

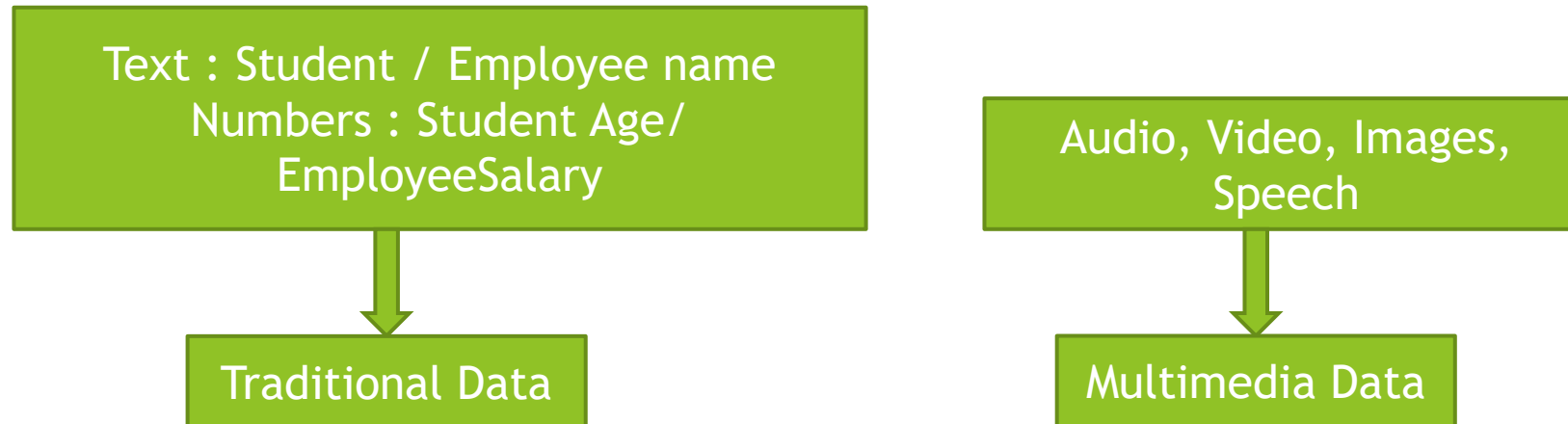
# RDBMS and Structured Query Language

# Agenda

- ▶ Overview of RDBMS
- ▶ Normalization
- ▶ Types of Normal Forms
- ▶ SQL Basics
- ▶ Joins
- ▶ Subqueries or Nested Queries

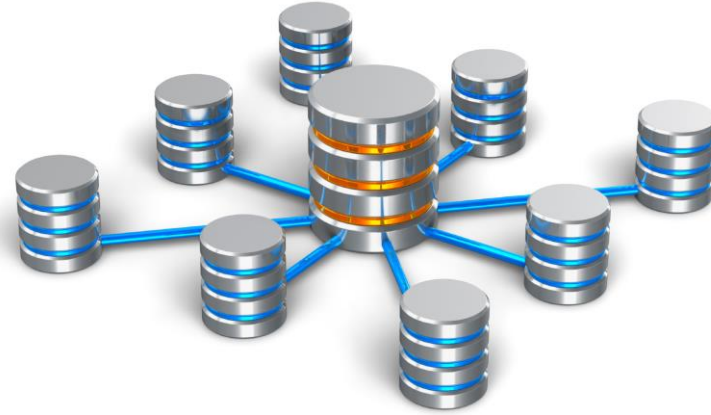
# What is data ?

**Data : Facts that can be recorded.**

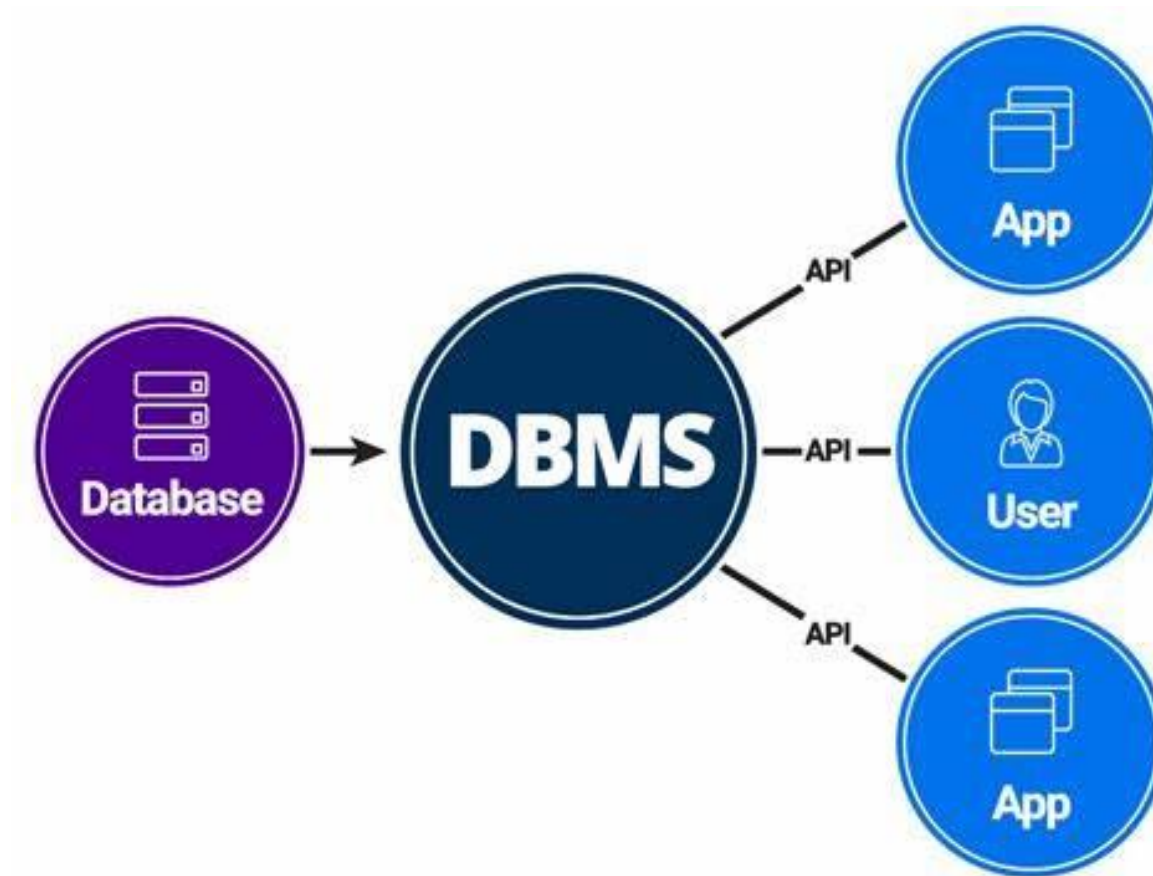


# What is a Database?

- ▶ Database is an organized collection of related information.
- ▶ It has some inherent meaning
- ▶ Represents some real world aspects.
- ▶ Designed, built and populated with data for a specific purpose.



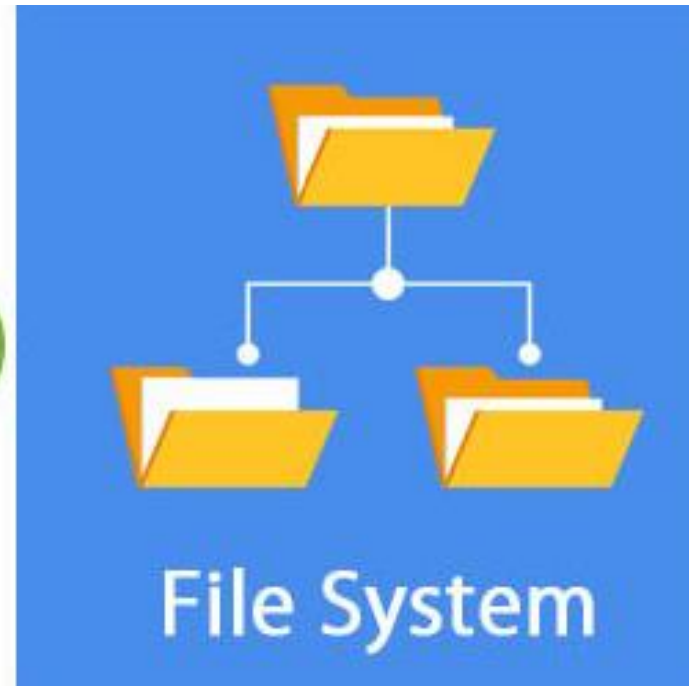
# Database Management System



# File System v/s DBMS



VS

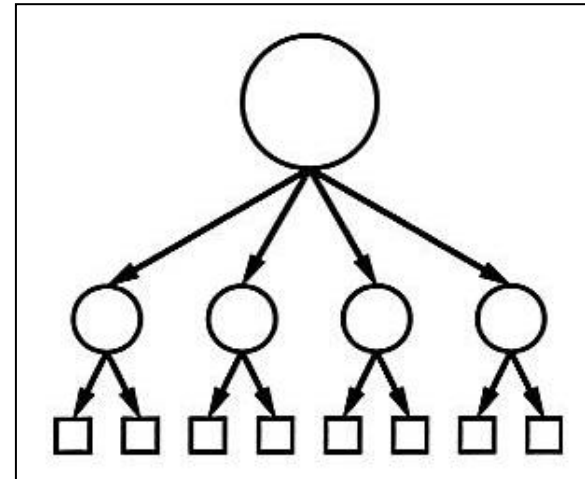


# Data Models

- Data Model is a collection of concepts that can be used to describe the structure of a database.
- Data Models are categorized into:
  - Hierarchical Model
  - Network Model
  - Relational Model

# Hierarchical Model

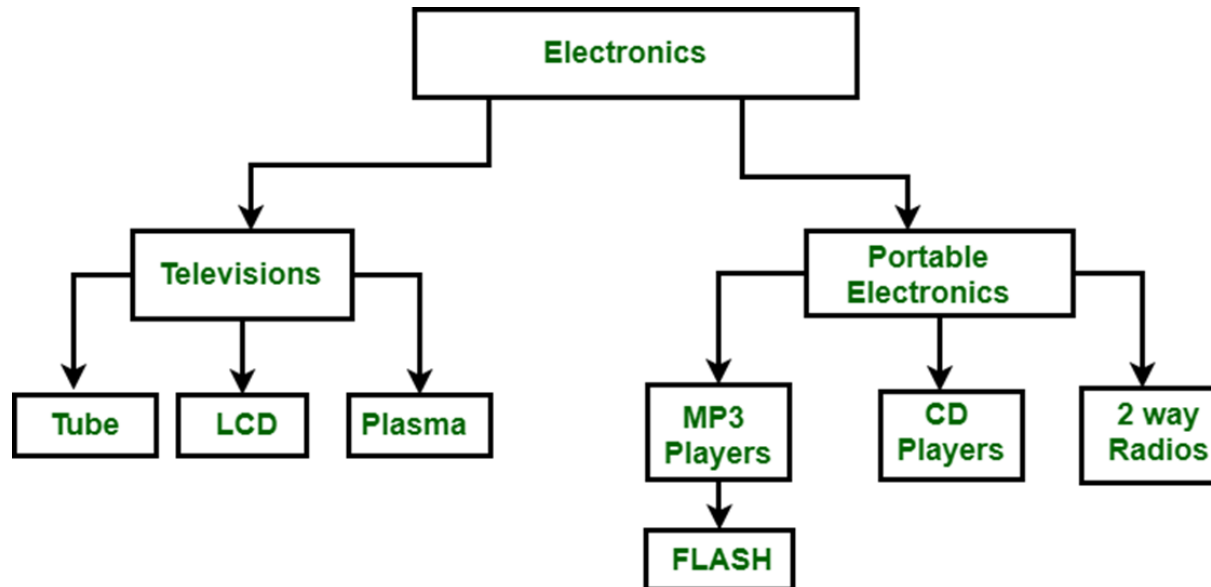
- ▶ Organizes data into tree structure.
- ▶ Hierarchy of parent-child data segment.
- ▶ Child data segments can have repeating information.
- ▶ Data is in series of records which has set of field values attached to it.





# Hierarchical Model

- One of the oldest models in a data model Developed by IBM, in the 1950s
- Data are viewed as a collection of tables, or segments that form a hierarchical relation
- Data is organized into a tree-like structure where each record consists of one parent record and many children.

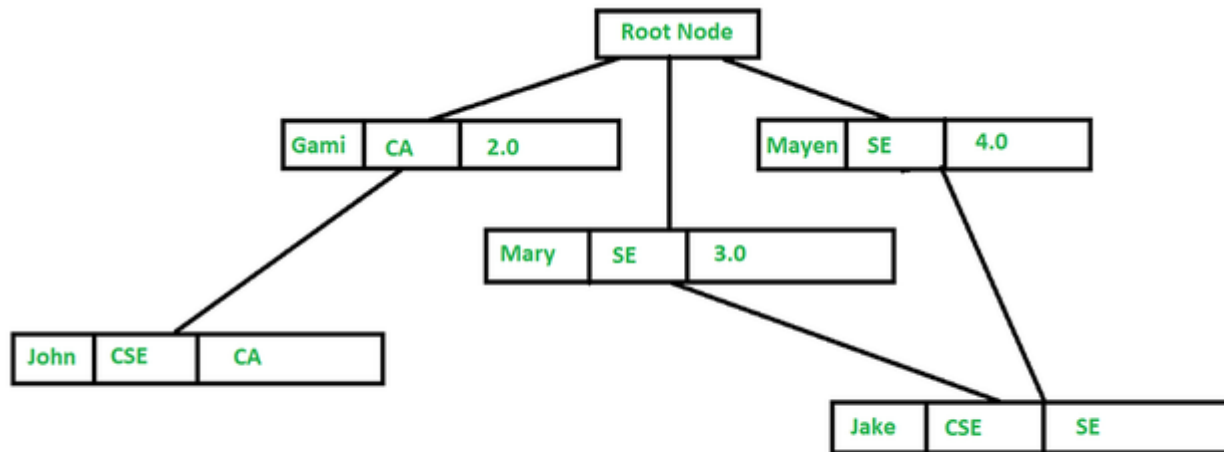


# Heirarchical Model

- In the hierarchical model, segments pointed to by the logical association are called the **child segment** and the other segment is called the **parent segment**.
- If there is a segment without a parent is then that will be called the **root** and the segment which has no children are called the **leaves**.
- The main disadvantage of the hierarchical model is that it can have one-to-one and one-to-many relationships between the nodes.

# Heirarchical Model

- In the figure, we have few students and few course-enroll and a course can be assigned to a single student only,
- but a student can enroll in any number of courses and with this the relationship becomes one-to-many.
- We can represent the given hierarchical model like the below relational tables:



# Applications

- Hierarchical models are generally used as semantic models in practice as many real-world occurrences of events are hierarchical in nature like biological structures, political, or social structures.
- Hierarchical models are also commonly used as physical models because of the inherent hierarchical structure of the disk storage system like tracks, cylinders, etc.
- There are various examples such as Information Management System (IMS) by IBM, NOMAD by NCSS, etc.

## Heirarchical Model Advantages

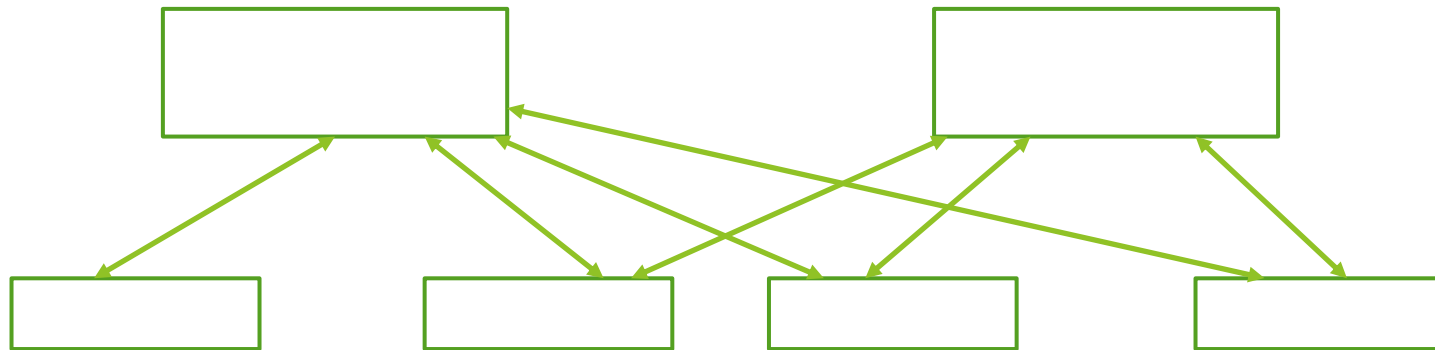
- As the database is based on this architecture the relationships between various layers are logically simple so, it has a very simple hierarchical database structure.
- It has data sharing as all data are held in a common database data and therefore sharing of data becomes practical.
- It offers data security and this model was the first database model that offered data security.
- There's also data integrity as it is based on the parent-child relationship and also there's always a link between the parents and the child segments.

