Database Management System

Overview of Database Management System

- Data – Data is a collection of raw facts and figures.
- Data vs. Information
- Information – Information is the processed, organized data presented in a given context and is useful to humans.
- Database – Collection of interrelated data.

Database

- A database is organized collection of related data of an organization stored in formatted way which is shared by multiple users.
- The main feature of data in a database are:

1. It must be well organized
2. it is related
3. It is accessible in a logical order without any difficulty
4. It is stored only once

Why Database?

In order to overcome the limitation of a file system, a new approach was required.

- The initial attempts were to provide a centralized collection of data.
- For easy data share and well data organization.
- A small database can be handled manually but for a large database and having multiple users it is difficult to maintain it, In that case a computerized database is useful.

Drawbacks of using file systems

- In the early days, database applications were built on top of file systems
- Drawbacks of using file systems to store data:
  - Data redundancy and inconsistency
  - Difficulty in accessing data
  - Data isolation — no features of multiple files and formats

Drawbacks of using file systems (cont.)

- No any solution to Integrity problems like name, mobile number, age.
- Atomicity of updates may be fail.
- No Concurrent access by multiple users
- Security problems

- Database systems offer solutions to all the above problems

Database Management System (DBMS)

- DBMS – A Database management system is a collection of interrelated data and a set of programs to access those data.
- DBMS contains information about a particular enterprise
- DBMS provides an environment that is both *convenient* and *efficient* to use.
- Database Applications:
  - Banking: all transactions
  - Airlines: reservations, schedules
  - Universities: registration, grades
  - Sales: customers, products, purchases
  - Manufacturing: production, inventory, orders, supply chain
  - Human resources: employee records, salaries, tax deductions
- Databases touch all aspects of our lives

Advantages of DBMS:

1. **Defining database structure.**
2. **Reduction of redundancies**
3. **Manipulation of the database.**
4. **Sharing of data.**
5. **Protection of database.**
6. **Database recovery.**

File System	DBMS
Storing and retrieving of data can't be done efficiently in a file system.	DBMS is efficient to use as there are a wide variety of methods to store and retrieve data.
It does not offer data recovery processes.	There is a backup recovery for data in DBMS.
Protecting a file system is very difficult.	DBMS offers good protection mechanism.
In a file management system, the redundancy of data is greater.	The redundancy of data is low in the DBMS system.
Data Sharing is difficult.	Data Sharing is easy.
There is no efficient query processing in the file system.	You can easily query data in a database using the SQL language.

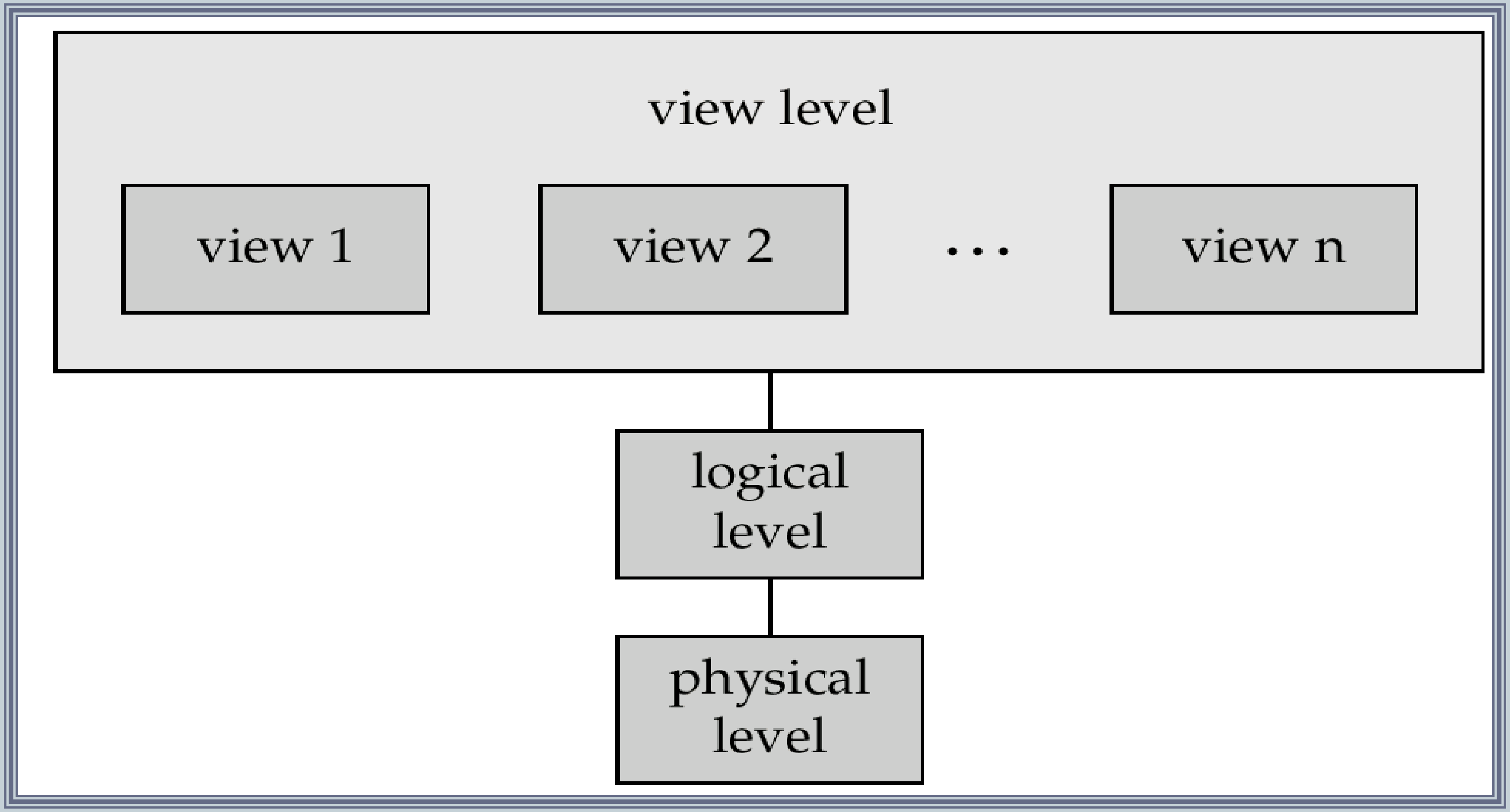
MOST POPULAR DATABASES

1. Oracle
2. MySQL
3. Microsoft SQL Server
4. PostgreSQL
5. MongoDB
6. DB2

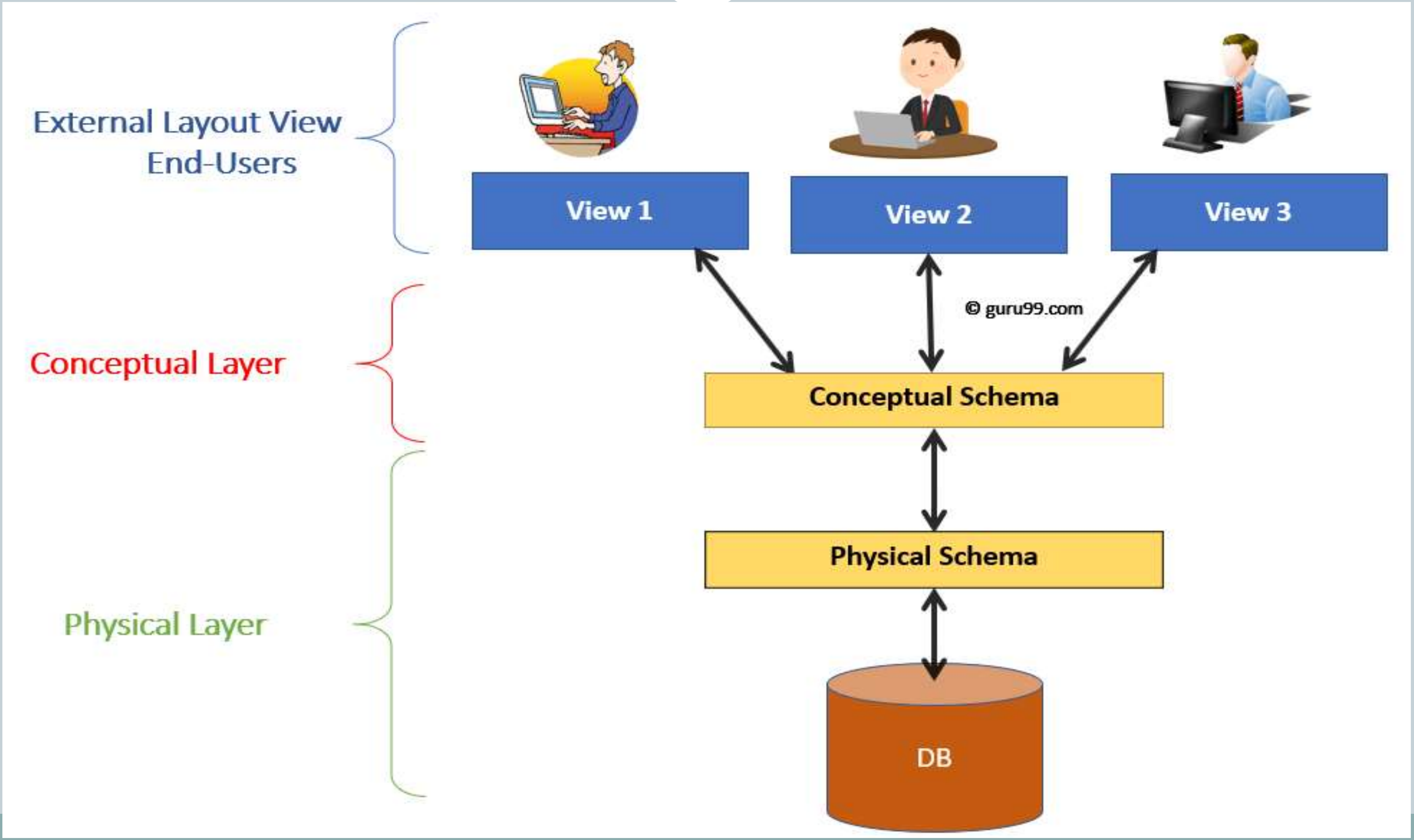
Questions :

- 1. What is DBMS? Explain Data, Information?
- 2. What are difference between file system and database system?

An Architecture for a Database System



An Architecture for a Database System



Instances and Schemas

- Similar to types and variables in programming languages
- **Schema** – the logical structure of the database
  - e.g., the database consists of information about a set of customers and accounts and the relationship between them)
  - Analogous to type information of a variable in a program
  - **Physical schema**: database design at the physical level
  - **Logical schema**: database design at the logical level
- **Instance** – the actual content of the database at a particular point in time
  - Analogous to the value of a variable

Levels of Abstraction

- Physical level describes how a record (e.g., customer) is stored.
- Logical level: describes data stored in database, and the relationships among the data.

```
type customer = record
                        name : string;
                        street : string;
                        city : integer;
                    end;
```

- View level: application programs hide details of data types. Views can also hide information (e.g., salary) for security purposes.

Data Independence

- **Data Independence** is defined as a property of DBMS that helps you to change the Database schema at one level of a database system without requiring to change the schema at the next higher level.
- **Data independence** helps you to keep **data** separated from all programs that make use of it.
- A database system normally contains a lot of data in addition to users' data. For example, it stores data about data, known as **metadata.**

TYPES OF DATA INDEPENDENCE

There are two types of Data Independence

1. **Physical level Data Independence**
2. **Logical level Data Independence**

Physical level Data Independence

- Physical data independence can be defined as the capacity to change the internal schema without having to change the conceptual schema.
- If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.

Example of Physical level Data Independence

- Using a new storage device like Hard Drive or Magnetic Tapes
- Modifying the file organization technique in the Database
- Switching to different data structures.
- Changing the access method.
- Modifying indexes.
- Changes to compression techniques or hashing algorithms.
- Change of Location of Database from say C drive to D Drive

Logical level Data Independence

- Logical data independence refers characteristic of being able to change the conceptual schema without having to change the external schema.
- Logical data independence is used to separate the external level from the conceptual view.
- If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.

Examples of Logical Data Independence

- Due to Logical independence, any of the below change will not affect the external layer.
- Add/Modify/Delete a new attribute, entity or relationship is possible without a rewrite of existing application programs
- Merging two records into one
- Breaking an existing record into two or more records

Database Administration Roles

- A **Database Administrator (DBA)** is individual or person responsible for controlling, maintenance, coordinating, and operation of database management system. Managing, securing, and taking care of database system is prime responsibility.

Role and Duties of Database Administrator (DBA)

- Decides hardware
- Manages data integrity and security
- Database design
- Database implementation
- Query processing performance

DBMS Architecture

- **Types of DBMS Architecture**

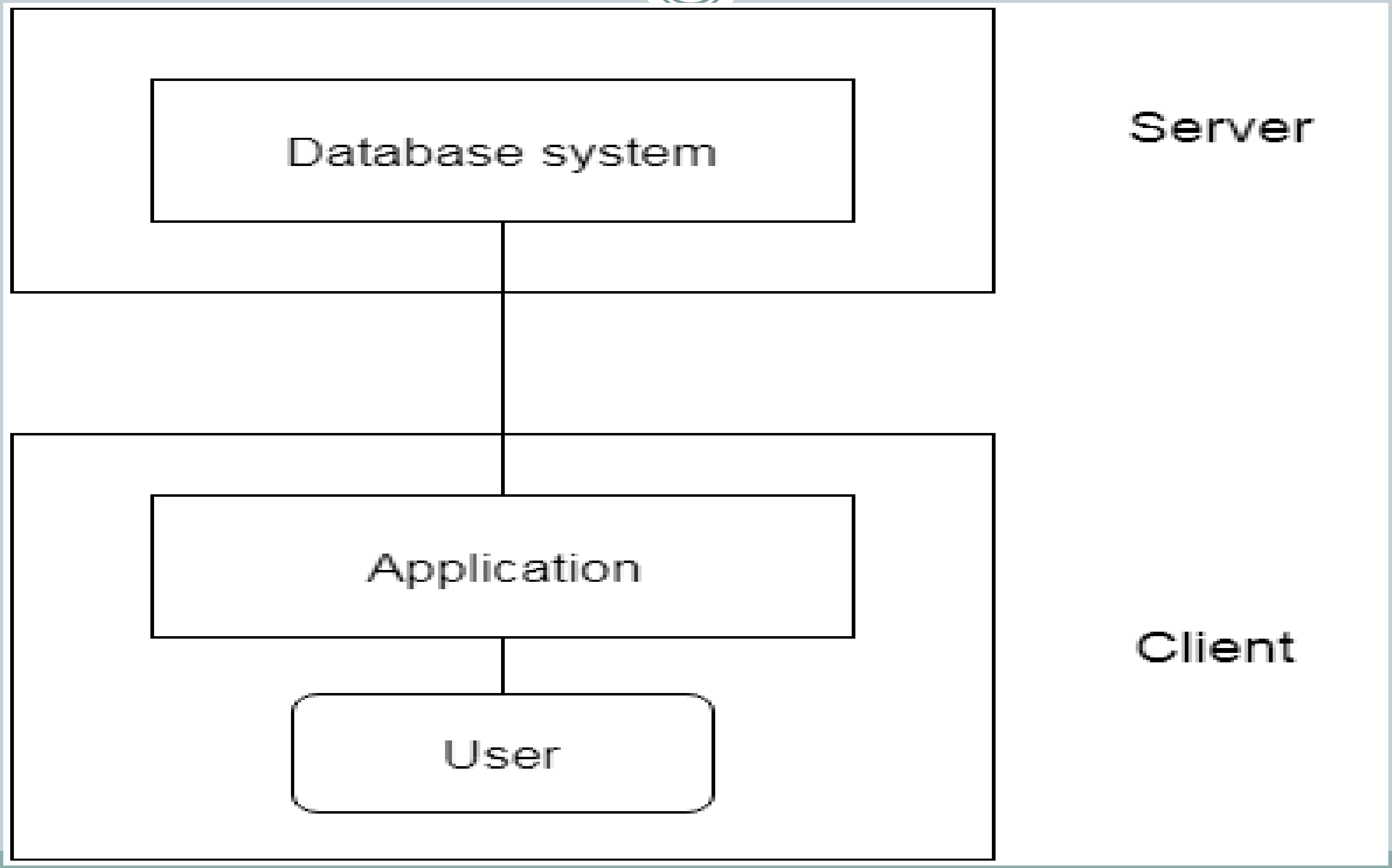
There are three types of DBMS architecture:

- 1. Single tier architecture
  2. Two tier architecture
  3. Three tier architecture

1. Single tier architecture

- In this type of architecture, the database is readily available on the client machine, any request made by client doesn't require a network connection to perform the action on the database.
- Example – Own System

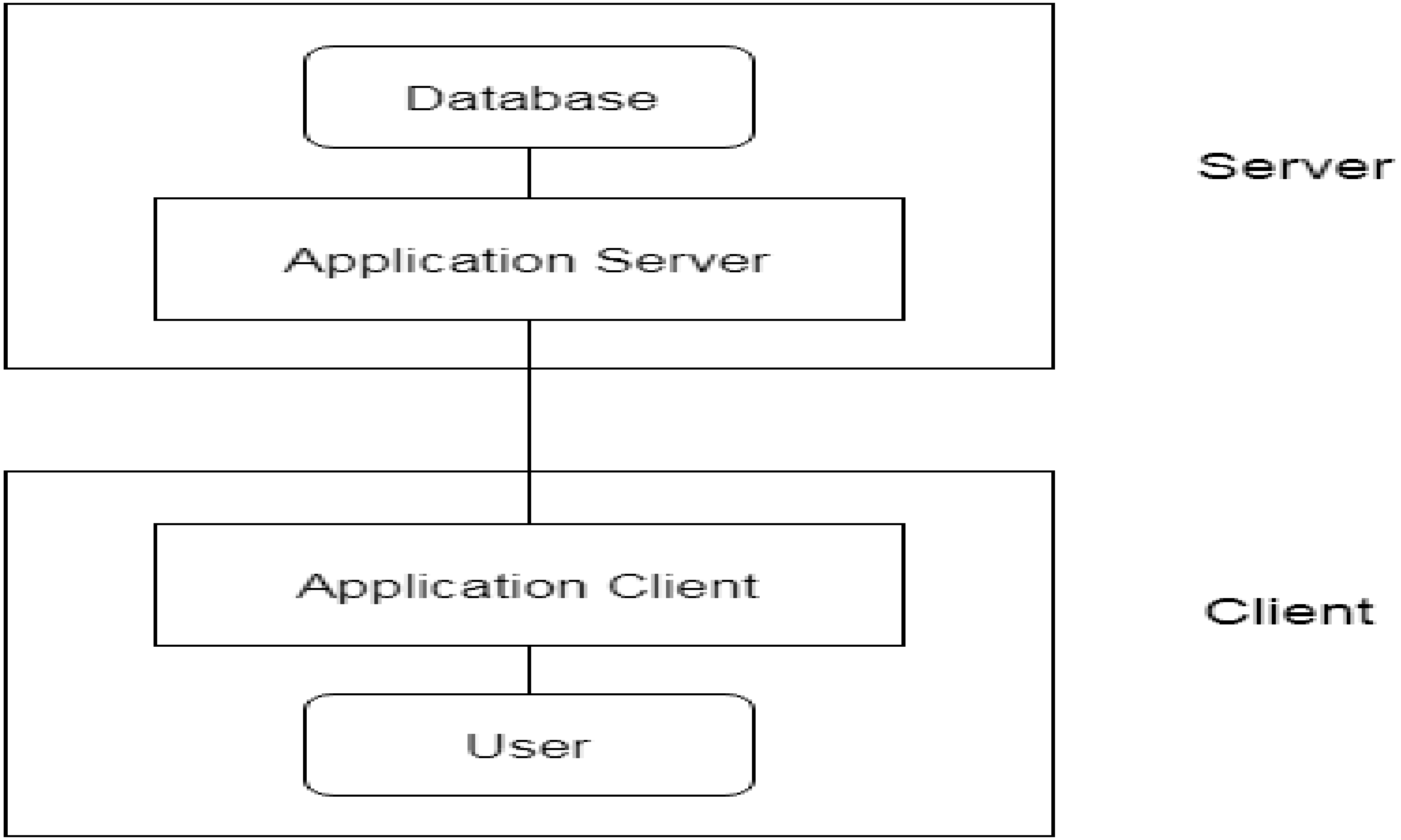
2. Two tier architecture



2. Two tier architecture

- The 2-Tier architecture is same as basic client-server.
- In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: **ODBC**, **JDBC** are used.
- **ODBC** = Open Database Connectivity
- **JDBC** = Java Database Connectivity

3. Three tier Architecture



3. Three tier Architecture

- The 3-Tier architecture contains another layer between the client and server.
- In this architecture, client can't directly communicate with the server.
- The application on the client-end interacts with an application server which further communicates with the database system.

