Database Management Systems

Data

Definition:

- Called as observations or measurements represented as text, numbers, or multimedia.

Or

- Data is the raw material that can be processed for any computing machine.

Properties:

- Unprocessed
- No meaning

Example:

101	dinesh	2000
102	mahesh	3000

Information

Definition:

- Collection of meaningful data which was generated after processing of data

Or

- Known as processed, organized or summarized data

Properties:

- Relevance
- Accuracy
- Completeness

Example:

StudentId	NameOfTheStudent	pendingFee
101	dinesh	2000
102	mahesh	3000

Data Store

Definition:

- A place where we stores data or information is known as data store

Types:

- Books and Papers
- Flat files
- Database

Drawbacks with Books:

- Physical storage space required
- Not secured
- Modification is tough
- Only short period of time

Drawbacks with flat files:

- Data retrieval
- Data redundancy
- Data security
- Data indexing
- Data integrity

Database

Definition: An organized collection of structured information, or data

Storage Type: Stored in rows and columns in a series of tables

Storage Location: Electronically in a computer system.

Controlled by: Database management system (DBMS)

Purpose: Make processing and data querying efficient and faster.

Language: Most databases use structured query language (SQL) for writing and querying data.

Advantages: The data can then be easily accessed, managed, modified, updated, controlled, and organized.

Database Management

Definition:

- Operations which we can perform on database is known as database management

Types of operations or CRUD operations:

- Create [**C**]
- Retrieval / Read [**R**]
- Update [**U**]
- Delete [**D**]

Database Management Systems

- A database requires a comprehensive database software program known as a database management system (DBMS).

- A DBMS serves as an interface between the database and its end users.

- DBMS allows users to perform CRUD operations and manage how the information is organized and optimized.

- A DBMS also facilitates oversight and control of databases, performance monitoring, tuning, and backup and recovery.

- Some examples of DBMS: MySQL, Microsoft Access, Microsoft SQL Server, FileMaker Pro, Oracle Database, and dBASE.

Database Challenges:

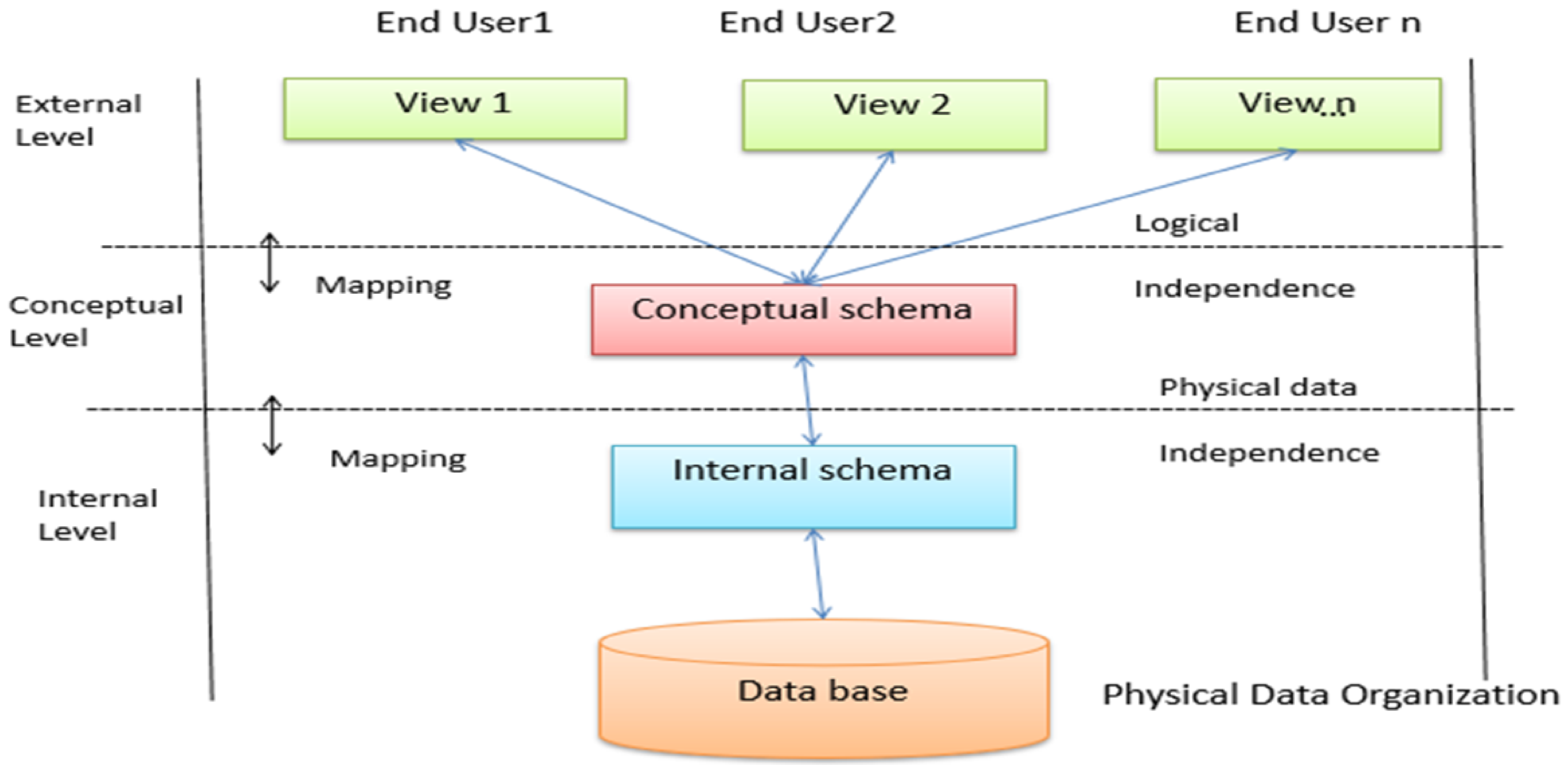
1. Absorbing significant increases in data volume
2. Ensuring data security
3. Keeping up with demand
4. Managing and maintaining the database and infrastructure
5. Removing limits on scalability
6. Ensuring data residency, data authority, or latency requirements

ANSI Database Architecture

ANSI Database Architecture:

The ANSI proposes a three level database architecture, those are

1. External Level / View Level
2. Conceptual Level / Relational Level
3. Internal Level / Physical Level



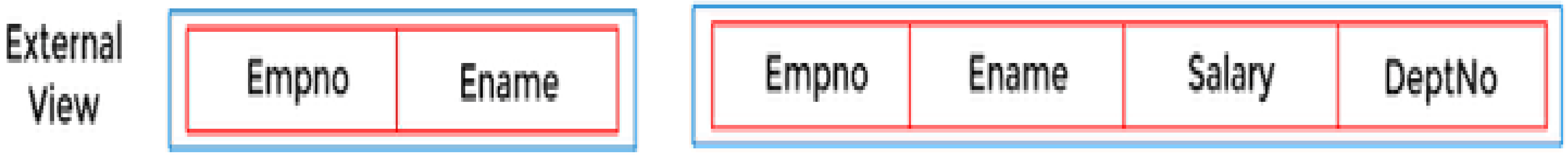
ANSI Database Architecture:

External Level:

- Level describes the database part that a particular user group is interested and hides the remaining database from that user group.

- Access only part of the database

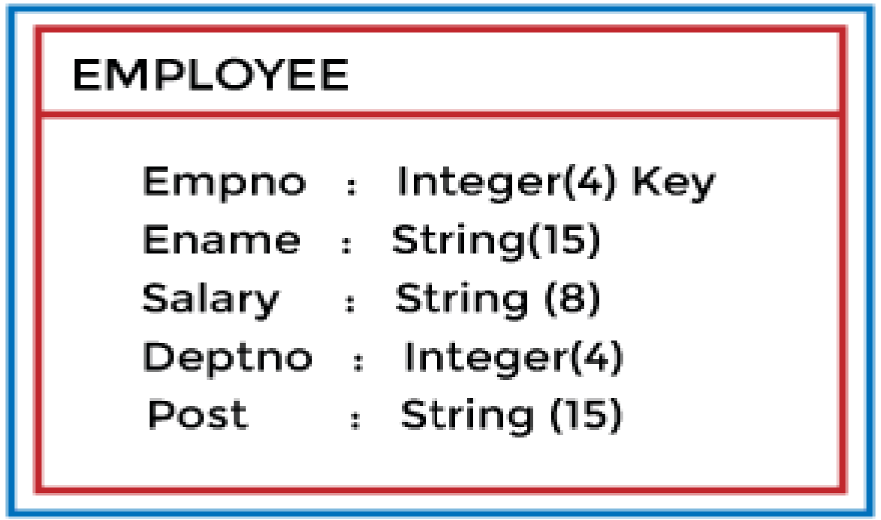
- End users are comes into external level part



ANSI Database Architecture:

Conceptual Level:

- Level describes what data are to be stored in the database and also describes what relationship exists among those data.
- Defines type of data storing in database and also defines what type of data does not store in database using constraints.
- Programmers and database administrators work at this level.



