UNIT – II

Biswajit Senapati,
Faculty,
Department of CSE,
NIT AP,
Tadepalligudem, AP.

NULL VALUES

- *It is possible for* ***tuples*** *to have a* ***null value***, denoted by ***null***, for some of their attributes
- ***null*** signifies
  - **an unknown value/missing**, or
  - **a value that does not exist**

Emp_Id	Name	Phone_No	Passport_No
001	Rahul	7777777777	111 111 111
002	Anil	8888888888	**NULL**

First_Name	Middle_Name	Last Name
Ranjit	Singh	Thakur
Amit	**NULL**	Chopra

- The result of any **arithmetic expression** (+, -, *, /) involving ***null*** must return a ***null***.
  - *Eg:* $5 + null = null$

    $null * 5 = null$
- **Aggregate functions** simply **ignore** **null values** (as in SQL)
  - Eg: Aggregate functions ***avg, min, max, sum*** ignores *null values* **except** for ***count***
- For **duplication, elimination** and **grouping**, **null** is ***treated like any other value***, and two nulls are assumed to be the same (as in SQL)

- **Comparisons** (<, <=, >,>=, =, ≠) with **null values** evaluate to *special value* ***unknown*** (as in SQL)
  - Because we are not sure whether the result is *true* or *false*
  - *Eg:* $5 = null$, $null > 5$, $5 > null$, $null = null$ ⟶ ***unknown***

- **Comparisons** in **Boolean expressions** involving **AND, OR, NOT** operations uses ***three-valued logic*** i.e. true (1), false (0), unknown (as in SQL)
- ***Three-valued logic*** using the truth value ***unknown***:
  - **OR:** (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - **AND:** (*true* **and** *unknown*) = *unknown*,
    (*false* **and** *unknown*) = *false*,
    (*unknown* **and** *unknown*) = *unknown*
  - **NOT:** (**not** *unknown*) = *unknown*
- Result of **select predicate** is treated as ***false*** if it evaluates to ***unknown***

Modification of the Database

- The **content of the database** may be **modified** using the following operations:
  1. **Deletion**
  2. **Insertion**
  3. **Updating**
- All these operations are expressed using the **assignment operator** (←).
  - *Eg:* $r_{new} \leftarrow$ **operations on** (r_{old})

1. Deletion

- A ***delete*** *request* is expressed ***similarly*** to a ***query***, except *instead of displaying tuples to the user*, ***the selected tuples are removed from the database.***
- ***In Deletion, tuples are deleted from the relation***
- **Can delete only whole tuples;** cannot delete values on only particular attributes
- A **deletion** is expressed in relational algebra by:

$$r \leftarrow r - E$$

where r is a relation and E is a relational algebra expression.

Example

$$r \leftarrow r - \sigma_{A=2}(r)$$

r

A	B	C
1	1	10
1	2	10
2	3	10
2	4	20

A	B	C
1	1	10
1	2	10

$$r \leftarrow r - \{(1, 2, 10)\}$$

A	B	C
1	1	10
2	3	10
2	4	20

Deletion Examples Query

- Delete all account records in the Perryridge branch.

$$account \leftarrow account - \sigma_{branch\text{-}name = \text{“Perryridge”}}(account)$$

- Delete all loan records with amount in the range of 0 to 50

$$loan \leftarrow loan - \sigma_{amount \geq 0 \text{ and } amount \leq 50}(loan)$$

- Delete all accounts at branches located in Needham.

$$r_1 \leftarrow \sigma_{branch\text{-}city = \text{“Needham”}}(account \bowtie branch)$$

$$r_2 \leftarrow \prod_{account\text{-}number,\ branch\text{-}name,\ balance}(r_1)$$

$$r_3 \leftarrow \prod_{customer\text{-}name,\ account\text{-}number}(r_2 \bowtie depositor)$$

$$account \leftarrow account - r_2$$

$$depositor \leftarrow depositor - r_3$$

2. Insertion

- ***Similar to deletion**, but **uses ∪ operator** instead of − operator.*
- ***In Insertion, tuples are added to the relation***
- To **insert** data into a relation, we either:
  - **specify a tuple to be inserted**, or
  - **write a query whose result is a set of tuples to be inserted**
- An **insertion** is expressed in relational algebra by:

$$r \leftarrow r \cup E$$

where r is a relation and E is a relational algebra expression.

- The ***insertion of a single tuple*** is expressed by letting *E* be a *constant relation containing one tuple.*

Example

$r \leftarrow r \cup \{(2, 4, 20)\}$

r

A	B	C
1	1	10
1	2	10
2	3	10

A	B	C
1	1	10
1	2	10
2	3	10
2	4	20

Insertion Examples Query

- Insert information in the database specifying that **Smith** has **$1200** in account **A-973** at the **Perryridge** branch.

$$account \leftarrow account \cup \{(A\text{-}973, \text{“Perryridge”}, 1200)\}$$

$$depositor \leftarrow depositor \cup \{(\text{“Smith”}, A\text{-}973)\}$$

We may want to insert tuples on the basis of result of a query.

- Provide as a gift for **all loan customers** in the **Perryridge** branch, a **$200 savings account.** Let the loan number serve as the account number for the new savings account.

$$r_1 \leftarrow (\sigma_{branch\text{-}name = \text{“Perryridge”}} (borrower \bowtie loan))$$

$$account \leftarrow account \cup \prod_{loan\text{-}number,\ branch\text{-}name,\ 200} (r_1)$$

$$depositor \leftarrow depositor \cup \prod_{customer\text{-}name,\ loan\text{-}number} (r_1)$$

3. Updating

- **Updating** is a mechanism ***to change a value in a tuple*** *without changing* ***all values*** *in the tuple*
- ***Use*** the ***generalized projection*** operator to do this task

$$r \leftarrow \prod_{F1, F2, \ldots, Fn} (r)$$

  - Each F_i can be either:
    - the ***attribute*** of r; or
    - an ***expression***, involving only ***constants and the attributes*** of r, which gives the new value for the attribute
  - **Note**: The schema of the expression resulting from the generalized projection expression must match the original schema of r.

Example

$$r \leftarrow \prod_{A,\ 2*B,\ C} (r)$$

r

A	B	C
1	1	10
1	2	10
2	4	20

A	B	C
1	2	10
1	4	10
2	8	20

$$r \leftarrow \prod_{A,\ 2*B,\ C} (\sigma_{A=1} (r))$$

A	B	C
1	2	10
1	4	10

Update Examples Query

- Make interest payments by increasing all balances by 5 percent.

$$account \leftarrow \prod_{account\text{-}number,\ branch\text{-}name,\ balance * 1.05} (account)$$

- Pay all accounts with balances over $10,000 a 6 percent interest and pay all others 5 percent

$$account \leftarrow \prod_{account\text{-}number,\ branch\text{-}name,\ balance * 1.06} (\sigma_{balance > 10000} (account)) \cup \prod_{account\text{-}number,\ branch\text{-}name,\ balance * 1.05} (\sigma_{balance \leq 10000} (account))$$

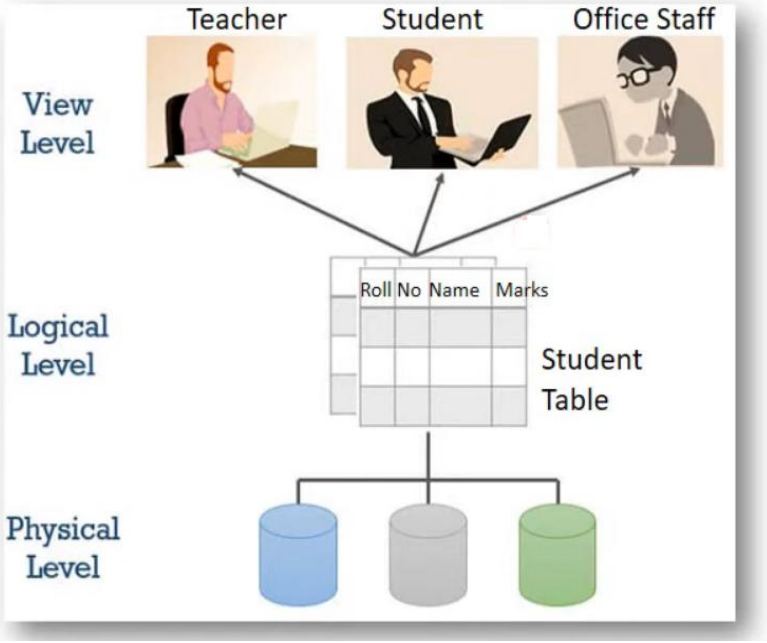
VIEW

What is Views in DBMS?

- Views in SQL are considered as a virtual table.
- A view also has rows and columns as they are in a real table in the database
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.

Operations on View:

1. Create View
2. Update View
3. Delete View



Create View

- **Syntax:**

```
CREATE VIEW view_name AS
SELECT column1, column2.....
FROM table_name
WHERE condition;
```

Create View on Single Table

```
CREATE VIEW DetailsView AS
SELECT Name, Address
FROM Stud_Details
WHERE ID < 4;
```

```
SELECT * FROM DetailsView;
```

Output:

Name	Address
Rahul	Delhi
Sara	Pune
Rohit	Mumbai

Create View on Multiple Table

```
CREATE VIEW MarksView AS
SELECT Stud_Detail.Name, Stud_Detail.Address,
Stud_Marks.Marks
FROM Stud_Detail, Stud_Mark
WHERE Stud_Detail.Name = Stud_Marks.Name;
```

```
SELECT * FROM MarksView;
```

Output:

Name	Address	Marks
Rahul	Delhi	25
Sara	Pune	29
Rohit	Mumbai	22
Parth	Pune	24

Given Tables in DB

ID	Name	Address
1	Rahul	Delhi
2	Sara	Pune
3	Rohit	Mumbai
4	Parth	Pune

Table 1: Stud_Details

ID	Name	Marks
1	Rahul	25
2	Sara	29
3	Rohit	22
4	Parth	24
5	Riya	30

Table 2: Stud_Marks

Update View

- **Syntax:**

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Given Tables in DB

ID	Name	Address
1	Rahul	Delhi
2	Sara	Pune
3	Rohit	Mumbai
4	Parth	Pune

Table 1: Stud_Details

ID	Name	Marks
1	Rahul	25
2	Sara	29
3	Rohit	22
4	Parth	24
5	Riya	30

Table 2: Stud_Marks

Create View DetailsView	Update View DetailsView
CREATE VIEW DetailsView AS SELECT Name, Address FROM Stud_Details WHERE ID < 3;	CREATE OR REPLACE VIEW DetailsView AS SELECT ID, Name, Address FROM Stud_Details WHERE ID < 3;
SELECT * FROM DetailsView;	SELECT * FROM DetailsView;
Output:	**Output:**

Name	Address
Rahul	Delhi
Sara	Pune
Rohit	Mumbai
Parth	Pune

ID	Name	Address
1	Rahul	Delhi
2	Sara	Pune
3	Rohit	Mumbai
4	Parth	Pune

Delete View

- **Syntax:**

DROP VIEW view_name;

Create View DetailsView	Delete View DetailsView
CREATE VIEW DetailsView AS SELECT Name, Address FROM Stud_Details WHERE ID < 3;	DROP VIEW DetailsView
SELECT * FROM DetailsView;	-
Output: Name | Address Rahul | Delhi Sara | Pune Rohit | Mumbai Parth | Pune	**Output:** (View Deleted)

Given Tables in DB

ID	Name	Address
1	Rahul	Delhi
2	Sara	Pune
3	Rohit	Mumbai
4	Parth	Pune

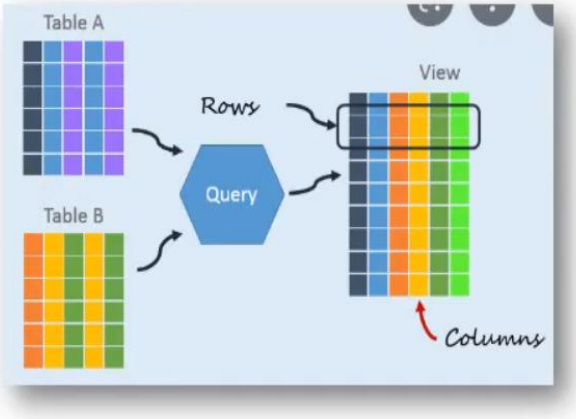
Table 1: Stud_Details

ID	Name	Marks
1	Rahul	25
2	Sara	29
3	Rohit	22
4	Parth	24
5	Riya	30

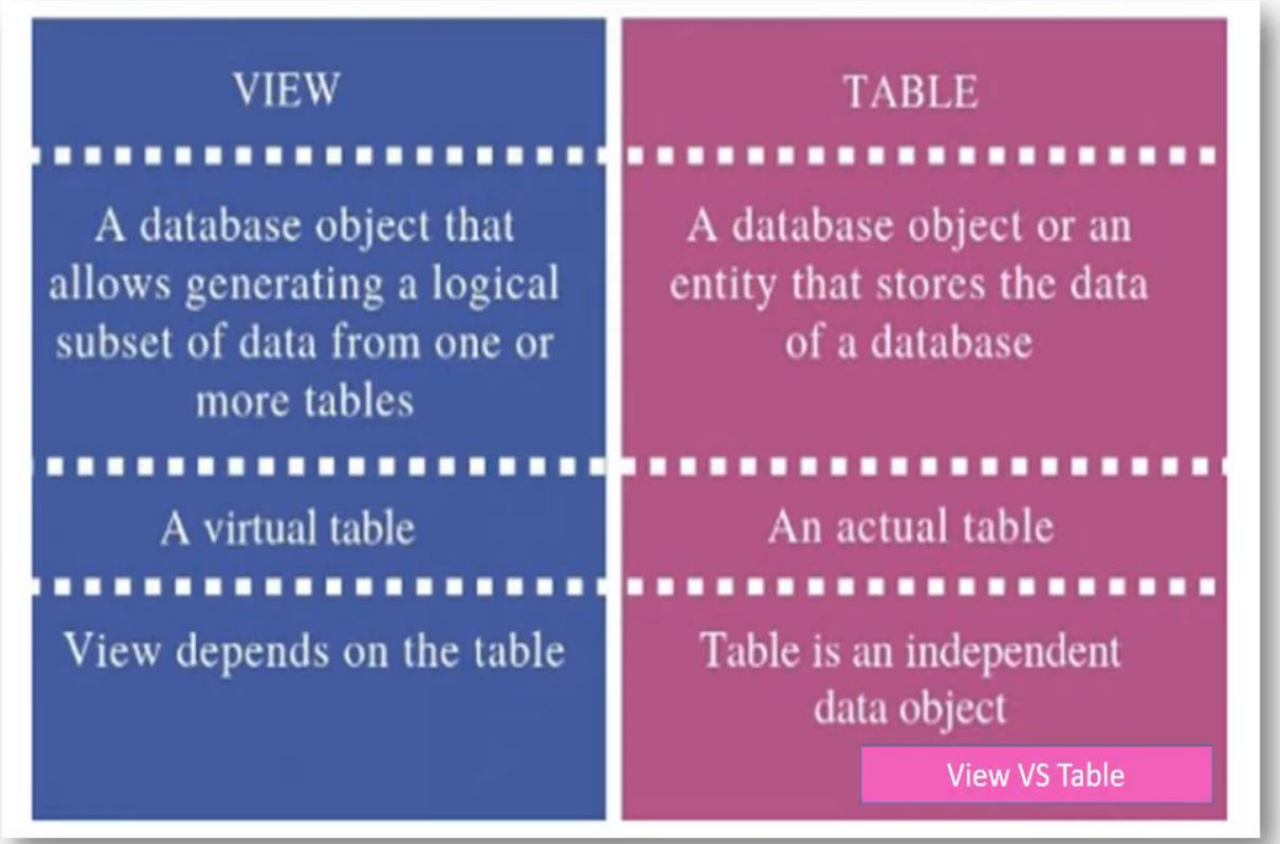
Table 2: Stud_Marks

Types of Views

1. **Simple View:** Restricts user to access only one table. No use of group by, where & any other functions.
2. **Complex View:** Restricts user to access more then one table using group by, where & any other functions.
3. **Horizontal View:** Restricts user to access data from only selected rows of table.
4. **Vertical View:** Restricts user to access data from only selected columns of table.
5. **Materialized View:** Create complete copy of one table & perform operations on that.



View VS Table

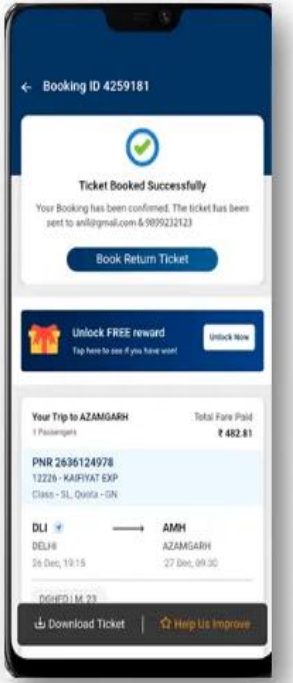


TRANSACTIONS

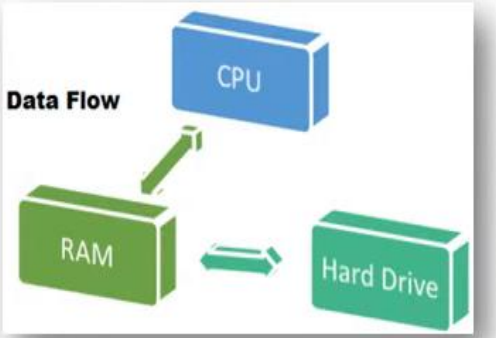
What is Transection?

- Transaction is a set of operations which are all logically related.
- Transaction is a single logical unit of work formed by a set of operations.

Examples:



- Database Connection
- DDL Commands
- DML Commands etc.

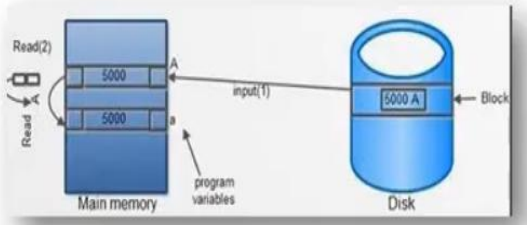


Operations in Transection

The main operations in a transaction are:

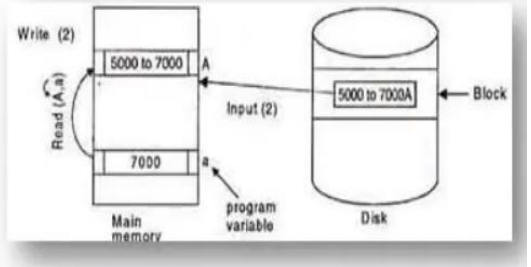
1. Read Operations:

- Read operation reads the data from the database and then stores it in the buffer in main memory.
- Example: Read(A) instruction will read the value of A from the database and will store it in the buffer in main memory.

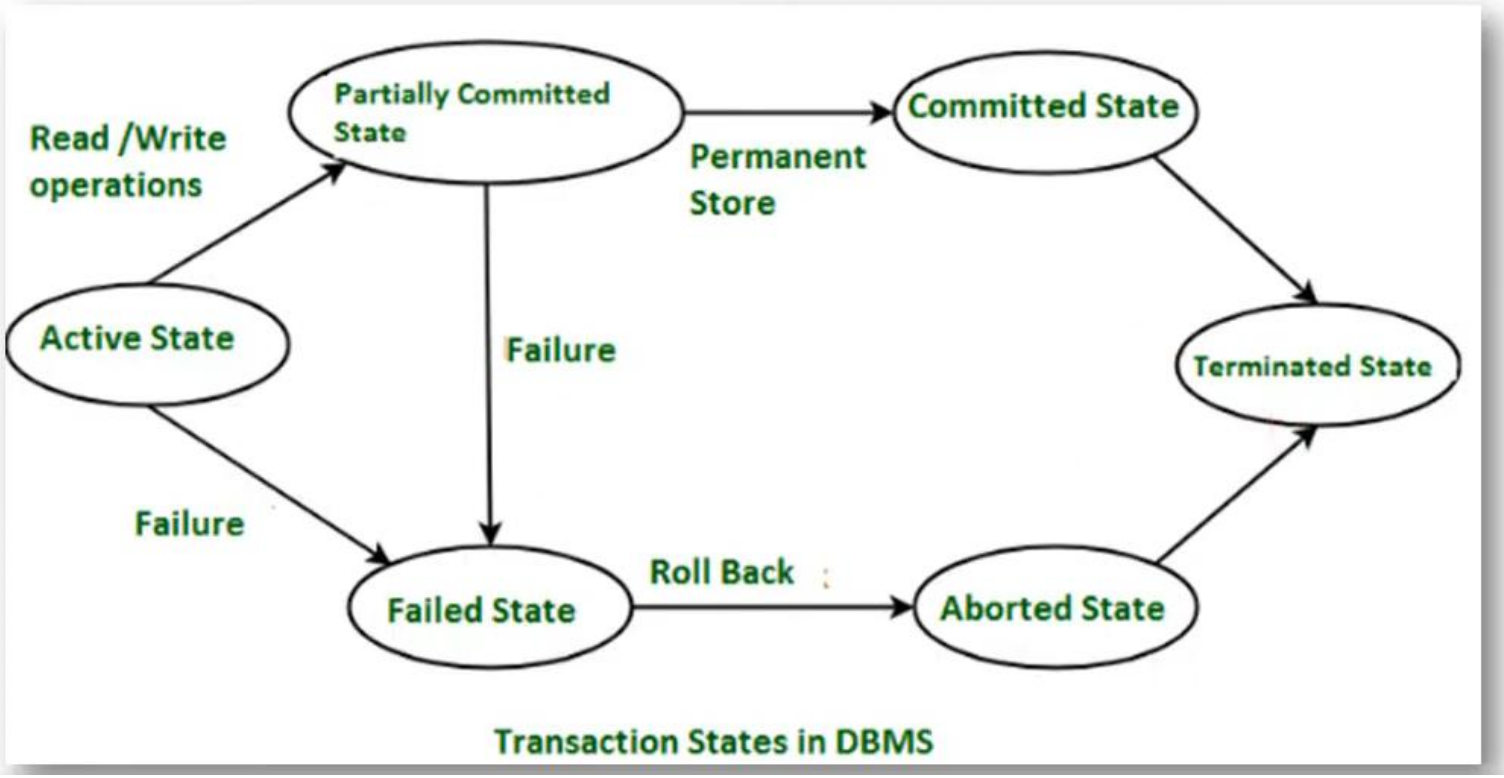


2. Write Operations:

- Write operation writes the updated data value back to the database from the buffer.
- Example: Write(A) will write the updated value of A from the buffer to the database.



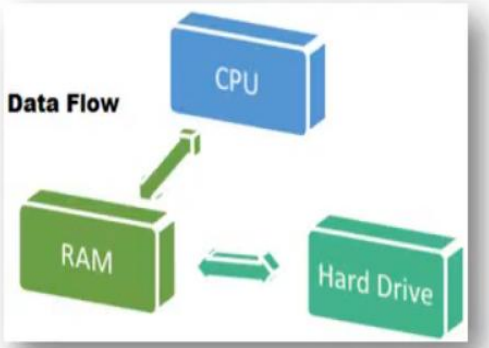
Transection State



1. Active State

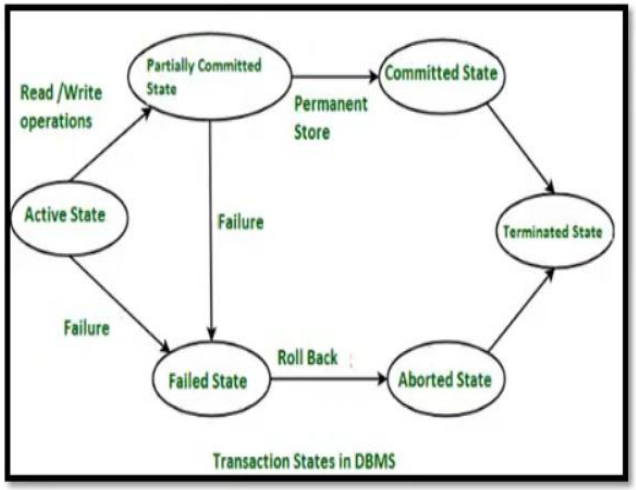
- This is the first state in the life cycle of a transaction.
- A transaction is called in an active state as long as its instructions are getting executed.
- All the changes made by the transaction now are stored in the buffer in main memory.

Examples:



Operating System

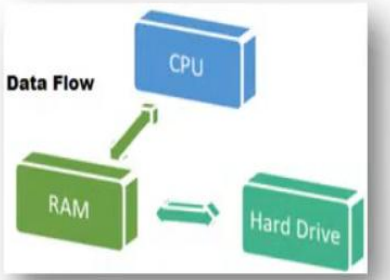
ATM



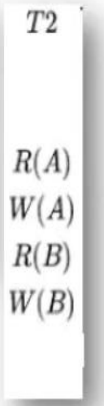
2. Partially Committed State

- After last instruction of transaction has executed, it enters into a partially committed state.
- After completion of all the read & write operation changes are made in main memory or buffer.
- If the changes are made permanent on the Database then the state will change to "committed state" and in case of failure it will go to the "failed state".

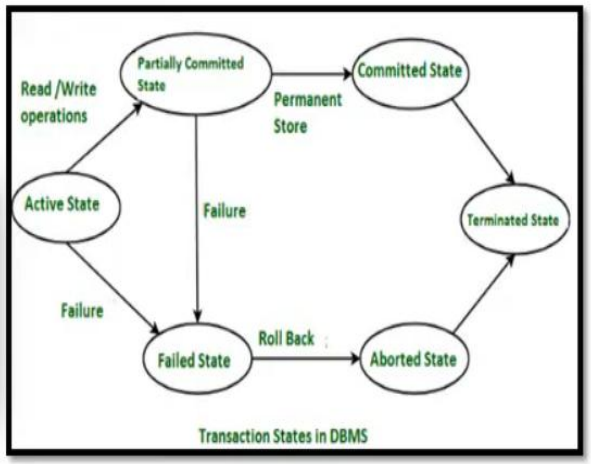
Examples:



Operating System



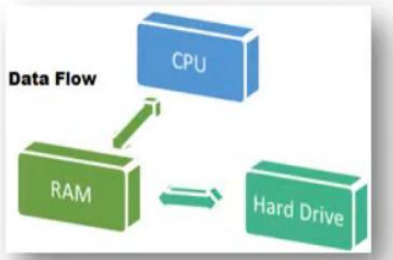
ATM



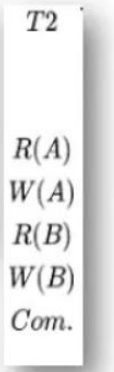
3. Committed State

- A transaction is said to be in a committed state if it executes all its operations successfully.
- After all the changes made by the transaction have been successfully stored into the database.
- Now, the transaction is considered to be fully committed.

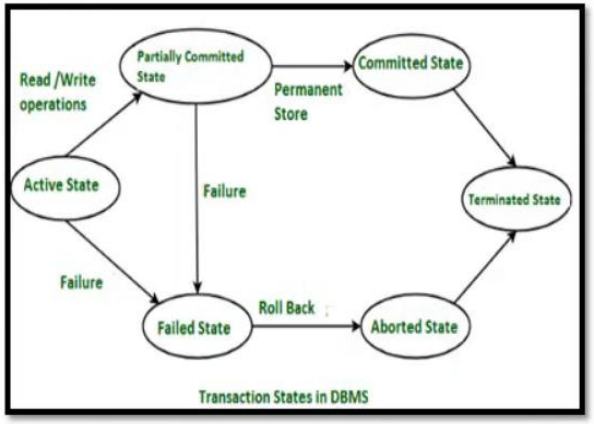
Examples:



Operating System



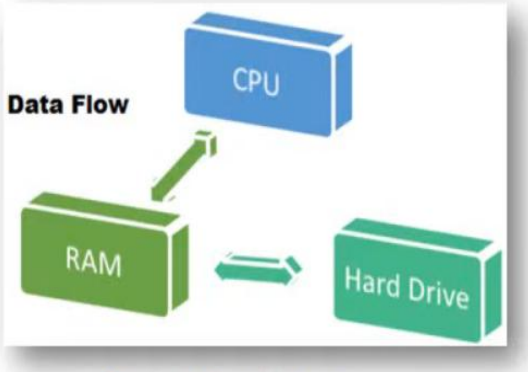
ATM



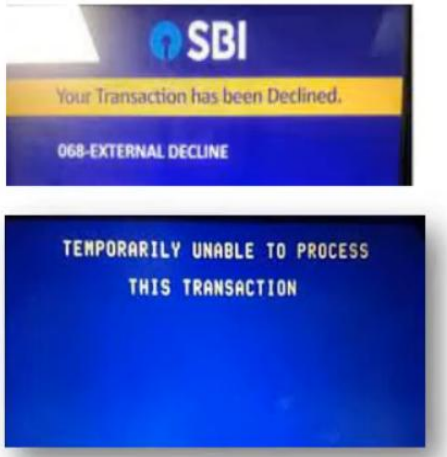
4. Failed State

- When a transaction is getting executed in the active state or partially committed state and some failure occurs.
- Due to which it becomes impossible to continue the execution, it enters into a failed state.
- Like, Electricity off, Battery down, Server error, No Internet Connection

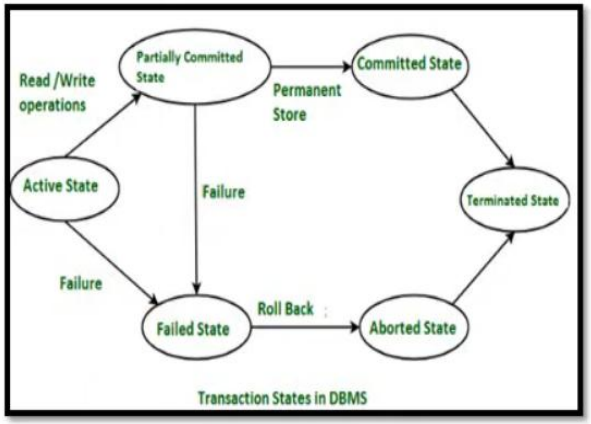
Examples:



Operating System

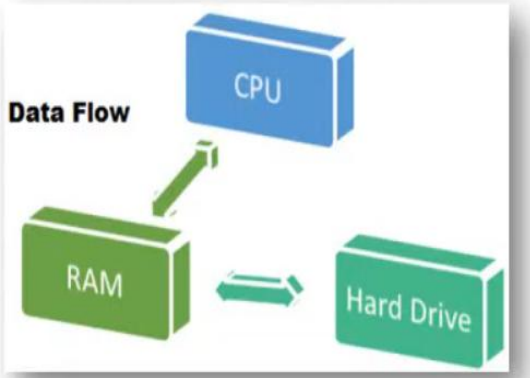


ATM

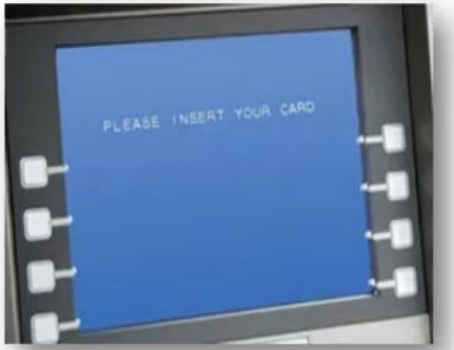


5. Aborted State

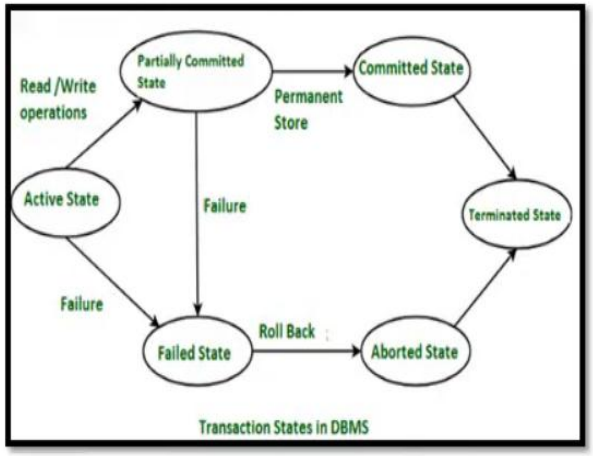
- After the transaction has failed and entered into a failed state.
- After aborting the transaction, the database recovery module will select one of the two operations: 1. Re-start the transaction 2. Kill the transaction
- To undo the changes made by the transaction, it becomes necessary to roll back the transaction.
- After the transaction has rolled back completely, it enters into an aborted state.