Types of Database Users

- Database users are the one who really use and take the benefits of database.
- There are different types of users depending on their need and way of accessing the database.

1. **Application Programmers**
2. **Sophisticated Users**
3. **Specialized Users**
4. **Naive Users**

.

Application Programmers

- **Application Programmers –** They are the developers who interact with the database by means of DML queries.
- These DML queries are written in the application programs like C, C++, JAVA, Pascal etc.
- These queries are converted into object code to communicate with the database.
- For example, writing a C program to generate the report of employees who are working in particular department will involve a query to fetch the data from database. It will include a embedded SQL query in the C Program.

Sophisticated Users

- They are database developers, who write SQL queries to select/insert/delete/update data.
- They do not use any application or programs to request the database. .
- They directly interact with the database by means of query language like SQL. These users will be scientists, engineers, analysts who thoroughly study SQL and DBMS to apply the concepts in their requirement.
- In short, we can say this category includes designers and developers of DBMS and SQL.

Specialized Users

- These are also sophisticated users, but they write special database application programs.
- They are the developers who develop the complex programs to the requirement.

Naive Users

- These are the users who use the existing application to interact with the database.
- For example, online library system, ticket booking systems, ATMs etc. which has existing application and users use them to interact with the database to fulfill their requests

Summary of Database Users

- Users are differentiated by the way they expect to interact with the system
- Application programmers – interact with system through DML calls
- Sophisticated users – form requests in a database query language
- Specialized users – write specialized database applications that do not fit into the traditional data processing framework
- Naïve users – invoke one of the permanent application programs that have been written previously
  - E.g. people accessing database over the web, bank tellers, clerical staff

Data Dictionary

- A Data Dictionary contains metadata i.e data about the database.
- The data dictionary is very important as it contains information such as what is in the database, who is allowed to access it, where is the database physically stored etc.
- The users of the database normally don't interact with the data dictionary, it is only handled by the database administrators.

Contents of Data Dictionary

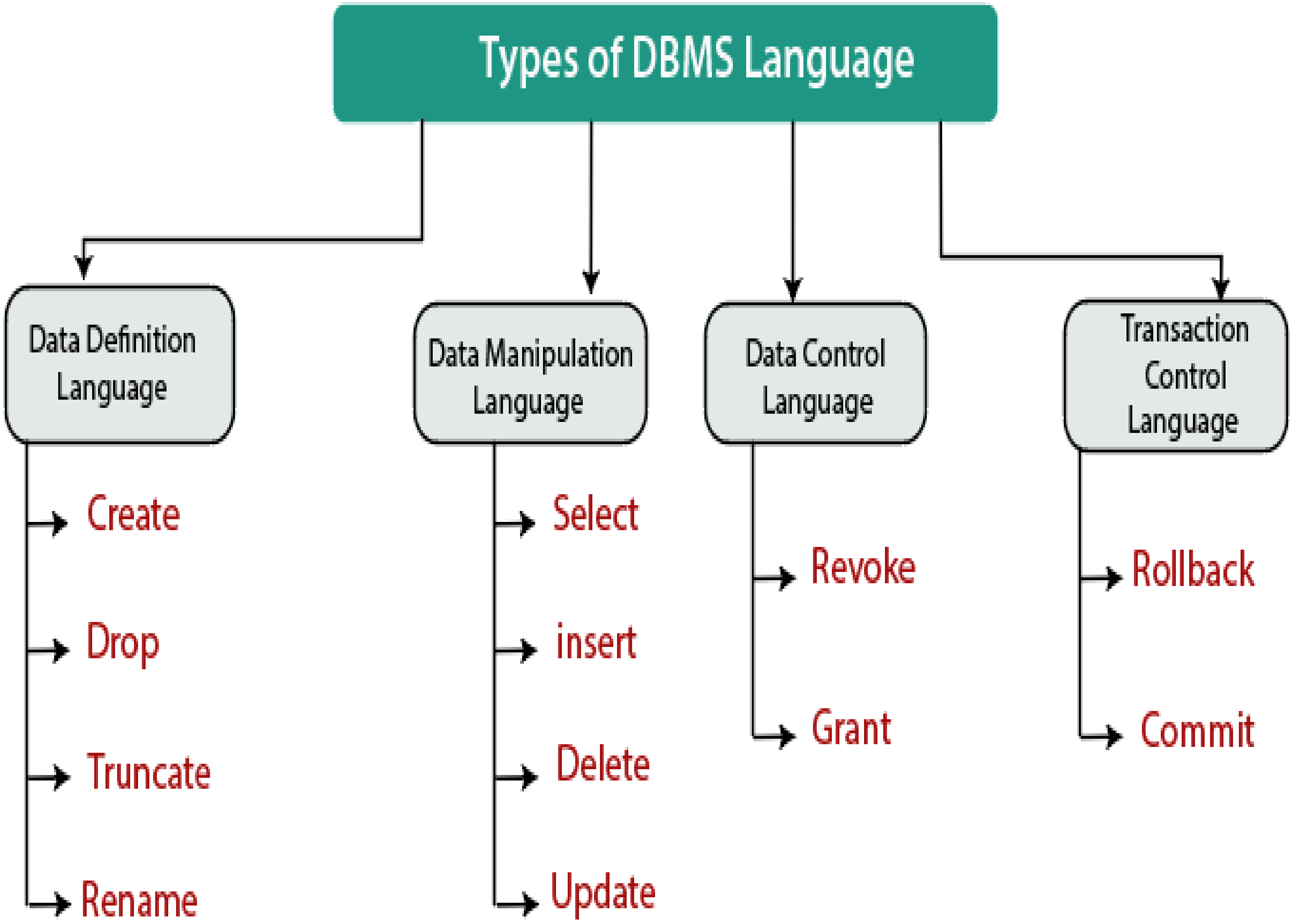
- Names of all the database tables and their schemas.
- Details about all the tables in the database, such as their owners, their security constraints, when they were created etc.
- Physical information about the tables such as where they are stored and how.
- Table constraints such as primary key attributes, foreign key information etc.
- Information about the database views that are visible.

This is a Data Dictionary describing a table that contains employee details.

Field Name	Data Type	Field Size for display	Description	Example
Employee Number	Integer	10	Unique ID of each employee	1645000001
Name	Text	20	Name of the employee	David Heston
Date of Birth	Date/Time	10	DOB of Employee	08/03/1995
Phone Number	Integer	10	Phone number of employee	6583648648

Database Language

- A DBMS has appropriate languages and interfaces to express database queries and updates.
- Database languages can be used to read, store and update the data in the database.


Types of DBMS Language
Data Definition Language
Create
Drop
Truncate
Rename
Data Manipulation Language
Select
insert
Delete
Update
Data Control Language
Revoke
Grant
Transaction Control Language
Rollback
Commit

1. Data Definition Language

- **DDL** stands for **D**ata **D**efinition **L**anguage. It is used to define database structure or pattern.
- It is used to create schema, tables, indexes, constraints, etc. in the database.
- Using the DDL statements, you can create the structure of the database.
- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Commands under DDL

- **Create:** It is used to create objects in the database.
- **Alter:** It is used to alter the structure of the database.
- **Drop:** It is used to delete objects from the database.
- **Truncate:** It is used to remove all records from a table.
- **Rename:** It is used to rename an object.
- These commands are used to update the database schema that's why they come under Data definition language.

2. Data Manipulation Language

- **DML** stands for **D**ata **M**anipulation **L**anguage.
- It is used for accessing and manipulating data in a database.
- It handles user requests.

Commands under DML

- **Select:** It is used to retrieve data from a database.
- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete all records from a table.

3. Data Control Language

- **DCL** stands for **D**ata **C**ontrol **L**anguage. It is used to retrieve the stored or saved data.
- The DCL execution is transactional. It also has rollback parameters.

Commands under DCL

- **Grant:** It is used to give user access privileges to a database.
- **Revoke:** It is used to take back permissions from the user.

4. Transaction Control Language

- TCL is used to run the changes made by the DML statement.
- TCL can be grouped into a logical transaction.

Commands under TCL

- **Commit:** It is used to save the transaction on the database.
- **Rollback:** It is used to restore the database to original since the last Commit.

Database Model

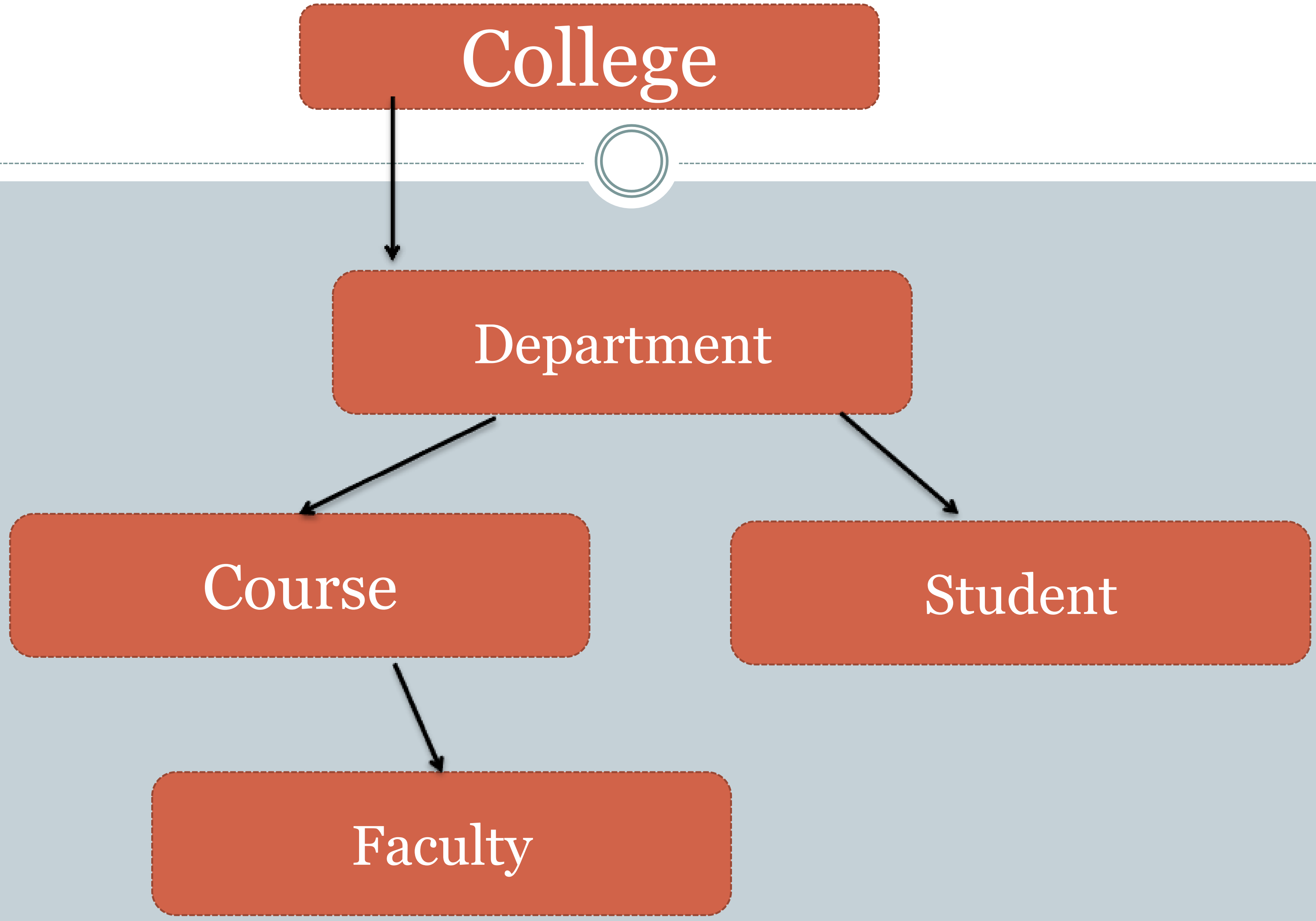
- Database model is a logical frame in which data is stored.
- The model also describes the relationship between different parts of the data.
- Logical vs Physical (ex- Cafeteria)

Types of Data Model

1. Hierarchical Data Model
2. Network Model
3. Relational Data Model

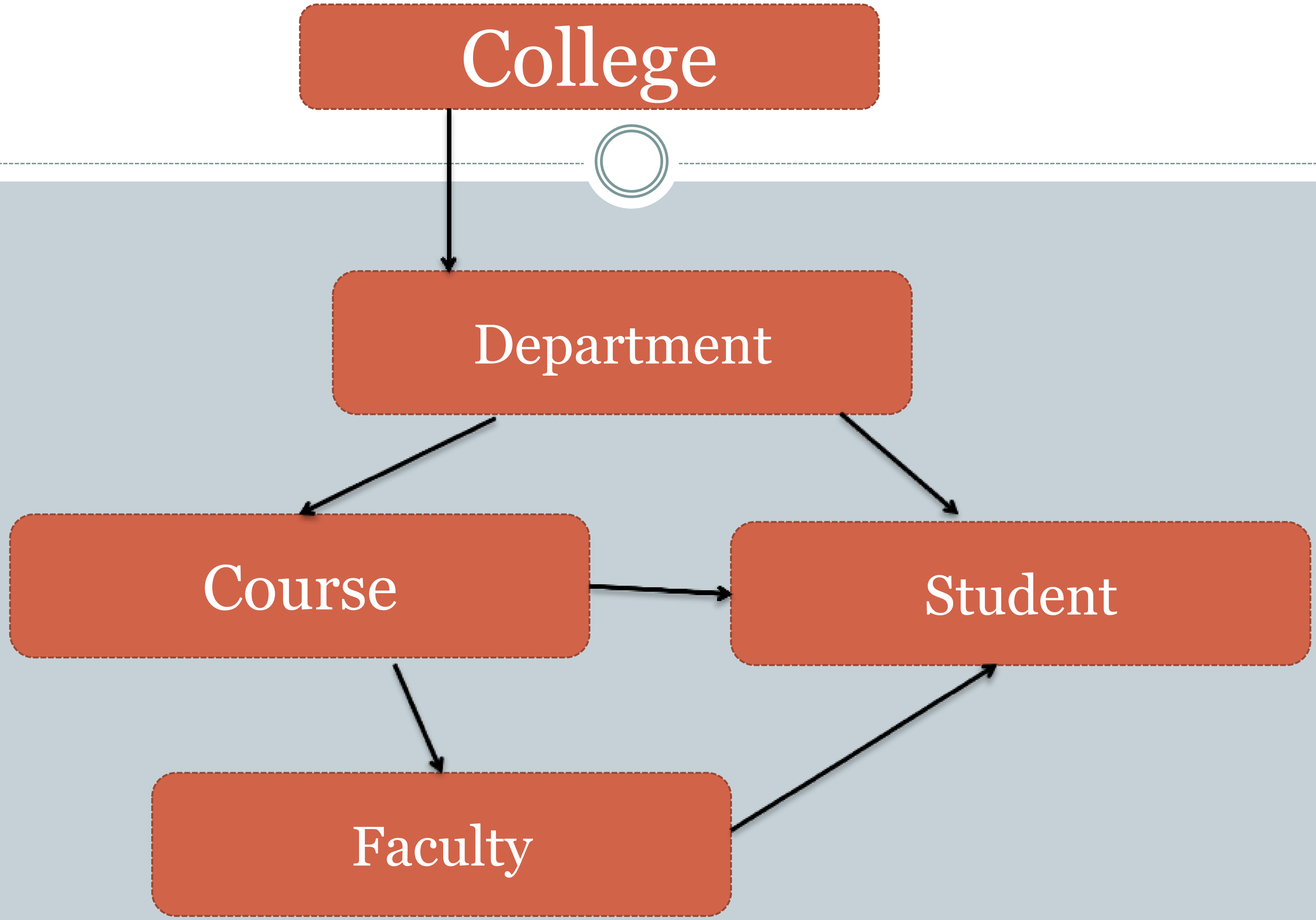
Hierarchical Data Model

- In this model each entity has only one parent but can have several children.
- At the top of the hierarchy there is only one entity which is called Root.
- Example - College


College
Department
Course
Student
Faculty

Network Model

- In this model, Entities are organized in a graph, in which entities can be accessed using several paths.
- There may be multiple relationship between entities.


College
Department
Course
Student
Faculty

Relational Model

- In this model, data is organized in the form of two dimension tables called Relations.
- Here we have entities, relationship between entities.

