on A via B).

Sub_Fac

Subject	Faculty	Age
DS	Shah	35
DBMS	Patel	32
DF	Shah	35

- Eg. *Subject → Faculty & Faculty → Age then Subject → Age*
- Therefore as per the rule of transitive dependency: **Subject → Age** should hold, that makes sense because if we know the subject name we can know the faculty's age.

Armstrong's Axioms: Inference Rules

- **Reflexivity**

If B is a subset of A , then $A \rightarrow B$ always holds.

- **Transitivity**

If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$ always holds.

- **Augmentation**

If $A \rightarrow B$, then $AC \rightarrow BC$ always holds.

- **Decomposition**

If $A \rightarrow BC$, then $A \rightarrow B$ and $A \rightarrow C$ always holds.

- **Union**

If $X \rightarrow Y$, then $X \rightarrow Z$ and $X \rightarrow YZ$ always holds.

Armstrong's Axioms: Inference Rules

Composition

If $A \rightarrow B$ and $C \rightarrow D$, then $AC \rightarrow BD$ always holds.

Additive

If $A \rightarrow B$ and $A \rightarrow C$, then $A \rightarrow BC$ always holds.

Pseudo-transitivity:

If $X \rightarrow Y$ and $YZ \rightarrow W$ then $XZ \rightarrow W$.

Armstrong's Axioms: Inference Rules

- In DBMS, soundness and completeness are properties related to functional dependencies (FDs) and inference rules (such as Armstrong's Axioms).
- They ensure that the rules used to infer new dependencies are both correct and sufficient.

Armstrong's Axioms: Inference Rules: Soundness

- **Soundness:** A system (or inference rule) is **sound** if it never derives incorrect results.
 - In terms of **functional dependencies**, an inference rule is **sound** if every FD it derives is **logically valid** in the given relation.
 - This ensures that we do not generate **false dependencies** that do not actually hold.

Example of Soundness

Given:

$$A \rightarrow B, \quad B \rightarrow C$$

Using transitivity:

$$A \rightarrow C \quad \checkmark \quad (\text{This is correct and valid.})$$

However, if we wrongly infer:

$$C \rightarrow A \quad \times \quad (\text{This is incorrect and unsound.})$$

Thus, sound rules only generate correct dependencies.

Armstrong's Axioms: Inference Rules: Completeness

- **Completeness:** A system (or inference rule) is **complete** if it derives all possible correct results.
 - In terms of **functional dependencies**, inference rules are **complete** if they can derive **every valid FD** that logically follows from a given set of FDs.
 - This ensures **no true dependency is missed**.

Example of Completeness

Given:

$$A \rightarrow B$$

$$B \rightarrow C$$

A complete inference system should be able to derive:

$$A \rightarrow C \quad \checkmark \quad (\text{Transitivity rule})$$

If the system fails to derive $A \rightarrow C$, it is **not complete**.

Equivalence of Two Sets of Functional Dependencies-

In DBMS,

- *Two different sets of functional dependencies for a given relation may or may not be equivalent.*
- *If F and G are the two sets of functional dependencies, then following 3 cases are possible-*
 - *Case-01: F covers G ($F \supseteq G$)*
 - *Case-02: G covers F ($G \supseteq F$)*
 - *Case-03: Both F and G cover each other ($F = G$)*

Case-01: Determining Whether F Covers G ($F \supseteq G$)

Following steps are followed to determine whether F covers G or not-

Step-01:

- *Take the functional dependencies of set G into consideration.*
- *For each functional dependency $X \rightarrow Y$, find the closure of X using the functional dependencies of set G.*

Step-02:

- *Take the functional dependencies of set G into consideration.*
- *For each functional dependency $X \rightarrow Y$, find the closure of X using the functional dependencies of set F.*

Step-03:

- *Compare the results of Step-01 and Step-02.*
- *If the functional dependencies of set F has determined all those attributes that were determined by the functional dependencies of set G, then it means F covers G.*
- *Thus, we conclude F covers G ($F \supseteq G$) otherwise not.*

Case-02: Determining Whether G Covers F ($G \supseteq F$)

Following steps are followed to determine whether G covers F or not-

Step-01:

- Take the functional dependencies of set F into consideration.
- For each functional dependency $X \rightarrow Y$, find the closure of X using the functional dependencies of set F .

Step-02:

- Take the functional dependencies of set F into consideration.
- For each functional dependency $X \rightarrow Y$, find the closure of X using the functional dependencies of set G .

Step-03: Compare the results of Step-01 and Step-02.

- If the functional dependencies of set G has determined all those attributes that were determined by the functional dependencies of set F , then it means G covers F .
- Thus, we conclude G covers F ($G \supseteq F$) otherwise not.

Case-03: Determining Whether Both F and G Cover Each Other-

- *If F covers G and G covers F, then both F and G cover each other.*
- *Thus, if both the above cases hold true, we conclude both F and G cover each other ($F = G$).*

A relation $R(A, C, D, E, H)$ is having two functional dependencies sets F and G as shown-

Set F-

$$A \rightarrow C$$

$$AC \rightarrow D$$

$$E \rightarrow AD$$

$$E \rightarrow H$$

Set G-

$$A \rightarrow CD$$

$$E \rightarrow AH$$

Which of the following holds true?

(A) $G \supseteq F$

(B) $F \supseteq G$

(D)

(C) $F = G$

(D) All of the above

Step-01:

$(A)^+ = \{ A, C, D \}$ // closure of left side of $A \rightarrow CD$ using set G

$(E)^+ = \{ A, C, D, E, H \}$ // closure of left side of $E \rightarrow AH$ using set G

Step-02:

$(A)^+ = \{ A, C, D \}$ // closure of left side of $A \rightarrow CD$ using set F

$(E)^+ = \{ A, C, D, E, H \}$ // closure of left side of $E \rightarrow AH$ using set F

Step-03:

Comparing the results of Step-01 and Step-02, we find-

Functional dependencies of set F can determine all the attributes which have been determined by the functional dependencies of set G. Thus, we conclude F covers G i.e. $F \supseteq G$.

Determining whether G covers F-

Step-01:

$(A)^+ = \{ A, C, D \}$ // closure of left side of $A \rightarrow C$ using set F

$(AC)^+ = \{ A, C, D \}$ // closure of left side of $AC \rightarrow D$ using set F

$(E)^+ = \{ A, C, D, E, H \}$ // closure of left side of $E \rightarrow AD$ and $E \rightarrow H$ using set F

Step-02:

$(A)^+ = \{ A, C, D \}$ // closure of left side of $A \rightarrow C$ using set G

$(AC)^+ = \{ A, C, D \}$ // closure of left side of $AC \rightarrow D$ using set G

$(E)^+ = \{ A, C, D, E, H \}$ // closure of left side of $E \rightarrow AD$ and $E \rightarrow H$ using set G

Step-03:

Comparing the results of Step-01 and Step-02, we find-

Functional dependencies of set G can determine all the attributes which have been determined by the functional dependencies of set F.

Thus, we conclude G covers F i.e. $G \supseteq F$.

Determining whether both F and G cover each other-

From Step-01, we conclude F covers G.

From Step-02, we conclude G covers F.

Thus, we conclude both F and G cover each other i.e. $F = G$.

Thus, Option (D) is correct.

PRACTICE QUESTION-1

Which of the following is true for the below given functional dependencies of the relation R(X, Y, Z, W) and S(X, Y, Z, W)?

R: {X → Y, XY → Z, W → XZ, Z → W}

S: {X → YZ, W → XY}

I. R ⊇ S

II. S ⊇ R

1. Only I

2. Only II

3. Both I and II

4. None of these

PRACTICE QUESTION-2

Which of the following is true given 2 schemes F & G

F : { $A \rightarrow BC$, $B \rightarrow C$, $AC \rightarrow B$ }

G : { $AB \rightarrow C$, $A \rightarrow B$, $A \rightarrow C$ }

1. F covers G
2. G covers F
3. F & G cover each other
4. None of these

Attribute closure

- If we want to check whether $X \rightarrow Y$ is in a closure of the set F , could compute F^+ and check – but **expensive** ☹
- **Cheaper:** We can instead compute the **attribute closure** (X^+), using the following algorithm:
 - Then $F \setminus X \rightarrow Y$ iff Y is a subset of X^+

```
closure:= X;  
repeat until no change{  
    if  $\exists U \rightarrow V \in F$ , where  $U \subseteq \text{closure}$   
    then closure:=closure  $\cup$  V  
};
```

What is closure of a set of FDs?

- Given a set F set of functional dependencies, there are certain other **functional dependencies that are logically implied by F .**
- E.g.: $F = \{A \rightarrow B \text{ and } B \rightarrow C\}$, then we can infer that $A \rightarrow C$ (by transitivity rule)
- The set of **functional dependencies (FDs) that is logically implied by F** is called the **closure of F** .
- It is denoted by F^+ .

Closure of a set of FDs [Example]

Suppose we are given a relation schema $R(A,B,C,G,H,I)$ and the set of functional dependencies are:

$$F = A \rightarrow B \quad (1)$$

$$A \rightarrow C \quad (2)$$

$$CG \rightarrow H \quad (3)$$

$$CG \rightarrow I \quad (4)$$

$$B \rightarrow H \quad (5)$$

- The functional dependency $A \rightarrow H$ is logical implied. (Transitive rule on 1 and 5)
- The functional dependency $CG \rightarrow HI$ is logical implied. (Union rule on 3 and 4)
- The functional dependency $AG \rightarrow I$ is logical implied. (pseudo transitivity on 2, 4)

Several members of F^+ are

$$F^+ = (A \rightarrow H, CG \rightarrow HI, AG \rightarrow I)$$

Closure of a set of FDs [Example]

- ▶ Compute the closure of the following set F of functional dependencies FDs for relational schema $R = (A, B, C, D, E, F)$:
 - $F = (A \rightarrow B, A \rightarrow C, CD \rightarrow E, CD \rightarrow F, B \rightarrow E)$
- Find out the closure of F.

