Structured Query Language (SQL).

Chapter – 1
Introduction to Structured Query Language (SQL)

Introduction of SQL:

SQL stands for „Structured Query Language“ and can be pronounced as „SQL“ or „SEQUEL“ (Structured English Query Language). It is a special purpose language used to define access and manipulate data in RDBMS. According to ANSI (American National Standards Institute), it is the standard language for relational database.

- SQL provides a set of statements for storing and retrieving data from the relational database.
- The original version called SEQUEL was designed by IBM in mid-1970. SQL was first introduced as a commercial database system in 1979 by Oracle Corporation.
- Now-a-days, almost all MRDBMS (Modern Relational Database Management Systems) like Microsoft SQL-Server, Microsoft Access, Oracle, BD2, Sybase, MySQL and Informix use SQL as standard database language.
- For example Microsoft SQL-Server specific version of the SQL is called T-SQL, Oracle version of SQL is called PL/SQL, and Microsoft Access version is SQL is called JET-SQL etc.
- SQL is different from other programming languages like C++, Visual-Basic etc.
- It is a Non-Procedural, English-like language that processes data in groups of records rather than one record at a time.
- This Non-Procedural nature of SQL makes it easier to access data in application programs.

Types of SQL: SQL is broadly classified into two types.

1. Interactive SQL.
2. Embedded SQL.

1. **Interactive SQL:** "Interactive SQL is used for directly access the data from the database". Whereas the output of operation is used for human consumptions. Once a command is specified is "executed and output is immediately view by the users".

2. **Embedded SQL:** "In the case of Embedded SQL commands are SQL Commands that are written in some another languages". Such as COBAL, ALP (Assembly Level Programming Language). This makes the programmer very fast and powerful manner.

Advantages of SQL:

- SQL is a high level language that provides a greater degree of „abstraction“.
- SQL deals with number of database management system at a time.
- SQL is an English like computer language which makes interaction between user and database very simply.
- SQL language which being simple and easy to learn can handle complex situation.
- SQL operations are performed at a set level. One select statement can retrieve multiple rows.

What SQL can Do? :

- SQL can retrieve data from a database.
- SQL can execute queries against a database.
- SQL can insert, update and delete records in a database.
- SQL can create new tables in a database.
- SQL can create views in a database.
- SQL cam set permissions on tables, procedures and views.

***** (Q) Define SQL*PLUS? And illustrate differences with SQL?**

SQL*PLUS is a software product from Oracle Corporation that allows users to interactively use the SQL commands, produce formatted reports and support written command-procedures to access Oracle Database. Though SQL*PLUS, a user can Do:

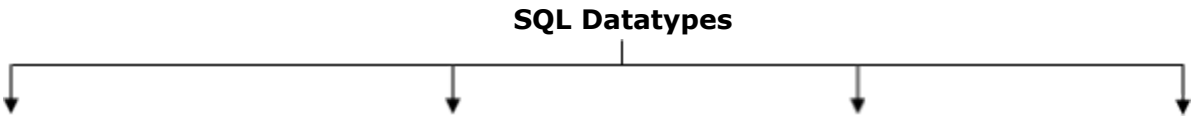
- Enter, Edit, Store, Retrieve and Run SQL Commands.
- Format, Perform calculations, Print Query results in the form of Reports.
- Access and Copy data between SQL Database.
- Send messages to and accepts responses from an End-Users.

Difference between SQL and SQL*Plus:

SQL	SQL*PLUS
SQL is a special-purpose language used to define, access and manipulate data in RDBMS like Oracle	SQL*PLUS is a command line tool that allows user to type SQL commands to be executed directly against an Oracle Database.
SQL provides set of commands used to communicate with the Database to perform specific tasks.	SQL*PLUS is the environment where we can actually type wer commands to perform specific tasks.
SQL commands can be used to create tables objects like Create, Alter, drop etc.	SQL*PLUS commands can be used to describe the structure of Database objects. For example: DESCRIBE/DESCRIPTION is DESC command.
SQL commands cannot be abbreviated.	In SQL*PLUS commands can be abbreviated. Like the commands EDIT can be ED.
Every SQL commands must be ended with semicolon (;)	In SQL*PLUS semicolon (;) is optional.

***** (Q) Define various datatypes in SQL?**

SQL Datatypes: Most of programming languages require programmed declaration by using datatypes. The most database system required the user to specific the type of each data fields. The available datatypes varying from one programming language to other languages. The SQL supports the following scalar data types.



Numeric Datatype	String/Character Datatype	Date & Time Datatype	LOB Datatype
Bit (size)	Char (n)	Date	BLOB
Tinyint (size)	Varchar2 (n)	Time	CLOB
Boolean	Text	Datetime	BFILE
Smallint (size)		Year	
Integer (size)		Timestamp	
Decimal (size, d)			
Float (d, p)			
Number (p, q)			
Long			
Raw (n)			

I. Numeric Datatype

1. **Bit (size) Datatype:** A bit-value type, the number of bits per value is specified in size. The size parameter can hold a value from 1 to 64. The default value for size is 1.

2. **Tinyint (size) Datatype:** A very small integer, signed range is from -128 to 127. Unsigned range is from 0 to 255. The size parameter specifies the maximum display width (which is 255).

3. **Boolean Datatype:** Zero is considered as false, nonzero values are considered as true.

4. **Smallint (size) Datatype:** A small integer, signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The size parameter specifies the maximum display width (which is 255).

5. **Integer (size) Datatype:** An integer, signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255).

6. **Decimal (size, d) Datatype:** An exact fixed-point number, the total number of digits is specified in size. The number of digits after the decimal point is specified in the „d“ parameter. The maximum number for size is 65. The maximum number for „d“ is 30. The default value for size is 10. The default value for „d“ is 0.

7. **Float (d, p) Datatype:** A floating point number, the total number of digits is specified in size. The number of digits after the decimal point is specified in the „d“ parameter. A floating point number. SQL uses the „p“ value to determine whether to use FLOAT or DOUBLE for the resulting data type. If „p“ is from 0 to 24, the data type becomes FLOAT(). If „p“ is from 25 to 53, the data type becomes DOUBLE().

8. **Number (p, q) Datatype:** Fixed precision and scale numbers. The „p“ parameter indicates the maximum total number of digits that can be stored „p“ must be a value from 1 to 38. Default is 18. The „q“ parameter indicates the maximum number of digits stored to the right of the decimal point. „q“ must

be a value from 0 to p. Default value is 0.

9. **Long Datatype:** Allows whole numbers between -2,147,483,648 and 2,147,483,647.

10. **Raw (n) Datatype:** Variable length raw binary data. Maximum of length „n“ is 255 bytes.

II. String/Character Datatype

1. **Char (n) Datatype:** A Fixed width character string (can contain letters and special characters). The size parameter specifies the column length in characters - can be from 0 to 255. Default is 1.

2. **Varchar2 (n) Datatype:** A varying length string (can contain letters and special characters). The size parameter specifies the maximum column length in characters - can be from 0 to 65535.

3. **Text Datatype:** Variable width character string. Holds a string with a maximum length of 65,535 bytes or 2GB of text data.

III. Date & Time Datatype

1. **Date Datatype:** A date, Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999- 12-31'. Example: 2020-11-12 or 2020-Nov-12.

2. **Time Datatype:** A time, Format: HH:MM:SS. The supported range is from '00:00:00' to '23:59:59'. Example: 22:45:50.

3. **Datetime Datatype:** A date and time combination. Format: YYYY-MM-DD HH:MM:SS. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time. Example: 2020-Nov-12 22:46:32.

4. **Year Datatype:** A year values allowed in four-digit format: 1000 to 9999. Example: 2020.

5. **Timestamp Datatype:** A timestamp values are stored as the number of seconds since the Universal Time Coordinate (UTC) Format: YYYY-MM-DD HH:MM:SS. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP.

IV. LOB Datatype (Large Object)

Multimedia data like sound, picture and video need more storage space. The LOB datatype such as BLOC, CLOB and BFILE allows us to store large block of data.

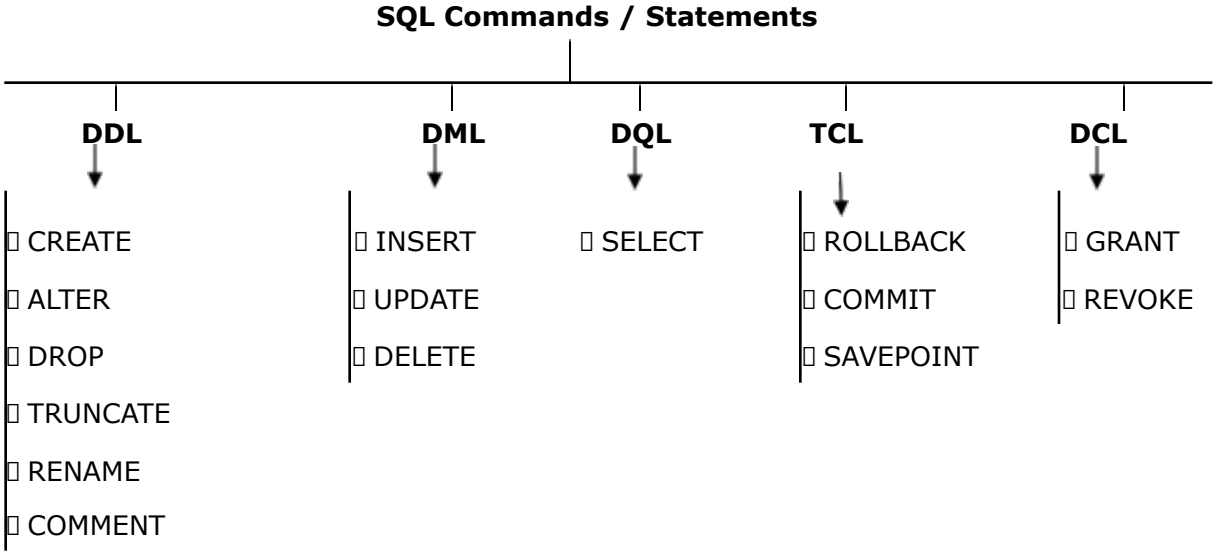
1. **BLOB (Binary Large Object) Datatype:** The BLOB datatype stores unstructured binary data in the database. BLOC can store up to 4GB of binary data.

2. **CLOB (Character Large Object) Datatype:** The CLOB datatype stores can store up to 4GB of character data in the database.

3. **BFILE (Binary File) Datatype:** The BFILE datatype stores unstructured binary data in operating system files outside the database. A BFILE can store up to 4GB of data.

******* (Q) Explain the various types of SQL Commands?**

Structure Query Language (SQL) is a database query language used for storing and managing data in Relational DBMS. SQL was the first commercial language introduced for Dr. E.F Codd's Relational model of database. Today almost all RDBMS (MySql, Oracle, Infomix and Sybase, MS-Access) use SQL as the standard database query language. SQL statements are divided into the following categories:



(I). **Data Definition Language (DDL):** DDL commands are used to "Define the Database Structure" by "Creating, Altering, Dropping, Truncating, Renaming and Commented Table". All DDL commands are auto-committed. That means it saves all the changes permanently in the database.

- **CREATE**: Create is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
- **ALTER**: Alter is used to altering the structure of the database.
- **DROP:** Drop is used to delete objects from the database.
- **TRUNCATE:** Truncate is used to remove all records from a table, including all spaces allocated for the records are removed.
- **RENAME**: Rename is used to rename a table existing in the database.
- **COMMENT:** Comment is used to add comments to the data dictionary.

(II). **Data Manipulation Language (DML):** DML commands allowed the user to "Manipulate data". Manipulate data refers to "Inserting, Updating, Deleting, Merging and calling of data". DML commands are not auto-committed. It means changes are not permanent to database, they can be rolled back.

- **INSERT**: Insert statement is used to insert new records into a table.
- **UPDATE**: Update is used to update existing data within a table.
- **DELETE**: Delete is used to delete records from a database table.

(III). **Data Query Language (DQL):** DQL commands is used to "Fetch or access data from tables based on conditions" that we can easily apply.

- **SELECT**: Select is used to retrieve records from one or more tables from the database.

(IV). **Transaction Control Language (TCL):** TCL commands are to keep a check on other commands and their affect on the database. These commands can annul changes made by other commands by

rolling the data back to its original state. It can also make any temporary change permanent.

- **ROLLBACK:** Rollback is used to a transaction in case of any error occurs.
- **COMMIT:** Commit is used to permanently save the transaction into a database.
- **SAVEPOINT:** Savepoint is used to temporally save the transaction into a database.

(V). **Data Control Language (DCL):** DCL is the commands to grant and take back authority from any database user. The Database Administrator (DBA) has the power to give and take the privileges to a specific user, thus giving or revoking access to the data. The task of the DCL is to prevent unauthorized access to data.

- **GRANT**: Grant is used to gives user"s access privileges to database.
- **REVOKE:** Revoke is used to withdraw user"s access privileges given by using the GRANT command.

(I) Data Definition Language Commands (DDL)

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

(a). **Create New table:** To create a new table into the database.

Syntax: SQL> create table <Table_Name>
(Column_Name1 datatype (size),
Column_Name2 datatype (size),
- - - - - - - - - - -
- - - - - - - - - - -
Column_Name n datatype (size));

SQL> create table STUDENT (
emp_Id number(3),
emp_Name varchar2(15),
experience number(2),
salary number(5));

Table created.

SQL> desc STUDENT;

Name	Null?	Type
EMP_ID		NUMBER(3)
EMP_NAME		VARCHAR2(15)
EXPERIENCE		NUMBER(2)
SALARY		NUMBER(5)

(b). **Create Table Using another Table:** A copy (duplicate) of an existing table can also be created using CREATE TABLE. The new table gets the same column definitions. All columns or specific columns can be selected. If we create a new table using an existing table, the new table will be filled with the existing values from the old table.

Syntax: SQL> create table <New Table_Name> AS
select (existing Column_Name1, existing Column_Name2 ,…… , existing Column_Name) n,
from <Existing Table_Name> where <conditions>;

SQL> create table STUDENT as select emp_id, emp_name, salary from STUDENT;

Table created.

SQL> desc STUDENT; <Existing Table>

Name	**Null?**	**Type**
EMP_ID		NUMBER(3)
EMP_NAME		VARCHAR2(15)
EXPERIENCE		NUMBER(2)
SALARY		NUMBER(5)

SQL> desc STUDENT; <New Table>

Name	**Null?**	**Type**
EMP_ID		NUMBER(3)
EMP_NAME		VARCHAR2(15)
SALARY		NUMBER(5)

2. **Alter command:** Alter is used to altering the structure of the database. Alter statements can support any adding new columns from existing tables or any of changes to the size or datatype of an existing column in database or dropped existing columns or rename from existing table to new table in the database.

(a). **Adding New Columns:** To add a New Columns from existing table.

Syntax: SQL> alter table <existing Table_Name> ADD
(new Column_Name1 datatype (size),
new Column_Name2 datatype (size),
- - - - - - - - - - -
- - - - - - - - - - -
new Column_Name n datatype (size));

SQL> alter table STUDENT add (Contact_Number number(12), Address varchar2(30));

Table altered.

SQL> desc STUDENT;

Name	Null?	Type
EMP_ID		NUMBER(3)
EMP_NAME		VARCHAR2(15)
EXPERIENCE		NUMBER(2)
SALARY		NUMBER(5)
CONTACT_NUMBER		NUMBER(10)
ADDRESS		VARCHAR2(15)

(b). **Modify the Existing Column values:** To modify the existing table values like datatype and size.

Syntax: SQL> alter table <Table_Name> MODIFY
(existing Column_Name1 datatype (size),
existing Column_Name2 datatype (size),
- - - - - - - - - - -
- - - - - - - - - - -
existing Column_Name n datatype (size));

SQL> alter table STUDENT modify (address char(20)); Table

altered.

SQL> desc STUDENT;

Name	Null?	Type
EMP_ID		NUMBER(3)
EMP_NAME		VARCHAR2(15)
EXPERIENCE		NUMBER(2)
SALARY		NUMBER(5)
CONTACT_NUMBER		NUMBER(10)
ADDRESS		CHAR(20)

(c). **Drop existing column:** To delete an existing column from existing table.

Syntax: SQL> alter table <Table_Name> DROP COLUMN <Existing Column_Name>;

SQL> alter table STUDENT drop column contact_number;
Table altered.
SQL> desc STUDENT;

Name	Null?	Type
EMP_ID		NUMBER(3)
EMP_NAME		VARCHAR2(15)
EXPERIENCE		NUMBER(2)
SALARY		NUMBER(5)
ADDRESS		CHAR(20)

(d). **Rename Table:** To rename the table from existing table to new table.

Syntax: SQL> alter table <existing Table_Name> RENAME TO < new Table_Name>;

SQL> alter table STUDENT rename to student123;

SQL> select * from STUDENT;

ERROR at line 1: ORA-00942: table or view does not exist

SQL> select * from student123;

EMP_ID	EMP_NAME	CONTACT_NUMBER
101	Mohammad student123	9491202987
102	Shaik Abu	9442011432

3. **Drop command:** This command is use "Drop the Structure / Table (or) Delete the Objects / Tables from the Database permanently". When using the drop table statement, we cannot „rollback".

Syntax: SQL> drop table <Table_Name>;

SQL> drop table STUDENT; Table
dropped.

4. **Truncate command:** Truncate is used to remove all records from a table, including all spaces allocated for the records are removed by the table, but table structure will be alive. When using the truncate table statement, we can"t „rollback".

Syntax: SQL> truncate table <Table_Name>;

SQL> truncate table STUDENT;

SQL> select * from STUDENT;

5. **Comments in SQL:** There are two ways in which we can comment in SQL, i.e. either the Single-Line Comments or the Multi-Line Comments.

(a). **Single-Line Comments:** The single line comment starts with two hyphens (--), till the end of a single line will be ignored by the compiler.

SQL> -- Display all the records from student123 table.

SQL> Select * from student123;

(b). **Multi-Line Comments:** The Multi-line comments start with **/*** and end with ***/**. So, any text mentioned between /* and */ will be ignored by the compiler.

SQL> /* Display all records which have
2. from the student123 table */
3. Select * from student123;

(II) Data Manipulation Language (DML)

1. **Insert command:** Insert statement is used to insert new records into a table. There are inserting three methods as following.

(a). Inserting dynamically.
(b). Inserting only with specify Column_Names.
(c). Inserting without any specify Column_Names.

(a). **Inserting dynamically:** In this method values can be inserting through reference address of the Column_Names, and in this method we can also inserted multiple rows without repeat syntax, Just use „/“ symbol.

Syntax: SQL> insert into <Table_Name> values
(„& Column_Name1“, „& Column_Name2“, „& Column_Name3“,
- - - - - - - - - - -
- - - - - - - - - - -
„& Column_Name n“);

SQL> insert into STUDENT values („&emp_id“, „&emp_name“, „&experience“, „&salary“, „&contact_number“, „&address“);

Enter value for emp_id: 101
Enter value for emp_name: Mohammad
Enter value for experience: 12
Enter value for salary: 36500
Enter value for contact_number: 9491202987
Enter value for address: Tirupati

1 row created.

SQL> /
Enter value for emp_id: 102
Enter value for emp_name:
student123 Enter value for
experience: 11 Enter value for
salary: 34650
Enter value for contact_number: 9442011432
Enter value for address: Hyderabad

1 row created.

SQL> select * from STUDENT;

EMP_ID	EMP_NAME	EXPERIENCE	SALARY	CONTACT_NUMBER	ADDRESS
101	Mohammad	12	36500	9491202987	Tirupati
102	student123	11	34650	9442011432	Hyderabad

2 rows selected.

(b). **Inserting only with specify Column_Names:** In this method values can be inserting only specify columns rather than all.

Syntax: SQL> insert into <Table_Name>
(Column_Name1, Column_Name3, Column_Name6, „& Column_Name n“)
values („value of Column_Name1“, „value of Column_Name3“, „value of Column_Name n“);

SQL> insert into STUDENT (emp_id, emp_name, experience, address)
values (103', 'Shaik', '08', 'Rompicherla');

1 row created.

SQL> insert into STUDENT (emp_id, emp_name, experience, address)
values ('104', 'Abu', '06', 'Tirupati');

1 row created.

SQL> select * from STUDENT;

EMP_ID	EMP_NAME	EXPERIENCE	SALARY	CONTACT_NUMBER	ADDRESS
101	Mohammad	12	36500	9491202987	Tirupati
102	student123	11	34650	9442011432	Hyderabad
103	Shaik	8			Rompicherla
104	Abu	6			Tirupati

4 rows selected.

(c). **Inserting without any specify Column_Names:** In this method values can be inserting without any specify Column_Names into the table.

Syntax: SQL> insert into <Table_Name>
values („value of Column_Name1“, „value of Column_Name2“, „value of Column_Name n“);

SQL> insert into STUDENT values ('105', 'Muneer', '5', '12000', '9491202169', 'Mumbai');

1 row created.