## Heirarchical Model Disadvantages

- Even though this model is conceptually simple and easy to design at the same time it is quite complex to implement.
- This model also lacks flexibility as the changes in the new tables or segments often yield very complex system management tasks. Here, a deletion of one segment can lead to the involuntary deletion of all segments under it.
- It has no standards as the implementation of this model does not provide any specific standard.
- It is also limited as many of the common relationships do not conform to the 1 to N format as required by the hierarchical model.

# Network Model

- ▶ Data were modeled with more than one parent per child.
- ▶ Network model permitted many to many relationships in data.



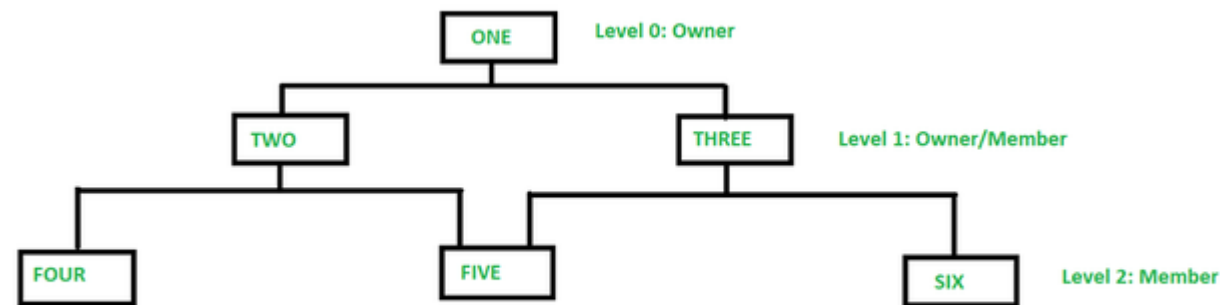
# Network Model

- This model was formalized by the Database Task group in the 1960s.
- This model is the generalization of the hierarchical model.
- This model can consist of multiple parent segments and these segments are grouped as levels but there exists a logical association between the segments belonging to any level.
- Mostly, there exists a many-to-many logical association between any of the two segments.
- We called **graphs** the logical associations between the segments.
- Therefore, this model replaces the hierarchical tree with a graph-like structure, and with that, there can more general connections among different nodes.



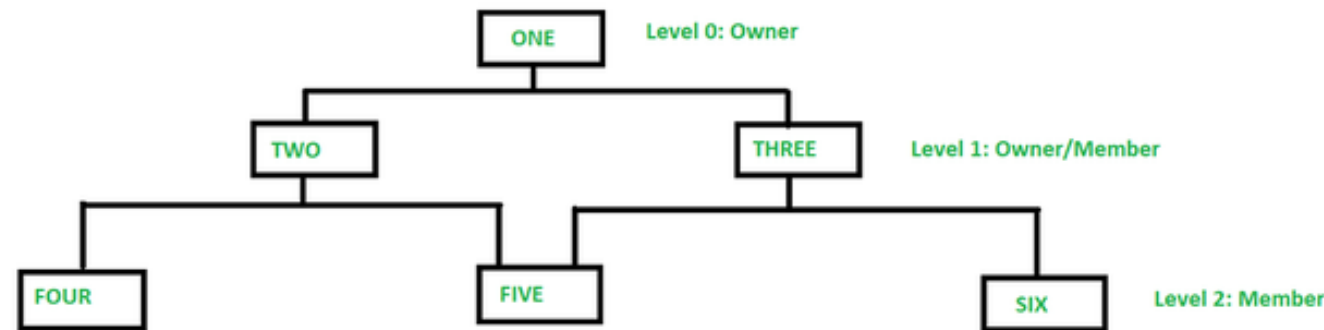
# Network Model

- It can have M: N relations i.e, many-to-many which allows a record to have more than one parent segment.
- Here, a relationship is called a set, and each set is made up of at least 2 types of record which are given below:
  - An owner record that is the same as of parent in the hierarchical model.
  - A member record that is the same as of child in the hierarchical model.



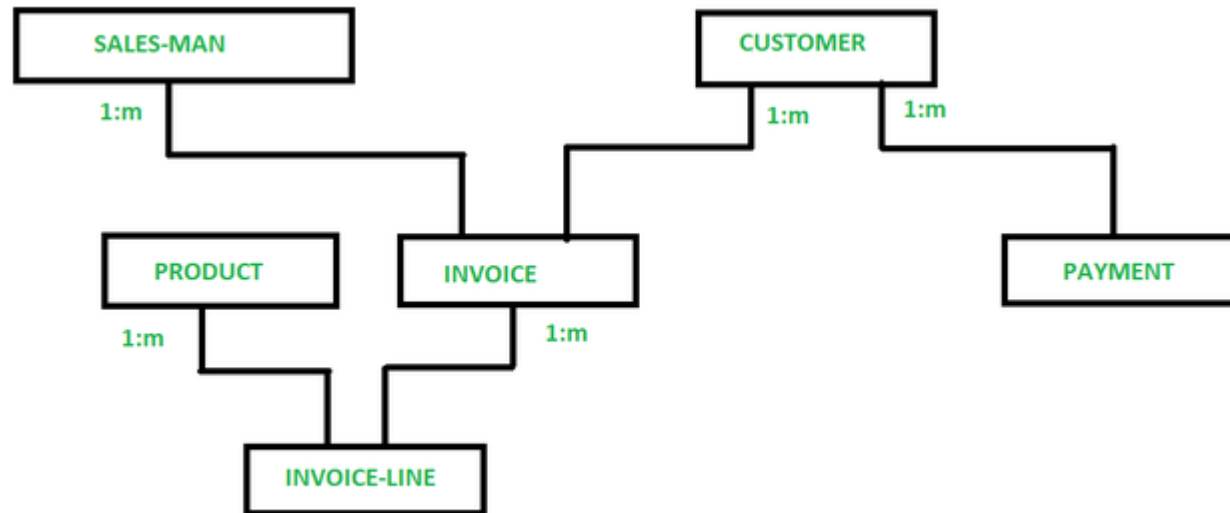
# Network Model

- In the figure, member TWO has only one owner 'ONE' whereas member FIVE has two owners i.e, TWO and THREE.
- Here, each link between the two record types represents 1 : M relationship between them.
- This model consists of both lateral and top-down connections between the nodes. Therefore, it allows 1: 1, 1 : M, M : N relationships among the given entities which helps in avoiding data redundancy problems as it supports multiple paths to the same record.



# Network Model

- Network Model for a Finance Dept.



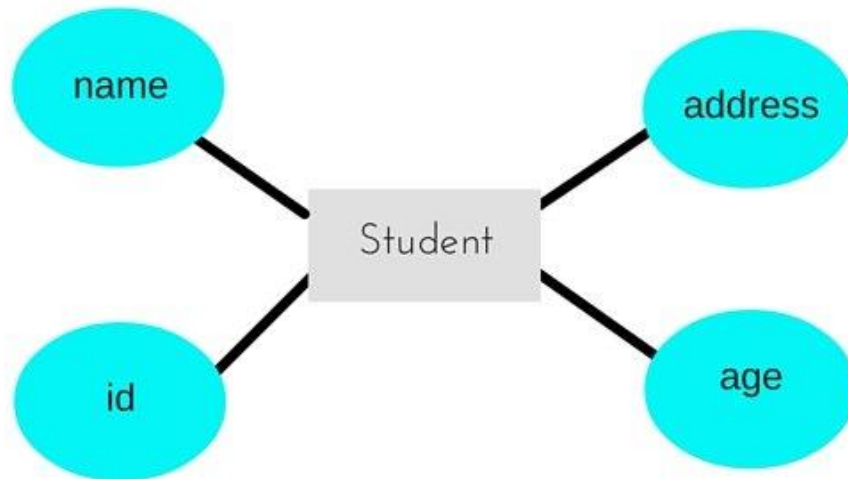
# Relational Model

- ▶ Relational database allows the definition of:
  - ▶ Data structures
  - ▶ Storage and retrieval operations and
  - ▶ Integrity constraints
- ▶ Data and relations between them are organized in tables.
- ▶ This model is developed by E. F. Codd.

Empno	Empname	Desig	Salary	Deptno
E001	Vandana	Manager	20000	10
E002	Amit	Executive	14000	20
E003	Amit	Accountant	10000	30
E004	Anju	Clerk	9000	10

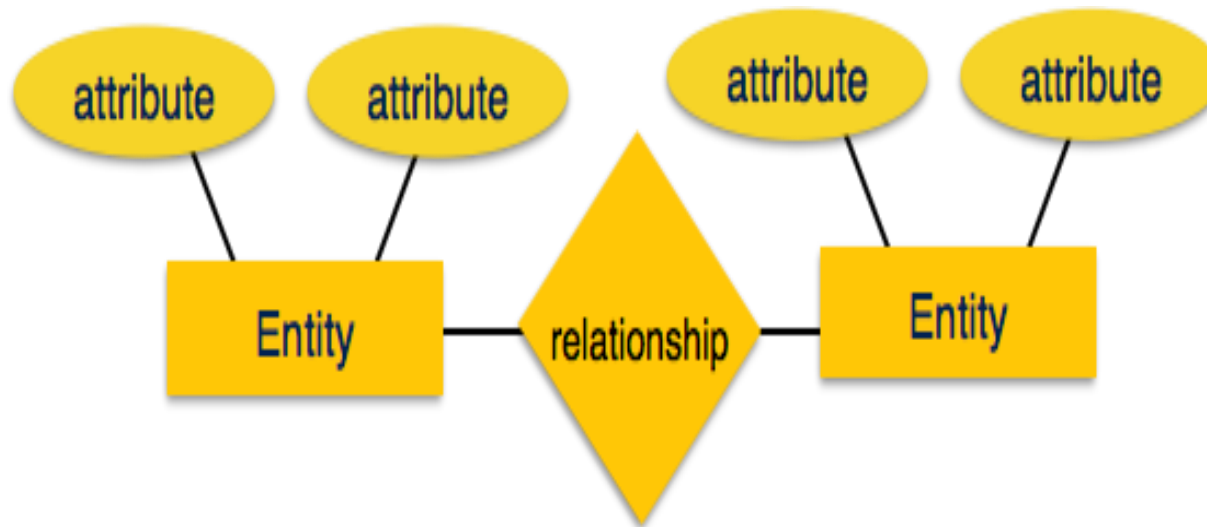
# Entity-relationship Model

- In this database model, relationships are created by dividing object of interest into entity and its characteristics into attributes.
- Different entities are related using relationships.
- E-R Models are defined to represent the relationships into pictorial form to make it easier for different stakeholders to understand.

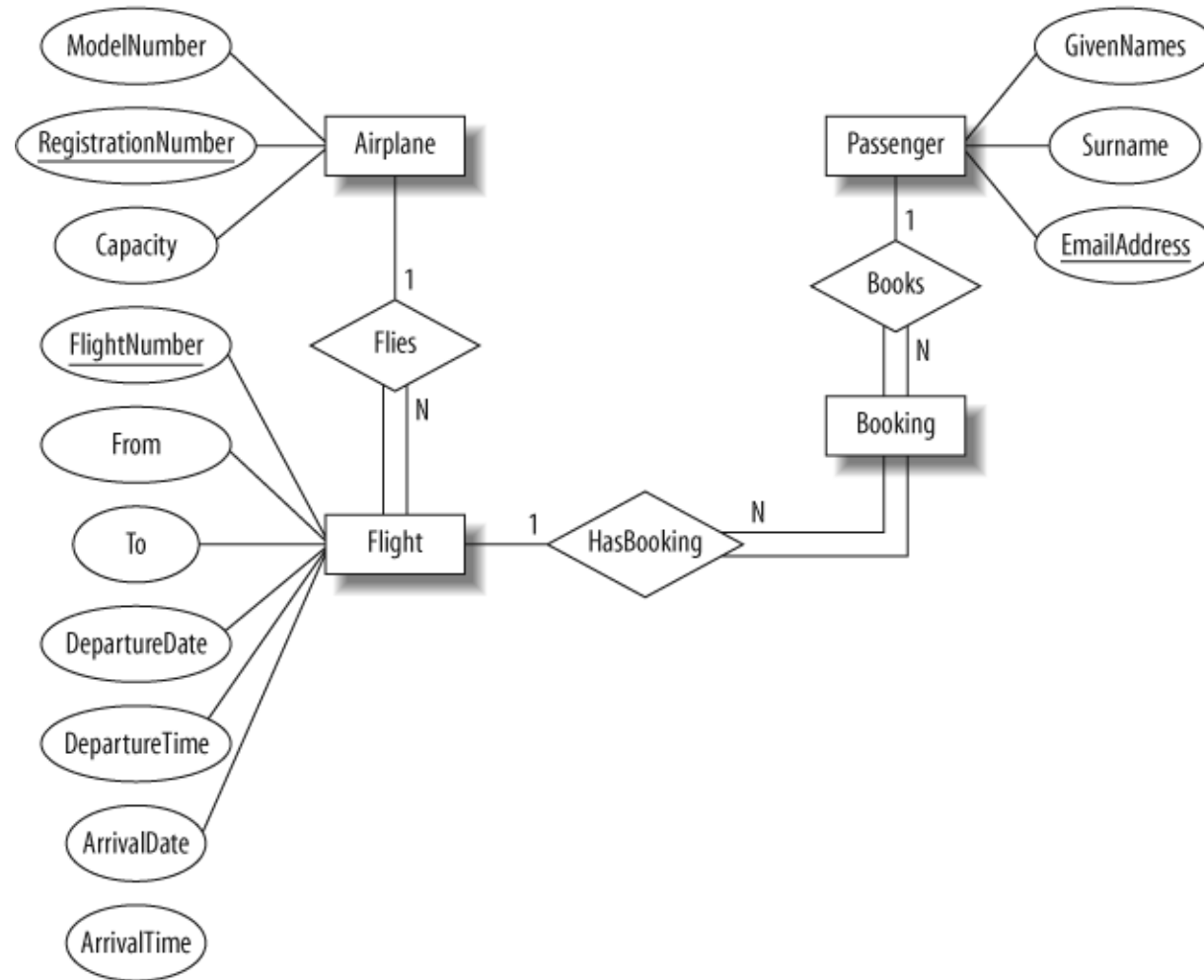


# Entity-Relationship Model

- ▶ Entity-Relationship (E-R) Modeling is a conceptual modeling tool.
- ▶ Perceives the business environment in terms of participating “entities” and the “relationship” between them.



# Entity-Relationship Model



# Entity-Relationship Model

- ▶ An **Airplane** is uniquely identified by its **RegistrationNumber**, so we use this as the **primary key**.
- ▶ A **Flight** is uniquely identified by its **FlightNumber**, so we use the flight number as the **primary key**.
- ▶ The departure and destination airports are captured in the **From and To attributes**, and we have separate attributes for **the departure and arrival date and time**.
- ▶ Because no two passengers will share an email address, we can use the **Email Address** as the **primary key** for the Passenger entity.
- ▶ An airplane can be involved in any number of flights, **while each flight uses exactly one airplane**, so the Flies relationship between the Airplane and Flight relationships has cardinality **1:N**;
- ▶ A passenger can book any number of flights, while a flight can be booked by any number of passengers. we could specify **an M:N Books** relationship between the **Passenger** and **Flight** relationship.
- ▶ There is a hidden entity here: the **booking** itself. **1:N** relationships between **Book** entity and the **Passenger and Flight** entities.