Relational Model

- Example of tabular data in the relational model

Attributes

Customer-id	*customer-name*	*customer-street*	*customer-city*	*account-number*
192-83-7465	Johnson	Alma	Palo Alto	A-101
019-28-3746	Smith	North	Rye	A-215
192-83-7465	Johnson	Alma	Palo Alto	A-201
321-12-3123	Jones	Main	Harrison	A-217
019-28-3746	Smith	North	Rye	A-201

A Sample Relational Database

customer-id	*customer-name*	*customer-street*	*customer-city*
192-83-7465	Johnson	12 Alma St.	Palo Alto
019-28-3746	Smith	4 North St.	Rye
677-89-9011	Hayes	3 Main St.	Harrison
182-73-6091	Turner	123 Putnam Ave.	Stamford
321-12-3123	Jones	100 Main St.	Harrison
336-66-9999	Lindsay	175 Park Ave.	Pittsfield
019-28-3746	Smith	72 North St.	Rye

(a) The *customer* table

account-number	*balance*
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

(b) The *account* table

customer-id	*account-number*
192-83-7465	A-101
192-83-7465	A-201
019-28-3746	A-215
677-89-9011	A-102
182-73-6091	A-305
321-12-3123	A-217
336-66-9999	A-222
019-28-3746	A-201

(c) The *depositor* table

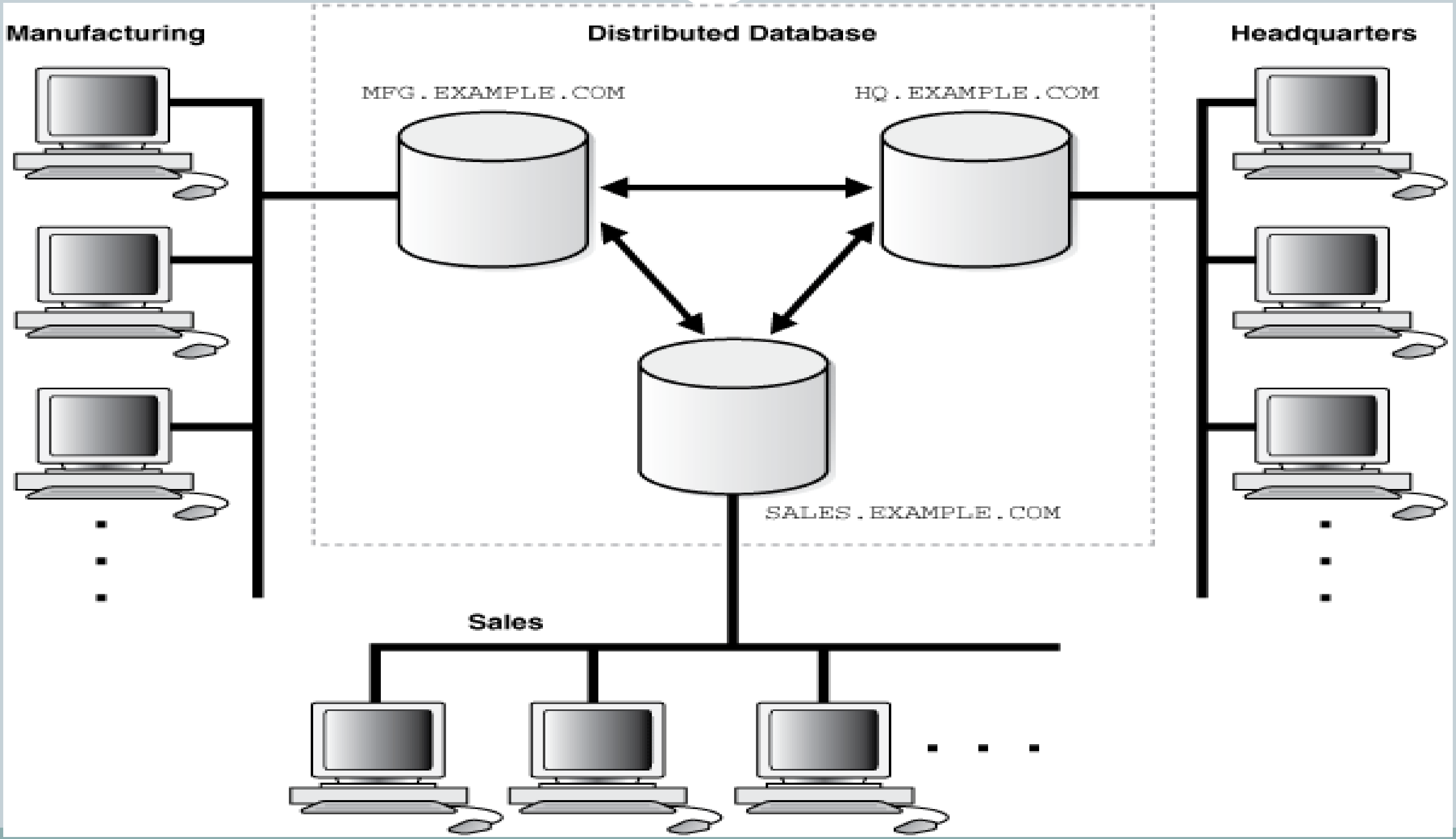
Distributed Database

- A **Distributed Database** is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.

Features

- Databases in the collection are logically interrelated with each other. Often they represent a single logical database.
- Data is physically stored across multiple sites. Data in each site can be managed by a DBMS independent of the other sites.

Distributed Database



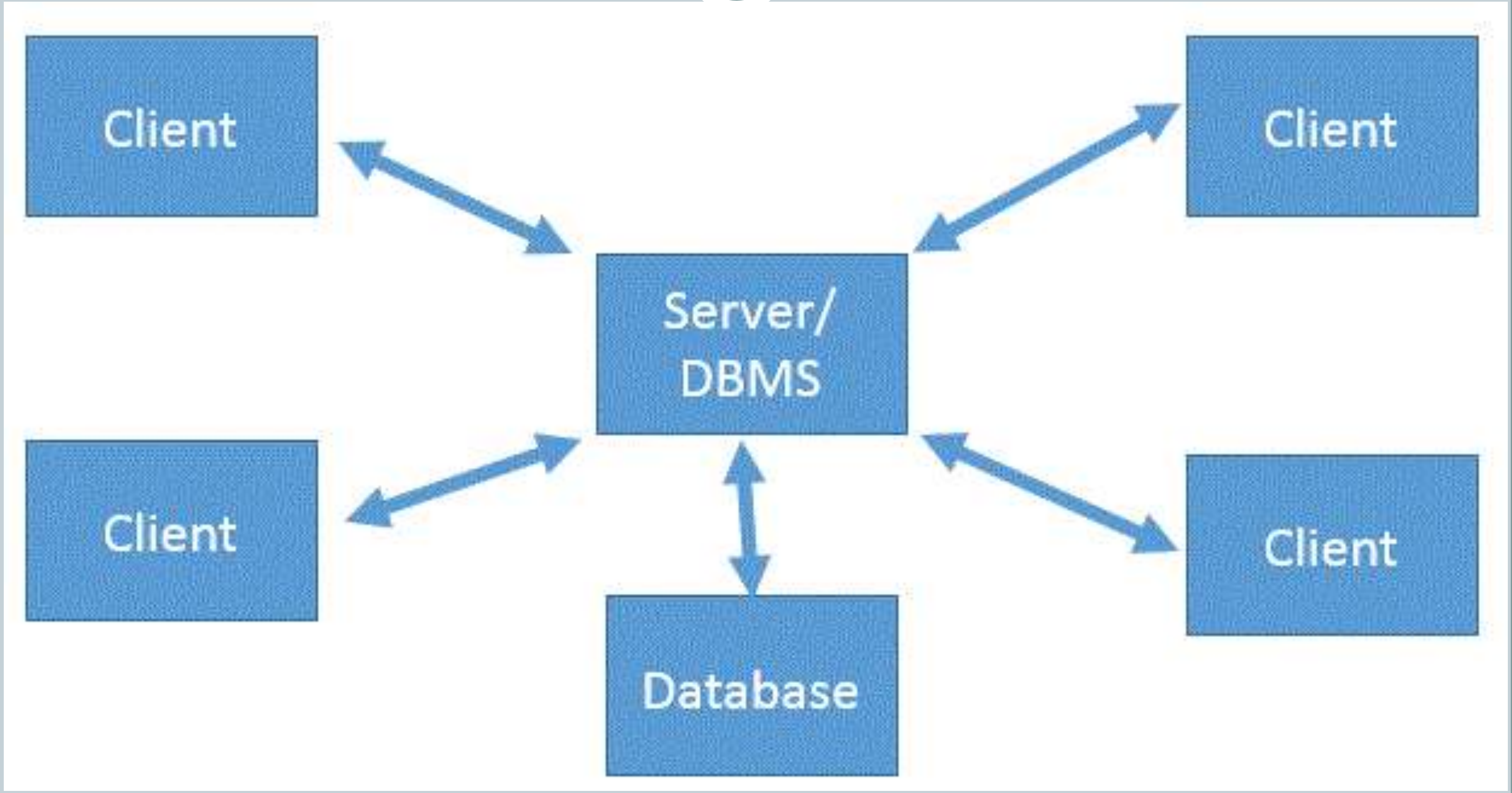
Advantages

- **Modular Development**
- **More Reliable**
- **Better Response**
- **Lower Communication Cost**

Client-Server Database

- **Client-server architecture**, **architecture** of a computer network in which many **clients** (remote processors) request and receive service from a centralized **server** (host computer).
- **Client** computers provide an interface to allow a computer user to request services of the **server** and to display the results the **server** returns.

Client-Server Database

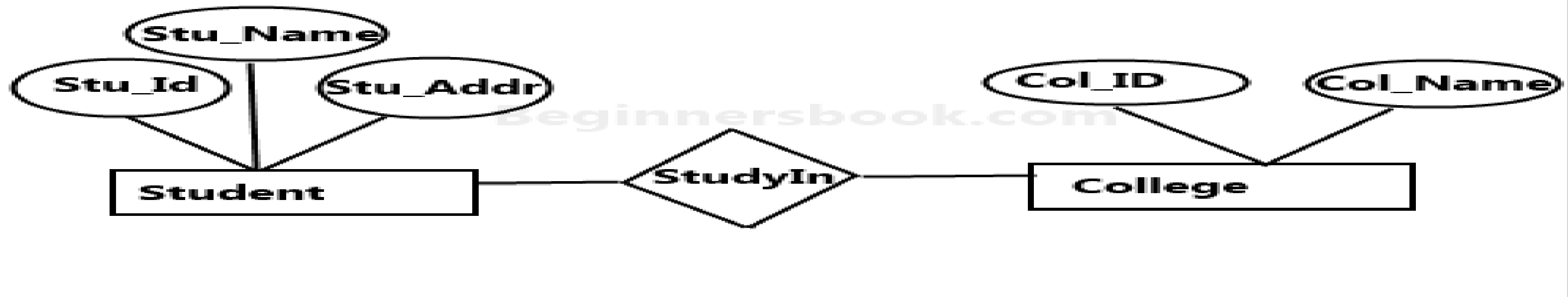


Advantages of Client/Server Database System

- Client/Server system has less expensive platforms to support applications that had previously been running only on large and expensive mini or mainframe computers.
- Client offer icon-based menu-driven interface, which is superior to the traditional command-line, dumb terminal interface typical of mini and mainframe computer systems.
- Client/Server environment facilitates in more productive work by the users and making better use of existing data.

ER Model - Basic Concepts

- 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)**.
- An ER model is a design or blueprint of a database that can later be implemented as a database.



Sample E-R Diagram

Entity

- An entity can be a real-world object, that can be easily identifiable.
- For example, in a school database, students, teachers, classes, and courses offered can be considered as entities.
- All these entities have some attributes or properties that give them their identity.
- An entity set is a collection of similar types of entities.
- Example – Group of Students.

Attributes

- Entities are represented by means of their properties, called **attributes**. All attributes have values.
- For example, a student entity may have name, class, and age as attributes.

Types of Attributes

1. **Simple attribute**
2. **Composite attribute**
3. **Derived attribute**
4. **Single-value attribute**
5. **Multi-value attribute**

Relationship

- The association among entities is called a relationship.
- For example, an employee **works at** a department, a student **enrolls** in a course. Here, Works at and Enrolls are called relationships

Relationship Set

- A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes.

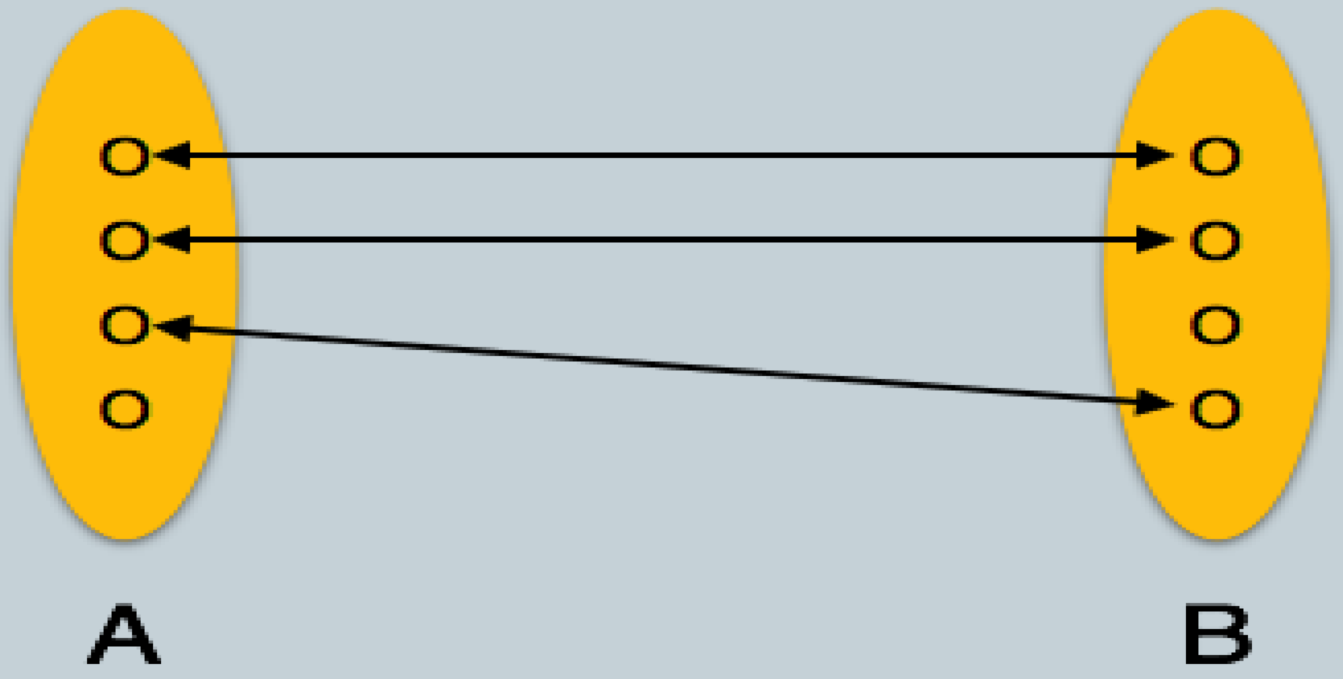
Mapping Cardinalities

- **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.
- There are Four Types of Mapping Cardinalities