Operating System



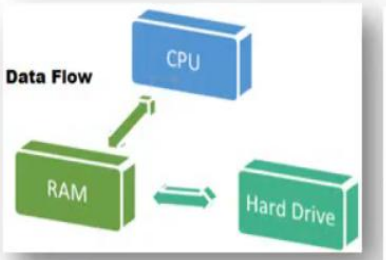
ATM



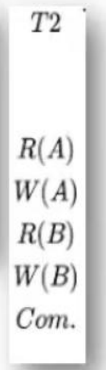
6. Terminated State

- This is the last state in the life cycle of a transaction.
- After entering the committed state or aborted state, the transaction finally enters into a terminated state where its life cycle finally comes to an end.
- The system is consistent and ready for new transaction and the old transaction is terminated.

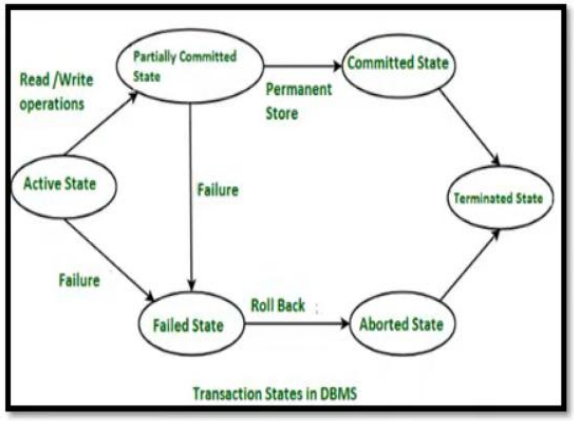
Examples:



Operating System

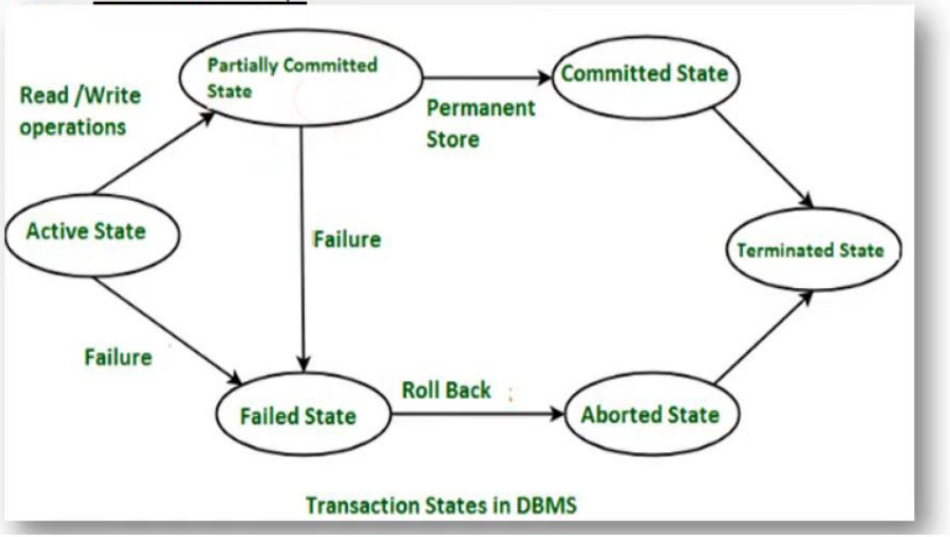


ATM



Transection State Revision

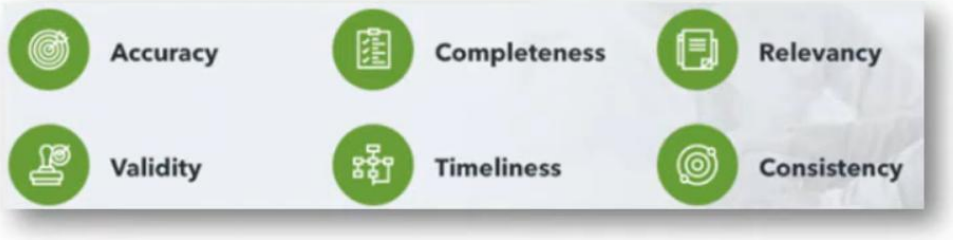
- **Active:** Transaction is executing.
- **Failed:** Transaction fails to complete successfully.
- **Abort:** changes made by transaction are cancelled (roll back).
- **Partially Commit:** Final statement of transaction is executed.
- **Commit:** Transaction completes its execution successfully.
- **Terminated:** Transaction is finished.



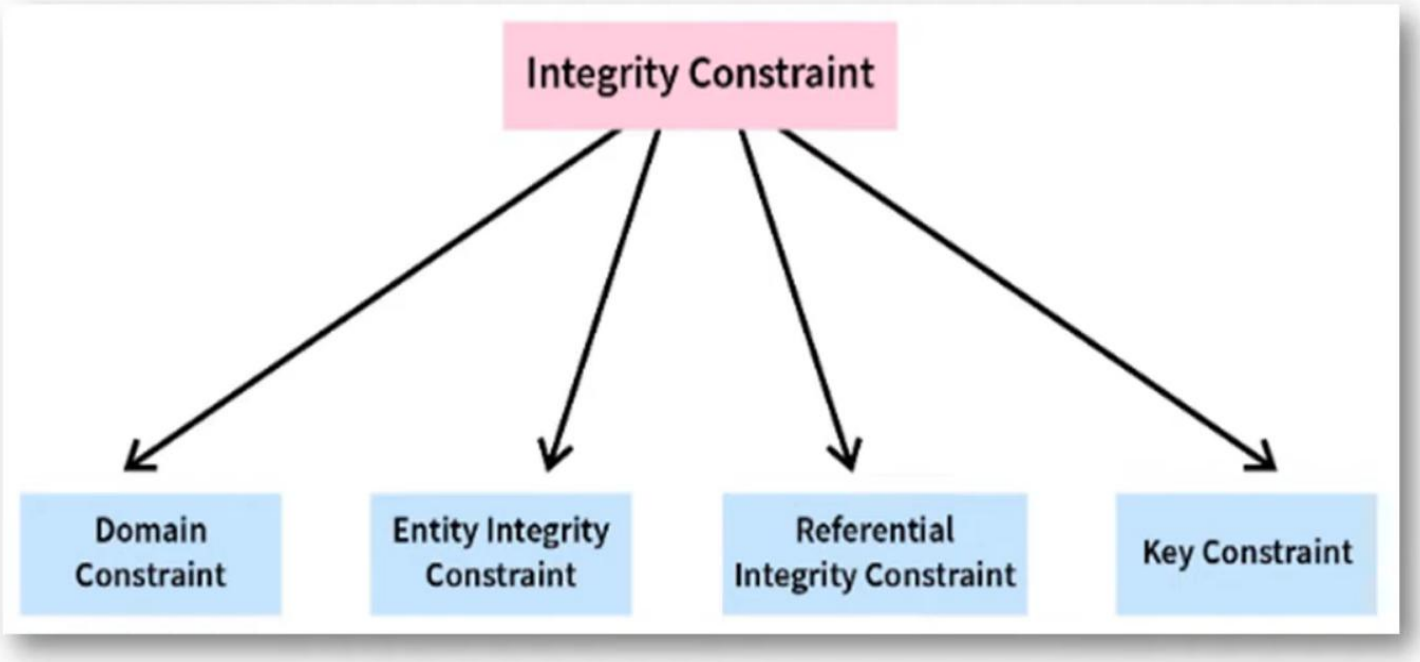
Integrity Constraints

What is Integrity Constraints?

- Integrity constraints are a set of rules or protocols.
- It is used to maintain the quality of information in your table in database.
- These are used to restrict the invalid types of information that can be entered into a table.
- Integrity constraints ensure that the data insertion, updating and other processes have to be performed more accurately.
- Thus, integrity constraint is used to guard against accidental damage to the database.
- You may apply integrity Constraints at the column or table level.
- The table-level Integrity constraints apply to the entire table, while the column level constraints are only applied to one column.



Types of Integrity Constraints



Type 1: Domain Constraints

- It definition of a valid set of values for an attribute.
- It contain atomic values only, it means composite and multi-valued attributes are not allowed.
- The data type of a domain can be string, integer, character, DateTime, currency, etc.
- If we assign the datatype of attribute age as int, we cant give it values other then int datatype.

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1004	Morgan	8th	A

Not allowed. Because AGE is an integer attribute

Type 2: Entity Integrity Constraints

- The entity integrity constraint states that primary key value can't be null.
- A primary key is used to identify individual records in a table and if the primary key has a null value, then we can't identify those records.
- There can be null values anywhere in the table except the primary key column.

Example:

EMPLOYEE

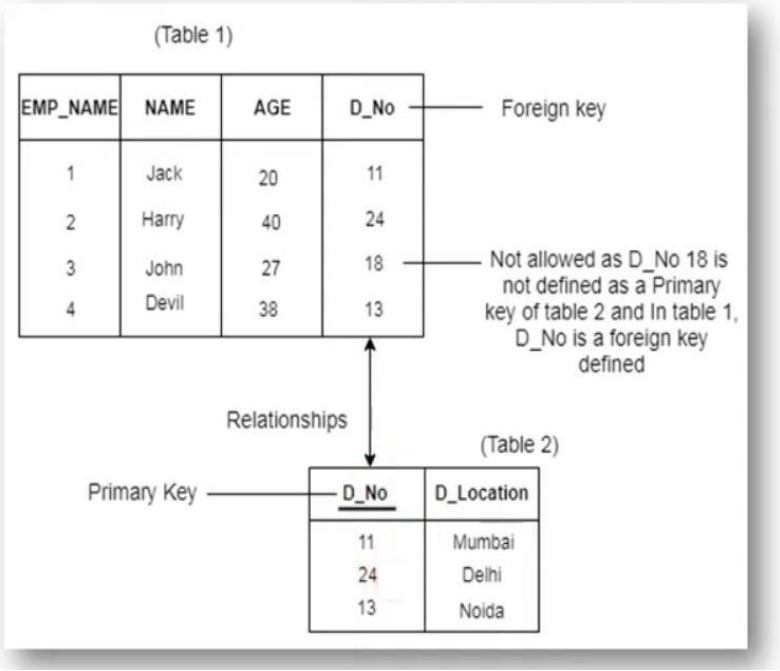
EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

Type 3: Referential Integrity Constraints

- It specified between two tables in database and used to maintain the consistency among them.
- If a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Example:



Type 4: Key Constraints

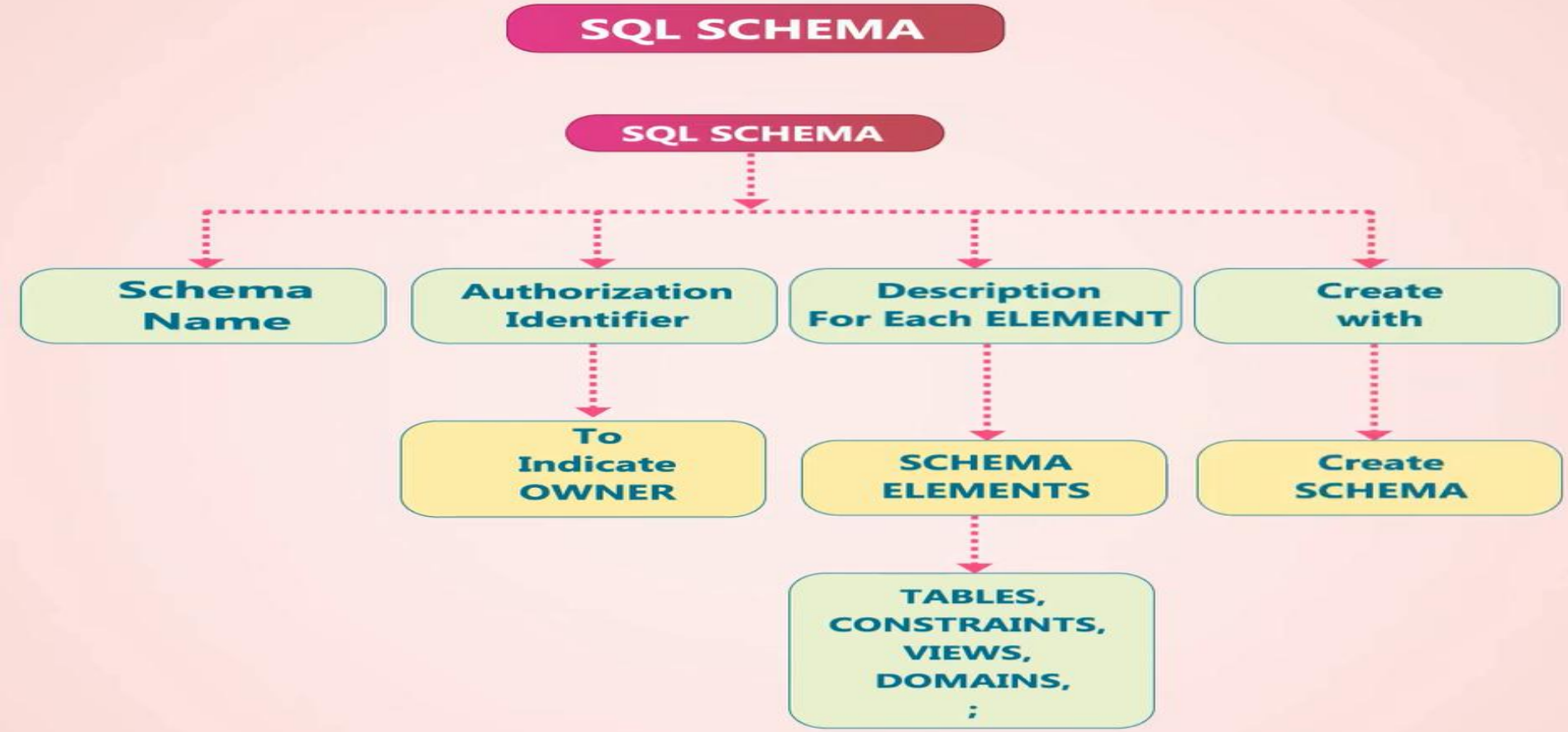
- These are called uniqueness constraints since it ensures that every tuple in the relation should be unique.
- An entity set can have multiple keys, but out of which one key will be the primary key.
- A primary key can contain a unique and null value in the relational table.

Example:

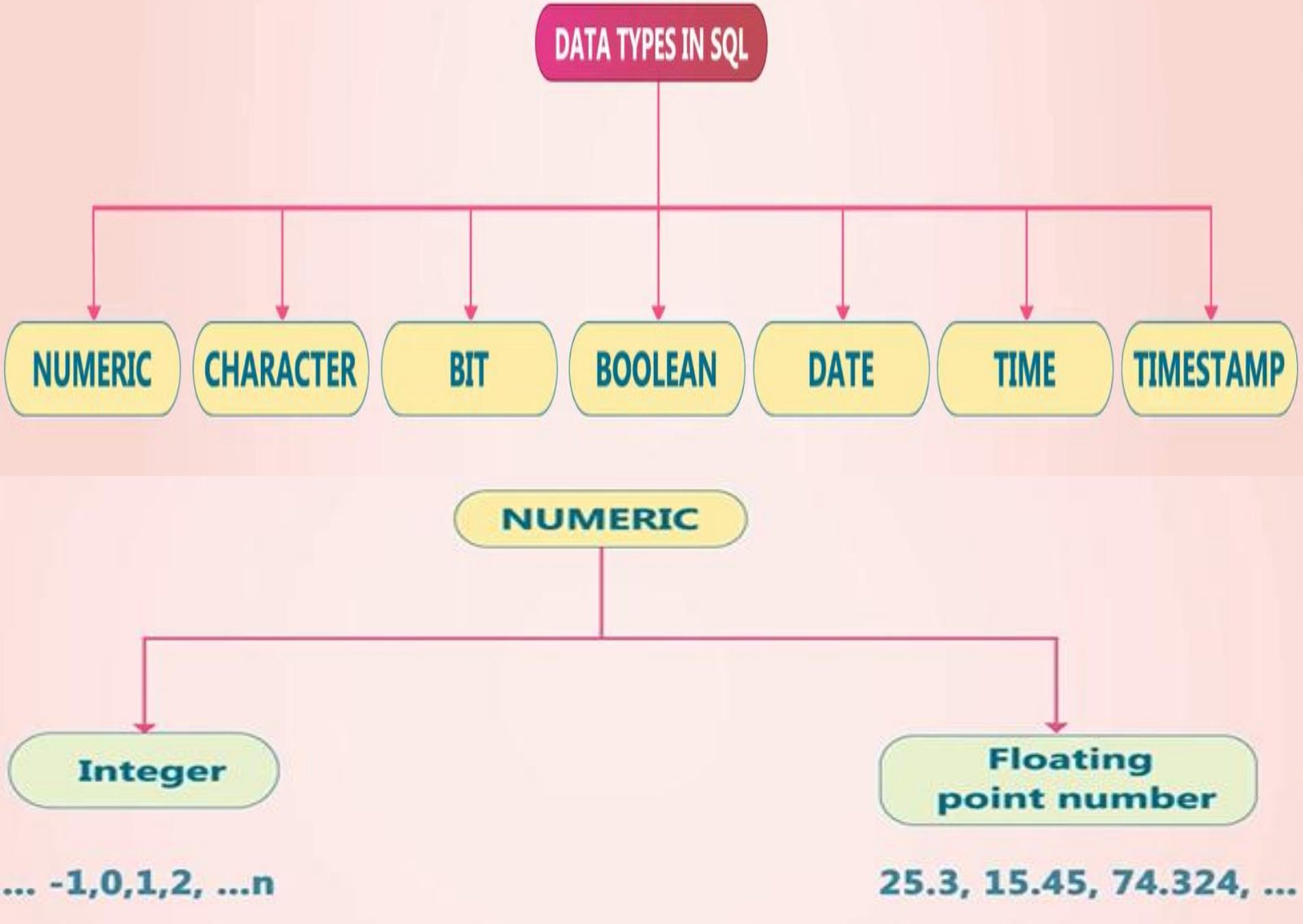
ID	NAME	SEMENSTER	AGE
1000	Tom	1st	17
1001	Johnson	2nd	24
1002	Leonardo	5th	21
1003	Kate	3rd	19
1002	Morgan	8th	22

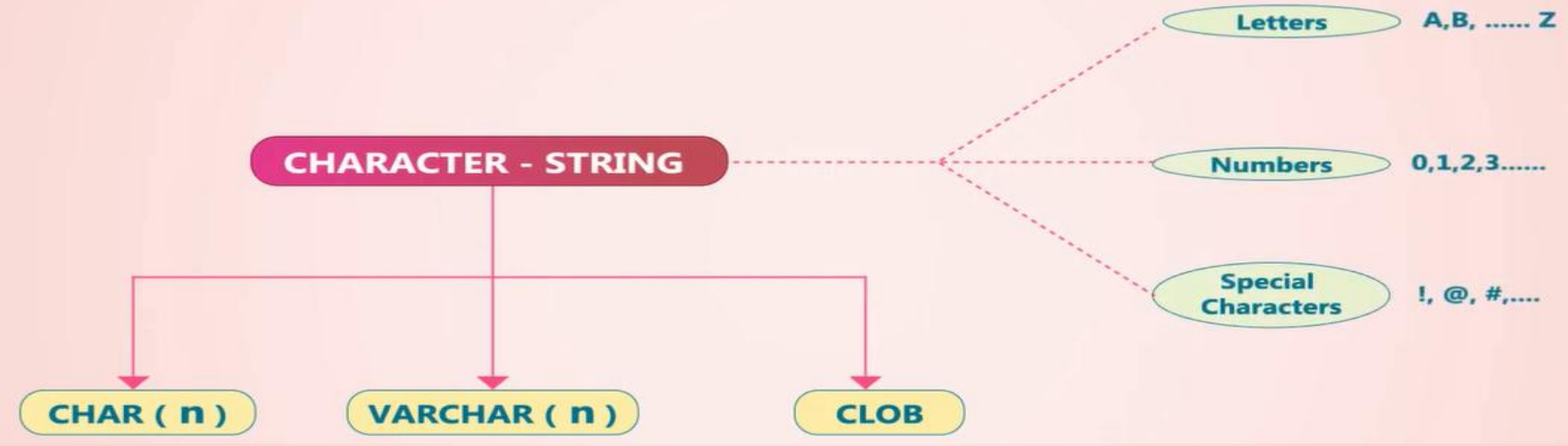
Not allowed. Because all row must be unique

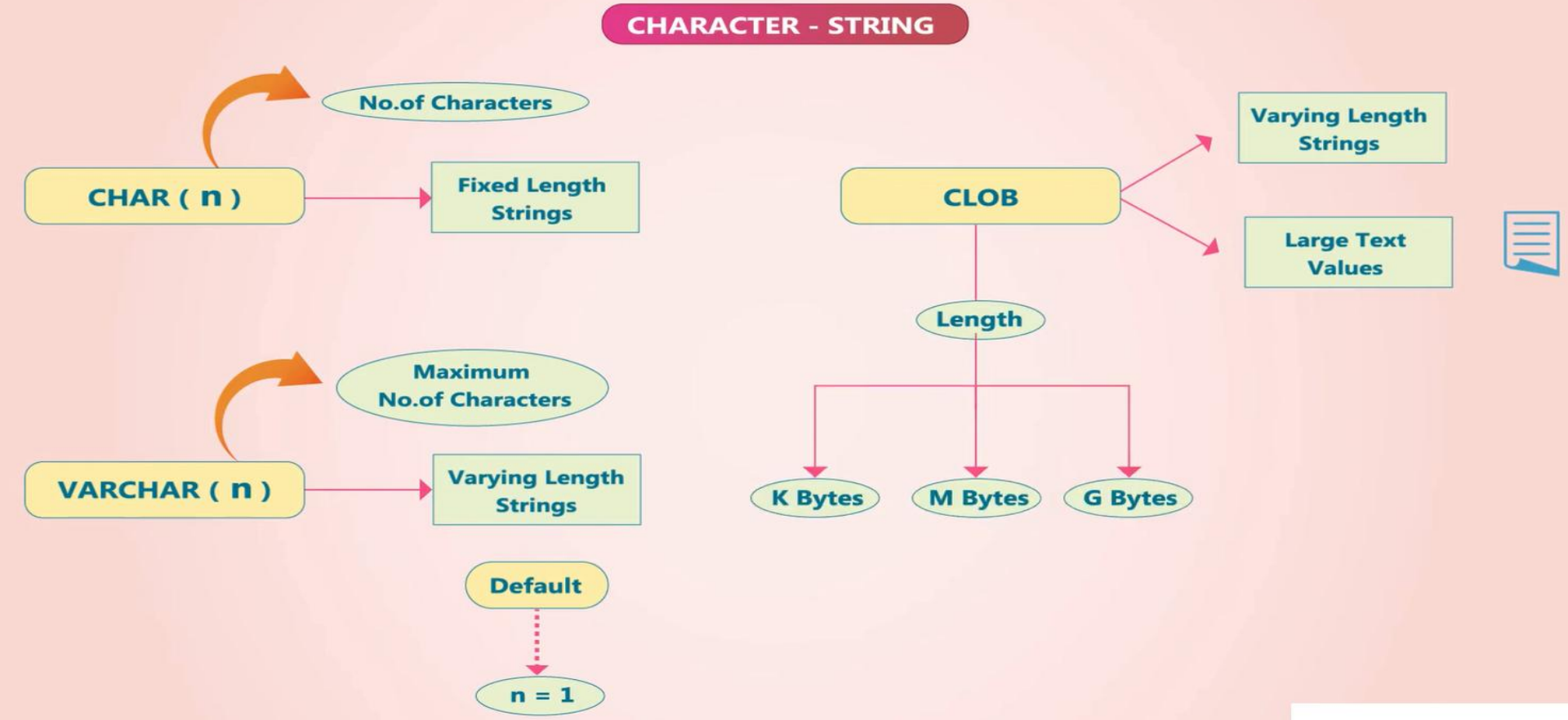
Data Types and Schemas

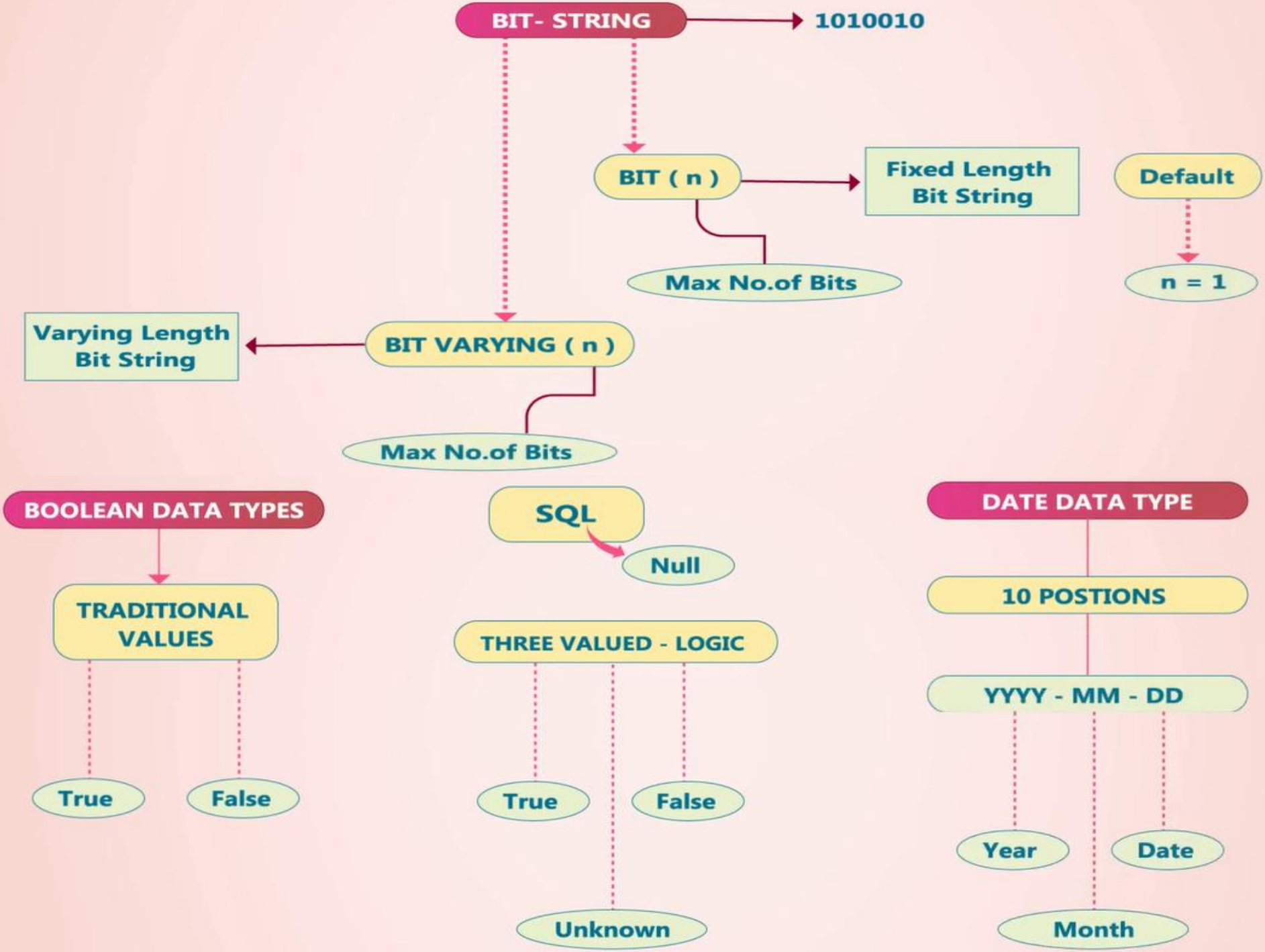


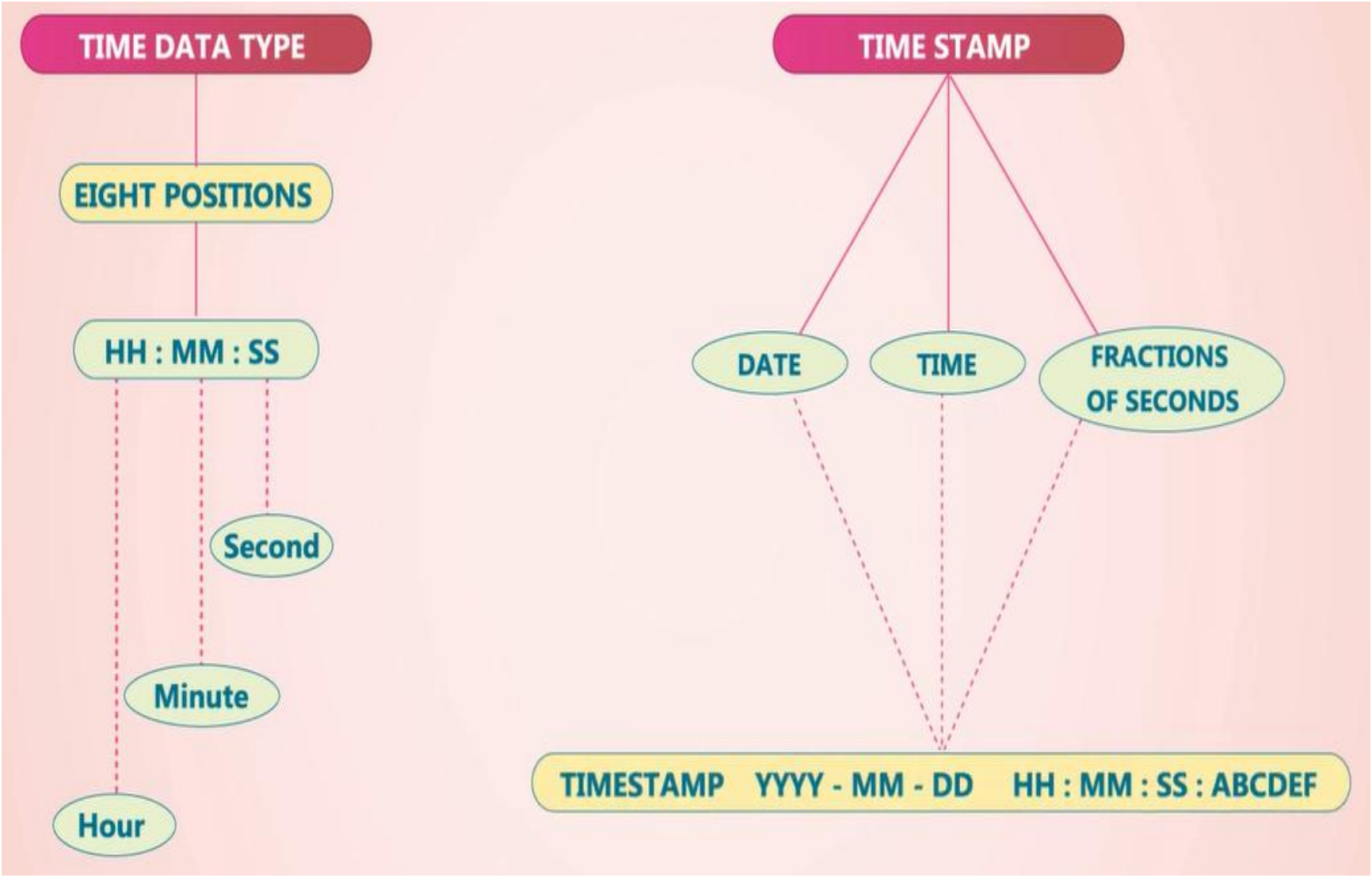
EXAMPLE


DATA TYPES IN SQL
NUMERIC
CHARACTER
BIT
BOOLEAN
DATE
TIME
TIMESTAMP
NUMERIC
Integer
Floating point number
... -1,0,1,2, ...n
25.3, 15.45, 74.324, ...