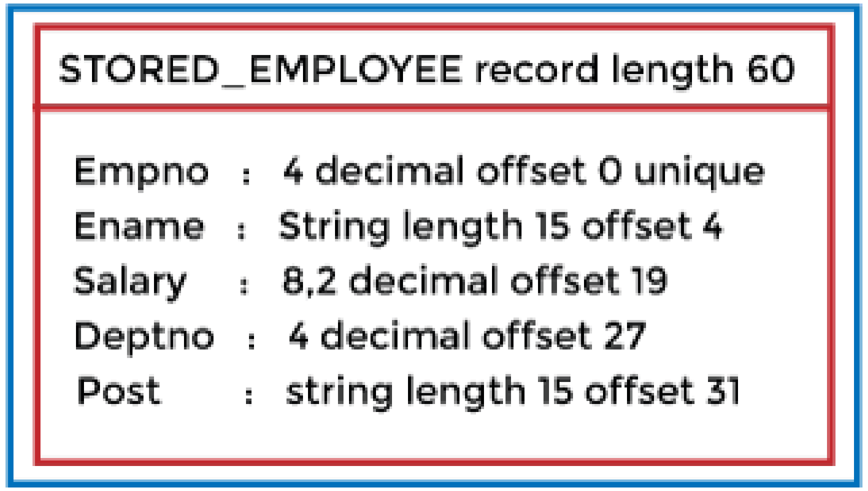
EMPLOYEE

Empno	:	Integer(4) Key
Ename	:	String(15)
Salary	:	String (8)
Deptno	:	Integer(4)
Post	:	String (15)

ANSI Database Architecture:

Internal Level:

- The internal level describes the physical storage structure of the database also known as a physical schema.
- It uses the physical data model. It is used to define that how the data will be stored in a block.
- The physical level is used to describe complex low-level data structures in detail.
- DBA works at this level



STORED_EMPLOYEE record length 60

Empno	:	4 decimal offset 0 unique
Ename	:	String length 15 offset 4
Salary	:	8,2 decimal offset 19
Deptno	:	4 decimal offset 27
Post	:	string length 15 offset 31

Data Models

Data Models

A Database model defines:

- The logical design and structure of a database
- How data will be stored, accessed and updated in a DBMS

Types:

- Hierarchical database model
- Relational model
- Network model
- Object-oriented database model
- Entity-relationship model
- Document model
- Entity-attribute-value model
- Star schema
- The object-relational model, which combines the two that make up its name

Relational Database:

- A relational database is a type of database that stores and provides access to data points that are related to one another.
- Relational databases are based on the relational model, straightforward way of representing data in tables.
- In a relational database, each row in the table is a record with a unique ID called the key.
- The columns of the table hold attributes of the data
- Each record usually has a value for each attribute, making it easy to establish the relationships among data points

Entity-relationship model : Introduction

What is ER – Model?

- The ER Model is a high-level conceptual data model diagram helps to analyze data requirements to produce a well-designed database.

- The ER Model represents real-world / logical world entities and the relationships between them.

- Creating an ER Model in DBMS is considered as a best practice before implementing your database.

- ER diagrams help to explain the logical structure of databases and are created based on three basic concepts:
    - Entities
    - Attributes
    - Relationships.

Entity-relationship model : Introduction

Why use ER Diagrams?

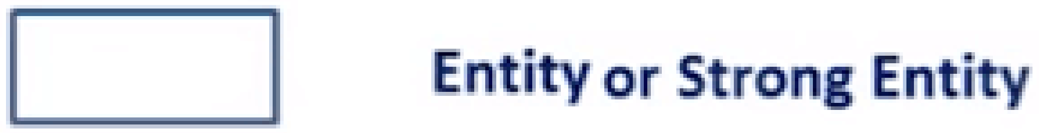
- Helps to describe entities, attributes, relationships
- Provide a preview of how all your tables should connect, what fields are going to be on each table
- ER diagrams are translatable into relational tables which allows you to build databases quickly
- ER diagrams can be used by database designers as a blueprint for implementing data in specific software applications
- The database designer gains a better understanding of the information to be contained in the database with the help of ERP diagram

Entity-relationship model : Entity

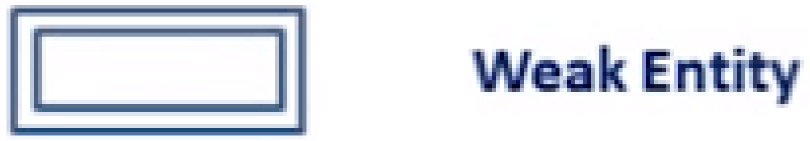
WHAT IS ENTITY?

- An entity is a real or logical world object like place, person, object, event or a concept, which stores data in the database.

- An entity can be represented by using rectangle



- Weak entity can be represented by using double rectangle



Entity Set:

- An entity set is a group of similar kind of entities.

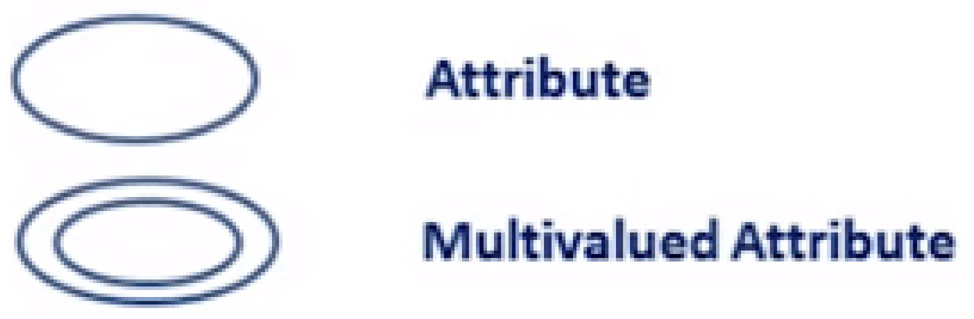
Entity-relationship model : Differences

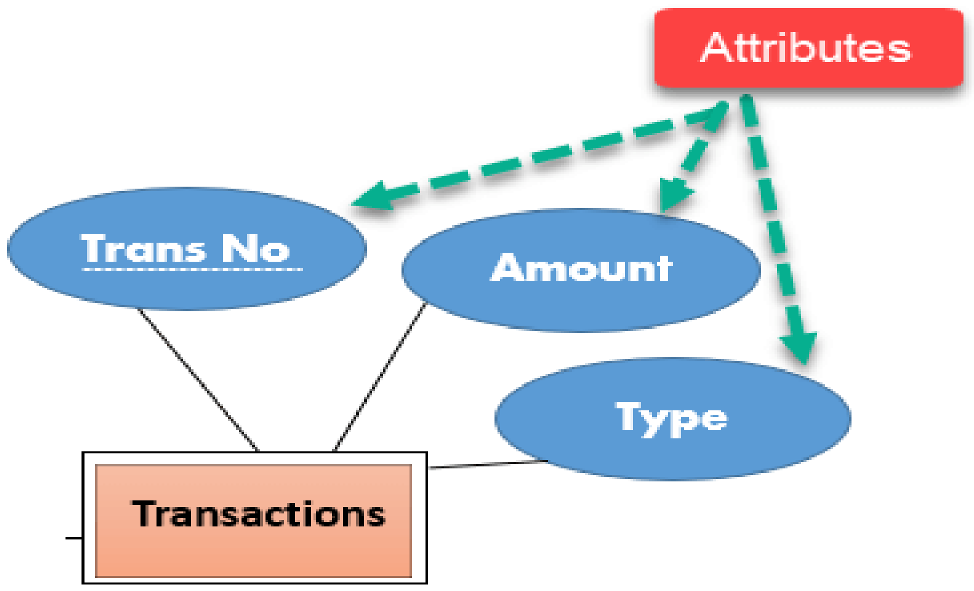
Strong Entity Set	**Weak Entity Set**
Strong entity set always has a primary key.	It does not have enough attributes to build a primary key.
It is represented by a rectangle symbol.	It is represented by a double rectangle symbol.
It contains a Primary key represented by the underline symbol.	It contains a Partial Key which is represented by a dashed underline symbol.
The member of a strong entity set is called as dominant entity set.	The member of a weak entity set called as a subordinate entity set.