1. One to One
2. One-to-many
3. Many-to-one
4. Many-to-many

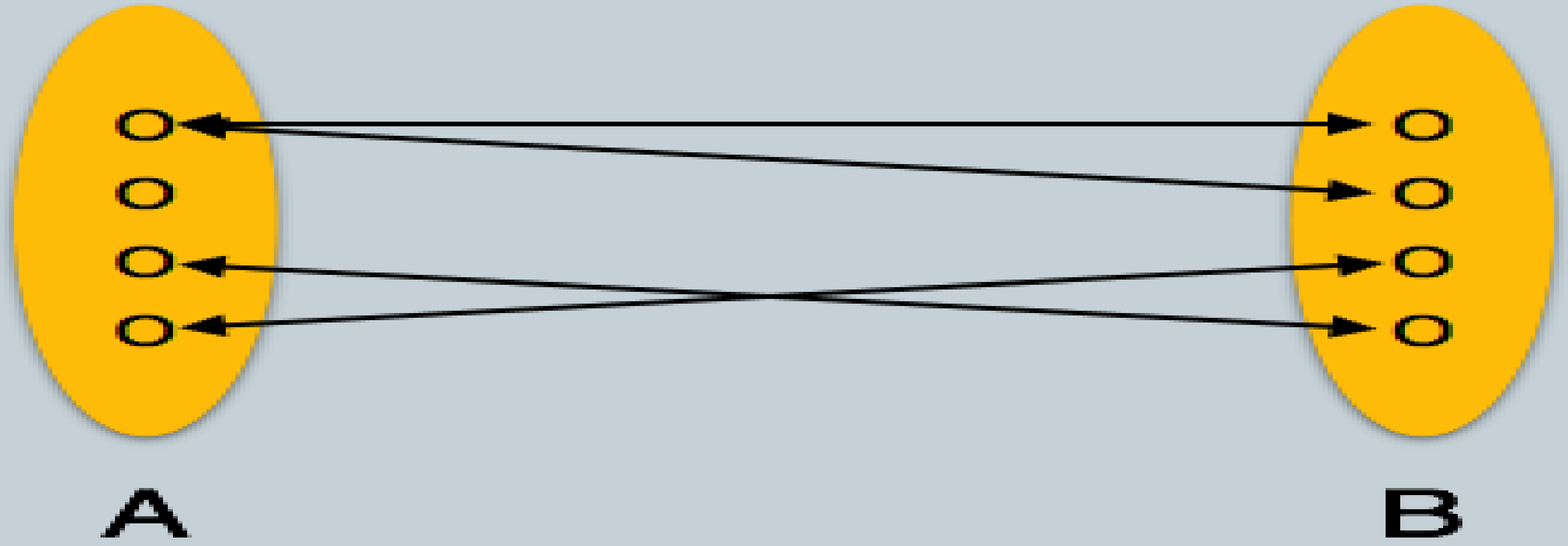
One-to-one Relationship

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



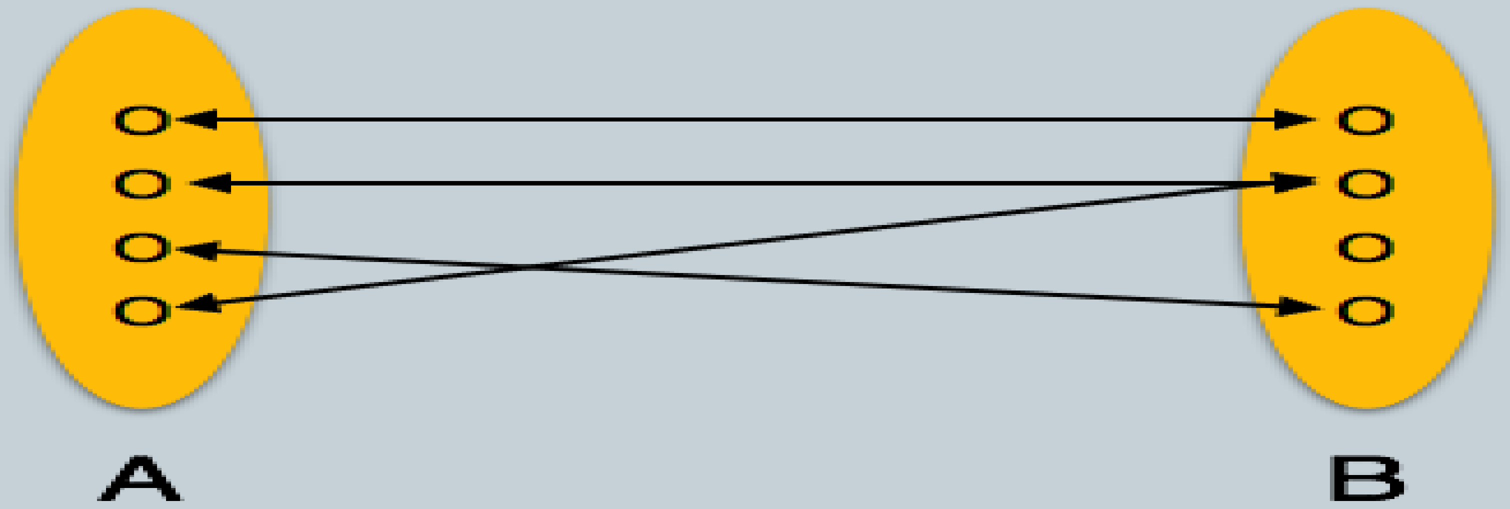
One-to-many Relationship

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



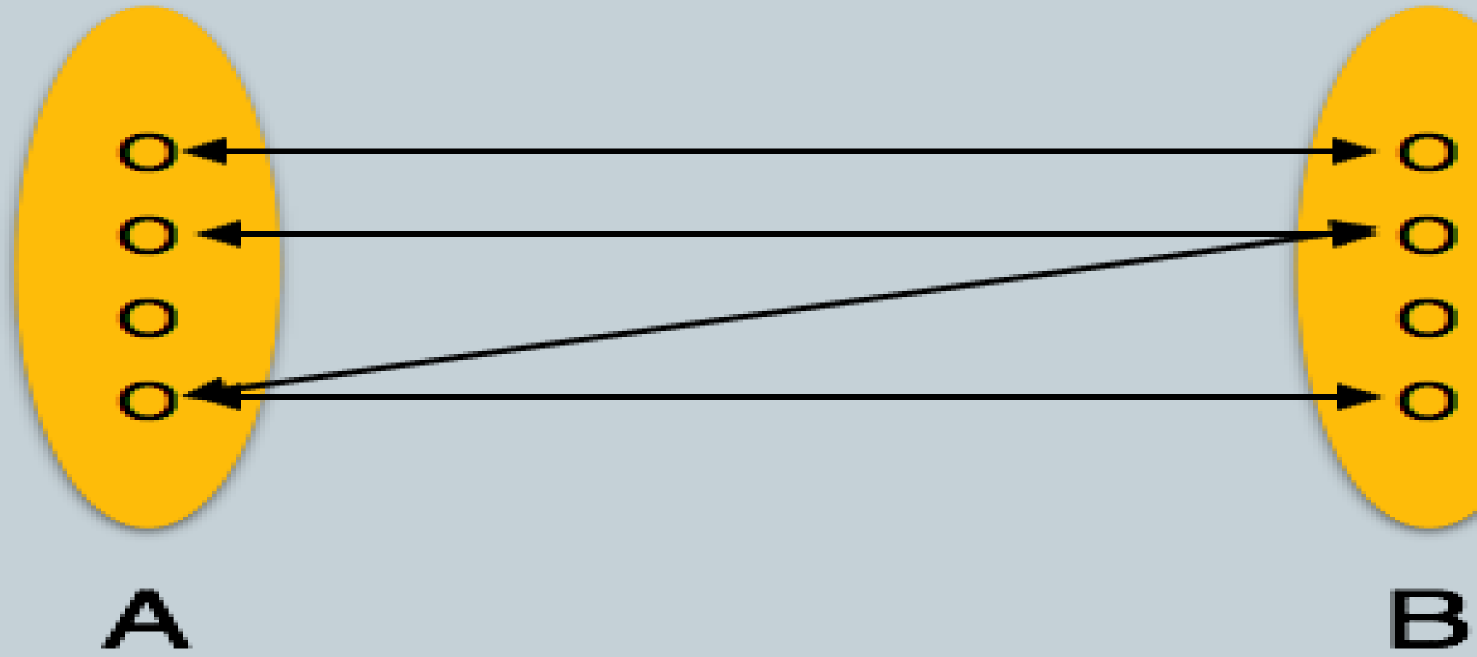
Many-to-one Relationship

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



Many-to-many Relationship

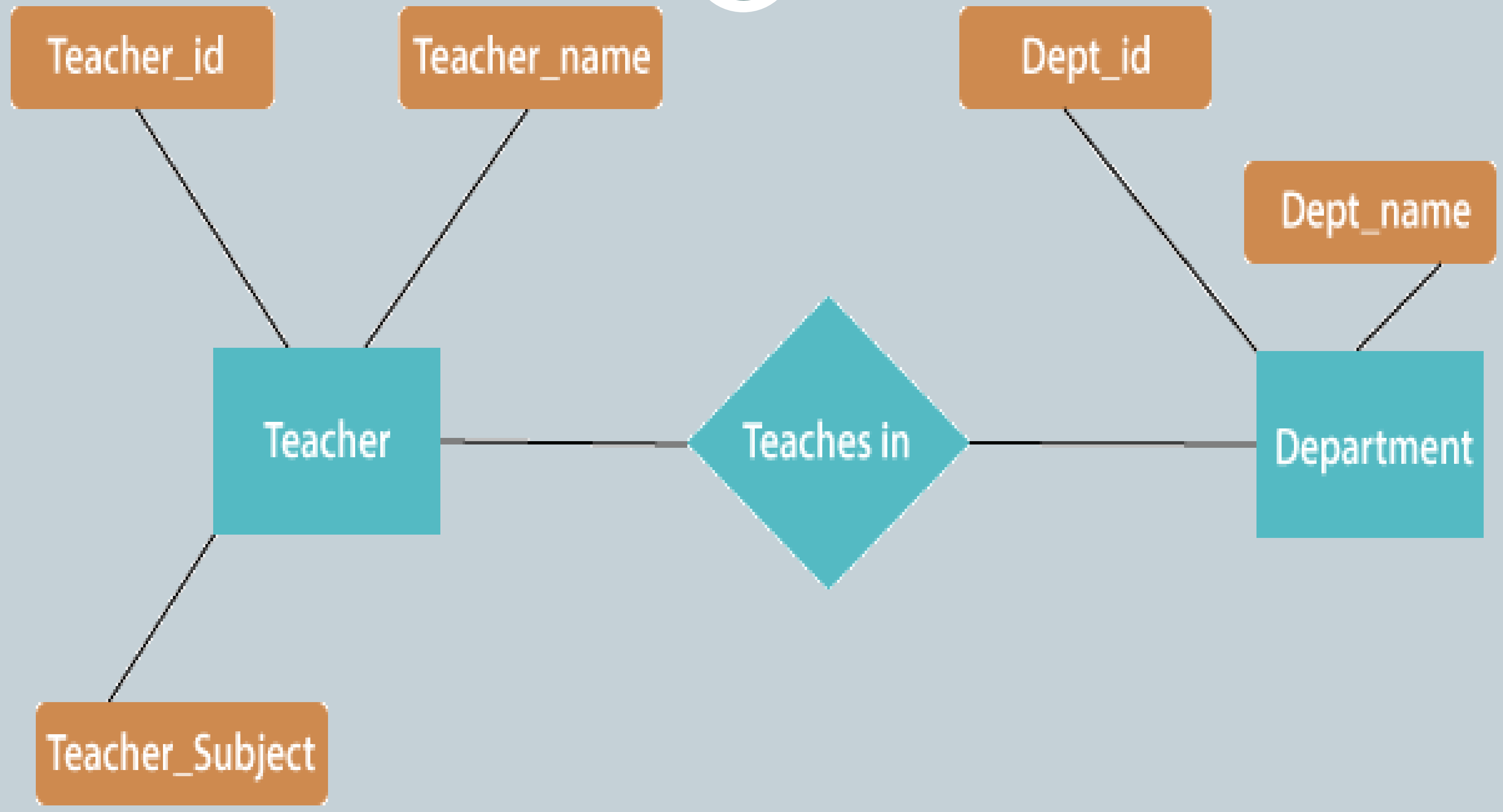
- **Many-to-many** – One entity from A can be associated with more than one entity from B and vice versa.



Entity Relationship diagram(ER Diagram)

- Introduced by Dr Peter Chen in 1976.
- It is a Non-technical design method works on conceptual level based on the perception of real world.
- Consists of entities, attributes and relationships.
- Basically it is a diagrammatic representation of database.

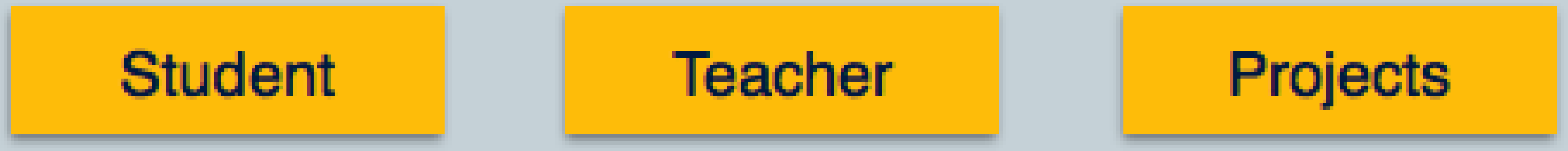
Entity-Relationship Model



ER Diagram Representation

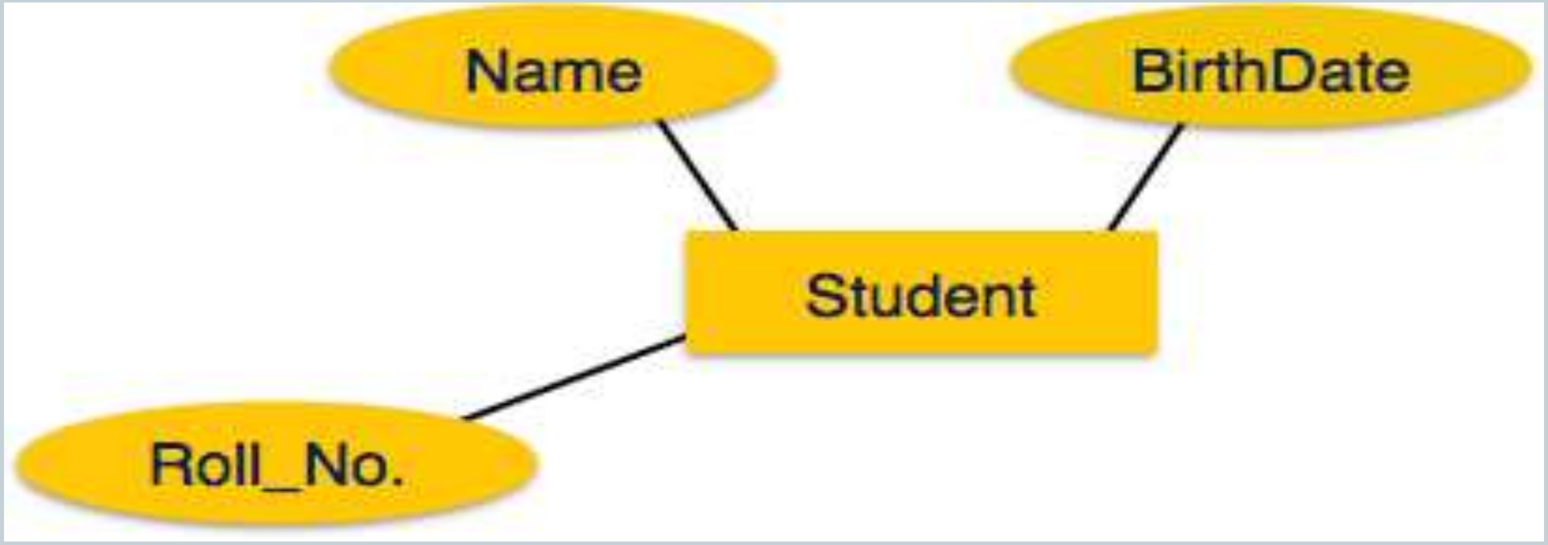
1. Entity

- Entities are represented by means of rectangles.
- Rectangles are named with the entity set they represent.

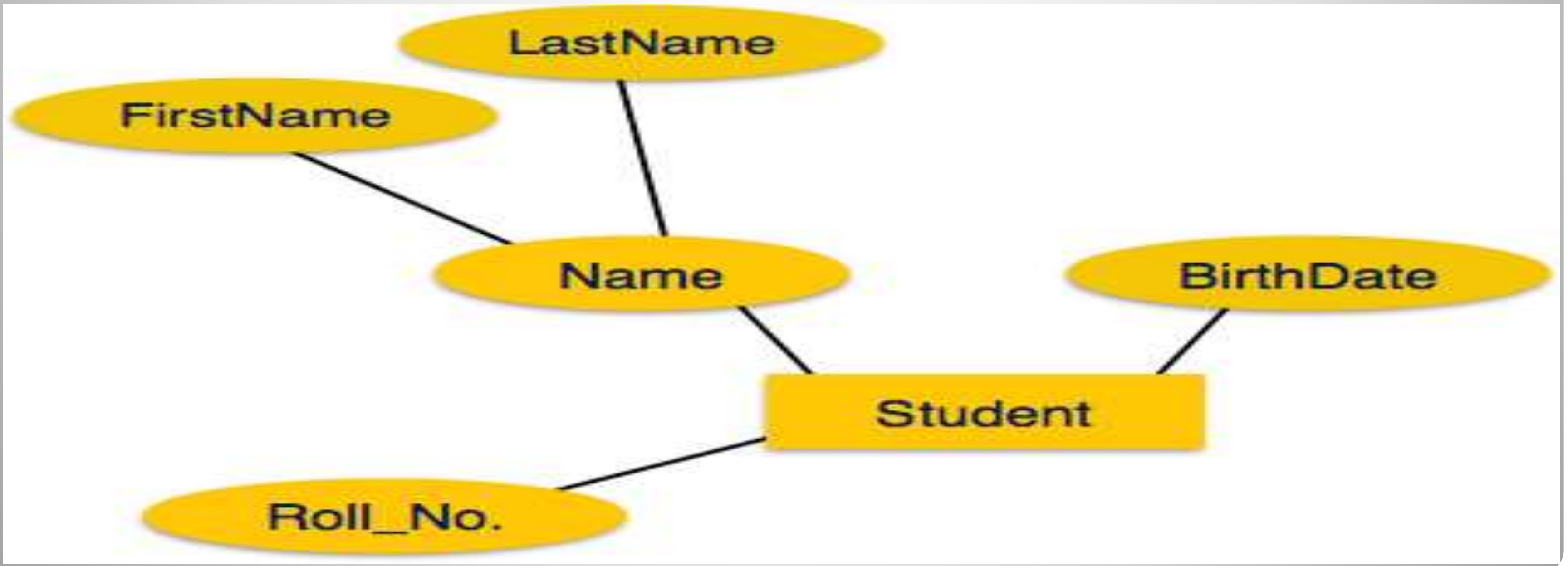


2. Attributes

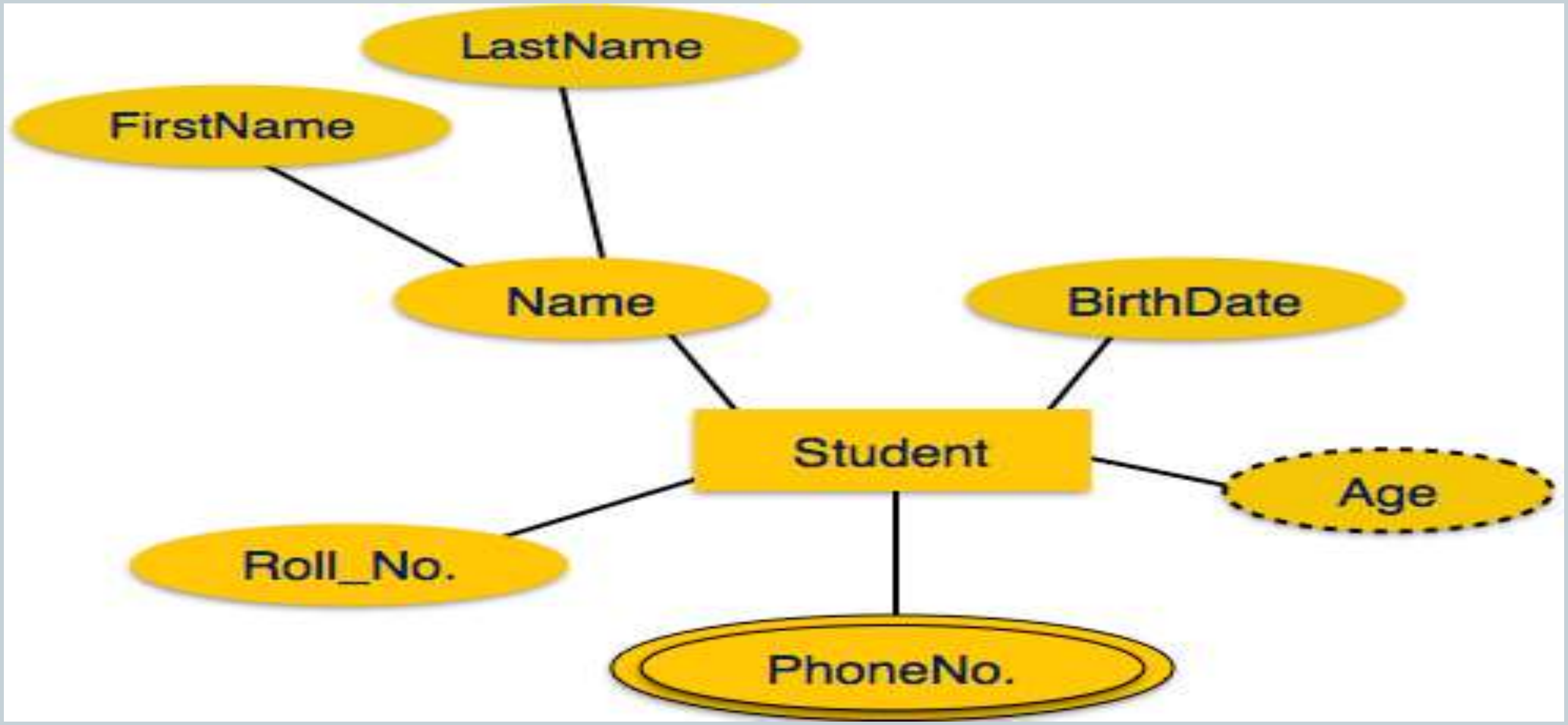
- Attributes are the properties of entities. Attributes are represented by means of ellipses.
- Every ellipse represents one attribute and is directly connected to its entity (rectangle).