SQL> insert into STUDENT values ('106', 'Mujeeb', '2', '8500', '9441201342', 'Chennai');

1 row created.
SQL> select * from STUDENT;

EMP_ID	EMP_NAME	EXPERIENCE	SALARY	CONTACT_NUMBER	ADDRESS
101	Mohammad	12	36500	9491202987	Tirupati
102	student123	11	34650	9442011432	Hyderabad
103	Shaik	8			Rompicherla
104	Abu	6			Tirupati
105	Muneer	5	12000	9491202169	Mumbai
106	Mujeeb	2	8500	9441201342	Chennai

6 rows selected.

2. **Update command:** The update statement is used to modify the data that is already in the database. The condition in the WHERE clause decides that which row is to be updated. There are three situations to use updated command.

(a). Update single records. (Null values)
(b). Update multiple records. (Replace values)
(c). Update all records at a time. (If omitted where clause)

Syntax: update <Table_Name> set <Column_Name1> = <Value 1>,
<Column_Name2> = <Value 2>,
- - - - - - - - - - -
- - - - - - - - - - -
<Column_Name n> = <Value n>
[where <condition>];

(a). **Update Single record (Null values):** if we inserting new value at the place of missing field one by one.

Syntax: SQL> update <Table_Name> set <Column_Name> = <value> [where <condition>];

SQL> update STUDENT set salary = '26580' where emp_id = '103';

1 row updated.
SQL> select * from STUDENT;

EMP_ID	EMP_NAME	EXPERIENCE	SALARY	CONTACT_NUMBER	ADDRESS
101	Mohammad	12	36500	9491202987	Tirupati
102	student123	11	34650	9442011432	Hyderabad
103	Shaik	8	26580		Rompicherla
104	Abu	6			Tirupati
105	Muneer	5	12000	9491202169	Mumbai
106	Mujeeb	2	8500	9441201342	Chennai

6 rows selected.
Update multiple records (Replace): If we want to update multiple columns, we should separate each field assigned with a comma. And also replace the values from existing values with new values.

Syntax: update <Table_Name> set <Column_Name1> = <Value 1>,
<Column_Name2> = <Value 2>,
- - - - - - - - - - -
- - - - - - - - - - -
<Column_Name n> = <Value n>
[where <condition>];

SQL> update STUDENT set emp_name = 'Masthan', address = 'Piler' where emp_id = '103';

SQL> select * from STUDENT;

EMP_ID	EMP_NAME	EXPERIENCE	SALARY	CONTACT_NUMBER	ADDRESS
101	Mohammad	12	36500	9491202987	Tirupati
102	student123	11	34650	9442011432	Hyderabad

103	Masthan	8	26580		Piler
104	Abu	6			Tirupati
105	Muneer	5	12000	9491202169	Mumbai
106	Mujeeb	2	8500	9441201342	Chennai

6 rows selected.

(b). **Update all records at a time:** If we want to update all row from a table, then we don't need to use the WHERE clause.

Syntax: SQL> update <Table_Name> set Column_Name = Value;

SQL> update STUDENT set emp_name = „Naveena“;

SQL> select * from STUDENT;

EMP_ID	EMP_NAME	EXPERIENCE	SALARY	CONTACT_NUMBER	ADDRESS
101	naveena	12	36500	9491202987	Tirupati
102	naveena	11	34650	9442011432	Hyderabad
103	naveena	8	26580		Piler
104	naveena	6			Tirupati
105	naveena	5	12000	9491202169	Mumbai
106	naveena	2	8500	9441201342	Chennai

6 rows selected.

3. **Delete command:** The delete statement is used to delete rows from a table. Generally, delete statement removes one or more records form a table. There are three situations to use delete command.

(a). Delete single row.
(b). Delete multiple rows.
(c). Delete all rows.

Syntax: SQL> delete from <Table_Name> [where <condition>];

(a). **Delete single row:** This command is used to remove complete one single record.

Syntax: SQL> delete from <Table_Name> [where <condition>];

SQL> delete from STUDENT where emp_name = „Masthan“;

SQL> select * from STUDENT;

EMP_ID	EMP_NAME	EXPERIENCE	SALARY	CONTACT_NUMBER	ADDRESS
101	Mohammad	12	36500	9491202987	Tirupati
102	student123	11	34650	9442011432	Hyderabad
104	Abu	6			Tirupati
105	Muneer	5	12000	9491202169	Mumbai
106	Mujeeb	2	8500	9441201342	Chennai

5 rows selected.

(b). **Delete multiple rows:** This command used to remove all records when the condition is matched.

Syntax: SQL> delete from <Table_Name> [where <condition>];

SQL> delete from STUDENT where address = „Tirupati“;

2 rows deleted

SQL> select * from STUDENT;

EMP_ID	EMP_NAME	EXPERIENCE	SALARY	CONTACT_NUMBER	ADDRESS
102	student123	11	34650	9442011432	Hyderabad
105	Muneer	5	12000	9491202169	Mumbai
106	Mujeeb	2	8500	9441201342	Chennai

3 rows selected.

(c). **Delete all rows:** Delete the entire rows from the table. After this, no records left to display. The table will become empty.

Syntax: SQL> delete from <Table_Name>; (or) delete * from <Table_Name>;

SQL> delete from STUDENT;

SQL> select * from STUDENT;

No rows selected

(III) Data Query Language (DQL)

1. **Select command:** The Select statement is used to “fetch or access or retrieve records from one or more tables from the database on conditions”. The returns data is stored in table. This is also three ways to fetch the data into the table.

(a). Fetch single / particular column.
(b). Fetch multiple / selective columns.
(c). Fetch all / total columns.

Syntax: SQL> select Column_Name1, Column_Name2,
- - - - - - - - - - -
- - - - - - - - - - -
Column_Name n from <Table_Name> [where <condition>];

(a). **Fetch single / particular column:** The select statement is used to fetch any particular column into the table.

Syntax: SQL> select Column_Name from <Table_Name>;

SQL> select emp_name from STUDENT;

EMP_NAME
Mohammad

student123
Masthan Abu
Muneer Mujeeb

6 rows selected.

SQL> select emp_name from STUDENT where emp_name = 'student123';

EMP_NAME
----
student123

(b). **Fetch multiple / selective columns:** The select statement is used to fetch multiple columns into the table.

Syntax: SQL> select Column_Name1, Column_Name2, ……., Column_Name n from <Table_Name>;

SQL> select emp_name, contact_number from STUDENT;

EMP_NAME	CONTACT_NUMBER
Mohammad	9491202987
student123	9442011432
Masthan	
Abu	
Muneer	9491202169
Mujeeb	9441201342

6 rows selected.

(c). **Fetch all / total columns:** The select statement is used to fetch all fields /columns into the table.

Syntax: SQL> Select * from <Table_Name>;

SQL> select * from STUDENT;

EMP_ID	EMP_NAME	EXPERIENCE	SALARY	CONTACT_NUMBER	ADDRESS
101	Mohammad	12	36500	9491202987	Tirupati
102	student123	11	34650	9442011432	Hyderabad
103	Masthan	8	26580		Piler
104	Abu	6			Tirupati
105	Muneer	5	12000	9491202169	Mumbai
106	Mujeeb	2	8500	9441201342	Chennai

6 rows selected.

(IV) Transaction Control Language (TCL)

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only. These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping tables.

1. **Rollback command:** Rollback is used to a transaction in case of any error occurs.

Syntax: SQL> rollback;

SQL> delete from STUDENT where emp_id = 104;

1 row deleted.

SQL> select * from STUDENT;

EMP_ID	EMP_NAME	EXPERIENCE	SALARY	CONTACT_NUMBER	ADDRESS
101	Mohammad	12	36500	9491202987	Tirupati
102	student123	11	34650	9442011432	Hyderabad
103	Masthan	8	26580		Piler
105	Muneer	5	12000	9491202169	Mumbai
106	Mujeeb	2	8500	9441201342	Chennai

5 rows selected.

SQL> rollback;

Rollback complete.

SQL> select * from STUDENT;

EMP_ID	EMP_NAME	EXPERIENCE	SALARY	CONTACT_NUMBER	ADDRESS
101	Mohammad	12	36500	9491202987	Tirupati
102	student123	11	34650	9442011432	Hyderabad
103	Masthan	8	26580		Piler
104	Abu	6			Tirupati
105	Muneer	5	12000	9491202169	Mumbai
106	Mujeeb	2	8500	9441201342	Chennai

6 rows selected.

2. **Commit command:** Commit is used to permanently save the transaction into a database.

Syntax: SQL> commit;

SQL> delete from STUDENT where emp_id = 104; 1

row deleted.

SQL> select * from STUDENT;

EMP_ID	EMP_NAME	EXPERIENCE	SALARY	CONTACT_NUMBER	ADDRESS
101	Mohammad	12	36500	9491202987	Tirupati
102	student123	11	34650	9442011432	Hyderabad
103	Masthan	8	26580		Piler
105	Muneer	5	12000	9491202169	Mumbai
106	Mujeeb	2	8500	9441201342	Chennai

5 rows selected.

SQL> commit;

Commit complete.

SQL> rollback;

Rollback complete.

SQL> select * from STUDENT;

EMP_ID	EMP_NAME	EXPERIENCE	SALARY	CONTACT_NUMBER	ADDRESS
101	Mohammad	12	36500	9491202987	Tirupati
102	student123	11	34650	9442011432	Hyderabad
103	Masthan	8	26580		Piler
105	Muneer	5	12000	9491202169	Mumbai
106	Mujeeb	2	8500	9441201342	Chennai

5 rows selected.

3. **Savepoint command:** Savepoint creates points within the groups of transactions in which to rollback. A savepoint is a point in a transaction in which we can roll the transaction back to a certain point without rolling back the entire transaction.

Syntax: SQL> savepoint <savepoint name>;

This command is used only in the creation of savepoint among all the transactions. In general rollback is used to undo a group of transactions.

Syntax for rolling back to Savepoint command:
SQL> rollback to <savepoint name>;

SQL> savepoint spstudent1231;

Savepoint created.

SQL> delete from STUDENT where emp_id = 105;

SQL> select * from STUDENT;

EMP_ID	EMP_NAME	EXPERIENCE	SALARY	CONTACT_NUMBER	ADDRESS
101	Mohammad	12	36500	9491202987	Tirupati
102	student123	11	34650	9442011432	Hyderabad
103	Masthan	8	26580		Piler

104	Abu	6			Tirupati
106	Mujeeb	2	8500	9441201342	Chennai

5 rows selected.

SQL> savepoint spstudent1232;

Savepoint created.

SQL> rollback to spstudent1231;

Rollback complete.

SQL> select * from STUDENT;

EMP_ID	EMP_NAME	EXPERIENCE	SALARY	CONTACT_NUMBER	ADDRESS
101	Mohammad	12	36500	9491202987	Tirupati
102	student123	11	34650	9442011432	Hyderabad
103	Masthan	8	26580		Piler
104	Abu	6			Tirupati
105	Muneer	5	12000	9491202169	Mumbai
106	Mujeeb	2	8500	9441201342	Chennai

6 rows selected.

Release savepoint command: This command is used to remove a savepoint that we have created.

Syntax: SQL> release savepoint <savepoint name>;

Once a savepoint has been released, we can no longer use the rollback command to undo transactions performed since the last savepoint.

(IV) Data Control Language (DCL)

1. **Grant:** This command is used to "Grant the permissions on one table to other user (or) Gives user"s access privileges to Database". To Grant all permission, we can"t drop the table.

Syntax: Grant [type of privilege] ON [user name]. [Table_Name] to [user name] [with Grant option];

- **Grant all:** This command is used to Grant the permissions on all tables to other user. To Grant all permission, we can"t drop the table.

Syntax: Grant All [type of privilege] ON [user name]. [Table_Name] to [user name] [with Grant option];

2. **Revoke:** This command is used to "Take-out some or all permissions on a table from a user". (or) "Withdraw access privileges given with the GRANT command.

Syntax: Revoke [type of privilege] ON [user name]. [Table_Name] from [user name];

******* (Q). Define Select command / statement / operation in SQL?**

The Select command is a DML command. This command used to query the data in a table. A Query is a question, which helps to fetch the data from the database. Every select statement returns the result in the form of table after the execution. The general syntax for select command:

Syntax: SQL> SELECT [DISTINCT | ALL | *] {Column_Names}
FROM [{Table_Name [Alias] | View_Name}]...
[WHERE <condition>]
[GROUP BY <conditional Column_Name(s)>]
[HAVING condition]
[ORDER BY {Column_Name [ASC | DESC]} ...

- **SELECT command:** The SELECT clause is mandatory and carries out the relational. Select is identifies the columns in the table.

- **DISTINCT command:** It returns only single copy of each column values.

  Syntax: SQL> select distinct (Column_Name) from <Table_Name>;

  SQL> select distinct (marks) from stud_info;

  MARKS

  83
  87
  95
  85
  94
  99
  98
  92
  8 rows selected.

- *** / ALL (Astric) operator:** It indicate all Column_Names in the table. Select * means all the rows, columns including duplicate rows will be display. * is also known as wild card character.

- **FROM command:** The FROM clause is also mandatory. It identifies one or more tables and/or views from which to retrieve the column data displayed in a result table.

- **Table_Name:** It is the name of the table from which data is selected.

  Syntax: Select * from <Table_Name>;

SQL> select * from stud_info;

STUD_ID	STUD_NAME	SUBJECT	MARKS
101	student123	Computers	98
102	Shaik	Physics	85
103	Abu Shaik	Biology	98
104	Hashim	Zoology	94
105	naveena	Computers	99
106	Muneer	Botany	92
107	Mujeeb	Chemistry	95
108	Masthan	Maths	95
109	Ismail	Maths	83
110	Fayaz	Computers	87

10 rows selected

- **WHERE command:** The WHERE clause is optional and carries out the relational select operation. It specifies used to display only those rows for which the condition is true. We have to specify a condition after the WHERE clause. If the condition is not satisfied then no rows will be display from the table.

  Syntax: SQL> Select (Column_Names) from <Table_Name> [where <condition>];

  SQL> select * from stud_info where marks > 95;

STUD_ID	STUD_NAME	SUBJECT	MARKS
101	student123	Computers	98
103	Abu Shaik	Biology	98
105	naveena	Computers	99

3 rows selected

Use expressions in the WHERE command: Expressions can also be used in the WHERE clause of the select statement.

SQL> select * from STUDENT;

EMP_ID	EMP_NAME	EXPERIENCE	SALARY	CONTACT_NUMBER	ADDRESS
101	Mohammad	12	36500	9491202987	Tirupati
102	student123	11	34650	9442011432	Hyderabad
103	Masthan	8	26580	7892021949	Piler
104	Abu	6	22580	1234567890	Tirupati
105	Muneer	5	12000	9491202169	Mumbai
106	Mujeeb	2	8500	9441201342	Chennai

6 rows selected.

SQL> select emp_name, salary, (salary + salary * 0.1) as Bonus_Salary from STUDENT where (salary + salary * 0.1) > 30000;

EMP_NAME	SALARY	BONUS_SALARY
Mohammad	36500	40150
student123	34650	38115

- **GROUP BY command:** The GROUP BY clause is optional. It is used with select statement to combine a group of rows based on the values of a particular column or expression. SQL groups the result after it retrieves the rows from a table.

  Syntax: select (Column_Names), <aggregation function>(Column_Name) from <Table_Name> GROUP BY <conational Column_Name>;

  SQL> select subject, avg (marks) from stud_info group by subject;

SUBJECT	AVG(MARKS)
Physics	85
Botany	92
Chemistry	95
Zoology	94
Maths	89
Computers	94.66667
Biology	98

7 rows selected

- **HAVING command:** The optional HAVING clause. It is used to filter database on the group functions. It is used to specify which groups are to be displayed, that is, restrict the groups that we return on the basis of group function. Simply, conditional retrieval of rows from a grouped result is possible with the HAVING clause.

  This is similar to WHERE condition but is used with group functions. Group functions cannot be used in WHERE clause but can be used in HAVING clause.

  Syntax: select (Column_Names), <aggregation function> (Column_Name) from <Table_Name> GROUP BY <conational Column_Name> HAVING <condition>;

  SQL> select subject, avg (marks) from stud_info group by subject having count (subject) >1;

SUBJECT	AVG(MARKS)
Maths	89
Computers	94.66667

2 rows selected

- **ORDER BY command:** The ORDER BY clause is optional. It is used to sort the rows in a table be default sorting order is „ascending“ order. We can use the keyword „ASC“. The keyword „DESC“ is used to sort in descending order. The ORDER BY clause must be the last clause in a select statement.

  Syntax: select (Column_Names) from <Table_Name> ORDER BY (Column_Name) ASC | DESC;

  SQL> select stud_name, subject, marks from stud_info order by (marks) desc;

STUD_NAME	SUBJECT	MARKS
naveena	Computers	99
Abu Shaik	Biology	98
student123	Computers	98
Masthan	Maths	95
Mujeeb	Chemistry	95
Hashim	Zoology	94
Muneer	Botany	92
Fayaz	Computers	87
Shaik	Physics	85
Ismail	Maths	83

10 rows selected.

***** (Q) Explain Aggregate Functions or Grouped Functions in SQL?**

Aggregate functions are the DDL statements. Aggregate functions also called Grouped functions are built-in SQL functions that operate on groups of rows and return one value for the entire group or table. Aggregate functions are used to produce summarized results. The SQL aggregate functions are:

1. COUNT()
2. COUNT(*)
3. SUM()
4. AVG()
5. MAX()
6. MIN()
7. STDDEV()
8. VARIANCE()

Consider the example table of Stud_Info:

SQL> select * from stud_info;

STUD_ID	STUD_NAME	SUBJECT	MARKS
101	student123	Computers	98
102	Shaik	Physics	85
103	Abu Shaik	Biology	98
104	Hashim	Zoology	94
105	naveena	Computers	99
106	Muneer	Botany	92
107	Mujeeb	Chemistry	95
108	Masthan	Maths	95
109	Ismail	Maths	83
110	Fayaz	Computers	87

10 rows selected

1. **COUNT():** COUNT function is used to Count the number of rows in a database table.

Syntax: SQL> select COUNT (DISTINCT Column_Name) from <Table_Name> [where <condition>];

Example 1: Find the number of subjects has in stud_info table.

SQL> select count (subject) from stud_info;

COUNT (SUBJECT)

10

Example 2: Find the only number of computers subject have in stud_info table.

SQL> select count (subject) as computer from stud_info where subject = 'Computers';

COMPUTER

3

Example 3: Find the number of marks which has non redundant in stud_info table;

SQL> select count (distinct marks) as marks from stud_info;

MARKS

8

2. COUNT(*): COUNT(*) function is used to count the total all number of records (repeated/duplicate) in a database table.

Syntax: SQL> select COUNT(*) from <Table_Name>;

Example: Find the total number of records has in stud_info table;

SQL> select count (*) from stud_info;

COUNT(*)

10

2. **SUM():** SUM() function is used to get the sum of the values (total values) of from column name.

Syntax: SQL> select sum (Column_Name) from <Table_Name>;

Example: Compute the total marks for all students in stud_info table.

SQL> select sum (marks) as Totla_Marks from stud_info;

TOTAL_MARKS

926

3. **AVG():** AVG() function is used to get the average of the values of from column name.

Syntax: SQL> select avg (Column_Name) from <Table_Name>;

Example: Compute the average marks about all students in stud_info table.

SQL> select avg (marks) as Average_Marks from stud_info;

AVERAGE_MARKS

92.6

4. **MAX():** MAX() function is used to find the maximum value (highest) of from column name.

Syntax: SQL> select max (Column_Name) from <Table_Name>;

Example: Compute the highest marks about all students in stud_info table.
SQL> select max (marks) as Maximum_Marks from stud_info;

MAXIMUM_MARKS

99

5. **MIN():** MIN() function is used to find the minimum value (least) of from column name.

Syntax: SQL> select min (Column_Name) from <Table_Name>;

Example: Compute the lowest marks about all students in stud_info table.

SQL> select min (marks) as Minimum_Marks from stud_info;

MINIMUM_MARKS

83

6. **STDDEV():** This function is used to compute the statistical standard deviation of the column name.

Syntax: SQL> select stddev (Column_Name) from <Table_Name>;

Example: Compute the standard deviation of student marks in stud_info

table.

SQL> select stddev (marks) as Standard_Deviation from stud_info;

STANDARD_DEVIATION

5.71936282

7. **VARIANCE ():** This function is used to compute the statistical variance of the column name.

Syntax: SQL> select variance (Column_Name) from <Table_Name>;

Example: Compute the variance of student marks in stud_info table.

SQL> select variance (marks) as Variance from stud_info;

```
VARIANCE
-------------
32.7111111
```

(Q) Explain the SQL Operators?

"Operators are a symbol, which represent some particular actions. Operator operates on operands". There are different types of operators used in SQL, namely Arithmetic Operators, Comparison Operators, Logical Operators and Special Operators. These operators are used mainly in the WHERE and HAVING clause to filter the data to be selected.

1. **Arithmetic Operators:** Arithmetic operators are used in SQL expressions to addition, subtraction, multiplication, and division data values. The result of this expression is a numeric value.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division

2. **Comparison Operators:** Comparison operators are used to compare the column data with specified values in a condition. Comparison operators are also used along with the SELECT statement to filter data based on specific conditions. The result of a comparison is True, False or Unknown. The below table describes each comparison operator.