Entity-relationship model : Attributes

Attributes:

- It is a property of an entity describing about feature(s) of it.
- Example: A lecture might have attributes: time, date, duration, place.
- An attribute in ER Diagram is represented by an Ellipse





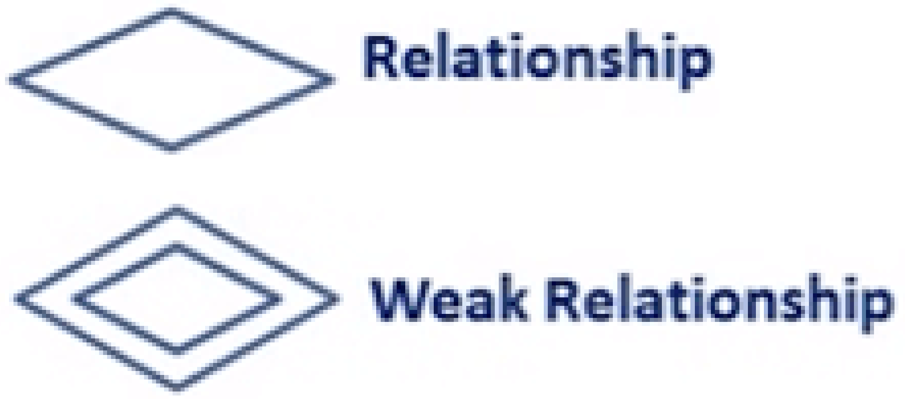
Entity-relationship model : Attribute Types

Types of Attributes	Description
Simple attribute	Can’t be divided any further.
Composite attribute	Break down composite attribute.
Derived attribute	Derived from other attributes present in the database
Multivalued attribute	Multivalued attributes can have more than one values

Entity-relationship model : Relationship

Relationship:

- Relationship is nothing but an association among two or more entities.
    - For Example: Tom works in the Chemistry department.
- A relationship can be represented by using diamond symbol.

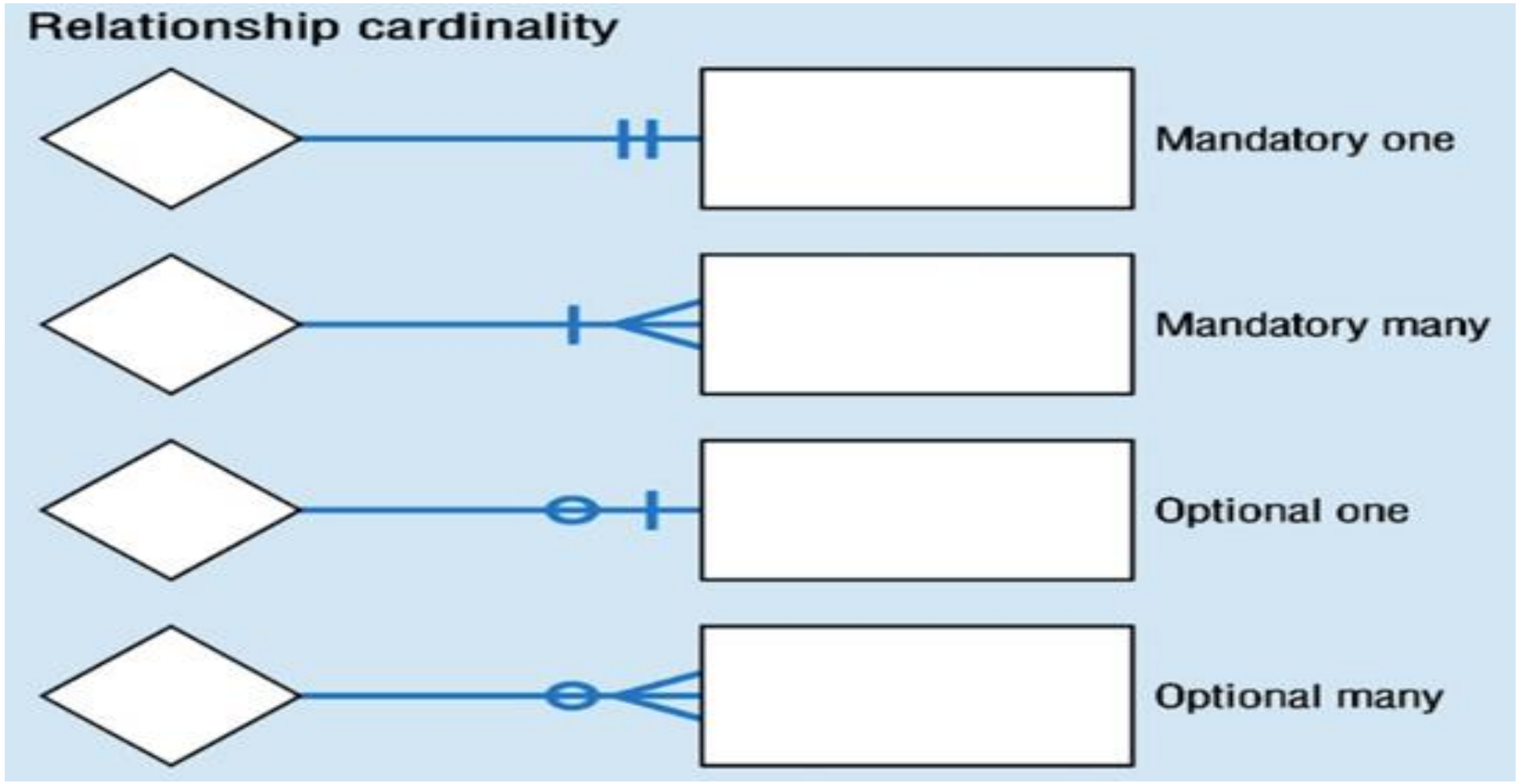


Entity-relationship model : Relationship Types

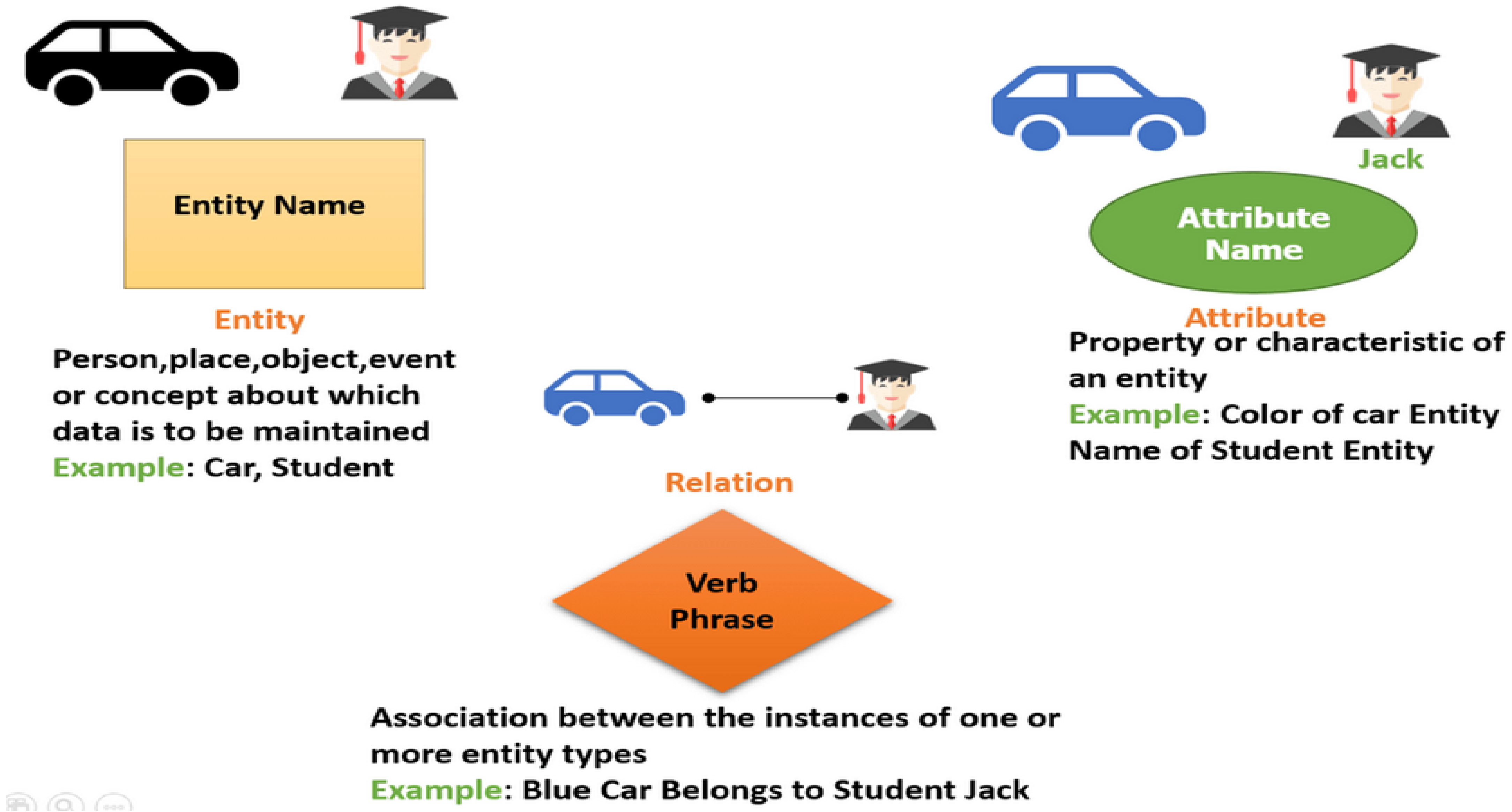
Relationship cardinality:

- Defines the numerical attributes of the relationship between two entities or entity sets.
- **One-to-One:** One entity from entity set X can be associated with at most one entity of entity set Y and vice versa.
- **One-to-Many:** One entity from entity set X can be associated with multiple entities of entity set Y, but an entity from entity set Y can be associated with at least one entity.
- **Many-to-One:** More than one entity from entity set X can be associated with at most one entity of entity set Y. However, an entity from entity set Y may be associated with more than one entity from entity set X.
- **Many-to-Many:** One entity from X can be associated with more than one entity from Y and vice versa.

Entity-relationship model : Relationship Types



Entity-relationship model : Example



Entity

Person,place,object,event or concept about which data is to be maintained

Example: Car, Student

Attribute

Property or characteristic of an entity

Example: Color of car Entity Name of Student Entity

Relation

Association between the instances of one or more entity types

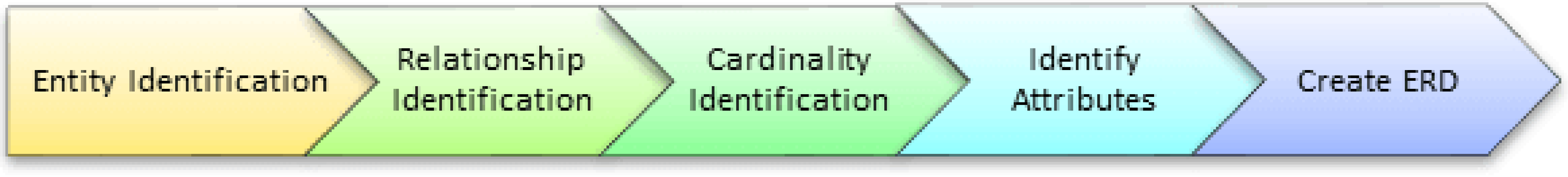
Example: Blue Car Belongs to Student Jack

Entity-relationship model : Creation of ERD

How to Create an Entity Relationship Diagram (ERD)?

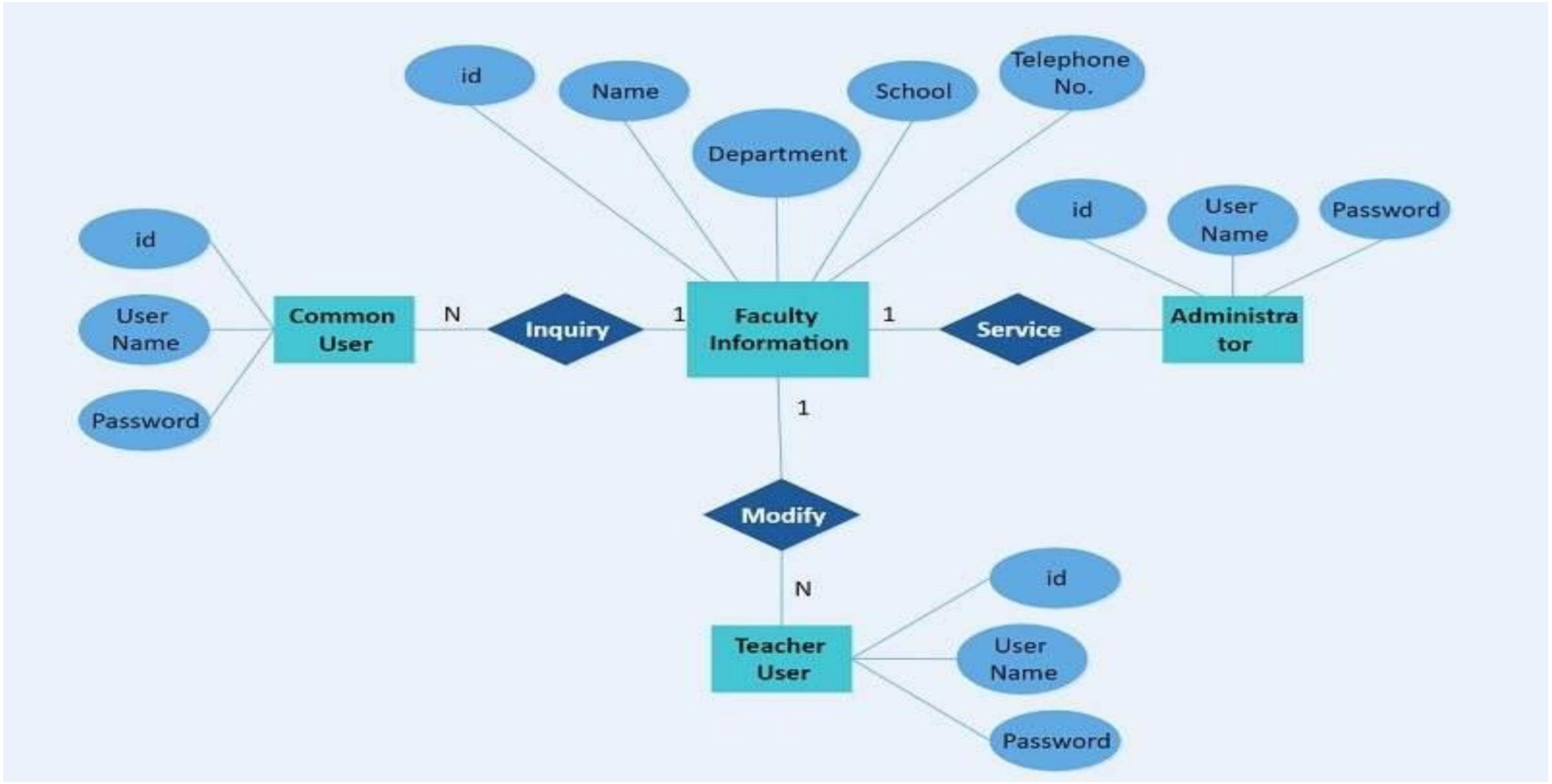
Following steps need to be followed:

1. Entity Identification
2. Relationship Identification
3. Cardinality Identification
4. Identify Attributes
5. Create ERD



Steps to Create an ER Diagram

Entity-relationship model : Example



SQL

Structured Query Language

SQL : Introduction

SQL Full Form:

- SQL stands for Structured Query language, pronounced as “S-Q-L” or sometimes as “See-Quel”.