# ER -Model

- ER Model is used to model the logical view of the system from a data perspective which consists of these components:

## Components of E-R Model

1-Rectangles



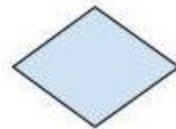
Represents entity

2-Ellipse



Represents attributes

3-Diamonds



Represents relationship among entities

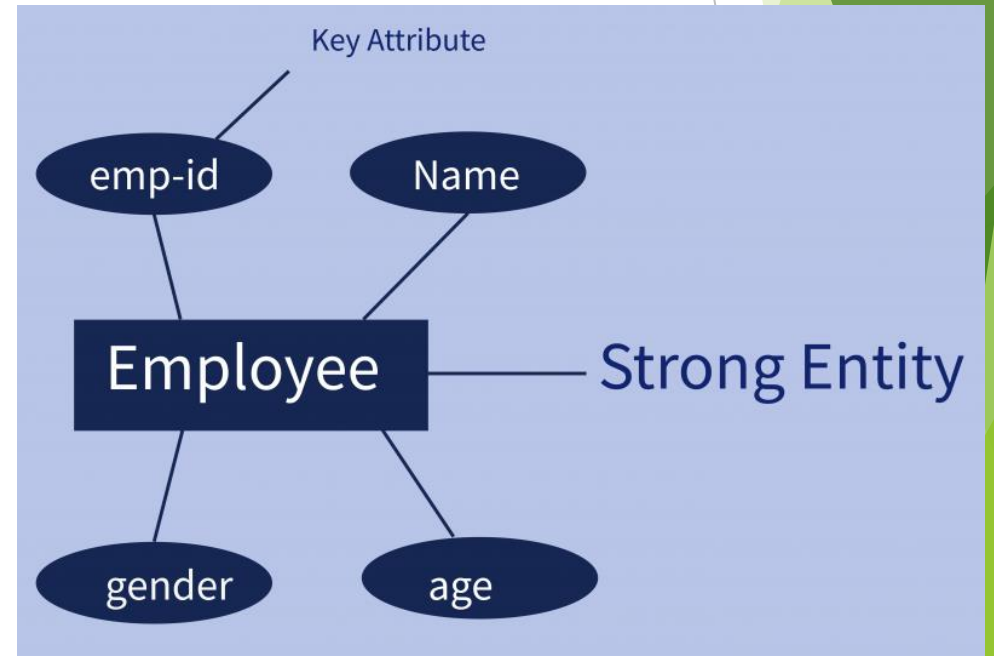
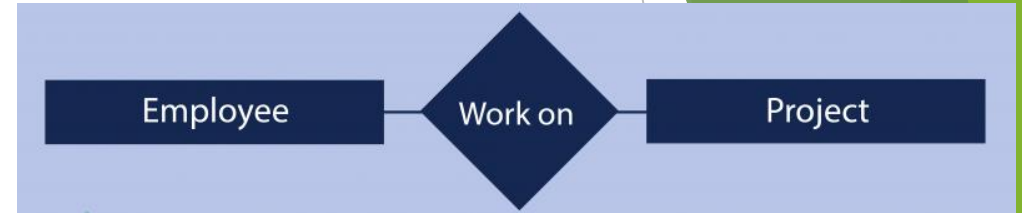
4-Lines



link attributes to entities and entity sets to relationship

# ER -Model

- Example of Entities and their relationships
- Attribute :
  - An attribute is a property or characteristic of an entity.
  - An entity may contain any number of attributes.
  - The attributes that can uniquely define an entity are considered as the primary key.
  - In an Entity-Relation model, attributes are represented in an elliptical shape.
  - It also may refer to a database field.



# ER -Model

**There are five such types of attributes:**

- **Simple attribute** - Attributes that are not further divisible into sub-attributes (atomic) are known as Simple attributes. **Ex:** Student Roll No, Employee Id.
- **Composite attribute** - Composite attributes can be divided into sub-attributes which represent more basic attributes with independent meanings. **Ex:** Employee Address can be divided into Street\_address, City, State, and Pincode.
- **Single-valued attribute** - Attributes having single value for a particular entity instance is known as single-valued attribute. **Ex:** the age of a person is single-valued.

# ER -Model

**There are five such types of attributes:**

- **Multi-valued attribute** - There are many instances where an attribute has a set of values for a specific entity, known as Multivalued attributes. **Ex:** Phone number. A person may have zero, one or more phone numbers

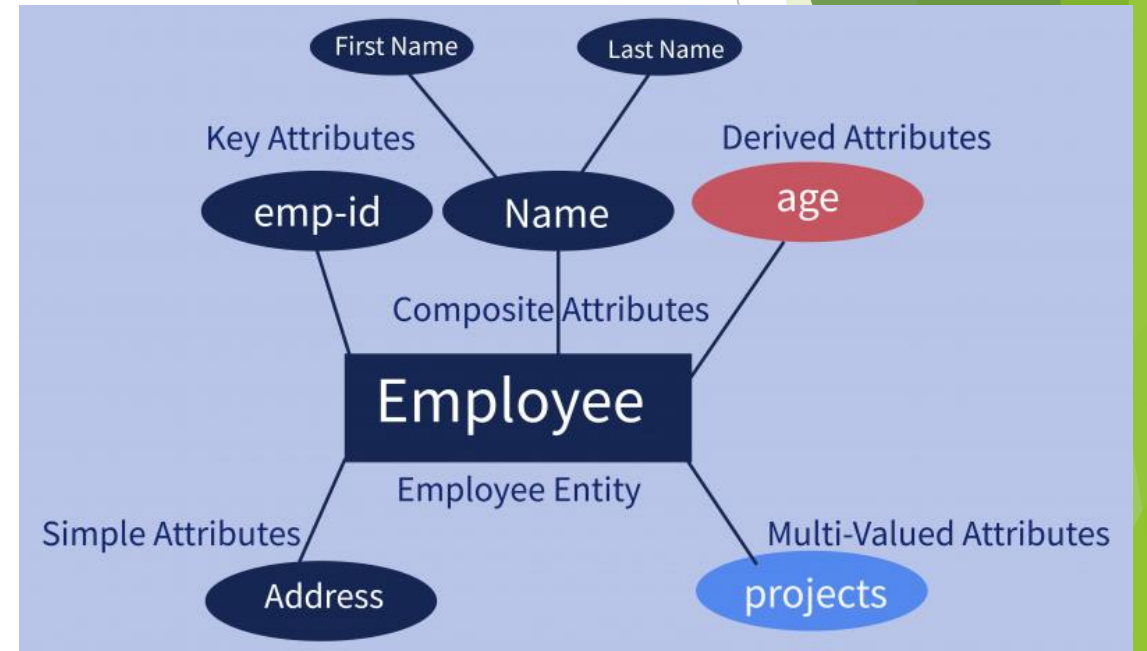
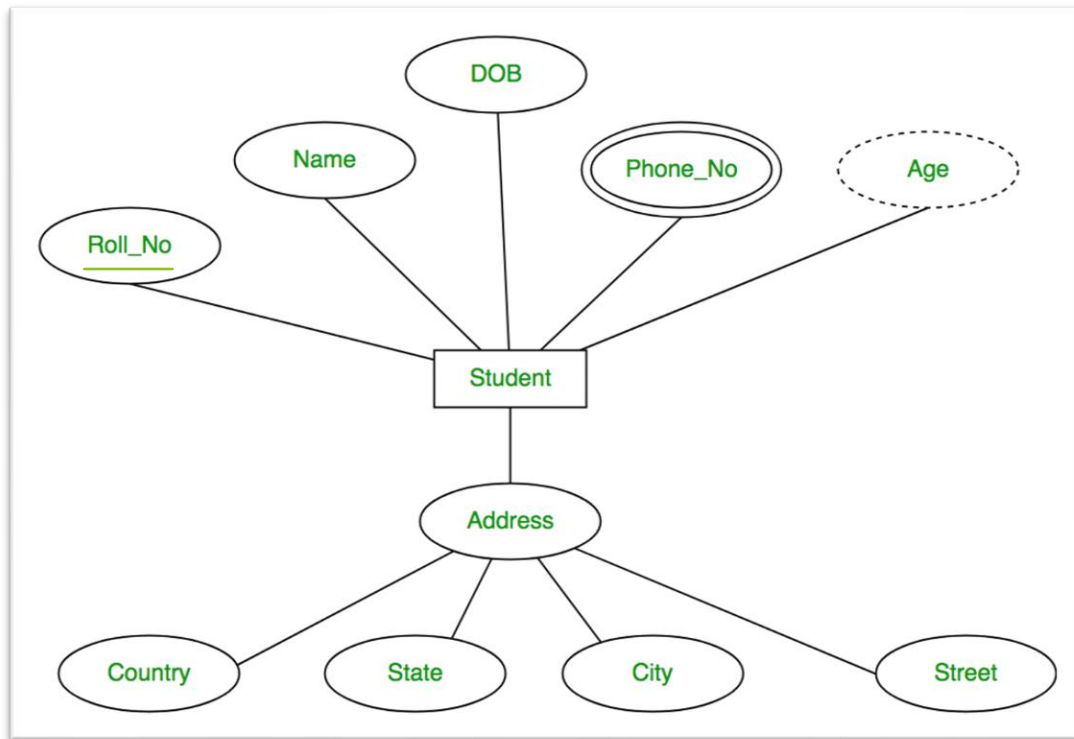
- **Derived attribute** - The value for this type of attribute can be derived from the values of other related attributes or entities instances. **Ex:** suppose that the employee entity set has an attribute age, which indicates the employee's age.

If the employee entity set also has an attribute date-of-birth, we can calculate age from date-of-birth and the current date. Thus, age is a derived attribute.

\*\*\*However, the derived attribute needs to be computed every time

# ER -Model

## Example of an Entity



# ER -Model

## **Relationships of Entities:**

- A relationship in a DBMS is primarily the way two or more data sets are linked.
- Relationships allow the datasets to share and store data in separate tables.
- They also help link disparate data with each other.

## **Types of relationships**

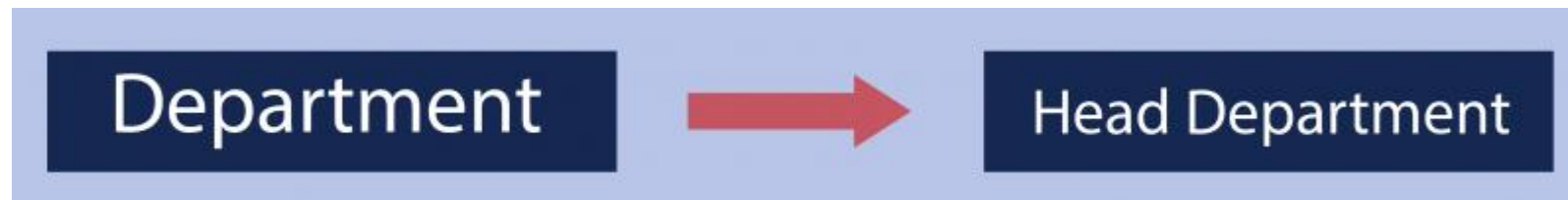
- One to One
- One to Many
- Many to Many

# ER -Model

## One to One –

- It is used to create a relationship between two tables in which a single row of the first table can only be related to one and only one record of a second table.
- This relationship tells us that a single record in Table A is related to a single record in Table B. And vice versa.

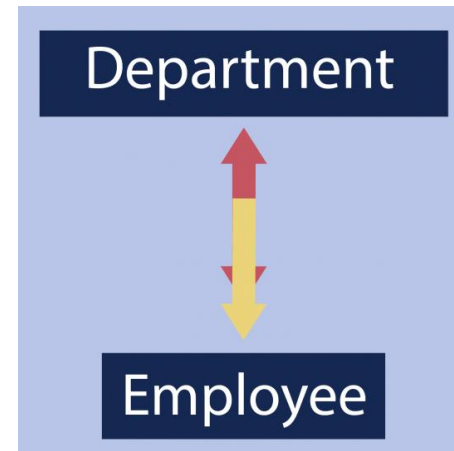
Example – In a university, each department has only one head of the department. And one HOD can take only one department. This shows a one-to-one (1:1) relationship between the department and the person as a head.



# ER -Model

## One to Many –

- It is used to create a relationship between two tables. Any single row of the first table can be related to one or more rows of the second table, but the rows of the second table can only relate to the only row in the first table.
- It is also known as a many-to-one relationship.
- Example: of a 1:M relationship is A department that has many employees, Each employee is assigned to one department.

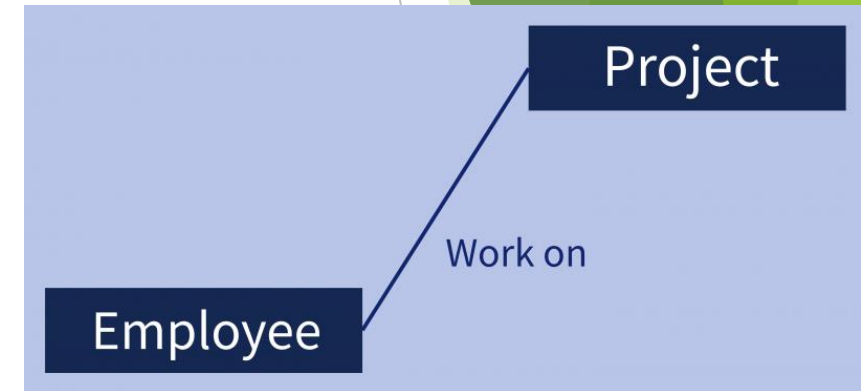




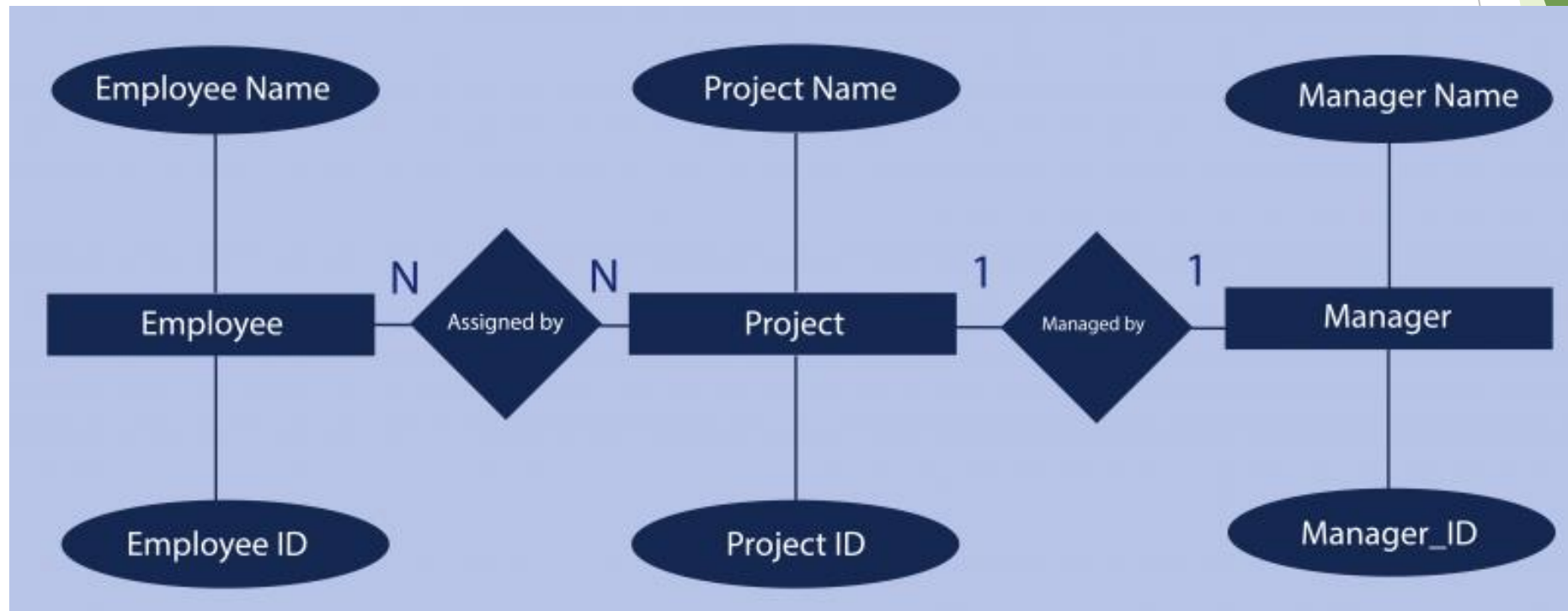
# ER -Model

## Many to Many –

- Many to many relationships that create a relationship between two tables.
- Each record of the first table can relate to any records (or no records) in the second table.
- Similarly, each record of the second table can also relate to more than one record of the first table. It also represented an N:N relationship.
- Example: there are many employees involved in each project, and every employee can involve in more than one project.



# Example of an Entity Relationship model



# Properties of RDBMS

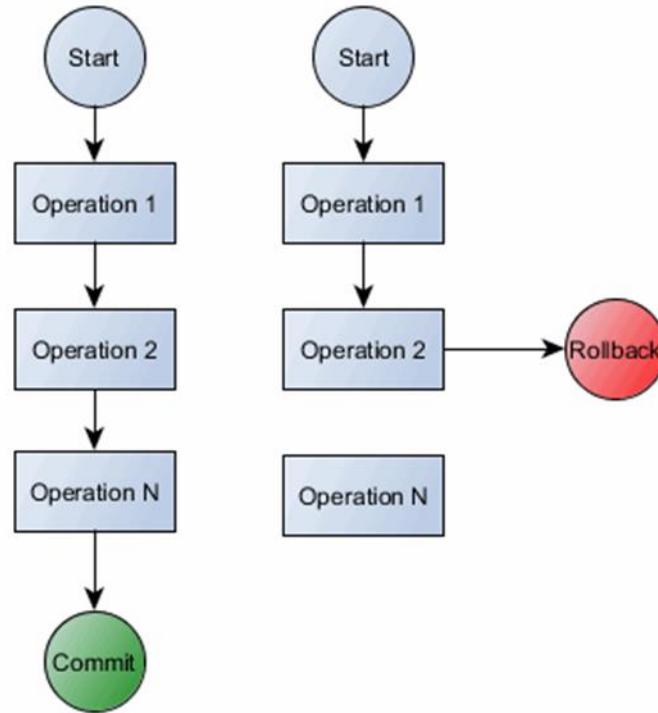
- Every row is unique
- All of the values present in a column hold the same data type
- Values are atomic
- The columns sequence is not significant
- The rows sequence is not significant
- The name of every column is unique

# Transactions -ACID properties

- ▶ **Atomicity** : What if the OS crashed after Rs.100 was deposited to the first account'? DBMS must assure that the Rs.100 is withdrawn from the first account.
- ▶ **Consistency**: What if a transaction just deposited Rs.100 into an account'? The programmer must ensure that all transactions are consistent
- ▶ **Isolation**: What if another transaction computed the total bank balance after Rs.100 was deposited to the first account'?
- ▶ **Durability**: What if, after the commit, the OS crashed before the withdrawal was written to disk'? DBMS must assure that the withdrawal was at least logged.