CHARACTER - STRING
Letters
A,B, ... Z
Numbers
0,1,2,3.....
Special Characters
!, @, #,....
CHAR (n)
VARCHAR (n)
CLOB


CHARACTER - STRING
No.of Characters
CHAR (n)
Fixed Length Strings
Maximum No.of Characters
VARCHAR (n)
Varying Length Strings
Default
n = 1
CLOB
Varying Length Strings
Large Text Values
Length
K Bytes
M Bytes
G Bytes


BIT- STRING
1010010
BIT (n)
Fixed Length Bit String
Default
Max No.of Bits
n = 1
Varying Length Bit String
BIT VARYING (n)
Max No.of Bits
BOOLEAN DATA TYPES
SQL
Null
DATE DATA TYPE
TRADITIONAL VALUES
10 POSTIONS
THREE VALUED - LOGIC
YYYY - MM - DD
True
False
True
False
Year
Date
Unknown
Month


TIME DATA TYPE
EIGHT POSITIONS
HH : MM : SS
Second
Minute
Hour
TIME STAMP
DATE
TIME
FRACTIONS OF SECONDS
TIMESTAMP YYYY - MM - DD HH : MM : SS : ABCDEF

Index in SQL

- An **index in SQL** is a data structure that provides a **quick way to look up data** from a table based on the values in one or more columns.
- An index is like a **roadmap** that helps the database engine to find the desired data on a **fast track.**

How to Work An Index in SQL?

- SQL indexes create a separate data structure that stores a subset of the data from one or more columns in a table. This data structure is optimized for quick lookups and is organized in a way that reduces the time it takes to find specific rows that match a query condition.

- Imagine you have a table of customer information with thousands of entries, and you want to find all customers from a particular city. Without an index, the database would have to scan every row in the table, one by one, to identify the matching records. However, with a properly created index, the database can efficiently jump directly to the relevant rows, saving time and resources.

Importance of An Index in SQL

There are various reasons that show the importance of an index in SQL as follows –

- **Faster Data Retrieval:** Indexes speed up data retrieval operations, making queries run much faster, especially on large datasets.
- **Improved Query Performance:** Indexes help optimize query performance, which is crucial for applications with heavy database usage.
- **Enforcing Data Integrity:** Unique indexes ensure that values in the indexed column(s) are unique, preventing duplicates.
- **Facilitating Joins:** Indexes on join columns improve the performance of JOIN operations when combining data from multiple tables.

Type of an Index in SQL

There are mainly 5 types of indexes in SQL as follows –

- **Single-Column Index:** Indexes created on a single column, called single-column index.
- **Multi-Column Index (Composite):** Indexes created on multiple columns, called composite index or multi-column index.
- **Unique Index:** A unique index ensures the uniqueness of values in specific indexed columns.
- **Clustered Index:** Clustered index dictates the physical order of data rows in specific indexed columns.
- **Non-clustered Index:** Non-clustered index provides a separate structure for indexing data.

1. Creation of Single-Column Index

In this example, we will create a single-column index named 'idx_Employee' on a column named 'EmployeeID' in the 'Employee' table using the following statement –

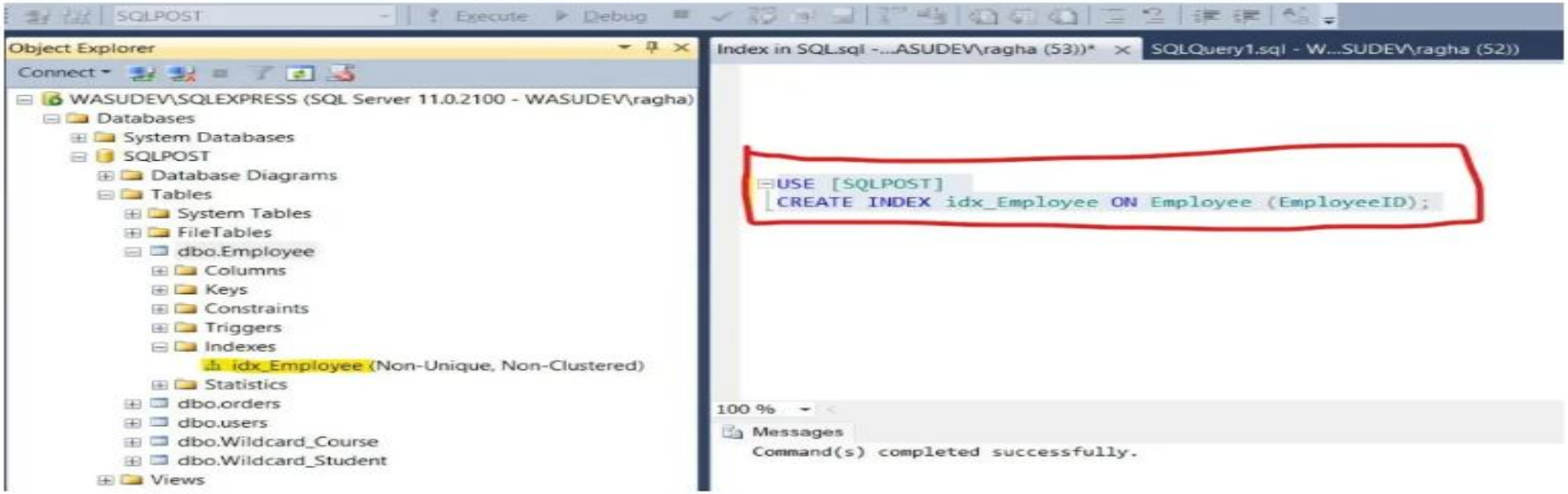
Syntax:

```
CREATE INDEX Index_Name
ON Table_Name ( Column_Name);
```

Example:

```
USE [SQLPOST]
CREATE INDEX idx_Employee
ON Employee (EmployeeID);
```

Screenshot: Below screenshot for reference –



Result: You can see in the above screenshot that the non-unique and non-clustered index named 'idx_Employee' is created on 'EmployeeID' single column of 'Employee' table.

2. Creation of Multi-Column Index

If you want to create an index on multiple columns then in this example, we will create a multi-column index named 'idx_Employee_Name_Des_Dep_Sal' on multiple columns i.e. 'EmployeeName', 'Designation', 'Department', and 'Salary' etc in the 'Employee' table using the following statement –

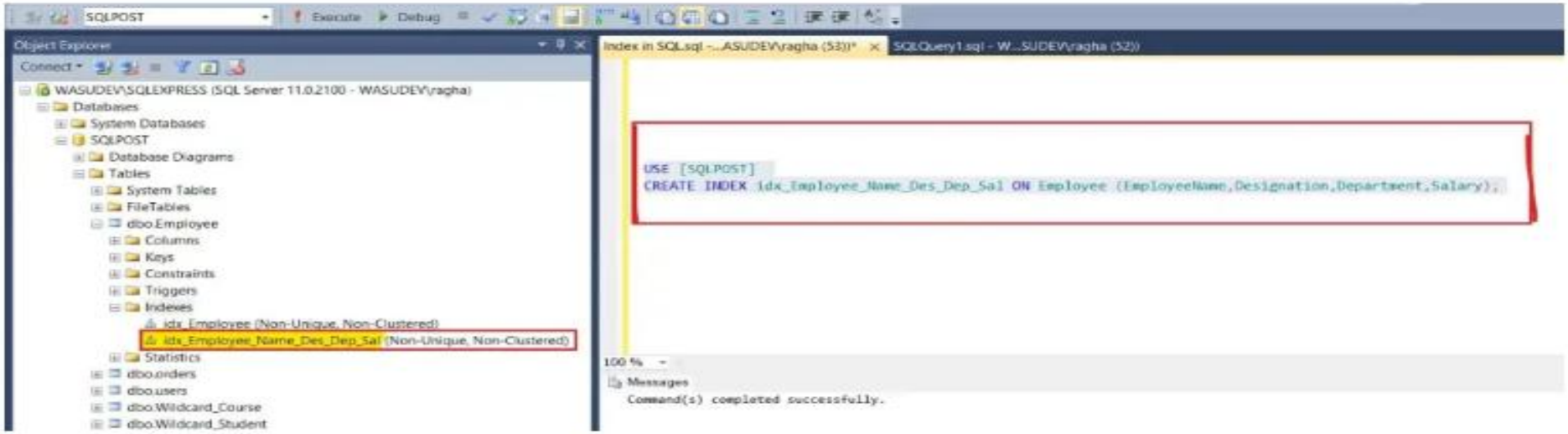
Syntax:

```
CREATE INDEX Index_Name
ON Table_Name ( column_name1, column_name2, …., column_nameN);
```

Example:

```
USE [SQLPOST]
CREATE INDEX idx_Employee_Name_Des_Dep_Sal
ON Employee (EmployeeName,Designation,Department,Salary);
```

Screenshot: Below screenshot for reference –



Result: You can see in the above screenshot that the non-unique and non-clustered index named 'idx_Employee_Name_Des_Dep_Sal' is created on multiple columns i.e. 'EmployeeName', 'Designation', 'Department', and 'Salary' etc in the 'Employee' table.

3. Creation of Unique Index

Unique index is similar to the primary key constraint and does not allow duplicate values in the column where it is defined. In this example, we will create a unique index named 'ux_Employee_EmployeeID' on a single column i.e. 'EmployeeID' in the 'Employee' table using the following statement –

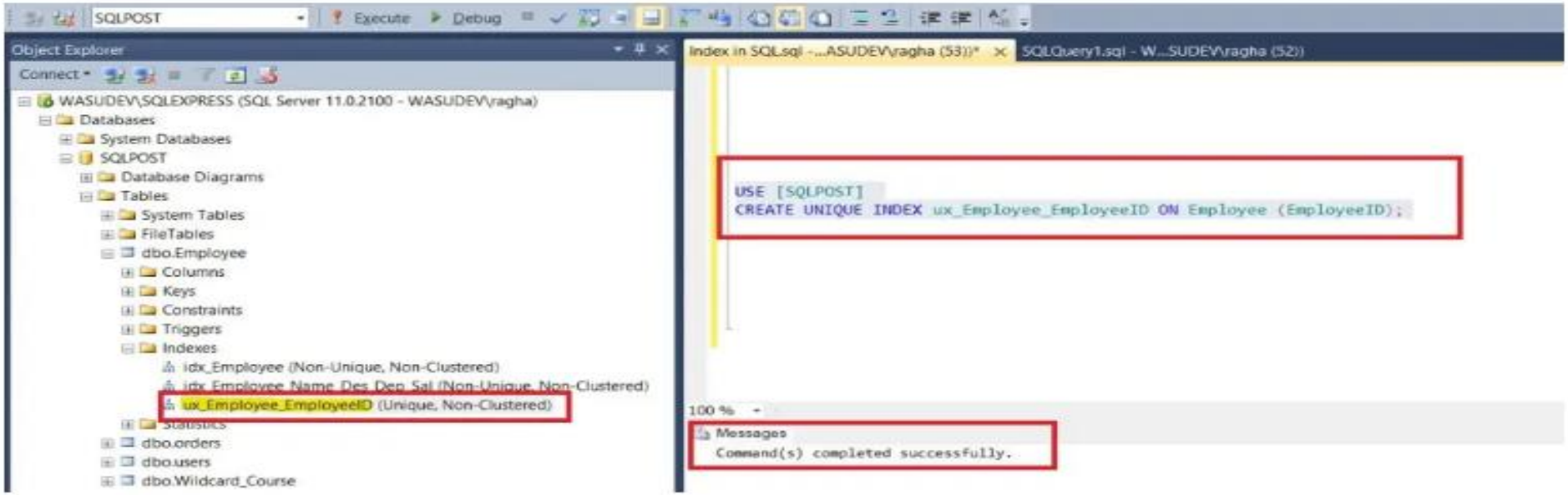
Syntax:

```
CREATE UNIQUE INDEX Index_Name
ON Table_Name ( Column_Name);
```

Example:

```
USE [SQLPOST]
CREATE UNIQUE INDEX ux_Employee_EmployeeID
ON Employee (EmployeeID);
```

Screenshot: Below screenshot for reference –



Result: You can see in the above screenshot that the unique and non-clustered index named 'ux_Employee_EmployeeID' is created on a single column i.e. 'EmployeeID' in the 'Employee' table.

4. Creation of Clustered Index

In this example, we will create a clustered index named 'CX_Employee_EmployeeID' on a single column i.e. 'EmployeeID' in the 'Employee' table using the following statement –

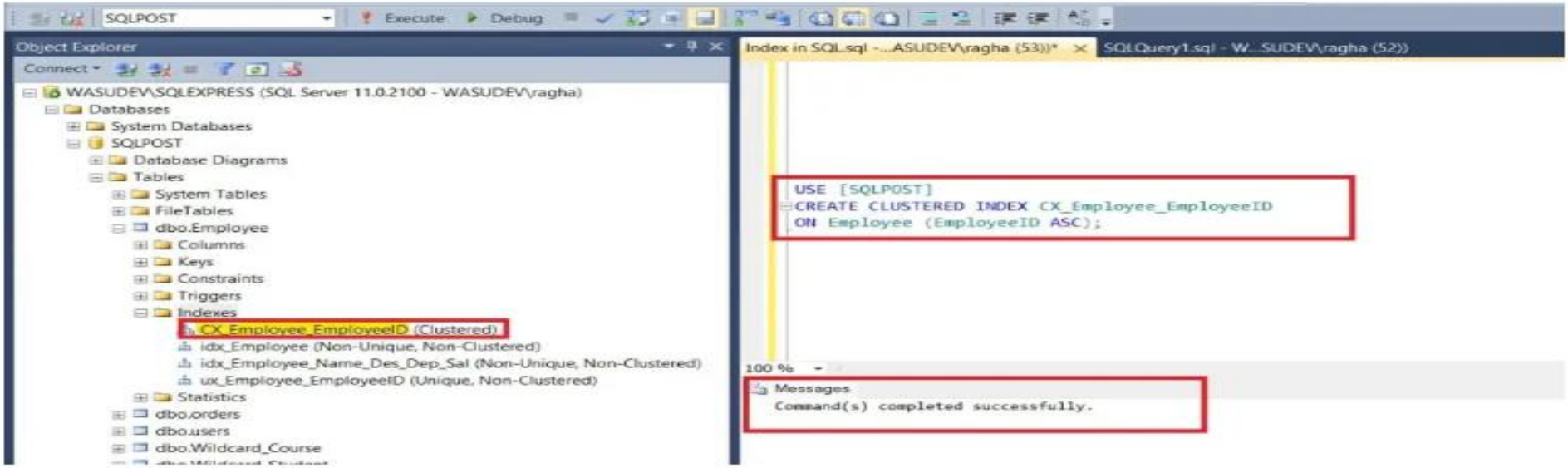
Syntax:

```
CREATE CLUSTERED INDEX Index_Name
ON Table_Name ( Column_Name ASC/DESC);
```

Example:

```
USE [SQLPOST]
CREATE CLUSTERED INDEX CX_Employee_EmployeeID
ON Employee (EmployeeID ASC);
```

Screenshot: Below screenshot for reference –



Result: You can see in the above screenshot that the clustered index named 'CX_Employee_EmployeeID' is created on a single column i.e. 'EmployeeID' in the 'Employee' table.

5. Creation of Non-Clustered Index

In this example, we will create a non-clustered index named 'NCX_Employee_EmployeeID' on a single column i.e. 'EmployeeID' in the 'Employee' table using the following statement –

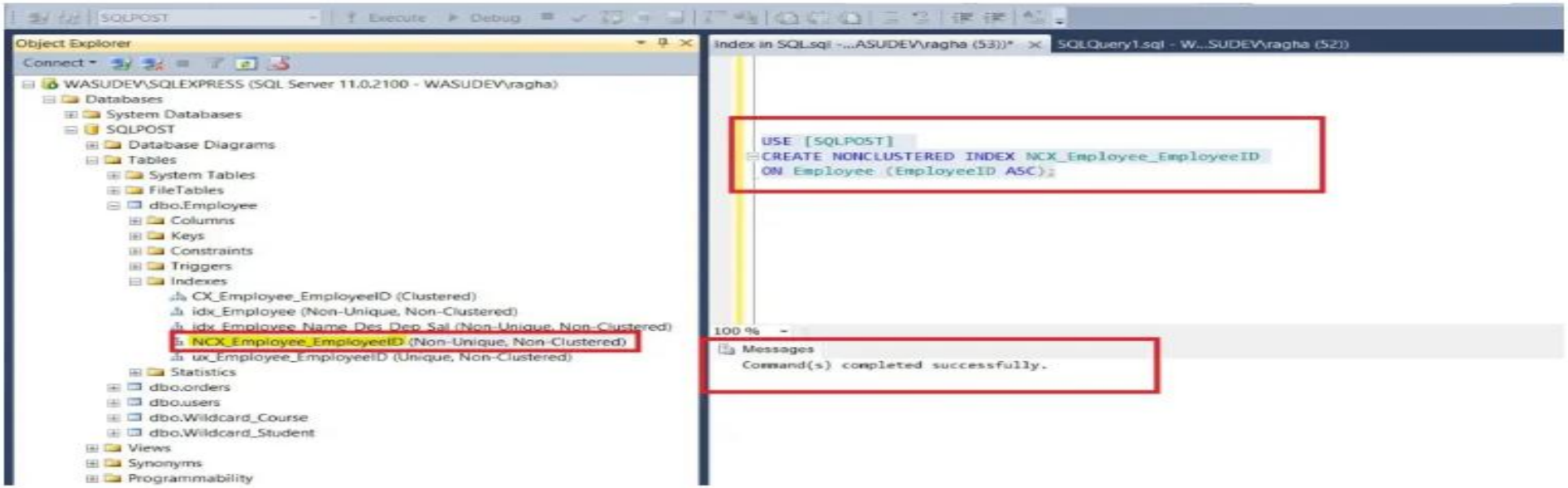
Syntax:

```
CREATE CLUSTERED INDEX Index_Name
ON Table_Name ( Column_Name ASC/DESC);
```

Example:

```
USE [SQLPOST]
CREATE NONCLUSTERED INDEX NCX_Employee_EmployeeID
ON Employee (EmployeeID ASC);
```

Screenshot: Below screenshot for reference –



Result: You can see in the above screenshot that the non-unique and non-clustered index named 'NCX_Employee_EmployeeID' is created on a single column i.e. 'EmployeeID' in the 'Employee' table.

6. Renaming of Index

In this example, we will rename an index named 'idx_Employee_Name_Des_Dep_Sal' with a new index named 'IDX_Employee_NDDS' in the 'Employee' table using the following statement –

Syntax:

```
EXEC sp_rename 'Table_Name.Old_Index_Name', 'New_Index_Name';
--OR
EXEC sp_rename 'Table_Name.Old_Index_Name', 'New_Index_Name', N'INDEX';
```

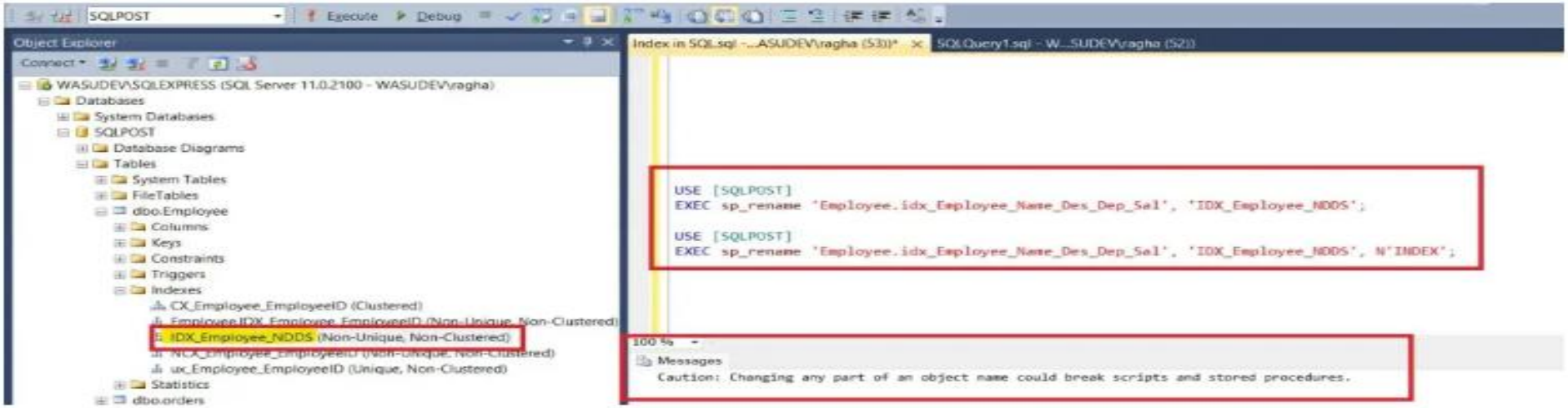
Example:

```
USE [SQLPOST]
EXEC sp_rename 'Employee.idx_Employee_Name_Des_Dep_Sal', 'IDX_Employee_NDDS';
```

— **OR**

```
USE [SQLPOST]
EXEC sp_rename 'Employee.idx_Employee_Name_Des_Dep_Sal', 'IDX_Employee_NDDS',
N'INDEX';
```

Screenshot: Below screenshot for reference –



Result: You can see in the above screenshot that the non-unique and non-clustered index named 'idx_Employee_Name_Des_Dep_Sal' is renamed with a new name 'IDX_Employee_NDDS' in the 'Employee' table.

7. Deletion or Removal of Index

In this example, we will remove/delete/drop an index named 'IDX_Employee_NDDS' from the 'Employee' table using the following statement –

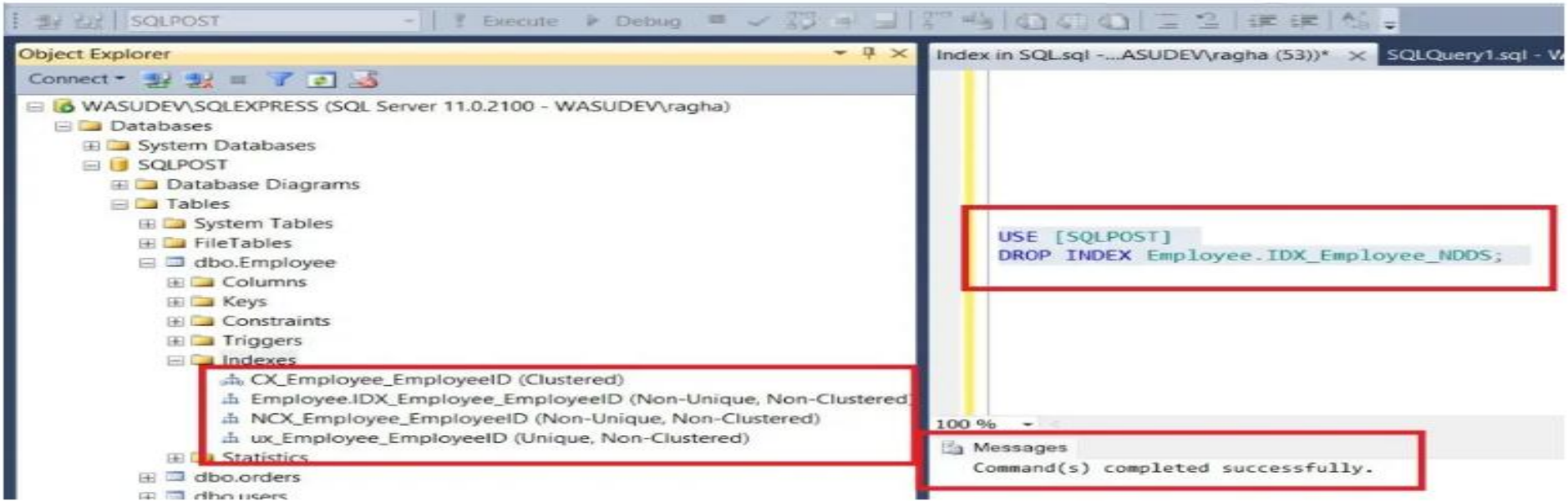
Syntax:

```
DROP INDEX Table_Name.Index_Name;
```

Example:

```
USE [SQLPOST]
DROP INDEX Employee.IDX_Employee_NDDS;
```

Screenshot: Below screenshot for reference –



Result: You can see in the above screenshot that the non-unique and non-clustered index named 'IDX_Employee_NDDS' is removed/deleted/dropped from the 'Employee' table.

8. Reorganize of Index

In this example, we will reorganize an index named 'IDX_Employee_NDDS' from the 'Employee' table using the following statement –

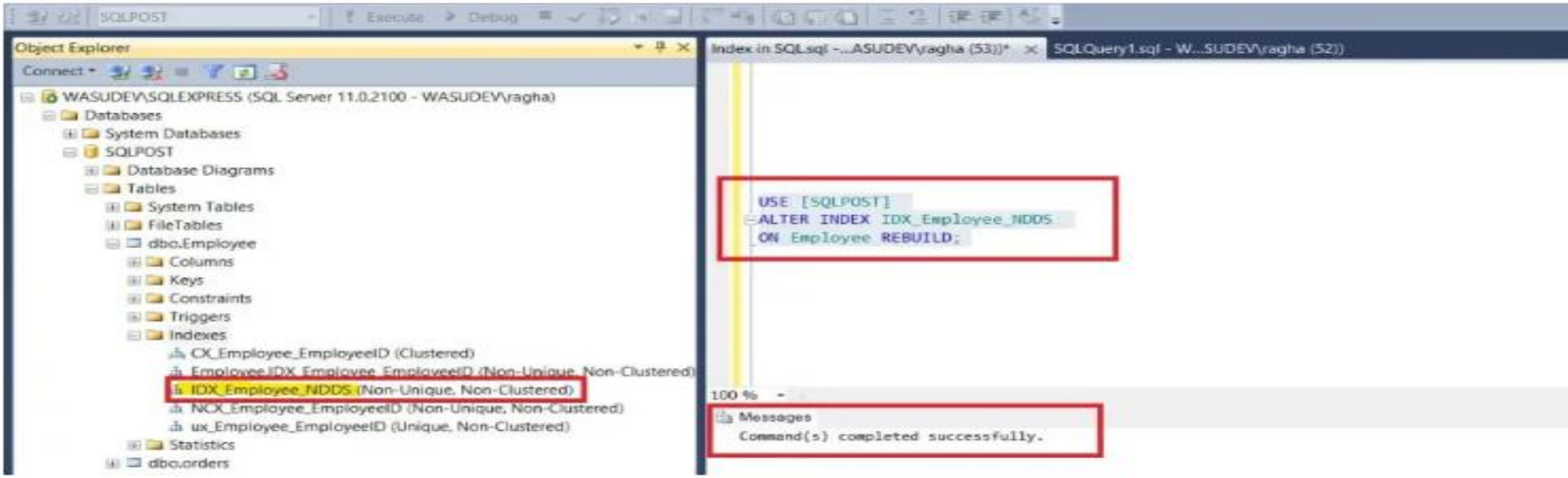
Syntax:

```
ALTER INDEX Index_Name
ON Table_Name REORGANIZE;
```

Example:

```
USE [SQLPOST]
ALTER INDEX IDX_Employee_NDDS
ON Employee REORGANIZE;
```

Screenshot: Below screenshot for reference –



Result: You can see in the above screenshot that the non-unique and non-clustered index named 'IDX_Employee_NDDS' from the 'Employee' table is reorganized now.