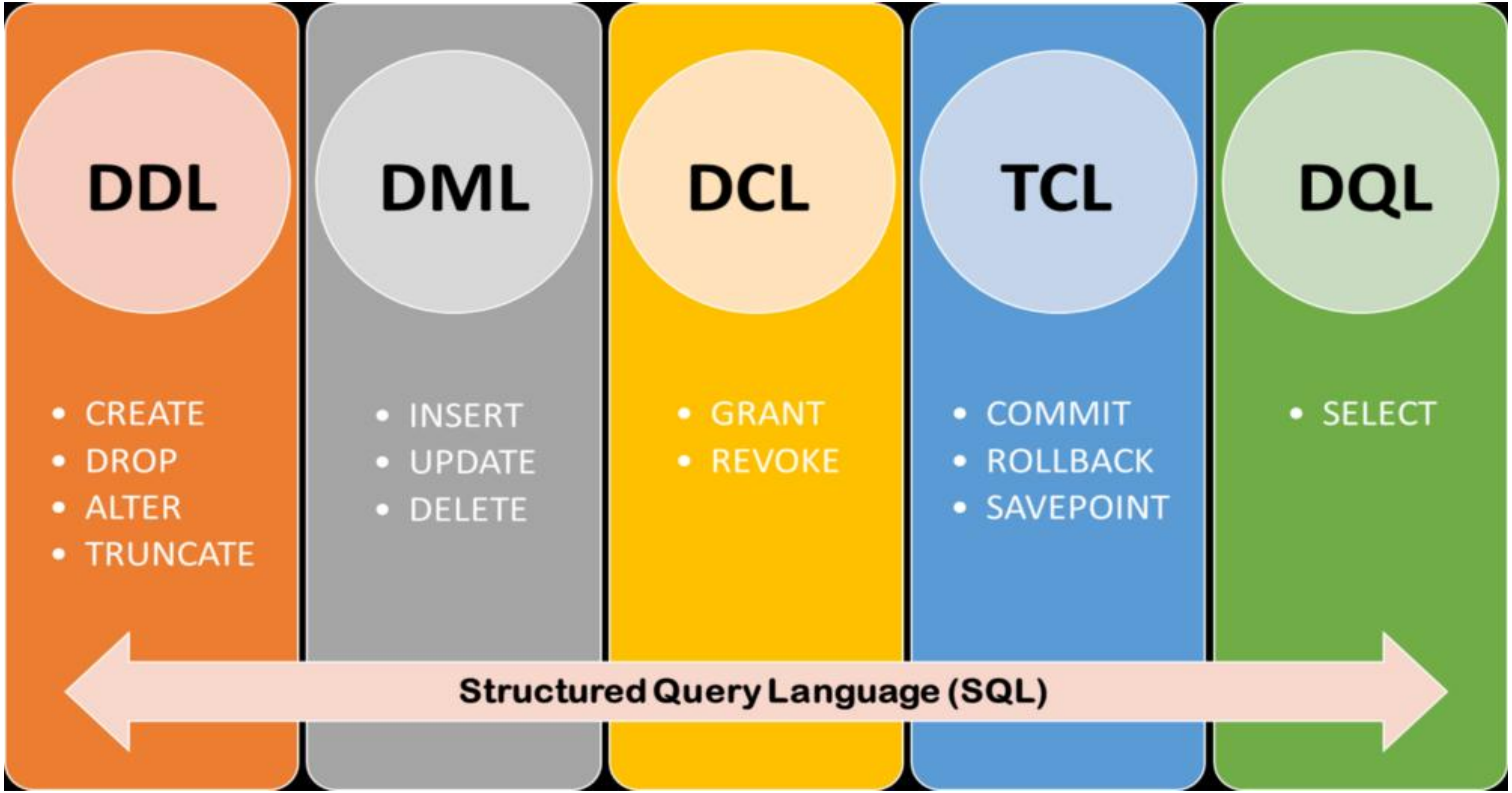
What is SQL used for?

- SQL is the standard language for dealing with Relational Databases.
- SQL can be used to perform CRUD operations on database records.

Types of SQL Statements: Here are five types of SQL queries.

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML)
3. Data Control Language (DCL)
4. Transaction Control Language (TCL)
5. Data Query Language (DQL)

Entity-relationship model : Introduction



SQL : Data Types

SQL : Data Types

Definition:

- Data types define the nature of the data that can be stored in a particular column of a table

Types:

- MySQL has 3 main categories of data types namely
  - **Numeric:** Numeric data types are used to store numeric values. It is very important to make sure range of your data is between lower and upper boundaries of numeric data types.
  - **Text:** Used to store text values. Always make sure you length of your textual data do not exceed maximum lengths.
  - **Date/time:** Used to store date and time information

SQL : Numeric Data Types

TINYINT( )	Signed: -128 to 127, Unsigned: 0 to 255
SMALLINT( )	Signed: -32768 to 32767, Unsigned: 0 to 65535
MEDIUMINT( )	Signed: -8388608 to 8388607, Unsigned: 0 to 16777215
INT( )	Signed: -2147483648 to 2147483647, Unsigned: 0 to 4294967295
BIGINT( )	Signed: -9223372036854775808 to 9223372036854775807 Unsigned: 0 to 18446744073709551615
FLOAT	A small approximate number with a floating decimal point.
DOUBLE(,)	A large number with a floating decimal point.
DECIMAL(,)	A DOUBLE stored as a string , allowing for a fixed decimal point. Choice for storing currency values.

SQL : Text Data Types

CHAR( )	A fixed section from 0 to 255 characters long.
VARCHAR( )	A variable section from 0 to 255 characters long.
TINYTEXT	A string with a maximum length of 255 characters.
TEXT	A string with a maximum length of 65535 characters.
BLOB	A string with a maximum length of 65535 characters.
MEDIUMTEXT	A string with a maximum length of 16777215 characters.
MEDIUMBLOB	A string with a maximum length of 16777215 characters.
LONGTEXT	A string with a maximum length of 4294967295 characters.
LONGBLOB	A string with a maximum length of 4294967295 characters.

SQL : Char Vs. Varchar

Type	**Char(n)**	**Varchar2(n)**
Useful for	Storing characters having pre-determined length	Storing characters whose length vary a lot
Storage size	Size for n characters	size for actual no. of characters + fixed size to store length
Storage	Trailing spaces are applied if data to be stored has smaller length than n.	Trailing spaces are not applied.
Max size	2000 Bytes	4000 Bytes
Alt Name	Character(n)	Character varying(n)
Example	A CHAR(10) field will store "Hello" as 10 bytes by appending 5 trailing spaces.	A VARCHAR2(10) field will store “Hello” as 7 bytes (assuming 2 bytes to store length).

SQL : Date / Time Data Type

DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD HH:MM:SS
TIMESTAMP	YYYY-MM-DD HH:MM:SS
TIME	HH:MM:SS
YEAR	YYYY or YY

SQL : DDL Commands

SQL : List of DDL Commands

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

- **DROP:** This command is used to delete objects from the database object.

- **ALTER:** This is used to alter the structure of the database object.

- **TRUNCATE:** This is used to remove all records from a table, including all spaces allocated for the records are removed but structure of table remains same.

- **RENAME:** This is used to rename an object existing in the database.

SQL : Creating Database

Creating a Database:

- To create a database in RDBMS, create command is used.
- **Syntax:**

  ```
  CREATE DATABASE <DB_NAME>;
  ```

- **Example:**

  ```
  CREATE DATABASE CCC;
  ```

SQL : Creating a Table

- The CREATE TABLE statement is used to create a new table in a database.

- **Syntax:**

```
CREATE TABLE [DB_Name].[tbl_Name]
(
        [Col_Name_1] [Datatype](Length) [Constraint_Name],
        [Col_Name_2] [Datatype](Length) [Constraint_Name] ,
        [Col_Name_3] [Datatype](Length),
        …. . . . .
);
```

- **Example:**

Create Table CCC.Student (studentId int, studentName varchar(30), contactNo int, emailId varchar(30), gender char(1));

SQL : Creation of table => Constraints

- SQL constraints are used to specify rules for the data in a table and are used to limit the type of data that can go into a table.

- If there is any violation between the constraint and the data action, the action is aborted.

- Constraints can be column level or table level.

- The following constraints are commonly used in SQL:
  - NOT NULL: Ensures that a column cannot have a NULL value
  - UNIQUE: Ensures that all values in a column are different
  - PRIMARY KEY: A combination of a NOT NULL and UNIQUE.
  - FOREIGN KEY: Prevents action that would destroy links on tables
  - CHECK: Ensures that the values in a column satisfies a specific condition
  - DEFAULT: Sets a default value for a column if 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 while performing insertion or update record in table.
- **Example:**

Create Table CCC.Student (studentId int NOT NULL, studentName varchar(30) NOT NULL, contactNo int, emailId varchar(30), gender char(1));

SQL : UNIQUE Constraint

- The UNIQUE constraint ensures that all values in a column are different.
- We can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.
- **Example:**

Create Table CCC.Student (studentId int NOT NULL UNIQUE, studentName varchar(30) NOT NULL, contactNo int, emailId varchar(30), gender char(1))

or

Create Table CCC.Student (studentId int NOT NULL, studentName varchar(30) NOT NULL, contactNo int, emailId varchar(30), gender char(1), unique(studentId));

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.
- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).
- **Example:**

Create Table CCC.Student (studentId int CONSTRAINT pk_Id PRIMARY KEY, studentName varchar(30), contactNo int, emailId varchar(30), gender char(1))

or

Create Table CCC.Student (studentId int PRIMARY KEY, studentName varchar(30) , contactNo int, emailId varchar(30), gender char(1));

SQL : Foreign Key Constraint

- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

- A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

- The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

- **Example:**

CREATE TABLE CCC.Marks (studentId int, courseId int, marks float, CGPA char(2), PRIMARY KEY (studentId, courseId), FOREIGN KEY (studentId) REFERENCES Student(studentId));

SQL : Check Constraint

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

- If you define a CHECK constraint on a column it
  - Will allow only certain values for this column.
  - Can limit the values in certain columns based on values in other columns in the row.