$A \rightarrow B \ \& \ A \rightarrow C$	Union Rule	$A \rightarrow BC$
--	------------	--------------------

$CD \rightarrow E \ \& \ CD \rightarrow F$	Union Rule	$CD \rightarrow EF$
--	------------	---------------------

$A \rightarrow B \ \& \ B \rightarrow E$	Transitivity Rule	$A \rightarrow E$
--	-------------------	-------------------

$A \rightarrow C \ \& \ CD \rightarrow F$	Pseudo-transitivity Rule	$AD \rightarrow F$
---	--------------------------	--------------------

$A \rightarrow C \ \& \ CD \rightarrow E$	Pseudo-transitivity Rule	$AD \rightarrow E$
---	--------------------------	--------------------

$$F^+ = (A \rightarrow BC, CD \rightarrow EF, A \rightarrow E, AD \rightarrow E, AD \rightarrow F)$$

Properties of Decomposition

The process of breaking up or dividing a single relation into two or more sub relations is called as decomposition of a relation.

Properties of Decomposition-

1. Lossless decomposition- *Lossless decomposition ensures-*

- *No information is lost from the original relation during decomposition.*
- *When the sub relations are joined back, the same relation is obtained that was decomposed.*
- *Every decomposition must always be lossless.*
- *Lossless join decomposition is also known as **non-additive join decomposition**.*
- *This is because the resultant relation after joining the sub relations is same as the decomposed relation.*
- *No extraneous tuples appear after joining of the sub-relations.*

Properties of Decomposition of a Relation

2. Dependency Preservation- Dependency preservation ensures-

- *None of the functional dependencies that holds on the original relation are lost.*
- *The sub relations still hold or satisfy the functional dependencies of the original relation.*

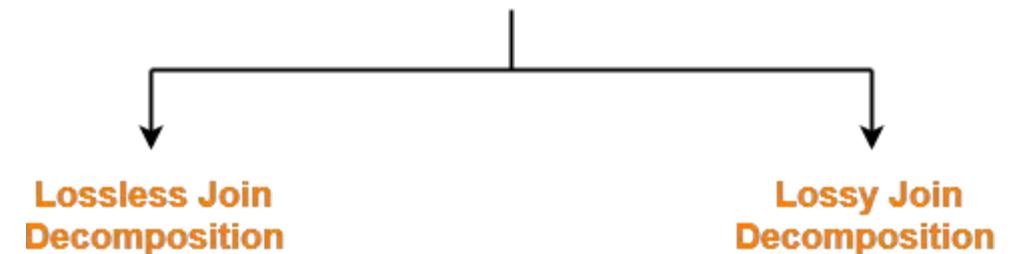
Lossless Join Decomposition

- *Lossless join decomposition is a decomposition of a relation R into relations R1, and R2 such that if we perform a natural join of relation R1 and R2, it will return the original relation R.*
- *This is effective in removing redundancy from databases while preserving the original data.*
- *In other words by lossless decomposition, it becomes feasible to reconstruct the relation R from decomposed tables R1 and R2 by using Joins.*

$R_1 \bowtie R_2 \bowtie R_3 \dots \bowtie R_n = R$

where \bowtie is a natural join operator

Types of Decomposition

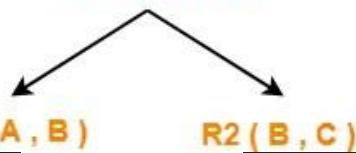


EXAMPLE:

A	B	C
1	2	1
2	5	3
3	3	3

R(A , B , C)

R (A , B , C)



A	B
1	2
2	5
3	3

R₁(A , B)

B	C
2	1
5	3
3	3

R₂(B , C)

- This relation is same as the original relation R.
- Thus, we conclude that the above decomposition is lossless join decomposition.

A	B	C
1	2	1
2	5	3
3	3	3

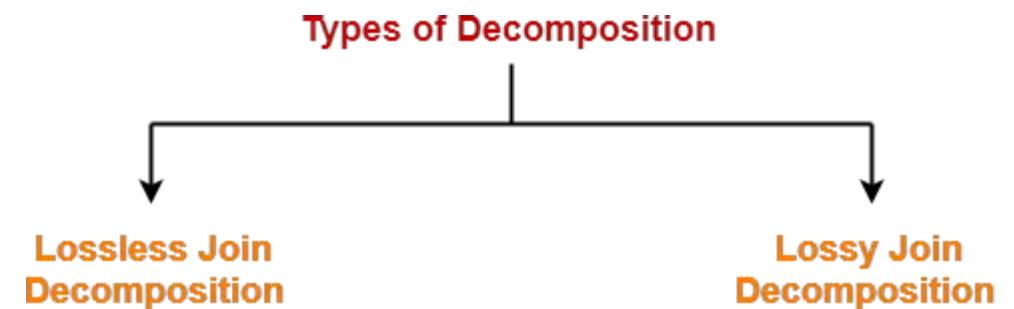
$$R_1 \bowtie R_2 = R$$

Lossy Join Decomposition

- Consider there is a relation R which is decomposed into sub relations R_1, R_2, \dots, R_n .
- This decomposition is called lossy join decomposition when the **join of the sub relations does not result in the same relation R that was decomposed.**
- The natural join of the sub relations is always found to have some **extraneous** tuples.
- For lossy join decomposition, we always have-

$$R_1 \bowtie R_2 \bowtie R_3 \dots \bowtie R_n \supseteq R$$

where \bowtie is a natural join operator



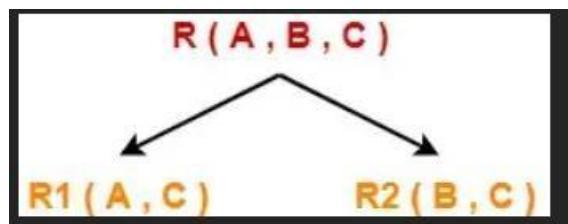
Example

A	B	C
1	2	1
2	5	3
3	3	3

$R(A, B, C)$

$R_1 \bowtie R_2 \bowtie R_3 \dots \dots \bowtie R_n \supset R$

where \bowtie is a natural join operator



A	C
1	1
2	3
3	3

$R_1(A, B)$

B	C
2	1
5	3
3	3

$R_2(B, C)$

This relation is not same as the original relation R and contains some extraneous tuples.

Clearly, $R_1 \bowtie R_2 \supset R$.

Thus, we conclude that the above decomposition is lossy join decomposition.

NOTE-

Lossy join decomposition is also known as **careless decomposition**.

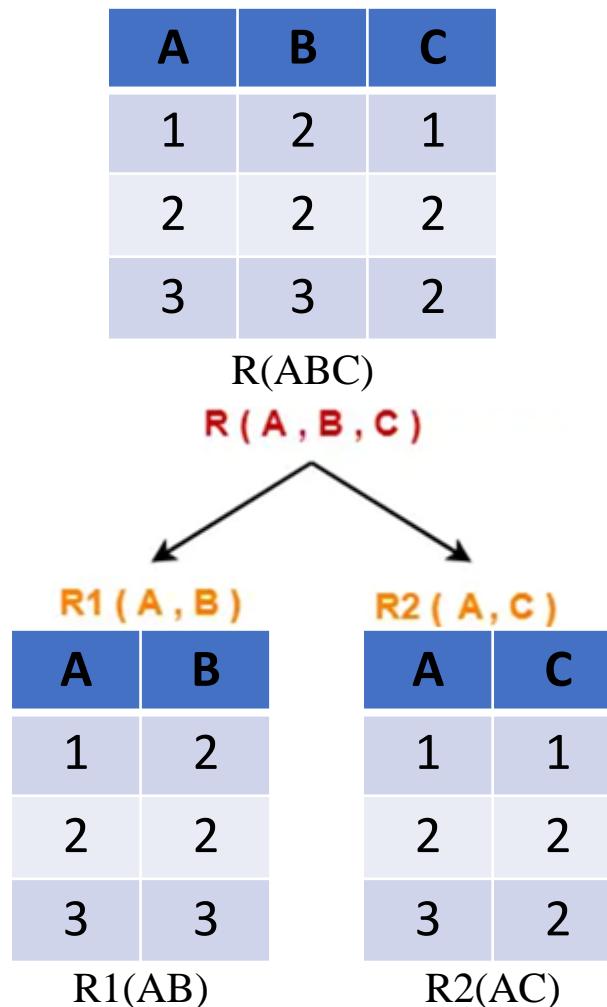
- This is because extraneous tuples get introduced in the natural join of the sub-relations.
- Extraneous tuples make the identification of the original tuples difficult.

- Find the value of C if the value of A= “1”. (Using decomposed table)

Select R2.C from R2 Natural Join R1 where R1.A=“1”.

We get 1 tuple from the original table and we get only one tuple for this query using decomposed tables as well.

- Hence, there is consistency. Therefore, this decomposition is Lossless.

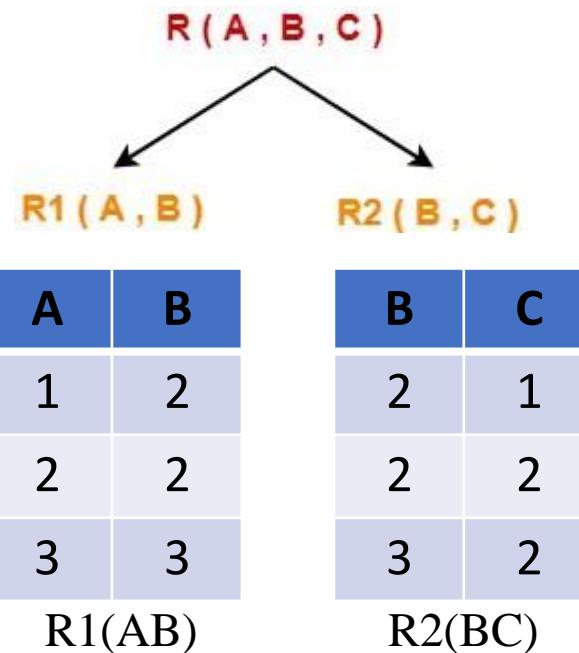


A	B	C
1	2	1✓
2	2	2
3	3	2

- If we see A here it has all values as unique.