Operator	Description
=	Equality
!= or <>	Not Equal to
>	Greater than
<	Less than
>=	Greater than Equal
<=	Less than Equal

3. **Logical Operators:** There are three logical operators namely, AND, OR and NOT. These operators compare two or more than two conditions at a time to determine whether a row can be selected for the output. When retrieving data using a SELECT statement, we can use logical operations in the WHERE clause, which allows we to combine more than one condition.

Operator	Description
AND	Return True; If Both conditions are True; otherwise return False.
OR	Return True; If Either / at least one of the conditions is True; otherwise return False

NOT IN	Return True; If the condition is False; Otherwise return False.

1. **Special Operator:** There are some special operators available in SQL which are uses to enhance the search capabilities of a SQL query. There are LIKE, IN and BETWEEN operators.

Operator	Description
LIKE	Matching a pattern from a column. Column value is similar to specified characters.
IN	Checking a value in a set. Column value is equal to any one of a specified set of values.
BETWEEN	Checking a value within a range. Column value is between two values, including the end values specified in the range.

1. Find the all student names from „stud_info“ table, whose names begins with „S“ only?

SQL> select stud_name from stud_info where stud_name LIKE 'S%';

STUD_NAME

Shaik
naveena

2. Find the all student names from „stud_info“ table, whose names begins with „S“ and end with „d“ only?

SQL> select stud_name from stud_info where stud_name LIKE 'S%_d%';

STUD_NAME
---
naveena

3. Find the all student names from „stud_info“ table, whose names end with „k“ only?

SQL> select stud_name, subject from stud_info where stud_name LIKE '_%k';

STUD_NAME

Shaik
Abu Shaik

4. Find the name of the student and subject from „stud_info“ table, who elective only computers and biology subjects?

SQL> select stud_name, subject from stud_info where subject IN („Computers“, 'Biology');

STUD_NAME	SUBJECT
student123	Computers
Abu Shaik	Biology
naveena	Computers
Fayaz	Computers

5. Find the all student names, subject and their marks from „stud_info“ table, which are secure marks between 90 to 98 marks only?

SQL> select stud_name, subject, marks from stud_info where marks between 90 and 98;

STUD_NAME	SUBJECT	Marks
student123	Computers	98
Abu Shaik	Biology	98
Hashim	Zoology	94
Muneer	Botany	92
Mujeeb	Chemistry	95
Masthan	Maths	95

******* (Q) Explain various Integrity / Imposition Constraints in SQL?**

SQL uses integrity constraints to prevent invalid data entry into the tables of the database. Constraints are a part of the table definition that is used to limit the values entered into its columns. Constraints are basically used to impose rules on the table, whenever a row is inserted, updated or deleted from the table.

The constraints clause is used to define the integrity constraints in "Create Table or Alter Table" statements. The constraints can be defined in two ways:

1. Column-Level Constraints.
2. Table-Level Constraints.

1. **Column Level Constrains:** The Constraints can be specified immediately after the column definition. This is called Column-Level constraints.

General Syntax of Column Level Constraints:

Syntax: create table <table name> (column name 1 data type (size) constrain <constrain name>
primary key, column name 2 data type(size) constrain <constrain name>
references referenced_table [(primary_column_name of referenced table)],
column name 3 data type(size) constrain <constrain name>
check (<condition>), column name 4 datatype (size) NOT NULL);

2. **Table Level Constrains:** The Constraints can be specified after all the columns are defined. This is called Table-Level Constraints. Therefore the table level constraints are defined at the end of the create table statement which can impose rules on any columns in the table. This syntax can define any type of constraints except NOT NULL.

General Syntax of Table Level Constraints:

Syntax: create table <table name> (column name 1 data type(size), column name 2 data
type (size),column name 3 data type(size),….column name n
data type(size), constrain<constrain name> primary key (column name
1) constrain <constrain name> foreign key(foreign column name)
references referenced_table [(primary _column _name of
referenced_table)], constrain <constrain name> check (<condition>));

Types of Integrity Constraints:

1. NOT NULL
2. Unique
3. Primary Key
4. Foreign / Reference Key
5. Check
6. Default

1. **NOT NULL:** The NOT NULL constraint enforces a column to always contain a value. By default, a table column can hold NULL values. The NOT NULL constraint enforces a column to not accept NULL values. This constraint is placed immediately after the datatype of a column.

Example: SQL> create table student123 (Emp_Id Number (5) **NOT NULL,**
Emp_Name Varchar2 (30) **NOT NULL,**
Address Varcahr2 (40),
Phone_No Number (10) **NOT NULL**);
SQL> insert into student123 values („& Emp_Id", „& Emp_Name", „& Address", „&
Phone_No"); Enter value for Emp_Id:
Enter value for Emp_Name: Shaik Mohammad
student123 Enter value for Address:
Enter value for Phone_No: 9491202987

In the above example it is clear that when we try to insert a null value into „Emp_Id" we get error message. But „Address" have empty value it"s not any error, because it"s not have any constraint.

2. **Unique:** The Unique constraint imposes that every value in a column or set of columns be unique. It means that no two rows of a table can have duplicate values in a specified column or set of columns.

Example: SQL> create table student123 (Emp_Id Number (5) NOT NULL,
Emp_Name Varchar2 (30) NOT NULL,
Address Varcahr2 (40),
Phone_No Number (10) **UNIQUE**);

Case 1: SQL> insert into student123 values („& Emp_Id", „& Emp_Name", „& Address", „&
Phone_No"); Enter value for Emp_Id: 10101
Enter value for Emp_Name: Shaik Mohammad
student123 Enter value for Address: Tirupati
Enter value for Phone_No: 9491202987
Case 2: SQL> insert into student123 values („& Emp_Id", „& Emp_Name", „& Address", „&

Phone_No"); Enter value for Emp_Id: 10102

Enter value for Emp_Name: Mohammad student123
Enter value for Address: Chittoor
Enter value for Phone_No: **9491202987**

In the above case 2 it is clear that when we try to insert duplicate values into „Phone_No“ we get error message. But „Emp_Name“ have duplicate value it“s not any error, because it“s not have unique constraint.

Example: UNIQUE (Emp_Id, Phone_No);

3. **Primary Key:** A Primary Key constraint creates a primary key for the table. Only one primary key can be created for each table. The primary key uniquely identifies every row in the table. In simple terms primary key is a combination of UNIQUE and NOT NULL. The primary key constraints can be defined at the column-level or table-level.

Example: SQL> create table student123 (Emp_Id Number (5) **PRIMARY KEY,**
Emp_Name Varchar2 (30) NOT NULL,
Address Varcahr2 (40),
Phone_No Number (10) UNIQUE);

Case 1: SQL> insert into student123 values („& Emp_Id“, „& Emp_Name“, „& Address“, „& Phone_No“); Enter value for Emp_Id: 10101
Enter value for Emp_Name: Shaik Mohammad
student123 Enter value for Address: Tirupati
Enter value for Phone_No: 9491202987

Case 2: SQL> insert into student123 values („& Emp_Id“, „& Emp_Name“, „& Address“, „& Phone_No“); Enter value for Emp_Id: **10101**
Enter value for Emp_Name: Mohammad
student123 Enter value for Address: Hyderabad
Enter value for Phone_No:

In the above case 2 it is clear that when we try to insert duplicate values into „Emp_Id“ we get error message. But „Phone_No“ have NULL value it“s not any error, because it“s have only unique constraint.

A Composite Primary Key is created by using the table-level definition:

Example: SQL> create table student123 (Emp_Id Number (5),

Emp_Name Varchar2 (30) NOT NULL,
Address Varcahr2 (40),
Phone_No Number (10),
Primary Key (Emp_Id, Phone_No));

4. **Foreign / Reference Key:** The Foreign / Reference integrity constraint, when a foreign key in one relation references primary key in another table, the foreign key value must match with the primary key value. In other words, the referential integrity say“s „pointed to“ information must exist.

The foreign key is defined in the „child table“, and the table containing the referenced column is the „parent table“. Foreign key constraint can be defined at the column-level or table-level. A composite foreign key must be created by using the table-level definition.

- **Foreign Key:** It is used to define the column in the child table at the table-level constraint.
- **References:** It identifies the parent table and primary key column in the parent table.

Example: Consider the parent table is „student123_DEPT“ and child table is „STUDENT, the child

table which has foreign key which references primary key in parent table. It is to be noted that the parent table should be created first, then the child table.

- **Parent table: student123_DEPT:** □ Dept_Id, Dept_Name, Location

- **Child table: STUDENT:** □ Emp_Id, Department_Id, Emp_Name

student123_DEPT

Dept_Id	Dept_Name	Location
D100	Electrical	Rompicherla
D101	Civil	Chittoor
D102	Computer	Tirupati

STUDENT

Emp_Id	Department_Id	Emp_Name
E505	D101	Shaik
E503	D100	Mohammad
E507	D102	student123

Parent table: student123_DEPT table is created with „Dept_Id" as primary key.

SQL> create table student123_DEPT (Dept_Id varchar2(5) **Primary key,**
Dept_Name
varcahr2(15),
Location
varchar2(20));

Child table: STUDENT table is created with „Department_Id" as references „Dept_Id".

SQL> create table STUDENT (Emp_Id varchar2(5),
Department_Id
varchar2(5), Emp_Name
varchar2(20),
Foreign key (Department_Id references

student123_DEPT(Dept_Id)); SQL> insert into STUDENT values („E502", „D105", „Shaik Abu");

Here the values are not able to insert into the STUDENT table, because we have try to insert „Department_Id" into the STUDENT table (child table) which is not matched with „Dept_Id" of the student123_DEPT table (parent table).

5. **Check:** The „Check" constraint defines a condition that each row must satisfy. It is used to limit the value range that can be placed in a column. The check constraint is added to the declaration of the attribute. Attribute value check is checked only when the value of the attribute is inserted or updated.

Example 1: SQL> create table voter (Voter_Name varchar2(15),
Voter_Age number(3),
Voter_Address
varchar2(25), **CHECK**
(Voter_Age >=18));

SQL> insert into voter values („Mohammad student123", **„17"**, „Tirupati");

In the above example we can observe that we can try to insert a value which violates the check constraint, we get error message.

Example 2: SQL> create table person (Name varchar2(15),
Gender char(8) **CHECK (Gender IN („M“, „F“)),**
Address varchar2(25));

SQL> insert into person values („Mohammad student123“, **„Male“**, „Tirupati“);

In the above example we can observe that we can try to insert a gender value „Male“ instated of „M“ which violates the check constraint, we get error message.

6. **Default:** The „Default“ constraint is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified. The most common default value is NULL. This can be used with columns not defined with a NOT NULL. Oracle automatically inserted the value specified as the default value when no value is specified for the particular column during inserting.

Example 1: SQL> create table emp (Emp_No number(4),
Emp_Name varchar2(15),
Hiredate **DEFAULT**
Sysdate, Salary
number(7, 2),
Primary key(Emp_No));

Example 2: SQL> create table student123 (First_Name
varchar2(15),
Last_Name varchar2(15),
Address varchar2(15) **DEFAULT „TIRUPATI“,**
Phone_No number(10));

********** (Q) Explain Join Operations in SQL?**

SQL Joins are used to combine data from two or more tables based on a related column.

- Retrieving connected data stored across multiple tables.
- Matching records using common columns.
- Improving data analysis by combining related information.
- Creating meaningful result sets from separate tables.

Types of SQL Joins

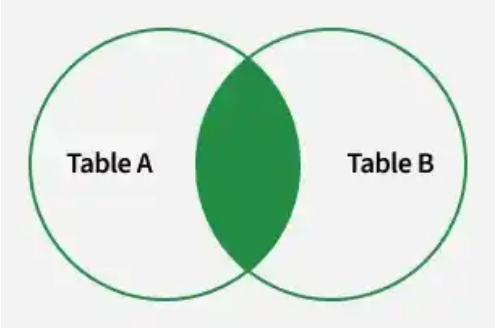
SQL joins are categorized into different types based on how rows from two tables are matched and combined.

1. INNER JOIN - INNER JOIN is used to retrieve rows where matching values exist in both tables.

- Combining records based on a related column.
- Returning only matching rows from both tables.
- Excluding non-matching data from the result set.
- Ensuring accurate data relationships between tables.

Syntax: SELECT table1.column1,table1.column2,table2.column1,.... FROM table1

INNER JOIN table2 ON table1.matching_column = table2.matching_column;



Inner join

Student Table:

ROLL_NO	Name	Address	Age
1	Liam	London	18
2	Sofia	Berlin	19
3	Mateo	Madrid	20
4	Emma	Paris	21
5	Noah	Rome	24

StudentCourse Table:

COURSE_ID	ROLL_NO
1	1
2	2
3	3
6	7

Eg: SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE

FROM Student

INNER JOIN StudentCourse

ON Student.ROLL_NO = StudentCourse.ROLL_NO;

Course_ID	Name	Age
1	Liam	18
2	Sofia	19
3	Mateo	20

2. LEFT JOIN - LEFT JOIN is used to retrieve all rows from the left table and matching rows from the right table.

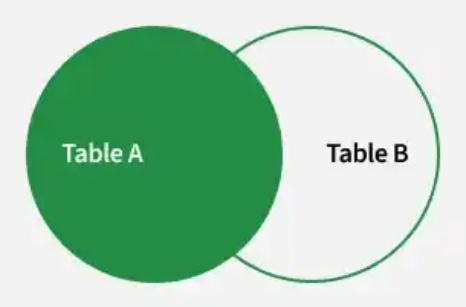
- Returning all records from the left table.
- Showing matching data from the right table.
- Displaying NULL values where no match exists in the right table.
- Performing outer joins, also known as LEFT OUTER JOIN.

Syntax: SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

LEFT JOIN table2

ON table1.matching_column = table2.matching_column;



Left Join

Eg: SELECT Student.NAME,StudentCourse.COURSE_ID

FROM Student

LEFT JOIN StudentCourse

ON StudentCourse.ROLL_NO = Student.ROLL_NO;

Output:

NAME	COURSE_ID
Liam Johnson	1
Emma Wilson	2
Noah Smith	3
Olivia Brown	
Ava Davis	

3. RIGHT JOIN - RIGHT JOIN is used to retrieve all rows from the right table and the matching rows from the left table.

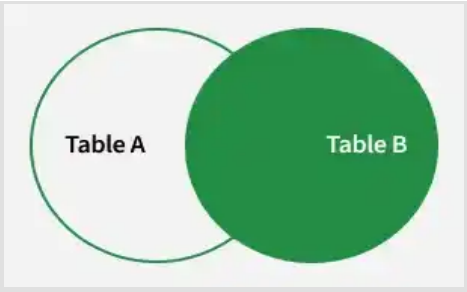
- Returning all records from the right-side table.
- Showing matching data from the left-side table.
- Displaying NULL values where no match exists in the left table.
- Performing outer joins, also known as RIGHT OUTER JOIN.

Syntax: SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

RIGHT JOIN table2

ON table1.matching_column = table2.matching_column;



Right Join

Eg: SELECT Student.NAME,StudentCourse.COURSE_ID

FROM Student

RIGHT JOIN StudentCourse

ON StudentCourse.ROLL_NO = Student.ROLL_NO;

Output:

NAME	COURSE_ID
Liam Johnson	1
Emma Wilson	2
Noah Smith	3
	6

4. FULL JOIN - FULL JOIN is used to combine the results of both LEFT JOIN and RIGHT JOIN.

- Returning all rows from both tables.
- Showing matching records from each table.
- Displaying NULL values where no match exists in either table.
- Providing complete data from both sides of the join.

Syntax: SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

FULL JOIN table2

ON table1.matching_column = table2.matching_column;



Full Join

Eg: SELECT Student.NAME,StudentCourse.COURSE_ID

FROM Student

FULL JOIN StudentCourse

ON StudentCourse.ROLL_NO = Student.ROLL_NO;

Output :

NAME	COURSE_ID
Liam Johnson	1
Emma Wilson	2
Noah Smith	3
Olivia Brown	
Ava Davis	
	6

***** (Q) Explain Set Operators in SQL?**

Set operators are used to combine information of similar type from one or more than one table. In SQL Set operators can be generally classified into following categories:

1. Union
2. Union all
3. Intersect
4. Minus

SQL> create table student123_shaik (Emp_Id number(5), Emp_Name varchar2(15), Salary

number(5)); Table created.

Inserting records into the table:

SQL> insert into student123_shaik values (100,

'Shaik', 50000); 1 row created.

View the resulting table:

SQL> select * from student123_shaik;

EMP_ID	EMP_NAME	SALARY
100	Shaik	50000
101	Mohammad	75000
102	student123	60000
103	Shaik Abu	55000
104	SMF	48000

Table_Name 2: shaik_student123. **Column_Name:** Emp_Id, Emp_Name, Salary

Creating Table Structured:

SQL> create table shaik_student123 (Emp_Id number (5), Emp_Name varchar2(15), Salary number(5)); Table created.

Description about Table:

SQL> desc shaik_student123;

Name	Null?	Type
EMP_ID		NUMBER(5)
EMP_NAME		VARCHAR2(15)
SALARY		NUMBER(5)

Inserting records into the table:

SQL> insert into shaik_student123 values (101,

'Mohammad', 75000); 1 row created.

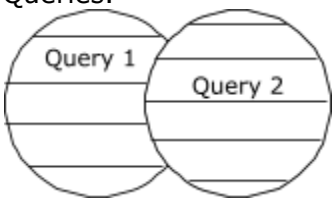
View the resulting table:

SQL> select * from shaik_student123;

EMP_ID	EMP_NAME	SALARY
101	Mohammad	75000
103	Shaik Abu	55000
105	Muneer	54000
106	Mujeeb	50000
107	Sudheer	58000

1. UNION Operator:

- SQL Union operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.
- To use „Union“, each SELECT must have the same number of columns selected, the same number of column expressions, the same datatype, and have them in the same order, but they do not have to be the same length.
- Aggregate functions can“t be use with Union clause.
- Union“s can“t be use in Sub-Queries.



Syntax for UNION Operator:

SQL> Select <Column names> from <Table Name1> where [<condition>]
UNION
Select <Column names> from <Table Name2> where [<condition>];

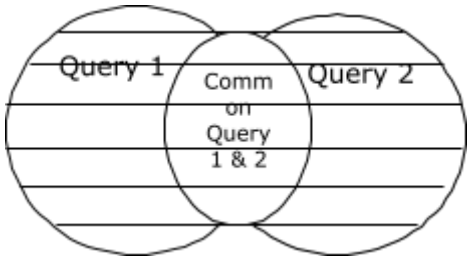
Example: SQL> select Emp_Id, Emp_Name, salary from student123_shaik
UNION
select Emp_Id, Emp_Name, salary from shaik_student123;

EMP_ID	EMP_NAME	SALARY

100	Shaik	50000
101	Mohammad	75000
102	student123	60000
103	Shaik Abu	55000
104	SMF	48000
105	Muneer	54000
106	Mujeeb	50000
107	Sudheer	58000

2. UNION ALL Operator:

- The UNION ALL operator is used to combine the results of two SELECT statements including duplicate rows.
- The same rules that apply to UNION apply to the UNION ALL operator.



Syntax for UNION ALL Operator:

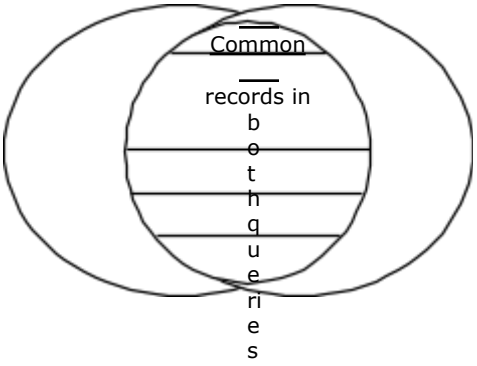
SQL> Select <Column names> from <Table Name1> where [<condition>]
UNION ALL
Select <Column names> from <Table Name2> where [<condition>];

Example: SQL> select Emp_Id, Emp_Name, salary from student123_shaik
UNION ALL
select Emp_Id, Emp_Name, salary from shaik_student123;

EMP_ID	EMP_NAME	SALARY
100	Shaik	50000
101	Mohammad	75000
102	student123	60000
103	Shaik Abu	55000
104	SMF	48000
101	Mohammad	75000
103	Shaik Abu	55000
105	Muneer	54000
106	Mujeeb	50000
107	Sudheer	58000

3. INTERSECT Operator:

- The SQL Intersect operator is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement.
- This means „Intersect“ returns only common rows returned by the two SELECT statements.



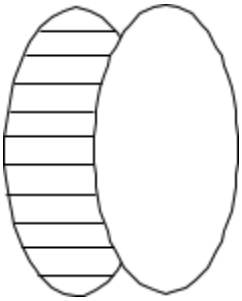
Syntax for INTERSECT Operator:

SQL> Select <Column names> from <Table Name1> where [<condition>]
INTERSECT
Select <Column names> from <Table Name2> where [<condition>];

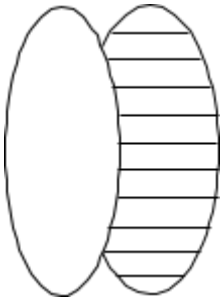
Example: SQL> select Emp_Id, Emp_Name, salary from
student123_shaik
INTERSECT
select Emp_Id, Emp_Name, salary from shaik_student123;

EMP_ID	EMP_NAME	SALARY
101	Mohammad	75000
103	Shaik Abu	55000

4. **MINUS Operator:** The Minus Operator combine the result of the two query and returns only those values selected by the "First Query and not the Second Query".