# Transactions -ACID properties

## ► Commit And RollBack



# Transactions -ACID properties

**Atomic** : A transaction occurs entirely or not at all

**Consistency** : each transaction preserves the consistency of the database

**Isolated** : concurrent transactions do not interface with each other

**Durable** : once completed, a transaction's changes are permanent

# Database Schema

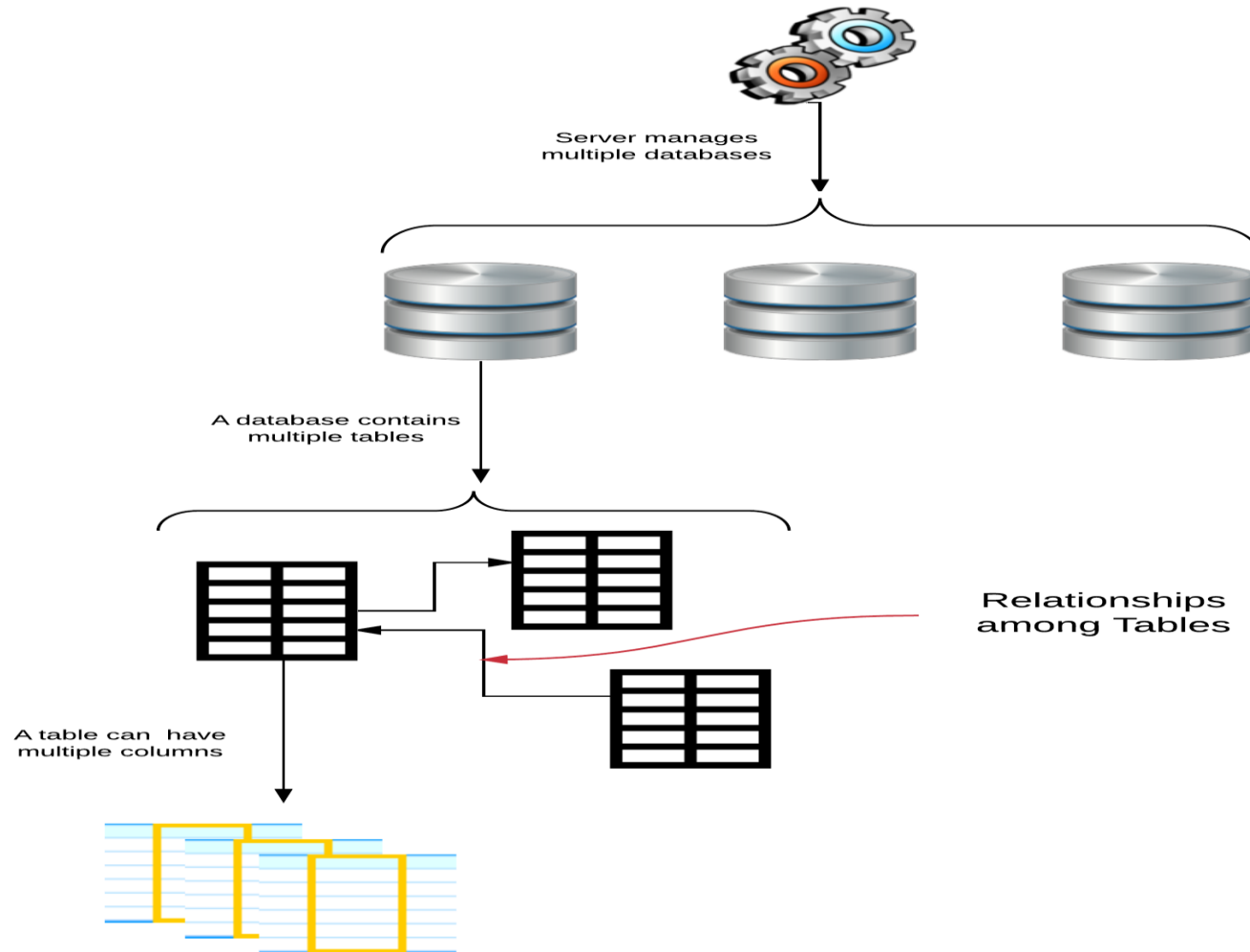
- ▶ Structure described in a formal language supported by the DBMS.
- ▶ Refers to the organization of data to create a blueprint of how a database will be divided into database tables.
- ▶ LMS Time!!!!

# Relational Model

- In this model, data is organised in two-dimensional **tables** and the relationship is maintained by storing a common field.
- This model was introduced by E.F Codd in 1970, and since then it has been the most widely used database model, infact, we can say the only database model used around the world.
- The basic structure of data in the relational model is **tables**. All the information related to a particular type is stored in **rows** of that table.
- Tables are also known as **relations** in relational model.



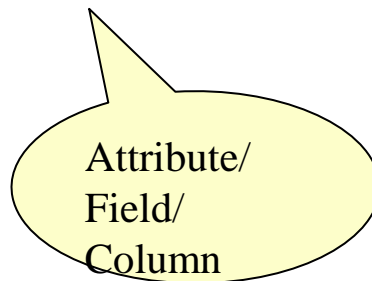
# What is Relational Database Management System (RDBMS)?



# RDBMS continued...

- Relational database model is a collection of tables.

DEPT	
Deptno	Dname



EMP				
Empno	Ename	Desig	Salary	Deptno

# RDBMS continued...

- ▶ A relational table is a flat file which is composed of :
  - ▶ A set of named columns and
  - ▶ Arbitrary number of rows.


In a relational database the data is maintained across multiple tables.

Record/  
Tuple/  
Row

EMP				
Empno	Ename	Desig	Salary	Deptno
E001	Vandana	Manager	20000	10
E002	Amit	Executive	14000	20
E003	Amit	Accountant	10000	30
Attributes				

student_id	name	age
1	Akon	17
2	Bkon	18
3	Ckon	17
4	Dkon	18

subject_id	name	teacher
1	Java	Mr. J
2	C++	Miss C
3	C#	Mr. C Hash
4	Php	Mr. P H P



The diagram illustrates a join operation. Two arrows originate from the bottom of the 'student' and 'subject' tables. These arrows converge and point down to a third table, indicating that the data from the first two tables is being combined into this new table.

student_id	subject_id	marks
1	1	98
1	2	78

# DBMS AND RDBMS

	DBMS	RDBMS
1.	DBMS applications store <b>data as file</b> .	RDBMS applications store <b>data in a tabular form</b> .
2.	In DBMS, data is generally stored in either a hierarchical form or a navigational form.	In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables.
3.	<b>Normalization is not</b> present in DBMS.	<b>Normalization is</b> present in RDBMS.
4.	DBMS does <b>not apply any security</b> with regards to data manipulation.	RDBMS <b>defines the integrity constraint</b> for the purpose of ACID (Atomicity, Consistency, Isolation and Durability) property.
5.	DBMS uses file system to store data, so there will be <b>no relation between the tables</b> .	in RDBMS, data values are stored in the form of tables, so a <b>relationship</b> between these data values will be stored in the form of a table as well.
6.	DBMS has to provide some uniform methods to access the stored information.	RDBMS system supports a tabular structure of the data and a relationship between them to access the stored information.
7.	DBMS <b>does not support distributed database</b> .	RDBMS <b>supports distributed database</b> .
8.	DBMS is meant to be for small organization and <b>deal with small data</b> . it supports <b>single user</b> .	RDBMS is designed to <b>handle large amount of data</b> . it supports <b>multiple users</b> .
9.	Data Redundancy is common in this model leading to difficulty in maintaining the data.	Keys and indexes are used in the tables to avoid redundancy.
10.	Example DBMS are dBase, Microsoft Access, LibreOffice Base, FoxPro.	Example RDBMS are SQL Server, Oracle , MySQL, Maria DB, SQLite.

# What is Normalization?

- ▶ Is the process of eliminating or removing any redundancy in the database.
- ▶ Normalization also helps in eliminating anomalies like Insertion, Deletion and Updation. It reduces redundancy from a relation or set of relations.
- ▶ Normalization rules divides larger tables into smaller tables and links them by using relationships.
- ▶ This concept was proposed by Edgar Codd, who created First Normal Form (1NF) then extended to Second Normal (2NF) and Third Normal Form (3NF). Later, he went on to collaborate with Raymond F. Boyce to develop the theory of Boyce-Codd Form (BCNF).

# Anomalies

- ▶ Anomalies in context of DBMS is nothing but problems.
- ▶ Databases which have redundant data may have anomalies.
- ▶ There are three types of Anomalies:
  - Insertion
  - Deletion
  - Updation
- LMS Time!!!

# What is Functional Dependency?

- ▶ Is a constraint that determines the relation of one attribute with another in DBMS.
- ▶ It helps in maintaining the quality of data. It plays vital role in differentiating between a good and bad database.
- ▶ It is denoted by “ $\rightarrow$ ” .
- ▶ Let's say X and Y are attributes of a table R, and Y is functionally dependent on X. It means Y is determined by X.



# Example of Functional Dependency

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

Here, we can say if we know the Employee Number then we can determine the Employee name, Salary and City

# Normalization

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

# Normalization

- The most commonly used Normalization forms are from 1 to 3

	1NF	2NF	3NF	4NF	5NF
Decomposition of Relation	R	R <sub>11</sub> R <sub>12</sub>	R <sub>21</sub> R <sub>22</sub> R <sub>23</sub>	R <sub>31</sub> R <sub>32</sub> R <sub>33</sub> R <sub>34</sub>	R <sub>41</sub> R <sub>42</sub> R <sub>43</sub> R <sub>44</sub> R <sub>45</sub>
Conditions	Eliminate Repeating Groups	Eliminate Partial Functional Dependency	Eliminate Transitive Dependency	Eliminate Multi-values Dependency	Eliminate Join Dependency

# Normalization

- Assume, a video library maintains a database of movies rented out.
- Without any normalization in database, all information is stored in one table as shown below.

FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean, Clash of the Titans	Ms.
Robert Phil	3 <sup>rd</sup> Street 34	Forgetting Sarah Marshal, Daddy's Little Girls	Mr.
Robert Phil	5 <sup>th</sup> Avenue	Clash of the Titans	Mr.

# Normalization

## •1NF :

- Each table cell should contain a single value.
- Each record needs to be unique.

FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean, Clash of the Titans	Ms.
Robert Phil	3 <sup>rd</sup> Street 34	Forgetting Sarah Marshal, Daddy's Little Girls	Mr.
Robert Phil	5 <sup>th</sup> Avenue	Clash of the Titans	Mr.

FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean	Ms.
Janet Jones	First Street Plot No 4	Clash of the Titans	Ms.
Robert Phil	3 <sup>rd</sup> Street 34	Forgetting Sarah Marshal	Mr.
Robert Phil	3 <sup>rd</sup> Street 34	Daddy's Little Girls	Mr.
Robert Phil	5 <sup>th</sup> Avenue	Clash of the Titans	Mr.

# What are Keys & Why do we need them?

- ▶ Is an attribute or a set of attributes which helps us uniquely identify a row (tuple) in a relation (table)
- ▶ To put it simply, Keys help you uniquely identify a row in a table by combination of one or more columns in a table.

## Why do we need Keys?

- ▶ Records could be complicated. In a real-world application, databases can have thousands of records. Hence, keys can help us identify each record uniquely.
- ▶ Allows you to establish a relationship between and identify the relation between them.
- ▶ Help you to enforce identity and integrity in the relationship

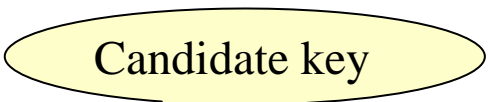
# Types of Keys in DBMS

► There are mainly eight types of Keys in DBMS:

1. Super Keys
2. Candidate Keys
3. Primary Keys
4. Alternate Keys
5. Foreign Keys
6. Compound Keys
7. Composite Keys
8. Surrogate Keys

# Candidate Keys

- ▶ The key which is in the race of primary key is candidate key.
- ▶ Each table may have one or more than one as a candidate key.
- ▶ One of the candidate key is selected as primary key.
- ▶ For example:



Country Table		
Country_Code	Country_Name	Capital
IN	India	Delhi
AU	Australia	Canabara
US	Unites States	Washington DC



# Primary Keys

- Consider the following table:

Empname	Desig	Salary	Deptno
Vandana	Manager	20000	10
Amit	Executive	14000	20
Amit	Executive	14000	20
Anju	Clerk	9000	10

- In this table, there exists two employees with the same data.
- It means the same data is repeated here.
- Uniqueness of the data is not maintained.
- To avoid this and maintain uniqueness of the data **Primary key** is used.

# Primary Keys continued...

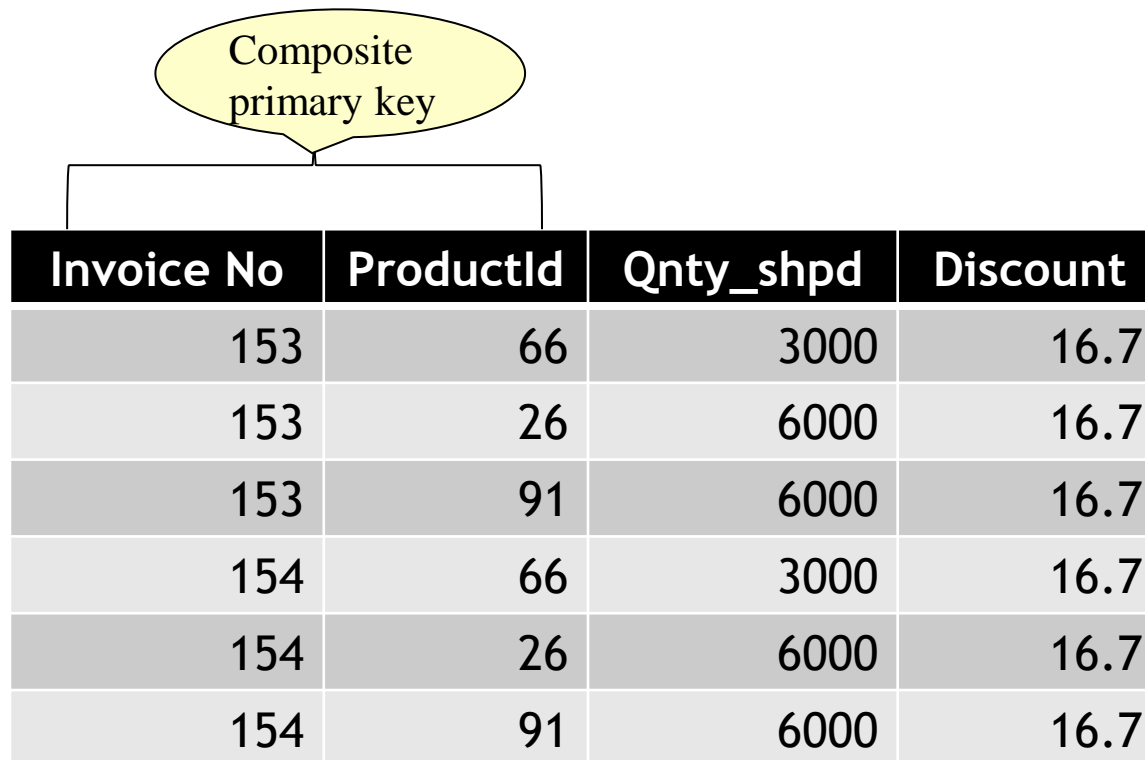
- ▶ In this table the employee is identified by it's Empno.
- ▶ Empno is used here to uniquely identify the Empno.
- ▶ Hence Empno is considered here as a primary key.

Primary  
key

Empno	Empname	Designation	Salary	Deptno
E001	Vandana	Manager	20000	10
E002	Amit	Executive	14000	20
E003	Amit	Accountant	10000	30
E004	Anju	Clerk	9000	10

# Composite Primary Keys

- ▶ Primary key consists of two or more than two columns is considered to be composite primary key.