- **Example:**

Create Table CCC.Student (studentId int CONSTRAINT pk_Id PRIMARY KEY, studentName varchar(30), contactNo int, emailId varchar(30), gender char(1), CHECK(gender in (‘M’,’F’)));

SQL : Default Constraint

- The DEFAULT constraint is used to set a default value for a column.
- The default value will be added to all new records, if no other value is specified.
- **Example Query:** *Create Table CCC.Student (studentId int PRIMARY KEY, studentName varchar(30), contactNo int default 9999999999, emailId varchar(30), gender char(1));*
- The DEFAULT constraint can also be used to insert system values, by using functions like GETDATE()
- **Example Query:** *Create table CCC.EMP (empId int, empName varchar(30), deptNo int, dateOfJoin date default sysdate);*

SQL : Create table if not exists

- While creating a table that already exists, throws an error. To fix this issue, we can add the optional IF NOT EXISTS command while creating a table.

- **Example:**
  Create Table CCC.Student (studentId int, studentName varchar(30), contactNo int, emailId varchar(30), gender char(1));

SQL : Create table as

- We can also create a table using records from any other existing table using the CREATE TABLE AS command.

- Example:

  *Create Table CCC.StudentDuplicate as Select * from CCC.Student;*

- Above SQL command
  - Creates a table named *CCC.StudentDuplicate*
  - Don't copy the constraints into new duplicate table
  - Copies the records of the nested query into the new table.

- In case you want get only structure of the table but not data of the table then you can use false condition in query.

SQL : Deletion of DB object

- Removes one or more table definitions and all data, indexes, triggers, constraints, and permission specifications for those tables by using DROP command.

- **Syntax:**

  Drop table if exists DBname.TableName;

  Or

  Drop table tableName

- Get it back from recycle bin.
  - Syntax: *flashback table tableName to before drop;*

- To drop permanently,
  - Syntax: *Drop table tableName purge;*

SQL : Deleting data from DB object

- Removes all rows from a table or specified partitions of a table, without logging the individual row deletions.

- TRUNCATE TABLE is similar to the DELETE statement with no WHERE clause

- TRUNCATE TABLE is faster and uses fewer system and transaction log resources.

- Syntax:

  Truncate table DBName.tableName;

SQL : DROP Vs. TRUNCATE

DROP	TRUNCATE
• It is used to eliminate the whole database from the table.	• It is used to eliminate the tuples from the table.
• Integrity constraints get removed in the DROP command.	• Integrity constraint doesn’t get removed in the Truncate command.
• The structure of the table does not exist.	• The structure of the table exists.
• Here the table is free from memory.	• Here, the table is not free from memory.
• It is slow as compared to the TRUNCATE command.	• It is fast as compared to the DROP command.

SQL : Modifying of existing table structure

- The ALTER TABLE statement in SQL allows you to
  - Add new column(s) (ADD)
  - Modify the existing column structure (MODIFY)
  - Delete columns of an existing table (DROP)
  - Add / Delete constraints of column(s)
  - Rename the columns / database object

SQL : Modifying of existing table structure

Add new column(s) into an existing table:
Adding one column:

- Syntax: *alter table tableName ADD columnName dataType;*
- Example: *alter table CCC.Student ADD address varchar2(30);*

Adding multiple columns:

- Syntax: *alter table tableName ADD (colName1 dataType, colName2 dataType, …… , colNameN dataType);*
- Example: *alter table CCC.Student ADD (CGPA float, rank int);*

Modifying existing column structure:

- Syntax: *alter table tableName MODIFY columnName dataType;*
- Example: *alter table CCC.Student MODIFY address varchar(50);*

SQL : Modifying of existing table structure

Deleting column(s) from an existing table:
Deleting single column:

- Syntax: *alter table drop (columnName);*
- Example: *alter table drop (CGPA);*

Or

- Syntax: *alter table drop column columnName;*
- Example: *alter table drop column CGPA*

Delete Multiple columns:

- Syntax: a*lter table drop (columnName1, columnName2);*
- Example: *alter table drop (CGPA, rank)*

SQL : Modifying of existing table structure

Add constraint to an existing table:

- Syntax: *alter table tableName ADD constraint nameOfConstraint constraintType (columnName(s));*

- Example: *alter table CCC.Student ADD CONSTRAINT pk_SID PRIMARY KEY (studentId)*

Drop constraint from an existing table:

- Syntax: *alter table tableName DROP constraint nameOfConstraint*

- Example: *alter table CCC.Student DROP constraint pk_SID;*

Note: We can’t alter constraints ever but can drop them and then recreate.

SQL : Modifying of existing table structure

Rename existing table:

- Syntax: *Rename table oldTableName TO newTableName;*
- Example: *Rename table CCC.Marks TO CCC.studentMarks;*

- Syntax: *ALTER TABLE old_table_name RENAME new_table_name;*
- Example: *ALTER TABLE CCC.Marks RENAME CCC.studentMarks;*

Rename column(s) names of an existing table:

- Syntax: *ALTER TABLE tableName RENAME COLUMN OldColName TO NewColName;*
- Example: *ALTER TABLE CCC.Marks RENAME COLUMN sId TO stId;*

- Syntax: *ALTER TABLE TableName CHANGE COLUMN OldColName NewColName Data Type;*
- Example: *ALTER TABLE CCC.Marks CHANGE COLUMN sId StutId int;*

SQL : DML Commands

SQL : DML operations – Insert

- Once a table is created the most natural thing to do is load this table with data to be manipulated later.

- **Syntax:**

  Insert into tableName (col1, col2, … , colN) values (v1, v2, … , vN);

  Or

  Insert into tableName values (v1, v2, v3, …. , vN);

- **Examples:**

  Insert into CCC.Student (studentId, studentName, contactNo, gender) values (1227, ‘Ravi Kumar’, 9999900000, ‘M’);

  Insert into CCC.Student (1227, ‘Ravi Kumar’, 9999900000, ‘Ravi@gmail.com’, ‘M’);

SQL : DML Operations – Delete

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

- **Deleting single / group of record(s):**
  - Syntax: d*elete from tableName where condition;*

  - Examples:

    delete from CCC.Student where studentId = 1227;
    delete from CCC.Marks where marks = 90;

- **Deleting all of the records from table:**
  - Syntax: *delete from tableName;*

  - Example: *delete from CCC.Marks;*

SQL : DML Operations – Update

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

- **Updating single or multiple column(s) in existing table:**
  - Syntax: *UPDATE tableName SET col1 = value1, col2 = value2, … WHERE condition;*
  - Examples:

    Update CCC.Student set studentName = 'Ravi Kumar P' where studentId = 1227;

    Update CCC.Marks set marks = 90 where marks <= 75;

- **Update all of the records of the table:**
  - Syntax: *Update tableName set col1 = v1, col2 = v2, … , cN = vN;*
  - Example: *Update CCC.Marks set marks = 95;*

SQL : DCL Commands

SQL : DCL Commands

- The DCL commands are normally available to database administrator that allows system and data privileges to be passed to various users.

- **Creating a new user:**
  - Syntax: *Create user username identified by password;*
  - Example: *Create user CCC101 identified by CCC123;*

- **GRANT Command:**
  - GRANT is a very powerful statement used to manage the privileges of both users and roles throughout the data base.
  - It is of two types
    - System level permission
    - Object level permission.

SQL : DCL Commands – System Level Permission

- **Providing Roles:**
  - Syntax:

    GRANT CONNECT TO userName;
    GRANT CONNECT, RESOURCE, DBA TO userName;
  - Examples:

    Grant Connect To CCC101;
    Grant Connect, RESOURCE, DBA to CCC101;

- **Assigning privileges:**
  - Syntax:
    - *Grant Create Session To Username;*
    - *Grant Create Session Grant Any Privilege To Username;*
    - *Grant Unlimited Tablespace To Username;*
  - Example:
    - *Grant Create Session Grant Any Privilege To CCC101;*

SQL : DCL Commands – Object Level Permission

- If we want our user to have the ability to perform SELECT, UPDATE, INSERT, and DELETE operations on the table, we might execute the following GRANT statement.

- Grant command will be executed by Data Base Administrator.

- **Syntax:** *Grant TypeOfOperation on tableName to username;*

- **Examples:**
    - *GRANT SELECT ON CCC.Student TO CCC101;*
    - *GRANT SELECT, INSERT, UPDATE ON CCC.Marks TO CCC101;*
    - *GRANT SELECT, UPDATE ON CCC.Student TO CCC101, CCC102;*
    - *GRANT ALL ON CCC.Student TO PUBLIC;*
    - Note: All permission to all users, do not use it. It is very dangerous for database

SQL : DCL Commands – Removing Permission

- **REVOKE:** It is used to cancel the permission granted.