R(ABC)

A	B	C
1	2	1
2	2	2
3	3	2



- Find the value of C if the value of A= “1”. (Using decomposed table)

Select R2.C from R2 Natural Join R1 where R1.A=“1”.

- We get 2 tuples but from the original table we get only one tuple for this query. Hence, there is inconsistency. Therefore, this decomposition is Lossy.

A	B	C
1	2	1✓
1	2	2✓
2	2	1
2	2	2
3	3	2

Natural Join

$$\mathbf{R}_1 \bowtie \mathbf{R}_2 = \mathbf{R}$$

- Here we have taken B as a common attribute which is a mistake bcz B has duplicate values by itself then how it can be used for consistent database

Rules to Determine Lossless and Lossy Decomposition

Consider a relation R is decomposed into two sub relations R_1 and R_2 .

- If all the following conditions satisfy, then the **decomposition is lossless**.
- If any of these conditions fail, then the decomposition is lossy.
- **Condition-01:** Union of both the sub relations must contain all the attributes that are present in the original relation R .

$$R_1 \cup R_2 = R$$

- **Condition-02:** Intersection of both the sub relations must not be null. In other words, there must be some common attribute which is present in both the sub relations.

$$R_1 \cap R_2 \neq \emptyset$$

- **Condition-03:** Intersection of both the sub relations must be a super key of either R_1 or R_2 or both.

$$R_1 \cap R_2 = \text{Super key of } R_1 \text{ or } R_2$$

EXAMPLE-1

1. Consider a relation schema $R (A , B , C , D)$ with the functional dependencies $A \rightarrow B$ and $C \rightarrow D$. Determine whether the decomposition of R into $R_1 (A , B)$ and $R_2 (C , D)$ is lossless or lossy.

Condition-01:

$$\begin{aligned} R_1 (A , B) \cup R_2 (C , D) \\ = R (A , B , C , D) \end{aligned}$$

Condition-02:

$$\begin{aligned} R_1 (A , B) \cap R_2 (C , D) \\ = \Phi \end{aligned}$$

Clearly, intersection of the sub relations is null.

So, condition-02 fails.

Thus, we conclude that the decomposition is lossy.

Rules for Multiple Decomposition of one Table

When a given relation is decomposed into more than two sub relations, then-

- Consider any **one possible ways in which the relation might have been decomposed into those sub relations.**
- First, divide the given relation into two sub relations.
- Then, divide the sub relations according to the sub relations given in the question.

Thumb rule:- Any relation can be decomposed only into two sub relations at a time.

EXAMPLE-2

Consider a relation schema R (A , B , C , D) with the following functional dependencies-

$$A \rightarrow B$$

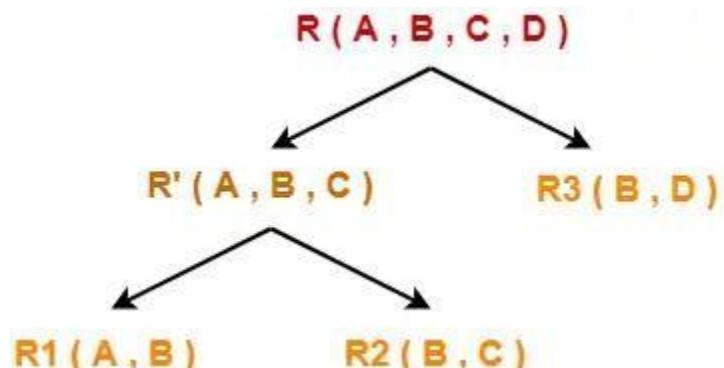
$$B \rightarrow C$$

$$C \rightarrow D$$

$$D \rightarrow B$$

Determine whether the decomposition of R into $R_1 (A , B)$, $R_2 (B , C)$ and $R_3 (B , D)$ is lossless or lossy.

First Decomposition: Condition 1:



$$R^c (A , B , C) \cup R_3 (B , D) = R (A , B , C , D)$$

First Decomposition: Condition 2:

$$R^c (A , B , C) \cap R_3 (B , D) = B$$

First Decomposition: Condition 3:

$$R^c (A , B , C) \cap R_3 (B , D) = B,$$

$$B^+ = \{ B , C , D \}$$

Thus, it is a super key of the sub relation R_3 .

EXAMPLE-2

Consider a relation schema R (A , B , C , D) with the following functional dependencies-

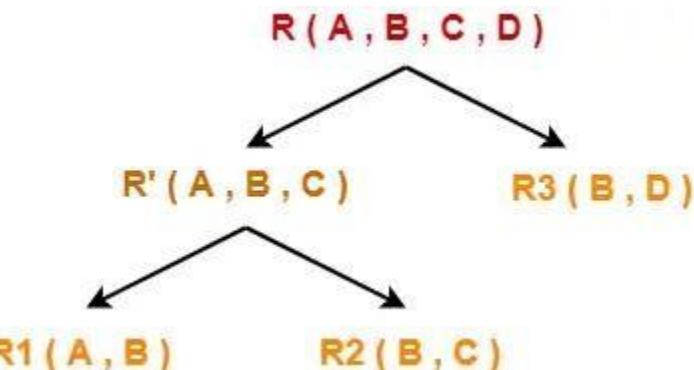
$$A \rightarrow B$$

$$B \rightarrow C$$

$$C \rightarrow D$$

$$D \rightarrow B$$

Determine whether the decomposition of R into $R_1 (A , B)$, $R_2 (B , C)$ and $R_3 (B , D)$ is lossless or lossy.



Second Decomposition: Condition 1:

$$R_1 (A , B) \cup R_2 (B , C) = R' (A , B , C)$$

Second Decomposition: Condition 2:

$$R_1 (A , B) \cap R_2 (B , C) = B$$

Second Decomposition: Condition 3:

$$R_1 (A , B) \cap R_2 (B , C) = B$$

$$B^+ = \{ B , C , D \}$$

Thus, it is a super key of the sub relation R_2 .

Functional Dependency Preservation: Rule

Example:

$R(A, B, C, D, E)$, $A \rightarrow D$, $B \rightarrow E$, $DE \rightarrow C$.

Let $S(A, B, C)$ be a decomposed relation of R . What FD-s do hold on S ?

Solution: Need to compute the closure of each subset of $\{A, B, C\}$

Compute $\{A\}^+ = AD$, $A \rightarrow D$, no new FD

Compute $\{B\}^+ = BE$, but E is not in S , so $B \rightarrow E$ does not hold

Compute $\{C\}^+ = C$, no new FD

Compute $\{AB\}^+ = ABCDE$, so $AB \rightarrow C$ holds for S (since DE are not in S)

Compute $\{BC\}^+ = BCE$, no new FD

Compute $\{AC\}^+ = ACD$, no new FD

Compute $\{ABC\}^+ = ABCDE$, no new FD

Hence, $AB \rightarrow C$ is the only nontrivial FD for S , so $\Pi_S(F^+) = \{A \rightarrow C, + \text{ trivial FDs}\}$

Dependency Preserving Decomposition

- *The process of breaking down a relation (table) into two or more smaller relations (tables). The idea is to simplify the design while preserving the ability to reconstruct the original relation.*
- *During decomposition, it's crucial to ensure that all functional dependencies (FDs) that were present in the original relation are still enforced in the decomposed relations.*
- *This means that every functional dependency that existed in the original relation should be expressible in terms of the decomposed relations.*

Functional Dependency Preservation: Rule

Let R be a relation schema and F be a set of functional dependencies on R . Suppose R is decomposed into R_1, R_2, \dots, R_n . The decomposition is **dependency preserving** if:

F_1 is subset of F

F_2 is subset of F

and $(F_1 \cup F_2 \cup \dots \cup F_n)^+ = F^+$

where:

- F^+ is the closure of the original functional dependencies.
- F_i is the set of functional dependencies that hold within R_i .
- The union of the FDs in the decomposed relations should be able to derive all dependencies in F (i.e., no FD is lost).

Functional Dependency Preservation: Rule

Let R be a relation schema and F be a set of functional dependencies on R . Suppose R is decomposed into R_1, R_2, \dots, R_n . The decomposition is **dependency preserving** if:

F_1 is subset of F

F_2 is subset of F

and $(F_1 \cup F_2 \cup \dots \cup F_n)^+ = F^+$

where:

- F^+ is the closure of the original functional dependencies.
- F_i is the set of functional dependencies that hold within R_i .
- The union of the FDs in the decomposed relations should be able to derive all dependencies in F (i.e., no FD is lost).

If we decompose a relation R into relations R_1 and R_2 , all dependencies of R must be part of either R_1 or R_2 or must be derivable from combination of functional dependencies(FD) of R_1 and R_2

Functional Dependency Preservation: Rule

Importance of Dependency Preservation

- *Ensures Data Integrity* → No dependency from the original relation is lost.
- *Avoids Expensive Joins* → If dependencies are not preserved, we may need to recombine relations frequently to check constraints.
- *Simplifies Updates and Queries* → Dependency preservation helps in efficiently enforcing constraints without complex joins.