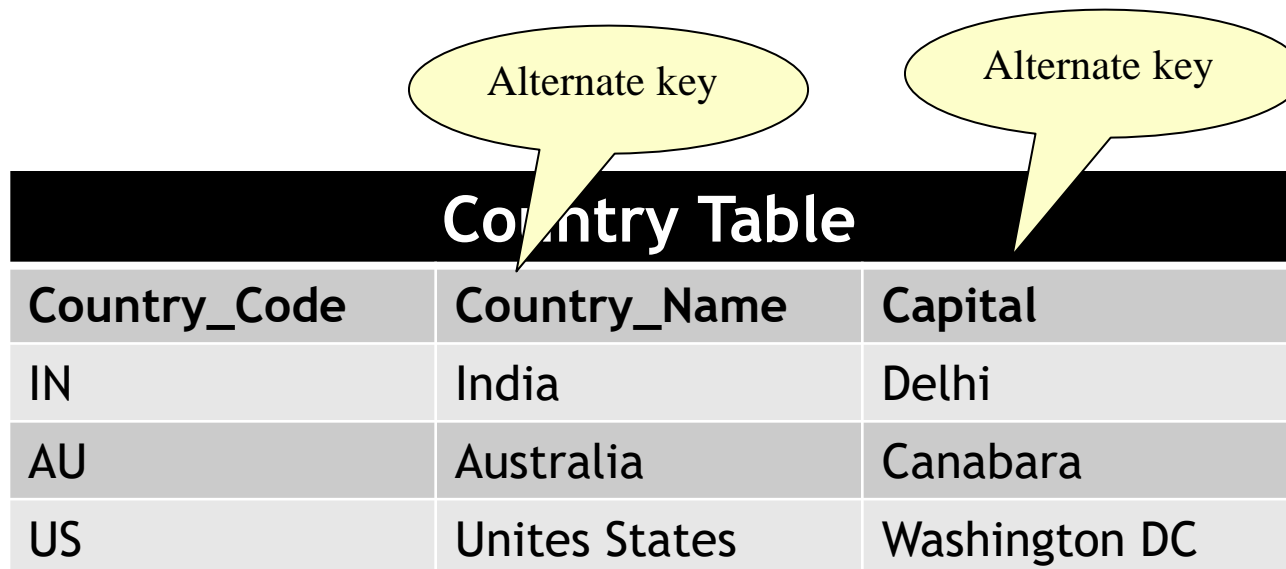


A diagram illustrates a composite primary key. A yellow oval containing the text "Composite primary key" is connected by a line to a bracket that spans the first two columns of the table below: "Invoice No" and "ProductId".

Invoice No	ProductId	Qty_shpd	Discount
153	66	3000	16.7
153	26	6000	16.7
153	91	6000	16.7
154	66	3000	16.7
154	26	6000	16.7
154	91	6000	16.7

# Alternate Keys

- ▶ An alternate key is similar to a primary key.
- ▶ It accepts null values; where as the primary key does not.
- ▶ The null values can be submitted to the attribute in a tuple.

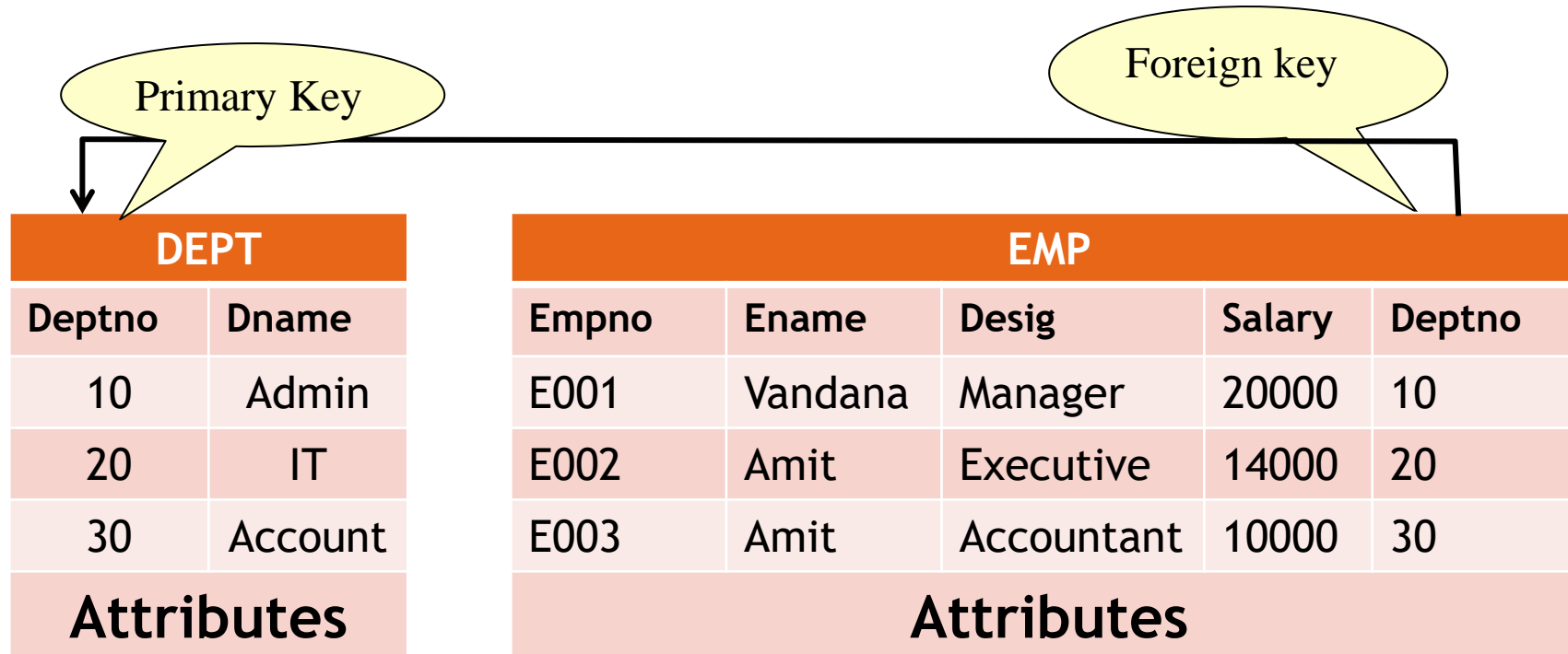


The diagram shows a table titled "Country Table". Above the table, two yellow speech bubbles, each labeled "Alternate key", point to the "Country\_Code" and "Country\_Name" columns respectively. The table has three columns: "Country\_Code", "Country\_Name", and "Capital". The data rows are: (IN, India, Delhi), (AU, Australia, Canabara), and (US, Unites States, Washington DC).

Country Table		
Country_Code	Country_Name	Capital
IN	India	Delhi
AU	Australia	Canabara
US	Unites States	Washington DC

# Foreign Keys

- ▶ Foreign key is a link or relationship between two tables.
- ▶ Ensures that the data stored in a database is consistent.



# Composite key

## **What is Composite Key?**

- A composite key is a primary key composed of multiple columns used to identify a record uniquely
- In our database, we have two people with the same name Robert Phil, but they live in different places.
- Hence, we require both Full Name and Address to identify a record uniquely. That is a composite key.

# Normalization

## •2NF :

- Rule 1- Be in 1NF
- Rule 2- Single Column Primary Key that does not functionally dependant on any subset of candidate key relation.
- To Achieve this, **partition** the table
- Introduce a new column called Membership\_id which is the primary key for table 1
- No partial Dependencies**

FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean	Ms.
Janet Jones	First Street Plot No 4	Clash of the Titans	Ms.
Robert Phil	3 <sup>rd</sup> Street 34	Forgetting Sarah Marshal	Mr.
Robert Phil	3 <sup>rd</sup> Street 34	Daddy's Little Girls	Mr.
Robert Phil	5 <sup>th</sup> Avenue	Clash of the Titans	Mr.

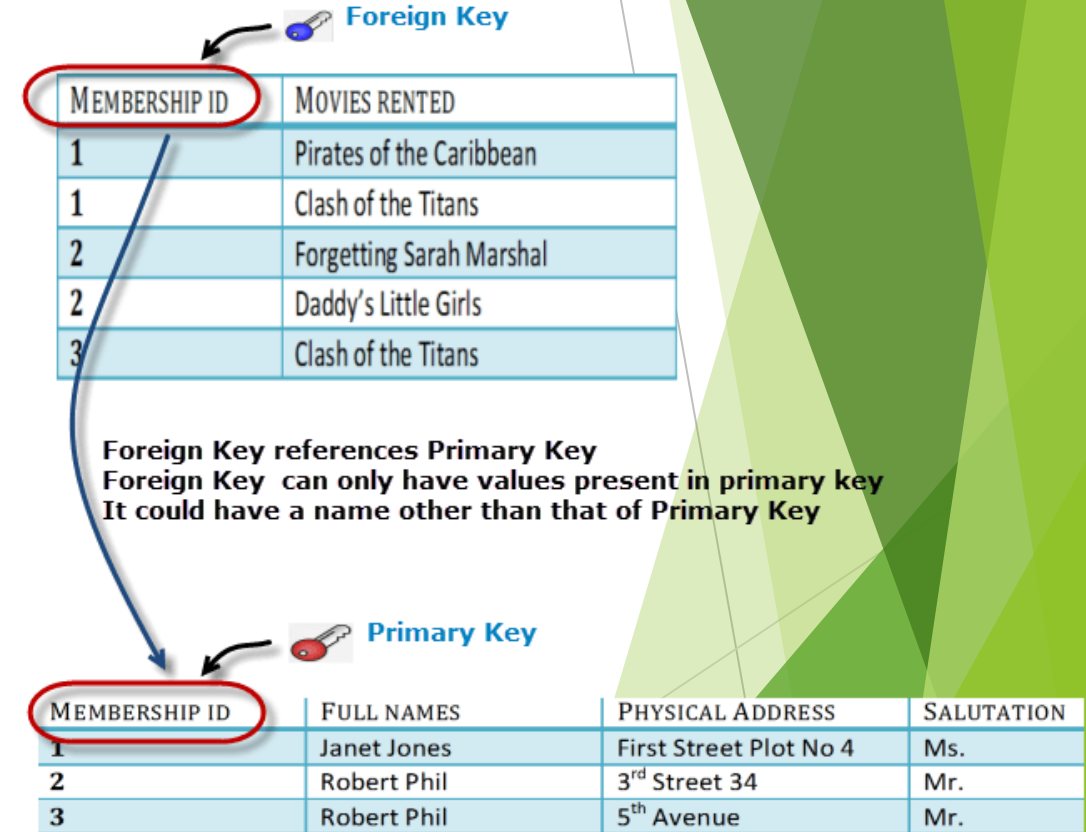
MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3 <sup>rd</sup> Street 34	Mr.
3	Robert Phil	5 <sup>th</sup> Avenue	Mr.

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

# Foreign key

## Foreign Key

- In Table 2, Membership\_ID is the Foreign Key.
- Foreign Key references the primary key of another Table! It helps connect your Tables
- A foreign key can have a different name from its primary key
- It ensures rows in one table have corresponding rows in another
- Unlike the Primary key, they do not have to be unique. Most often they aren't
- Foreign keys can be null even though primary keys can not





# What are Transitive Functional Dependencies

A transitive functional dependency is when changing a non-key column, might cause any of the other non-key columns to change

Consider the table 1. Changing the non-key column Full Name may change Salutation.

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3 <sup>rd</sup> Street 34	Mr.
3	Robert Phil	5 <sup>th</sup> Avenue	Mr.

*Change in Name*

*May Change  
Salutation*

# Normalization

## •3NF :

- Rule 1- Be in 2NF
  - Rule 2- Has no transitive functional dependencies
- To move our 2NF table into 3NF, we again need to again divide our table.

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3 <sup>rd</sup> Street 34	Mr.
3	Robert Phil	5 <sup>th</sup> Avenue	Mr.

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION ID
1	Janet Jones	First Street Plot No 4	2
2	Robert Phil	3 <sup>rd</sup> Street 34	1
3	Robert Phil	5 <sup>th</sup> Avenue	1

SALUTATION ID	SALUTATION
1	Mr.
2	Ms.
3	Mrs.
4	Dr.

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

# Normalization

## BCNF (Boyce-Codd Normal Form)

- Even when a database is in 3<sup>rd</sup> Normal Form, still there would be anomalies resulted if it has more than one **Candidate** Key.
- Sometimes is BCNF is also referred as **3.5 Normal Form**.
- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency  $X \rightarrow Y$ , X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.
- Example: Let's assume there is a company where employees work in more than one department.

# Normalization

## BCNF (Boyce-Codd Normal Form)

- Even when a database is in 3<sup>rd</sup> Normal Form, still there would be anomalies resulted if it has more than one **Candidate** Key.
- Sometimes is BCNF is also referred as **3.5 Normal Form**.
- Functional Dependencies of the table
  - EMP\_ID → EMP\_COUNTRY
  - EMP\_DEPT → {DEPT\_TYPE, EMP\_DEPT\_NO}
- Candidate key: {EMP-ID, EMP-DEPT}

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

# Normalization

## BCNF (Boyce-Codd Normal Form)

The table is not in BCNF because neither EMP\_DEPT nor EMP\_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP\_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP\_DEPT\_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

EMP\_ID → EMP\_COUNTRY  
EMP\_DEPT → {DEPT\_TYPE, EMP\_DEPT\_NO}

## 4<sup>th</sup> Normal Form

- Two attributes (or columns) in a table are independent of one another, but both depend on a third attribute. A
- multivalued dependency always requires at least three attributes because it consists of at least two attributes that are dependent on a third.
- For a dependency  $A \twoheadrightarrow B$ , if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.
- The table should have at least 3 attributes and B and C should be independent for  $A \twoheadrightarrow B$  multivalued dependency. For example,

# 4<sup>th</sup> Normal Form

- A relation will be in 4NF if
  - it is in Boyce Codd normal form and
  - has no multi-valued dependency.
- For a dependency  $A \twoheadrightarrow B$ , if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

# 4<sup>th</sup> Normal Form

- The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.
- In the STUDENT relation, a student with STU\_ID, 21 contains two courses, Computer and Math and two hobbies, Dancing and Singing. So there is a Multi-valued dependency on STU\_ID, which leads to unnecessary repetition of data.

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey



# Normalization

## 5NF (Fifth Normal Form) Rules

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

# Normalization

## **5NF (Fifth Normal Form) Rules**

- In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.
- Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.
- So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