A – B



B – A

Syntax for MINUS Operator:

SQL> Select <Column names> from <Table Name1> where [<condition>]
MINUS
Select <Column names> from <Table Name2> where [<condition>];

Example (A – B): SQL> select Emp_Id, Emp_Name, salary from
student123_shaik
MINUS
select Emp_Id, Emp_Name, salary from shaik_student123;

EMP_ID	EMP_NAME	SALARY
100	Shaik	50000
102	student123	60000

104	SMF	48000

Example (B – A): SQL> select Emp_Id, Emp_Name, salary from shaik_student123
MINUS
select Emp_Id, Emp_Name, salary from student123_shaik;

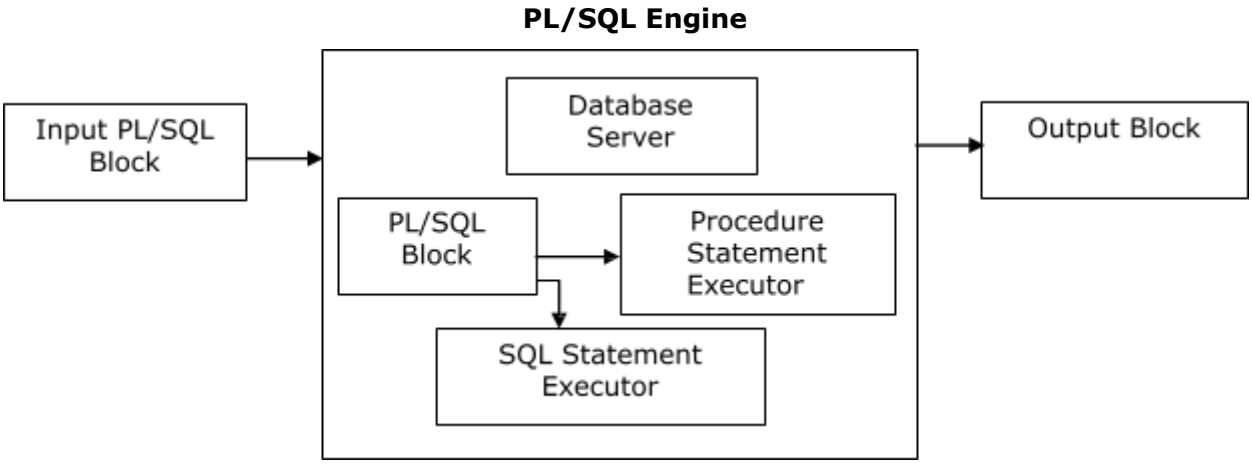
EMP_ID	EMP_NAME	SALARY
105	Muneer	54000
106	Mujeeb	50000
107	Sudheer	58000

UNIT – V
Procedural Language / Structured Query Language (PL/SQL).
Chapter – 1
Introduction to PL/SQL

Introduction of PL/SQL:

- PL/SQL is provided by Oracle as a procedural extension to SQL.
- PL/SQL is a block structured language. This means a PL/SQL program is made up of blocks, where block is a smallest piece of PL/SQL code.
- SQL is a declarative language the statements have no control to the program and can be executed in any order.
- PL/SQL arose from the desire of programmers to have a language structure that was more familiar than SQL"s purely declarative nature.
- PL/SQL supports almost all programming constructs like variables, conditional statements, looping etc.,

(Q) Explain Basic Structure or Architecture of PL/SQL?



PL/SQL Block Structure: The Programming area in PL/SQL is called block. This block mainly contains three parts. They are:

1. Declarative Part.
2. Executable Part.
3. Exception Handling part.

Structure:

```
SQL> Declare
        ..............  }
        ..............  }  Variable Declaration
        ..............  }
     Begin
          Statement 1  }
          Statement 2  }  Executable Part
           ..............  }
     End;
   Exception              Error Handling Part
```

1. **Declarative Part:** Declarative section of PL/SQL block starts with the keyword „Declare“. This section is used to declare any placeholders (storage area) like variables, constants, records and cursors, which are used to manipulate data in the executable section.

2. **Executable Part:** The executable section of a PL/SQL block starts with the keyword „Begin“ and ends with „End“. This section where the program logic is written to perform any task. The programmatic constructs like loops, conditional statement and SQL statements from the part of executable section. It is a mandatory section.

3. **Exception Part:** The exception section of a PL/SQL block starts with the keyword „Exception“. Any errors in the program can be handled in this section, so that the PL/SQL blocks terminates gracefully. If the PL/SQL block contains exceptions that cannot be handled, the block terminates immediately with errors.

Input & Output Statement in PL/SQL:

1. **Input Statement:** The Input statement in PL/SQL is "&" Operator. The „&“ Operator it is to give the Input values to the Variables.

Example: a : = & a;
x : = & x; ……

2. Output Statement: The Output Statement in PL/SQL is:

dbms_output.put_line(„Message“/Variables);

Example: SQL> dbms_output.put_line ('WELCOME');

Advantages of PL/SQL:

1. **Support for SQL:** PL/SQL allows the use all SQL commands, as well as all SQL functions, operations and datatypes. So we can manipulate Oracle data flexibility and safely.

2. **Block Structured:** PL/SQL is a block-structured language. Each program written in PL/SQL is written as a block. Blocks can also be nested. Each block is meant for a particular task. PL/SQL blocks can be stored in the database and reused.

3. **Modularity:** Modularity means divide an application or a process into manageable, well defined modules. PL/SQL allows process to be divided into different modules such as procedures and functions, called sub-programs.

4. **Control Structures:** PL/SQL allows control structures like If-statement, for loop-statement, while loop-statement to use in the block. Control structures are most important in PL/SQL extension to SQL. It controls the procedural flow of the program, deciding if and when SQL and other actions are to be executed.

5. **Integration:** As PL/SQL is the product of Oracle Corporation. So, it integrates well with SQL*Plus and other application development products of Oracle. It bridges the gap between convenient access to database technology and the need for procedural programming capability.

6. **Portability:** Applications written in PL/SQL are portable to any platform on which Oracles runs. Once we write a program in PL/SQL, it can be used in any environment without any changes.

7. **Error handling:** PL/SQL handles errors or exceptions effectively during the execution of a PL/SQL program. Once an exception is caught, specific actions can be taken depending upon the type of the exception or it can be displayed to the user with a message.

Disadvantages: SQL is a powerful tool for accessing the database but it suffers from some deficiencies as follows:

- SQL statements can be executed only one at a time. Every time to execute a SQL statement, a call is made to Oracle Engine, thus it results in an increase in database overheads.

- While processing an SQL statement. If an error occurs, Oracle generates its own error message, which is some time difficult to understand.
- If a user want to display some other meaningful error message. SQL does not have provision for that

(Q) Explain about various datatypes using in PL/SQL?

A datatype specifies the space to be reserved in the memory, type of operations that can be performed, and valid range of values. PL/SQL supports all the built-in SQL datatypes. A part from those datatypes, PL/SQL provides some other datatypes. Some commonly used PL/SQL datatypes are as follows:

1. Number
2. Char
3. Varchar2
4. Date
5. Boolean
6. %Type
7. %Rowtype

1. **Number:** The Number datatype to store fixed or floating point number of virtually any size. we can specify precision, which is the total number of digits, and scale, which determines where rounding, occurs.

Syntax: number (Precision, scale);

2. **Char:** The Char datatype to store fixed-length character data.

Syntax: char (maximum length);

3. **Varchar2:** The Varchar2 datatype to store varying-length character data.

Syntax: varchar2 (maximum length);

4. **Date:** The Date datatype to store fixed-length data value.

Syntax: date;

5. **Boolean:** A Boolean datatype is assigned to those variables, which are required for logical operations.

Syntax: Boolean;

6. **%Type:** To avoid type and size conflict between a variable and the column of a table, the attribute „%type“ is used. Advantage of this method of defining a variable is that, whenever the type and / or size of a column in the table are changed, it is automatically reflected in the variables declaration.

Syntax: <new column name> <table name>.<existing column name>%type;

Example: Temp_name emp.ename%type;

Here, the new variable name as „Temp_name“ will be of the same size and type as that of the „ename“ column of „emp“ table.

7. **%Rowtype:** The %Rowtype attribute provide a record type that represents a row in a database table or view. The record can stored an entire row of data selected from the table or fetched from a cursor or cursor variable. In case, variables for the entire row of a table need to be declared, then instead of declaring then individually, the attribute „%rowtype“ is used.

Syntax: <variable name> <table name>%rowtype;

Example: emp_row emp%rowtype;

Here, the variable „emp_row“ will be a composite variable, consisting of the column names of the table as its members.

(Q). Explain Operators Precedence in PL/SQL?

The operators within an expression are done in a particular order depending on their precedence (priority). Table below listed the operator“s level of precedence from top to bottom. Operators with higher precedence are applied first, but if parentheses are used, expression within innermost parenthesis is evaluated first.

Operators	Descriptions
( )	Function Call
++, --	Incremental and Detrimental
* / %	Multiplication, Division, Modulo Division
+ -	Addition and Subtraction
<, >, < =, > =	Less than, Greater Than, Less than or equal to, greater than or equal.
== , !=	Equality and Inequality
AND	Logical AND / Conjunction
OR	Logical OR / Disjunction
NOT IN	Logical NOT
LIKE	Comparison of Like
BETWEEN	Comparison of Between
IN	Comparison of In

(Q) Explain how to declare the variables and constants in PL/SQL?

(I). Variables in PL/SQL:

- Variables are placeholders (storage area) used to hold the data values that can changes through the PL/SQL block.
- Each variable in PL/SQL has a specific datatype, which determines the size and lawet of the variables memory, the range of values that can be stored within that memory and he set of operations that can be applied to the variables.

Variable declaration on PL/SQL: PL/SQL variables must be declared in the declaration section of PL/SQL block. When we declare a variable, PL/SQL allocates memory for the variable"s value and the storage location is identified by the variable name.

Syntax: <variable name> datatype (size); **(or)** <variable name> datatype [Not Null:= Value];

Example: SQL> Declare
Salary Number (5);
The value of a variable can change in the execution or exception section of the PL/SQL block. We can assign values to variables in two ways given below:

(a). **Direct method:** We can directly assign values to variables:

Syntax: <variable name>:= Value;

(b). **Select...Into Statement:** We assign values to variables directly from the database columns by using a Select…Into statement.

Syntax: select column name into <variable name> from <table name> where [<condition>];

Example: The below program will get the salary of an employee with id '9491202987'.

```
SQL> DECLARE
       var_salary number(6);
       var_emp_id number(10):= 9491202987 ;
     BEGIN
        SELECT salary INTO var_salary FROM employee WHERE emp_id = var_emp_id;
        dbms_output.put_line (var_salary);
        dbms_output.put_line ('The employee '|| var_emp_id || ' has salary ' || var_salary);
    END;
  /
```

Scope of Variables:

PL/SQL allows the nesting of Blocks within Blocks i.e., the Execution section of an outer block can contain inner blocks. Therefore, a variable which is accessible to an outer Block is also accessible to all nested inner Blocks. The variables declared in the inner blocks are not accessible to outer blocks. Based on their declaration we can classify variables into two types.

Local Variables: These are declared in an inner block and cannot be referenced by outside blocks.

Global Variables: These are declared in an outer block and can be referenced by itself and by its inner blocks.

```
DECLARE
  global_var NUMBER; -- global variable
BEGIN
  -- PL/SQL code using global_var
  DECLARE
    local_var NUMBER; -- local variable
  BEGIN
    -- PL/SQL code using local_var and global_var
  END;
  -- Here you can't access local_var
    END;
```

```
Eg: DECLARE
  -- Global variables
  num1 number := 95;
  num2 number := 85;
BEGIN
  dbms_output.put_line('Outer Variable num1: ' || num1);
  dbms_output.put_line('Outer Variable num2: ' || num2);
        DECLARE
          -- Local variables
          num1 number := 195;
          num2 number := 185;
        BEGIN
          dbms_output.put_line('Inner Variable num1: ' || num1);
          dbms_output.put_line('Inner Variable num2: ' || num2);
        END;
END;
/
```

(II). **Constants in PL/SQL:** As the name implies a constant is a value used in a PL/SQL block that remains unchanged throughout the program. A constant is a user-defined literal value. we can declare a constant and use it instead of actual value.

Syntax: <constant name> CONSTANT datatype (size): = Value;

- Constant name is the name of the constant i.e. similar to a variable name.
- The word CONSTANT is a reserved word and ensures that the value does not change.
- Value it is a value which must be assigned to a constant when it is declared. we cannot assign a value later.

For example: To declare salary_increase, we can write code as follows:

```
SQL>  DECLARE
          salary_increase CONSTANT number (3):= 100;
```

****** (Q) Explain the Control Statements in PL/SQL?**

PL/SQL supports programming language features like conditional statements, iterative statements. PL/SQL control structure allows we to control the behaviour of the block as it runs. we can change the logical flow of statements within the PL/SQL block with a number of control structures.

(I). **Conditional Control Structures:** A conditional control structure tests a condition to find out whether it is true or false and accordingly executes the different blocks of SQL statements. Conditional control is generally performed by IF statement. There are three forms of IF statements:

(a). IF…THEN - END IF.
(b). IF…THEN…ELSE – END IF.
(c). IF…THEN…ELSIF – END IF.

(a). **IF…THEN – END IF:** It is used test only one condition. It returns when if the condition is True. Otherwise the condition will exit. Every IF statement should be ended with keyword "End if".

Syntax: If <condition> then
Statement 1;
Statement 2;
…….
End If;

Example: SQL> Declare
a number;
b number;
Begin
a:= 20;
b:= 10;
if (a > b) then
dbms_output.put_line („A is Bigger than B");
End If;
End;
PL/SQL procedure successfully completed.

(b). **IF…THEN...ELSE – END IF:** It is used to test a condition if condition is true then return True statement, otherwise its return"s false statements.

Syntax: If <condition> then
Statements;
Else
Statements;
…….
End If;

Example: SQL> Declare
a number;
b number;
Begin
a:= 10;
b:= 20;
if (a > b) then
dbms_output.put_line („A is Bigger than B");
else
dbms_output.put_line („B is Bigger than A");
End If;

```
End;
```

PL/SQL procedure successfully completed.

(c). **IF…THEN…ELSIF – END IF:** This statement selects an action from several mutually exclusive alternatives.

Syntax:
```
if <condition 1> then
    Statement 1;
Elsif <condition 2> then
    Statement 2;
Elsif <condition 3> then
    Statement 3;
    ……..
Else
      Statement n;
End if;
```

Example: SQL>
```
declare
a number(2):=10;
b number(2):=20;
c number(2):=30;
begin
if ((a>b) and (a>c)) then
  dbms_output.put_line ('The Biggest Number is a');
else if (b<c) and (a>c) then
  dbms_output.put_line ('The Biggest Number is b');
else
  dbms_output.put_line ('The Biggest Number is c'); end if;
      end if;
      end;
       /
```

(II). **Loop Control Structures:** Iterative control statements or Loop control statements are used when we want to repeat the execution of one or more statements for specified number of times. PL/SQL provides the following types of loops:

(a). Simple / Basic Loop.
(b). While Loop.
(c). For Loop.

(a). **Simple / Basic Loop:** The simplest form of loop statement is the basic (or infinite) loop. Which enclose a sequence of statements between the keywords „LOOP“ and „END LOOP“. Each time the flow of execution reaches the „END LOOP“ statement, control is returned to the corresponding „LOOP“ statement above it. A basic loop allows execution of its statement at least once, even if the condition is already met upon entering the loop. Without the „EXIT“ statement, the loop would be infinite or endless.

EXIT Statement: The EXIT statement to terminate a loop. Control passes to the next statement after the „END LOOP“ statement. The EXIT is used either as an action within an IF statement or as a stand- alone statement within the loop. The EXIT statement must be

placed inside of a loop. we can also attach a „WHEN“ clause to allow conditional termination of the loop. When the EXIT statement is encountered, the condition in the WHEN clause is evaluated.

Syntax: LOOP
Statements;
- - - - - - - - - - -
- - - - - - - - - - -
EXIT
[WHEN
condition]; END
LOOP;

Example:

```
DECLARE
        a NUMBER := 1;
BEGIN
  LOOP
DBMS_OUTPUT.PUT_LINE('A  value is : ' || a);
   a := a + 1;
   EXIT WHEN a > 3;
  END LOOP;
END;
/
```

(b). **While Loop:** The While loop to repeat a sequence of statements until the controlling condition is

„True“. The condition is evaluated at the start of each iteration. The loop terminates when the condition is

„False“. If the condition if False at the start of the loop, then no further iterations are performed.

Syntax: While <Condition> Loop
Statements;
- - - - - - - - - - -
- - - - - - - - - - -
End Loop;

Example:

```
DECLARE
  a NUMBER := 1;
BEGIN
  WHILE a <= 3 LOOP
    DBMS_OUTPUT.PUT_LINE('Value of a: ' || a);
    a := a + 1;
  END LOOP;
END;
/
```

(c) For Looping: This looping statement will execute loop body repeatedly initial values to final value for every execution. The loop counter will be incremented by one. So by using this statement we can also find the number of iterations. This statement also called as “Recursive Looping” statement.

Syntax: for counter IN [Reverse] <lower bound>..<upper bound>

```
Loop
        Statement 1;
          ..............  }
          ..............  } Loop Body.
          ..............  }
End Loop;
```

- **Counter:** Is an implicitly declared integer whose value automatically increases or decreases by 1 (if the reverse keyword is used) until the upper or lower bound is reached.

- **Reverse:** Reverse causes the counter to decrement with each iteration from the upper bound to the lower bound.

- **Lower bound:** It specifies the lower bound for the range of counter values.

- **Upper bound:** It specifies the upper bound for the range of counter values.

Example:

```
BEGIN
  FOR i IN 1..3 LOOP
    DBMS_OUTPUT.PUT_LINE('Ascending order: ' || i);
  END LOOP;

  FOR j IN REVERSE 1..3 LOOP
    DBMS_OUTPUT.PUT_LINE('Descending order: ' || j);
  END LOOP;
END;
/
```

********* (Q) Discusses about the Cursors in PL/SQL?**

Cursors are a private working area or cursors are a temporary working area. "It is mainly used for retrieving multiple rows based on multiple attributes at a time. Cursors are very useful to update big structure of a table at a time". Cursors can calculate calculation according to given conditions and fetching rows into base tables.

A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it. This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the active set.

Cursors are mainly divided into two types in SQL. There are:

I. Implicit Cursor. (System defined)
II. Explicit Cursor. (User defined)

I. **Implicit Cursor:** The Implicit Cursor is declared by PL/SQL implicitly when DML statements like INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed. These are automatically executed by SQL Engine Internally.

Ex: Create Table, Alter Table, Create View, Update etc..,

Implicit Cursor Attributes: These attributes are associated with the implicit SQL cursor and can be accessed by appending the attribute name to the implicit cursor name (SQL).

Syntax: SQL%<attribute name>;

In PL/SQL implicit cursor has 4 attributes there are:

1. SQL%ISOPEN
2. SQL%FOUND
3. SQL%NOTFOUND
4. SQL%ROWCOUNT

1. **SQL%ISOPEN:** It is used to determine if a cursor is already open. It is always „False“ because Oracle automatically closes implicit cursor after executing its associated SQL statements.

2. **SQL%FOUND:** It is always „True“ if the most recently executed DML statement as successful.

3. **SQL%NOTFOUND:** It is „True“ of the most recently executed DML statement was not successful.

4. **SQL%ROWCOUNT:** This attribute is used to determine the number of rows that are processed by SQL statements.

Example: Consider a PL/SQL code to display message to check whether the record is deleted or not.

SQL> edit stud_infoplsql.sql;

```
declare
   begin
      delete from stud_info where stud_id = &stud_id;
      if SQL%FOUND then
            dbms_output.put_line ('Record Deleted');
      else
            dbms_output.put_line ('Record Not Deleted');
      end if;
      end;
/
```

II. **Explicit Cursor:** The Explicit cursor is declared explicitly by the user, along with other identifiers to be used in a PL/SQL block. These are also known as „User defined“ cursors, defined in the „Declare“ section of the PL/SQL block. They must be declared when

we are executing a „Select“ statement that returns more than one row. Even through the cursor stores multiple records, only one record can be processed at a time, which is called as „Current row“ or „active set“. A PL/SQL program opens a cursor to process rows returned by a query and then closes the cursor.

Steps to create a cursor:

1. Declare the cursor
2. Open the cursor
3. Fetch data from the cursor
4. Close the cursor

1. **Declare the cursor:** Before using any cursor that should be declared first. In one PL/SQL Program we may use many cursors; all cursors must be defined under declarative part of the PL/SQL block. A cursor is defined in the declarative part by naming it and specifying a „SELECT“ statement.