Dependency Preserving Decomposition

$R(A, B, C, D, E)$ Decomposed into $R_1(A, B, C)$ and $R_2(C, D, E)$

$$F := \{ A \rightarrow B, \text{ (FD1)}, B \rightarrow C \text{ (FD2)}, C \rightarrow D \text{ (FD3)}, D \rightarrow A \text{ (FD3)} \}$$

<u>$R_1(A, B, C)$</u>		<u>$R_2(C, D, E)$</u>	
$A^+ = ABCD$: $A \rightarrow BC$ (A \rightarrow A is trivial) and (D not in R1)	$C^+ = CDAB$: $C \rightarrow D$
$B^+ = BCDA$: $B \rightarrow CA$ (B \rightarrow B is trivial) and (D not in R1)	$D^+ = DABC$: $D \rightarrow C$
$C^+ = CDAB$: $C \rightarrow AB$ (C \rightarrow C is trivial) and (D not in R1)	$E^+ = \{ \}$:
$AB^+ = ABCD$: $AB \rightarrow C$ (AB \rightarrow AB is trivial) and (D not in R1)	$CD^+ = CDAB$ (AB not in R2)	
$BC^+ = BCDA$: $BC \rightarrow A$ (BC \rightarrow BC is trivial) and (D not in R1)	$DE^+ = DEABC$: $DE \rightarrow C$
$AC^+ = ACBD$: $AC \rightarrow B$ (AC \rightarrow AC is trivial) and (D not in R1)	$CE^+ = CEDAB$: $CE \rightarrow D$

$$F_1 := \{ A \rightarrow BC, B \rightarrow CA, C \rightarrow AB, AB \rightarrow C, BC \rightarrow A, AC \rightarrow B \}$$

$$F_2 := \{ C \rightarrow D, D \rightarrow C, DE \rightarrow C, CE \rightarrow D \}$$

You can write ($C \rightarrow AB$ as $C \rightarrow A, C \rightarrow B$ Then use $D \rightarrow C, C \rightarrow A$ to drive $D \rightarrow A$)

$$(F1 \text{ union } F2)^+ = F^+$$

Dependency Preserving Decomposition

Let $R(A,B,C,D)$ with Functional Dependencies

$$R\{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow B\}$$

R is decomposed into $R_1(AB)$, $R_2(BC)$, $R_3(BD)$. Check whether the decomposition is dependency preserving or not?

R1(AB)	R2(BC)	R3(BD)
$A \rightarrow A$ X	$C \rightarrow C$ X	$D \rightarrow D$ X
$B \rightarrow B$ X	$B \rightarrow B$ X	$B \rightarrow B$ X
$A \rightarrow B$ ✓	$B \rightarrow C$ ✓	$B \rightarrow D$ ✓
$B \rightarrow A$ X	$C \rightarrow B$ ✓	$D \rightarrow B$ ✓

$$A \rightarrow B$$

$$B \rightarrow C$$

$$C \rightarrow B$$

$$B \rightarrow D$$

$$D \rightarrow B$$

Now, take Original Functional Dependencies and take closure from the Functional Dependencies which are obtained from the decomposed table. By checking it is clear that it is dependency preserving decomposition.

Yes, it is dependency preserving decomposition

Dependency Preserving Decomposition

Let $R(A,B,C,D)$ with Functional Dependencies

$$R\{AB \rightarrow CD, D \rightarrow A\}$$

R is decomposed into $R1(AD)$, $R2(BCD)$. Check whether the decomposition is dependency preserving or not?

R1(AD)	R2(BCD)
$A \rightarrow A$ X	$BC \rightarrow D$ X
$D \rightarrow D$ X	$BD \rightarrow C$ ✓
$A \rightarrow D$ X	$CD \rightarrow B$ X
$D \rightarrow A$ ✓	$B \rightarrow CD$ X
	$C \rightarrow BD$ X
	$D \rightarrow CB$ X

$$\begin{array}{l} D \rightarrow A \\ BD \rightarrow C \end{array}$$

Now, take Original Functional Dependencies and take closure from the Functional Dependencies which are obtained from the decomposed table. By checking it is clear that it is NOT dependency preserving decomposition.

No, it is not dependency preserving decomposition

[Example]

$R = (A, B, C)$, $F = \{A \rightarrow B, B \rightarrow C\}$

Decomposition of R: $R_1 = (A, C)$, $R_2 = (B, C)$

Does this decomposition preserve the given dependencies?

In R_1 , the following dependencies hold:

$F'_1 = \{A \rightarrow A, C \rightarrow C, A \rightarrow C, AC \rightarrow AC\}$

In R_2 , the following dependencies hold:

$F'_2 = \{B \rightarrow B, C \rightarrow C, B \rightarrow C, BC \rightarrow BC\}$

The set of nontrivial dependencies that hold on R_1 and R_2 :

$F' = \{B \rightarrow C, A \rightarrow C\}$

$A \rightarrow B$ cannot be derived from F' , so this decomposition is **NOT dependency preserving**.

[Example]

R = (A, B, C), F = {A → B, B → C}, Decomposition of R: R1 = (A, B), R2 = (B, C) Does this decomposition preserve the given dependencies?

Solution:

In R1 the following dependencies hold:

$$F_1 = \{A \rightarrow B, A \rightarrow A, B \rightarrow B, AB \rightarrow AB\}$$

In R2 the following dependencies hold:

$$F_2 = \{B \rightarrow B, C \rightarrow C, B \rightarrow C, BC \rightarrow BC\}$$

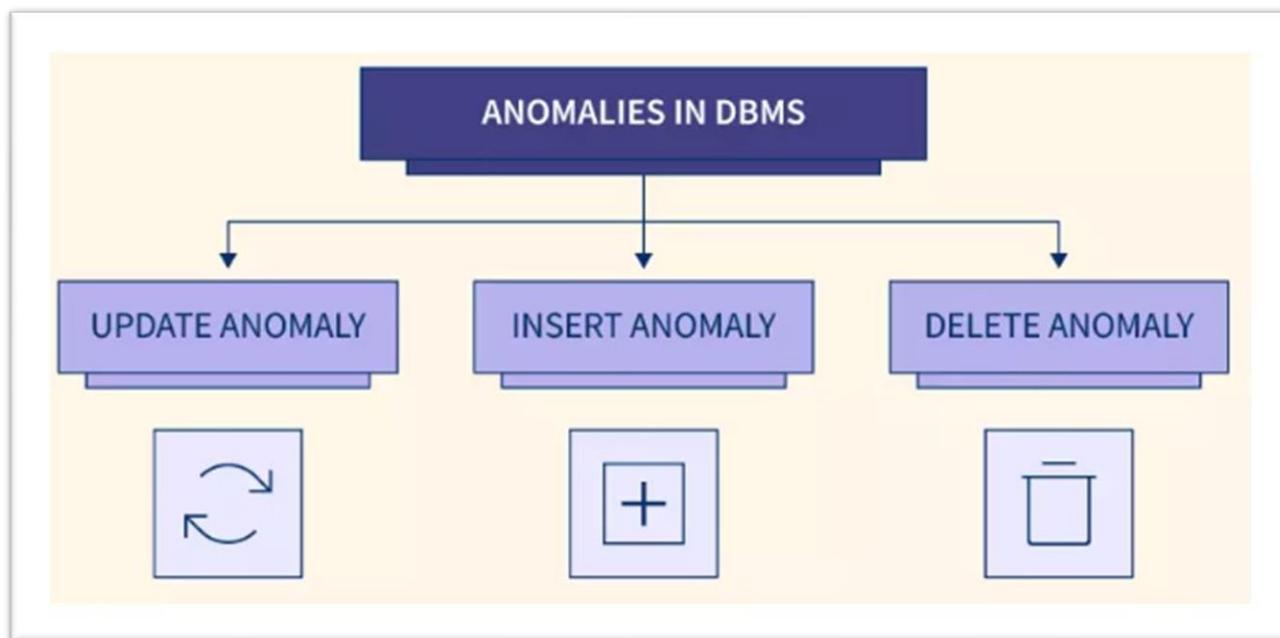
$$F' = F_1' \cup F_2' = \{A \rightarrow B, B \rightarrow C, \text{trivial dependencies}\}$$

In F', all the original dependencies occur, so this decomposition preserves dependencies.

Anomalies in Relational Model

In Database Management Systems (DBMS), anomalies refer to inconsistencies or issues that arise when performing insert, update, or delete operations on poorly designed databases. These anomalies occur due to redundant data and can lead to inefficiencies and data integrity problems.

Causes: Poor management of storing everything in the flat database, lack of normalization, data redundancy, and improper use of primary or foreign keys. leading to data integrity problems.



Insertion Anomaly

When adding new data requires unnecessary additional information.

Consider the following STUDENT table:

- *If we want to insert a new course (AI, taught by Sarah) but no students have enrolled yet, we **cannot** insert it because there is no Student_ID associated with it.*
- *We are forced to enter a NULL value or wait for a student to register.*
- *OR In a student database, if we need to add a new student but must also add a course (even if they haven't enrolled yet*

Student_ID	Student_Name	Course	Instructor	Instructor_Phone
101	Alice	DBMS	John	9876543210
102	Bob	ML	Mike	8765432109

01

Insertion Anomaly

When one cannot insert a tuple into a relationship due to lack of data is called as Insertion Anomaly.

Deletion Anomaly

Removing one piece of data unintentionally removes important related data.

Consider the following STUDENT table:

- If we remove **Bob** from the table, we also lose information about the **ML course and instructor Mike**, which should ideally remain. Now, the ML course is lost even though it still exists.*

Student_ID	Student_Name	Course	Instructor	Instructor_Phone
101	Alice	DBMS	John	9876543210
102	Bob	ML	Mike	8765432109

After Deletion

Student_ID	Student_Name	Course	Instructor	Instructor_Phone
101	Alice	DBMS	John	9876543210



Deletion Anomaly

When a deletion of data results in some unintended loss of important data, then it is referred to as **Deletion Anomaly**.

Updation Anomaly

If data is stored redundantly, updating it in one place but not another can lead to inconsistencies.

Example: If Instructor "John" changes his phone number, we must update it in all rows where he appears. If we forget to update some rows, inconsistent data remains in the table. Now, the instructor has two different phone numbers, which leads to confusion.

Student_ID	Student_Name	Course	Instructor	Instructor_Phone
101	Alice	DBMS	John	9876543210
103	Bob	DBMS	John	8765432109



Update Anomaly

When a single data value requires an update for multiple rows of data to be updated, it is referred to as update Anomaly.