# Normalization

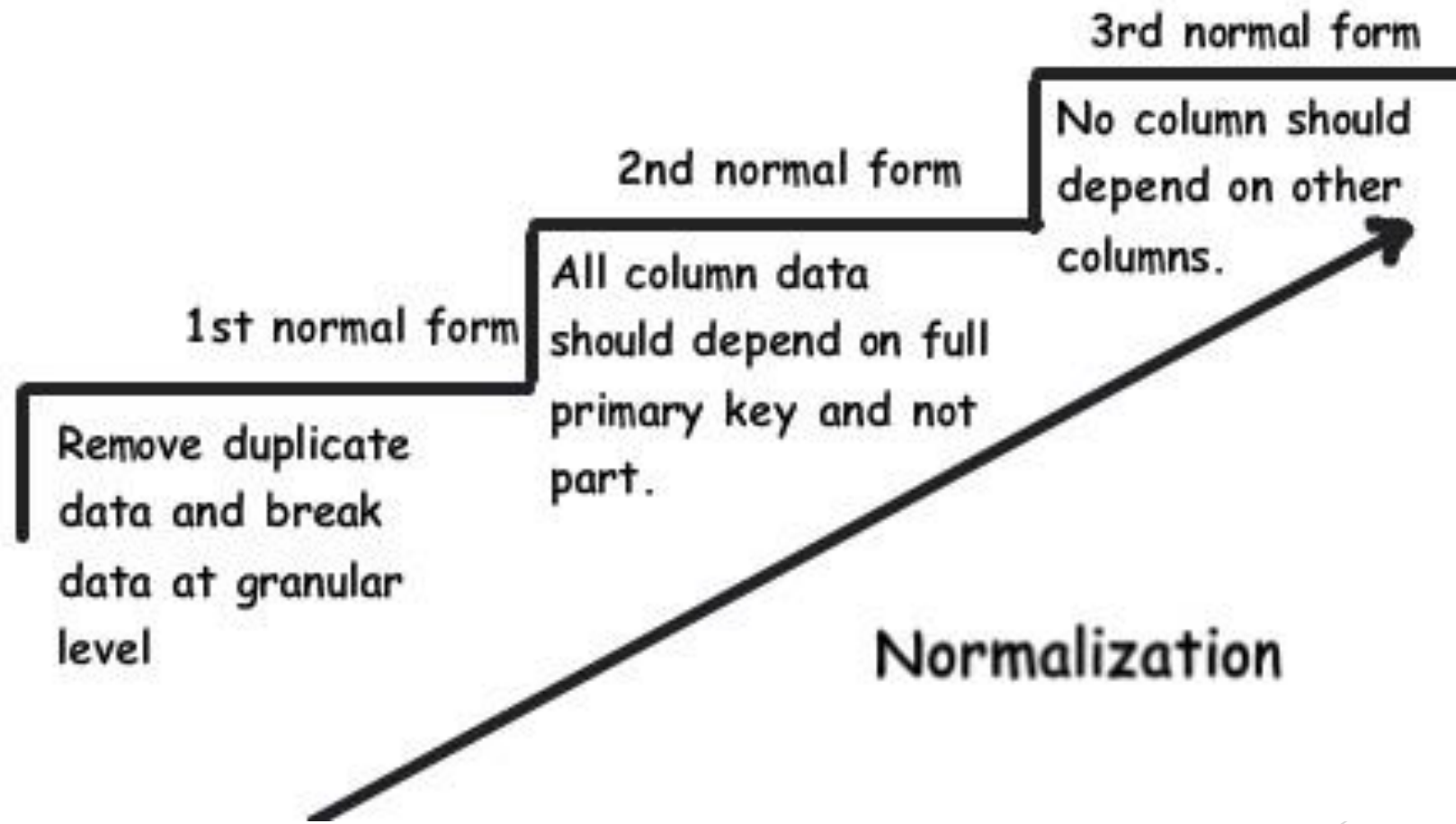
## 5NF (Fifth Normal Form) Rules

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

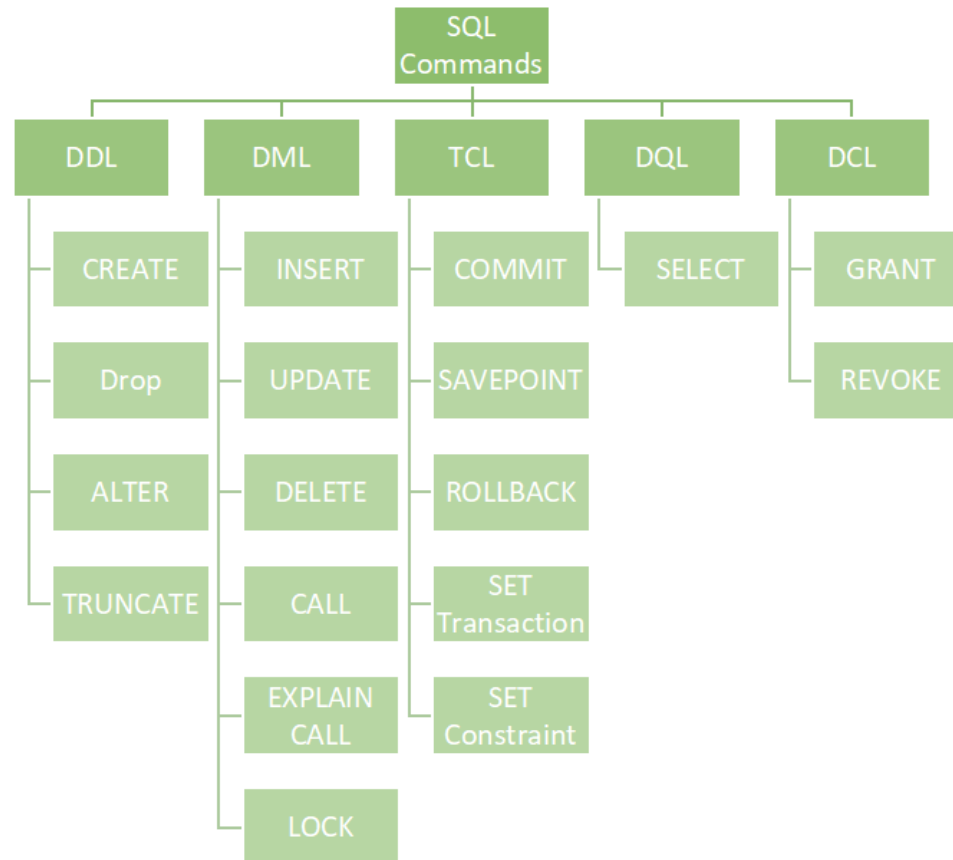
# Summary



# SQL | DDL, DQL, DML, DCL and TCL Commands

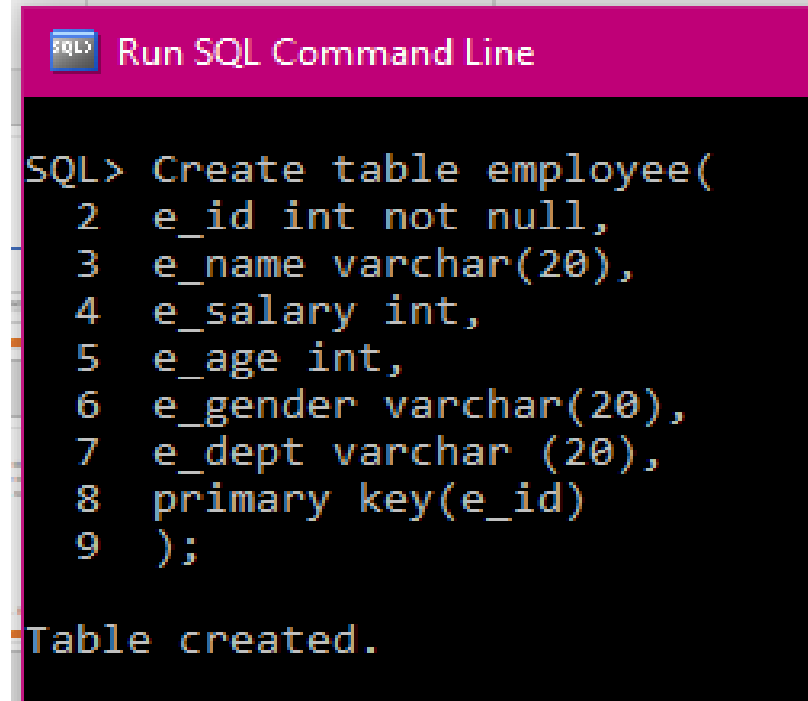
- ▶ Structured Query Language(SQL) is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database.
- ▶ These SQL commands are mainly categorized into four categories as:
  1. DDL - Data Definition Language
  2. DQL - Data Query Language
  3. DML - Data Manipulation Language
  4. DCL - Data Control Language
- ▶ Though many resources claim there to be another category of SQL clauses **TCL - Transaction Control Language**.

# SQL | DDL, DML, DCL and TCL Commands



# 1. DDL Statements

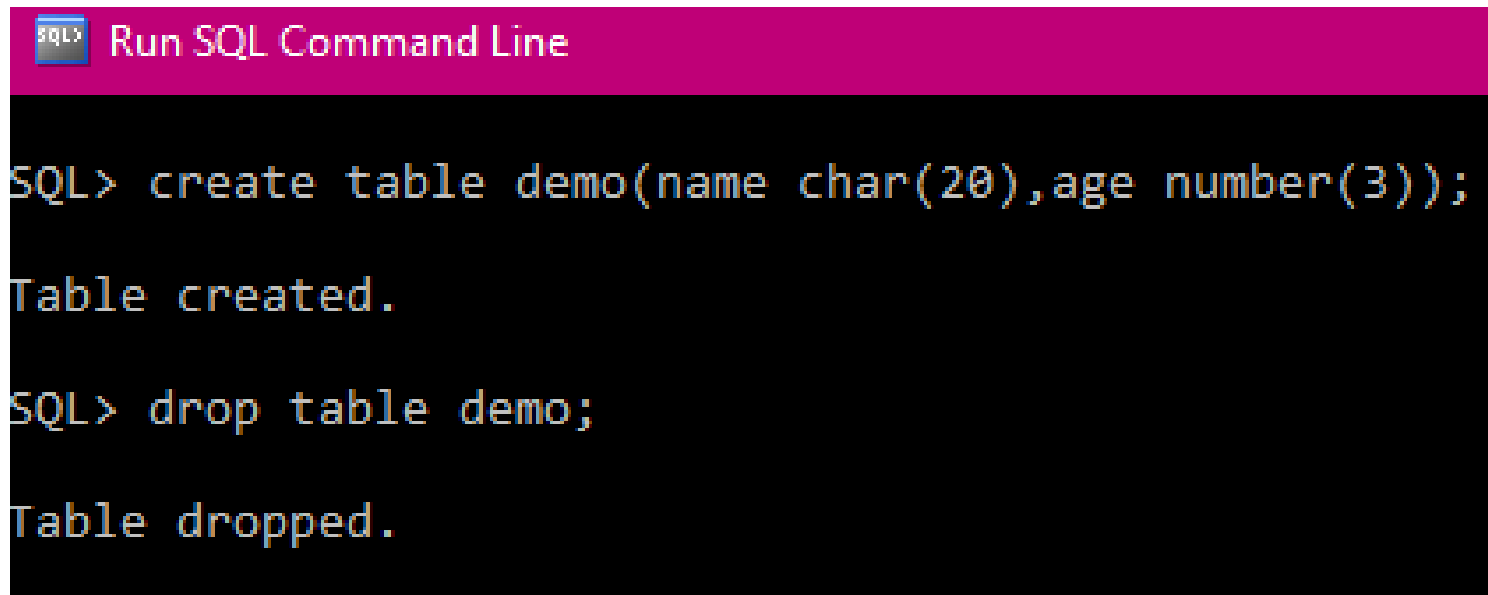
- **CREATE:** This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).

A screenshot of a terminal window titled "Run SQL Command Line". The window has a black background with white text. The text shows a SQL command to create a table named 'employee' with columns 'e\_id', 'e\_name', 'e\_salary', 'e\_age', and 'e\_gender', and a primary key on 'e\_id'. The command is entered line by line, and the output "Table created." is shown at the bottom.

```
SQL> Create table employee(  
2 e_id int not null,  
3 e_name varchar(20),  
4 e_salary int,  
5 e_age int,  
6 e_gender varchar(20),  
7 e_dept varchar (20),  
8 primary key(e_id)  
9 );  
  
Table created.
```

# 1. DDL Statements

- **DROP:** This command is used to delete the complete table from database.



```
SQL> create table demo(name char(20),age number(3));  
Table created.  
  
SQL> drop table demo;  
Table dropped.
```



# 1. DDL Statements

## ► ALTER:

### 1. Adding a Column

To existing table.

```
SQL> select * from doctor;
```

DOC_ID	DOC_NAME	DOC_AGE
1	Dr.Suresh_Mohodik	44
2	Dr.Milind_Mandlik	38
3	Dr.B.Bose	62

```
SQL> alter table doctor add Doc_spc varchar2(7);
```

Table altered.

```
SQL> desc doctor;
```

Name	Null?	Type
DOC_ID	NOT NULL	NUMBER(3)
DOC_NAME		CHAR(20)
DOC_AGE		NUMBER(3)
DOC_SPC		VARCHAR2(7)

# 1. DDL Statements

```
Run SQL Command Line

SQL> desc studentinfo;
Name                               Null?   Type
-----
ROLLNO                             NOT NULL NUMBER(4)
SNAME                              CHAR(15)
MARKS                               NUMBER(3)
COLG_NAME                           CHAR(10)

SQL> alter table studentinfo drop column colg_name;

Table altered.

SQL> desc studentinfo;
Name                               Null?   Type
-----
ROLLNO                             NOT NULL NUMBER(4)
SNAME                              CHAR(15)
MARKS                               NUMBER(3)
```

- ▶ **ALTER:**
- ▶ Deleting a Column from existing table.

# 1. DDL Statements

- ▶ ALTER:
- ▶ Renaming Table and  
Renaming
- ▶ Column.

```
SQL> Run SQL Command Line
```

101	Disha	56	2000	Pune
102	Charu	65	2000	Mumbai
103	Himanshu	98	2000	Delhi
104	Ritu	85	2000	Nagpur
105	Parul	93	2000	Nagpur

```
SQL> alter table sinfo rename to student;
Table altered.
SQL> select * from student;
```

ROLLNO	SNAME	MARKS	YEAR	CITY
101	Disha	56	2000	Pune
102	Charu	65	2000	Mumbai
103	Himanshu	98	2000	Delhi
104	Ritu	85	2000	Nagpur
105	Parul	93	2000	Nagpur

```
SQL> alter table student rename column marks to totalmarks;
Table altered.
SQL> alter table student rename column totalmarks to marks;
Table altered.
```



## 2. DML Statements - insert

Select Run SQL Command Line

```
SQL> insert into employee values(1,'Rahul',20000,26,'male','Procurment');
```

```
1 row created.
```

```
SQL> insert into employee values(2,'Dev',30000,36,'male','Operations');
```

```
1 row created.
```

```
SQL> insert into employee values(2,'Dev',30000,36,'male','Operations');
```

```
insert into employee values(2,'Dev',30000,36,'male','Operations')
```

```
*
```

```
ERROR at line 1:
```

```
ORA-00001: unique constraint (SYSTEM.SYS_C004045) violated
```

```
SQL> insert into employee values(3,'Sonia',25000,28,'female','HR');
```

```
1 row created.
```

```
SQL> insert into employee values(4,'Hema',25000,32,'female','Procurment');
```

```
1 row created.
```

```
SQL> insert into employee values(5,'Himdevi',45000,35,'female','Operations');
```

```
1 row created.
```

## 2. DML Statements

```
SQL> select e_name, e_id from employee;

E_NAME          E_ID
-----
Dev              2
Sonia            3
Hema             4
Himdevi          5

SQL> delete from employee where e_id=4;

1 row deleted.

SQL> select e_name, e_id from employee;

E_NAME          E_ID
-----
Dev              2
Sonia            3
Himdevi          5
```

- ▶ **DELETE Query:**
- ▶ Used to delete or remove one or more existing records from a table.
- ▶ Used along with the Where clause.
- ▶ Though, the Where condition is optional in the Delete query.

### 3. DQL Statements - Select

```
SQL> Run SQL Command Line
SQL> select * from employee;
```

E_ID	E_NAME	E_SALARY	E_AGE	E_GENDER
1	Rahul	20000	26	male
2	Dev	30000	36	male
3	Sonia	25000	28	female
4	Hema	25000	32	female
5	Himdevi	45000	35	female

### 3. DQL Statements

```
SQL> Select distinct e_gender from employee;
```

```
E_GENDER
```

```
-----
```

```
male
```

```
female
```

```
SQL> Select e_gender from employee;
```

```
E_GENDER
```

```
-----
```

```
male
```

```
male
```

```
female
```

```
female
```

```
female
```

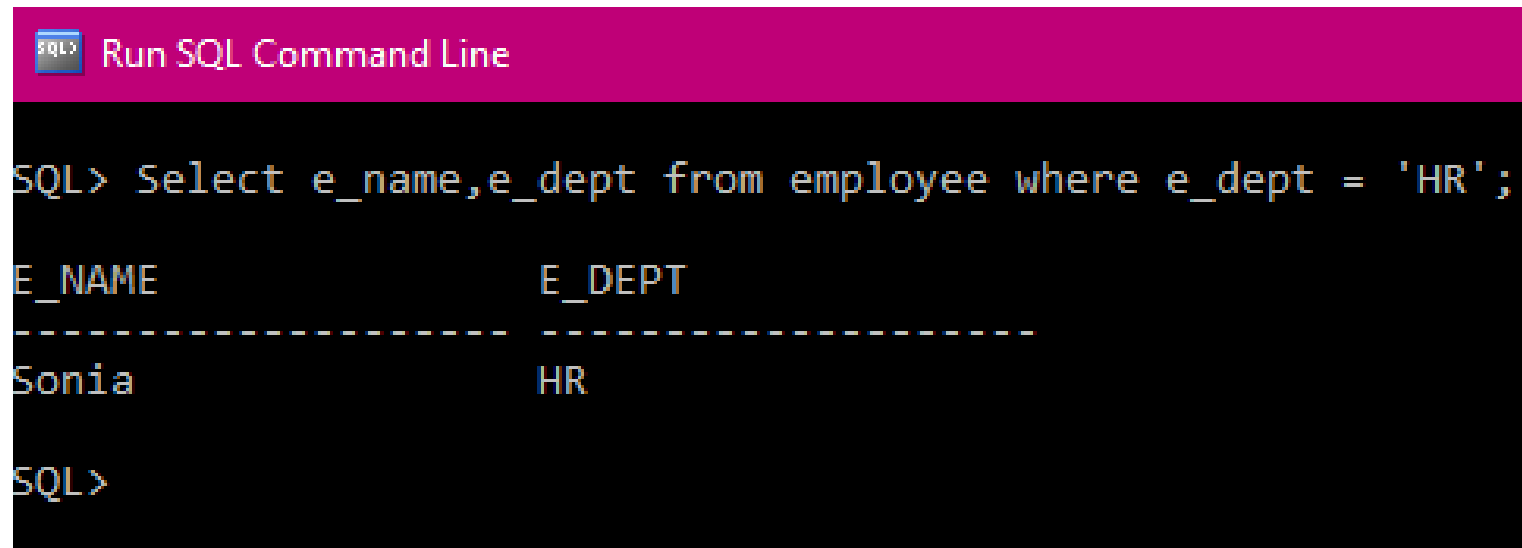
► **SELECT with Distinct:**

► Used to retrieve records that are repeated in the column with their distinct values from the table.