- **Syntax:**

  REVOKE typeOfOperation on tableName from username;

- **Examples:**

  REVOKE UPDATE ON CCC.Student FROM CCC101;
  REVOKE ALL ON CCC.Student FROM CCC101;

SQL : TCL Commands

SQL : TCL Commands

- Transaction Control Language specifies commands for beginning and ending a transaction.

- A transaction consists of a sequence of SQL statements that are applied in an atomic (all or none) manner.

- **The Commands In TCL:**
    - **Commit:** Used to save database changes permanently
    - **Savepoint:** Used to save different parts of the same transaction using different names.
    - **Rollback:** Used to undo the changes that are not committed

SQL : TCL – Commit

- COMMIT command in SQL is used to save all the transaction-related changes permanently to the disk.

- Whenever DML commands such as INSERT, UPDATE and DELETE are used, the changes made by these commands are permanent only after closing the current session.

- If we want the changes to be saved permanently to the disk without closing the session, we will use the commit command.

- Syntax:
  - *Commit;*

SQL : TCL – Commit Example

Create table emp (first_name char(20) not null, last_name char(20), age int, sex char(1), income float);

INSERT INTO EMP VALUES ('Krishna', 'Sharma', 19, 'M', 2000), ('Raj', 'Kandukuri', 20, 'M', 7000), ('Ramya', 'Ramapriya', 25, 'F', 5000);

START TRANSACTION;

UPDATE EMP SET AGE = AGE + 1;

SQL : TCL – Savepoint

- We can divide the database operations into parts.
- For example:
    - We can consider all the insert related queries that we will execute consecutively as one part of the transaction and the delete command as the other part of the transaction.

- Using the SAVEPOINT command in SQL, we can save these different parts of the same transaction using different names.
- For example:
    - We can save all the insert related queries with the savepoint named INS, insert related queries with the savepoint named DEL.

- **Syntax:** *SAVEPOINT savePointName;*
- **Example:** *SAVEPOINT samplesavepoint;*

SQL : TCL – Rollback

- According to the user's changing requirements, he/she can roll back the transaction to different savepoints.

- Consider a scenario:
  - *We have initiated a transaction followed by the table creation and record insertion into the table. After inserting records, we have created a savepoint INS. Then we executed a delete query, but later we thought that mistakenly we had removed the useful record.*

- To roll back our transaction we can use the ROLLBACK command to the savepoint.

- **Syntax:** *ROLLBACK TO SAVEPOINT NameOfSavePoint;*
- **Example:** *ROLLBACK TO SAVEPOINT samplesavepoint;*

SQL : DQL Command

SQL : DQL – Select

- **Query:** It is an operation that retrieves data from one or more tables or views

- Select statement is used for retrieving information from the databases.

- **Operators Used in "Select" statement:**
  1. Arithmetic operator(+,-,*,/)
  2. Relational operator(=,<,<=,>,>=,[<> or!=]not equal).
  3. Logical operator (AND,OR,NOT)

- **Operator Precedence:**
  - The basic operators used in SQL are * / + -
  - Operators of the same priority are evaluated From Left to right
  - Parentheses are used to force prioritized evaluation.

SQL : DQL – Select

- **Syntax:**

  Select col1, col2, … , colN from tableName; --To get only few cols

  *Select * from tableName; --To get all cols of record*

- **Examples:**
  - *SELECT ENAME, SAL, SAL+300 FROM EMP;*
  - *SELECT ENAME, SAL, 12*SAL+100 FROM EMP;*
  - *SELECT ENAME, SAL, 12*(SAL+100) FROM EMP;*
  - *SELECT * FROM EMP;*

SQL : DQL – Select

- **Column alias:**
  - **Question:** Print column names as NAME and SAL*12 as ANNUAL SALARY
  - **Query:** *Select ename “NAME”, sal*12 as ANNUALSALARY from emp;*
- **Concatenation operator:**
  - **Question:** Print name and job as one string column named as employees:
  - **Query:** *SELECT ENAME | | JOB “EMPLOYEES” FROM EMP;*

SQL : Special Operators

SQL : Special Operators

- SQL supports various special operators. Few of them are
  - IN opposite is NOT IN
  - BETWEEN opposite is NOT BETWEEN
  - IS NULL opposite is IS NOT NULL
  - LIKE opposite is NOT LIKE

SQL : Special Operators | IN

- The IN operator is a shorthand for multiple OR conditions allows you to specify multiple values in a WHERE clause.

- Syntax: *SELECT colname(s) FROM tableName WHERE colname IN (value1, value2, ...);*

- Question: Selects all customers that are located in "Germany", "France" or "UK"
- Query: *SELECT * FROM Customers WHERE Country IN ('Germany', 'France', 'UK');*

- Question: Select list of employees whose managerId value is 7902, 7566 and 7788
- Query: *Select empno, ename, sal, mgr from emp where mgr in (7902, 7566, 7788);*

SQL : Special Operators | NOT IN

- The NOT IN operator is a shorthand for multiple OR conditions not allows you to specify multiple values in a WHERE clause.

- Syntax: *SELECT colname(s) FROM tableName WHERE colname NOT IN (value1, value2, ...);*

- Question: Selects all customers that are not located in "Germany", "France" or "UK":
- Query: *SELECT * FROM Customers WHERE Country NOT IN ('Germany', 'France', 'UK');*

- Question: Select list of employees whose managerId value is not 7902, 7566 and 7788
- Query: *Select empno, ename, sal, mgr from emp where mgr not in (7902, 7566, 7788);*

SQL : Special Operators | BETWEEN

- 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.

- Syntax: *SELECT columname(s) FROM tableName WHERE columnName BETWEEN value1 AND value2;*

- Question: Selects all products with a price between 10 and 20:
- Query: *SELECT * FROM Products WHERE Price BETWEEN 10 AND 20;*

- Question: Selects all employees with a cityNames between Chennai and Hyderabad
- Query: *SELECT * FROM EMP WHERE cityName BETWEEN 'Chennai' AND 'Hyderabad'*

SQL : Special Operators | NOT BETWEEN

- To display the products outside the range we can use NOT BETWEEN:

- Syntax: *SELECT columname(s) FROM tableName WHERE columnName NOT BETWEEN value1 AND value2;*

- Question: Selects all products with a price not between 10 and 20. In addition; do not show products with a CategoryID of 1,2, or 3
- Query: *SELECT * FROM Products WHERE Price NOT BETWEEN 10 AND 20 AND CategoryID NOT IN (1,2,3);*

- Question: Selects all employees whose joining date is not between '01-July-1994' and '31-July-1996'
- Query: *SELECT * FROM EMP WHERE DOJ NOT BETWEEN '1994-07-01' AND '1996-07-31';*

SQL : Special Operators | NULL

- A field with a NULL value is a field with no value and not possible to test for NULL values with comparison operators, such as =, <, or <>.

- Syntax: *SELECT columnnames FROM tableName WHERE columnname IS NULL;*

- Question: Lists all employees with a NULL value in the "Address" field
- Query: *Select * from EMP where address IS NULL;*

- Question: Write a query to display the employees who are not getting commission from emp table.
- Query: *Select * from EMP WHERE comm IS NULL;*

- Question: Find emp whose manager-id is not specified from emp table.
- Query: *SELECT ENAME, MGR FROM EMP WHERE MGR IS NULL;*

SQL : Special Operators | NOT NULL

- The IS NOT NULL operator is used to test for non-empty values.

- Syntax: *SELECT columnnames FROM tablename WHERE columnname IS NOT NULL;*

- Question: Display all the employees who are getting commission from emp table
- Query: *Select * from emp where comm IS NOT NULL;*

- Question: Lists all employees with a value in the "Address" field
- Query: *Select * from emp where address IS NOT NULL;*

SQL : NVL() => Null Value Logic

- Question: Write a query to display ename, sal, comm, sal+comm of the employee, Whose empId is 1201 from emp table.
- Query: *Select ename, sal, comm, sal+comm from emp where empid = 1201;*

- Output:

```
ename             sal        comm            sal+comm
--------------------------------------------------------
Ravi Kumar        25550      NULL            NULL
Jeevan            38500      1500            40000
```

- To overcome this problem oracle provided "NVL()" function.

- NVL(): A predefined function which is used to replace or substitute user defined value in place of "null".

SQL : NVL() => Null Value Logic

- **Syntax:**

  NVL(exp1,exp2);

- Here exp1,exp2 must belongs to same data type. If exp1 is null then it returns exp2. Otherwise it returns exp1.