9. Rebuild of Index

In this example, we will rebuild an index named 'IDX_Employee_NDDS' from the 'Employee' table using the following statement –

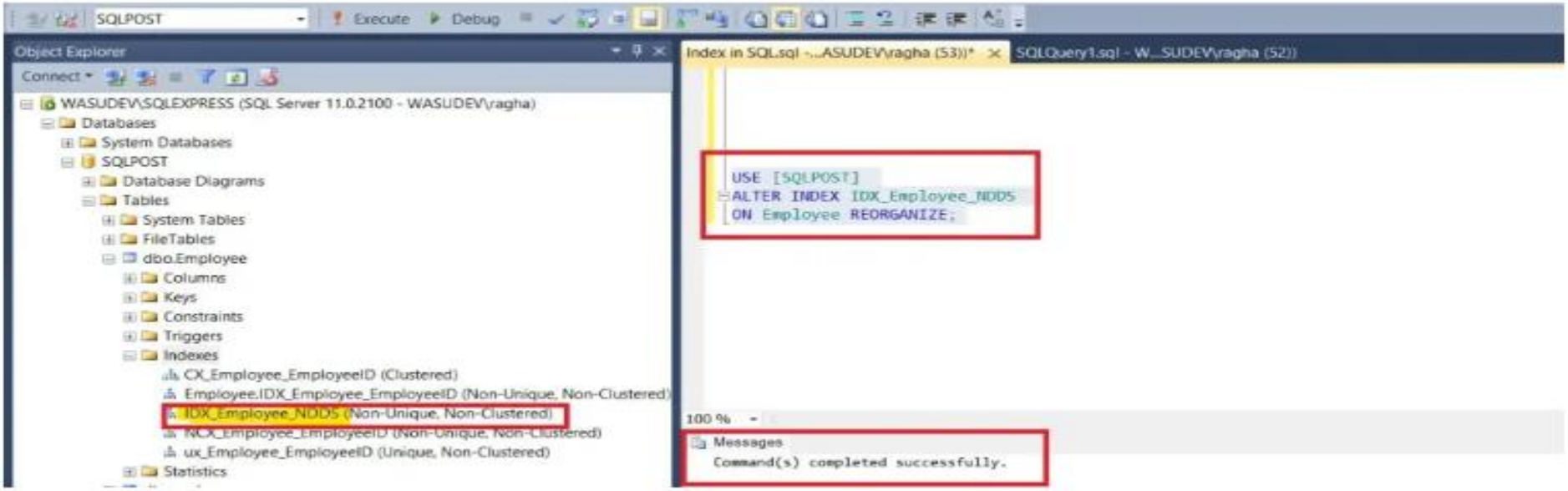
Syntax:

```
ALTER INDEX Index_Name
ON Table_Name REBUILD;
```

Example:

```
USE [SQLPOST]
ALTER INDEX IDX_Employee_NDDS
ON Employee REBUILD;
```

Screenshot: Below screenshot for reference –



Result: You can see in the above screenshot that the non-unique and non-clustered index named 'IDX_Employee_NDDS' from the 'Employee' table is rebuilt now.

Accessing SQL from a Programming Language

- **SQL**: widely used non-procedural, commercial language
- Example: Find the name of the instructor with ID 22222

```
select   name
from     instructor
where    instructor.ID = '22222'
```

- **SQL** is NOT a Turing equivalent language which means that everything that need to be computed cannot be computed in SQL
- **SQL** alone is not very powerful; because there are some computations that cannot be obtained by any SQL query. Such computations must be written in a host language, such as C, C++, Java with embedded SQL queries that access the data in the database
  - Application Programs are programs that are used to interact with the database in this fashion. Eg: Banking system are programs that generate: Payroll checks, Debit accounts, Credit Accounts or transfer funds between accounts
  - To access the database, DML statements need to be executed from the host language
- **Application Programs generally access the database through one of the 2 ways:**
  1. APIs (ODBC/JDBC) which allows SQL Queries to be sent to database
  2. Language extensions to allow embedded SQL

Programming Language generally access the Data Base through one of the 2 ways

1. *By providing an Application Programming Interface i.e API (the set of procedures) that can be used to send DDL & DML statements (SQL Queries) to the database and retrieve the results* i.e. **APIs (ODBC/JDBC) which allows SQL Queries to be sent to database**

   - Eg: 1. ODBC (Open Database Connectivity) standard
     - It is created by Microsoft in 1992 that is used by Windows software applications, using C/C++ language, to access databases via SQL

     2. JDBC (Java Database Connectivity) standard
     - It is created by Sun Microsystems in 1997 that is used by JAVA application, using only JAVA Language, to access databases via SQL

2. *By extending the host language syntax to embed DML calls within the host language program* i.e **Language extensions to allow embedded SQL**

   - Eg: C,C+, JAVA with embedded SQL queries
   - Usually special character (like EXEC) prefaces DML Calls, and a Preprocessor called the DML Pre-Compiler, that converts the DML statements to normal procedure calls in the host language

SQL Procedures

What is an SQL Procedure?

An SQL Procedure contains a group of sql statements which solve a common purpose.

Syntax:

```
Create proc <procedureName>
As
.... Statements...
```

Consider a simple SQL Procedure:

```
Create proc printProcedure
As
Print 'Hello World'
go
```

Print is a command in SQL Server 2008. It is used to print a string on the screen.

NOTE: The SQL Procedures that we learn here are stored in the system by the DBMS. They are hence known as **stored procedures**

Altering stored procedures

How to alter a SQL stored procedure?

A Stored procedure can be altered using the **alter proc** command.

Syntax:

```
Alter proc<procedureName>
As
.... New Statements...
```

Consider a simple SQL Procedure:

```
Alter proc printProcedure
As
Print 'New Hello World'
go
```

Executing Procedures

The Main advantage of using a stored procedure is its **reusability**, i.e., a procedure can be called any time that it needs to be executed.

An SQL procedure can be executed using the **exec** command:

```
Exec <ProcedureName>
```

Example:

```
Create proc printProcedure
As
Print 'New Hello World'
Go

Exec printProcedure
```

OUTPUT:

New Hello World

Procedure Parameters

A Procedure, like a C or C++ Procedure(function) can take up input parameters and produce return values.

A Procedure with Input Parameters:

```
Create proc hellop
@info1 int= 1,
@info2 int
As
Print @info1+@info2
```

Calling a Procedure with parameters:

```
Exec proc hellop 3,2
```

Calling a Procedure with parameters:

Exec proc hellop 3,2

Here, the value '3' over-rides the default value. If a paramater does not have a default value defined for it, then the DBMS requires the user to pass a value for it before execution.

Eg: exec proc hellop 1: will return an error as the second parameter is not assigned any value.

Output Parameters:

A Procedure can have output parameters also. The Output parameters are specified by the keyword **output.**

Consider a procedure to add two numbers:

```
Create proc adder
@num1 int =0,
@num2 int =0,
As
Declare @num3 int= @num1 + @num2;
Print @num3;
Go
```

Executing the procedure: exec proc adder 3,4 **Output:** 7

A Procedure can also return a value as: **return <value>**

NOTE

The Data members in SQL (@num1) resemble the variables in c/c++. But there is a big difference: They **can't be altered directly**.
Eg: @num1 = @num1 +1 Is invalid. The values are assigned at the time of their declaration only.

Using the Output parameters:

```
CREATE PROC Sales
  @CustomerID int,
  @AccNum varchar(10) OUTPUT
AS
SELECT @AccNum = AccNum
 FROM Sales.Customer
 WHERE CustomerID = @CustomerID;
GO
```

Calling a procedure with return values:

```
DECLARE @AccNum varchar(10);
EXEC GetCustomerAccountNumber, @AccNum OUTPUT;
PRINT @AccNum;
```

Deleting procedures

The SQL statement to drop/delete a procedure is:

```
Drop proc <procedureName>
```

Example:

```
Drop proc addNumbers;
Go;
```

Functions

What is a Function?

A Function is a set of sql statements which aim to accomplish the same task.

How is a function different from a procedure?

- A Function is more clear, organized and structured
- It is easy to modify
- It is easy to handle return values than in procedures. In procedures, a temporary table might be necessary to carry a return value and join it with an existing table. This becomes unnecessary if functions are used instead.

User-defined functions are routines written using **Transact-SQL** or the **Dot NET Common Language Runtime.**

Transact-SQL:

Transact-SQL (**T-SQL**) is an extension to SQL, created by Microsoft' and Sybase.
Transact-SQL enhances SQL with these additional features:

- Enhancements to DELETE and UPDATE statements
- Control-of-flow language
- Local variables
- Various support functions for string processing, date processing, mathematics, etc.

Dot Net Common Language Runtime(CLR):

It is a runtime environment which runs code and provides services that make development process much easier.

Functions are better

The Advantages of using functions in SQL Server 2008:

- Modular Programming
- Network Traffic Reduction
- Execution Plan Caching

The SQL Syntax for creating a function resembles that of a procedure:

```
Create function <functionName>
(argumentList)
 Returns <returnValueType>
 As
        Begin
        ...statements...
        Return <returnValue>
 End
Go
```

A Sample function:

Consider a function that takes two numbers and finds their sum:

```
Create function adder
(@num1 int,@num2 int)
Returns int
As
Begin
Declare @num3 int;
Set @num3 = @num1+@num2;
Return @num3;
End
go
```

Note that the **SET** keyword is used to change the value of a variable

Calling the function:

```
Print dbo.adder(2,4)
go
```

The Prefix **dbo** instructs the DBMS that you are the **database owner**

Deleting Functions

The SQL statement to drop/delete a function is similar to that of a procedure:

```
Drop function <procedureName>
```

Example:

```
Drop function addNumbers;
Go;
```

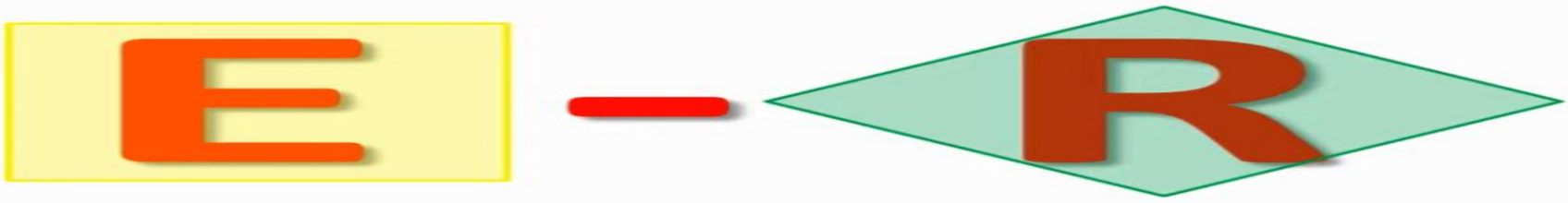
Summary

Summary

- **Procedures**
  - Creating Procedures
  - Modifying existing procedures
  - Deleting procedures
- **Functions**
  - Creating functions
  - Modifying existing functions
  - Deleting functions

DATABASE DESIGN PROCESS

Requirement Analysis	Requirement Specification
Conceptual Design	ER Model
Logical Design	Relational Model
Schema Refinement	Normalized Relations
Physical Design	File Organization & Access Methods
Application/Security Design	Access Control Policy



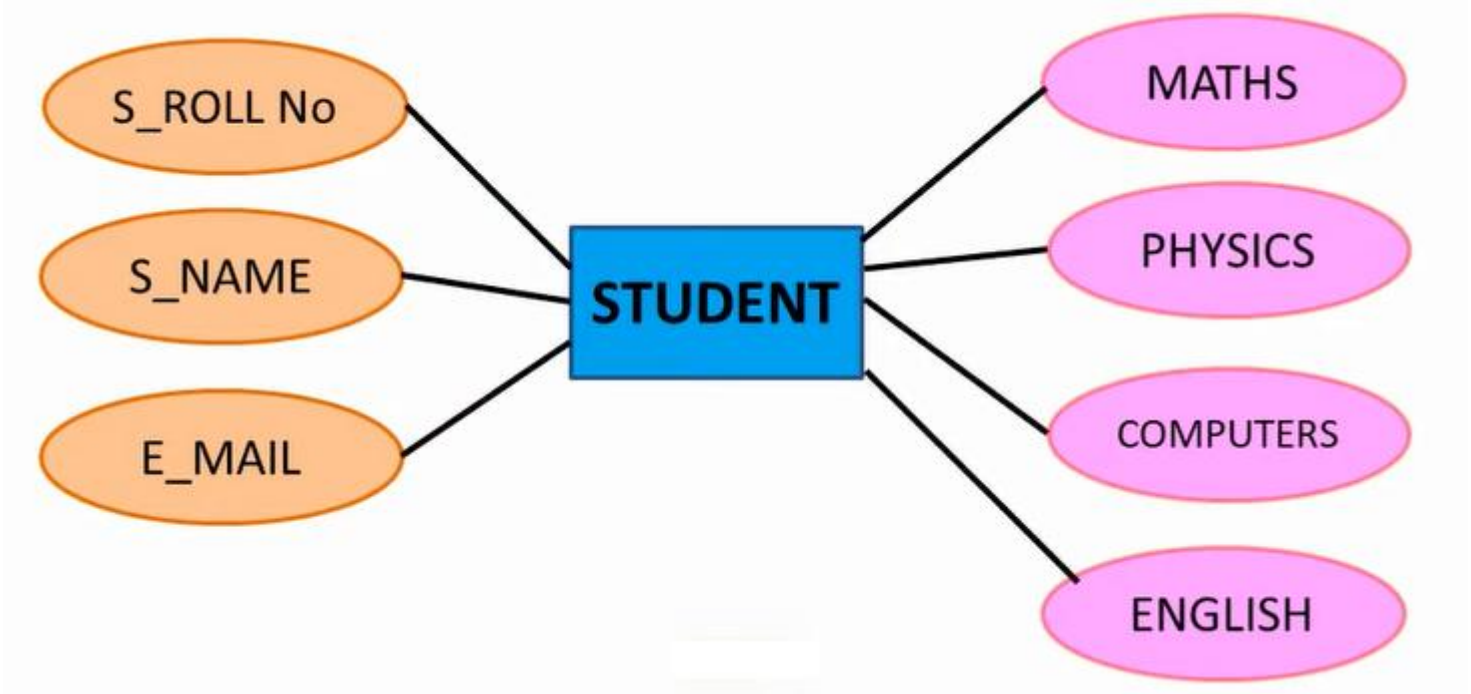
Model

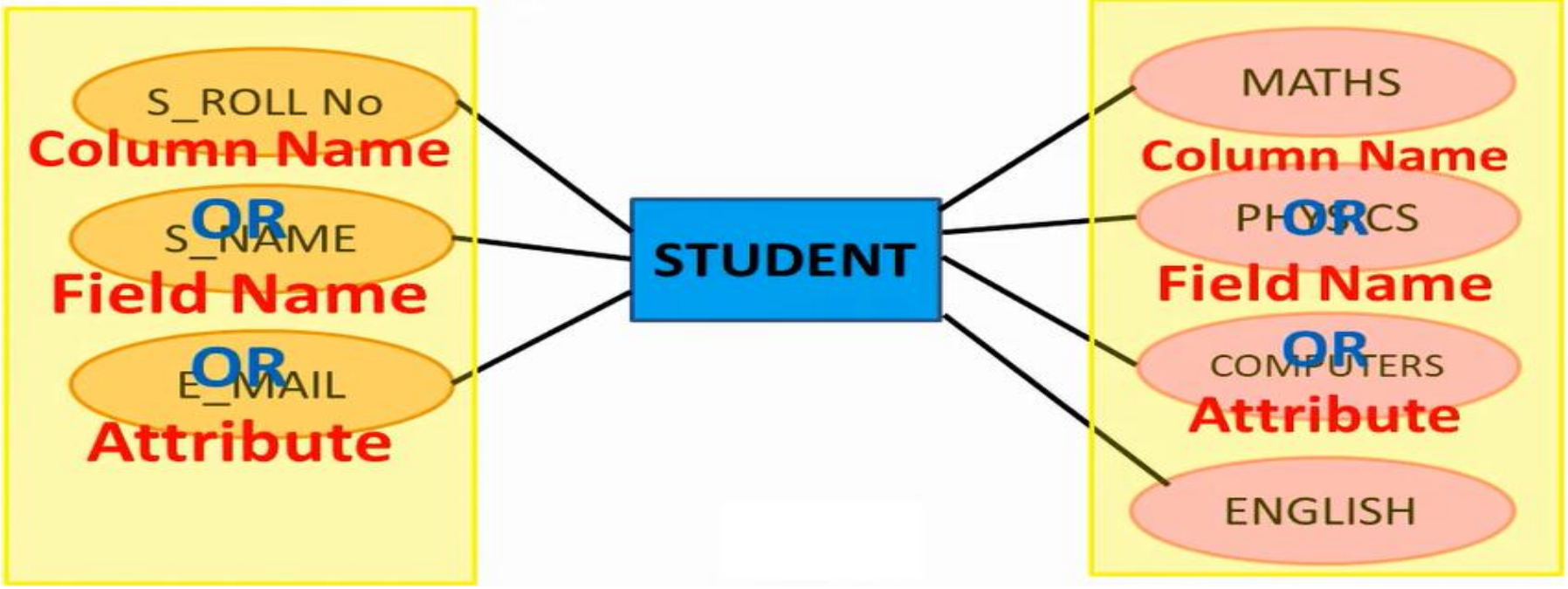
WHY IS E-R MODEL ?

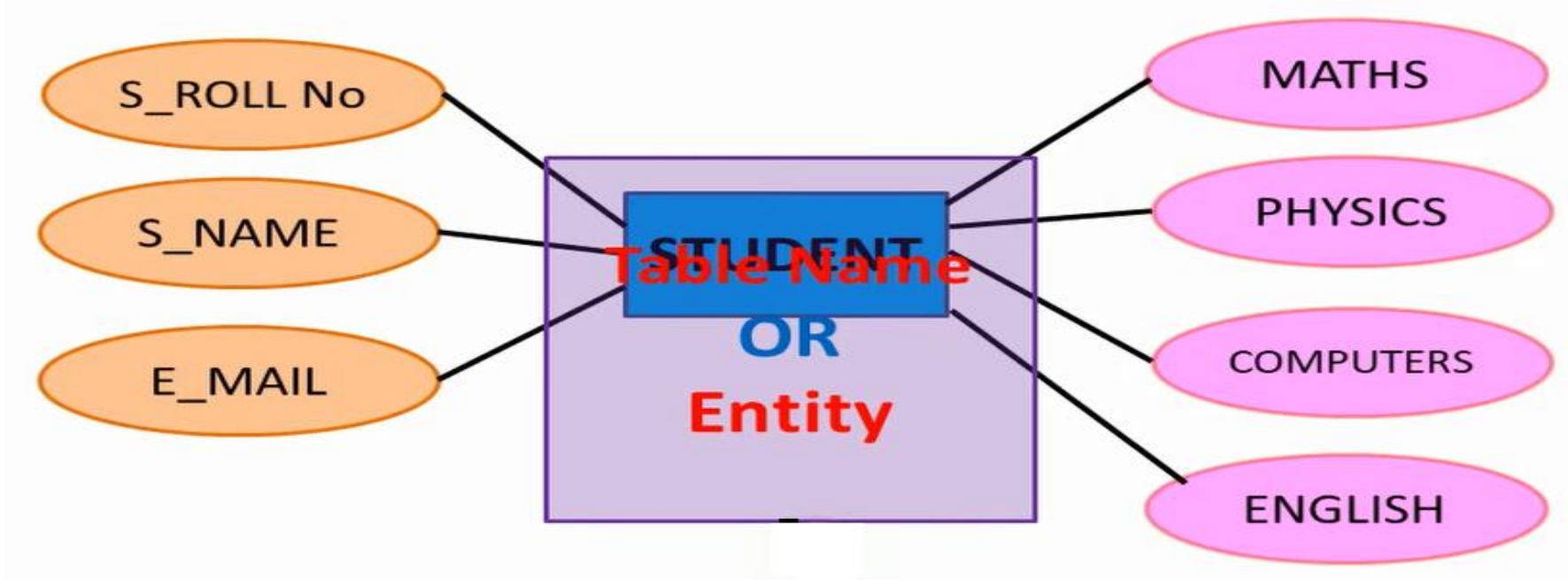
Student Table					
S_ROLL No	S_NAME	MATHS	PHYSICS	COMPUTERS	ENGLISH
102	RAVI	45	58	63	48
108	RAJU	78	86	45	56
103	SAI	58	78	56	45
114	RAVI	66	58	76	55

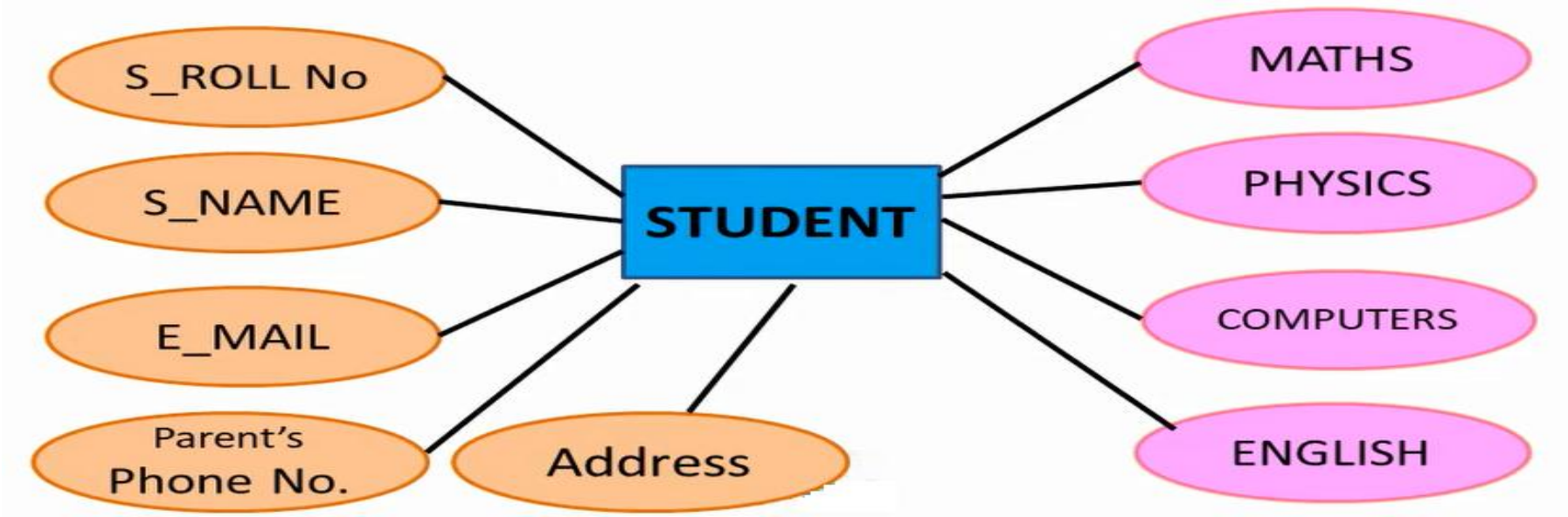
Student Table						
S_ROLL No	S_NAME	E_Mail	MATHS	PHYSICS	COMPUTERS	ENGLISH
102	RAVI		45	58	63	48
108	RAJU		78	86	45	56
103	SAI		58	78	56	45
114	RAVI		66	58	76	55

Student Table						
S_ROLL No	S_NAME	E_Mail	MATHS	PHYSICS	COMPUTERS	ENGLISH
102	RAVI		45	58	63	48
108	RAJU		78	86	45	56
103	SAI		58	78	56	45
114	RAVI		66	58	76	55




S_ROLL No
Column Name
OR
S_NAME
Field Name
OR
E_MAIL
Attribute
STUDENT
MATHS
Column Name
PHYSICS
OR
Field Name
COMPUTERS
OR
Attribute
ENGLISH


S_ROLL No
S_NAME
E_MAIL
STUDENT
Table Name
OR
Entity
MATHS
PHYSICS
COMPUTERS
ENGLISH



WHY IS E-R DIAGRAM ?

- Helps you to define terms related to ENTITY RELATIONSHIP MODELING
- Provide a PREVIEW of how all your TABLES SHOULD CONNECT, what fields are going to be on each table
- Helps to describe ENTITIES, ATTRIBUTES, RELATIONSHIPS
- As ER diagrams is a BLUEPRINT, they are TRANSLATABLE INTO RELATIONAL TABLES which allows you to BUILD DATABASES QUICKLY
- The database DESIGNER GAINS A BETTER UNDERSTANDING of the information to be contained in the database.

ER MODEL

- ER model stands for an ENTITY-RELATIONSHIP MODEL.
- This model is used to define the DATA ELEMENTS and RELATIONSHIP for a specified system.
- The ER model defines the CONCEPTUAL VIEW or a DESIGN or BLUEPRINT of a database.
- It is a HIGH-LEVEL DATA MODEL used to model the logical view of the system from DATA PERSPECTIVE having below components:

ENTITY , ENTITY TYPE & ENTITY SET

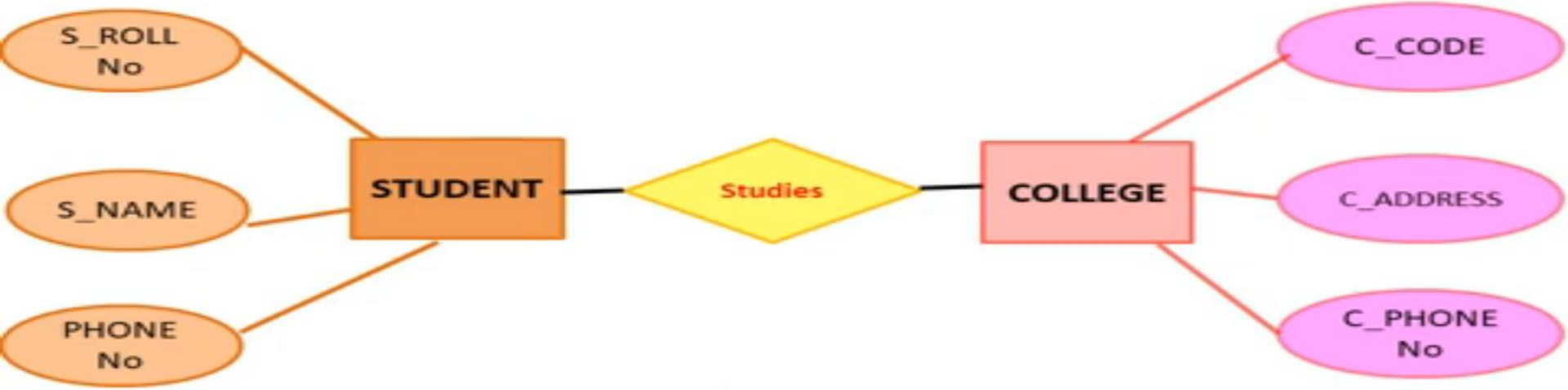
- In ER modeling, the database STRUCTURE IS PORTRAYED AS A DIAGRAM called an entity-relationship diagram.
- An Entity—relationship model (ER model) describes the STRUCTURE OF A DATABASE with the help of a DIAGRAM, which is known as Entity Relationship Diagram (ER Diagram).
- ER modeling helps you to ANALYZE DATA REQUIREMENTS SYSTEMATICALLY to produce a WELL-DESIGNED DATABASE.

Example - ER DIAGRAM

Student Table		
S_ROLL No	S_NAME	Phone No
102	RAVI	78569
108	RAJU	45876
103	SAI	12456
114	RAVI	45876

Studies

College Table		
C_Code	C_Address	C_Phone No
124	Hills Top, Street No. 4	7856984



Geometric Shapes in E-R Diagram

Rectangle: Represents Entity sets.