### 3. DQL Statements

- ▶ **SELECT with Where clause:** This command is used to retrieve records that are GIVEN by the where clause from the table.



```
SQL> Run SQL Command Line

SQL> Select e_name,e_dept from employee where e_dept = 'HR';

E_NAME          E_DEPT
-----
Sonia           HR

SQL>
```

The screenshot shows a terminal window titled "Run SQL Command Line". It displays an SQL query: `SQL> Select e_name,e_dept from employee where e_dept = 'HR';`. The output is a table with two columns, `E_NAME` and `E_DEPT`, separated by a dashed line. The data row shows `Sonia` in the `E_NAME` column and `HR` in the `E_DEPT` column. The prompt `SQL>` is shown at the bottom.

### 3. DQL Statements : SELECT

- **SELECT with Where clause & And operator** : The AND operator displays only those records where all conditions are evaluated to true. For example, if you want to find out all the doctors aged greater than 60, the syntax would be as follows:

```
SQL> Select e_name,e_age,e_dept from employee where e_dept = 'HR' AND e_age<40;
```

E_NAME	E_AGE	E_DEPT
Sonia	28	HR

```
SQL> Select e_name,e_age,e_dept from employee where e_dept = 'Procurment' AND e_age<40;
```

E_NAME	E_AGE	E_DEPT
Rahul	26	Procurment
Hema	32	Procurment

```
SQL>
```

### 3. DQL Statements : SELECT

- ▶ **SELECT with Where clause and OR operator** : displays records for any condition separated by OR which is evaluated to true. E.g., to filter out a doctor with his age concern, use the OR operator.

```
SQL> Run SQL Command Line

SQL> select * from doctor;

  DOC_ID  DOC_NAME          DOC_AGE
-----
      1  Dr.Suresh_Mohodik      44
      2  Dr.Milind_Mandlik      38
      3  Dr.Bose                  62

SQL> Select doc_name from doctor where doc_age<30 OR doc_age>60;

DOC_NAME
-----
Dr.Bose
```

### 3. DQL Statements : SELECT

- ▶ **SELECT with Where clause and NOT operator** : The NOT operator displays a record if the condition is not true. E.g., To extract all records where the occupation of a person is not a software engineer, the Not operator is used.

```
SQL> select e_name,e_dept from employee where not e_salary=25000;

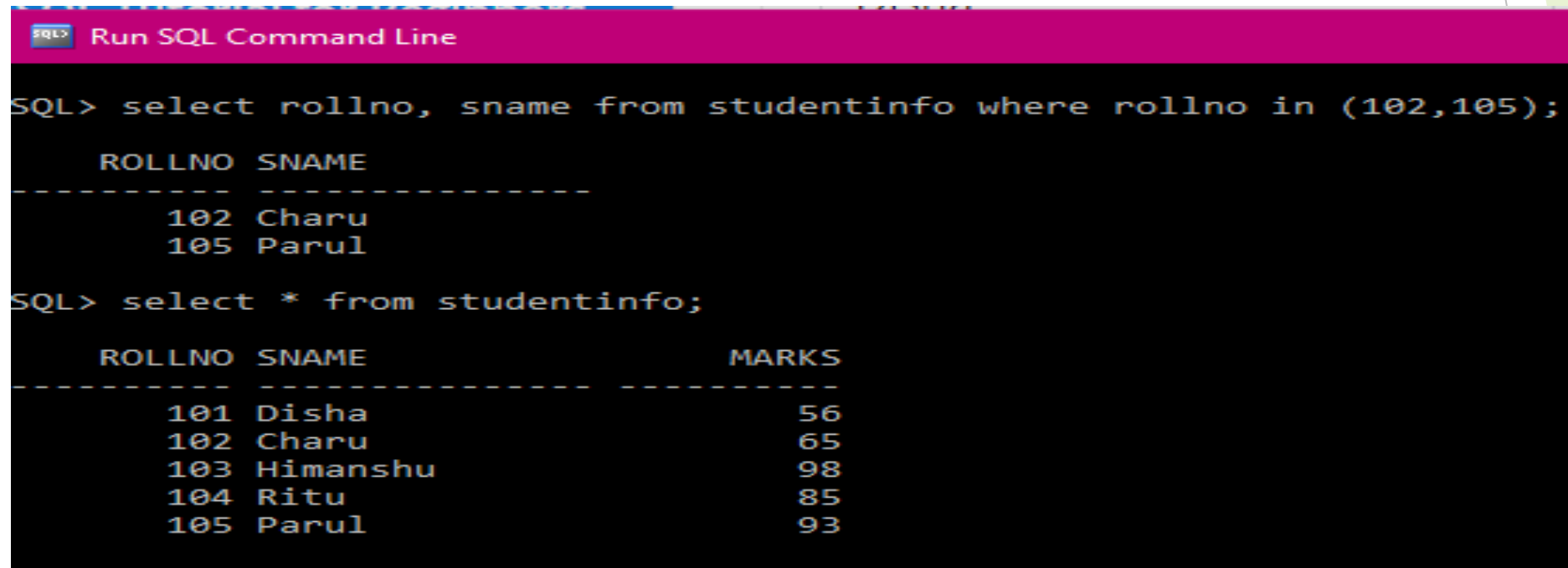
E_NAME          E_DEPT
-----
Rahul           Procurement
Dev             Operations
Himdevi         Operations

SQL> select e_name,e_dept from employee where not e_gender='female';

E_NAME          E_DEPT
-----
Rahul           Procurement
Dev             Operations
```

### 3. DQL Statements : SELECT

- ▶ **SELECT with Where clause and IN operator** : IN clause is used with the WHERE clause to specify discrete values.



```
SQL> select rollno, sname from studentinfo where rollno in (102,105);
```

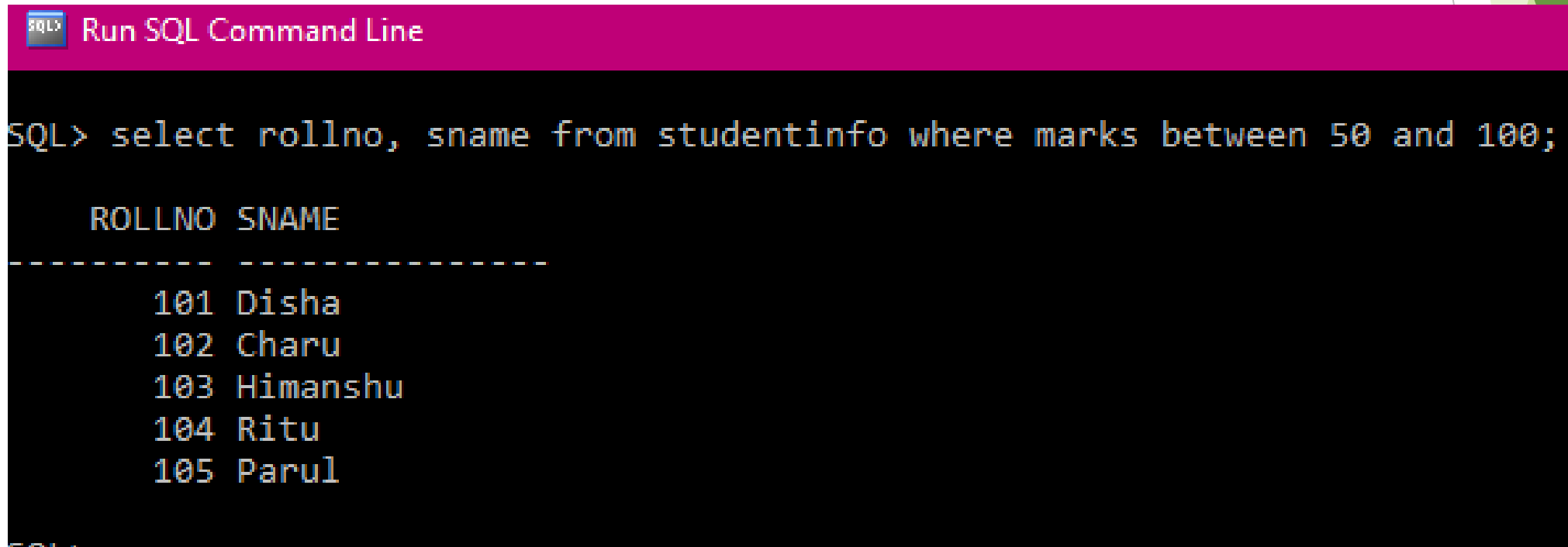
ROLLNO	SNAME
102	Charu
105	Parul

```
SQL> select * from studentinfo;
```

ROLLNO	SNAME	MARKS
101	Disha	56
102	Charu	65
103	Himanshu	98
104	Ritu	85
105	Parul	93

### 3. DQL Statements : SELECT

- ▶ **SELECT with Where clause and Between operator** : It is used to select a value within the specified range.



The screenshot shows a terminal window titled "Run SQL Command Line". The prompt is "SQL>". The command entered is "select rollno, sname from studentinfo where marks between 50 and 100;". The output is a table with two columns: "ROLLNO" and "SNAME". The data rows are: 101 Disha, 102 Charu, 103 Himanshu, 104 Ritu, and 105 Parul.

```
SQL> select rollno, sname from studentinfo where marks between 50 and 100;
```

ROLLNO	SNAME
101	Disha
102	Charu
103	Himanshu
104	Ritu
105	Parul

# What are Constraints?

- ▶ Constraints are the rules enforced on the data stored in the database.
- ▶ Constraints can be applied to already existing tables as well as while at creation time.
- ▶ Constraints can be applied at the column level as well as the table level.

```
CREATE TABLE sample (  
    sample_name varchar(255) DEFAULT 'Unknown sample'  
);
```

# Types of Constraints

- ▶ Not Null
- ▶ Unique
- ▶ Primary Key
- ▶ Foreign Key
- ▶ Check
- ▶ Create Index



# How to Create Constraints?

- ▶ We can impose constraints on the table either at the time of the creation of the table or after the creation of the table. There are two ways of imposing Constraints in SQL:
- ▶ At the time of creation, we only must add the constraint's name to impose the specified constraint.
- ▶ After the creation, we need to use **ALTER TABLE** and then the constraint's name.

# Syntax of Constraints in SQL

- ▶ Constraints can be imposed at two different times namely -
  - ▶ at the time of creation and
  - ▶ after creation.
- ▶ Constraint imposed at the time of table creation using CREATE TABLE command:

- ▶ **Syntax**

```
CREATE TABLE table_name  
(  
  cloumn_name date_type(size) constraint_name,  
  .....  
)
```

- ▶ Constraint imposed at the time of table modification using ALTER TABLE command:

- ▶ **Syntax**

```
ALTER TABLE table_name  
(  
  MODIFY cloumn_name date_type(size) constraint_name,  
  .....  
)
```

# Unique Constraint

- ▶ UNIQUE constraint can be used to ensure that no two records in the column have the same value.
- ▶ But you can have one null value for a table.

1. Constraint define at the time of creating a table.

```
CREATE TABLE sample  
(  
  sample-number int UNIQUE,  
  .....  
)
```

2. Constraint define at the time of altering a table.

```
ALTER TABLE sample  
(  
  MODIFY sample-number int UNIQUE;  
)
```

# Not Null Constraint

- If one wants to ensure that a record is not added to a table unless some specific fields have some value, then NOT NULL constraint is used.

Empname column must have value so this column will be NOT NULL

Empno	Empname	Designation	Salary	Deptno
E001	Vandana	Manager	20000	10
E002	Amit	Executive	14000	20
E003	Amit	Accountant	10000	30
E004	Anju	Clerk	9000	10

# Not null Constraint

- Define Not null constraint

```
CREATE TABLE table_name (  
    ...  
    column_name data_type NOT NULL  
    ...  
);
```

- Alter table for Not Null constraint

```
ALTER TABLE table_name MODIFY ( column_name NOT NULL);
```

# Primary Key Constraints

- ▶ No two records can have the same value for the column.
- ▶ NOT NULL is automatically applied to the column.
- ▶ An index is created on the column. This results into quicker retrieval of records.

Primary  
Key

Empno	Empname	Desig	Salary	Deptno
E001	Vandana	Manager	20000	10
E002	Amit	Executive	14000	20
E003	Amit	Accountant	10000	30
E004	Anju	Clerk	9000	10

# Foreign Key Constraints or References Constraints

- ▶ Values of the foreign key
  - ▶ must be either null, or
  - ▶ if non-null, must match with the primary key value of some record of the Master relation.

DEPT	
Deptno	Dname
10	Admin
20	IT
30	Account
Attributes	

EMP				
Empno	Ename	Desig	Salary	Deptno
E001	Vandana	Manager	20000	10
E002	Amit	Executive	14000	20
E003	Amit	Accountant	10000	30
Attributes				

Foreign  
Key



# Check Constraint

- Restrict a column value to a set of values defined by the constraint.

Check designations only in  
'Manager, Executive,  
Clerk, Accountant etc.'

Empno	Empname	Designation	Salary	Deptno
E001	Vandana	Manager	20000	10
E002	Amit	Executive	14000	20
E003	Amit	Accountant	10000	30
E004	Anju	Clerk	9000	10

# Oracle Drop Table

- ▶ Oracle DROP TABLE statement is used to remove or delete a table from the Oracle database.
- ▶ Syntax
- ▶ DROP [schema\_name].TABLE table\_name
- ▶ [ CASCADE CONSTRAINTS ]
- ▶ [ PURGE ];
- ▶ Parameters
- ▶ schema\_name: It specifies the name of the schema that owns the table.
- ▶ table\_name: It specifies the name of the table which you want to remove from the Oracle database..

# Single Row Functions

- ▶ As the name suggests, these functions operate on a single row and return one result for each row.
- ▶ They are many types and they are as follows:
  - **Number:** trunc, round, mod, ceil, log, floor, sign
  - **Date:** months\_between, sysdate, add\_months, next\_day, last\_day, trunc, current\_date, to\_timestamp, dbtimezone
  - **Character:** upper, lower, initcap, concat, length, lengthb, instr, lpad, ltrim, replace, substr, substrb, replace, soundex, translate, trim
  - **Conversion:** to\_char, to\_number, to\_date
- ▶ These functions can be used with Select, Where, Order by, Set clause and Update command.

# Aggregate Functions

- ▶ An SQL Aggregate function calculates on a set of values and returns a single values.
- ▶ For example, the avg function takes a list of values and returns the average.
- ▶ Since, aggregate functions operated on a set of values, it is often used with group by clause of Select Statement.
- ▶ The group by clause divides the result set into groups of values and the aggregate function returns a single value for each group.
- ▶ The types of aggregate functions are: min, max, avg, count and sum.

# The LIKE Operator

The LIKE condition uses pattern matching to restrict rows in a SELECT statement.

```
SELECT employee_id, last_name FROM employees WHERE last_name  
LIKE 'Fa%';
```

- ▶ The pattern should be enclosed in single quotes (' ').
- ▶ The percent sign (%) acts as a wildcard for one or more characters
- ▶ The asterisk (\*) acts as many or all characters .
- ▶ The single underscore character (\_) acts as a wildcard for one and only one character.