Redundant Data Anomaly

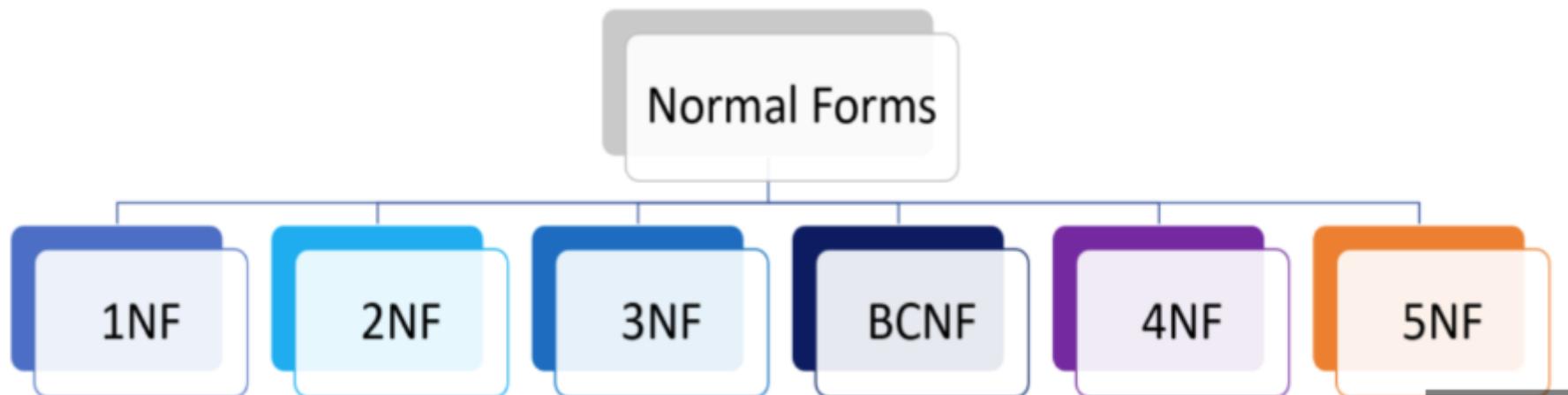
Employee Number	First Name	Last Name	Date of Birth	Department Code	Department Name	Department Head
1001	Steave	Jakson	25-09-1985	SA001	Sales	Paul Colgan
1002	Kitty	Mathew	06-04-1998	ACC008	Accounts	Jerry Mathew
1003	Meena	Patel	11-05-1992	SA001	Sales	Paul Colgan
1004	Nancy	Samual	02-12-1996	ACC008	Accounts	Jerry Mathew
1005	Michael	Smith	28-03-1995	SA001	Sales	Paul Colgan
1006	James	Garcia	22-01-1994	SA002	Sales	David Smith
1007	Nancy	Samual	11-02-1996	ACC008	Accounts	Charles Williams

Duplicate Data Is Called
Redundant Data.



Normalization in DBMS

- Normalization is the process of *organizing a database to reduce redundancy and eliminate anomalies by dividing tables into smaller related ones*. It ensures data integrity and improves efficiency.
- *Normalization ensures a well-structured database.*
- *Reduces data redundancy, improves integrity, and avoids anomalies.*
- *Helps maintain consistent and efficient data updates.*



Normalization in DBMS

Key Purposes of Normalization

1. Eliminate Data Redundancy

- Redundant data increases storage costs and can cause inconsistencies.
- Normalization ensures that data is stored in one place only, reducing duplication.

2. Prevent Data Anomalies

3. Improve Data Integrity & Consistency

- By ensuring that each piece of data is stored logically, normalization enforces referential integrity (foreign keys) and domain integrity (constraints).

4. Enhance Query Performance

- Although excessive joins can slow queries, proper normalization reduces the size of tables, leading to faster searches and updates.

5. Maintain Scalability & Flexibility

- A normalized database is easier to modify and expand without affecting existing data relationships.

1st Normal Form (1NF) – Remove Repeating Groups

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Student_id	Name	Subjects
100	Akshay	Computer Networks, Designing
101	Aman	Database Management System
102	Anjali	Automata, Compiler Design

**The following relation is not in 1NF-
Relation is not in 1NF (Subject is multivalued)**

- This relation can be brought into 1NF.
- This can be done by rewriting the relation such that each cell of the table contains only one value.

Student_id	Name	Subjects
100	Akshay	Computer Networks
100	Akshay	Designing
101	Aman	Database Management System
102	Anjali	Automata
102	Anjali	Compiler Design

1st Normal Form (1NF) – Remove Repeating Groups

The given table can be converted into 1NF by considering one of the following three cases

Case 1

Dname	Dnumber	Dmgrsn	Dlocation
Research	5	33	Delhi
Research	5	33	Mumbai
Research	5	33	Chennai
Administration	4	98	Hyderabad
Headquarters	1	88	Bhopal



suffers from redundancy.

Hence, inappropriate

Case 2

Dname	Dnumber	Dmgrsn	Dlocation1	Dlocation2	Dlocation3
Research	5	33	Delhi	Mumbai	Chennai
Administration	4	98	Hyderabad	—	—
Headquarters	1	88	Bhopal	—	—



It has disadvantage of introducing
NULL values if department
has fewer than three locations.
Hence, it is also inappropriate.

1st Normal Form (1NF) – Remove Repeating Groups

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form



STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	INDIA
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

1st Normal Form (1NF) – Remove Repeating Groups

Not in 1NF (Violating Atomicity)

Order_ID	Customer_Name	Items
501	Alice	Pizza, Burger
502	Bob	Pasta
503	Charlie	Sandwich, Fries

 **Problem:** The "Items" column contains **multiple values** in a single cell.

1NF (Fixed Version)

Order_ID	Customer_Name	Item
501	Alice	Pizza
501	Alice	Burger
502	Bob	Pasta
503	Charlie	Sandwich
503	Charlie	Fries

- ✓ Now, each column contains **atomic values**
- ✓ The table has a **primary key** (Order_ID, Item)
- ✓ The data follows **1NF rules**

2nd Normal Form (2NF) – Remove Partial Dependency

*A relation is in **Second Normal Form (2NF)** if:*

- It is already in **First Normal Form (1NF)** (i.e., no repeating groups, atomic values).
- It has **no partial dependencies** (i.e., every non-prime attribute must be fully functionally dependent on the entire Candidate key, not just a part of it).

2nd Normal Form (2NF) – Remove Partial Dependency

Partial Dependency

A partial dependency is a dependency where a ***portion of the candidate key or incomplete or few attributes of a candidate key determines non-prime attribute(s)***. In other words, $A \rightarrow B$ is called a partial dependency if and only if-

- 1. A is a subset of some candidate key*
- 2. B is a non-prime attribute.*

If any one condition fails, then it will not be a partial dependency.

NOTE-

To avoid partial dependency, incomplete candidate key must not determine any non-prime attribute.

However, incomplete candidate key can determine prime attributes.

2nd Normal Form (2NF) – Remove Partial Dependency

Steps to Convert a Relation into 2NF

Step 1: Identify the Candidate Key(s)

- *If the candidate key is a single attribute, the relation is already in 2NF.*
- *If the candidate key is composite (multiple attributes), check for partial dependencies.*

Step 2: Identify Partial Dependencies

- *A partial dependency exists if a non-prime attribute depends on only a part of the composite key rather than the full key.*

Step 3: Remove Partial Dependencies

- *Create new relations (tables) for attributes that have a partial dependency.*
- *Create new table has primary key that fully determines all its attributes.*

Step 4: Retain the Main Relation

- *Keep a relation that includes only the candidate key and fully dependent attributes.*

2nd Normal Form (2NF) – Remove Partial Dependency

Example:

$R(A, B, C, D, E, F)$

$FD \{C \rightarrow F, E \rightarrow A, EC \rightarrow D, A \rightarrow B\}$

Candidate Key: $(EC)^+ = (ECDABF)$

Check for Partial Dependency :

$C \rightarrow F \leftarrow$ Partial Dependency (C is part of EC, but not whole key)

$E \rightarrow A \leftarrow$ Partial Dependency (E is part of EC, but not whole key)

$EC \rightarrow D \leftarrow$ No Partial Dependency (EC is the full key)

$A \rightarrow B \leftarrow$ No Partial Dependency

● Partial Dependencies Exist! So the relation is not in 2NF.

Decomposition into 2NF:

$R1(C, F)$ (For PD1)

$R2(E, A, B)$ (For PD2)

$R3(E, C, D)$ (For Candidate key)

- $R_1(C, F) \rightarrow$ Captures $C \rightarrow F$ ✓
- $R_2(E, A, B) \rightarrow$ Captures $E \rightarrow A$ and $A \rightarrow B$ ✓
- $R_3(E, C, D) \rightarrow$ Captures $EC \rightarrow D$ ✓

2NF: Non-Prime Attribute should not be determined by the part of the Candidate Key.

2nd Normal Form (2NF) – Remove Partial Dependency

Example: Decomposition Lossy or LossLess $R(A, B, C, D, E, F)$

$$FD \{C \rightarrow F, E \rightarrow A, EC \rightarrow D, A \rightarrow B\}$$

Decomposition into 2NF:

$R1(\underline{C}, F), R2(\underline{E}, A, B), R3(\underline{E, C}, D)$

Step1: Decompose into two: $R' (C, F, E, D)$ and $R2 (E, A, B)$

COND1 : $R' \cup R2 = R$

COND2 : $R' \cap R2 = E$

COND3: E is the super key of R2

So, first decomposition is lossless.

Step2: Decompose $R' (C, F, E, D)$ into two: $R1 (C, F), R3 (E, D)$

COND1 : $R1 \cup R3 = R'$

COND2 : $R' \cap R3 = C$

COND3: C is the super key of R1

So, Second decomposition is also lossless.