Ellipses: Attributes

Diamonds: Relationship Set

Lines: They link attributes to Entity Sets and Entity sets to Relationship Set

Double Ellipses: Multivalued Attributes

Dashed Ellipses: Derived Attributes

Double Rectangles: Weak Entity Sets

Double Lines: Total participation of an entity in a relationship set

The ER model defines the CONCEPTUAL VIEW or a DESIGN or BLUEPRINT of a database

ENTITY:

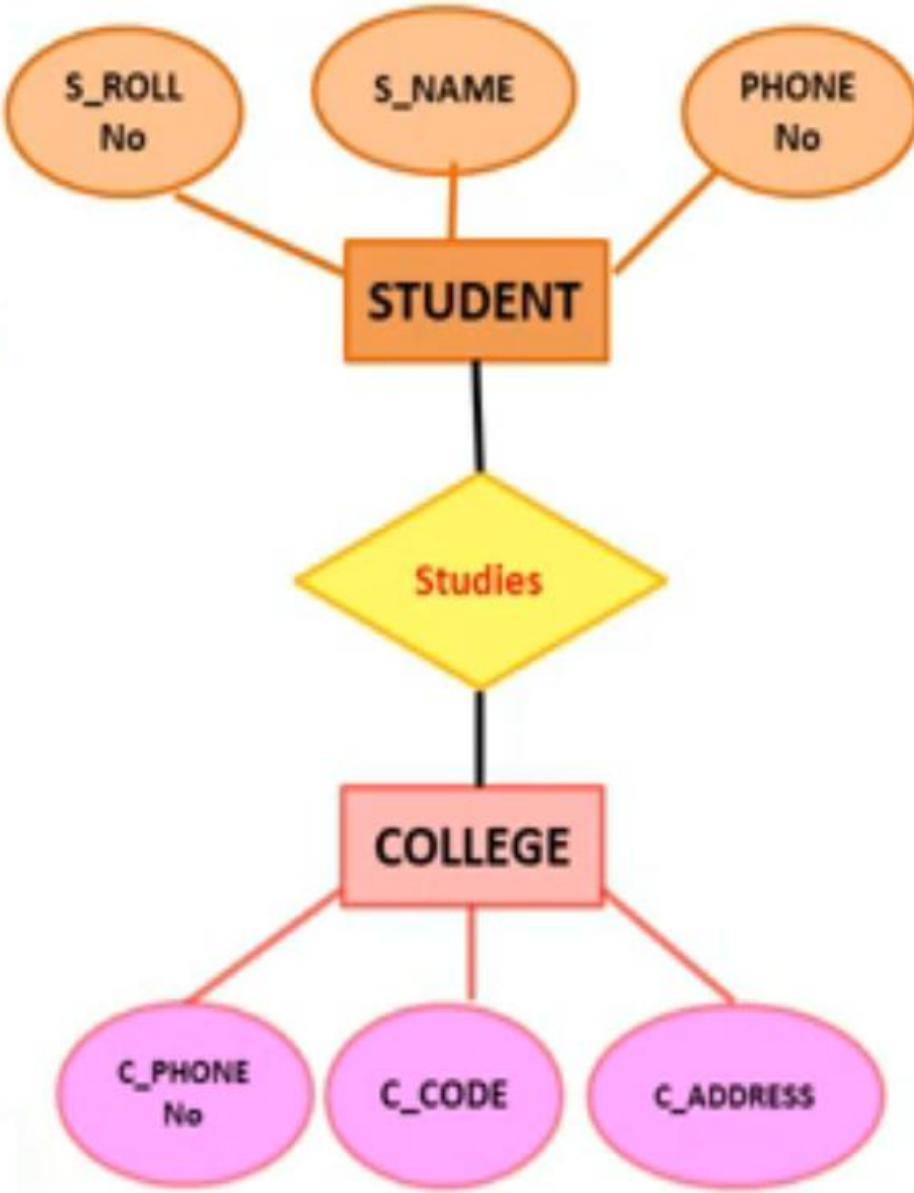
An entity is a REAL-WORLD THING/OBJECT which can be easily identified like a PERSON, PLACE OR A CONCEPT.

ATTRIBUTE:

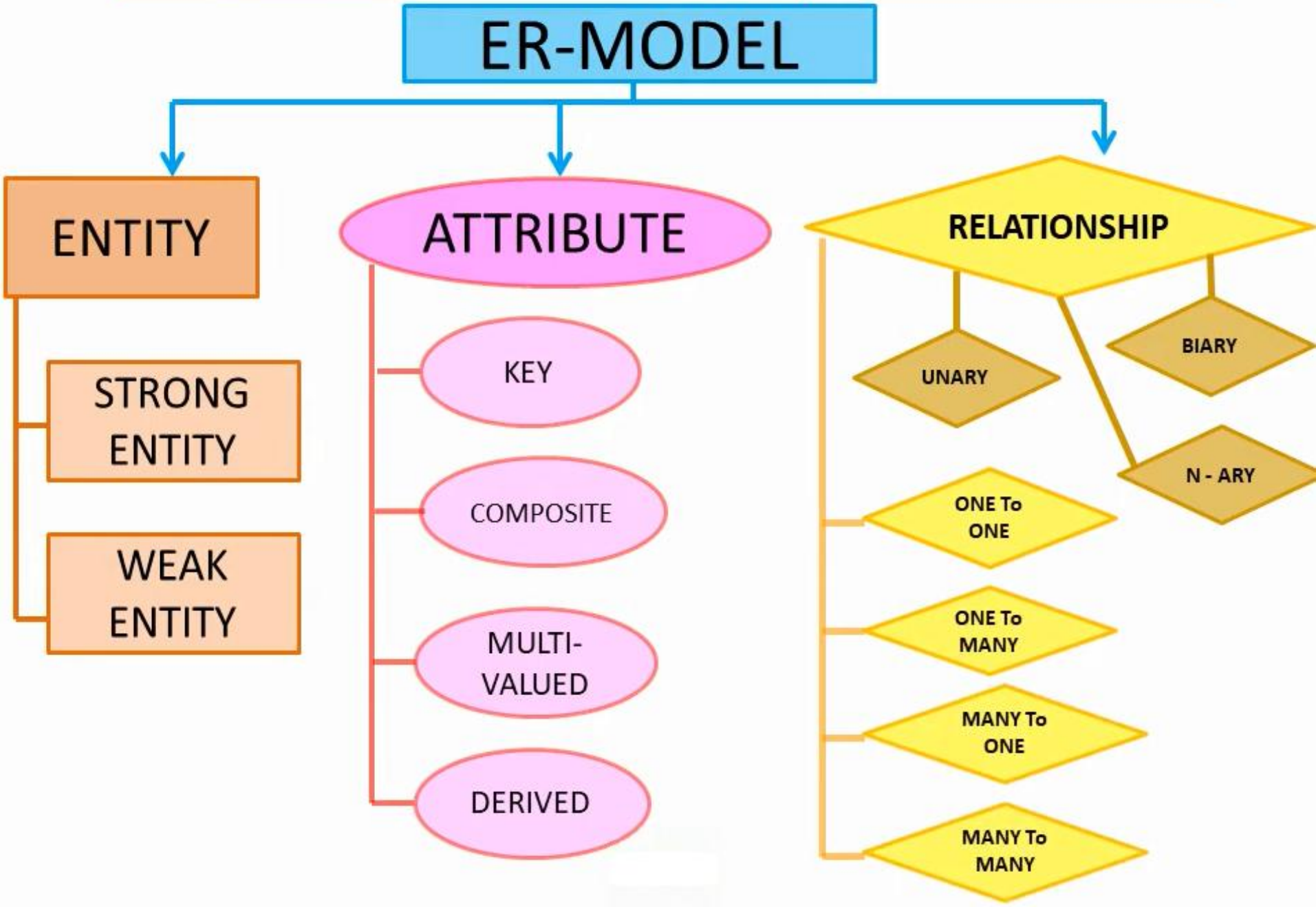
Attributes are the DESCRIPTIVE PROPERTIES which are OWNED BY EACH ENTITY of an Entity Set.

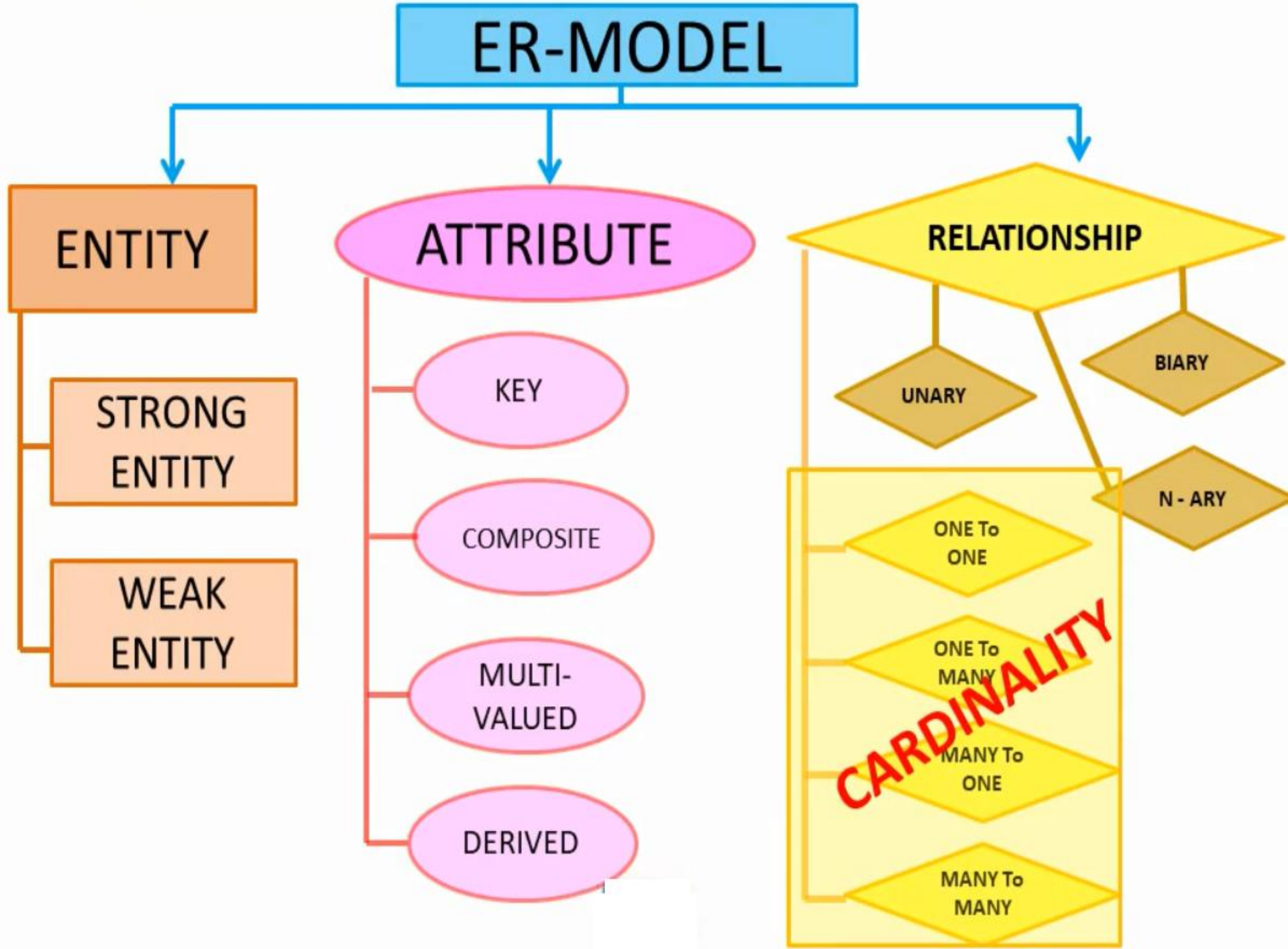
RELATIONSHIP:

The association among Entities



COMPONENTS OF A ER DIAGRAM




ER-MODEL
ENTITY
STRONG ENTITY
WEAK ENTITY
ATTRIBUTE
KEY
COMPOSITE
MULTI-VALUED
DERIVED
RELATIONSHIP
UNARY
BIARY
N - ARY
CARDINALITY
ONE To ONE
ONE To MANY
MANY To ONE
MANY To MANY

COMPONENTS OF A ER DIAGRAM → ENTITY

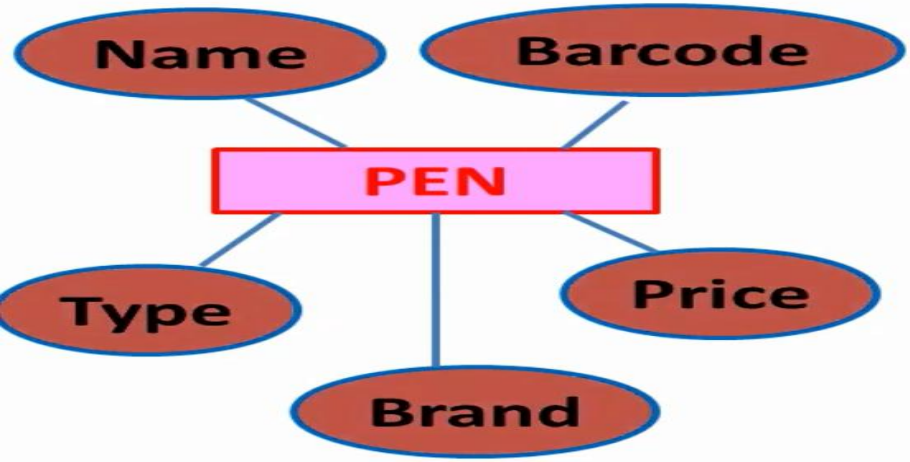
- An entity is a REAL-WORLD THING/OBJECT which can be easily identified like a PERSON, PLACE OR A CONCEPT.
- An entity can be of two types:
- **TANGIBLE ENTITY:**

EXIST in the real world PHYSICALLY. Example: Person, car, etc.

- **INTANGIBLE ENTITY:**

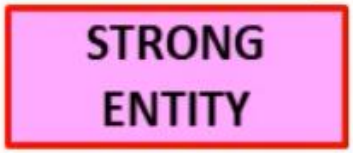
EXIST only LOGICALLY and have no physical existence. Example: Bank Account, etc.

COMPONENTS OF A ER DIAGRAM → ENTITY



COMPONENTS OF A ER DIAGRAM → ENTITY

STRONG ENTITY



- A strong entity will ALWAYS HAVE A PRIMARY KEY.
- A strong entity is NOT DEPENDENT of any other entity in the schema.
- Strong entities are REPRESENTED by a SINGLE RECTANGLE.
- The relationship of TWO STRONG ENTITIES is represented by a SINGLE DIAMOND.
- VARIOUS STRONG ENTITIES, when COMBINED together, create a STRONG ENTITY SET

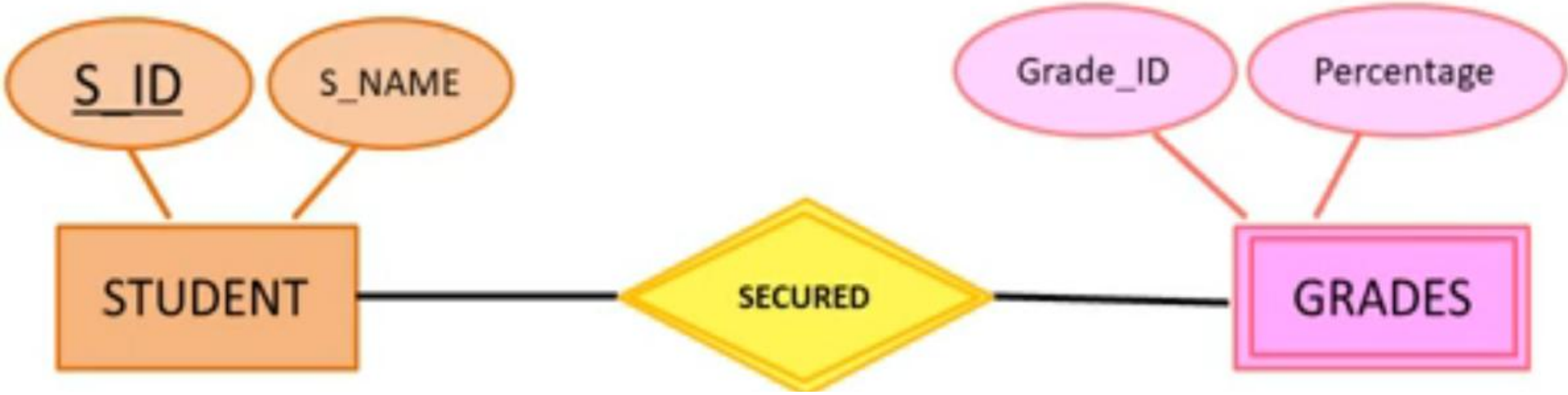
COMPONENTS OF A ER DIAGRAM → ENTITY

WEAK ENTITY

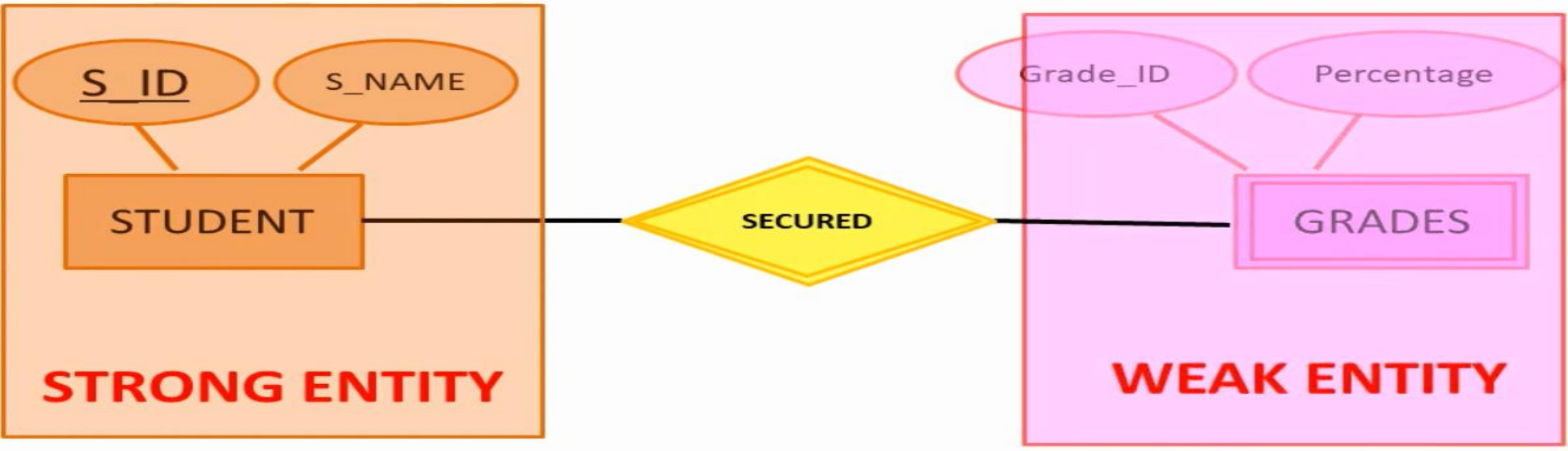
- A weak entity does NOT HAVE ANY PRIMARY KEY.
- A weak entity is DEPENDENT ON A STRONG ENTITY to ensure the its existence.
- A weak entity is REPRESENTED BY A DOUBLE RECTANGLE.
- The relation between ONE STRONG AND ONE WEAK entity is represented by a DOUBLE DIAMOND.

COMPONENTS OF A ER DIAGRAM → ENTITY

STRONG ENTITY – WEAK ENTITY - EXAMPLE



STRONG ENTITY – WEAK ENTITY - EXAMPLE



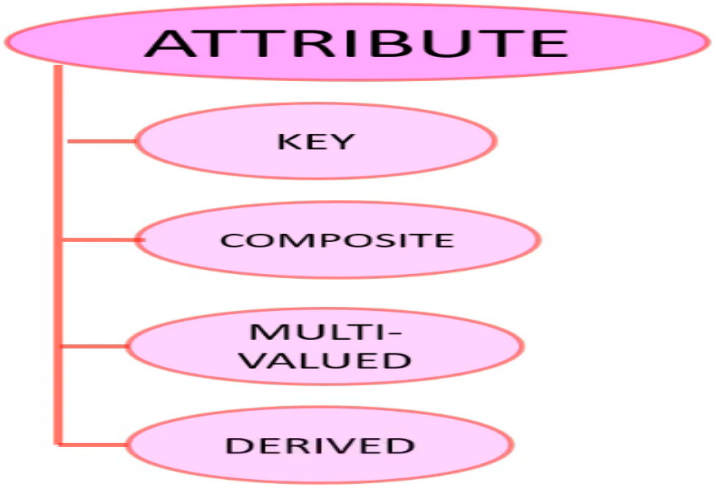
COMPONENTS OF A ER DIAGRAM → ATTRIBUTE

ATTRIBUTE

- Attributes are the DESCRIPTIVE PROPERTIES which are OWNED BY EACH ENTITY of an Entity Set.
- For example, Roll_No, Name, DOB, Age, Address, Mobile_No are the attributes which defines entity type Student.
- In ER diagram, ATTRIBUTE IS REPRESENTED by an OVAL.
- There are FOUR types of attributes, KEY ATTRIBUTE COMPOSITE ATTRIBUTE , MULTIVALUED ATTRIBUTE DERIVED ATTRIBUTE

COMPONENTS OF A ER DIAGRAM → ATTRIBUTE

ATTRIBUTE

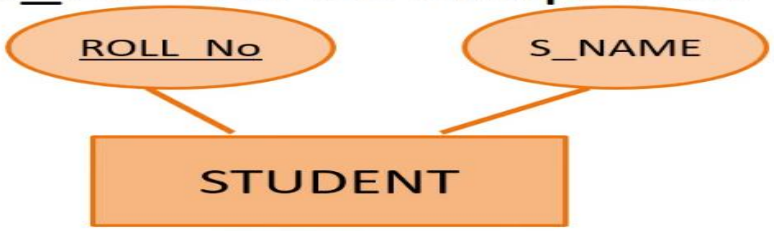


COMPONENTS OF A ER DIAGRAM → ATTRIBUTE

KEY ATTRIBUTE

KEY

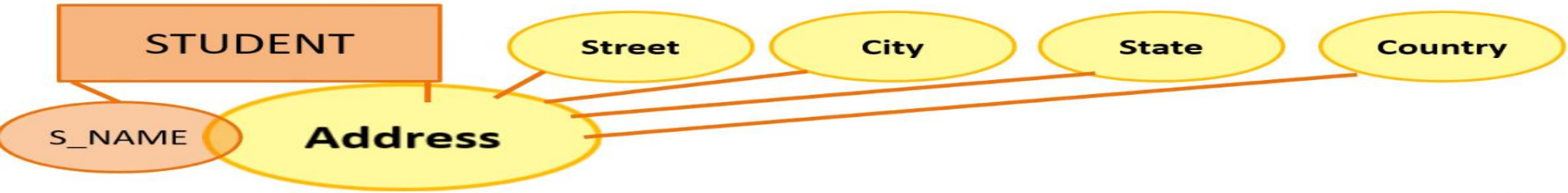
- The attribute which uniquely identifies (PRIMARY KEY) each entity in the entity set is called key attribute.
- In ER diagram, key attribute is represented by an oval with underlying lines.
- For example, Roll_No will be unique for each student.



COMPONENTS OF A ER DIAGRAM → ATTRIBUTE

COMPOSITE ATTRIBUTE

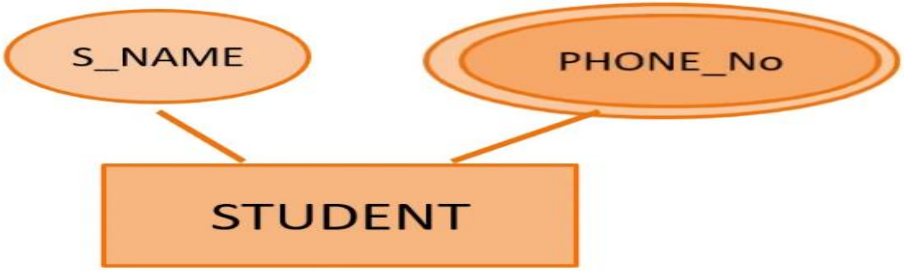
- An attribute COMPOSED OF MANY OTHER ATTRIBUTE is called as composite attribute.
- For example, Address attribute of student Entity type consists of Street, City, State, and Country.
- In ER diagram, COMPOSITE ATTRIBUTE is represented by an oval COMPRISING OF OVALS.



COMPONENTS OF A ER DIAGRAM → ATTRIBUTE

MULTIVALUED ATTRIBUTE

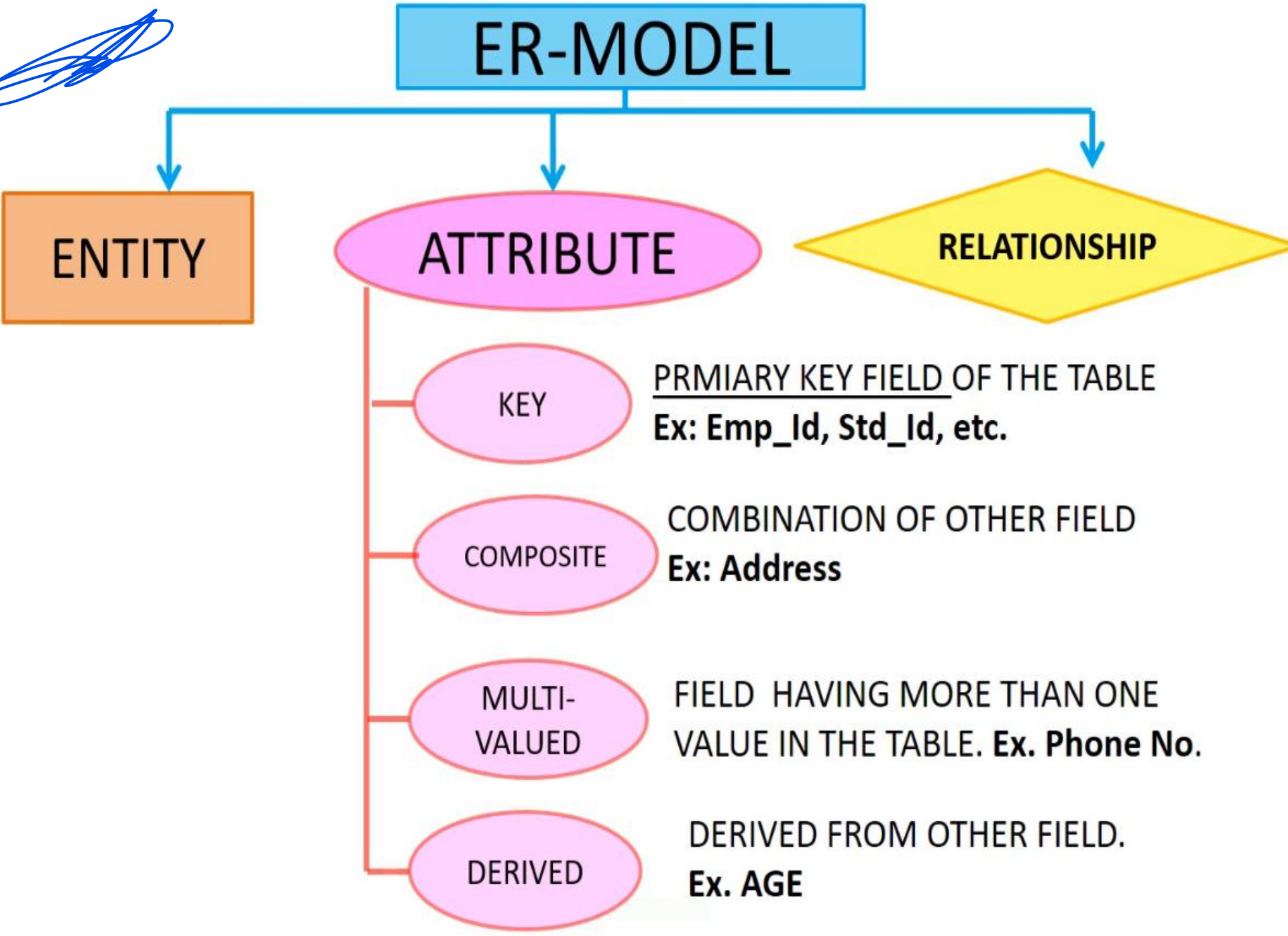
- An attribute consisting MORE THAN ONE VALUE for a given entity.
- For example, Phone_No (can be more than one for a given student).
- In ER diagram, multivalued attribute is REPRESENTED BY DOUBLE OVAL



COMPONENTS OF A ER DIAGRAM → ATTRIBUTE

DERIVED ATTRIBUTE

- An attribute which can be DERIVED FROM OTHER ATTRIBUTES of the entity type is known as derived attribute.
- Example: Age (can be derived from DOB).
- In ER diagram, derived attribute is REPRESENTED BY DASHED OVAL..


ER-MODEL
ENTITY
ATTRIBUTE
RELATIONSHIP
KEY
PRMIARY KEY FIELD OF THE TABLE
Ex: Emp_Id, Std_Id, etc.
COMPOSITE
COMBINATION OF OTHER FIELD
Ex: Address
MULTI-VALUED
FIELD HAVING MORE THAN ONE VALUE IN THE TABLE. Ex. Phone No.
DERIVED
DERIVED FROM OTHER FIELD.
Ex. AGE

COMPONENTS OF A ER DIAGRAM → RELATIONSHIP

The association among entities is called a relationship.

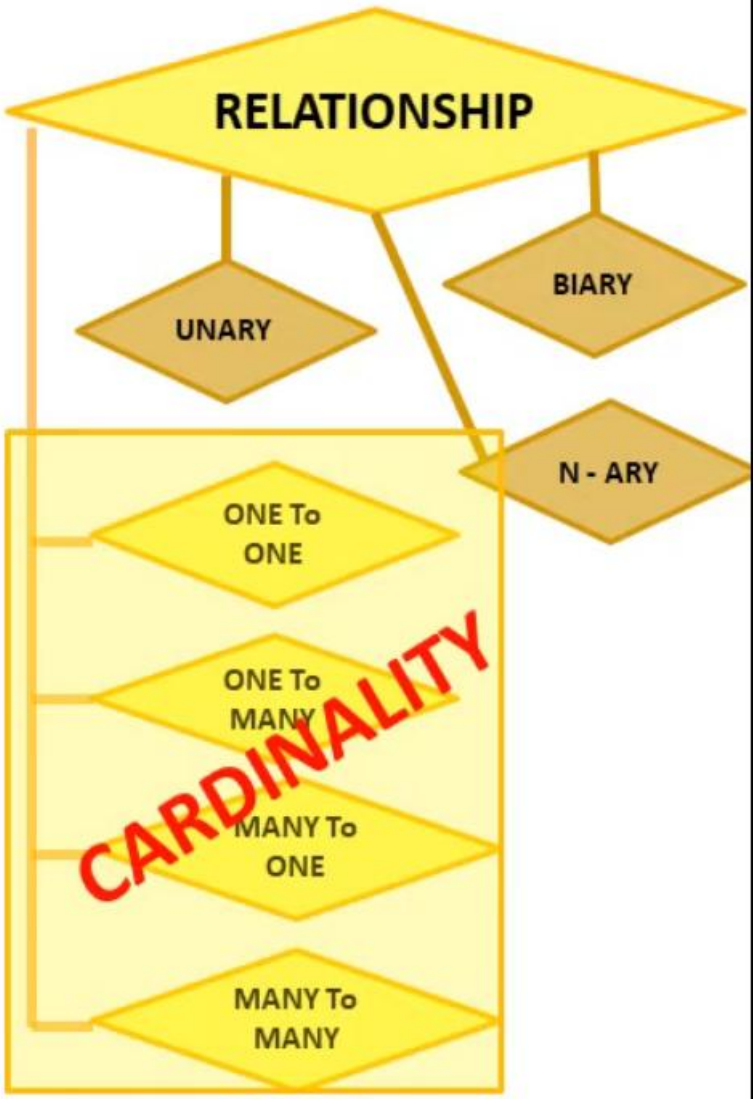
Relationship Set : A set of relationships of similar type is called a relationship set.

