The Shirpur Education Society's

R. C. Patel College of Engineering & Polytechnic, Shirpur

Department of Computer Engineering and Computer Science & Engineering

---

Course Title - DATABASE MANAGEMENT SYSTEM (DMS)

Course Code - 313302

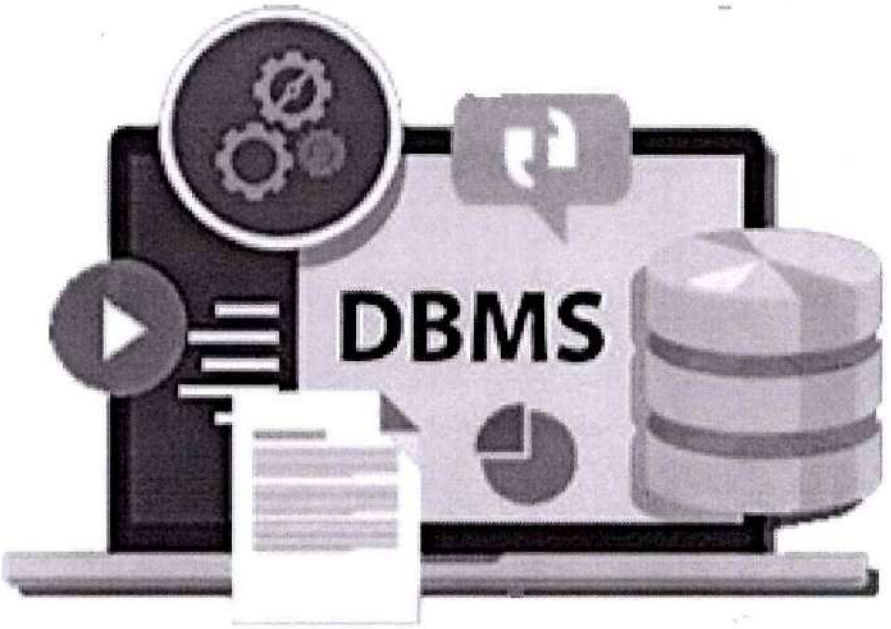
Programme Name - Computer Engineering and Computer Sci. & Engineering

Semester - Third

Unit - III Interactive SQL and Performance Tuning

Total Marks: 18

Prepared By: Mr. Akshay P. Chaudhari



Unit - III Interactive SQL and Performance Tuning

3.1 Introduction:

Queries are an essential part of a Database management system (DBMS). A query is a request made to the database to retriere, insert, update or delete data. In relational databases, queries are written using Structured Query Language (SQL) which is the standard language for interacting with databases.

A query allows users to communicate with a database. SQL is a declarative language, meaning the user specifies what data is required, not how it should be retrived. SQL helps users manage large volumes of data efficiently and accurately.

Main purposes of queries:

- Retrieve specific data from one or more tables
- Insert new records into tables
- Update existing records
- Delete unwanted records
- perform calculations and data analysis.

History of SQL:

SQL has a strong historical background linked with the development of the relational database model.

- In 1970, Dr. E.F. codd introduced the Relational Model, which laid the foundation for relational databases.
- In the early 1970s, IBM developed a language called SEQUEL (Structured English Query Language) to work with relational databases.
- Later, SEQUEL was renamed as SQL.

- In 1979, Oracle released the first commercial SQL-based relational database system.
- In 1986, SQL was accepted as a standard by ANSI (American National Standards Institute).
- In 1987, SQL was also standardized by ISO (International organization for standardization).

Today, SQL is widely used in almost all relational DBMS such as MySQL, Oracle, SQL Server, PostgreSQL and SQLite.

Data Types:

In SQL (Structured Query Language), data types define the kind of data that can be stored in a column of a table. Each column in a database table is required to have a name and a data type.

There are three main types of data types available in SQL: character string, numeric, date & time.

① Character String Data Types:

Data Type	Description
CHAR (size)	It is used to store fixed-length character data within the user specified length.
VARCHAR (size)	It is used to store variable-length character string data within the predefined length.

② Numeric Data Types:

This datatype is used to store a number values that can be decimal or floating-point values.

ⓐ Integer number:

The numbers which are not having any decimal point.

Data type	Description	Range
SMALLINT	Small Integers	-32,768 to 32,767
INT	Standard integer values	-2,147,483,648 to 2,147,483,647
BIGINT	Large integer numbers	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

(b) Floating-point number:

The numbers which are having a decimal point.