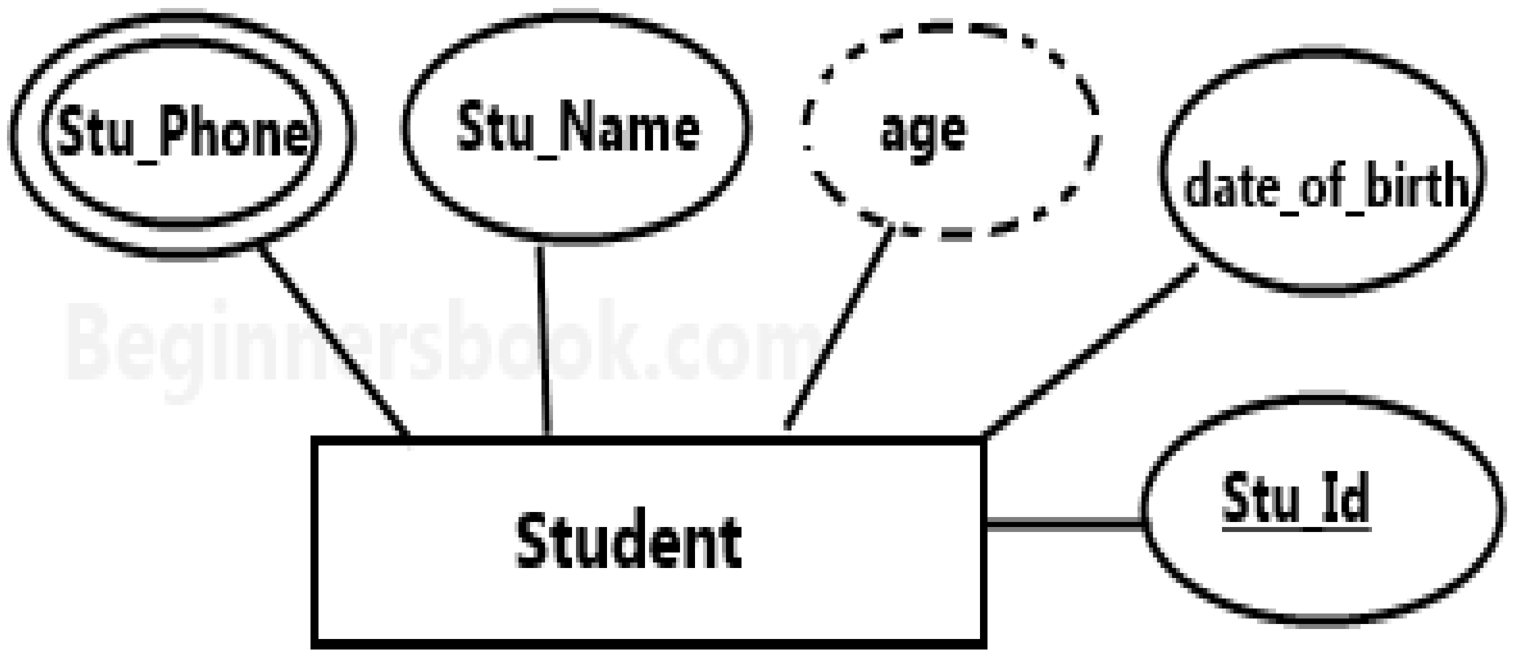
- If the attributes are **composite**, they are further divided in a tree like structure. Every node is then connected to its attribute.
- That is, composite attributes are represented by ellipses that are connected with an ellipse.



- **Derived** attributes are depicted by dashed ellipse.
- **Multivalued** attributes are depicted by double ellipse.



Different Attributes

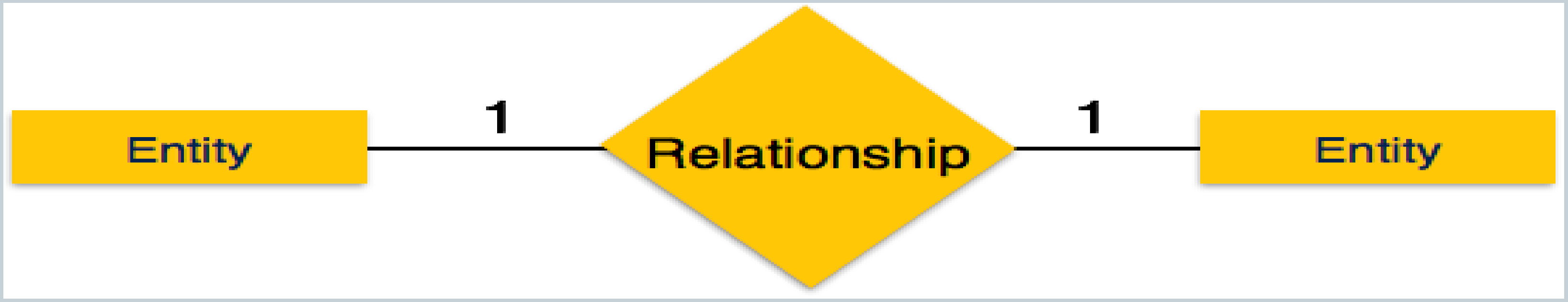


Relationship

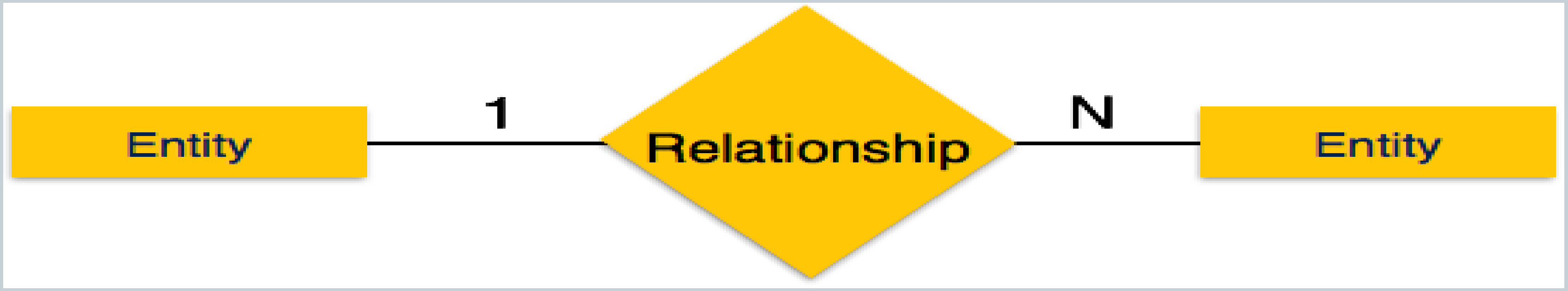
- Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box.
- All the entities (rectangles) participating in a relationship, are connected to it by a line.

Types Of Relationship

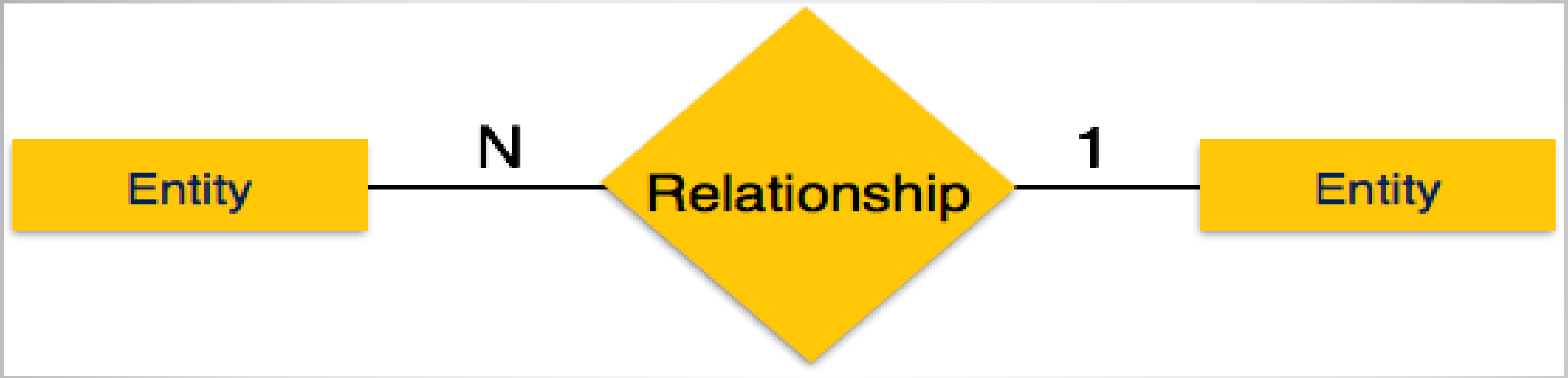
One to One



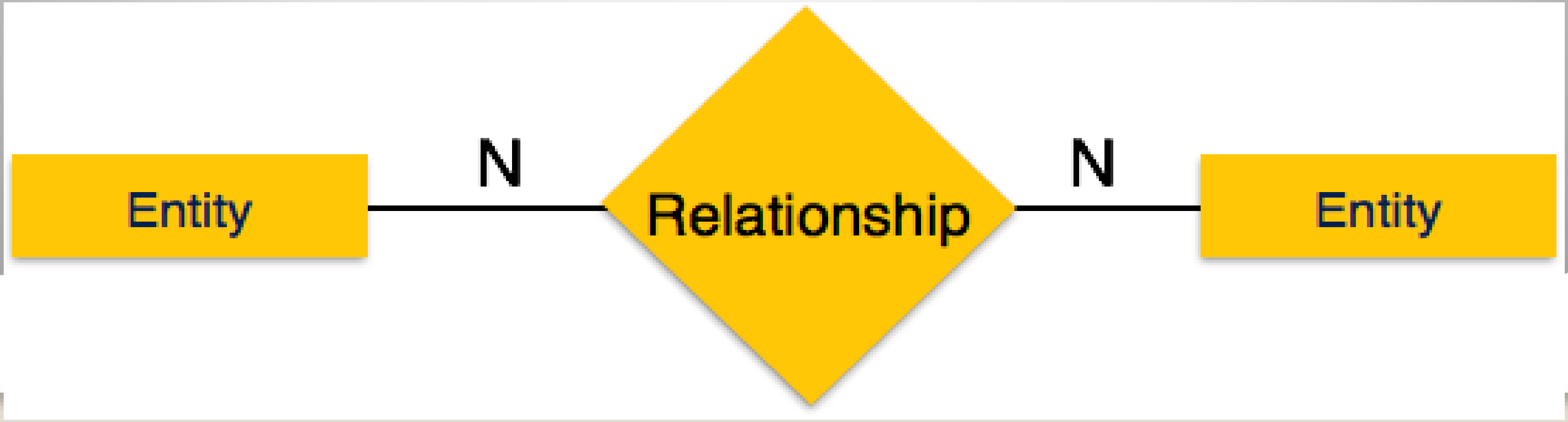
One to Many



Many to One



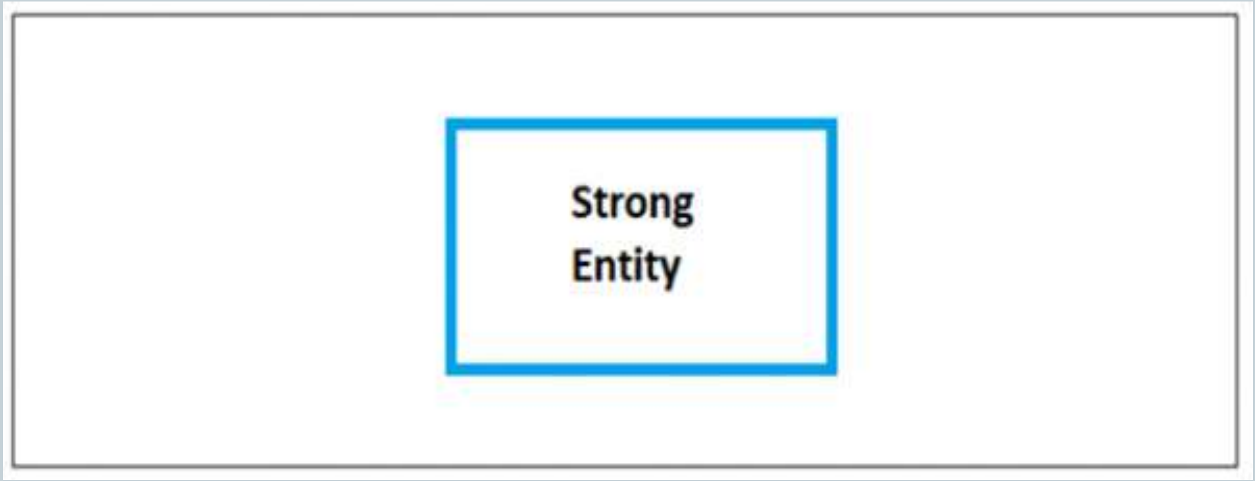
Many to Many



Strong and Weak Entity Types

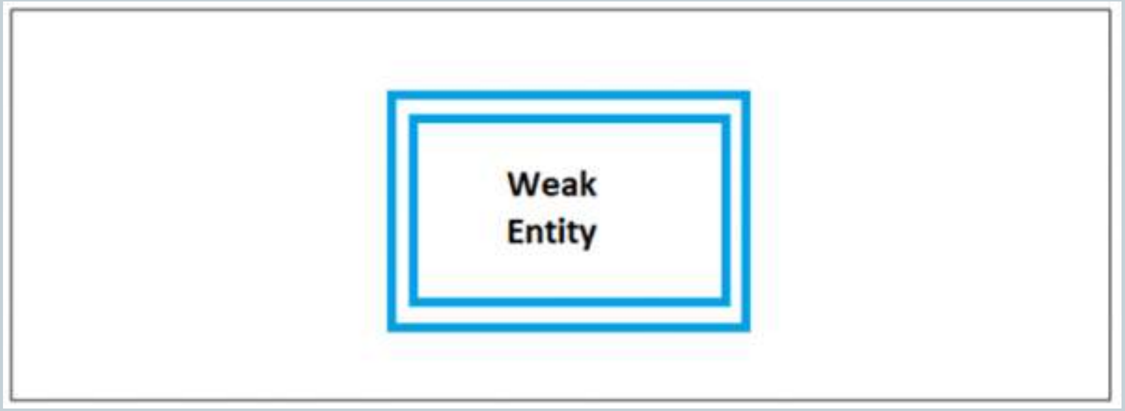
1. Strong Entity

- The strong entity has a primary key.
- Strong Entity is represented by a single rectangle
- In Professor table we have **Professor_Name, Professor _ID and Professor _Salary** are attributes so **Professor_ID** is the primary key



2. Weak Entity

- The weak entity in DBMS do not have a primary key and are dependent on the parent entity. It mainly depends on other entities.
- Weak Entity is represented by double rectangle.



Continuing our previous example, **Professor** is a strong entity, and the primary key is **Professor_ID**. However, another entity is **Professor_Dependents**, which is our Weak Entity.

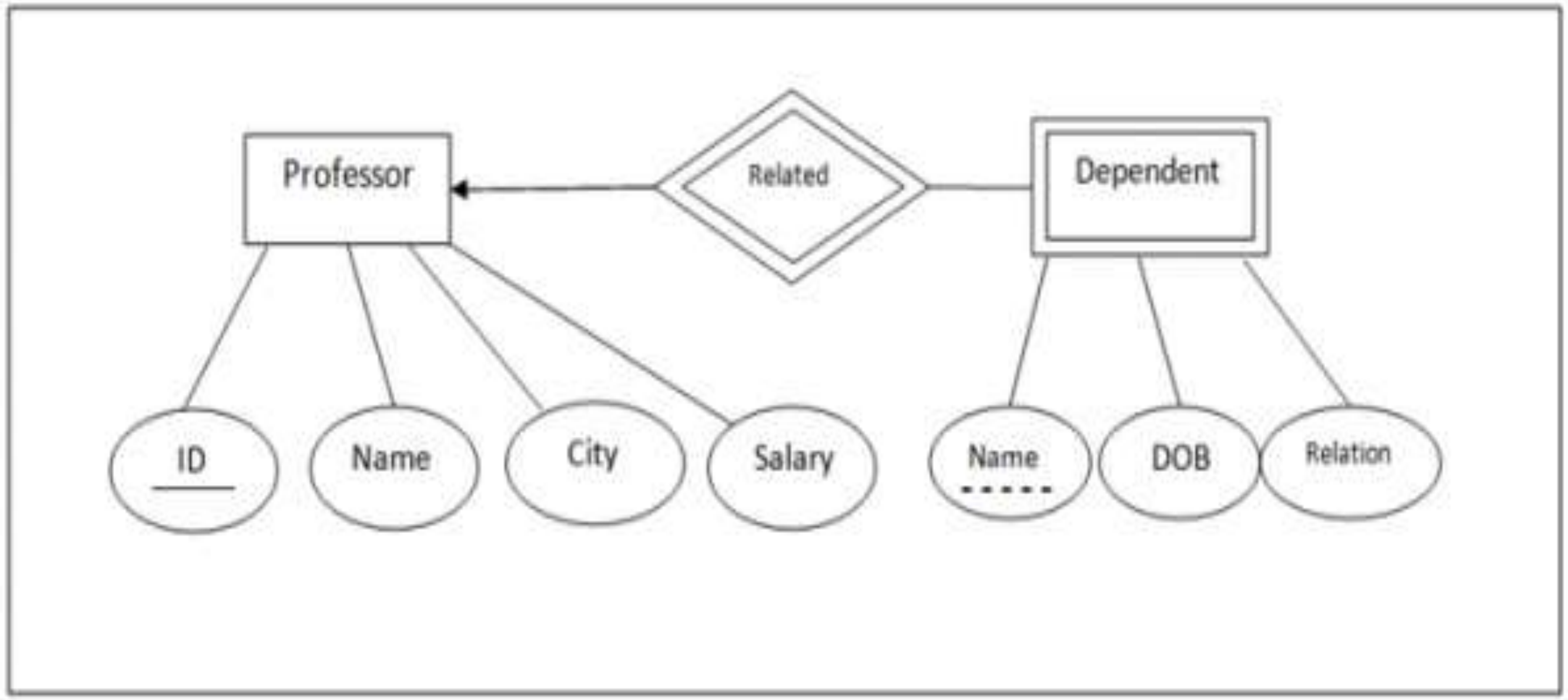
Table - Professor_Dependents

Name	DOB	Relation

This is a weak entity since its existence is dependent on another entity **Professor**, which we saw above. A Professor has Dependents.