Degree of Relationship

- The number of participating entities in a relationship defines the degree of the relationship.
  - Unary = degree 1
  - Binary = degree 2
  - n-ary = degree

Mapping Cardinalities

- Cardinality defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.

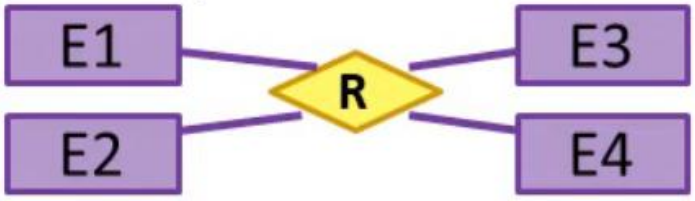


COMPONENTS OF A ER DIAGRAM → RELATIONSHIP

UNARY RELATIONSHIP : When there is only ONE entity set participating in a relation, the relationship is called as unary relationship. For example, one person is married to only one person.

BINARY RELATIONSHIP: When there are TWO entities set participating in a relation, the relationship is called as binary relationship. For example, Student is enrolled in Course.

N-ARY RELATIONSHIP: When there are n entities set participating in a relation, the relationship is called as n-ary relationship.



COMPONENTS OF A ER DIAGRAM → RELATIONSHIP

Cardinality ONE-TO-ONE

- **One to one** – When each entity in each entity set can take part only once in the relationship, the cardinality is one to one.
- Let us assume a female can marry to one male. So the relationship will be one to one.

COMPONENTS OF A ER DIAGRAM → RELATIONSHIP

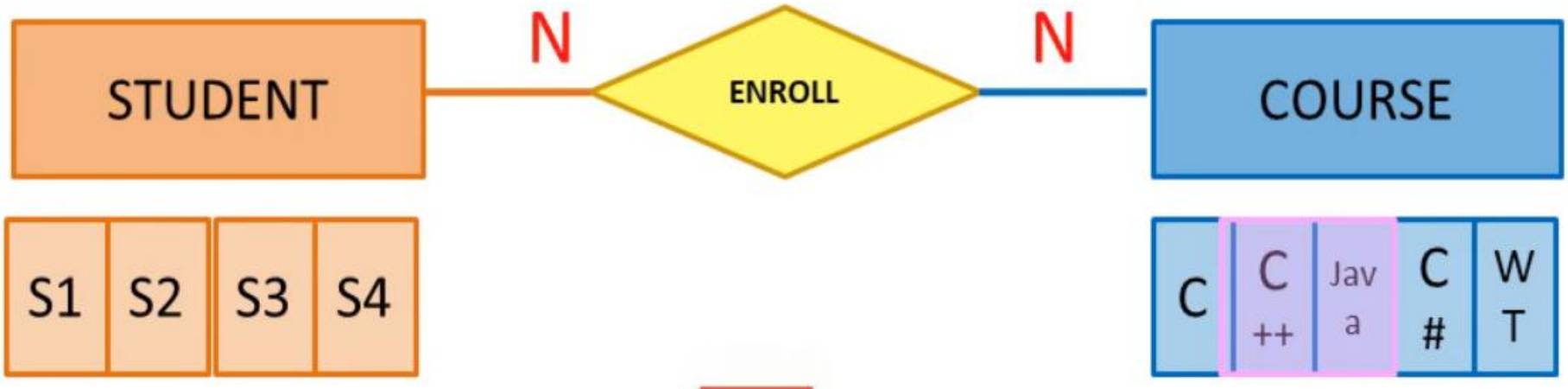
Cardinality MANY-TO-ONE

Many to one – When entities in one entity set can take part only once in the relationship set and entities in other entity set can take part more than once in the relationship set, cardinality is many to one.

COMPONENTS OF A ER DIAGRAM → RELATIONSHIP

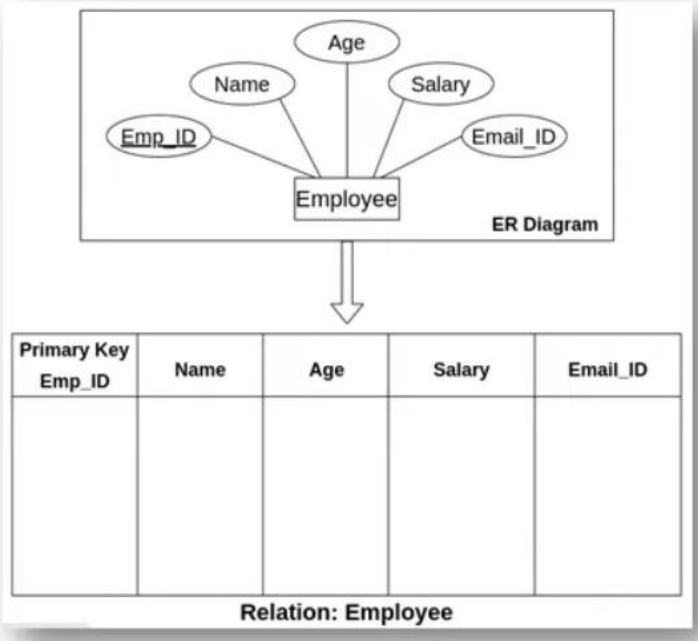
Cardinality MANY-TO-MANY

- **Many to many** – When entities in all entity sets can take part more than once in the relationship cardinality is called as many to many.



RULES OF CONVERTING ER DIAGRAM INTO THE TABLE

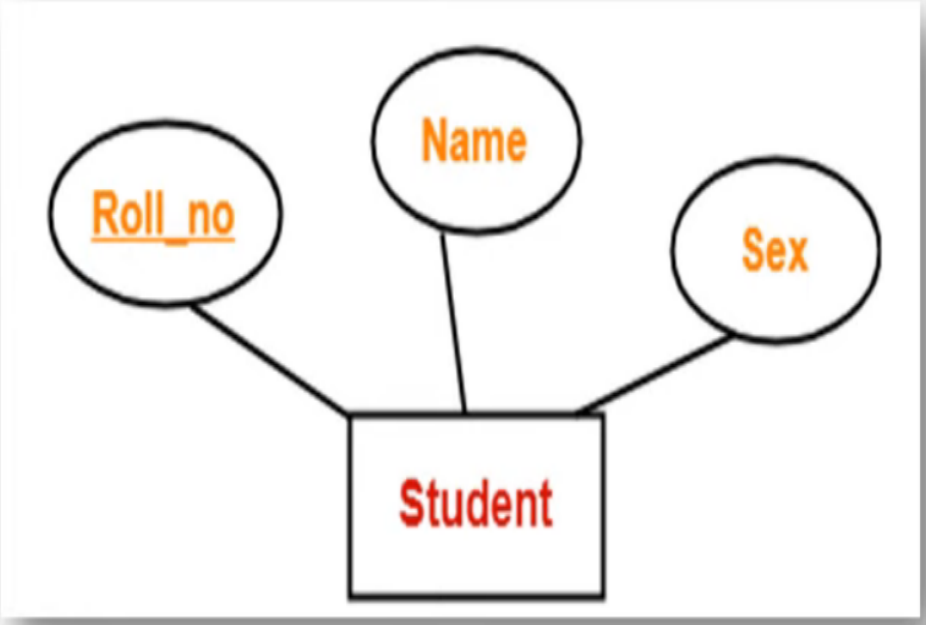
1. 5 Rules of Converting ER Diagrams into the Table
2. Example of Converting ER Diagrams into the Table



Rules of Converting ER Diagram into Table

➢ Rule 1: Strong Entity set with Simple attributes

- Attributes of the table will be the attributes of the entity set.
- The primary key of the table will be the key attribute of the entity set.



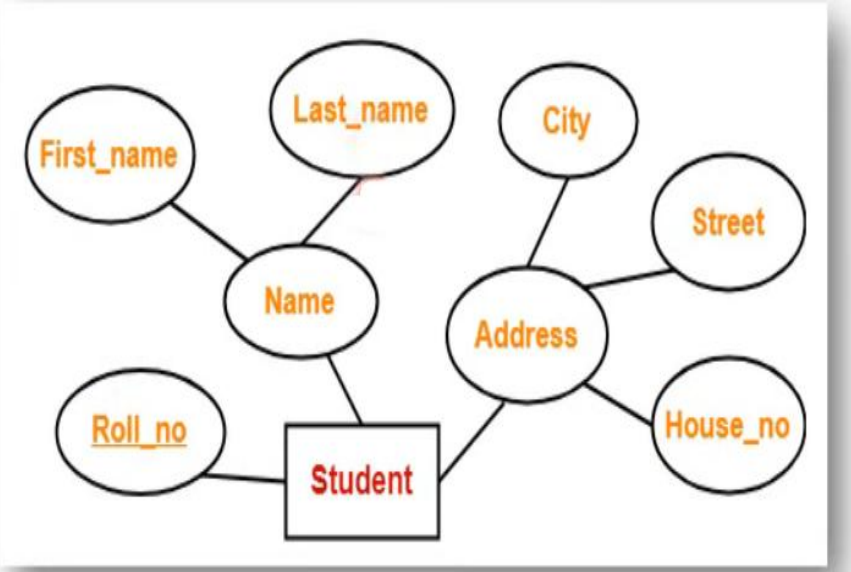
Roll_no	Name	Sex

Schema : Student (Roll_no , Name , Sex)

Rules of Converting ER Diagram into Table

➢ Rule 2: For Strong Entity Set With Composite Attributes

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



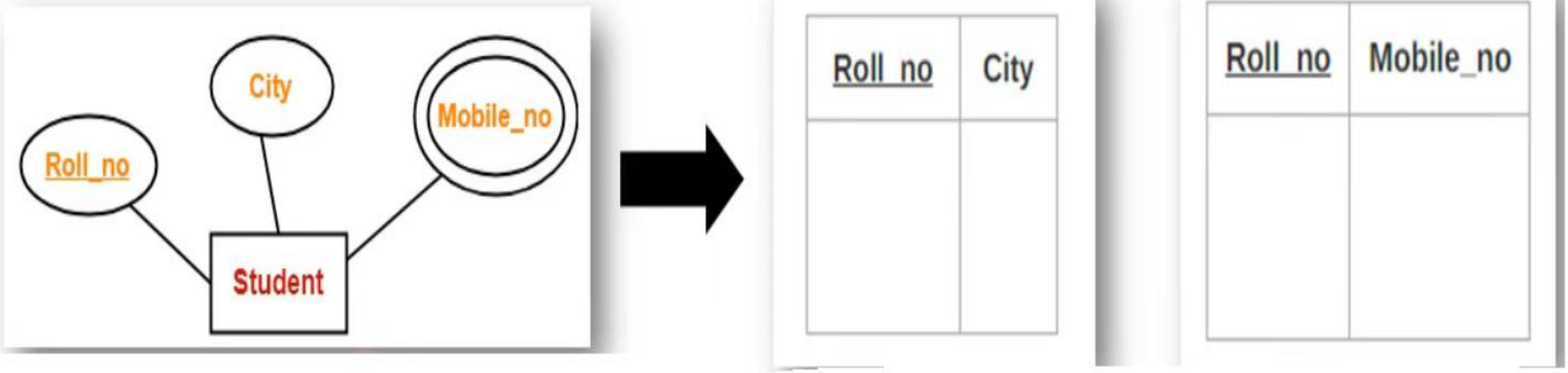
Roll_no	First_name	Last_name	House_no	Street	City

Schema : Student (Roll_no , First_name , Last_name , House_no , Street , City)

Rules of Converting ER Diagram into Table

➢Rule 3: For Strong Entity Set With Multi Valued Attributes

- A strong entity set with any number of multi valued attributes will require two 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 multi valued attributes.

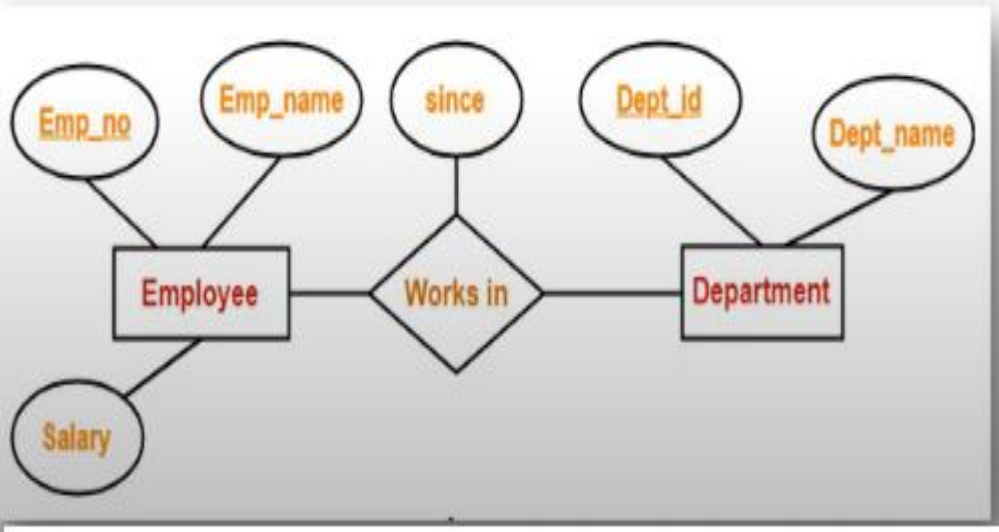


Rules of Converting ER Diagram into Table

➢Rule 4: Translating Relationship Set into a Table

Attributes of the table are-

- Primary key attributes of the participating entity sets
- Its own descriptive attributes if any.
- Set of non-descriptive attributes will be the primary key.



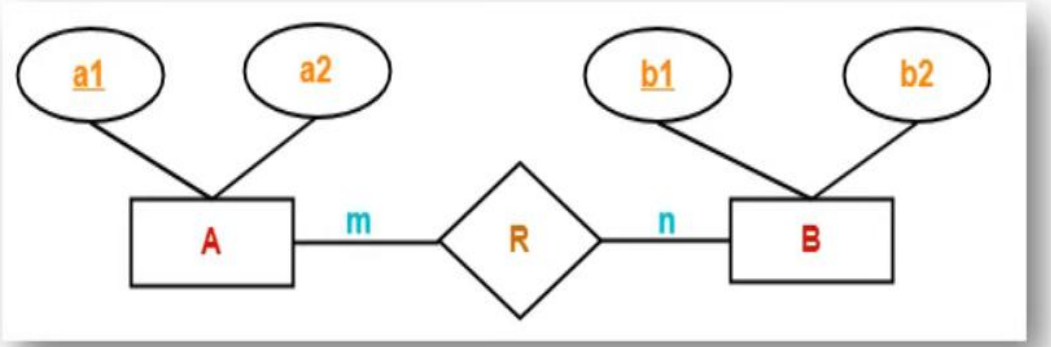
Emp_no	Dept_id	since

Schema : Works in (Emp_no , Dept_id , since)

Rules of Converting ER Diagram into Table

➢ Rule 5: For Binary Relationships With Cardinality Ratios

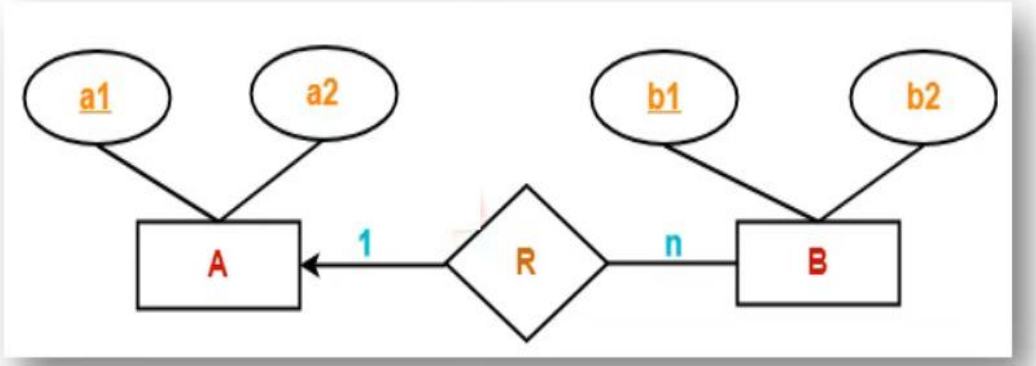
✓ Case-1: For Binary Relationship With Cardinality Ratio m:n



Here, three tables will be required-

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

✓ Case-2: For Binary Relationship With Cardinality Ratio 1:n



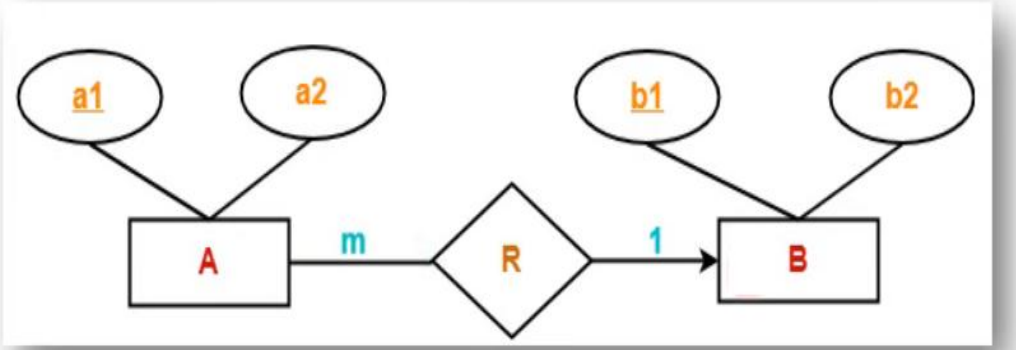
Here, two tables will be required-

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

Rules of Converting ER Diagram into Table

➢ Rule 5: For Binary Relationships With Cardinality Ratios

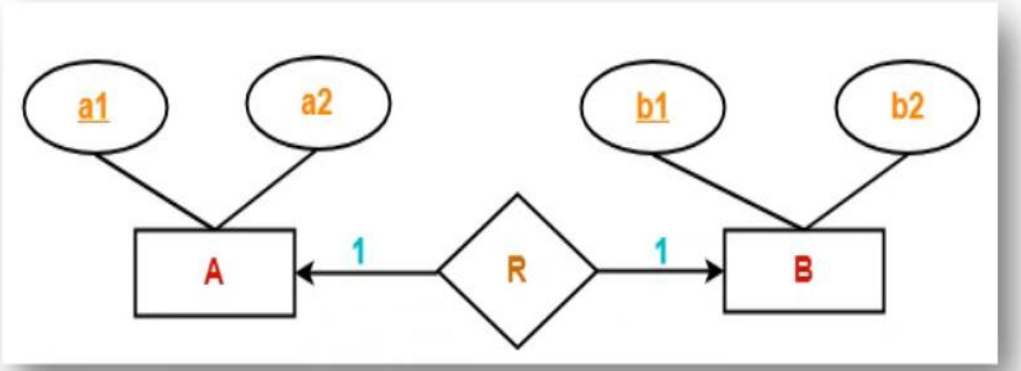
✓ Case-3: For Binary Relationship With Cardinality Ratio m:1



Here, two tables will be required-

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

✓ Case-4: For Binary Relationship With Cardinality Ratio 1:1



Way-01:

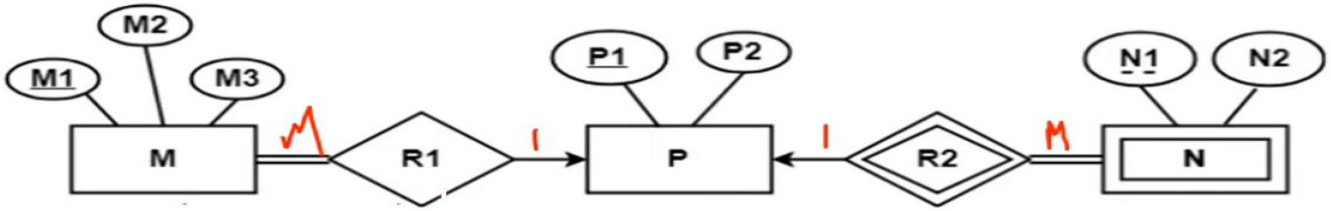
1. AR (a1 , a2 , b1)
2. B (b1 , b2)

Way-02:

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

Question 1:

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



Solution-

Minimum 3 tables will be required-

1. MR1 (M1 , M2 , M3 , P1)
2. P (P1 , P2)
3. NR2 (P1 , N1 , N2)

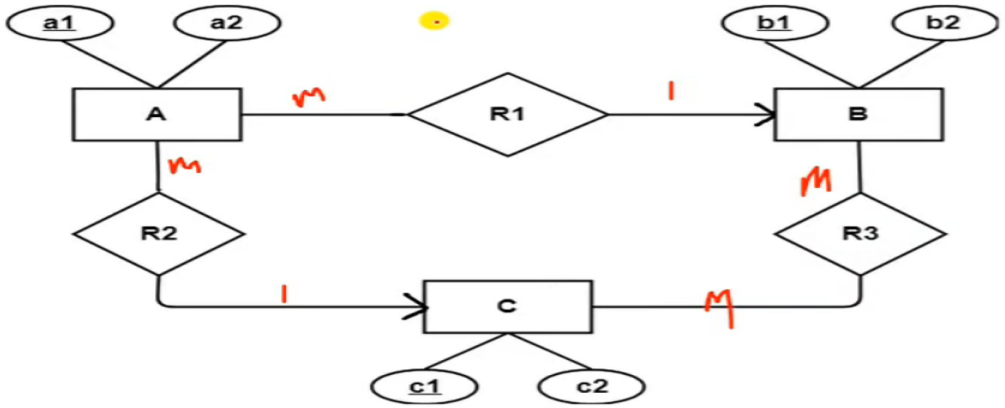
Question 2:

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

Solution-

Minimum 4 tables will be required:

1. AR1R2 (a1 , a2 , b1 , c1)
2. B (b1 , b2)
3. C (c1 , c2)
4. R3 (b1 , c1)



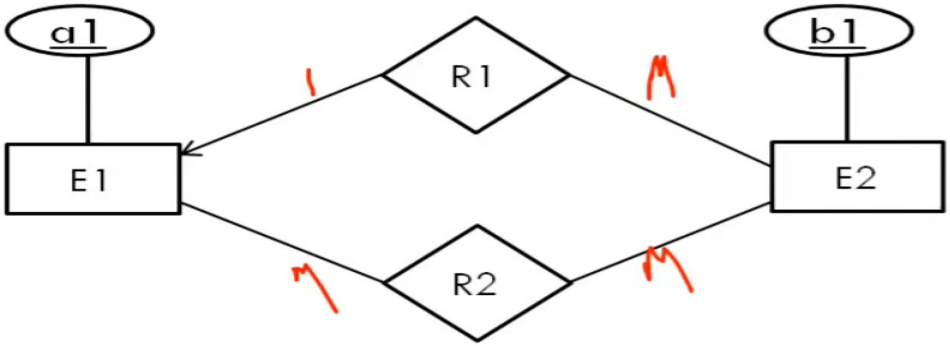
Question 3:

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

- **Solution**:

  Three tables will be formed

  1. E1(a1)
  2. E2R1 (b1, a1)
  3. R2 (a1, b1)



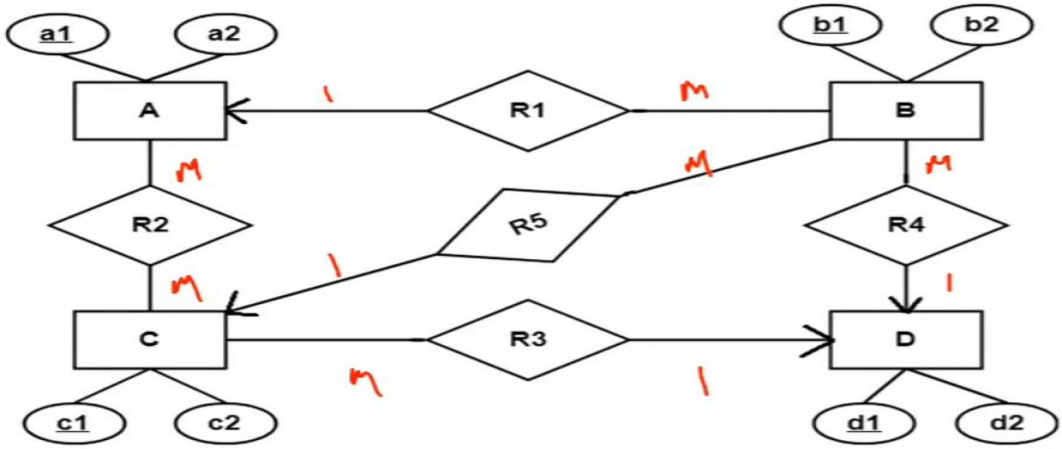
Question 4:

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

Solution-

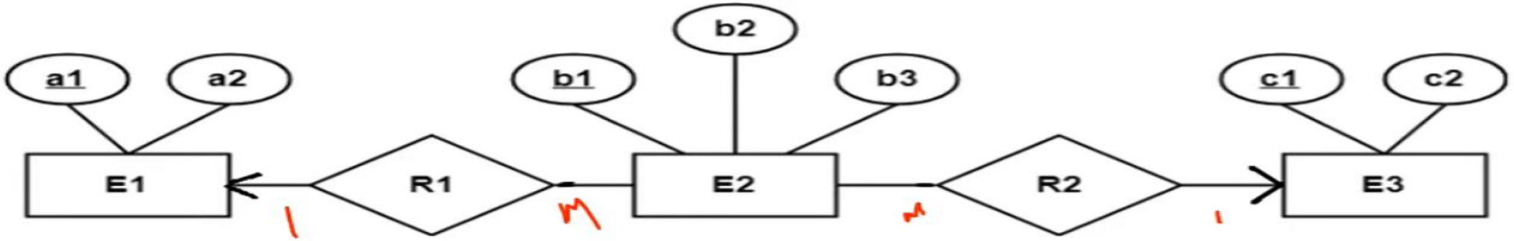
Minimum 5 tables will be required:

1. BR1R4R5 (b1 , b2 , a1 , c1 , d1)
2. A (a1 , a2)
3. R2 (a1 , c1)
4. CR3 (c1 , c2 , d1)
5. D (d1 , d2)



Question 5:

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



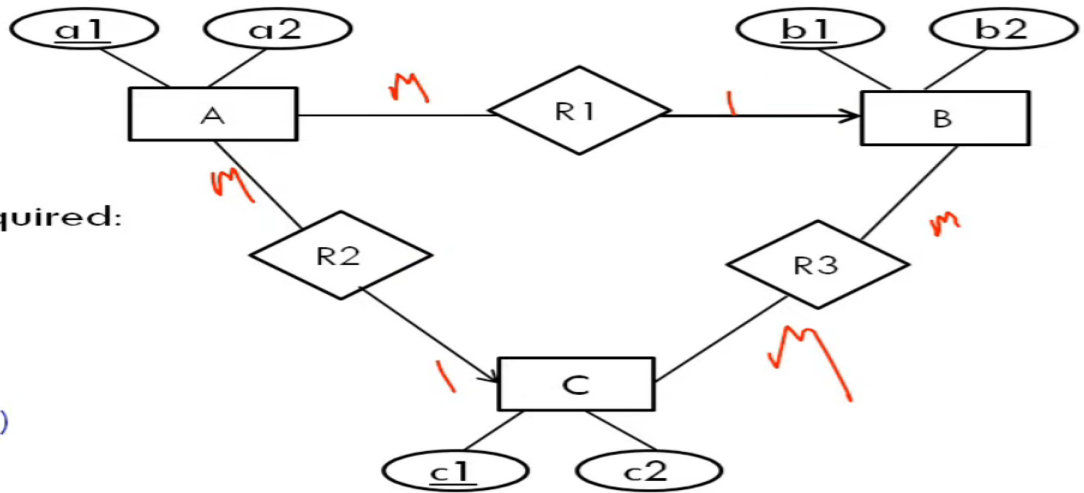
Solution-

Minimum 3 tables will be required:

1. E1 (a1 , a2)
2. E2R1R2 (b1 , b2 , b3, a1 , c1)
3. E3 (c1 , c2)

Question 6:

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

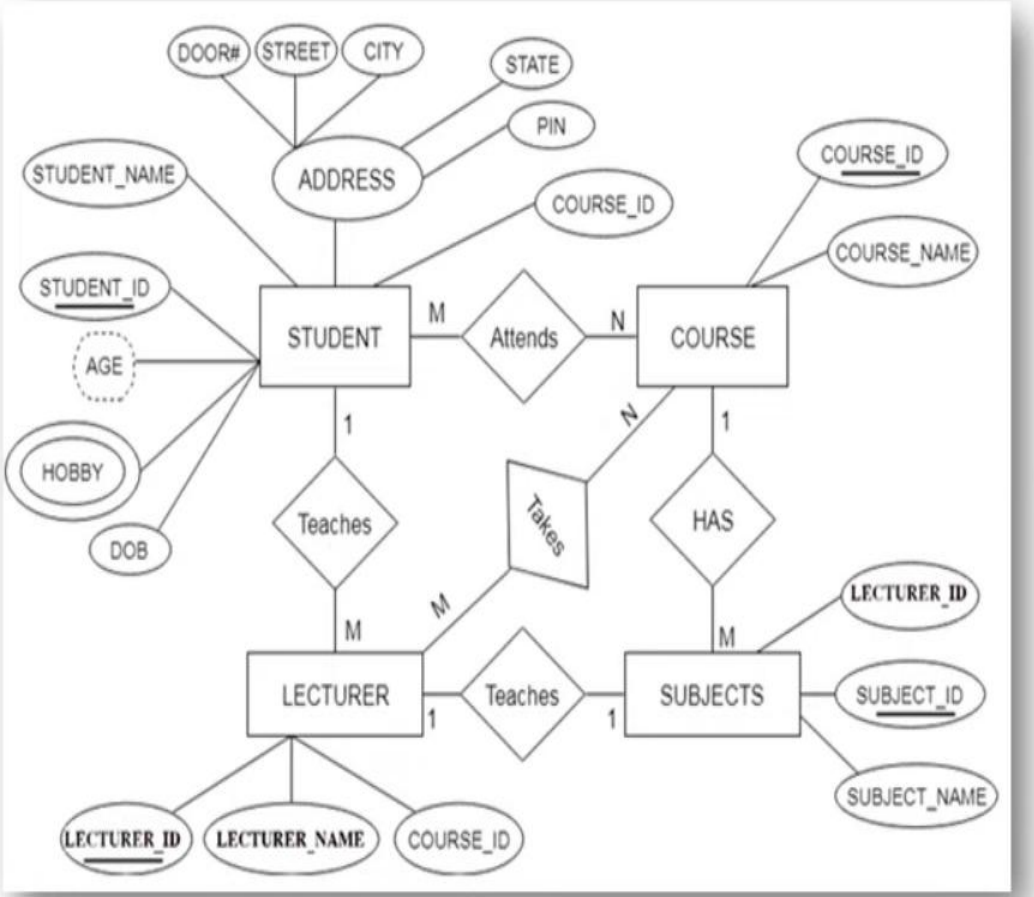


Solution-

Minimum 4 tables will be required:

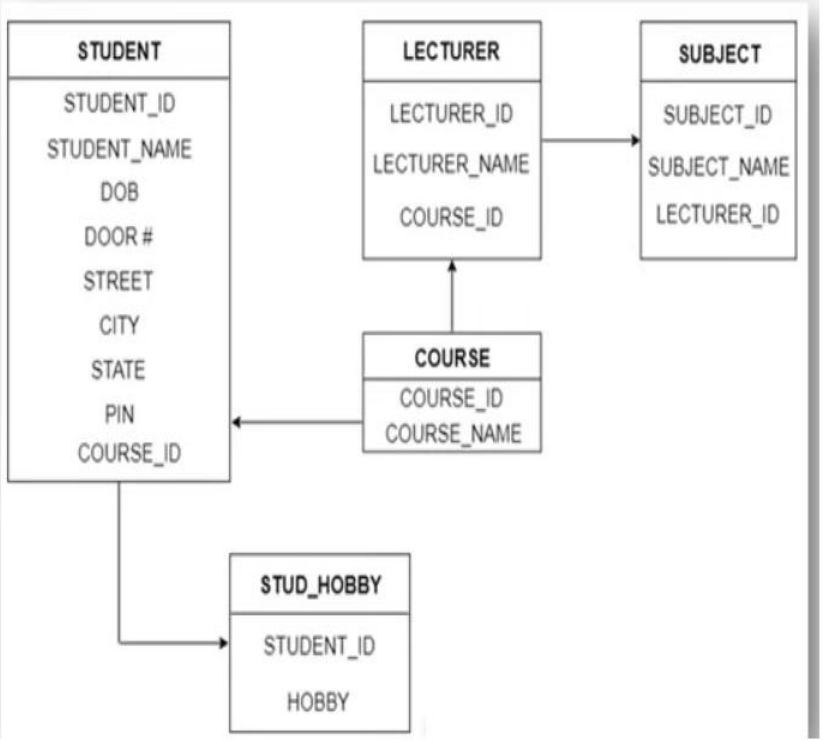
1. B (b1 , b2)
2. C (c1 , c2)
3. R3(b1, c1)
4. AR1R2 (a1 , a2, b1, c1)

Example of Converting ER Diagram into Table

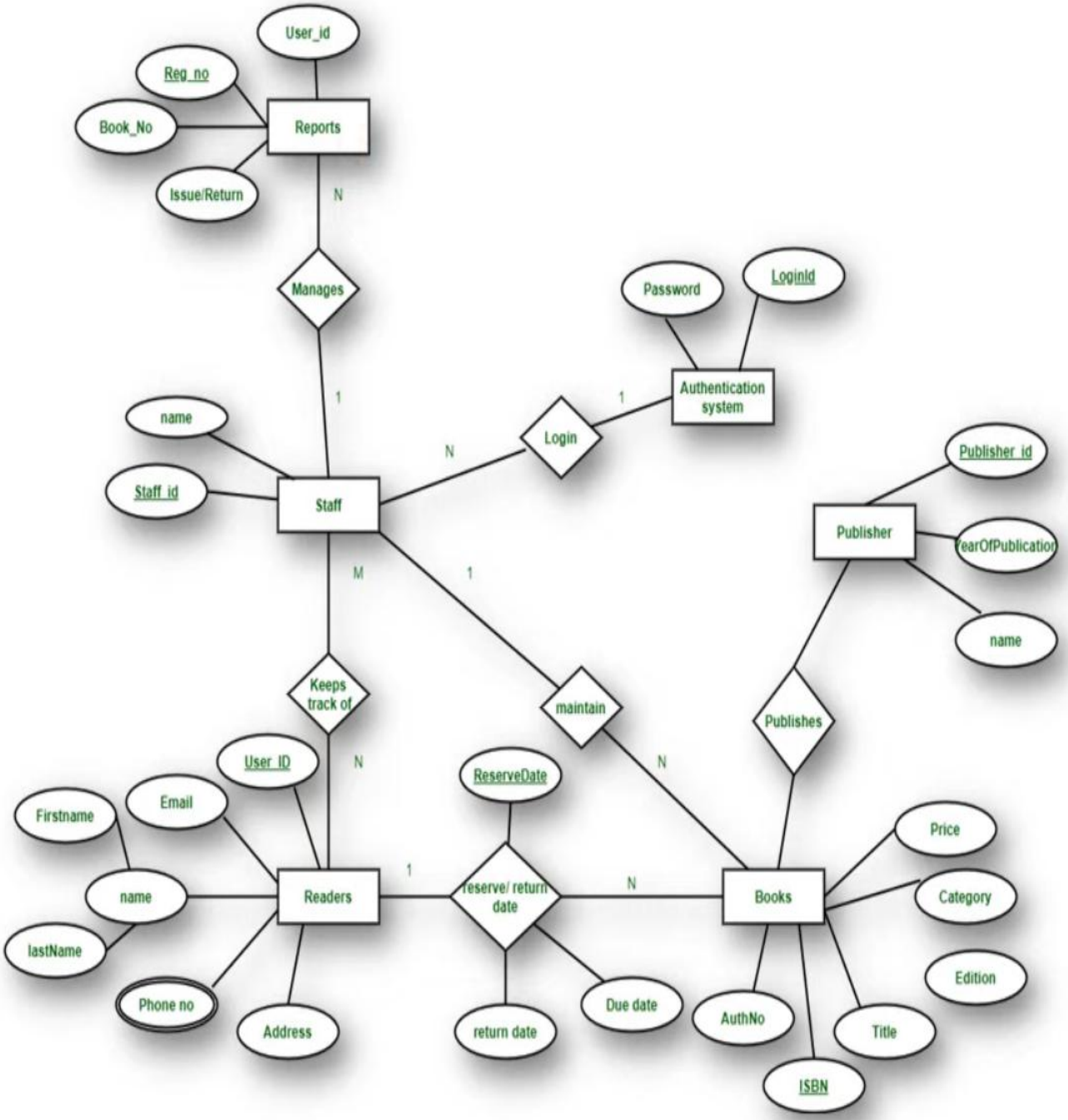


ER Diagram

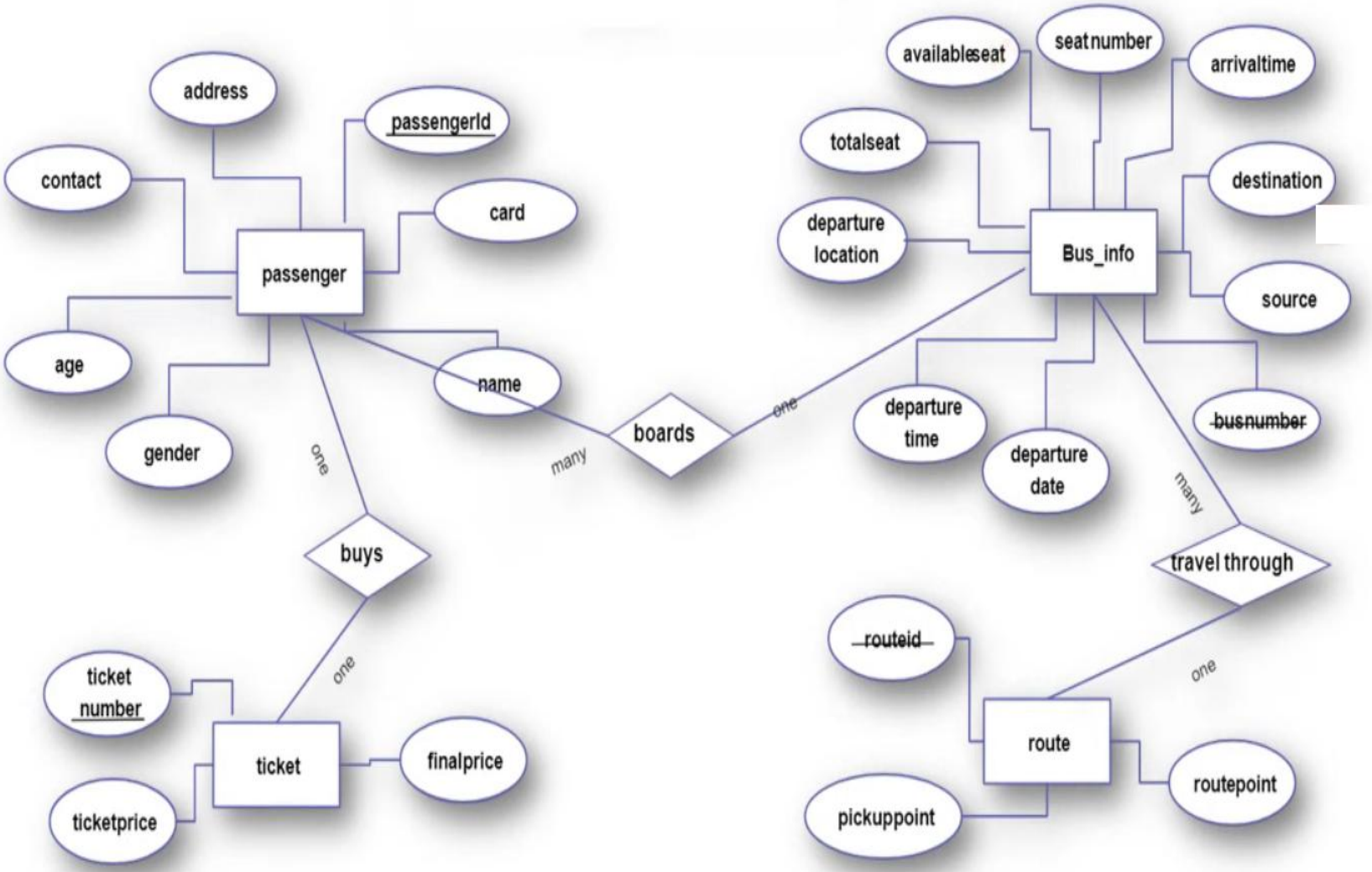
Table



ER Diagram of Library Management System



ER Diagram of Bus Ticketing System



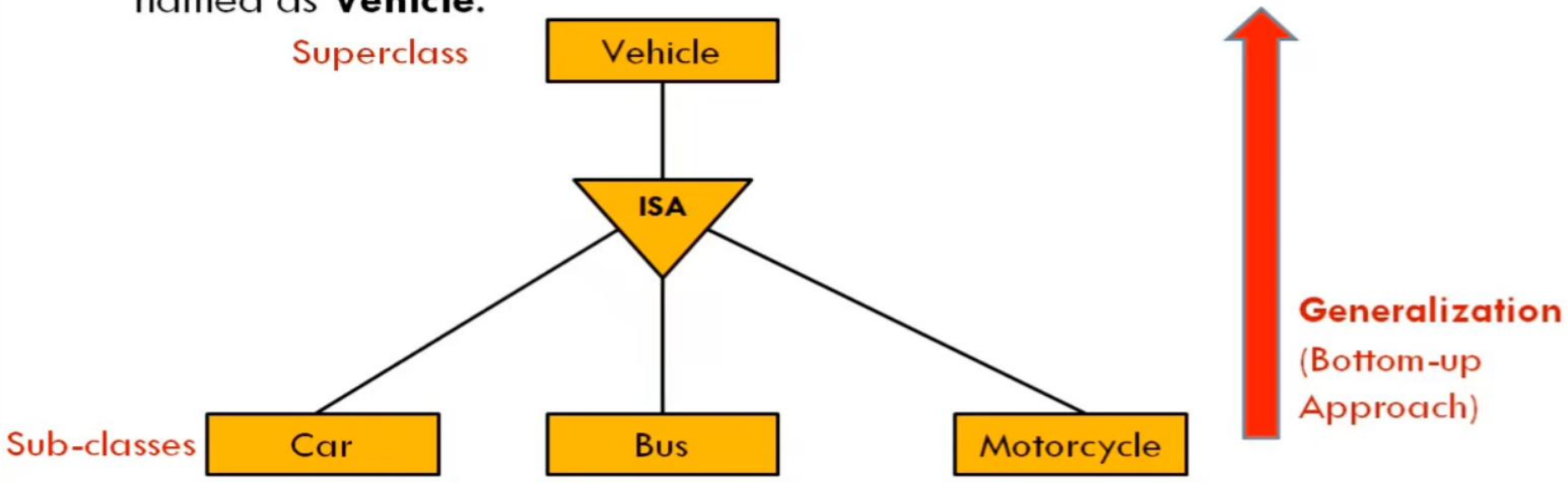
Extended E-R Features

- As the complexity of data increased in the late 1980s, it became more and more difficult to use the traditional ER Model for database modelling. Hence some improvements or enhancements were made to the existing ER Model to make it able to handle the complex applications better.
- Hence, as part of the **Extended ER Model**, along with other improvements, three new concepts were added to the existing ER Model:

1. Generalization
2. Specialization
3. Aggregation

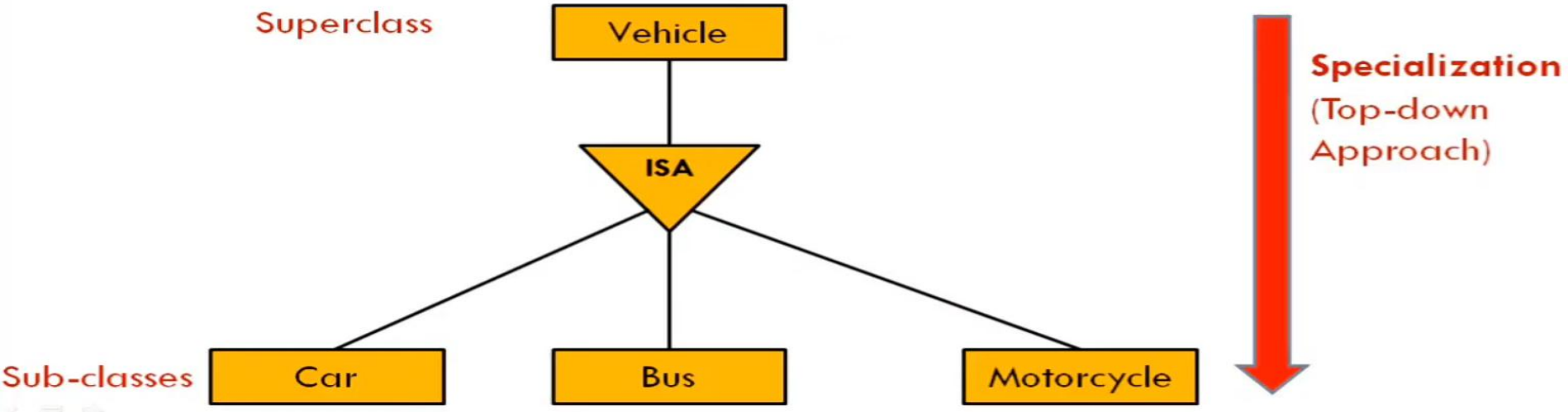
Generalization

- **Generalization** is the process of extracting common properties from a set of entities and create a generalized entity from it.
- **Generalization is a "bottom-up approach"** in which two or more entities can be combined to form a higher level entity if they have some attributes in common.
  - subclasses are combined to make a superclass.
- **Generalization is used** to emphasize the similarities among lower-level entity set and to hide differences in the schema
  - Consider we have 3 sub entities Car, Bus and Motorcycle. Now these three entities can be generalized into one higher-level entity (or super class) named as **Vehicle.**



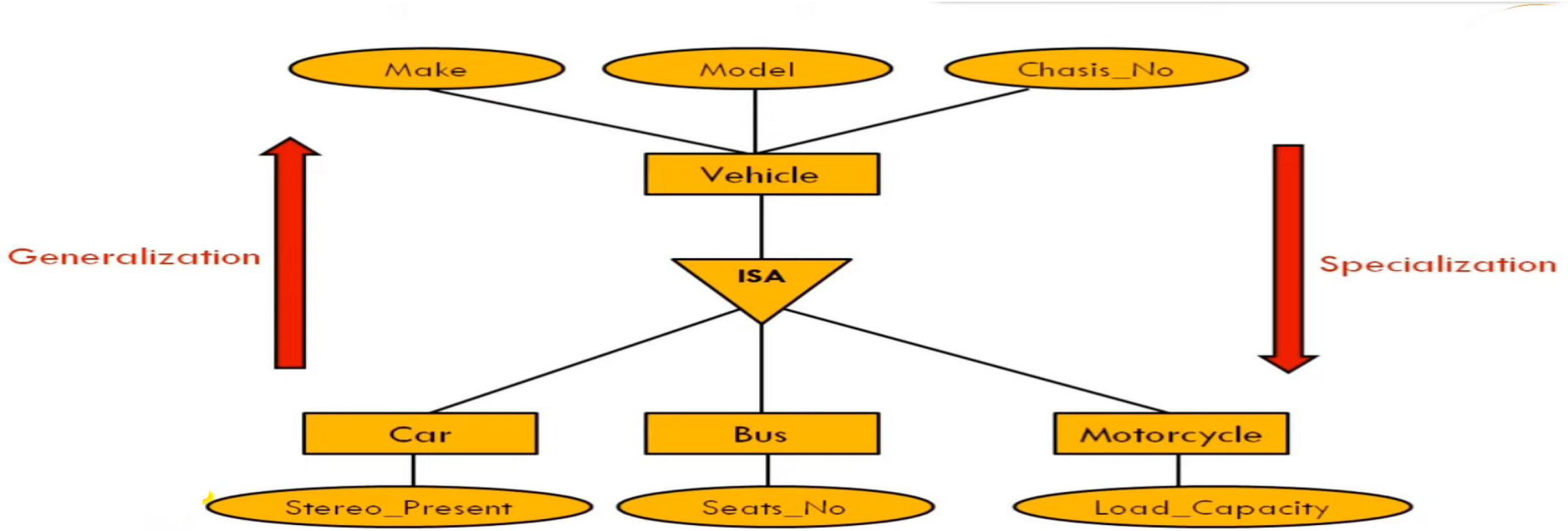
Specialization

- Specialization is **opposite** of Generalization
- In **Specialization**, **an entity is broken down into sub-entities based on their characteristics.**
- **Specialization is a "Top-down approach"** where higher level entity is specialized into two or more lower level entities.
- **Specialization is used** to identify the subset of an entity set that shares some distinguishing characteristics.
- **Specialization** can be repeatedly applied to refine the design of schema
- depicted by triangle component labeled **ISA**
  - **Vehicle** entity can be a Car, Truck or Motorcycle.
    - Normally, the superclass is defined first, the subclass and its related attributes are defined next, and relationship set are then added.

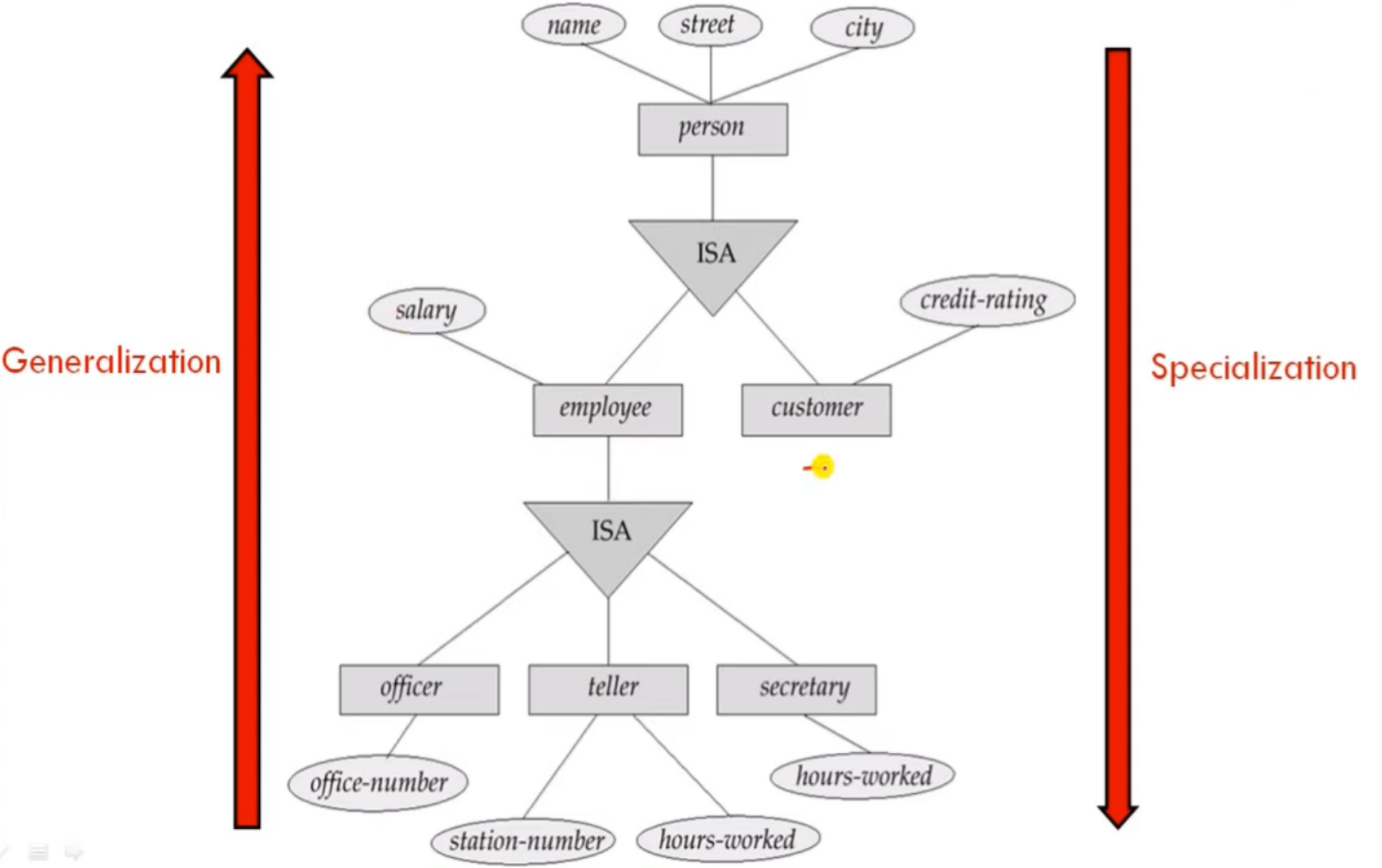


Inheritance

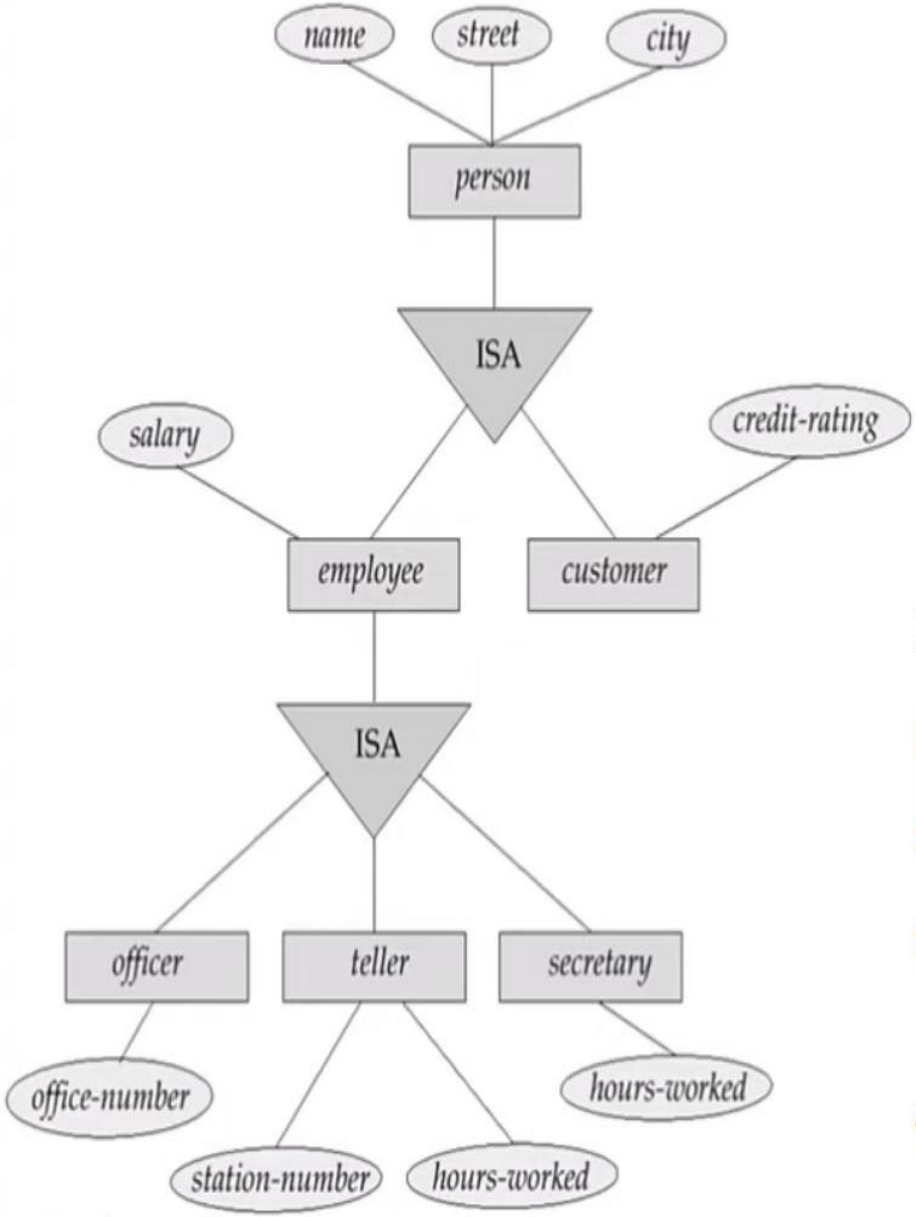
- **Inheritance** is an important feature of generalization and specialization.
- **Attribute inheritance allows lower level entities to *inherit the attributes* of higher level entities.**
  - **For example,** Consider relations Car and Bus inheriting the attributes of Vehicle. Thus, Car is described by attributes of super-class Vehicle as well as its own attributes.
- This also extends to **Participation Inheritance** in which ***relationships involving higher-level entity-sets are also inherited by lower-level entity-sets.***
  - A lower-level entity-set can participate in its own relationship-sets, too



Example


Generalization
Specialization
name
street
city
person
ISA
salary
credit-rating
employee
customer
ISA
officer
teller
secretary
office-number
station-number
hours-worked
hours-worked

How Schema or Tables can be formed?