Data type	Description	Range
FLOAT	It is used for floating-point number.	-1.79E+308 to 1.79E+308
REAL	Similar to FLOAT, but with less precision	-3.40E+38 to 3.40E+38

(3) Date and Time Data Types:

Data type	Description	Example
DATE	It is used to store a valid date format yyyy-mm-dd with a fixed length is 10.	2026-08-21
TIME	It is used to store a valid time format HH:MM:SS with a fixed length is 8.	11:24:49
TIMESTAMP/ DATETIME	It is used to store both date and time fields format yyyy-mm-dd HH:MM:SS	2026-08-21 11:24:49

Data Definition Language (DDL):

DDL commands are used to define, create, modify and delete the structure of database objects such as tables, databases, indexes and views. These commands deal with the schema (structure) of the database rather than the actual data.

Characteristics of DDL commands :

- Used to define and manage database structure.
- Affect schema, not table records.
- Changes are permanent (auto-commit)
- Used mainly by database designers and administrators

The important DDL commands are :

① CREATE ② ALTER ③ DROP ④ TRUNCATE ⑤ RENAME

① CREATE Command :

The CREATE command is used to create new database objects such as databases, tables, views and indexes.

ⓐ Create Database :

Syntax : CREATE DATABASE database_name;

e.g. : CREATE DATABASE rcpcoepdb;

This command creates a new database named rcpcoepdb.

ⓑ Create Table :

Syntax :
```
CREATE TABLE table_name
(
    column_1  datatype,
    column_2  datatype,
       :
    column_n  datatype
);
```

e.g :
```
CREATE TABLE student
(
    ROLLNO  INT,
    Name    VARCHAR(50),
    Age     INT,
    Marks   FLOAT
);
```

Result :

ROLLNO	Name	Age	marks
NULL	NULL	NULL	NULL

(2) ALTER Command:

The ALTER command is used to modify the structure of an existing table. It can add, delete or modify columns and constraints.

(a) Add a column:

Syntax: ALTER TABLE table_name
ADD column_name datatype;

e.g: ALTER TABLE student
ADD Address VARCHAR(100);

Result:

RollNO	Name	Age	Marks	Address
NULL	NULL	NULL	NULL	NULL

(b) Modify a Column:

Syntax: ALTER TABLE table_name
MODIFY column_name datatype;

e.g: ALTER TABLE Student
MODIFY marks INT;

Note: only the data type changes, data remains safe.

(c) Drop a column:

Syntax: ALTER TABLE table_name
DROP column_name;

e.g: ALTER TABLE Student
DROP Address;

Result:

RollNO	Name	Age	Marks
NULL	NULL	NULL	NULL

(3) DROP Command:

The DROP command is used to delete database objects permanently. Once dropped, the object and its data are completely removed.

(a) Drop Table:

Syntax: DROP TABLE table_name;
DROP TABLE Student;

- Table deleted permanently.
- Structure deleted
- All data deleted

(6) Drop Database :

Syntax : DROP DATABASE database_name;
e.g : DROP DATABASE rcpcoepdb;

- DROP removes both the structure and the data. It cannot be undone.

(4) TRUNCATE Command :

This command is used to remove all records from a table, but it keeps the table structure intact.

Syntax : TRUNCATE TABLE table_name;
e.g : TRUNCATE TABLE student;

- Deletes all rows quickly.
- Does not remove table structure.
- Cannot be rolled back.
- Faster than DELETE.

(5) RENAME command :

This command is used to change the name of a database object, mainly tables.

Syntax : RENAME TABLE table_name TO new_name;
e.g : RENAME TABLE student TO student_info;

- only the table name changes, the structure & data remain the same.

Difference Between DROP and TRUNCATE :

Feature	DROP	TRUNCATE
Removes structure	Yes	NO
Removes Data	Yes	Yes
Can be rolled back	NO	NO
Faster	NO	Yes

Advantages of DDL Commands :

- Helps in designing and managing database structure.
- Ensures data integrity using constraints.
- Easy modification of tables.
- Improves database organization.

Data Manipulation Language (DML) :

DML commands are used to insert, retrieve, modify and delete data stored in database tables. Unlike DDL commands, DML commands work on the data, not on the structure of the table.

DML commands can be commited or rolled back, which makes them very important for transaction management.