Syntax: CURSOR <cursor_name> IS <SELECT – statements>;

- **cursor_name:** A suitable name for the cursor.
- **select – statement:** A select query which returns multiple rows.

Example: declare
cursor student123_cur is select * from emp where sal > 2000;

In the above example, we are declaring a cursor „student123_cur“ on a query which returns the records of all the employees with salary greater than 2000. Here „emp“ is the table which contains records of all the employees.

2. **Open the cursor:** After declaration, the cursor is opened with an „Open“ statement for processing rows in the cursor. The SELECT statement associated with the cursor is executed when the cursor is opened.

Syntax: OPEN <cursor_name>;

Example: OPEN student123_cur;

3. **Fetch / Retrieve data from the cursor:** After the cursor is opened, the current row is loading into variables. The current row is the row at which the cursor is currently pointing. The transfer of data into PL/SQL variables is done through „FETCH“ statement.

Syntax: FETCH <cursor_name> INTO <variables>; (or) FETCH <cursor_name> INTO <record_named>

SQL> fetch rec into student123_cur;

4. **Close the cursor:** After working successfully with a cursor, it must be closed when we close a cursor. It will release all recourse.

Syntax: CLOSE <cursor_name>;

Example: CLOSE student123_cur;

```
Eg: DECLARE
   CURSOR c_emp IS SELECT id FROM emp;
      BEGIN
   FOR emp_rec IN c_emp
      LOOP
            DBMS_OUTPUT.PUT_LINE(emp_rec.id);
      END LOOP;
      END;
      /
```

PL/SQL Triggers

PL/SQL triggers are block structures and predefined programs invoked automatically when some event occurs. They are stored in the database and invoked repeatedly in a particular scenario. There are two states of the triggers, they are enabled and disabled. When the trigger is created it is enabled. CREATE TRIGGER statement creates a trigger. A triggering event is specified on a table, a view, a schema, or a database. BEFORE and AFTER are the trigger Timing points.DML triggers are created on a table or view, and triggers. Crossedition triggers are created on Edition-based redefinition. System Triggers are created on schema or database using DDL or database operation statements.

Syntax:

```
CREATE OR REPLACE TRIGGER trigger_name
BEFORE or AFTER or INSTEAD OF                    //trigger timings
INSERT  or UPDATE or  DELETE                     // Operation to be performed
of column_name on Table_name FOR EACH ROW
DECLARE
      Declaration section
BEGIN
      Execution section
EXCEPTION
      Exception section
END;
```

- CREATE [OR REPLACE] TRIGGER trigger_name is used to create a trigger or replace the existing

trigger.

- BEFORE | AFTER | INSTEAD OF specifies trigger timing.
- INSERT | UPDATE | DELETE are the DML operations performed on table or views.
- OF column_name specifies the column that would be updated.
- ON table_name species the table for the operation.
- FOR EACH ROW specify that trigger is executed on each row .

Eg: CREATE OR REPLACE TRIGGER before_insert_employee

```
BEFORE INSERT ON employees
FOR EACH ROW
BEGIN
        :NEW.creation_date := SYSDATE;
END;
```

PL/SQL Procedures

Procedures are stored subprograms primarily used to perform actions or execute a sequence of statements. They can have IN, OUT, and IN OUT parameters to receive data, return results, or both.

Syntax:

```
CREATE [OR REPLACE] PROCEDURE procedure_name (
   parameter_name [IN | OUT | IN OUT] type [, ...]
)
{IS | AS}
BEGIN
   -- procedure body (executable statements)
EXCEPTION
   -- exception handling part (optional)
END [procedure_name];
```

Eg:

```
CREATE OR REPLACE PROCEDURE greetings
        AS
                BEGIN
                  dbms_output.put_line('Hello World!');
                END;
```

PL/SQL Functions

Functions are similar to procedures but are designed to return a single value using a RETURN clause.

Syntax:

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
  < function_body >
END [function_name];
```

Eg:

```
CREATE OR REPLACE FUNCTION totalemp
RETURN number IS
  total number(2) := 0;
BEGIN
  SELECT count(*) into total FROM emp;
   RETURN total;
END;
```

Calling a Function

function to perform the defined task. When a program calls a function, the program control is transferred to the called function.

Eg:

```
DECLARE
  c number(2);
BEGIN
  c := totalemp();
  dbms_output.put_line('Total no. of Employee: ' || c);
END;
```

PL/SQL Package

A PL/SQL package is a collection of related Procedures, Functions, Variables, and other elements that are grouped for Modularity and Reusability.

Eg:

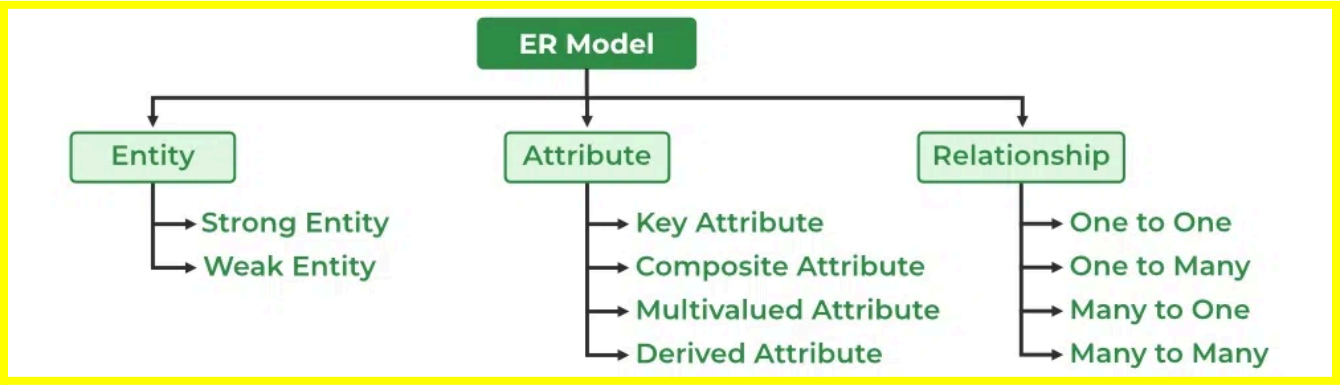
```
CREATE OR REPLACE PACKAGE math_operations AS
   FUNCTION add_numbers(x NUMBER, y NUMBER) RETURN NUMBER;
   FUNCTION multiply_numbers(x NUMBER, y NUMBER) RETURN NUMBER;
END math_operations;
DECLARE
   sum_result NUMBER;
```

```
    product_result NUMBER;
    num1 NUMBER := 10;
    num2 NUMBER := 5;
BEGIN
    -- Call the functions from the package
    sum_result := math_operations.add_numbers(num1, num2);
    product_result := math_operations.multiply_numbers(num1, num2);