2nd Normal Form (2NF) – Remove Partial Dependency

Example: Is Decomposition is dependency Preserving $R(A, B, C, D, E, F)$

$$FD \{C \rightarrow F, E \rightarrow A, EC \rightarrow D, A \rightarrow B\}$$

R1 (C, F),	R2 (E, A, B),	R3 (E, C, D)
$C^+ = (C, F) : C \rightarrow F$ $F^+ = (F)$	$E^+ = (E, A, B) : E \rightarrow AB$ $A^+ = (A, B) : A \rightarrow B$ $B^+ = (B) :$	$E^+ = (E, A, B) :$ $C^+ = (C) :$ $D^+ = (D) :$
	$EA^+ = (E, A, B) : EA \rightarrow B$ $AB^+ = (A, B) :$ $EB^+ = (E, B, A) : EB \rightarrow A$	$EC^+ = (E, C, D, A, B) : EC \rightarrow D$ $CD^+ = (C, D, F) : CD \rightarrow F$ $ED^+ = (E, D, A, B) :$
Dependency holed on R1: $C \rightarrow F$	Dependency holed on R2: $E \rightarrow AB, A \rightarrow B, EA \rightarrow B, EB \rightarrow A$	Dependency holed on R3: $EC \rightarrow D, CD \rightarrow F$

✓ The decomposition is dependency preserving.

2nd Normal Form (2NF) – Remove Partial Dependency

Consider a relation- R (A, B, C, D, E) with functional dependencies-

$$AB \rightarrow C$$

$$D \rightarrow E$$

Candidate Key: **(ABD)+ = R**

Check for Partial Dependency :

$AB \rightarrow C$ ← Partial Dependency (AB is part of ABD)

$D \rightarrow E$ ← Partial Dependency (D is part of ABD)

● Partial Dependencies Exist! So the relation is not in 2NF.

The proposed decomposition In 2NF:

R1 (A, B, C) → Handling $AB \rightarrow C$

R2 (D, E) → Handling $D \rightarrow E$

R3 (A, B, D) → Ensuring the candidate key ABD is preserved

2NF: Non-Prime Attribute should not be determined by the part of the Candidate Key.

2nd Normal Form (2NF) – Remove Partial Dependency

Consider a relation- R (A, B, C, D, E) with functional dependencies-

$$AB \rightarrow C$$

$$D \rightarrow E$$

R1 (A, B, C), R2 (D, E) and R3 (A, B, D) is lossy or lossless decomposition.

Step1: Decompose into two: R' (A, B, C, D) and R2 (D, E)

$$COND1 : R' \cup R2 = R$$

$$COND2 : R' \cap R2 = A, B, D$$

COND3: **ABD** is the super key of R2

So, first decomposition is lossless.

Step2: Decompose R' (A, B, C, D) into two: R1(A, B, C), R2(A, B, D)

$$COND1 : R1 \cup R2 = R'$$

$$COND2 : R' \cap R3 = A, B$$

COND3: AB is the super key of R1

So, Second decomposition is also lossless.

2nd Normal Form (2NF) – Remove Partial Dependency

Consider a relation- R (A, B, C, D, E) with functional dependencies-

$$AB \rightarrow C$$

$$D \rightarrow E$$

R1 (A, B, C), R2 (D, E) and R3 (A, B, D) is dependency preserving.

R1 (<u>A, B</u> , C)	R2 (<u>D</u> , E)	R3 (<u>A, B, D</u>)
$A^+ = (A) : A \rightarrow A$ (Tr) $B^+ = (B) : B \rightarrow B$ (Tr) $C^+ = (C) : C \rightarrow C$ (Tr)	$D^+ = (D, E) : D \rightarrow DE$ $(D \rightarrow D$ (Tr), New Only: $D \rightarrow E$)	$A^+ = (A) : (A \rightarrow A$ (Tr)) $B^+ = (B) : (B \rightarrow B$ (Tr)) $D^+ = (D, E) : D \rightarrow D$ (Tr), $D \rightarrow E$ (New)
$AB^+ = (A, B, C) : AB \rightarrow ABC$ $AB \rightarrow AB$ New Only: $AB \rightarrow C$		$AB^+ = (A, B, C) : AB \rightarrow ABC$ $: (AB \rightarrow AB$ (Tr), $AB \rightarrow C$ (New)) $BD^+ = (BDE) : BD \rightarrow BDE$ $: (BD \rightarrow BD$ (Tr), $BD \rightarrow E$ (New)) $AD^+ = (A, D, E) : (AD \rightarrow AD$ (Tr), $AD \rightarrow E$ (New))
Dependency holed on R1: $AB \rightarrow C$ (FD1)	Dependency holed on R2: $D \rightarrow E$ (FD2)	Dependency holed on R3: $D \rightarrow E, AB \rightarrow C, BD \rightarrow E, AD \rightarrow E$

- $AB \rightarrow C$ is preserved in R1(A,B,C).
- $D \rightarrow ED$ is preserved in R2(D,E).
- R3(A,B,D) helps ensure that all attributes remain connected and recoverable.

The given decomposition preserves all dependencies, so it is dependency preserving and correct.

2nd Normal Form (2NF) – Remove Partial Dependency

Consider a relation- R (A, B, C, D, E) with functional dependencies-

$$A \rightarrow B$$

$$B \rightarrow E$$

$$C \rightarrow E$$

Candidate Key: **(ACD)+ = R**

Check for Partial Dependency :

A → B ← Partial Dependency (AB is part of ABD)

B → E ← NPD

C → E ← PD

● Partial Dependencies Exist! So the relation is not in 2NF.

Decomposition into 2NF:

R1 (A, B, E) (For PD1)

R2 (B, E) (For PD2)

R3 (A, C, D) (For Candidate Key)

2NF: Non-Prime Attribute should not be determined by the part of the Candidate Key.

2nd Normal Form (2NF) – Remove Partial Dependency

Consider a relation- R (A, B, C, D, E) with functional dependencies-

$$A \rightarrow B$$

$$B \rightarrow E$$

$$C \rightarrow D$$

Candidate Key: $(AC)^+ = R$

Check for Partial Dependency :

$A \rightarrow B$ ← Partial Dependency

$B \rightarrow E$ ← Not a Partial Dependency

$C \rightarrow D$ ← Partial Dependency

● Partial Dependencies Exist! So the relation is not in 2NF.

Decomposition into 2NF:

R1 (A, B, E) (For PD1)

R2 (C, D) (For PD2)

R3 (A, C) (For Candidate Key)

2NF: Non-Prime Attribute should not be determined by the part of the Candidate Key.

2nd Normal Form (2NF) – Remove Partial Dependency

Consider a relation- R (A, B, C, D, E, F, G, H, I, J) with functional dependencies-

$$AB \rightarrow C,$$

$$AD \rightarrow GH,$$

$$BD \rightarrow EF,$$

$$A \rightarrow I$$

$$H \rightarrow J$$

Candidate Key: $(ABD)^+ = R$, Check for Partial Dependency :

$AB \rightarrow C \leftarrow$ Partial Dependency

$AD \rightarrow GH \leftarrow$ Partial Dependency

$BD \rightarrow EF \leftarrow$ Partial Dependency

$A \rightarrow I \leftarrow$ Partial Dependency

$H \rightarrow J \leftarrow$ Not Partial Dependency

● Partial Dependencies Exist! So the relation is not in 2NF.

Decomposition into 2NF:

R1 (A, B, C) (For PD1)

R2 (A, D, G, H, J) (For PD2)

R3 (B, D, E, F) (For PD3)

R4 (A, I) (For PD4)

R5 (ABD) (For Candidate Key)

2NF: Non-Prime Attribute should not be determined by the part of the Candidate Key.

3rd Normal Form (3NF) – Remove Transitive Dependency

A given relation is called in Third Normal Form (3NF) if and only if-

1. Relation already exists in 2NF.
2. *No transitive dependency exists for non-prime attributes.* meaning no non-key attribute depends on another non-key attribute.
3. In 3NF, every non-key attribute must only depend on the **primary key**, not on another non-key attribute.

OR

A relation is called in Third Normal Form (3NF) if and only if-

- Any one condition holds for each non-trivial functional dependency $A \rightarrow B$
 1. *A is a super key*
 2. *B is a prime attribute*

3rd Normal Form (3NF) – Remove Transitive Dependency

Transitive Dependency

$A \rightarrow B$ is called a transitive dependency if and only if-

1. *A is not a super key.*
2. *B is a non-prime attribute.*

If any one condition fails, then it is not a transitive dependency.

NOTE-

1. *Transitive dependency must not exist for non-prime attributes.*
2. *However, transitive dependency can exist for prime attributes.*

3rd Normal Form (3NF) – Remove Transitive Dependency

Consider a relation- R (A , B , C) with functional dependencies-

$$A \rightarrow B$$

$$B \rightarrow C$$

The possible candidate keys for this relation are-

A

From here,

- Prime attributes = { A }
- Non-prime attributes = { B , C }

NOTE: Non-prime B generates to non prime C

A	B	C
1	x	y
2	x	y
3	x	y
4	y	a
5	y	a
6	y	a
7	y	a

A	B	B	C
1	x		
2	X		
3	x		

Redundancy Get reduced after decomposition. Decomposition is lossless because B is super key of second relation, and all dependencies are preserved

So, Given relation is not in 3NF. Decomposed into 3NF having R1(A, B), R2(B, C)

3rd Normal Form (3NF) – Remove Transitive Dependency

Consider a relation- R (A , B , C , D , E) with functional dependencies-

$$A \rightarrow BC$$

$$CD \rightarrow E$$

$$B \rightarrow D$$

$$E \rightarrow A$$

The possible candidate keys for this relation are-

$$A , E , CD , BC$$

From here,

- *Prime attributes = { A , B , C , D , E }*
- *There are no non-prime attributes*

Now: Thus, we conclude that the given relation is in 3NF.

3rd Normal Form (3NF) – Remove Transitive Dependency

R(A, B, C, D)

$$FD \{AB \rightarrow C, \textcolor{red}{C} \rightarrow D\}$$

$$AB^+ = ABCD$$

Prime attribute = {A, B}, Non-Prime attribute = {C, D}

Not in 3NF But in 2NF because there is no Partial Dependency.

Conversion into 3NF: R(A, B, C, D) = R1(\underline{A} \underline{B} C) and R2(\underline{C} D)

Lossy or Lossless? : COND1 : R1 ∪ R2 = R

COND2 : R1 ∩ R2 = C

COND3: C is the super key of R2, So decomposition is lossless.