Example of Strong and Weak Entity

The example of strong and weak entity can be understood by the below figure.



The Strong Entity is **Professor**, whereas **Dependent** is a Weak Entity. **ID** is the primary key (represented with a line) and Name in **Dependent** entity is called **Partial Key** (represented with a dotted line).

SQL

- **SQL** is a database computer language designed for the retrieval and management of data in a relational database.
- **SQL** stands for **Structured Query Language**.
- SQL is the standard language for Relational Database System.
- All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

Applications of SQL

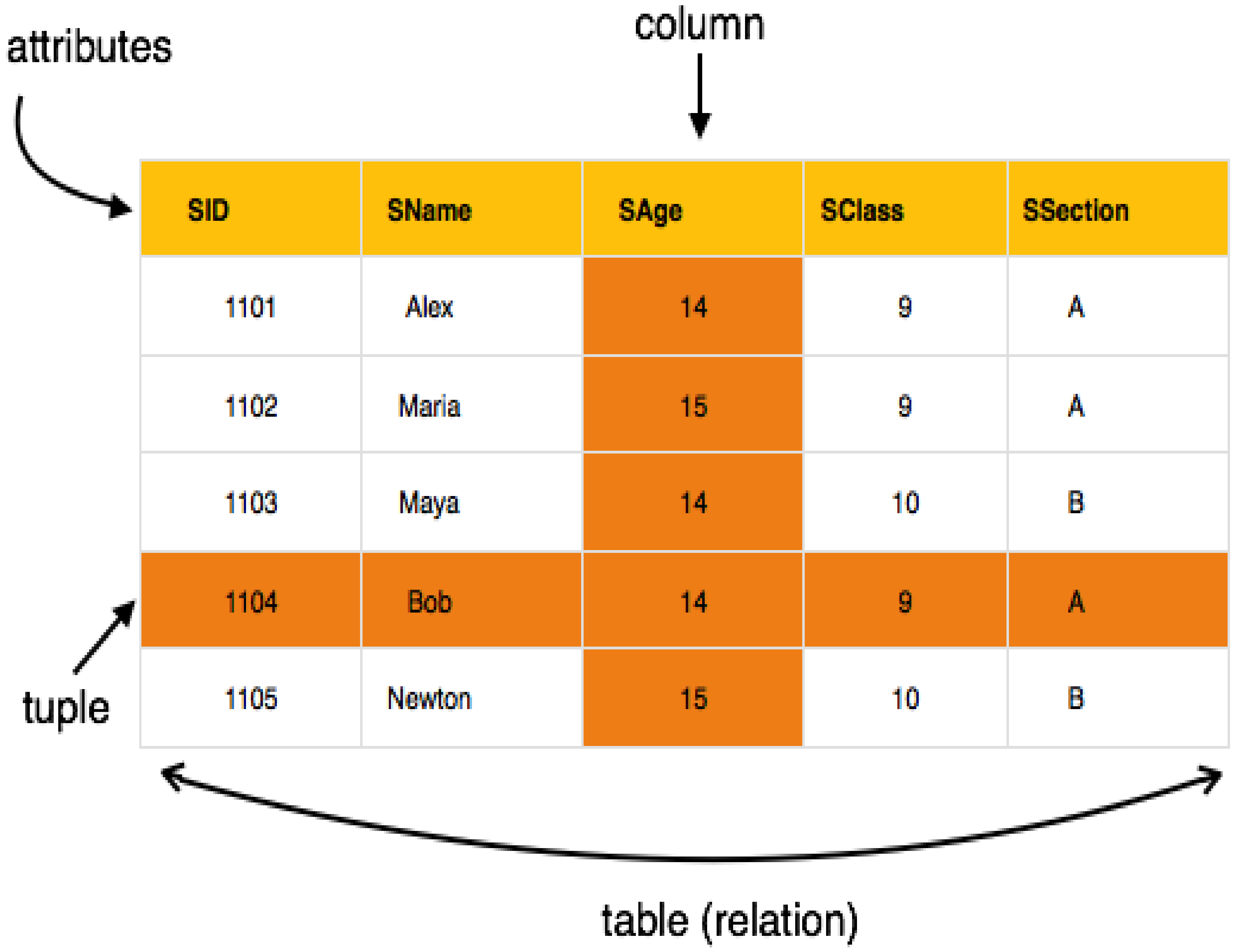
- Allows users to access data in the relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

RDBMS

- A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

Some important terms:

1. **Table -** The data in an RDBMS is stored in database objects which are called as **tables**. This table is basically a collection of related data entries and it is a collection of columns and rows.



SID	SName	SAge	SClass	SSection
1101	Alex	14	9	A
1102	Maria	15	9	A
1103	Maya	14	10	B
1104	Bob	14	9	A
1105	Newton	15	10	B

2. Row/ Tuple

- A record is also called as a row of data is each individual entry that exists in a table.
- A record is a horizontal entity in a table.

3. Column/Attribute

- A column is a vertical entity in a table that contains all information associated with a specific field in a table.

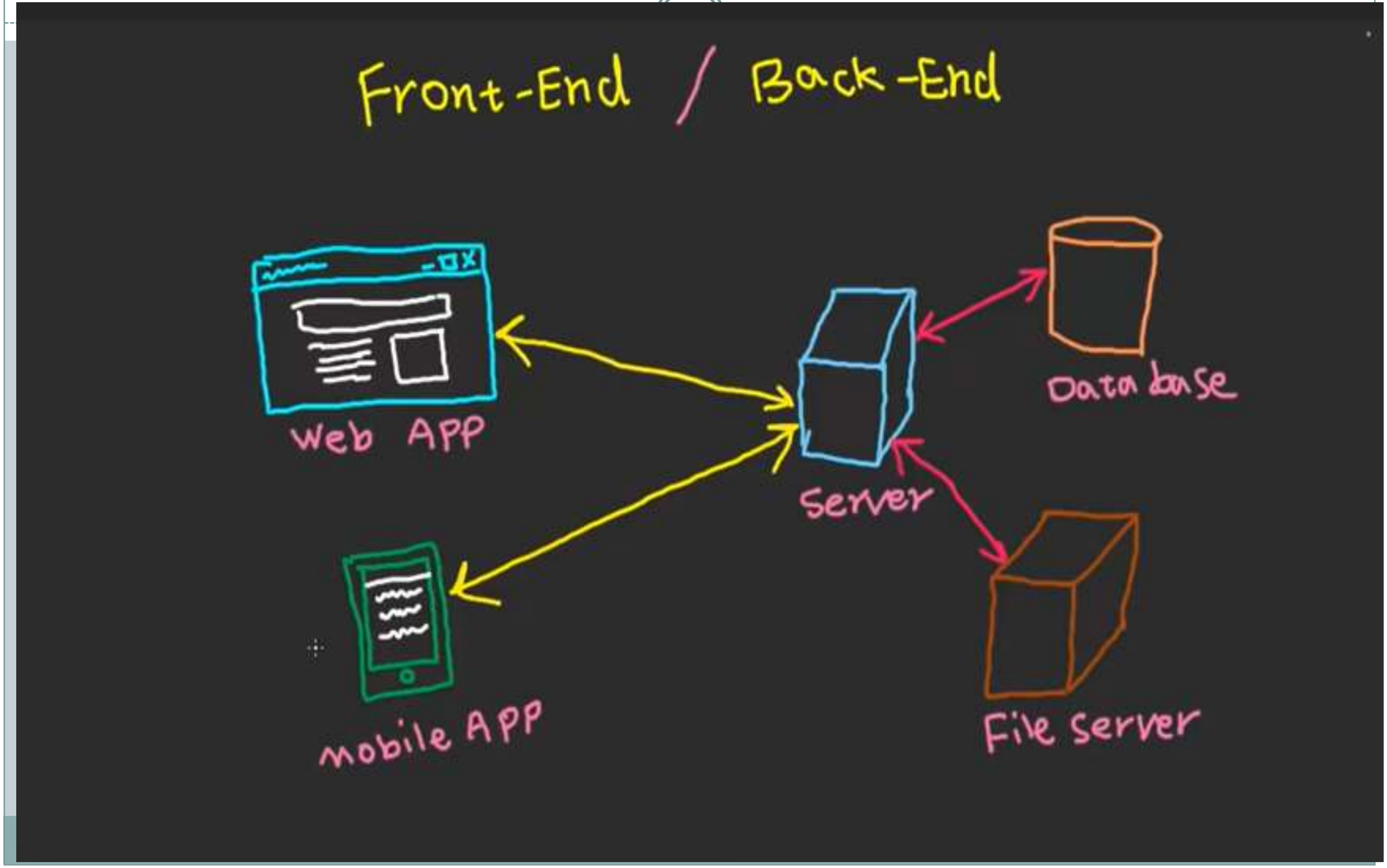
4. **Cardinality**

- The number of rows in a table is called cardinality of table.

Example of DBMS software

- SQL Server 8 , 10 , 12, 16 (By Microsoft)
- Oracle 9 ,11, 12 etc. (By Oracle)
- My SQL (By Oracle)
- DB2 (By IBM)

Front End vs Back End



SQL Data Types

- The data type of a column defines what value the column can hold: integer, character, date and time and so on.

1. **Char(size)** – Character data type of defined size.

   example- name char(15)

2. **Varchar(size)** – size is variable and of character type.

   example- fname varchar(15)

* Maximum we can store 15 bytes i.e. 15 characters, but if we store less character then it changes to less size.

3. Integer/int (size)

- Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295.

4. Nchar(size)

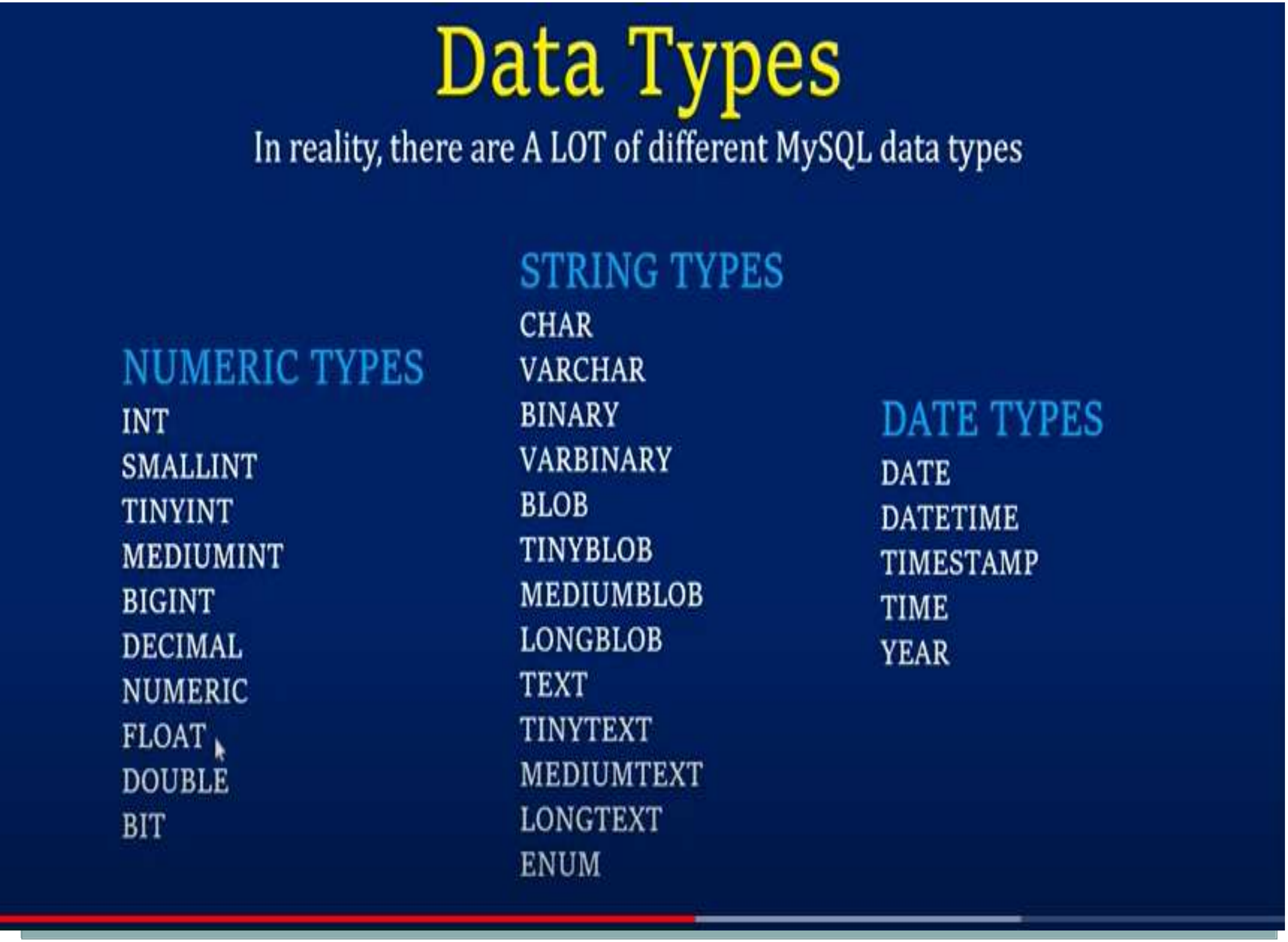
- Here 1 char = 2 bytes, so Nchar(15) = 30 bytes

5. Nvarchar(size)

- Here also 1 character= 2 bytes
- Variable size.

6. **Date**

- Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'



Data Types

In reality, there are A LOT of different MySQL data types

NUMERIC TYPES

INT
SMALLINT
TINYINT
MEDIUMINT
BIGINT
DECIMAL
NUMERIC
FLOAT
DOUBLE
BIT

STRING TYPES

CHAR
VARCHAR
BINARY
VARBINARY
BLOB
TINYBLOB
MEDIUMBLOB
LONGBLOB
TEXT
TINYTEXT
MEDIUMTEXT
LONGTEXT
ENUM

DATE TYPES

DATE
DATETIME
TIMESTAMP
TIME
YEAR

- **Database contains multiple tables.**
- **Table contains multiple records.**
- **Records are stored in the table in the form of rows and column.**

Basic Commands in Mysql

1. To create database – create <dbname>;
2. To Select any database – use <dbname>;
3. To view available databases – show databases;
4. To view tables inside any databases – show tables;
5. To structure/schema of any table – desc <tablename>;
   describe <tablename>;

Create table Command in SQL

- Syntax

```
create  table <table name>
(
  attribute1 data type constraint,
  attribute1 data type constraint,
   :
   :
);
```

Sample code of Create Command

```
Create table student
(
    sid char(5) primary key,                -attribute level
    sname varchar(15),
    fname varchar(15),
    marks integer(3)
   //primary key(sid)                         -table level
);
```

Table name - Student

Sid	Sname	Fname	Marks

Insert Command

- The INSERT INTO statement is used to insert new records in a table.

<u>Insert Into Syntax</u>

- It is possible to write the INSERT INTO statement in two ways.

1. insert into table_name (column1, column2, column3, ...)
   values (value1, value2, value3, ...);
2. insert into table_name
   values (value1, value2, value3, ...);

Example

1. insert into student (sid, sname, scity, smarks) values (‘s101’, ‘akash’, ‘raipur’, ‘95’);
2. insert into student
   values (‘s102’, ‘amit', ‘raipur’, ’95’);
3. insert into student (sid, sname, smarks) values (‘s103’, ‘rahul’, ’95’);

SQL SELECT Statement

- The SELECT statement is used to select data from a database.
- The data returned is stored in a result table, called the result-set.

Syntax

- SELECT *column1*, *column2*, ... FROM <*table_name*>*;*

If you want to select all the fields available in the table, use the following syntax:

- SELECT * FROM *table_name;*

Example

- Select * from student;
- Select Sid , Sname from student;

SQL SELECT DISTINCT Statement

- The **select distinct** statement is used to return only distinct (different) values.
- Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT Syntax

- select distinct *column1*, *column2*, ...
  from *table_name*;

The SQL WHERE Clause

- The WHERE clause is used to filter records.
- The WHERE clause is used to extract only those records that fulfill a specified condition.

<u>WHERE Syntax</u>

- select *column1*, *column2*, ... from *table_name* where *condition*;

Example

- Select * from student;
- Select Sid , Sname from student;
- Select * from student where marks >70;
- Select * from student where Sname= 'Rahul';
- Select Sname, marks from student where marks<40;

Sequence

Select <attribute list>

From <tablename>

[Where <condition>];

Operators in The WHERE Clause

Operator	**Description**
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
< >	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Like Operator

Two Symbols:

1. % = Any number of characters
2. _ = Single number of character

Q1 – Find out records of those students whose name starts with letter ‘A’.

Answers- select * from student where name like ‘A%’

2. Find out records of those students whose name ends with letter ‘A’.

Answers- select * from student where name like ‘ %A’;

Raman, Karan, Aman, Ananaya

- 3. Find out records of those students whose second letter is A.

Answer: select * from student where name like ‘_A%’;

4. Find out records of those students whose first letter is A and last letter is I .

Answers- select * from student where name like ‘A%I’;

5. Find out records of those students whose name is having atleast two times A .

Answer: select * from student where name like ‘%A%A%’;

Rahul, karan, vamsi, mohan,

6. Find out records of those students whose name length is 5.

Answer: select * from students where name like '_____'

IN OPERATOR

- In operator is used to find out selected values.

Ques: Find out the records of those student whose marks are either 90 or 80 or 10;

Query : select * from student where marks IN(90,80,10);

BETWEEN

- It is used for any range.

Question: Find out the records of those students whose marks are between 60 to 90;

Query : select * from student where marks between 60 and 90;

Logical Expression

- AND
- OR
- NOT

Example

- Select * from student where marks >40 AND marks <80;
- Select * from student where marks 40 or marks <80;
- Select * from student where not marks=50;

SQL ORDER BY Keyword

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.
- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.
- select *column1*, *column2*, ...
  from *table_name*
  order by *column1*, *column2*, ... asc|desc;

ORDER BY Example

- The following SQL statement selects all customers from the "Customers" table, sorted by the "Country" column:
- select * from customers
  order by country;

ORDER BY Several Columns Example

- select * from customers
  order by country, customername;
- select * from customers
  order by country asc, customername desc;

SQL NULL Values

- A field with a NULL value is a field with no value.
- A NULL value is different from a zero value or a field that contains spaces.
- It is not possible to test for NULL values with comparison operators, such as =, <, or <>.
- We will have to use the IS NULL and IS NOT NULL operators instead.