    -- Display the results using the built-in DBMS_OUTPUT package
    DBMS_OUTPUT.PUT_LINE('The sum of ' || num1 || ' and ' || num2 || ' is: ' || sum_result);
    DBMS_OUTPUT.PUT_LINE('The product of ' || num1 || ' and ' || num2 || ' is: ' || product_result);
END;
```

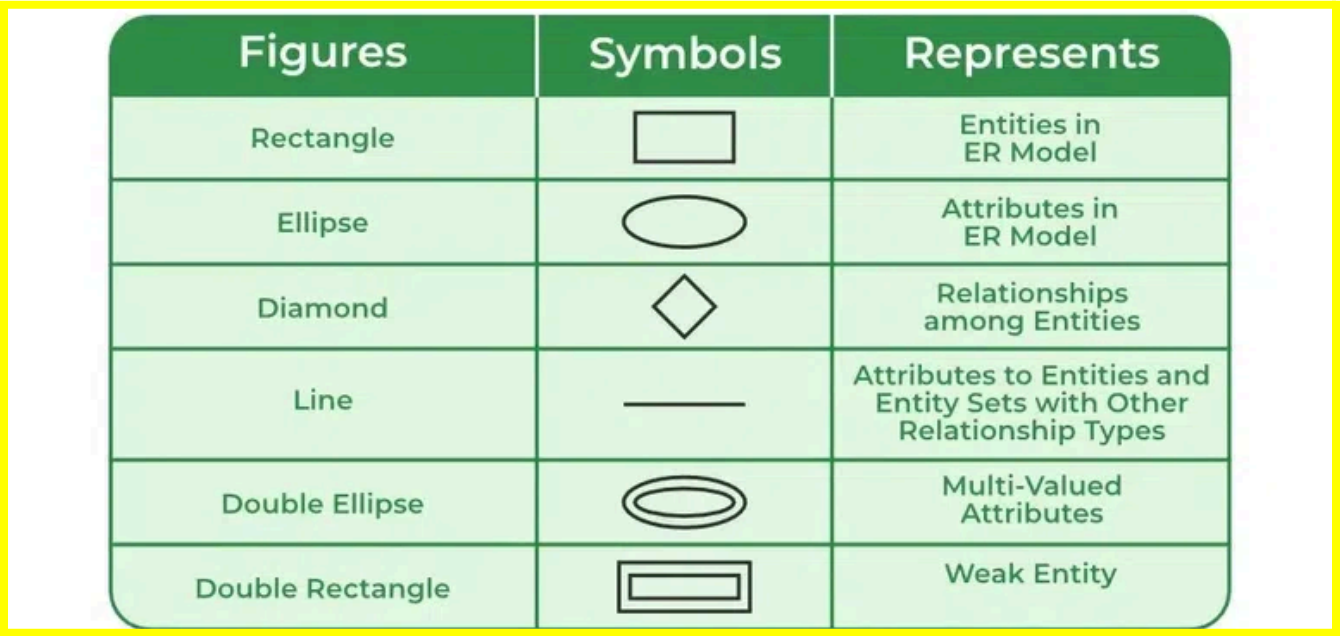
UNIT II

ER Model: An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.



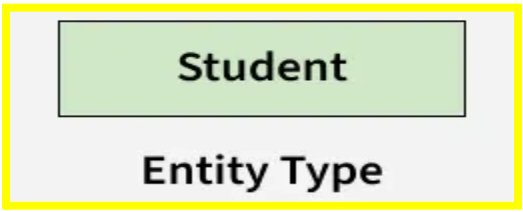
Symbols Used in ER Model

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



Entity

An Entity represents a real-world object, concept or thing about which data is stored in a database. It acts as a building block of a database. Tables in relational databases represent these entities.

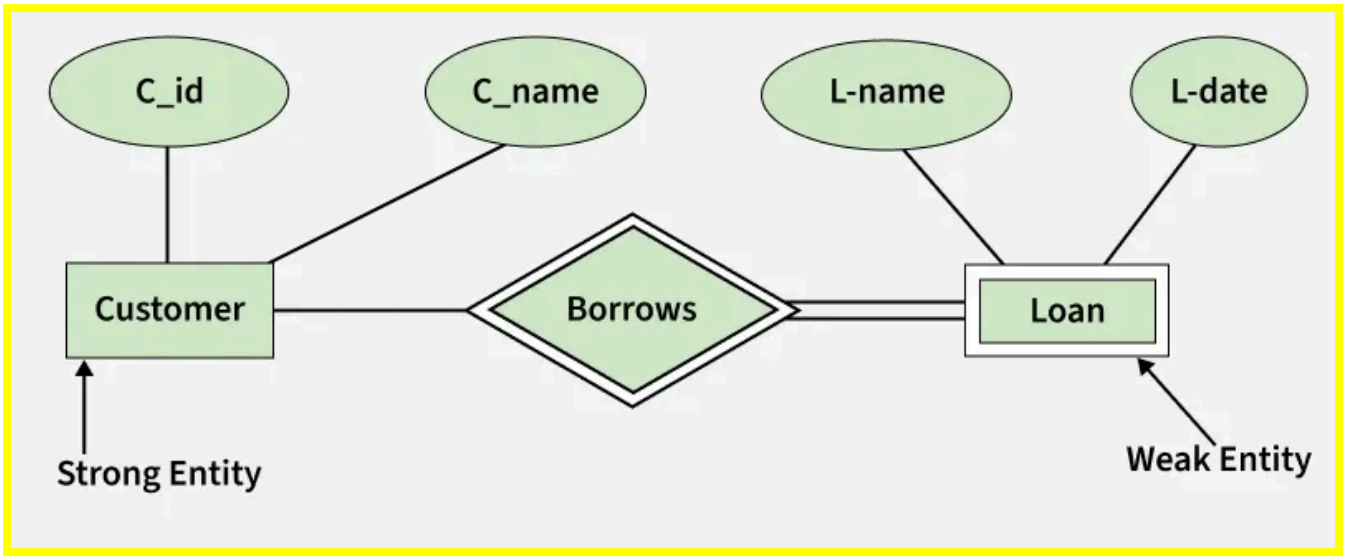


Strong Entity

A strong entity is not dependent on any other entity in the schema. A strong entity will always have a primary key. Strong entities are represented by a single rectangle.

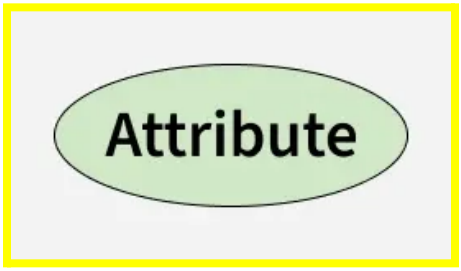
Weak Entity

A weak entity is dependent on a strong entity to ensure its existence. Unlike a strong entity, a weak entity does not have any primary key. A weak entity is represented by a double rectangle.



Attributes in ER Model

Attributes are the properties that define the entity type. For example, for a Student entity Roll_No, Name, DOB, Age, Address, and Mobile_No are the attributes that define entity type Student. In ER diagram, the attribute is represented by an oval.

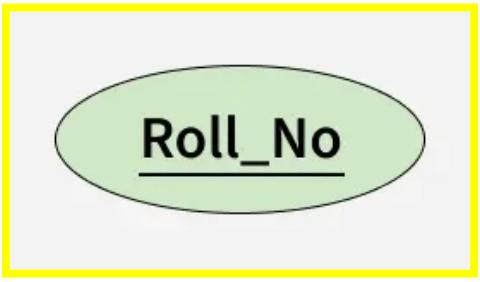


Attribute

Types of Attributes

1. Key Attribute

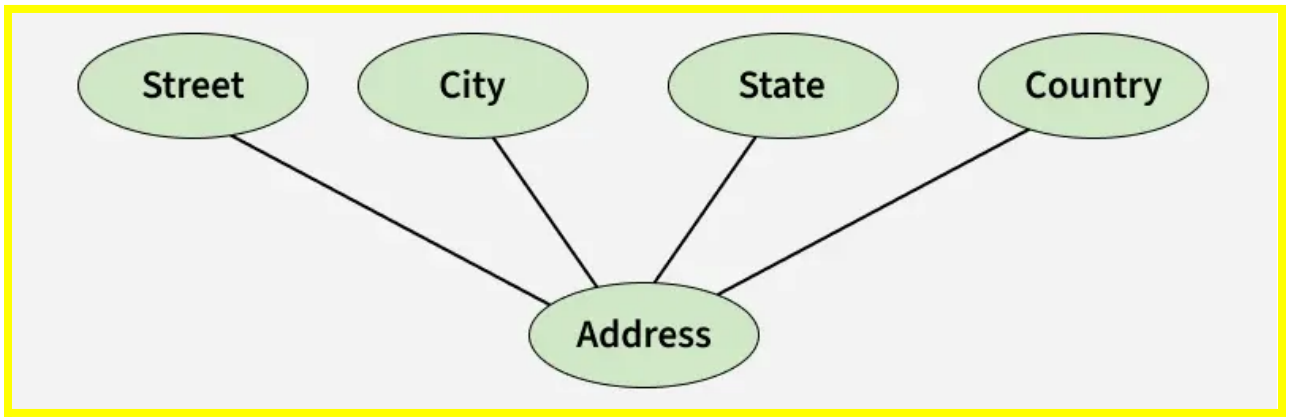
The attribute which uniquely identifies each entity in the entity set is called the key attribute. For example, Roll_No will be unique for each student. In ER diagram, the key attribute is represented by an oval with an underline.



Key Attribute

2. Composite Attribute

An attribute composed of many other attributes is called a composite attribute. For example, the Address attribute of the student Entity type consists of Street, City, State, and Country. In the ER diagram, the composite attribute is represented by an oval consisting of ovals.



Composite Attribute

3. Multivalued Attribute

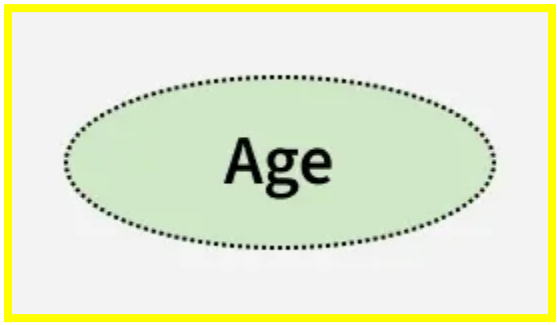
An attribute consisting of more than one value for a given entity. For example, Phone_No. In ER diagram, a multivalued attribute is represented by a double oval.



Multivalued Attribute

4. Derived Attribute

An attribute that can be derived from other attributes of the entity type is known as a derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, the derived attribute is represented by a dashed oval.

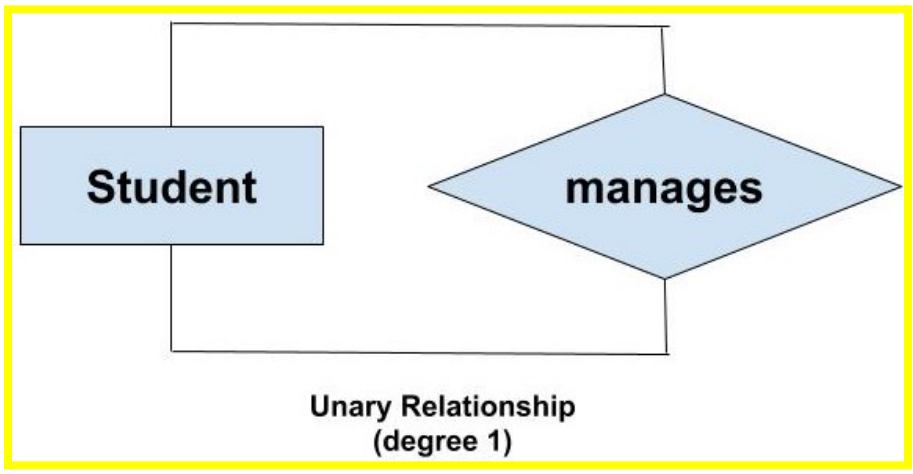


Derived Attribute

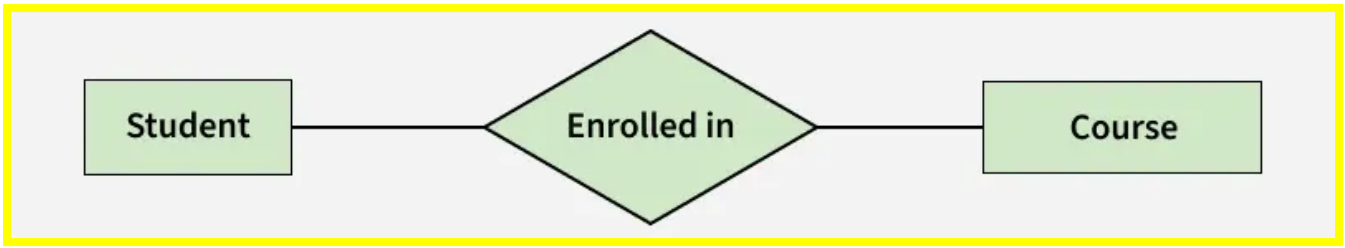
3. Relationship

Cardinality: Defines the numerical attributes of the relationship between two entities or entity Sets.

1. Unary Relationship - An relationship between only one entity is called Unary relationship.



2. Binary Relationship: An relationship between only two entities is called Binary relationship. For example, a Student is enrolled in a Course.



Types of Relationships - A relationship is represented by diamond shape in ER diagram, it shows the relationship among

entities. There are four types of cardinal relationships:

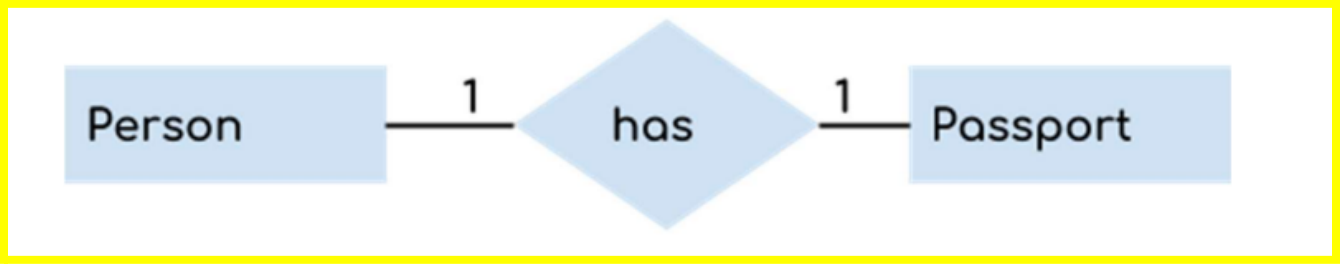
1. One to One

2. One to Many

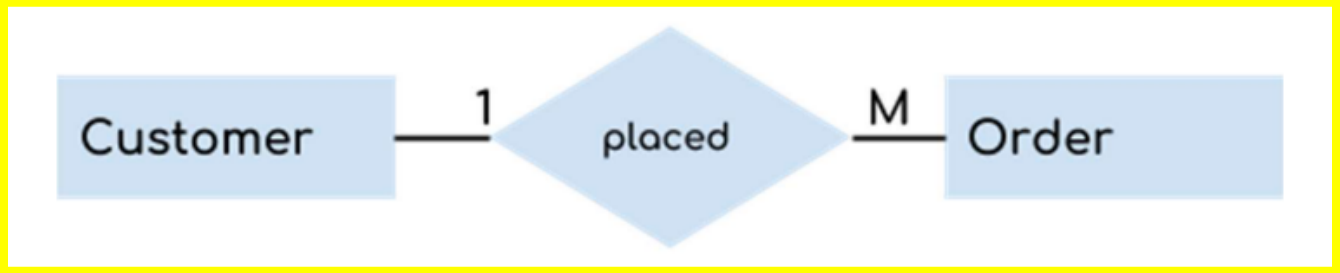
3. Many to One

4. Many to Many

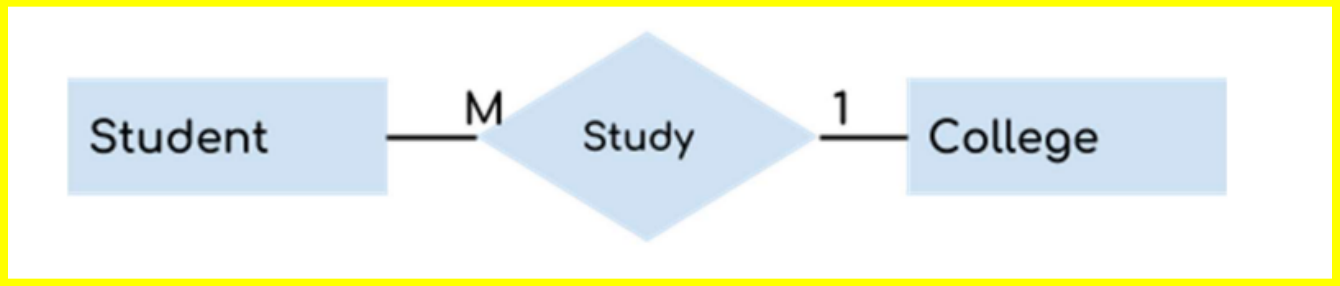
1. One to One Relationship - An entity is associated with a single instance of another entity then it is called one to one relationship. For example, a person has only one passport and a passport is given to one person.



2. One to Many Relationship - An entity is associated with more than one instances of another entity then it is called one to many relationship. For example – a customer can place many orders but a order cannot be placed by many customers.



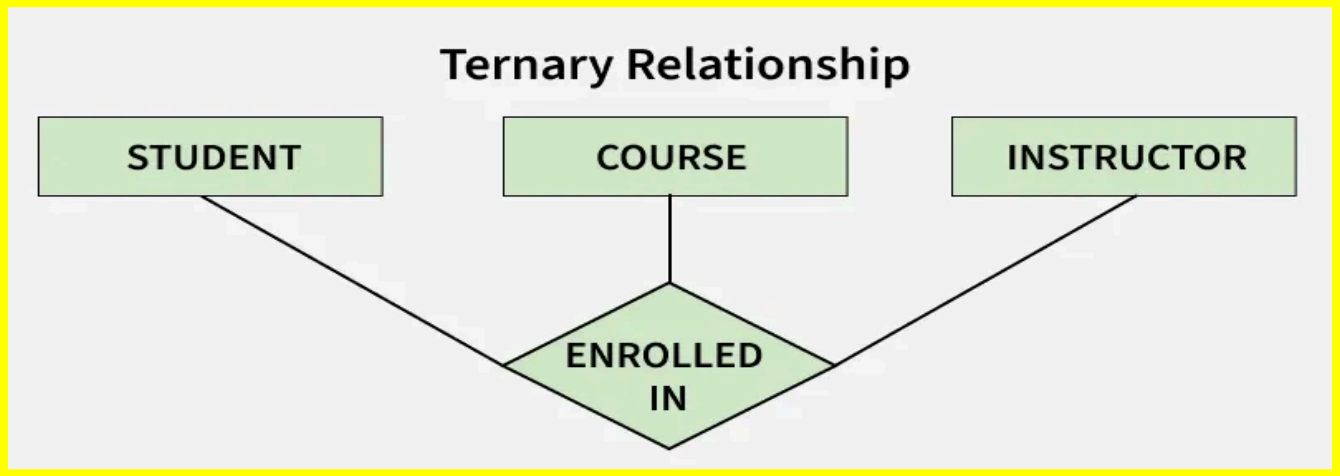
3. Many to One Relationship - An entity is associated with a single instance of another entity then it is called many to one relationship. For example – many students can study in a single college but a student cannot study in many colleges at the same time.



4. Many to Many Relationship - An entity is associated with more than one instances of another entity then it is called many to many relationship. For example, a can be assigned to many projects and a project can be assigned to many students.



3. Ternary Relationship: An relationship between only two or more entities is called ternary relationship.



Enhanced ER Model

The Enhanced ER Model (EER model) is an advanced database design model that extends the traditional Entity-Relationship (ER) model. It adds semantic concepts like inheritance, specialization, and categorization to handle complex data requirements.

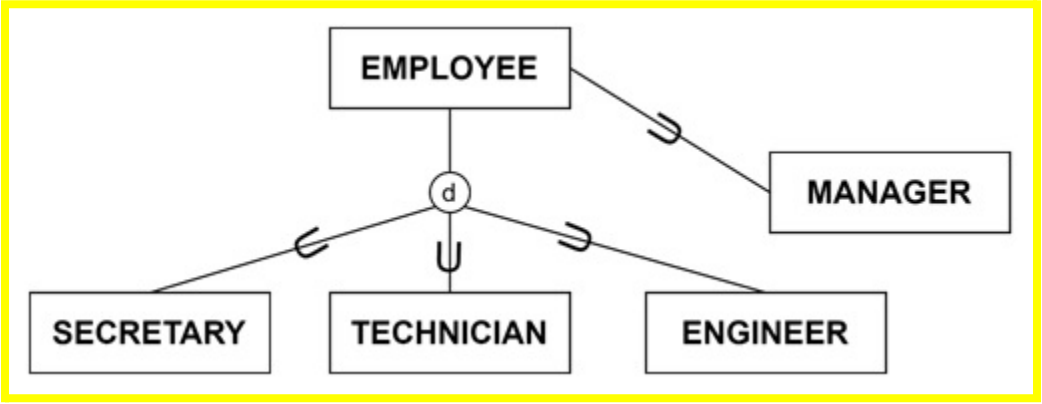
Core Concepts

- **Superclass and Subclass**: A superclass is a higher-level entity set, and a subclass is a lower-level subset that inherits attributes and relationships from the superclass.
- **Specialization**: A top-down process where a higher-level entity is divided into lower-level subclasses based on unique characteristics.
- **Generalization**: A bottom-up process where multiple lower-level entities with common attributes are combined into a single higher-level superclass.

Subclasses and Superclasses

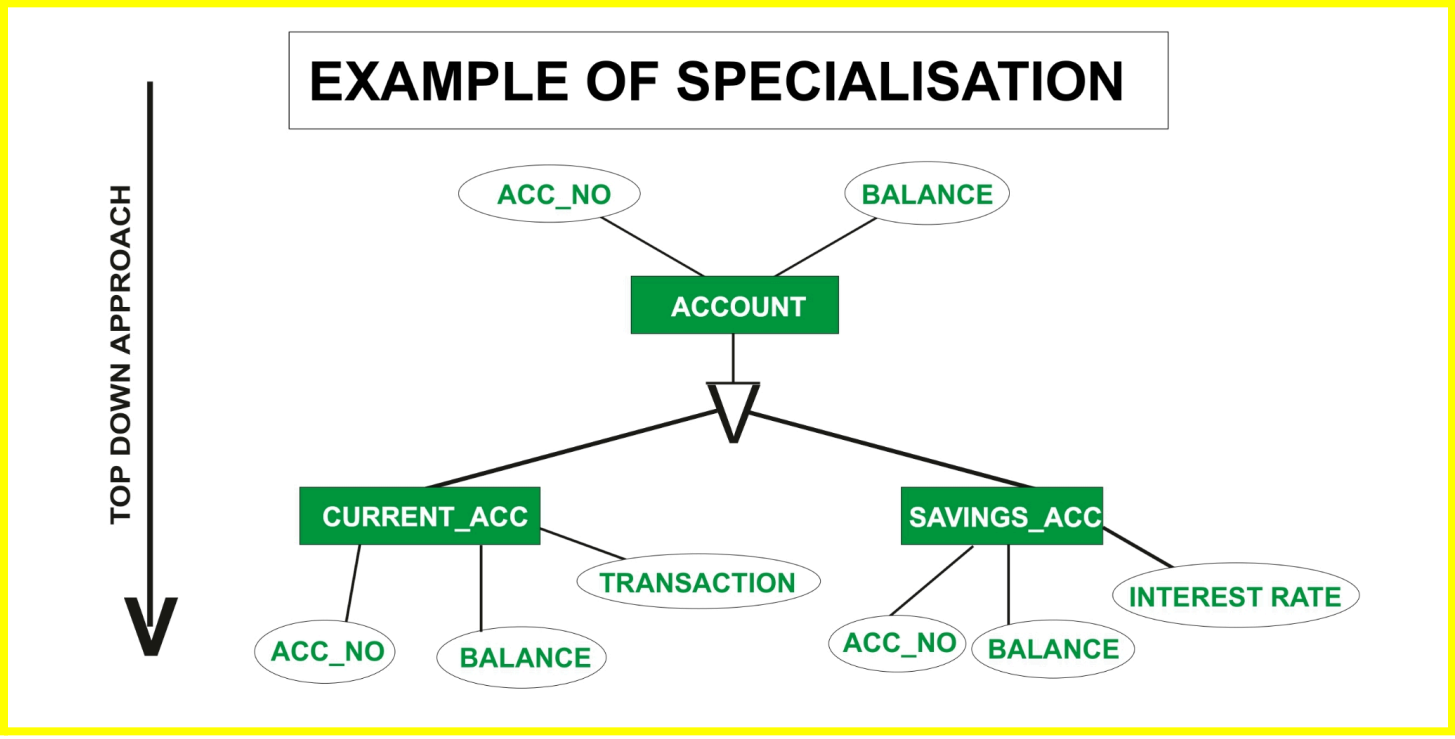
The EER Model uses concepts such as subclasses and superclasses which are normally used in OOPs programming. A superclass represents a general entity type, while a subclass is a more specific grouping of that entity.

For instance, consider the EMPLOYEE entity type. Within a company, the EMPLOYEE category could include distinct groups such as SECRETARY, ENGINEER, TECHNICIAN, and MANAGER. Now each of these subgroups forms a subclass of EMPLOYEE.

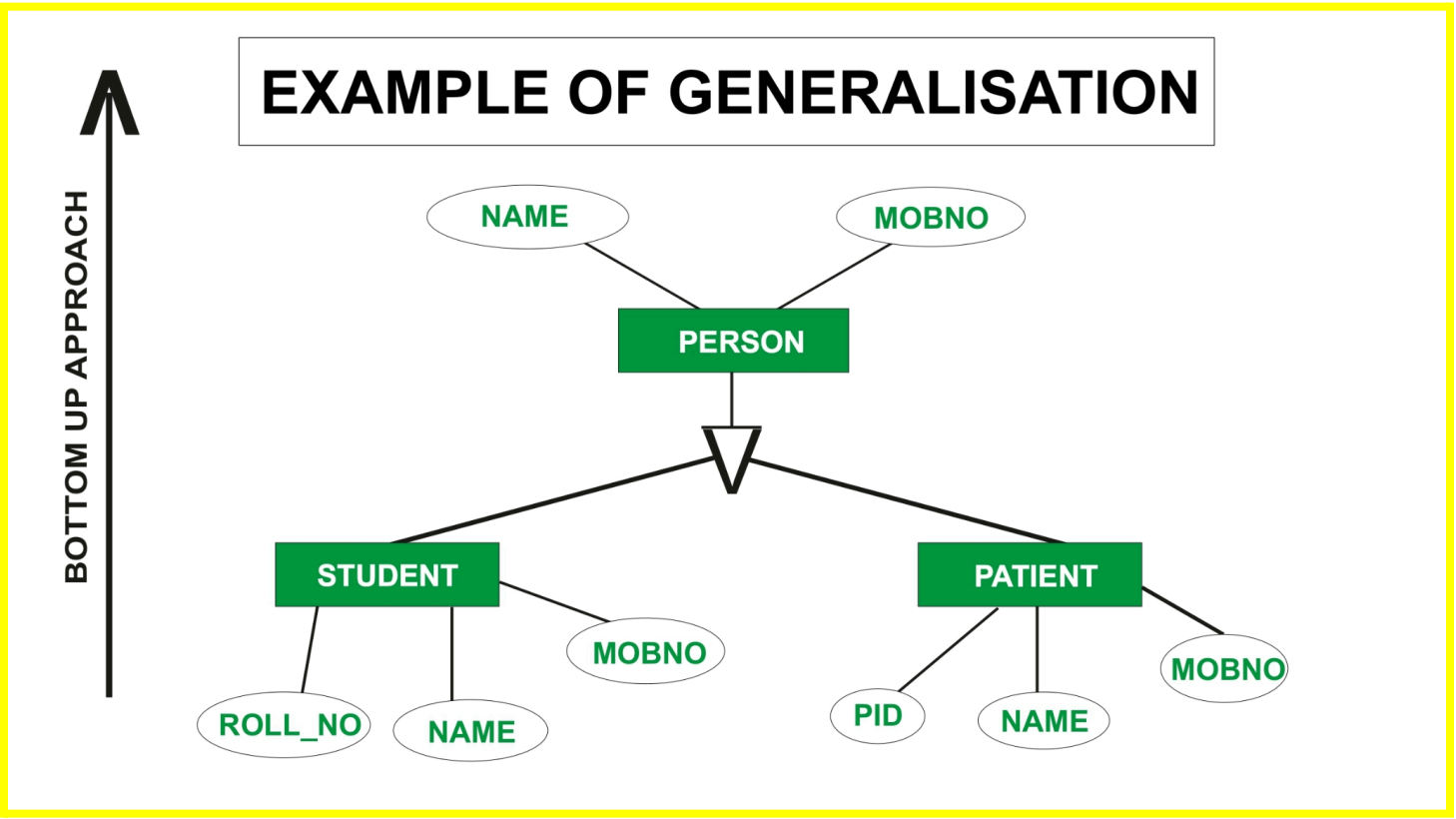


Here, the relationship between a superclass and its subclasses is often described as an "IS-A" relationship.

Specialization: A top-down process where a higher-level entity is divided into lower-level subclasses based on unique characteristics.(or) The process of defining a set of subclass of an entity type is called specialization.



Generalization: A bottom-up process where multiple lower-level entities with common attributes are combined into a single higher-level superclass.(or) Generalization is the reverse of specialization process.



.Constraints on Specialization and Generalization

- **Disjointness Constraint**: Decides whether a superclass entity can belong to more than one subclass.
    - **Disjoint (d)**: An entity can belong to **only one** subclass (e.g., an employee can be a *Full-Time* **OR** a *Part-Time* employee, never both).
    - **Overlapping (o)**: An entity can belong to **more than one** subclass simultaneously (e.g., a university person can be both a *Staff* member **AND** a *Student*).
- **Completeness Constraint** (Participation Constraint): Decides whether every entity in the superclass must participate in at least one subclass.
    - **Total Completeness / Specialization**: Every superclass entity **must** belong to at least one subclass, represented by a double line on an ER diagram (e.g., a bank *Account* must be either a *Savings* account **OR** a *Current* account).
    - **Partial Completeness / Specialization**: A superclass entity **may or may not** belong to a subclass, represented by a single line (e.g., a *Person* in a university database may be a *Student* or *Faculty*, or neither).
- **Membership Constraint** (Predicates): Determines how an entity is placed into a specific subclass.
    - **Attribute-Defined**: Membership is automatically decided based on a specific attribute value (e.g., if Age < 18, place in the *Minor* subclass).

    - **User-Defined**: Membership is manually decided and assigned by the database user or application logic rather than a rule (e.g., assigning an employee to a *Project Manager* role).

Advantages of ER Modeling

- **Conceptual Simplicity**: Provides a high-level, easy-to-understand graphical blueprint of data and business rules before physical database creation.
- **Visual Representation**: Uses straightforward shapes (rectangles, diamonds, ovals) that make it simple for technical and non-technical stakeholders to inspect and validate data structures.
- **Effective Communication Tool**: Acts as a clear visual bridge between database designers, developers, and business analysts or clients.
- **Easy Integration with Relational Models**: Can be directly and systematically translated into relational database tables, primary keys, and foreign keys.
- **Minimizes Data Redundancy**: Helps identify duplicate data and functional dependencies early, leading to cleaner data organization and better normalization.
- **Facilitates Troubleshooting & Maintenance**: Serves as permanent visual documentation that makes it easier to track errors, modify schemas, or scale the system over time.

UNIT - III

Dr. E.F. Codd's 12 Rules (Numbered 0 to 12)

Dr. Codd formulated 13 rules (indexed 0 to 12) to define what makes a true Relational Database Management System (RDBMS):

1. **Rule 0 (Foundation Rule)**: The system must be able to manage database tasks completely through its relational capabilities.
2. **Rule 1 (Information Rule)**: All data must be represented explicitly as values in tables (rows and columns).
3. **Rule 2 (Guaranteed Access Rule)**: Every single data item (atomic value) is guaranteed to be logically accessible via a combination of a table name, primary key value, and attribute name.
4. **Rule 3 (Systematic Treatment of Null Values)**: Null values must be supported uniformly to represent missing or inapplicable data, distinct from zero, blank spaces, or strings.
5. **Rule 4 (Active Online Catalog / Metadata)**: The database description (schema) must be stored in the online catalog, accessible to authorized users using the same query language as regular data.
6. **Rule 5 (Comprehensive Data Sublanguage)**: The system must support at least one language with linear syntax that supports data definition, view definition, data manipulation, integrity constraints, and transaction management.
7. **Rule 6 (View Updating Rule)**: All views that are theoretically updatable must be updatable by the system.
8. **Rule 7 (High-Level Insert, Update, and Delete)**: The capability of handling a base or derived relation as a single operand applies not only to data retrieval (queries) but also to insert, update, and delete operations.
9. **Rule 8 (Physical Data Independence)**: Application programs and terminal activities remain logically unchanged when physical storage methods or access paths are modified.
10. **Rule 9 (Logical Data Independence)**: Application programs remain logically unchanged when information-preserving changes are made to the base tables (such as splitting tables).
11. **Rule 10 (Integrity Independence)**: Integrity constraints specific to a relational database must be definable in the relational data sublanguage and stored in the catalog, not in application programs.
12. **Rule 11 (Distribution Independence)**: The RDBMS must have distribution independence; data distribution across various locations should be transparent to users and applications.