Is Dependency Preserving? Yes

<i>R1(\underline{A} \underline{B} C)</i>	<i>R1(\underline{C}, D)</i>
<i>Preserves AB → C</i>	<i>Preserves C → D</i>
$A^+ = \{A\}$: Preserves $A \rightarrow A$ (Tri) $B^+ = \{B\}$: Preserves $B \rightarrow B$ (Tri) $C^+ = \{CD\}$: Preserves $C \rightarrow C$ (Tri) $C \rightarrow C$ (D not present in R1)	
<i>Try other dependencies generated by combination of 2 attribute and 3 attributes by yourself</i>	

3rd Normal Form (3NF) – Remove Transitive Dependency

R(A, B, C, D,)

FD {AB → CD,

D → A}

$AB^+ = \text{ABCD}$

$DB^+ = \text{ABCD}$

Prime attribute= {A, B, D}

Non-Prime attribute= {C}

It is in 2NF as well as in 3NF.

3rd Normal Form (3NF) – Remove Transitive Dependency

R(A, B, C, D, E)

$FD: AB \rightarrow C$ (3NF)

$B \rightarrow D$ (PD)

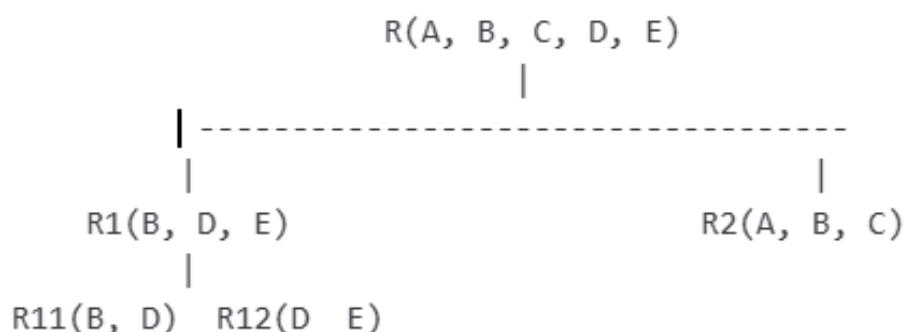
$D \rightarrow E$ (TD)

Candidate Key: $AB^+ = ABCDE$

Prime attribute = {A, B}

Non-Prime attribute = {CDE}

It is not in 2NF as well as in 3NF.



Decomposition into 3NF: R11 (B, D, E), R12 (D, E) and R2(A, B, C)

Also, Lossless and dependency preserving.

First decomposition has common attribute B superkey of R1

Second decomposition of R1 has common attribute D superkey of R12.

So decomposition is lossless.

3rd Normal Form (3NF) – Remove Transitive Dependency

R(A, B, C, D, E, F, G, H, I, J)

FD: AB → C (3NF)

A → DE (PD)

B → F (PD)

F → GH (TD)

D → IJ (TD)

Candidate Key: $AB^+ = ABCDEFGHIJ$

Prime attribute = {A, B}

Non-Prime attribute = {CDEFGHIJ}

It is not in 2NF as well as in 3NF.

R(A, B, C, D, E, F, G, H, I, J)
|

2NF R1(A, D, E, I, J) R2(B, F, G, H) R3(A, B, C)

Decomposition into 3NF: Total 5 tables, R11, R12, R21, R22, R3

Also, Lossless and dependency preserving. (Prove by yourself)

3NF R11(A, D, E) R12(D, I, J) R21(B, F) R22(F, G, H)

3rd Normal Form (3NF) – Remove Transitive Dependency

R(A, B, C, D, E, F, G, H, I, J)

FD: AB → C (PD)

AD → GH (PD)

BD → EF (PD)

A → I (PD)

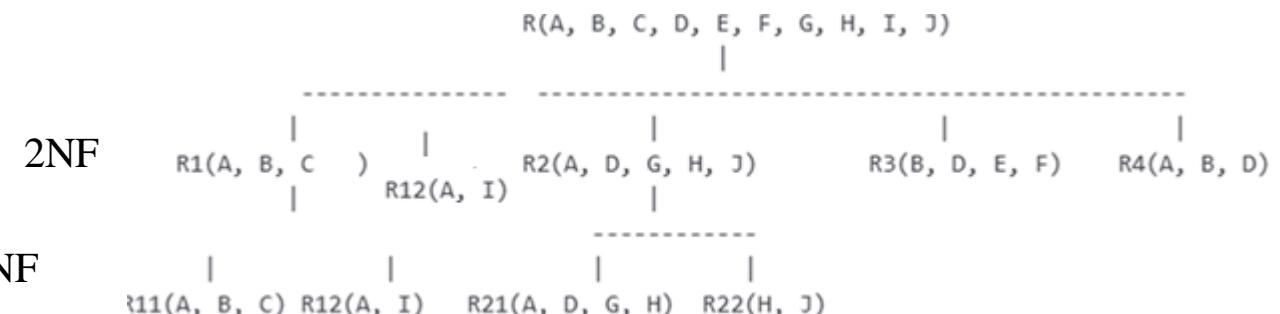
H → J (TD)

Candidate Key: $ABD^+ = ABCDEFGHIJ$

Prime attribute= {A, B, D}

Non-Prime attribute= {CEFGHIJ}

It is not in 2NF as well as in 3NF.



Decomposition into 3NF: Total 6 tables, R11, R12, R21, R22, R3

Also, Lossless and dependency preserving. (Prove by yourself)

3rd Normal Form (3NF) – Remove Transitive Dependency

$R(A, B, C, D, E, F, G, H, I, J)$

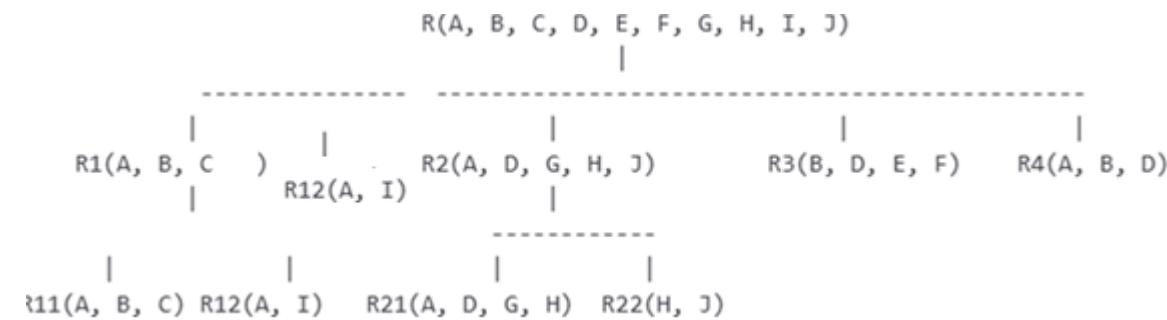
$FD: AB \rightarrow C$ (Preserved by R11)

$AD \rightarrow GH$ (Preserved by R21)

$BD \rightarrow EF$ (Preserved by R3)

$A \rightarrow I$ (Preserved by R12)

$H \rightarrow J$ (Preserved by R22)



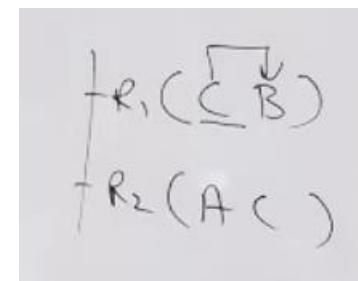
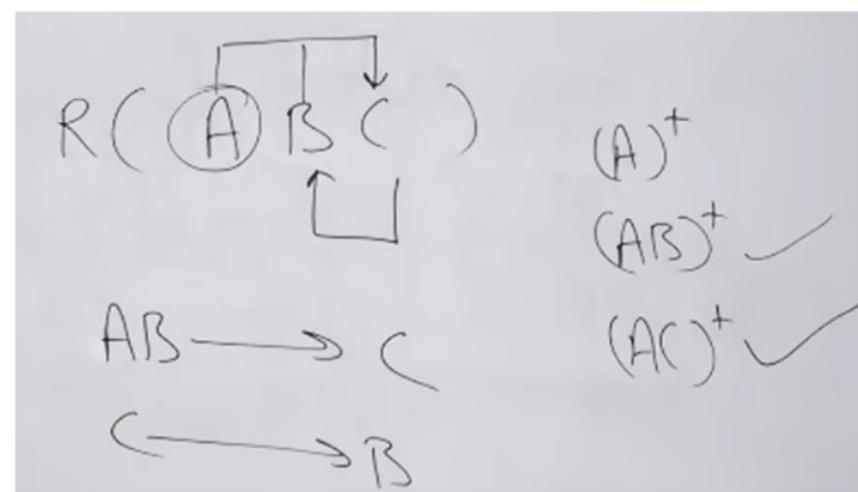
Dependency Preserving: Yes, Lossless Decomposition : Yes

Note: INT means Intersection

Boyce-Codd Normal Form

A given relation is called in BCNF if and only if-

1. Relation already exists in 3NF.
2. For each non-trivial functional dependency $A \rightarrow B$, A is a super key of the relation.



$R(A B C)$

$AB \rightarrow C$

$C \rightarrow B$

R		
A	B	C
a	1	x
b	2	y
c	2	z
c	3	w
d	3	w
e	3	w

A	C
a	x
b	y
c	z
c	w
d	w
e	w

C	B
x	1
y	2
z	2
w	3

Decomposition is lossless but not dependency preserving $AB \rightarrow C$ Lost.

Question

$R(A, B, C, D, E, F, G, H)$ Decompose in BCNF

Property-ID #	County Name	Lot #	Area	Price	Tax-rate
A	B	C	D	E	F

2NF

A	B	C	D	E

LOTS1	LOTS2
B F	

In 1NF.
Here, A and BC
are candidate
keys.

Here the FDs,
where LHS contain
either B or C will
violate 2NF.