Main DML commands :

① INSERT ② SELECT ③ UPDATE ④ DELETE

Example Table : Initial Data in Student Table

RollNO	Name	Age	Marks
1	Mayur	20	85
2	Riya	19	78
3	Rahul	21	90

① INSERT Command :

The INSERT command is used to add new records into a table.

ⓐ Insert single Record :

Syntax : INSERT INTO table_name
VALUES (value1, value2, value3, ...);

e.g : INSERT INTO student
VALUES (4, 'Neha', 20, 88);

RollNO	Name	Age	Marks
1	Mayur	20	85
2	Riya	19	78
3	Rahul	21	90
4	Neha	20	88

(b) Insert with selected columns:

Syntax: INSERT INTO table_name(column1, column2, ---)
VALUES (value1, value2, ----);

e.g: INSERT INTO Student (RollNo, Name, marks)
VALUES (5, 'Amit', 75);

RollNO	Name	Age	marks
5	Amit	NULL	75

(2) SELECT command:

The SELECT command is used to retrieve data from one or more tables.

(a) Select all records:

Syntax: SELECT *
FROM table_name;

e.g: SELECT *
FROM Student;

RollNO	Name	Age	Marks
1	Mayur	20	85
2	Riya	19	78
3	Rahul	21	90
4	Neha	20	88
5	Amit	NULL	75

(b) Select Specific columns:

Syntax: SELECT column1, column2, ...
FROM table_name;

e.g: SELECT Name, marks
FROM Student;

Name	Marks
Mayur	85
Riya	78
Rahul	90
Neha	88
Amit	75

(c) Select with WHERE condition:

Syntax: SELECT column1, column2, ...
FROM table_name
WHERE condition;

e.g: SELECT *
FROM Student
WHERE marks > 80;

RollNO	Name	Age	marks
1	mayur	20	85
3	Rahul	21	90
4	Neha	20	88

(3) UPDATE command:

The UPDATE command is used to modify existing records in a table.

(a) Update single Record:

Syntax: UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;

e.g: UPDATE Student
SET marks = 82
WHERE RollNO = 2;

Result:

RollNO	Name	Age	marks
2	Riya	19	82

(b) Update multiple columns:

e.g: UPDATE Student
SET Age = 20, marks = 80
WHERE RollNO = 5;

RollNO	Name	Age	marks
5	Amit	20	80

(c) Update multiple Rows:

e.g: UPDATE Student
SET marks = marks + 5
WHERE Age = 20;

- Adds grace marks to all students aged 20.

(4) DELETE command:

The DELETE command is used to remove records from a table.

(a) Delete specific Record:

Syntax: DELETE FROM table_name
WHERE condition;

e.g: DELETE FROM Student
WHERE RollNO = 4;

Result:

RollNO	Name	Age	marks
1	Mayur	20	85
2	Riya	19	82
3	Rahul	21	90
5	Amit	20	80

(b) Delete All Records :

Syntax : DELETE FROM table_name;

e.g : DELETE FROM student;

- It deletes all rows but table structure remains.

Difference between DELETE and TRUNCATE :

Feature	DELETE	TRUNCATE
Type	DML	DDL
Removes Rows	yes	yes
WHERE Clause	Allowed	Not Allowed
Rollback	Possible	Not Possible
Speed	Slow	fast

Advantages of DML commands :

- Allows data insertion, modification and deletion.
- Supports transaction control (COMMIT, ROLLBACK)
- Ensures data consistency
- Easy data manipulation

Data control Language (DCL) :

DCL helps in controlling to access the stored data. It is mainly used for revoke and to grant the user for required access to a database. DCL commands manage security, access and permissions. They control who can view or edit data, ensuring only authorized users can modify sensitive information.

① GRANT command :

It is employed to grant a privilege to a user. GRANT command allows specified users to perform ~~sepe~~ specific tasks.

Syntax: GRANT privilege_name on objectname to user;

Here,

- privilege names are SELECT, UPDATE, DELETE, INSERT, ALTER, ALL
- Objectname is table name
- User is the name of the user to whom we grant privileges

e.g: GRANT SELECT, UPDATE ON Employee TO Athary;

② REVOKE Command:

It is employed to remove a privilege from a user. REVOKE helps the owner to cancel previously granted permissions.