13. **Rule 12 (Non-Subversion Rule)**: If the system supports a low-level (row-at-a-time) language, that low-level language cannot be used to bypass or subvert the integrity rules or constraints of the

relational language.

Keys in Relational Model

Keys are fundamental elements of the relational database model that ensure uniqueness, data integrity, and efficient data access.

- They uniquely identify each row in a table.
- They prevent data duplication and maintain consistency.
- They create relationships between different tables.

Types of Database Keys

1. Candidate Key - The minimal set of attributes that can uniquely identify a tuple is known as a candidate key. For Example,

Example: For the STUD_NO in STUDENT relation. STUDENT table below, STUD_NO can be a candidate key, as it uniquely identifies each record.

STUD_NO	SNAME	ADDRESS	PHONE
1	Tom	New York	123456789
2	John	Los Angeles	223365796
3	Alice	Chicago	175468965

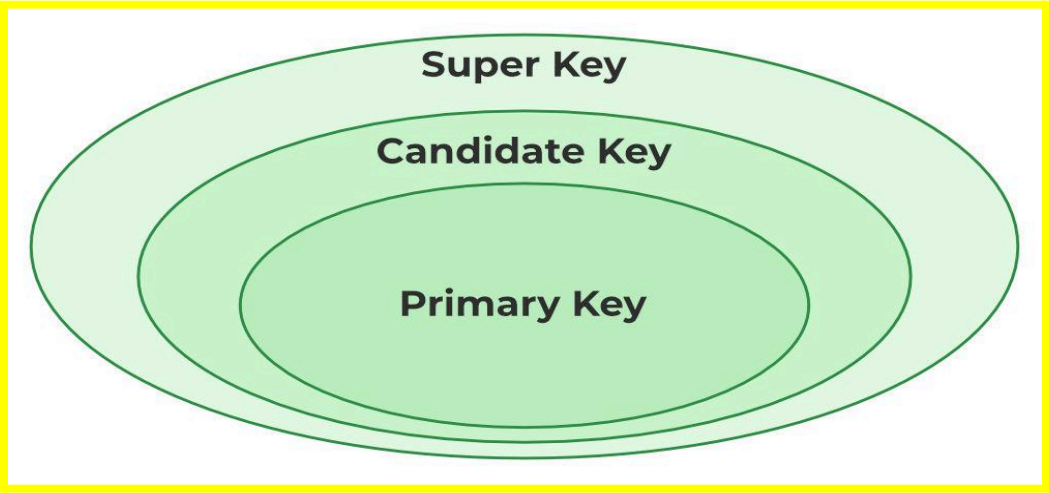
Table: STUDENT_COURSE

STUD_NO	TEACHER_NO	COURSE_NO
1	001	C001
2	056	C005

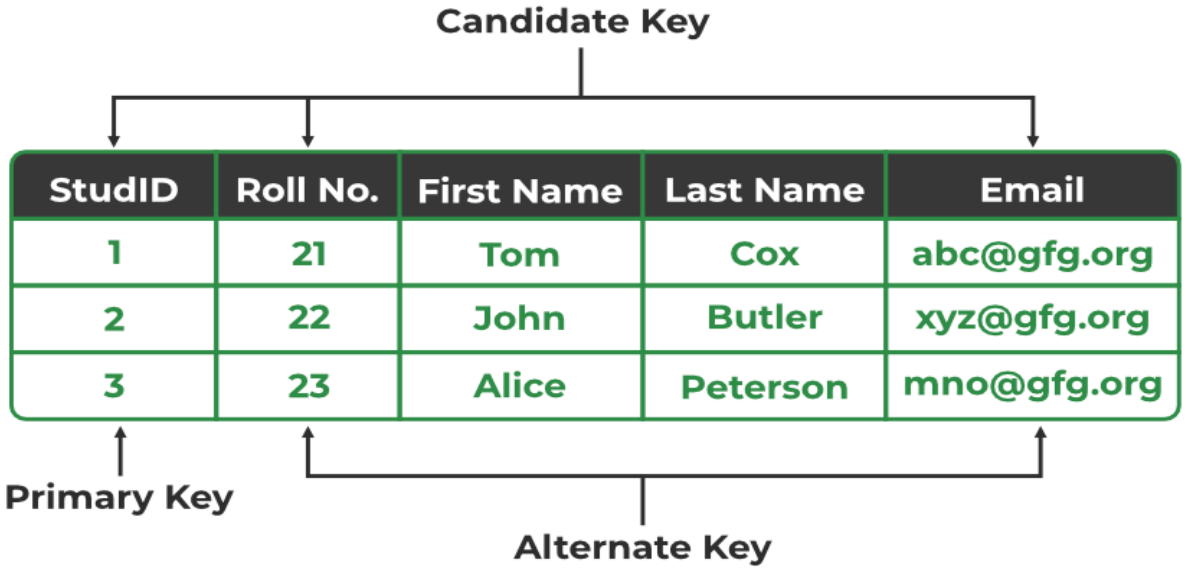
2. Super Key - The set of one or more attributes (columns) that can uniquely identify a tuple (record) is known as Super Key. It may include extra attributes that aren't important for uniqueness but still uniquely identify the row. For Example, STUD_NO, (STUD_NO, STUD_NAME), etc.
Example: Consider the STUDENT table

STUD_NO	SNAME	ADDRESS	PHONE
1	Tom	New York	123456789
2	John	Los Angeles	223365796
3	Alice	Chicago	175468965

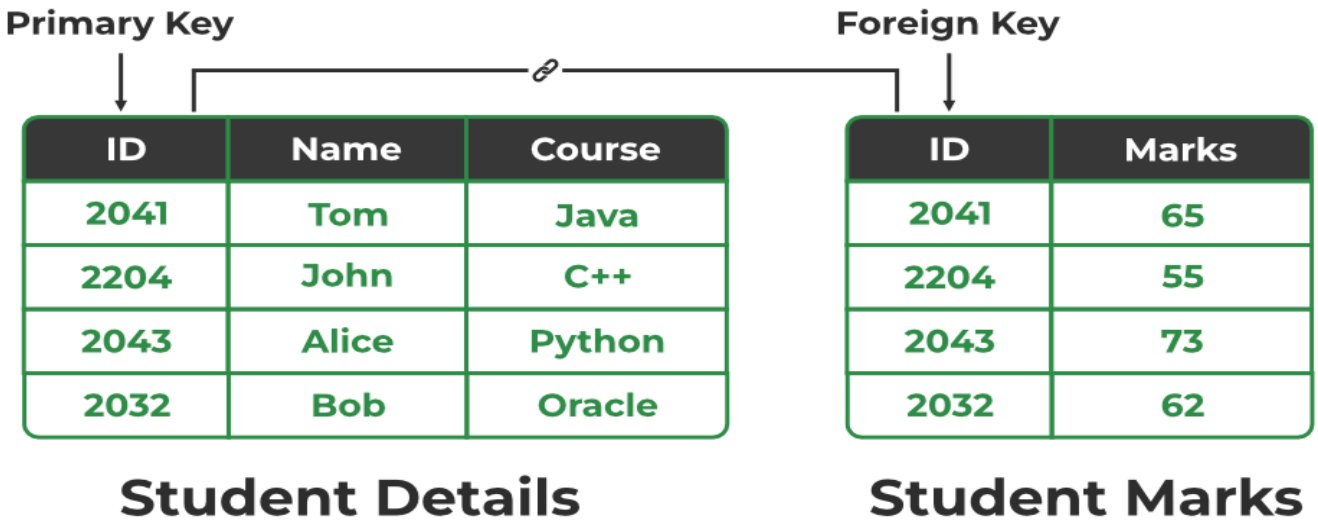
A super key could be a combination of STUD_NO and PHONE, as this combination uniquely identifies a student.



3. Alternate Key - An alternate key is any candidate key in a table that is not chosen as the primary key.
Example: In the STUDENT table, both STUD_NO and PHONE are candidate keys. If STUD_NO is chosen as the primary key, then PHONE would be considered an alternate key.



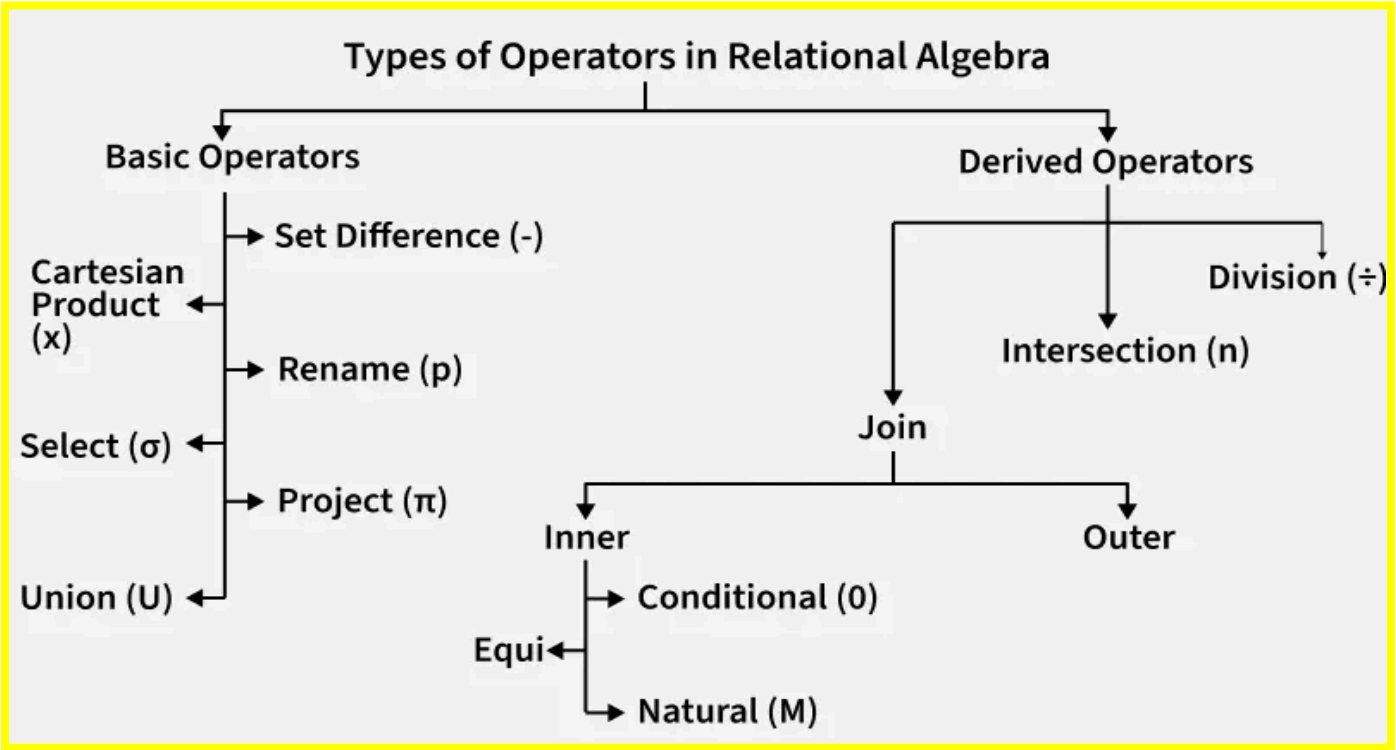
4. Foreign Key - A foreign key is an attribute in one table that refers to the primary key in another table. The table that contains the foreign key is called the referencing table and the table that is referenced is called the referenced table.



6. Primary Key - A primary key is chosen from the set of candidate keys to uniquely identify each record in a table. For example, in the STUDENT table, both STUD_NO and STUD_PHONE can be candidate keys, but STUD_NO is selected as the primary key.

- It cannot be NULL, as each record must have a valid identifier.
- It may be single-column or composite (made of multiple columns).
- Databases often organize data using the primary key to allow faster access and searching.

Example: The STUDENT table has the structure Student(STUD_NO, SNAME, ADDRESS, PHONE), where STUD_NO is the primary key.



STUD_NO	SNAME	ADDRESS	PHONE
1	Tom	New York	123456789
2	John	Los Angeles	223365796
3	Alice	Chicago	175468965

8. Unique Key - A Unique Key is a database constraint that ensures that all values in a specific column or a combination of columns are unique across all the rows in a table. It guarantees that no two rows in the table can have the same value in the columns defined as part of the unique key.
Example: In the STUDENT_COURSE table, the combination of STUD_EMAIL and STUD_NAME can form a Unique Key to ensure that each student's email and name pair is unique across the table.

STUD_NO	STUD_EMAIL	STUD_NAME
101	john@example.com	John
102	emily@example.com	Emily
103	michael@example.com	Michael
104	sara@example.com	Sara
105	david@example.com	David

Relational Algebra in DBMS

Relational Algebra is a formal language used to query and manipulate relational databases, consisting of a set of operations like selection, projection, union, and join. It provides a mathematical framework for querying databases, ensuring efficient data retrieval and manipulation.

Basic Operators in Relational Algebra

Basic operators are fundamental operations that include selection (σ), projection (π), union (U), set difference (−), Cartesian product (×), and rename (ρ).

1. Selection(σ)

The Selection Operation is basically used to filter out rows from a given table based on certain given condition. It basically allows us to retrieve only those rows that match the condition as per condition passed during SQL Query.

Example: If we have a relation R with attributes A, B, and C, and we want to select tuples where C > 3, we

write:

A	B	C
1	2	4
2	2	3
3	2	3
4	3	4

$\sigma_{(c>3)}$(R) will select the tuples which have c more than 3.

Output:

A	B	C
1	2	4
4	3	4

2. Projection(π)

Selection operation works on rows, similarly projection operation of relational algebra works on columns. It basically allows us to pick specific columns from a given relational table based on the given condition and ignoring all the other remaining columns.

Example: Suppose we want columns B and C from Relation R.

$\pi_{(B,C)}$(R) will show following columns.

Output:

B	C
2	4
2	3

3	4

3. Union(U)

The Union Operator is basically used to combine the results of two queries into a single result. The only condition is that both queries must return same number of columns with same data types. Union operation in relational algebra is the same as union operation in set theory.

Example: Consider the following table of Students having different optional subjects in their course.

FRENCH

Student_Name	Roll_Number
Ram	01
Mohan	02
Vivek	13
Geeta	17

GERMAN

Student_Name	Roll_Number
Vivek	13
Geeta	17
Shyam	21
Rohan	25

If FRENCH and GERMAN relations represent student names in two subjects, we can combine their student names as follows:

$\pi_{(Student\_Name)}(FRENCH) \cup \pi_{(Student\_Name)}(GERMAN)$

Output:

Student_Name
Ram
Mohan
Vivek
Geeta
Shyam
Rohan

4. Set Difference(-)

Set difference basically provides the rows that are present in one table, but not in another tables. Set Difference in relational algebra is the same set difference operation as in set theory.

Example: To find students enrolled only in FRENCH but not in GERMAN, we write:

$\pi_{(Student\_Name)}(FRENCH) - \pi_{(Student\_Name)}(GERMAN)$

Student_Name
Ram
Mohan

5. Rename(ρ)

Rename operator basically allows you to give a temporary name to a specific relational table or to its columns. It is very useful when we want to avoid ambiguity, especially in complex Queries. Rename is a unary operation used for renaming attributes of a relation.

Example: We can rename an attribute B in relation R to D

A	B	C

1	2	4
2	2	3
3	2	3
4	3	4

$\rho_{(D/B)}R$ will rename the attribute 'B' of the relation by "D".

Output Table:

A	D	C
1	2	4
2	2	3
3	2	3
4	3	4

6. Cartesian Product(X)

Combines every row of one table with every row of another table, producing all the possible combination. It's mostly used as a precursor to more complex operation like joins. Let's say A and B, so the cross product between A X B will result in all the attributes of A followed by each attribute of B. Each record of A will pair with every record of B.

Relation A:

Name	Age	Sex
Ram	14	M
Sona	15	F

Kim	20	M

Relation B:

ID	Course
1	DS
2	DBMS

Output: If relation A has 3 rows and relation B has 2 rows, the Cartesian product A × B will result in 6 rows.

Name	Age	Sex	ID	Course
Ram	14	M	1	DS
Ram	14	M	2	DBMS
Sona	15	F	1	DS
Sona	15	F	2	DBMS
Kim	20	M	1	DS
Kim	20	M	2	DBMS

Explanation: If A has 'n' tuples and B has 'm' tuples then A X B will have 'n*m' tuples.

Advantages of Relational Algebra

- Relational algebra has a solid mathematical background.
- The mathematical background of relational algebra is the basis of many interesting developments and theorems.
- If we have two expressions for the same operation, and if the expressions prove to be equivalent, then the query optimizer can automatically substitute the more efficient form.

Disadvantages of Relational Algebra

- Relational algebra cannot perform arithmetic operations.

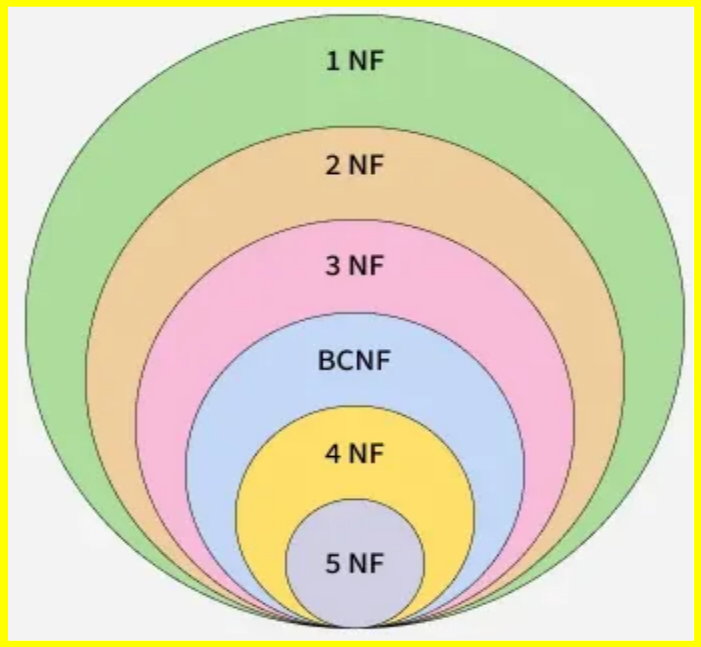
- It is unable to perform aggregation operations, although it cannot compute a transitive closure.
- It cannot change the data present in the database.

Normalization: Normalization is an important process in database design that helps improve the database's efficiency, consistency, and accuracy.

Benefits of using Normal Forms

- Reduce duplicate data and wasted storage.
- Prevent insert, update, and delete anomalies.
- Improve data consistency and integrity.
- Make the schema easier to maintain and evolve.

Normal Forms: Normal forms are a set of rules for relational schemas that reduce redundancy and prevent data anomalies. Each normal form - 1NF, 2NF, 3NF, BCNF, 4NF, 5NF



Normal Forms in DBMS

Normal Forms	Description of Normal Forms
First Normal Form (1NF)	A relation is in first normal form if every attribute in that relation is a single-valued attribute.

Second Normal Form (2NF)	A relation that is in First Normal Form and every non-primary-key attribute is fully functionally dependent on the primary key, then the relation is in Second Normal Form (2NF).
Third Normal Form (3NF)	A relation is in the third normal form, if there is no transitive dependency for non-prime attributes as well as it is in the second normal form. A relation is in 3NF if at least one of the following conditions holds in every non-trivial function dependency X –> Y. ● X is a super key. ● Y is a prime attribute (each element of Y is part of some candidate key).
Boyce-Codd Normal Form (BCNF)	For BCNF the relation should satisfy the below conditions ● The relation should be in the 3rd Normal Form. ● X should be a super-key for every functional dependency (FD) X−>Y in a given relation.
Fourth Normal Form (4NF)	A relation R is in 4NF if and only if the following conditions are satisfied: ● It should be in the Boyce-Codd Normal Form (BCNF). ● The table should not have any Multi-valued Dependency.
Fifth Normal Form (5NF)	A relation R is in 5NF if and only if it satisfies the following conditions: ● R should be already in 4NF. ● It cannot be further non loss decomposed (join dependency)

First Normal Form

A relation (table) is said to be in First Normal Form (1NF) if:

- All the attributes (columns) contain only atomic (indivisible) values.
- Each column contains values of a single type.
- Each record (row) is unique, meaning it can be identified by a primary key.
- There are no repeating groups or arrays in any row.

Eg:

Course	Content
Programming	Java, c++
Web	HTML, PHP, ASP

We re-arrange the relation (table) as below, to convert it to First Normal Form.

Course	Content
Programming	Java
Programming	c++
Web	HTML
Web	PHP
Web	ASP

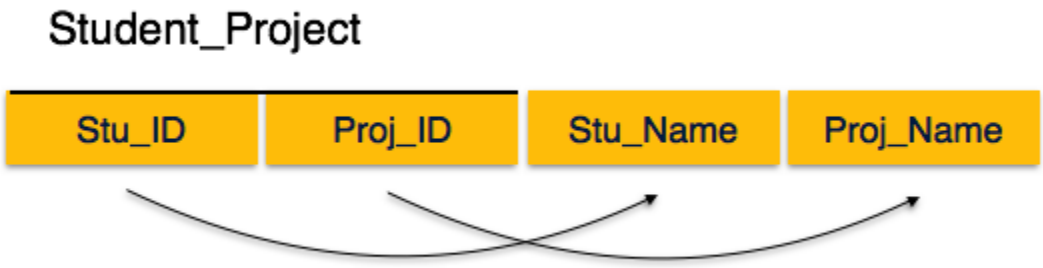
Each attribute must contain only a single value from its pre-defined domain.

Second Normal Form

The second normal form, need to understand the following −

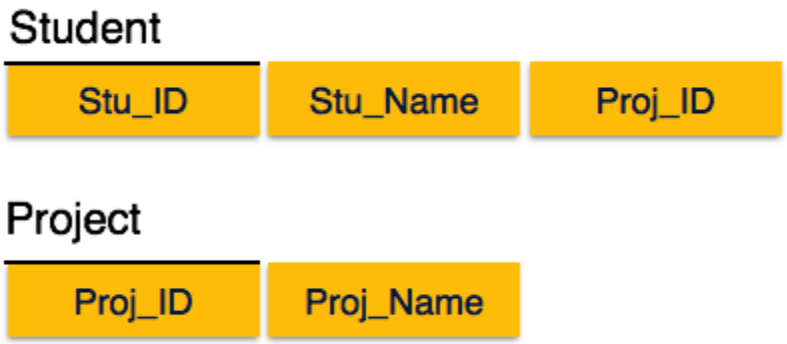
- **Prime attribute** − An attribute, which is a part of the candidate-key, is known as a prime attribute.
- **Non-prime attribute** − An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

If we follow second normal form, then every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if X → A holds, then there should not be any proper subset Y of X, for which Y → A also holds true.



We see here in Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non-key attributes, i.e. Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attribute individually. But we find that Stu_Name can be identified by Stu_ID and