# A COLUMN ALIAS

- ▶ Renames a column heading
- ▶ Is useful with calculations
- ▶ Immediately follows the column name (There can also be the optional AS keyword between the column name and alias.)
- ▶ Requires double quotation marks (“ ”) if it contains spaces or special characters, or if it is case-sensitive

```
SELECT last_name AS Surname, commission_pct commission FROM employees  
WHERE commission_pct IS NOT NULL;
```

# Oracle Data Types

# Oracle Built-in Datatypes

**CHAR** (*size*)

fixed-length char array

**VARCHAR2** (*size*)

Variable-length char string

**NUMBER** (*precision, scale*)

any numeric

**DATE**

date

**TIMESTAMP**

date+time

**CLOB**

char large object

**BLOB**

binary large object

**BINARY\_FLOAT**

32 bit floating point

**BINARY\_DOUBLE**

64 bit floating point ... + *some others*



# Conversion Functions

- ▶ The most common of these functions are:
- ▶ **TO\_CHAR** function converts a floating number to a string.
- ▶ **TO\_NUMBER** function converts a floating number or a string to a number.
- ▶ **TO\_DATE** function converts character data to a DATE data type.
- ▶ **SELECT TO\_CHAR(TO\_DATE('20-JUL-08', 'DD-MON-RR') , 'YYYY') "Year"**  
**FROM DUAL;**
- ▶ **SELECT TO\_CHAR(SYSDATE, 'DD-MON-YYYY') FROM DUAL;**

# Character Functions :

Function	Description
CONCAT	Allows you to concatenate two strings together
CONVERT	Converts a string from <u>one character</u> set to another
INITCAP	Sets the first character in each word to uppercase and the rest to lowercase
INSTR	Returns the location of a substring in a string
LENGTH	Returns the length of the specified string
LOWER	Converts all letters in the specified string to lowercase
LPAD	Pads the left-side of a string with a specific set of characters
LTRIM	Removes all specified characters from the left-hand side of a string
REGEXP_INSTR	Returns the location of a regular expression pattern in a string
REGEXP_REPLACE	Allows you to replace a sequence of characters in a string with another set of characters using regular expression pattern matching
REGEXP_SUBSTR	Allows you to extract a substring from a string using regular expression pattern matching

# Character Functions :

REPLACE	Replaces a sequence of characters in a string with another set of characters
RPAD	Pads the right-side of a string with a specific set of characters
RTRIM	Removes all specified characters from the right-hand side of a string
SUBSTR	Allows you to extract a substring from a string
TRANSLATE	Replaces a sequence of characters in a string with another set of characters
TRIM	Removes all specified characters either from the beginning or the end of a string
UPPER	Converts all letters in the specified string to uppercase
VSIZE	Returns the number of bytes in the internal representation of an expression

# Number Functions

Function	Description
COUNT	Returns the count of an expression
EXP	Returns $e$ raised to the power of number
GREATEST	Returns the greatest value in a list of expressions
LEAST	Returns the smallest value in a list of expressions
MOD	Returns the remainder of n divided by m
POWER	Returns m raised to the nth power
ROUND	Returns a number rounded to a certain number of decimal places
SQRT	Returns the square root of a number
SUM	Returns the summed value of an expression
TRUNC	Returns a number truncated to a certain number of decimal places

# Date Functions

Function	Description
ADD_MONTHS	Returns a date with a specified number of months added
CURRENT_DATE	Returns the current date in the time zone of the current SQL session as set by the ALTER SESSION command
ROUND	Returns a date rounded to a specific unit of measure
LAST_DAY	Returns the last day of the month based on a date value
MONTHS_BETWEEN	Returns the number of months between date1 and date2
NEXT_DAY	Returns the first weekday that is greater than a date
SYSDATE	Returns the current system date and time on your local database
SYSTIMESTAMP	Returns the current system date and time (including fractional seconds and time zone) on your local database
TRUNC	Returns a date truncated to a specific unit of measure

# Joins in SQL

- ▶ Are queries used to combine records of two or more tables in a database.
- ▶
- ▶ A Join means combining or joining two field from two tables, which are common values to both the tables.

# Example of Join

Now type in the following query:

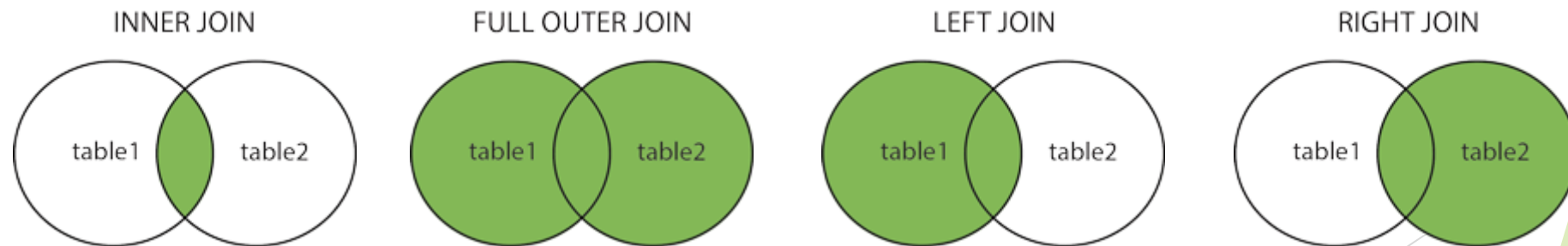
```
SELECT ID, NAME, AGE, AMOUNT FROM CUSTOMER, ORDER WHERE CUSTOMER.ID =  
ORDER.CUSTOMER ID.
```

It will produce the following result:

ID	Name	Age	Amount
3	Om	23	3000.00
3	Om	23	4000.00
2	Shalini	25	2000.00
4	Minal	25	2500.00

# Types of Joins

- ▶ Inner Join: Returns records which have matching values in both the table.
- ▶ Outer or Full Join: Returns all records when there is a match in either right or left table.
- ▶ Left Outer Join: Returns all records from the left table and matching records from right table.
- ▶ Right Outer Join: Returns all records from the right table and matching records from left table.





# Types of Joins

- ▶ Self Join: is used to join a table to itself as if the table were two tables.
- ▶ Cross Join: returns cartesian product of set of records from two or more joined tables.
- ▶ Equi Joins: performs joins using '=' operator
- ▶ Non-Equi or Theta Join: performs join using comparing operators apart from '=' operator like <, >, <=, >=.

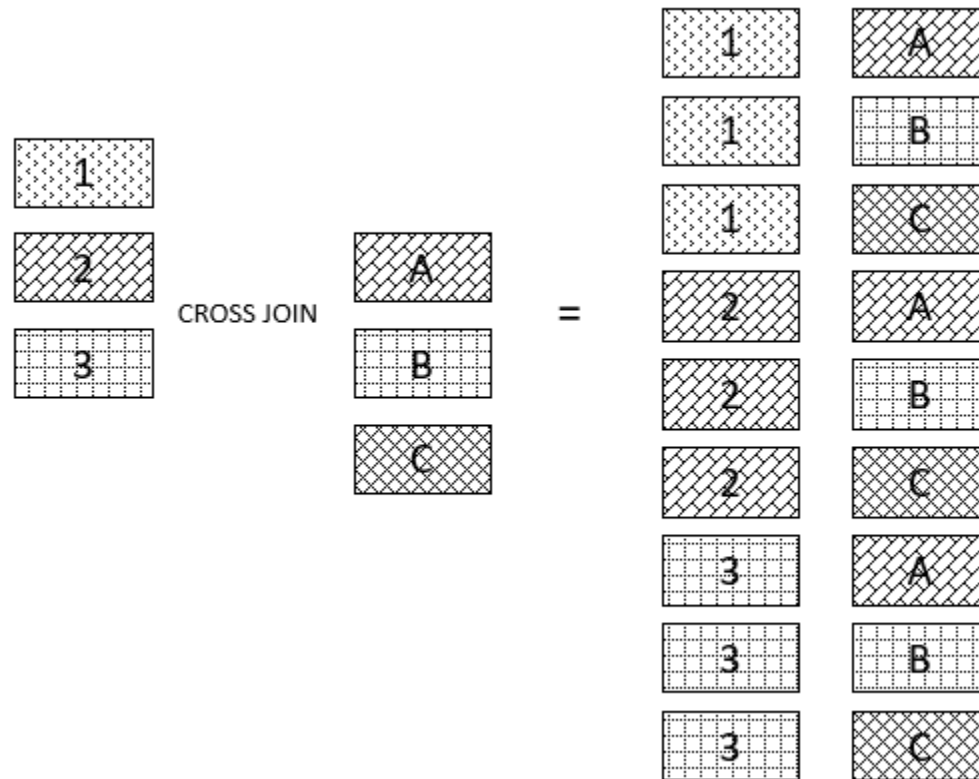
# Cross Join

- ▶ This join will join two unrelated tables.
- ▶ In other words, cross join returns cartesian product of rows from both the tables.
- ▶ Unlike the Inner Join or Left Join, the cross join does not establish a relationship between the joined tables.
- ▶ Syntax for Cross Join is as follows:

```
SELECT column_name(s)
```

```
FROM table1, table2
```

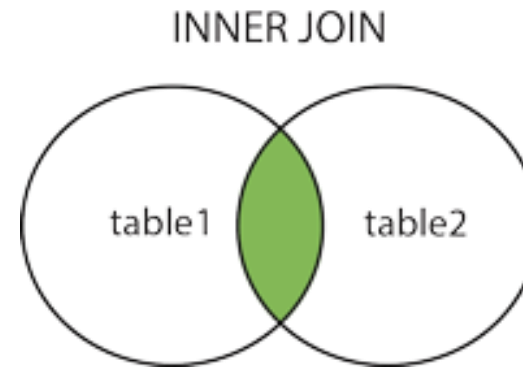
# What does Cartesian product mean?



# Inner or Natural Join

- ▶ INNER JOIN keyword selects the records that have matching values in both the tables.
- ▶ INNER JOIN syntax is as follows:

```
SELECT column_name(s)  
FROM table1  
INNER JOIN table2  
ON table1.column_name = table2.column_name
```



Consider the following example:

Customer Table				
ID	Name	Age	Address	Salary
1	Anjali	32	Mumbai	3000.00
2	Shalini	25	Bhopal	3500.00
3	Om	23	Kolkata	1500.00
4	Minal	25	Bangalore	2500.00
5	Kapil	27	Pune	3700.00
6	Karishma	22	Delhi	1500.00
7	Pooja	24	Indore	2000.00

Order Table			
OID	Date	Customer_ID	Amount
102	10/02/2020	3	300.00
100	15/05/2021	3	400.00
101	17/10/2020	2	200.00
103	31/01/2021	4	250.00

► `SELECT ID, Name, Age`  
`FROM Customer`  
`INNER JOIN Order ON Order.ID = Customer.ID`

ID	Name	Age	Amount
3	Om	23	400.00
3	Om	23	400.00
2	Shalini	25	300.00
4	Minal	25	250.00

# Left Join

- ▶ Returns all rows from the left table even if there are no matches in the right table.
- ▶ In other words, it returns all rows from the left table and all the matching rows from the right. If no matching row is found in the right NULL is used.

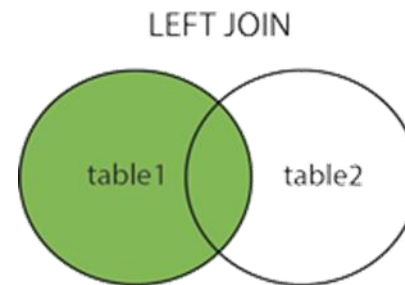
- ▶ Syntax of Left Join is as follows:

*SELECT column\_name(s)*

*FROM table1*

*LEFT JOIN table2*

*ON table1.matching\_column\_name = table2.matching\_column\_name*



# Right Join

- ▶ Returns all rows from the right table even when there is no match in the left table.
- ▶ If a row in the right table does not have any matching rows from the left table, the column of the left table in the result set will have NULLs.

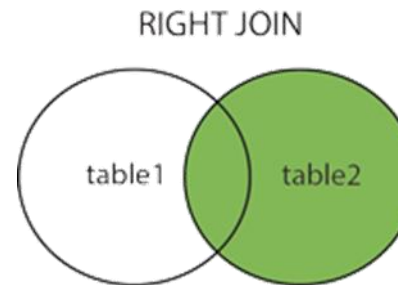
- ▶ Syntax for right join is as follows:

*SELECT column\_name(s)*

*FROM table1*

*RIGHT JOIN table2*

*ON table1.matching\_column\_name = table2.matching\_column\_name*





# Outer or Full Join

- ▶ Combines results of left and right join
- ▶ When no matching rows exist for the row in the left table, the columns of the right table will have nulls. Similarly, when no matching rows exist for the row in the right table, the column of the left table will have nulls.

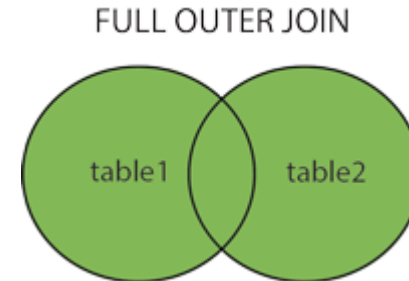
- ▶ Syntax for Outer or Full Join is as follows:

`SELECT column_name(s)`

`FROM table1`

`FULL JOIN table2`

`ON table1.matching_column_name = table2.matching_column_name`



# Self Join

- ▶ Self join is used to join a table with itself as if the table were a two table.
- ▶ It is useful for querying hierarchical data or comparing rows within the same table.
- ▶ A self join uses the Inner join or left join clause.

- ▶ Syntax for Self Join is as follows:

```
SELECT column_name(s)  
FROM table1, table2  
WHERE some_condition
```

# Subqueries or Nested Queries

- ▶ Are queries which have queries nested within them and embedded with the WHERE clause.
- ▶ used to return data which will be used in the main query as a condition to further restrict the data to be retrieved.
- ▶ Subqueries can be used with the
- ▶ SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

```
SELECT
    order_id,
    order_date,
    customer_id
FROM
    sales.orders
WHERE
    customer_id IN (
        SELECT
            customer_id
        FROM
            sales.customers
        WHERE
            city = 'New York'
    )
ORDER BY
    order_date DESC;
```

outer query

subquery

# When to use a Subquery?

- ▶ Compare an expression to the result of the query
- ▶ Determine if an expression is included in the results of the query
- ▶ Check whether the query selects any rows

Thank You

# Autoincrement - Primary Key

- ▶ Consider a Student table having Id and name columns and we want to autoincrement the id column
- ▶ Create tblstudent (stud\_id number, stud\_name varchar(20)
- ▶ Constraint stud\_pk primary key)
  
- ▶ In Oracle 12g we can use the following
- ▶ Create tblStudent(stud\_id number Generated always as Identity
- ▶ [start with 1 increment by 1])

# Autoincrement - Primary Key

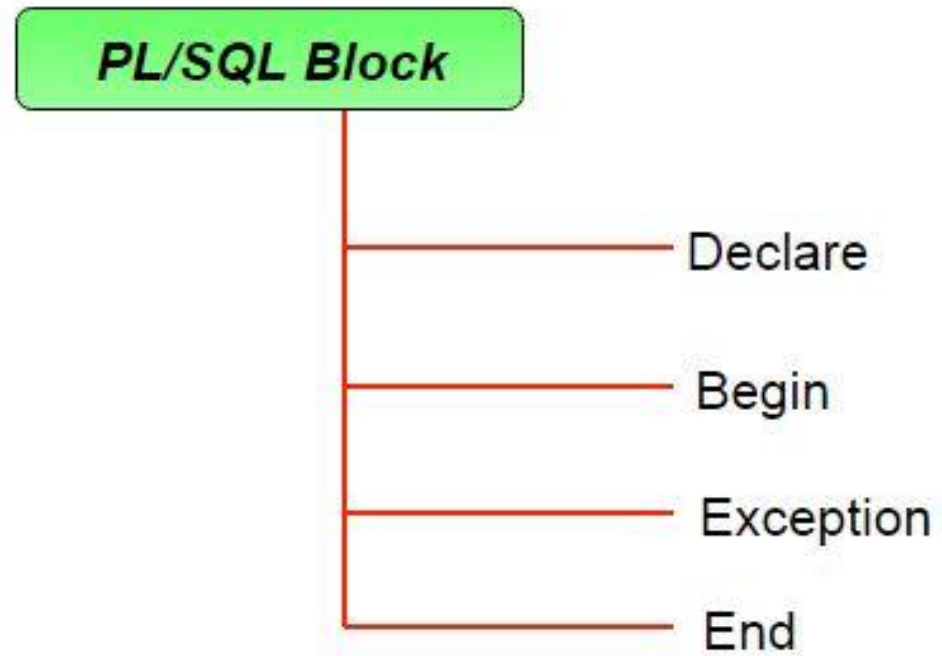
- ▶ Create a sequence - object in oracle that generates unique numbers
- ▶ Create a BEFORE INSERT trigger that uses this sequence
- ▶ Example:
  - ▶ Create SEQUENCE stud\_seq; OR create SEQUENCE stud\_seq START WITH 100 INCREMENT BY 1;
- ▶ Sequence starts with 1 and increments 1 by default
- ▶ Else specify start and increment

# Autoincrement - Primary Key

- ▶ Create or Replace Trigger stud\_trg
- ▶ Before Insert on tblStudent
- ▶ For each row
- ▶ When (new.stud\_id is null)
- ▶ Begin
  - ▶ Select stud\_seq.NEXTVAL
  - ▶ Into :new.id
  - ▶ From dual
- ▶ End
- ▶ //dual is a table in Oracle that has only one row, it's a dummy row. Many expressions, date etc. is obtained using this table as From clause is mandatory in Oracle



# PL/SQL



# PL/SQL Stored Procedure

The screenshot displays the Oracle SQL Developer environment. On the left, a 'Procedures' tree shows 'TESTPROC' selected. The main window, titled 'TESTPROC', has tabs for 'Code', 'Dependencies', 'Errors', and 'Grants'. The 'Code' tab is active, showing a message: 'PL/SQL code successfully compiled (10:32:17)'. Below the message, the PL/SQL code for the procedure is visible:

```
1 create or replace procedure TestProc
2   IS
3     result TESTONE.NAME%TYPE;
4   BEGIN
5     SELECT NAME
6     INTO result
7     FROM TESTONE
8     WHERE ID= 5;
9   dbms_output.put_line('Final ' || result);
10  EXCEPTION
11    WHEN NO_DATA_FOUND THEN
12      raise_application_error(-20100,'error');
13
14  END;
```

Thank You