- **Examples:**

  NVL(null,30) => Result is 30.
  NVL(10,20) => Result is 10.

- **Solution Query:** *Select ename, sal, comm, sal+NVL(comm,0) from emp where empid = 1201;*

SQL : Special Operators | LIKE

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

- Like operator is used only with char and Varchar2 to match a pattern

- There are two wildcards often used in conjunction with the LIKE operator:
  - The percent sign (%) represents zero, one, or multiple characters
  - The underscore sign (_) represents one, single character

- Syntax: *SELECT col1, col2, ... FROM tableName WHERE column LIKE pattern;*

SQL : Special Operators | LIKE Examples

- Question: Selects all employees whose name starts with "a“
- Query: *Select * from EMP where ename LIKE ‘a%’*

- Question: Selects all employees whose name ends with "a“
- Query: *Select * from EMP where ename LIKE ‘%a’*

- Question: Selects all employees with a empName that have "or" in any position:
- Query: *Select * from EMP where ename LIKE '%or%';*

- Question: Selects all employees with a empName that have "r" in the second position
- Query: *Select * from EMP where ename LIKE ‘_r%’;*

SQL : Special Operators | LIKE Examples

- Question: Selects all employees with a empName that starts with "a" and are at least 3 characters in length
- Query: Select * from EMP where ename LIKE ‘a__%’;

- Question: Selects all employees with a empName that starts with "a" and ends with “r"
- Query: Select * from EMP where ename LIKE ‘a%r’;

- Question: Selects the employees who are joining in the month December
- Query: Select * from EMP where hiredate LIKE ‘%D%’;

- Question: Selects the employees who are joining in the year “81”
- Query: Select * from EMP where hiredate LIKE ‘%81’;

SQL : Special Operators | NOT LIKE

- The NOT LIKE operator is used in a WHERE clause to search for a specified pattern not in a column.

- Syntax: *SELECT col1, col2, … FROM tableName WHERE column NOT LIKE pattern;*

Examples:

- Question: Selects all employees whose name not ends with "a“
- Query: *Select * from EMP where ename NOT LIKE ‘%a’*

- Question: Write a sql query with the conditions as 'NOT LIKE IN‘
- **Note:** You cannot combine LIKE and IN.
- Query: *Select * from Table1 where EmpPU NOT Like '%CSE%' AND EmpPU NOT Like '%ECE%' AND EmpPU NOT Like '%EEE%'*

SQL : Functions

SQL : Functions

- Functions: Used to solve particular task and also must return a value.

- SQL have two types of functions.
  1. Predefined functions
     1. Number function
     2. Character function
     3. Date function
     4. Group function(or) Aggregate function

  2. User defined functions

SQL : Number Functions

Abs (): It is used to convert negative values into positive values.

- Query: *Select abs (-50) from dual;*

Mod(m,n): It will gives remainder after „m" divided by „n".

- Query: *Select mod(10,5) from dual;*

Greatest(ex1,ex2,.., exn): Returns max value among given expressions.

- Query: *Select greatest(3,5,8,9) from dual;*

Least(ex1,ex2,...., exn): Returns min value among given expressions.

- Query: *Select least(3,5,8,9) from dual;*

- Query: *Select ename, sal, comm, greatest(sal,comm) from emp where comm is not null;*

SQL : Character Functions

Upper(): It is used to convert a string or column values into upper case.

- Query: *select upper ('abc') from dual;*

Lower(): It is used to convert a string or column values into lower case.

- Query: *select lower(ename) from emp;*

- **Question:** Updating or modifying data within table:
- Query: *update emp set ename=lower(ename);*

Initcap(): Returns first letter is capital and remaining letters are small.

- Query: *select initcap(ename) from emp;*
- *Eg: SQL> select initcap('ab cd ef') from dual;*

Length(): Returns total length of the string including spaces.

- Query: *select length('AB_CD') from dual;*

SQL : Date Functions

Sysdate: It returns current date of the system in the oracle date format

- Query: *Select sysdate from dual;*

Add_months(): Used to add or subtract number of months to the specified date based on second parameter.

- Syntax: *add_months(date, number);* Number-> can be +ve / -ve
  - Query1: *Select add_months(sysdate,1)from dual;*
  - Query2: *Select add_months(sysdate,-1) from dual;*

Last_day(): Returns last date of the given months based on the date.

- Syntax: *last_day(date);*
- Query: *Select next_day(sysdate,"monday") from dual;*

Months_between(): Returns no. of months between two specified dates.

- Syntax: months_between(date1,date2)
- Query: Select ename, round(months_between (sysdate, doj))from emp;

SQL : Date Functions

DATE CONVERSION FUNCTIONS: Oracle provided two date conversion function. They are:

To_char(): It is used to convert date time into character type i.e., it converts date type into date string.

- Example Query: *Select to_char(sysdate,'dd/mm/yy')from dual;*

- **Note:** Basically "to_char" character formats are case sensitive formats.
- Example Query: *Select to_char(sysdate,'DAY')from dual;*

To_date(): It is used to convert string dates into date type.

- Example Query: *Select to_date('24/01/yy','dd/mm/yy') from dual;*

SQL : Group / Aggregate Functions

- Oracle having following group function
  1. max()
  2. min()
  3. avg()
  4. sum()
  5. count(*)
  6. count(column name)

- Question: Find the total number of employees.
- Query: *SELECT COUNT(*) FROM EMP;*

- Question: Find the min, max and average salaries of employees of department 20.
- Query: *SELECT MIN(SAL), MAX(SAL), AVG(SAL) FROM EMP WHERE DEPTNO = 20;*

SQL : Clauses in SQL

SQL : DQL | Where - Clauses in Select

WHERE clause:

- A WHERE clause is an optional part of a Select Expression, DELETE statement, or UPDATE statement.

- The WHERE clause lets you select rows based on a Boolean expression.

- Only rows for which the expression evaluates to TRUE are returned in the result, or, in the case of a DELETE statement, deleted, or, in the case of an UPDATE statement, updated.

- Syntax: *WHERE Boolean expression*

SQL : DQL | Where - Clauses in Select

Examples with WHERE clause:

- **Question:** Print an information about the student whose student id value is equal to 1227
- **Query:** *select * from CCC.Student where studentId = 1227;*

- **Question:** Print an information about the employee whose manager id is 7845 and department number is 5
- **Query:** *select * from CCC.EMP where mgrId = 7845 and deptNo = 5;*

- **Question:** Print an information about the employee who is taking salary less than 15000 and greater than 50000;
- **Query:** *select * from CCC.Emp where salary < 15000 and salary > 50000;*

SQL : DQL | Group By - Clauses in Select

- It is used to group database tuples on the basis of certain common attribute value.
  - Example: Employees of a department.

- Question: Find department number and Number of Employees working in that department.
- Query: *Select deptno, count(empno), from emp, group by deptno;*

- Note that while using group by and aggregate functions the only attributes that can be put in select clause are the aggregated functions and the attributes that have been used for grouping the information.

SQL : DQL | Group By - Clauses in Select

- Question: Write a query to display number of employees in each job from emp table using group by?
- Query: *Select job, count(*) from emp group by job;*

- Question: Write a query to display deptno, minimum and maximum salary from emp using group by?
- Query: *Select deptno, min(sal), max(sal) from emp group by deptno;*

RULE:

- Other than group function columns specified after select those all columns must be used in after "group by". Otherwise oracle server returns an error "not a GROUP BY expression".
- Example: *Select deptno, max(sal), ename from emp group by deptno, ename;*

```
ERROR at line 1:
ORA-00979: not a GROUP BY expression
```

SQL : DQL | Having - Clauses in Select

- This clause is used for creating conditions on grouped information.
- Question: Find department number and maximum salary of those departments where maximum salary is more than Rs 20000/-.
- Query: *SELECT DEPTNO, MAX(SAL) FROM EMP GROUP BY DEPTNO HAVING MAX(SAL) > 20000;*

SQL : DQL | Order By – Clauses in Select

- **Order By clause:**
  - It is used in the last portion of select statement
  - By using this rows can be sorted, default it takes ascending order
  - DESC: is used for sorting in descending order
  - Sorting by column which is not in select list is possible
  - Sorting can be done by column Alias

- Question: Print the employee information based on annual salary in ascending order
- Query: SELECT EMPNO, ENAME, SAL*12 “ANNUAL” FROM EMP ORDER BY ANNUAL;

- Question: Print the employees information in ascending order on department number and descending order of salary in each department.
- Query: SELECT ENAME, DEPTNO, SAL FROM EMP ORDER BY DEPTNO, SAL DESC;

Thank You!