Proj_Name can be identified by Proj_ID independently. This is called partial dependency, which is not allowed in Second Normal Form.

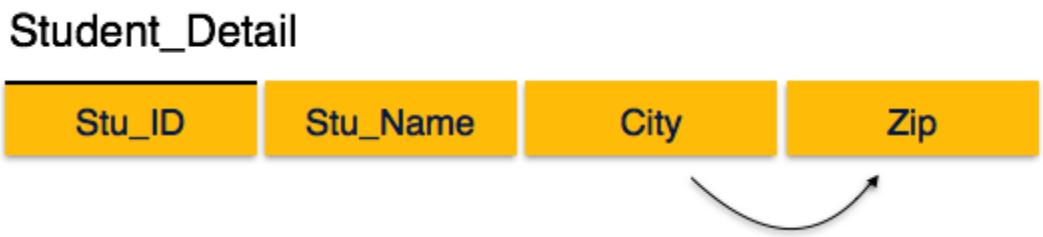


We broke the relation in two as depicted in the above picture. So there exists no partial dependency.

Third Normal Form

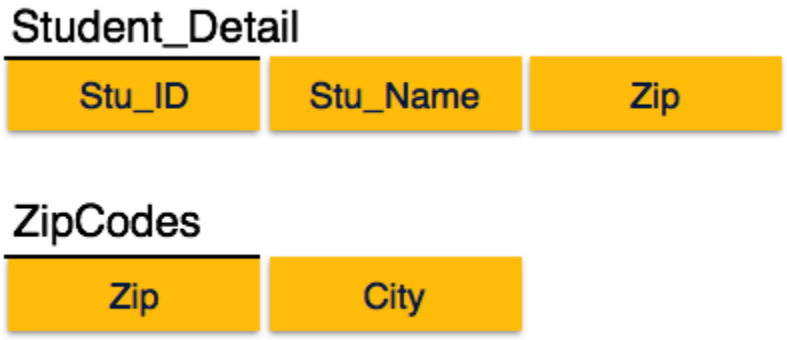
In Third Normal Form, it must be in Second Normal form and the following must satisfy −

- No non-prime attribute is transitively dependent on prime key attribute.
- For any non-trivial functional dependency, X → A, then either −
- X is a superkey or,
- A is prime attribute.



We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that City can be identified by Stu_ID as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, Stu_ID → Zip → City, so there exists transitive dependency.

To bring this relation into third normal form, we break the relation into two relations as follows −



Boyce-Codd Normal Form

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms. BCNF states that − For any non-trivial functional dependency, X → A, X must be a super-key.

In the above image, Stu_ID is the super-key in the relation Student_Detail and Zip is the super-key in the relation ZipCodes. So,

Stu_ID → Stu_Name, Zip

and

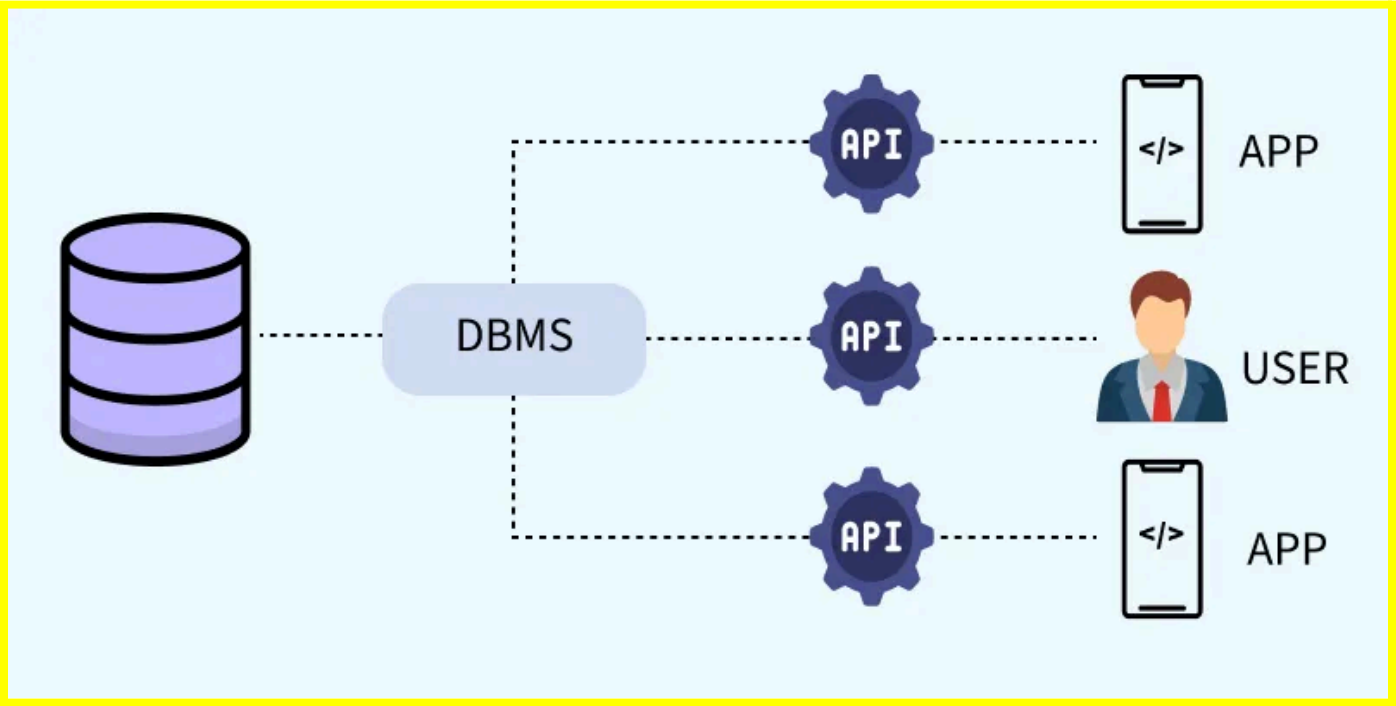
Zip → City

Which confirms that both the relations are in BCNF.

UNIT - I

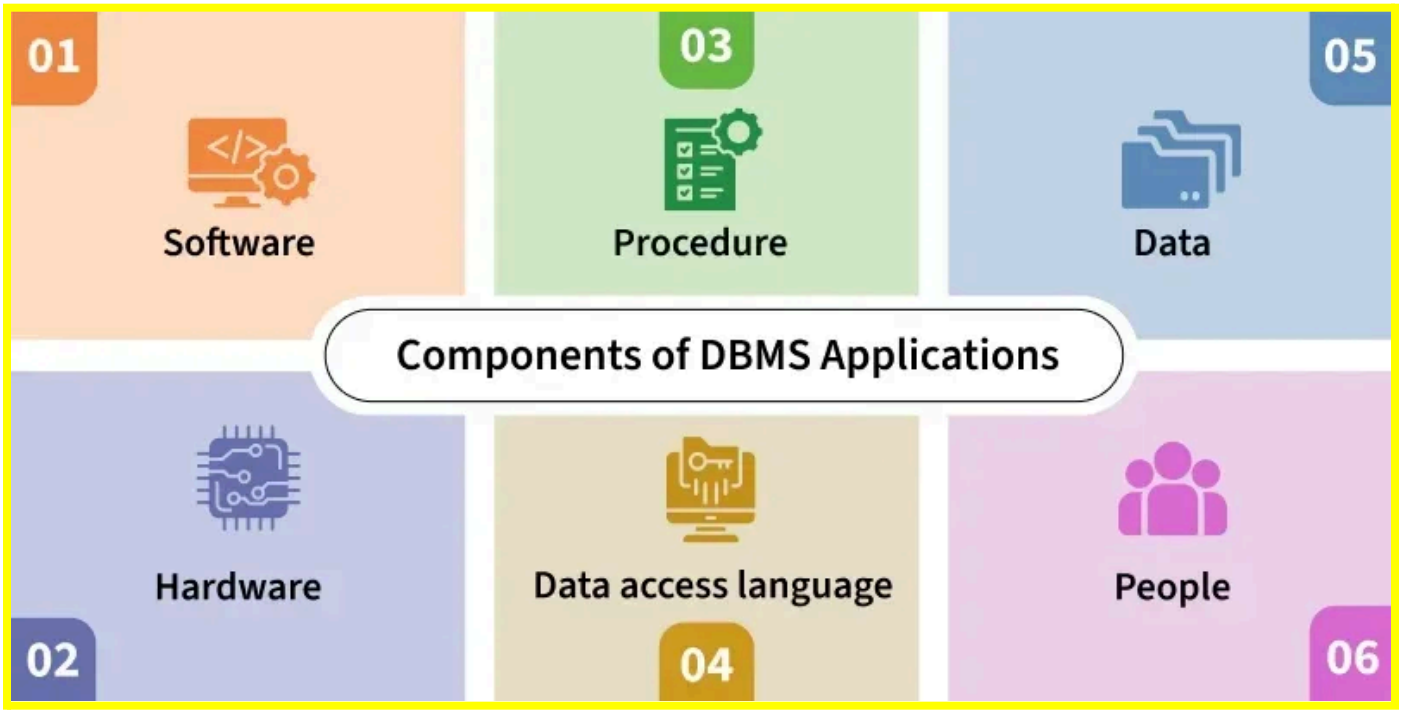
DBMS (Database Management System) is a software system that manages, stores, and retrieves data efficiently in a structured format. DBMS acts as a bridge between a central database and multiple clients, including apps and users.

- A DBMS connects the central database with multiple clients (applications and users).
- It allows users to create, update, and query databases efficiently.
- Ensures data integrity, consistency, and security across multiple users and applications.
- Reduces data redundancy and inconsistency through centralized control.
- Supports concurrent data access, transaction management, and automatic backups.
- Uses APIs(Application Programming Interface) to handle data requests, ensuring secure and efficient access.



Components of DBMS Applications

Any DBMS based application is made up of six key components that work together to handle data effectively.



Components of DBMS Applications

1. Hardware

- Physical devices like servers, disks, input-output devices (keyboard, monitor, printer).
- Stores and processes data; interfaces between real-world inputs and digital systems.
- Examples: Personal computer hard disk, RAM, network devices used for DBMS operations.

2. Software

- Actual DBMS software like MySQL, Oracle, PostgreSQL.
- Includes the database engine, OS, network software, and application tools.
- Translates database access languages into operations.

3. Data

- Raw facts stored in structured or unstructured formats.
- **Operational Data**: Actual user data (e.g., name, age).
- **Metadata**: Data about data (e.g., storage time, size, data type).
- Core reason DBMS exists—to manage and store data efficiently.

4. Procedures

- Instructions and rules for using DBMS effectively.
- Covers setup, login/logout, data validation, backup, access control, and report generation.
- Helps ensure consistent and secure use of the system.

5. Database Access Language

- Used to interact with the database (create, read, update, delete data).
- Examples: SQL, MyAccess, Oracle PL/SQL.
- **DDL (Data Definition Language)** – CREATE, ALTER, DROP

- **DML (Data Manipulation Language)** – INSERT, UPDATE, DELETE

6. People

- Users interacting with DBMS at different levels:
- **Database Administrators (DBA)** – Manage security, performance, user access.
- **Developers** – Build applications using the database.
- **End Users** – Use applications to access the database (e.g., students, employees).

CLASSIFICATION OF DBMS

Database Management Systems (DBMS) are classified based on several key characteristics.

1. Classification by Data Model

The data model determines how data is structured, stored, and manipulated:

- **Relational DBMS (RDBMS):** Stores data in a tabular format with rows (records) and columns (attributes), linking tables using keys and querying via SQL (e.g., MySQL, PostgreSQL, Oracle).
- **Hierarchical DBMS:** Organizes data in a strict tree-like structure with parent-child relationships, where each child has only one parent (e.g., IBM IMS).
- **Network DBMS:** Extends the hierarchical model by allowing a child record to have multiple parent records, supporting many-to-many relationships (e.g., Integrated Data Store).
- **Object-Oriented DBMS (OODBMS):** Stores data as complex objects complete with attributes, methods, inheritance, and polymorphism (e.g., ObjectDB).
- **NoSQL DBMS:** Built for unstructured or semi-structured data, supporting document stores, key-value stores, column-family stores, and graph databases (e.g., MongoDB, Cassandra).
- **Graph DBMS:** Stores entities as nodes and relationships as edges using specialized graph query languages (e.g., Neo4j).

2. Classification by Number of Users

- **Single-User Systems:** Support only one user at a time on a single computer.
- **Multi-User Systems:** Support multiple concurrent users accessing and modifying the database at the same time.

3. Classification by Number of Sites / Distribution

- **Centralized DBMS:** The entire database and DBMS software reside at a single server location.
- **Distributed DBMS:** Data and processing are spread across multiple network-connected sites:
    - **Homogeneous:** Uses the same DBMS software across all participating sites.
    - **Heterogeneous:** Uses different DBMS software or operating systems across sites, bridged by middleware.

4. Classification by Purpose

- **General-Purpose DBMS:** Designed to handle a wide variety of standard applications and general operations.

- **Special-Purpose DBMS:** Built for a specific, dedicated application (e.g., real-time Online Transaction Processing (OLTP) or railway reservation systems).

5. Classification by Cost / Licensing

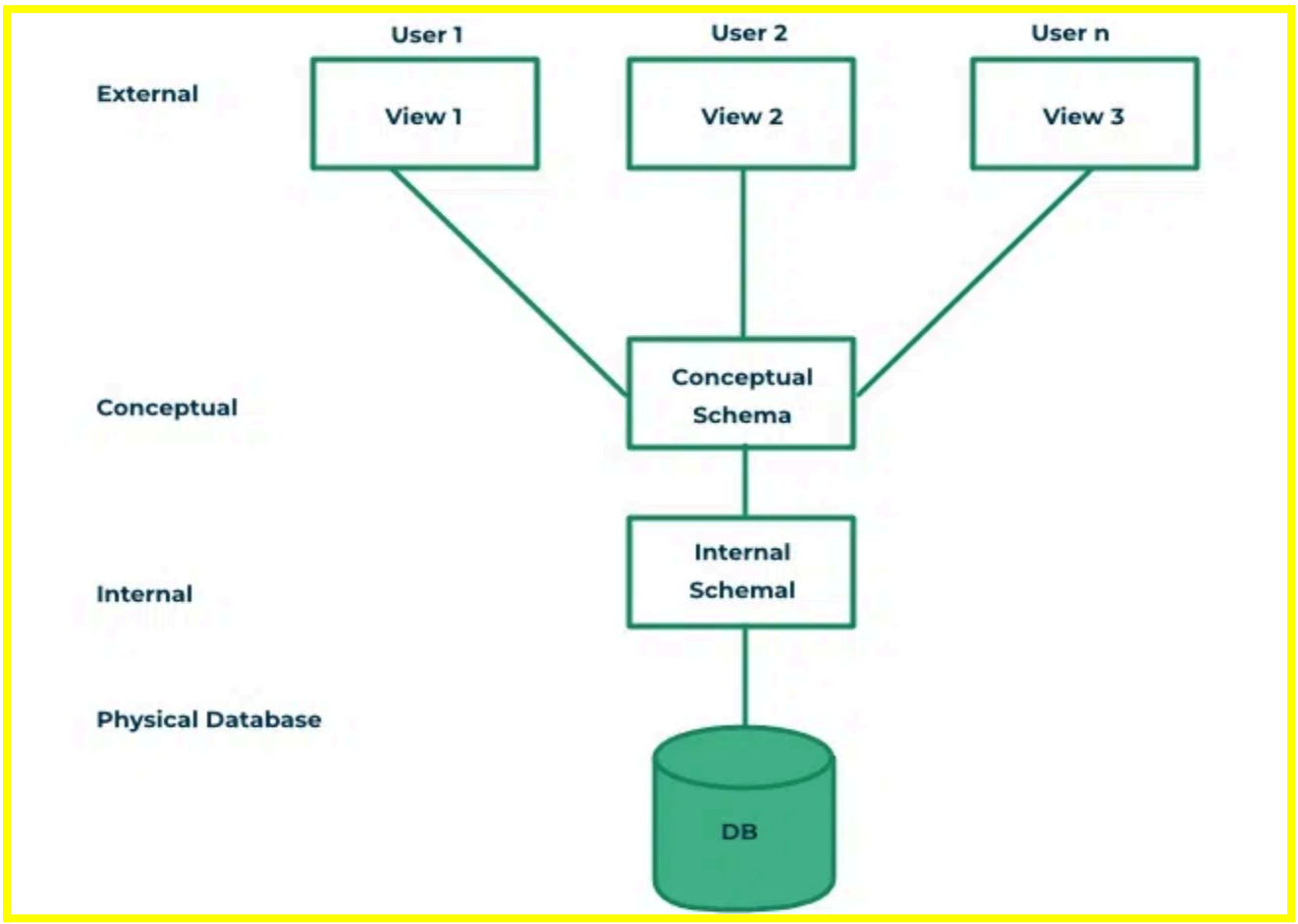
- **Open-Source DBMS:** Free to use, modify, and distribute (e.g., MySQL, PostgreSQL).
- **Proprietary (Commercial) DBMS:** Requires a paid license and support contract (e.g., Oracle Database, Microsoft SQL Server).

Three schema architecture of database

The **Three-Schema Architecture** (also known as the **ANSI/SPARC architecture**) is a framework used in Database Management Systems (DBMS) to separate the user applications from the physical database.

In DBMS, the 3-tier architecture is a **client-server architecture** that separates the **user interface**, **application processing**, and **data management** into three distinct tiers or layers. The 3-tier architecture is widely used in modern web applications and enterprise systems because it offers scalability, flexibility, and security. Here is a brief description of each tier in the 3-tier architecture:

1. **Presentation Tier (User Interface Layer)**
2. **Application Tier (Business Logic Layer)**
3. **Data Management Tier (Database Layer)**



Components of the 3-Tier Architecture

1. Presentation Tier (User Interface Layer)

The presentation tier is the **user interface** or **client layer** of the application. It is responsible for presenting data to the user and receiving input from the user. This tier communicates with the Application Tier to process user requests and display relevant information. This tier can be a web browser, mobile app, or desktop application.

2. Application Tier (Business Logic Layer)

The application tier is the **middle layer** of the 3-tier architecture. It acts as the intermediary between the Presentation Tier and the Data Management Tier. It is responsible for **processing** and **managing** the business logic of the application. This tier communicates with the presentation tier to receive user input and communicates with the data management tier to retrieve or store data. This tier may include application servers, web servers, or APIs.

3. Data Management Tier (Database Layer)

The **Data Management tier** is the bottom layer of the 3-tier architecture. It is responsible for managing and storing data. This tier can include databases, data warehouses, or any other persistent data storage solution. The data management tier communicates with the application tier save, retrieve, or manipulate data according to the business logic.

Advantages of 3-Tier Architecture

1. **Scalability**: The separation of the three tiers allows each layer to scale independently.

2. **Flexibility**: Each layer is loosely coupled, meaning you can modify or upgrade any of the layers without disrupting the others.

3. **Security**: The Presentation Tier and Application Tier interact via secure APIs or middleware, reducing the risk of unauthorized access.

4. **Maintainability**: It's easier to maintain and update. The separation of concerns also means that developers can specialize in one of the layers, allowing for more focused development and faster updates.

5. **Improved Performance**: The layered approach improves system performance by ensuring that each tier is optimized for its specific function.

Costs and risks of database approach

Costs of the Database Approach

- **Initial Setup and Licensing:** Purchasing commercial Database Management System (DBMS) software, high-speed processors, servers, and reliable storage hardware requires a large upfront capital expenditure.
- **Installation and Maintenance:** Multi-user systems involve complex deployment, ongoing vendor support, recurring annual maintenance fees, and hardware/network upgrades.
- **Personnel and Training:** Organizations must hire specialized staff—such as Database Administrators (DBAs), data architects, and support engineers—and continuously fund training so employees can operate the system.
- **Data Conversion and Migration:** Shifting legacy file-processing systems into a modern database

architecture demands substantial time, money, and organizational resources to rewrite applications and reformat historical data.
- **Performance Overhead:** Running queries, index management, and transaction logging consume substantial main memory and disk space.

Risks of the Database Approach

- **System Complexity:** DBMS software is massive and intricate, increasing the likelihood of configuration mistakes, design errors, and steep learning curves across departments.
- **Single Point of Failure:** Because a centralized database consolidates an organization's critical information, a severe hardware crash, software bug, or network outage can paralyze nearly all business applications simultaneously.
- **Data Loss Vulnerabilities:** Without explicit, well-tested backup and recovery protocols, human error or system corruption can result in catastrophic loss of shared data.
- **Security and Privacy Threats:** Centralizing data increases the damage potential of a breach, making the system a prime target for unauthorized access, leaks, or cyberattacks if security controls are weak.
- **Vendor Lock-In:** Deep integration with a specific proprietary database platform can make it difficult or costly to switch technologies later.
- **Organizational Friction:** Departmental disagreements over data ownership, access rules, and standards require strict governance and strong leadership to resolve.