IS NULL Syntax

select *column_names*
from *table_name*
where *column_name* is
null;

IS NOT NULL Syntax

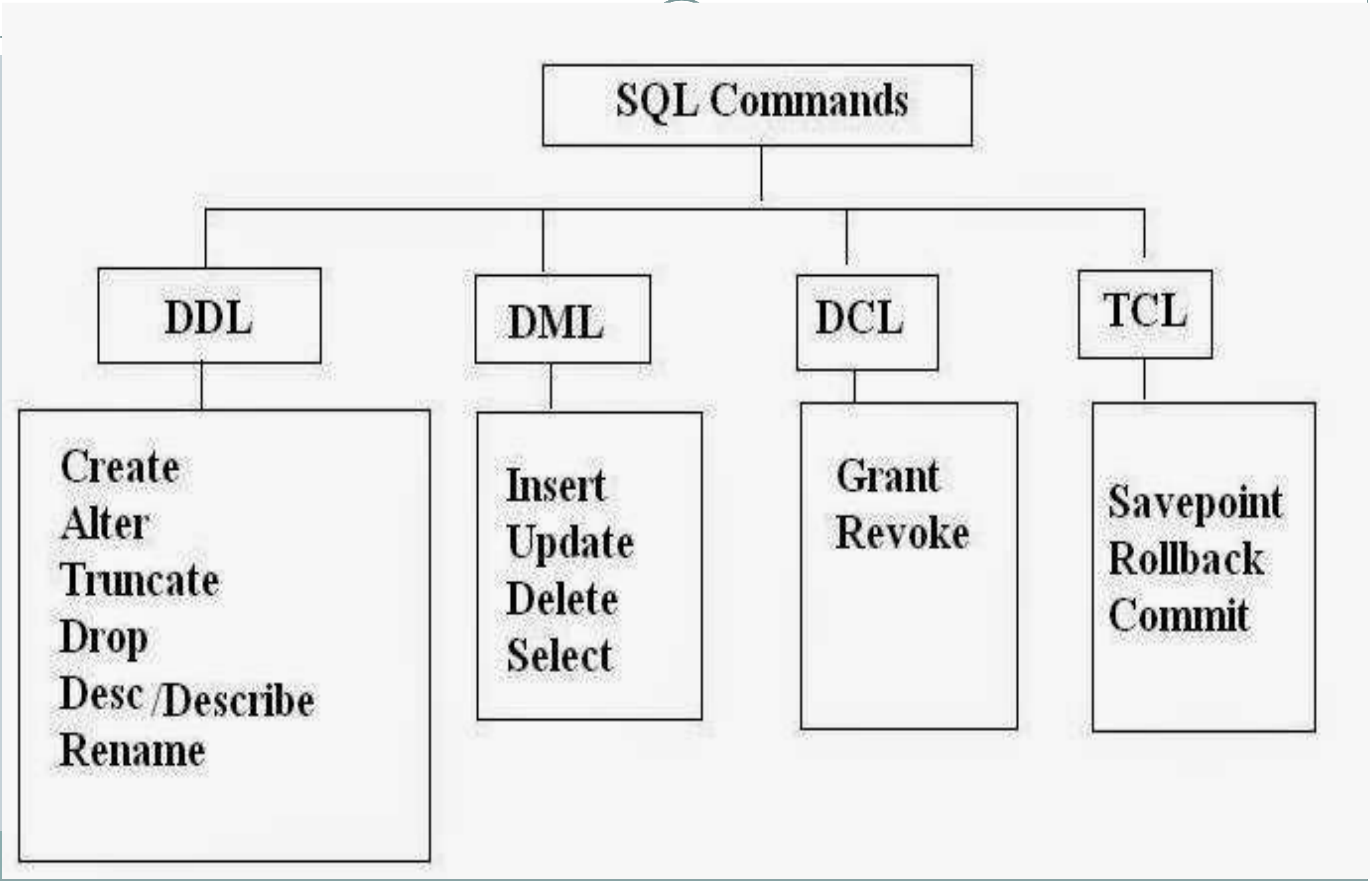
select *column_names*
from *table_name*
where *column_name* is
not null;

Types of SQL Commands

- These SQL commands are mainly categorized into four categories as:

1. DDL – Data Definition Language
2. DML – Data Manipulation Language
3. DCL – Data Control Language
4. TCL - Transaction Control Language

Types of SQL Commands



The SQL UPDATE Statement

- The UPDATE statement is used to modify the existing records in a table.

UPDATE Syntax

- UPDATE *table_name*
  SET *column1* = *value1*, *column2* = *value2*, ...
  WHERE *condition*;

SQL DELETE Statement

- The DELETE statement is used to delete existing records in a table.

<u>DELETE Syntax</u>

- DELETE FROM *table_name* WHERE *condition;*

<u>Example -</u>

- DELETE FROM S3 WHERE sid = 3;

SQL DROP DATABASE Statement

- The DROP DATABASE statement is used to drop an existing SQL database.

Syntax

DROP DATABASE *database_name*;

SQL DROP TABLE Statement

- The DROP TABLE statement is used to drop an existing table in a database.

Syntax

```
DROP TABLE table_name;
```

SQL TRUNCATE TABLE

- The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

Syntax

- TRUNCATE TABLE *table_name*;
- TRUNCATE *table_name*;

SQL ALTER TABLE Statement

- The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

- The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

- To add a column in a table, use the following syntax:

ALTER TABLE *table_name*
ADD <column> *column_name datatype*;

Example - The following SQL adds an "Email" column to the "Customers" table:

- ALTER TABLE Customers
  ADD <column> Email varchar(255);

ALTER TABLE - DROP COLUMN

- To delete a column in a table, use the following syntax.

ALTER TABLE *table_name*
DROP <COLUMN> *column_name*;

Example - The following SQL deletes the "Email" column from the "Customers" table:

- ALTER TABLE Customers
  DROP <COLUMN> Email;

ALTER TABLE - ALTER/MODIFY COLUMN

- To change the data type of a column in a table, use the following syntax:

ALTER TABLE *table_name*
MODIFY <COLUMN> *column_name datatype*;

SQL Constraints

- SQL constraints are used to specify rules for data in a table.
- Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

Syntax

```
  CREATE TABLE table_name (
column1 datatype constraint,
column2 datatype constraint,
column3 datatype constraint,
  ....
  );
```

SQL Constraints(cont.)

- Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

- Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

The following constraints are commonly used in SQL:

1) **NOT NULL** - Ensures that a column cannot have a NULL value
2) **UNIQUE** - Ensures that all values in a column are different
3) **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
4) **CHECK** - Ensures that all values in a column satisfies a specific condition
5) **DEFAULT** - Sets a default value for a column when no value is specified

SQL NOT NULL Constraint

- By default, a column can hold NULL values.
- The NOT NULL constraint enforces a column to NOT accept NULL values.
- This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

Syntax

1. CREATE TABLE Persons (ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255) NOT NULL,
   Age int);
2. ALTER TABLE Persons
   MODIFY Age int NOT NULL;

SQL UNIQUE Constraint

- The UNIQUE constraint ensures that all values in a column are different.
- Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint.
- However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

Syntax

- CREATE TABLE Persons (
  ```
  ID int NOT NULL,
  LastName varchar(255) NOT NULL,
  FirstName varchar(255),
  Age int,
  UNIQUE (ID) );
  ```
- ALTER TABLE Persons
  ADD UNIQUE (ID);

SQL PRIMARY KEY Constraint

- The PRIMARY KEY constraint uniquely identifies each record in a table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.

SQL PRIMARY KEY on CREATE TABLE

-
  ```
  CREATE TABLE Persons (
      ID int NOT NULL,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255),
      Age int,
      PRIMARY KEY (ID) );
  ```

-
  ```
  ALTER TABLE Persons
  ADD PRIMARY KEY (ID);
  ```

-
  ```
  ALTER TABLE Persons
  DROP PRIMARY KEY;
  ```

SQL CHECK Constraint

- The CHECK constraint is used to limit the value range that can be placed in a column.

Example

-
  ```
  CREATE TABLE Persons (
      ID int NOT NULL,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255),
      Age int,
      CHECK (Age>=18) );
  ```

-
  ```
  ALTER TABLE Persons
  ADD CHECK (Age>=18);
  ```

SQL DEFAULT Constraint

- The DEFAULT constraint is used to provide a default value for a column.
- The default value will be added to all new records IF no other value is specified.

Example

-
  ```
  CREATE TABLE Persons (
      ID int NOT NULL,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255),
      Age int,
      City varchar(255) DEFAULT 'Sandnes');
  ```

-
  ```
  ALTER TABLE Persons
  ALTER City SET DEFAULT 'Sandnes';
  ```

SQL AUTO INCREMENT Field

- Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Syntax

```
CREATE TABLE Persons (
Personid int NOT NULL AUTO_INCREMENT,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int,
PRIMARY KEY (Personid));
```

AUTO INCREMENT

- Often this is the primary key field that we would like to be created automatically every time a new record is inserted.
- By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.
- To let the AUTO_INCREMENT sequence start with another value, use the following SQL statement:
- ALTER TABLE Persons AUTO_INCREMENT=100;

SQL LIKE Operator

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator:
- % - The percent sign represents zero, one, or multiple characters
- _ - The underscore represents a single character

LIKE Syntax

- SELECT *column1*, *column2*, ...
  FROM *table_name*
  WHERE *column* LIKE *pattern*;

Examples:

1. Selects all customers with a CustomerName starting with "a":

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

2. Selects all customers with a CustomerName ending with "a":

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%a';
```

3. Selects all customers with a CustomerName that have “ a " in any position:

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%a%';
```

4. Selects all customers with a CustomerName that have "r" in the second position:

```
SELECT * FROM Customers
WHERE CustomerName LIKE '_r%';
```

5. selects all customers with a CustomerName that starts with "a" and are at least 3 characters in length:

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a__%';
```

6. selects all customers with a ContactName that starts with "a" and ends with “h":

```
SELECT * FROM Customers
WHERE ContactName LIKE 'a%o';
```

The SQL BETWEEN Operator

- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.

SELECT *column_name(s)* FROM *table_name*
WHERE *column_name* BETWEEN *value1* AND *value2;*

Example

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

SQL IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.
- The IN operator is a shorthand for multiple OR conditions.
- SELECT *column_name(s)* FROM *table_name* WHERE *column_name* IN (*value1*, *value2*, ...);

Example-

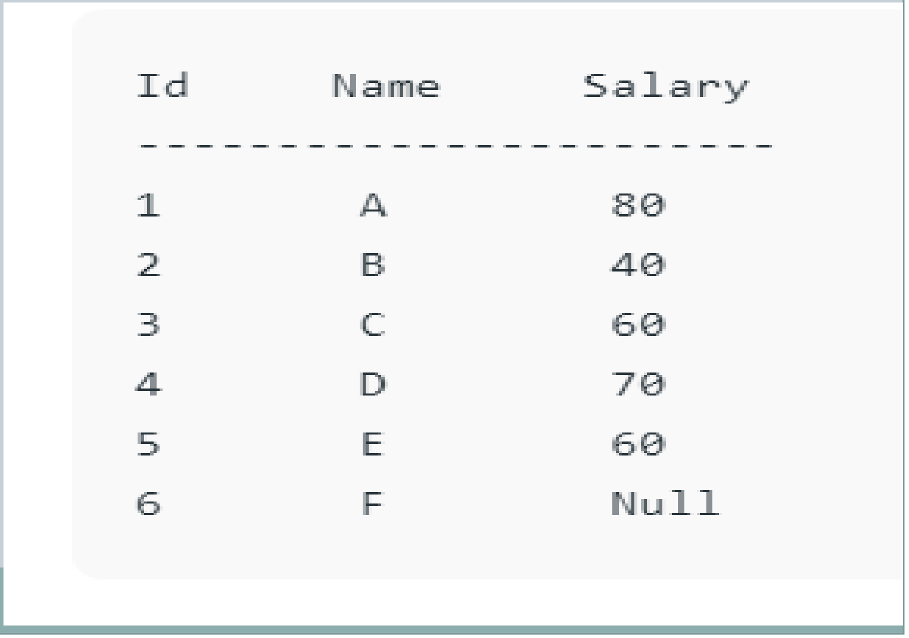
```
SELECT * FROM Students
WHERE  Marks IN (80,60,50);
```

Aggregate functions in SQL

- In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

Various Aggregate Functions

1. Min()
2. Max()
3. Count()
4. Sum()
5. Avg()



```
Id      Name       Salary
--------------------------
1         A          80
2         B          40
3         C          60
4         D          70
5         E          60
6         F          Null
```

The SQL MIN() and MAX() Functions

- The MIN() function returns the smallest value of the selected column.

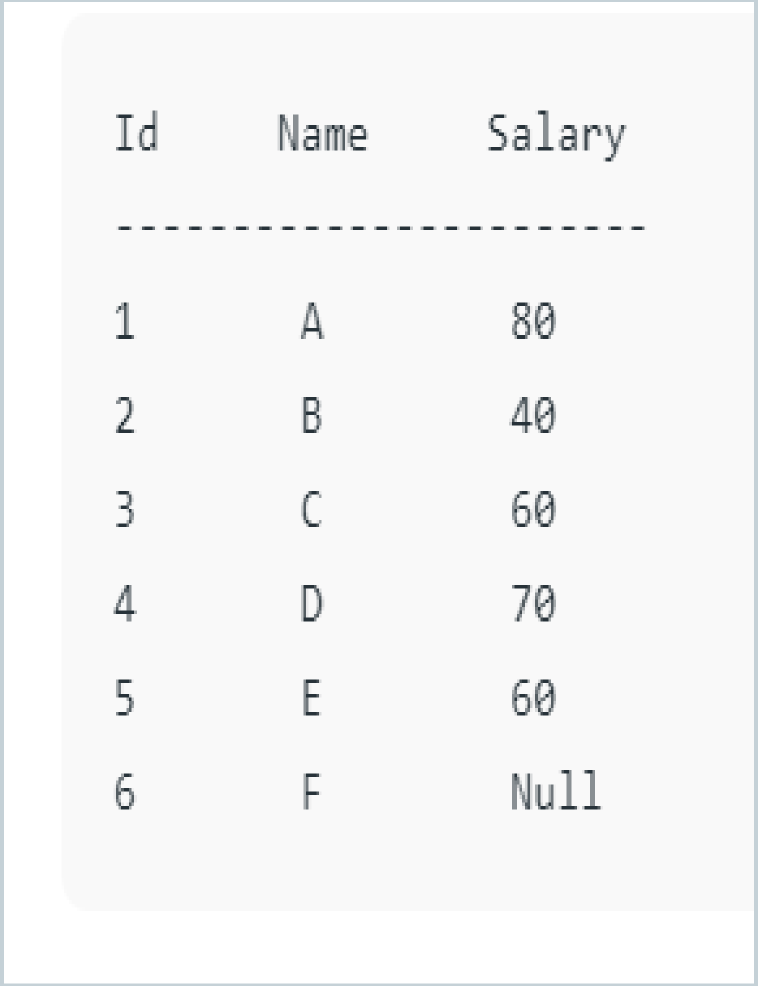
```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

- The MAX() function returns the largest value of the selected column.

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

Example of Min() & Max()

- ***Min(salary):*** Minimum value in the salary column except NULL i.e., 40.
- ***Max(salary):*** Maximum value in the salary i.e., 80.



```
Id      Name      Salary
------------------------
1        A         80
2        B         40
3        C         60
4        D         70
5        E         60
6        F         Null
```

The SQL COUNT( ) Functions

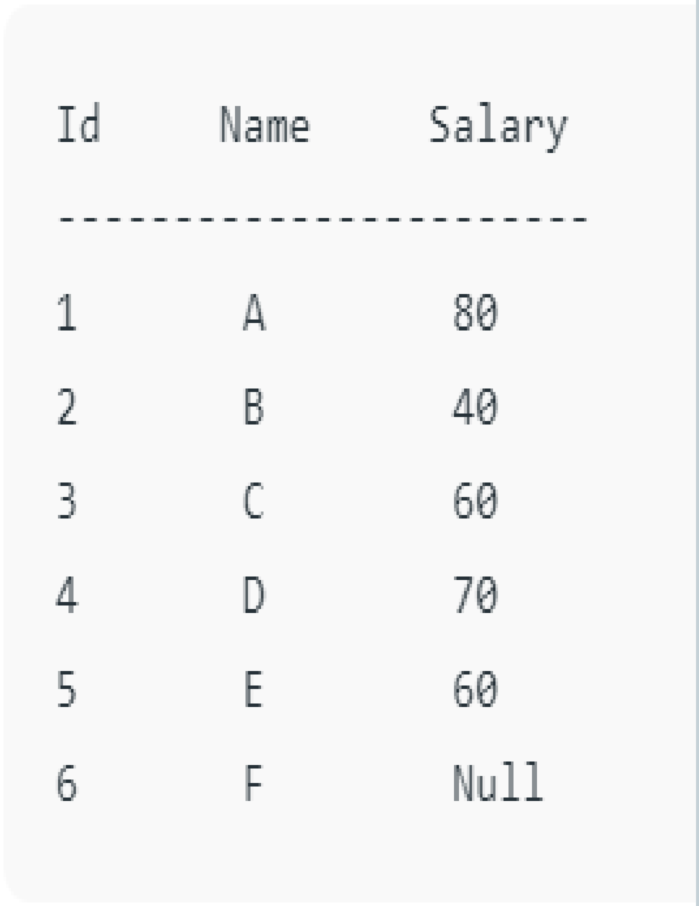
- The COUNT() function returns the number of rows that matches a specified criterion.

  SELECT COUNT(*column_name*)
  FROM *table_name*
  WHERE *condition*;

- ***Count(*):*** Returns total number of records .
- ***Count(column_name):*** Return number of Non Null values over that column.
- ***Count(Distinct column_name):*** Return number of distinct Non Null values over that column.