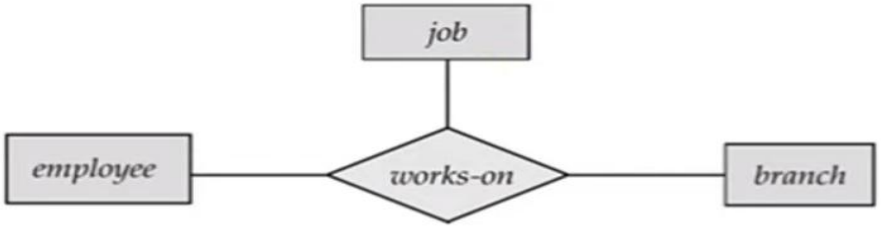


Four tables can be formed:

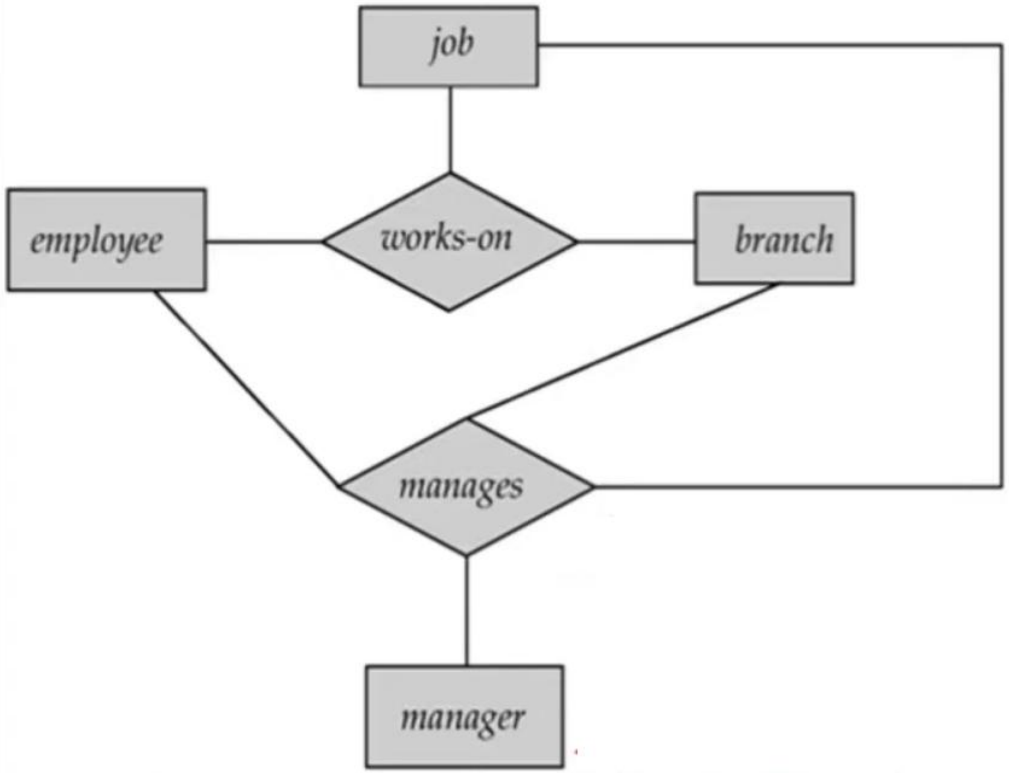
1. **customer** (name, street, city, credit_rating)
2. **officer** (name, street, city, salary, office_number)
3. **teller** (name, street, city, salary, station_number, hours_worked)
4. **secretary** (name, street, city, salary, hours_worked)

Aggregation

- **Aggregation** is used when we need to express a **relationship among relationships**
- **Aggregation** is an **abstraction** through which **relationships are treated as higher level entities**
- **Aggregation** is a process when a **relationship between two entities is considered as a single entity** and again this single entity has a **relationship** with **another entity**
- Basic E-R model can't represent relationships involving other relationships
- Consider a ternary relationship **works_on** between **Employee, Branch and Job.**
  - Am **employee** ***works on*** a particular **job** at a particular **branch**



- Suppose we want to assign a **manager** for jobs performed by an employee at a branch (i.e. want to assign managers to each employee, job, branch combination)
  - Need a separate manager entity-set
  - Relationship between each manager, employee, branch, and job entity

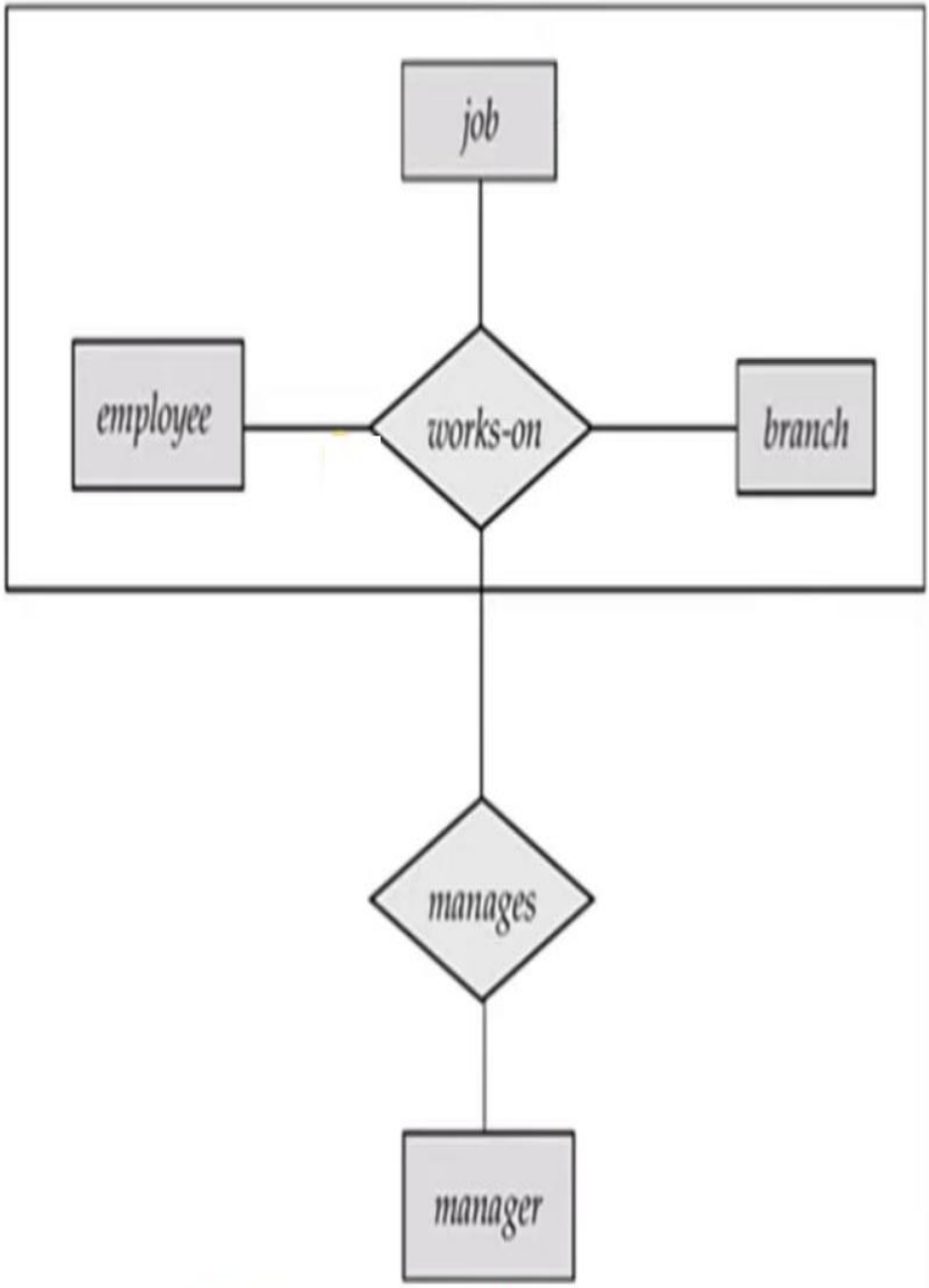


ER Diagram with Redundant Relationship

- Relationship sets ***works-on*** and ***manages*** represent overlapping (redundant) information
  - Every *manages* relationship corresponds to a *works-on* relationship
  - However, some *works-on* relationships may not correspond to any *manages* relationships
    - So we can't discard the *works-on* relationship

- Eliminate this redundancy via ***aggregation***
  - Treat relationship as an abstract entity
  - Allows relationships between relationships
  - Abstraction of relationship into new entity

- With **Aggregation** (without introducing redundancy) the ER diagram can be represented as:
  - An employee works on a particular job at a particular branch
  - An employee, branch, job combination may have an associated manager



ER Diagram with Aggregation

ER Diagram Issues

✓The notions of an entity set and a relationship set are not precise.

✓It is possible to define a set of entities and the relationships among them in a number of different ways.

✓The followings are the basic issues in ER Diagram

- Use of Entity Sets versus Attributes
- Use of Entity Sets versus Relationship Sets
- Binary versus n-ary Relationship Sets
- Placement of Relationship Attributes

Use of Entity Set Vs Attributes

- Choice mainly depends on the structure of the enterprise being modeled, and on the semantics associated with the attribute in question

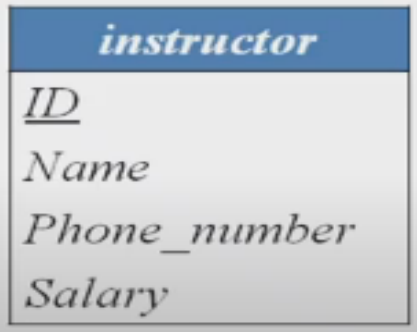


FIG A

FIG B

Use of Entity Sets Vs Relationship Sets

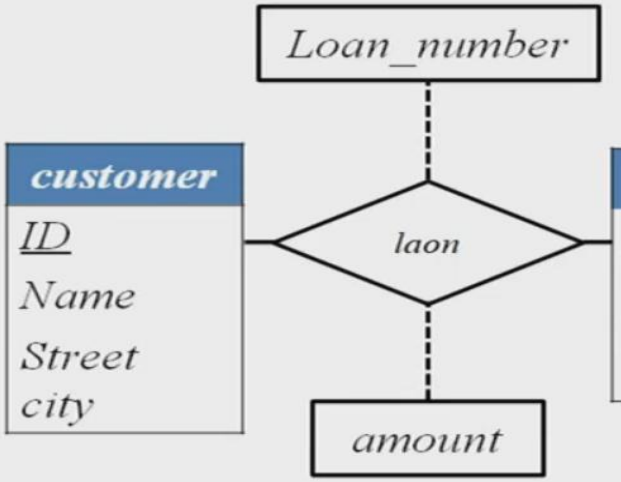


FIG A

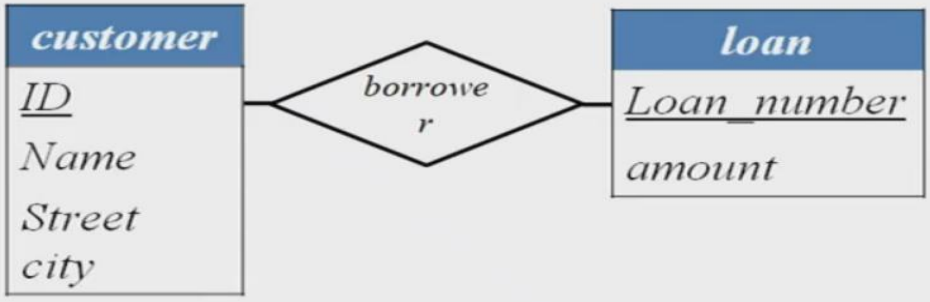


FIG B

Binary Vs *n-array* Relationship Sets

- It is always possible to replace a non-binary relationship set by a number of distinct binary relationship set

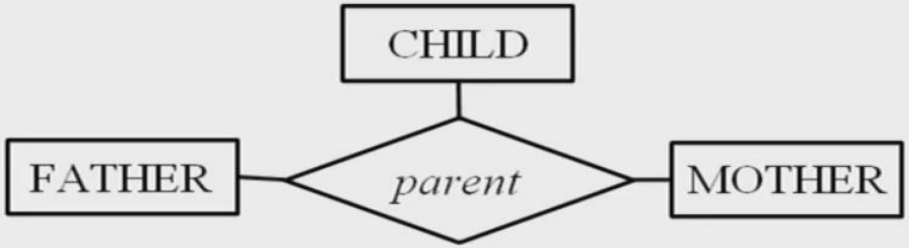


FIG A

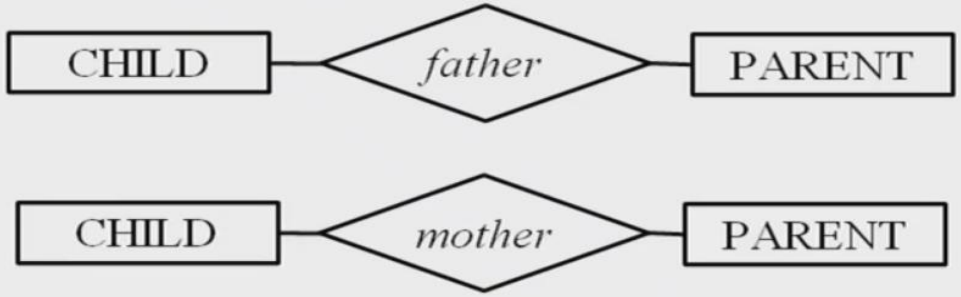
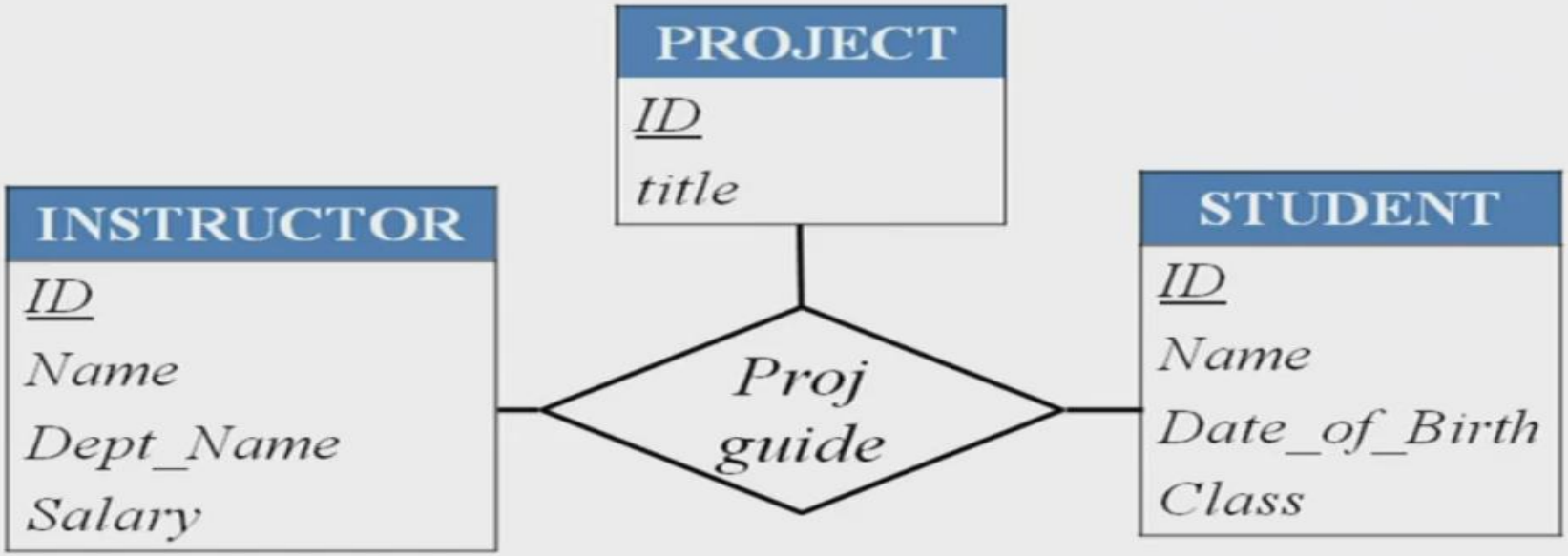


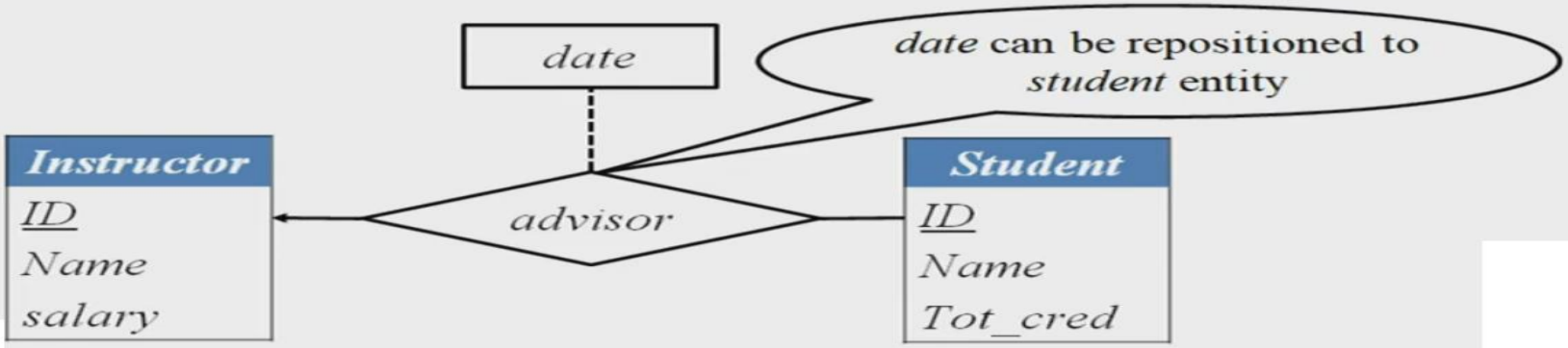
FIG B

- There are some relationship that are naturally non-binary
- Eg: *proj_guide*



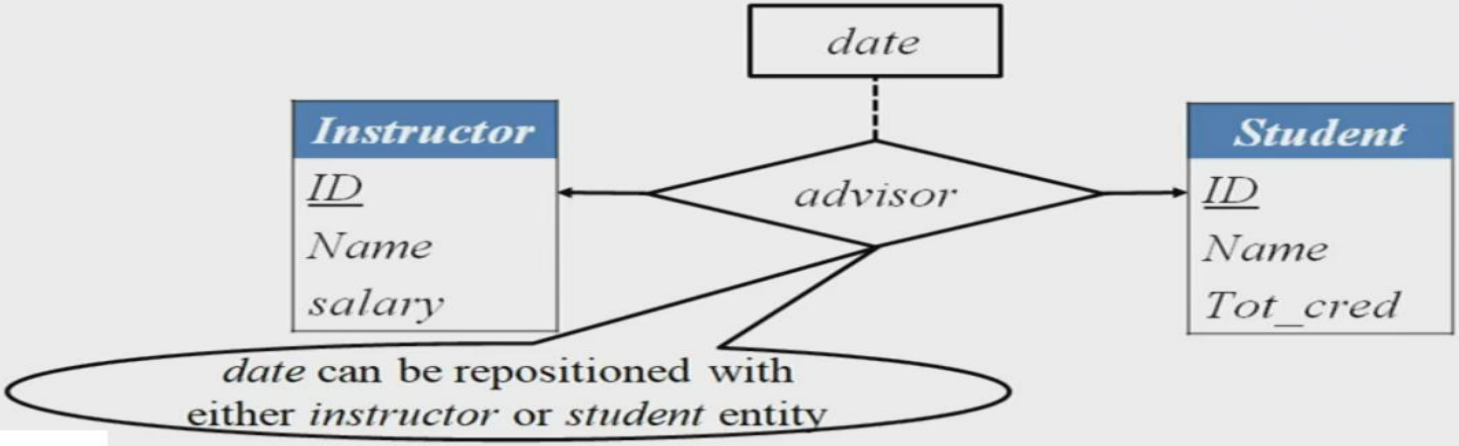
Placement of Relationship Attributes

- Attributes of one – to – many relationship sets can be repositioned to only the entity set on the many side of the relationship.



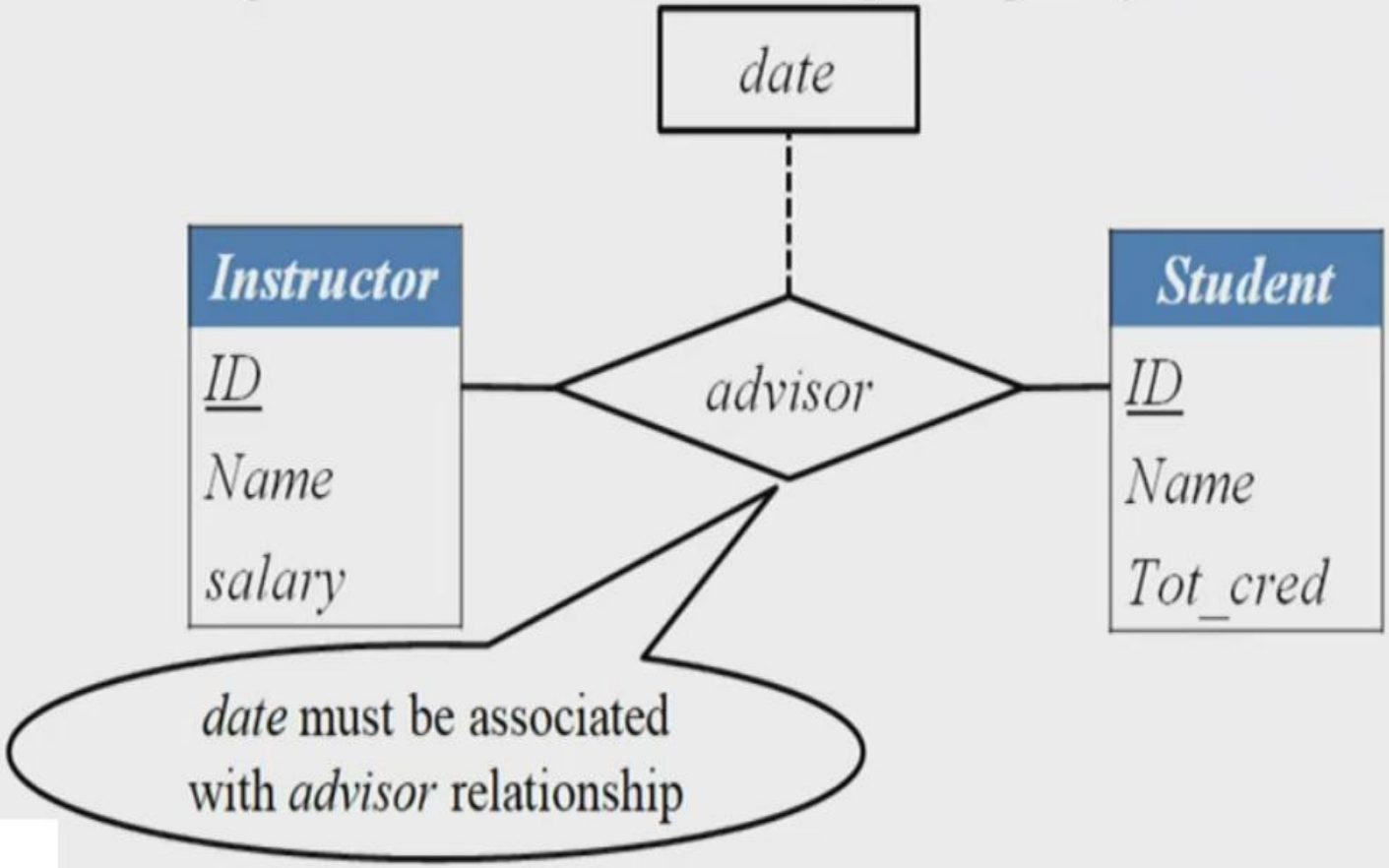
Placement of Relationship Attributes

- For one – to – one relationship sets, the relationship attribute can be associated with either one of the participating entities.

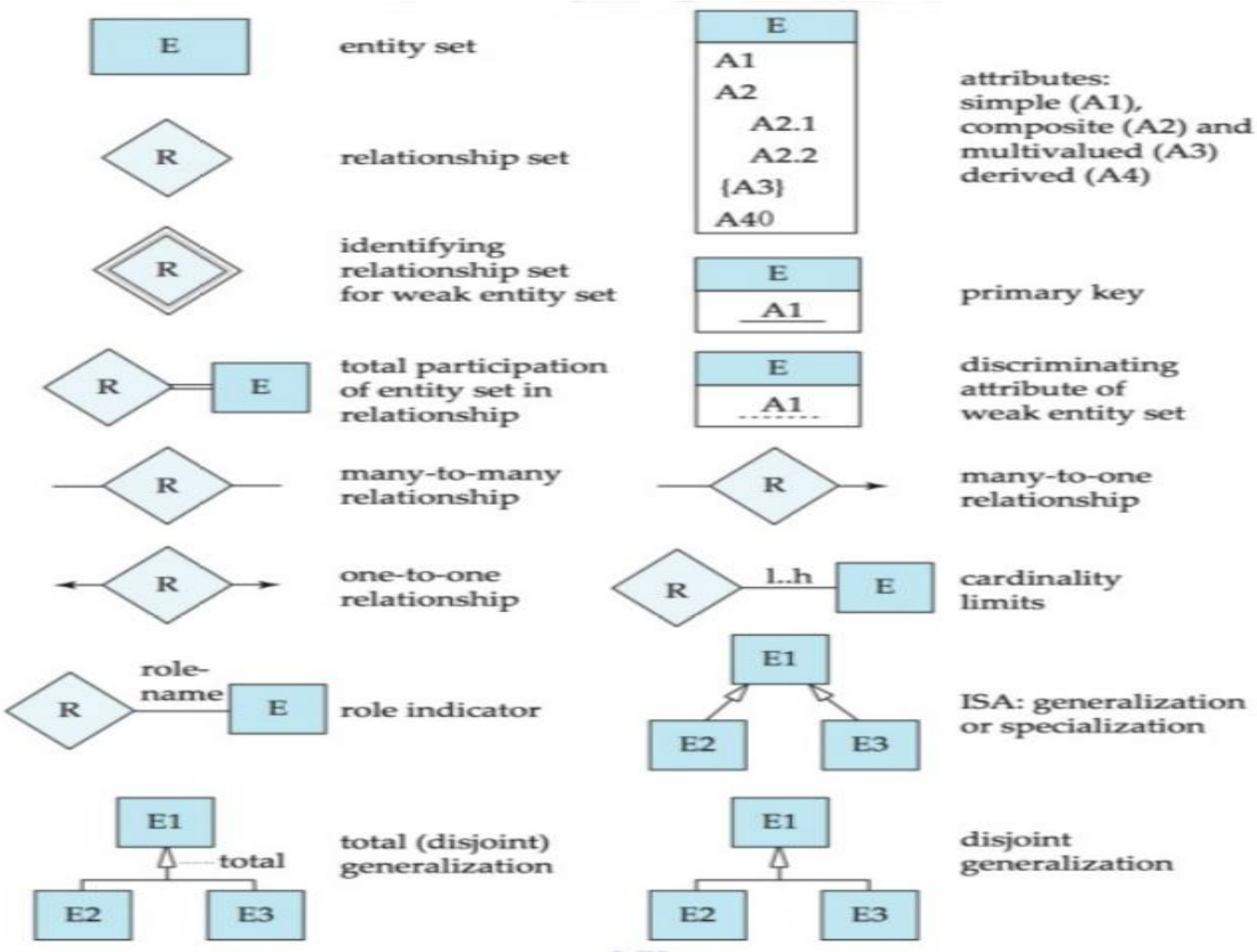


Placement of Relationship Attributes

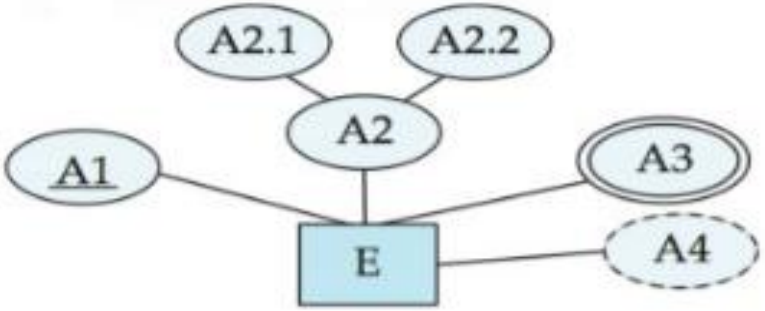
- For many – to – many relationship set, attributes must be associated with relationship sets rather than one of the participating entities.



ALTERNATIVE NOTATIONS FOR MODELING DATA

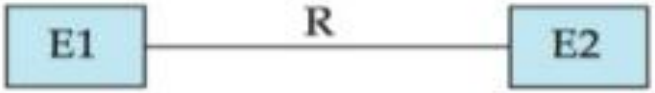


entity set E with
simple attribute A1,
composite attribute A2,
multivalued attribute A3,
derived attribute A4,
and primary key A1



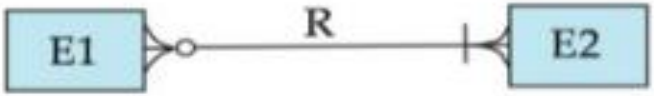
many-to-many relationship

one-to-one relationship



many-to-one relationship

participation in R: total (E1) and partial (E2)

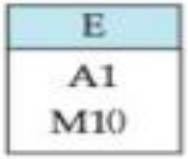


weak entity set

generalization

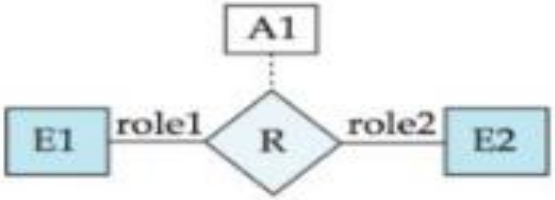
total generalization

ER Diagram Notation



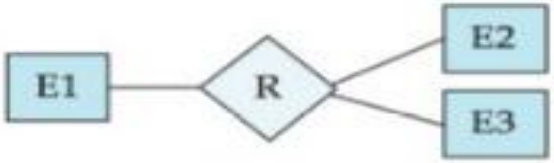
entity with attributes (simple, composite, multivalued, derived)

binary relationship

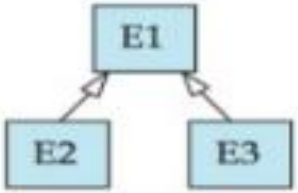


relationship attributes

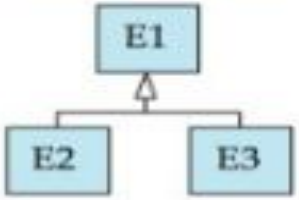
cardinality constraints



n-ary relationships

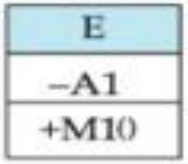


overlapping generalization

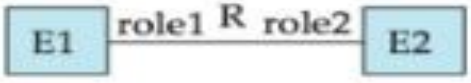


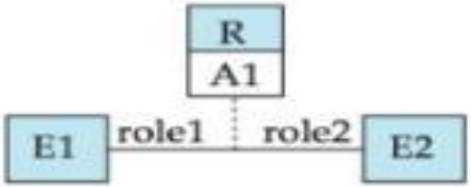
disjoint generalization

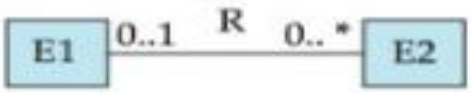
Equivalent in UML

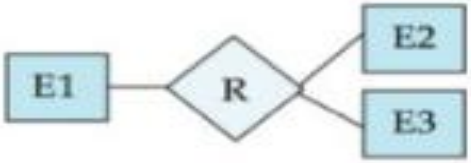


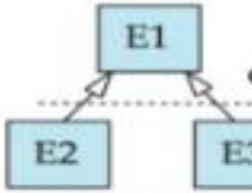
class with simple attributes and methods (attribute prefixes: + = public, – = private, # = protected)



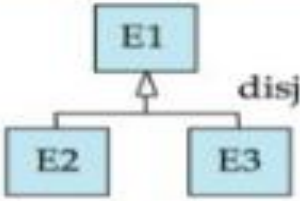








overlapping



disjoint