$A \rightarrow BCDEF$

$BC \rightarrow ADEF$

$B \rightarrow F$ (PD)

$D \rightarrow E$

$D \rightarrow B$

$D \rightarrow B$ (*B is prime so not TD,*)

LOTS1A			
A	B	C	D

3NF

LOTS1B	
D	E

$B \rightarrow F$ (*B is super key so in BCNF*)

$BC \rightarrow AD$ (*BC is super key so in BCNF*)

$A \rightarrow BCD$ (*A is super key so in BCNF*)

$D \rightarrow B$ (*D is not super key so not in BCNF*)

Decompose

LOTS1A1		
A	C	D

LOTS1A2	
D	B

LOTS1B	
D	E

LOTS2	
B	F

$D \rightarrow E$ (*D is super key so in BCNF*)

BCNF

Here, while decomposing LOTS1A to convert it in BCNF,
some FD is lost. Here D is common in LOTS1A1 and LOTS1A2.
Here, B cannot be common in these two relations since in that
case B is not a candidate key of either of the relation.

Boyce-Codd Normal Form

Consider a relation- R (A , B , C) with the functional dependencies-

$$A \rightarrow B$$

$$B \rightarrow C$$

$$C \rightarrow A$$

The possible candidate keys for this relation are-

$$A, B, C$$

Now, we can observe that RHS of each given functional dependency is a candidate key.

Thus, we conclude that the given relation is in BCNF.

Normal Form

- Third Normal Form (3NF) is considered adequate for normal relational database design
- BCNF decomposition is always lossless but not always dependency preserving.
- Lossy decomposition is not allowed in 2NF, 3NF and BCNF. Unlike BCNF, Lossless and dependency preserving decomposition into 3NF and 2NF is always possible.

Summary of normal forms based on primary keys and normalization

Normal form	Test	Remedy (Normalization)
1NF	Relation should have no nonatomic attributes or nested relations.	Form new relations for each nonatomic or nested relation.
2NF	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
3NF	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

Question

The relation scheme Student Performance (name, courseNo, rollNo, grade) has the following functional dependencies:

```
name, courseNo → grade  
rollNo, courseNo → grade  
name → rollNo  
rollNo → name
```

The highest normal form of this relation scheme is:

Question

```
name, courseNo → grade  
rollNo, courseNo → grade  
name → rollNo  
rollNo → name
```

{name, Courseno} and {rollno, Courseno} will be the candidate keys because these can determine all the attributes of a relation.

Prime Attributes: {name, Courseno, rollno}

Non-Prime Attributes: {grade}

Since, there is no partial dependency in any of the functional dependency therefore, it is in 2NF. Also, the relation is in 3NF but not in BCNF because LHS of the 3rd and 4th functional dependencies is not a candidate key or super key.

Question

The relation schema Registration (Rollno, Course) has the following functional dependencies:

$$FD \{Rollno \rightarrow Course\}$$

The highest normal form of this relation scheme is given that Rollno is the Primary Key?

Question

The relation schema Registration (Rollno, Course) has the following functional dependencies:

$$FD \{Rollno \rightarrow Course\}$$

The highest normal form of this relation scheme is given that Rollno is the Primary Key?

BCNF (because we have only one functional dependency and the LHS of that Functional dependency is the Candidate Key). Since it is in BCNF, therefore it would be in 3NF, 2NF and 1NF as well.

Question

The relation schema Registration (Rollno, Courseid, Email) has the following functional dependencies:

$$FD \{ Rollno, Courseid \rightarrow Email, Email \rightarrow Rollno \}$$

The highest normal form of this relation scheme is given that Rollno, Courseid is the Primary Key?

Question

The relation schema Registration (Rollno, Courseid, Email) has the following functional dependencies:

$$FD \{ Rollno, Courseid \rightarrow Email, Email \rightarrow Rollno \}$$

The highest normal form of this relation scheme is given that Rollno, Courseid is the Primary Key?

Not in BCNF (because in $Email \rightarrow Rollno$ LHS is not a candidate key or a super key). But it is in 3NF (because in $Email \rightarrow Rollno$ RHS is a prime attribute), therefore it would be in 2NF and 1NF as well.

Question

The relation schema Registration (Rollno, Courseid, marks, grade) has the following functional dependencies:

$$FD \left\{ \begin{array}{l} Rollno, Courseid \rightarrow marks, grade \\ marks \rightarrow grade \end{array} \right\}$$

The highest normal form of this relation scheme is given that Rollno, Courseid is the Primary Key?

Question

The relation schema Registration (Rollno, Courseid, marks, grade) has the following functional dependencies:

$$FD \left\{ \begin{array}{l} Rollno, Courseid \rightarrow marks, grade \\ \textcolor{red}{marks \rightarrow grade} \end{array} \right\}$$

The highest normal form of this relation scheme is given that Rollno, Courseid is the Primary Key?

Not in BCNF (because in $\textcolor{red}{marks \rightarrow grade}$ LHS is not a candidate key or a super key). Also it is not in 3NF (because in $\textcolor{red}{marks \rightarrow grade}$ RHS is not a prime attribute), But it is in 2NF (because no partial dependency exists), therefore it is in 1NF as well.

Question

The relation schema Registration (Rollno, Courseid, Credit) has the following functional dependencies:

$$FD \left\{ \begin{array}{l} Rollno, Courseid \rightarrow Credit \\ Courseid \rightarrow Credit \end{array} \right\}$$

The highest normal form of this relation scheme is given that Rollno, Courseid is the Primary Key?

Question

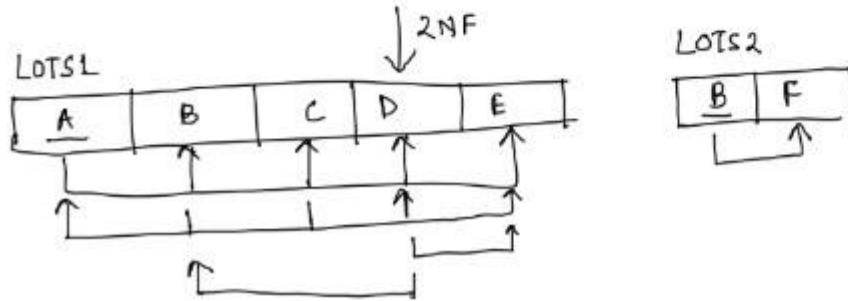
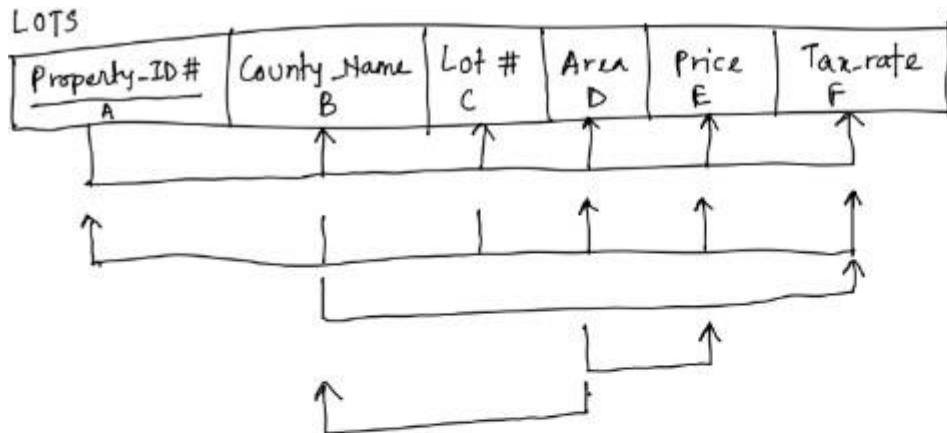
The relation schema Registration (Rollno, Courseid, Credit) has the following functional dependencies:

$$FD \left\{ \begin{array}{l} Rollno, Courseid \rightarrow Credit \\ \textcolor{red}{Courseid \rightarrow Credit} \end{array} \right\}$$

The highest normal form of this relation scheme is given that Rollno, Courseid is the Primary Key?

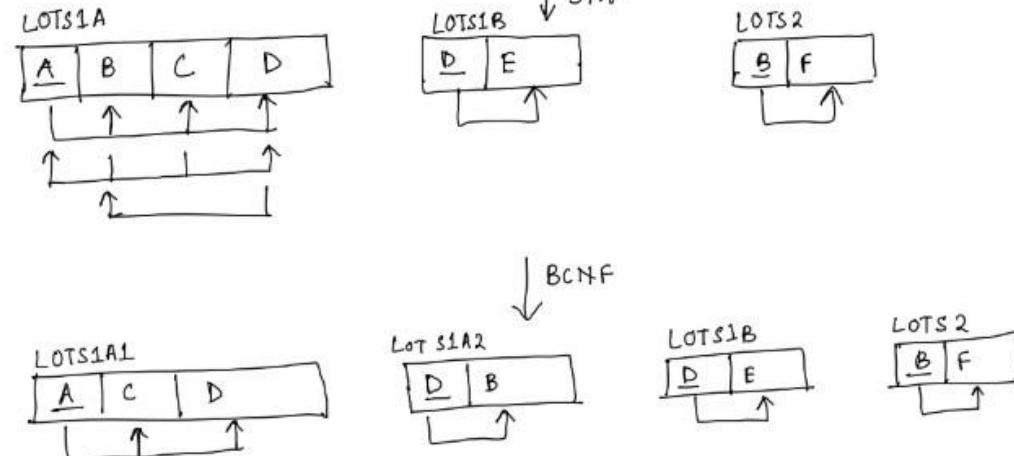
Not in BCNF (because in $\textcolor{red}{Courseid \rightarrow Credit}$ LHS is not a candidate key or a super key). Also it is not in 3NF (because in $\textcolor{red}{Courseid \rightarrow Credit}$ RHS is not a prime attribute). Also, it is not in 2NF (because partial dependency exists), therefore it is in 1NF only.

Question



In 1NF.
Here, A and BC
are candidate
keys.

Here the FDs,
where LHS contain
either B or C will
violate 2NF.



Here, while decomposing LOTS1A to convert it in BCNF,
some FD is lost. Here D is common in LOTS1A1 and LOTS1A2.
Here, B cannot be common in these two relations since in that
case B is not a candidate key of either of the relation.