Example of count

- ***Count(*):*** Returns total number of records .i.e 6.
- ***Count(salary):*** Return number of Non Null values over the column salary. i.e 5.
- ***Count(Distinct Salary):*** Return number of distinct Non Null values over the column salary .i.e 4



```
Id      Name      Salary
------------------------
1        A         80
2        B         40
3        C         60
4        D         70
5        E         60
6        F         Null
```

The SQL SUM( ) Functions

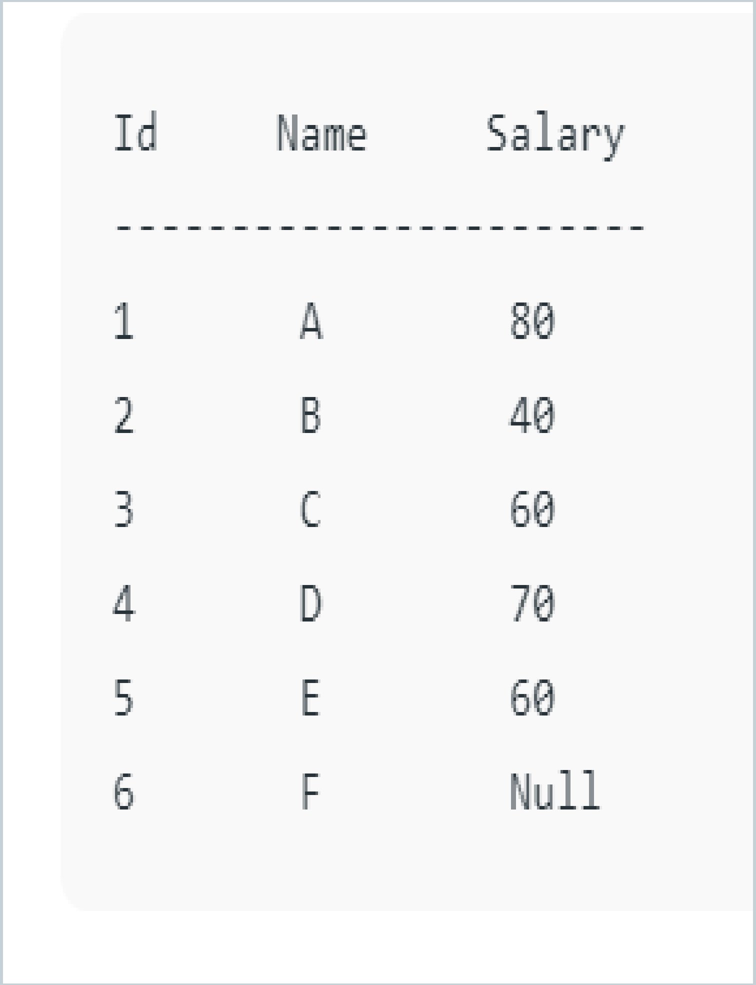
- The SUM() function returns the total sum of a numeric column.

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

- ***Sum(column_name)***
- ***Sum(Distinct column_name)***

Example of Sum():

- ***sum(salary):*** Sum all Non Null values of Column salary i.e., 310.
- ***sum(Distinct salary):*** Sum of all distinct Non-Null values i.e., 250.



Id	Name	Salary
1	A	80
2	B	40
3	C	60
4	D	70
5	E	60
6	F	Null

The SQL AVG( )Functions

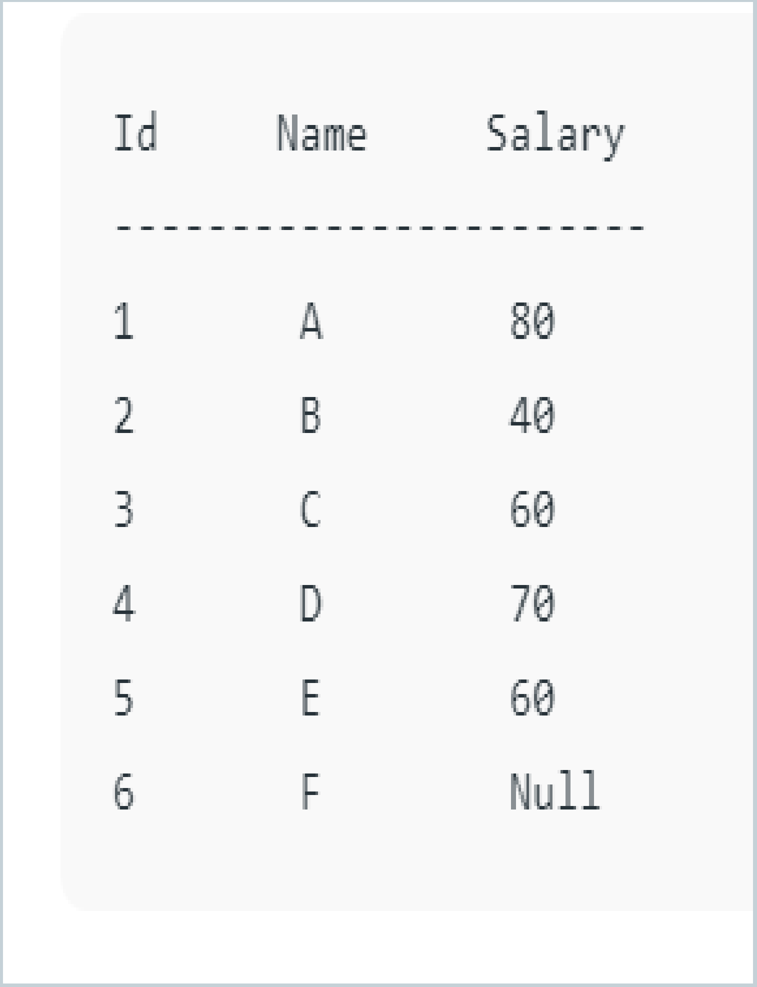
- The AVG() function returns the average value of a numeric column.

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

- ***Avg(column_name)***
- ***Avg(Distinct column_name)***

Example of AVG():

- ***Avg(salary)*** = Sum(salary) / count(salary) = 310/5
- ***Avg(Distinct salary)*** = sum(Distinct salary) / Count(Distinct Salary) = 250/4



Id	Name	Salary
1	A	80
2	B	40
3	C	60
4	D	70
5	E	60
6	F	Null

General Syntax of SQL statements

- SELECT [Distinct] <attribute list>
- FROM <TABLE NAME>
- [WHERE <condition>]
- [GROUP BY (Attribute) [having Condition]]
- [ORDER BY (Attribute) [ASC/Desc]];

The SQL GROUP BY Statement

- The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".
- The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

GROUP BY Syntax

- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *condition*
  GROUP BY *column_name(s)*
  ORDER BY *column_name(s);*

SQL GROUP BY Examples

- The following SQL statement lists the number of customers in each country:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

- The following SQL statement lists the number of customers in each country, sorted high to low:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

SQL HAVING Clause

- The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.
  - HAVING Syntax
- SELECT *column_name(s)*
  FROM *table_name*
  WHERE *condition*
  GROUP BY *column_name(s)*
  HAVING *condition*
  ORDER BY *column_name(s);*

SQL HAVING Examples

- The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

SQL Views

- In SQL, a view is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

<u>Syntax</u>

- CREATE VIEW *view_name* AS
  SELECT *column1*, *column2*, ...
  FROM *table_name*;

Creating a table from another table

Syntax

- Create table <table name>

  As < Select Query>;

Example

Create table chotastudent

as select sid, name, marks from student;

Inserting records from another table

- Syntax
- Insert into <table name> <a select Query>;

Example-

1. Insert into chotastudent
   select sid,name,marks from student;
2. Insert into chotastudent
   select sid,name,marks from student
   where marks>35;

Join

- A join clause is used to fetch data from two or more tables, based on join condition.
- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

STUDENT

SID	Sname	Marks	Dept No
s1	A	30	D1
S2	B	40	D2
S3	C	45	D1
S4	D	35	D2

DEPARTMENT

Dno	Dname	Location
D1	CS	RAIPUR
D2	CHEM	DURG
D3	PHY	RAIPUR

Question 1: Retrieve the sid and department no. of students whose marks <40;

Select sid, dept no from student where marks<40;

SID	DEPT NO
S1	D1
S4	D2

Question 2: Retrieve the sid and department no. of students whose marks <40;

Cartesian Product

Student X Department

SID	SNAME	MARKS	DEPTNO	DNO	DNAME	LOCATION
S1	A	30	D1	D1	CS	RAIPUR
S1	A	30	D1	D2	CHEM	DURG
S1	A	30	D1	D3	PHY	RAIPUR
S2	B	40	D2	D1	CS	RAIPUR
S2	B	40	D2	D2	CHEM	DURG

SQL Aliases

- SQL aliases are used to give a table, or a column in a table, a temporary name.
- Aliases are often used to make column names more readable.

Aliases can be useful when:

- There are more than one table involved in a query.
- Functions are used in the query.
- Column names are big or not very readable.
- Two or more columns are combined together.

Alias Column Syntax

SELECT *column_name* AS *alias_name*
FROM *table_name;*

Alias for Columns Examples

- SELECT Sname AS sn, marks AS m
  FROM student;

Alias Table Syntax

SELECT *column_name(s)*
FROM *table_name* AS *alias_name;*

Alias for Tables Example

- SELECT s.sname, s.marks, d.dname
  FROM Student AS s, Department AS d
  WHERE s.sname=‘A' AND s.deptno=d.dno;

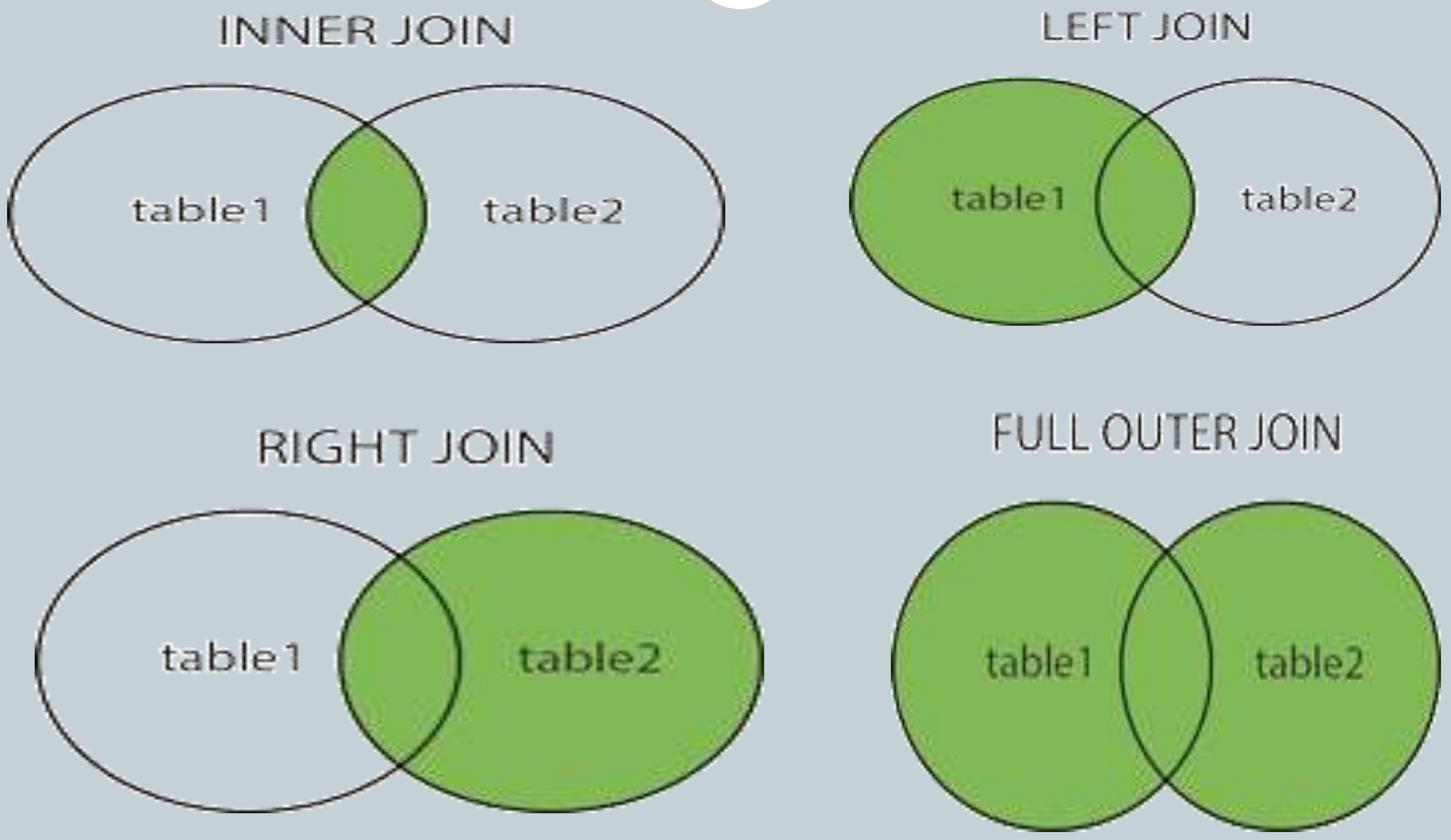
- SELECT Student.sname, Student.marks, department.dname
  FROM Student,department
  WHERE Student.sname =‘Ram' AND
  student.deptno=department.dno;

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

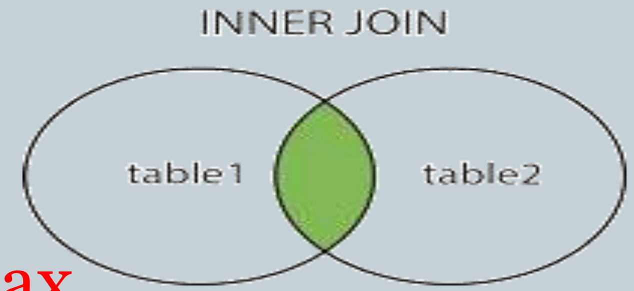
1. **(INNER) JOIN**: Returns records that have matching values in both tables.
2. **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table.
3. **RIGHT (OUTER) JOIN**: Returns all records from the right table, and the matched records from the left table.
4. **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table.

Different Types of SQL JOINs



SQL INNER JOIN Keyword

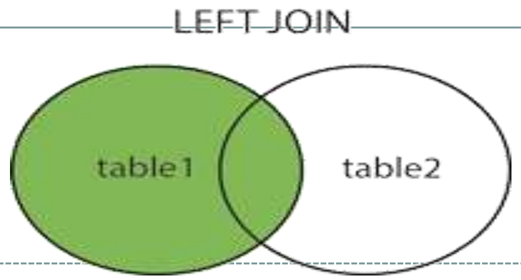
- The INNER JOIN keyword selects records that have matching values in both tables.



INNER JOIN Syntax

- SELECT *column_name(s)*
  FROM *table1*
  INNER JOIN *table2*
  ON *table1.column_name* = *table2.column_name;*

SQL LEFT JOIN Keyword

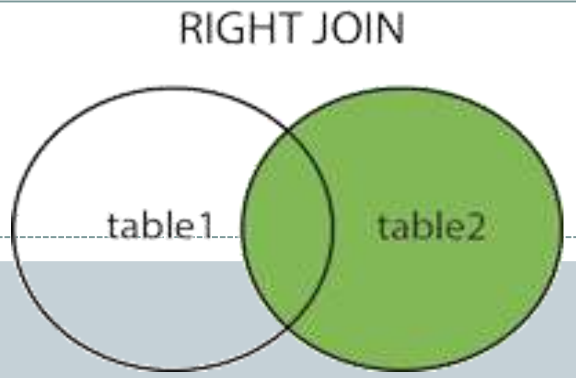


- The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

LEFT JOIN Syntax

- SELECT *column_name(s)*
  FROM *table1*
  LEFT JOIN *table2*
  ON *table1.column_name = table2.column_name;*

SQL RIGHT JOIN Keyword

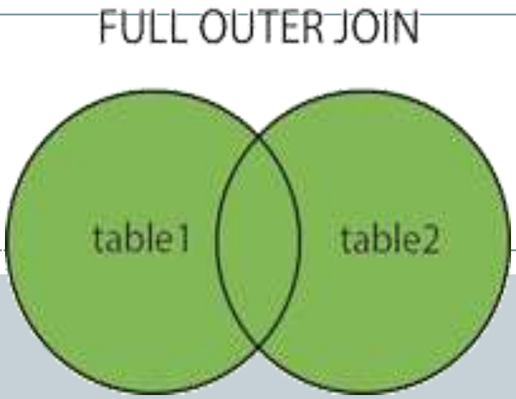


- The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

RIGHT JOIN Syntax

- SELECT *column_name(s)*
  FROM *table1*
  RIGHT JOIN *table2*
  ON *table1.column_name = table2.column_name;*

SQL FULL OUTER JOIN Keyword



- The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.
- FULL OUTER JOIN and FULL JOIN are the same.

FULL OUTER JOIN Syntax

- SELECT *column_name(s)*
  FROM *table1*
  FULL OUTER JOIN *table2*
  ON *table1.column_name = table2.column_name*
  WHERE *condition;*

SQL Self JOIN

- A self JOIN is a regular join, but the table is joined with itself.

Self JOIN Syntax

- SELECT *column_name(s)*
  FROM *table1 T1, table1 T2*
  WHERE *condition*;
- *T1* and *T2* are different table aliases for the same table.

SQL FOREIGN KEY Constraint

- A FOREIGN KEY is a key used to link two tables together.
- A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
- The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Look at the following two tables:

- "Persons" table:

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

"Orders" table:

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

- Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.
- The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.
- The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.
- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

Functional dependency

- In a relation R, let these be two attributes X and Y, the we can say X->Y (X determines Y or Y functionally determined by X) exists for any two tuples t1 and t2 only if they satisfy following conditions.
- If t1. X = t2. X then

  t1. Y = t2. Y

	X	Y
t1	X1	Y1
t2	X1	Y1

	X	Y
t1	X1	Y1
t2	X1	Y2

TABLE 1 Is Functionally Dependent but table 2 is not Functionally Dependent

Find the Functional Dependency in following table

A	B	C
2	3	8
1	1	3
3	3	2
1	1	6
2	3	1
2	3	5
1	1	4

Check Whether

1. A -> B
2. A ->C
3. B -> A
4. C ->A
5. C ->B
6. AB ->C
7. AC ->B
8. BC ->A
9. B -> C
10. C->B
11. A ->BC
12. B->AC
13. C ->AB

Attribute Closure (X^+)

- This is the set of attributes determined by X.
- Let R(A,B,C,D)
- $A^+ = \{ A,B,C,D \}$
- A defines itself.

Trivial Dependency

- Trivial means whose answer is already known to us.

like What is your name Ram?

- Example : A -> A
- AB -> A
- BC - > C

Inference Rules

1. Reflexive Rules

   if X>= Y then X -> y (Trivial)

2. Augmentation Rules –

   if X -> Y exists then XZ -> YZ will also exists.

3. Transitive Rules

   if X -> Y and Y->Z then X -> Z will also exist.

4. Decomposition Rules

   X -> YZ then X -> Y and X -> Z also .

5. Additive Rules

   if X -> Y AND X -> Z THEN X -> YZ