Syntax: REVOKE privilege_name on objectname from user;

Here,

e.g: REVOKE SELECT, UPDATE ON Employee FROM Athary;

Transaction Control Language (TCL):

TCL in SQL is used to manage and control transactions in a database. It allows users to save changes permanently, undo changes, or create checkpoints within a transaction.

The main purpose of TCL is to maintain data integrity by ensuring that database operations are completed successfully or rolled back if an error occurs. TCL commands such as commIT, ROLLBACK and SAVEPOINT help control how and when changes are applied to the database.

The main TCL Commands in SQL include:

① SAVEPOINT
② ROLLBACK
③ commIT

① SAVEPOINT :

The SAVEPOINT command in SQL allows us to set a point within a transaction to which can rollback without affecting the entire transaction. This is particularly useful in managing long or complex transactions.

~~Consider~~

e.g:
```
UPDATE Customers SET age = 24
WHERE customer_id = 101;
SAVEPOINT SP1;
```

Here, SAVEPOINT SP1 allows us to rollback to those points without undoing all previous changes.

② ROLLBACK :

The ROLLBACK command in SQL is used to undo transactions that have not been committed to the database. It's a way to revert the state of the database to the last committed state.

e.g:
```
DELETE FROM Customers
WHERE customer_id = 102;
ROLLBACK;
```

Here, the ROLLBACK statement here undoes the deletion of the customer.

③ COMMIT :

The COMMIT command in SQL is used to save all changes made during the current transaction permanently.

e.g:
```
UPDATE Customers SET city = 'Dhule'
WHERE customer_id = 102;
COMMIT;
```

Here, the COMMIT statement permanently save the changes made to the customers table.

3.2 Clauses and Join:

Different types of clauses:

(1) WHERE clause:

The SQL WHERE clause is used to filter records based on specific conditions. It ensures that only the rows meeting the given criteria are affected or returned by a SELECT, UPDATE or DELETE statement.

Without the WHERE clause, these statements would apply to all rows in the table. The clause is commonly used with comparison operators, logical operators, and other expressions to precisely target specific data.

Syntax:
```
DML_statement column1, column2,...
FROM table_name
WHERE [condition];
```

Here, the DML-statement can be any statement, such as SELECT, UPDATE, DELETE etc. We can specify a condition using the comparision or logical operators such as, >, <, =, LIKE, NOT, etc.

E.g:
```
SELECT *
FROM customers
WHERE Amount > 2000;
```

② GROUP BY clause :

The SQL GROUP BY clause is used to group (organize) rows that have the same values in specified columns into summary rows. It is useful when we want to summarize data and get totals or averages for each category or group.

It is used with aggregate functions like COUNT(), SUM(), AVG(), MAX() and MIN() to perform calculations on each group of data.

Syntax:
```
SELECT column_name(s)
FROM table_name
GROUP BY column_name(s);
```

Sample Table (Customer):

ID	Customer_Name	Age	City	Amount
1	Pankaj	33	Dhule	1200.00
2	Atharv	29	Pune	2500.00
3	Rahul	45	Shirpur	1800.00
4	Ananda	27	Shirpur	3700.00
5	Harshal	29	Dhule	3100.00
6	Abhijit	38	Shirpur	2200.00
7	prashant	44	Pune	1500.00

e.g:
```
SELECT city, COUNT(Customer_Name) AS customers
FROM customer
GROUP BY city;
```

city	customers
Dhule	2
Shirpur	3
Pune	2

③ ORDER BY Clause :

The ORDER BY clause is used to sort the result set of a query by one or more columns. By default, it sorts the data in ascending order (from lowest to highest), but we can use the DESC keyword to sort the result in descending order (from highest to lowest).

This clause is placed at the end of query, following the WHERE, HAVING and GROUP BY clauses, if present.

Syntax:
```
SELECT column_list
FROM table_name
ORDER BY column1, column2, ... [ASC/DESC];
```

e.g:
```
SELECT customer_Name, Age, City
FROM customers
ORDER BY customer_Name ASC;
```

Customer_Name	Age	City
Abhijit	38	Shirpur
Ananda	27	Shirpur
Atharv	29	Pune
Harshal	29	Dhule
Pankaj	33	Dhule
Prashant	44	Pune
Rahul	45	Shirpur

④ Having clause :

The HAVING clause is used to filter grouped records based on a condition, involving aggregate functions such as COUNT(), SUM(), AVG(), MAX() or MIN(). It works in conjunction with the GROUP BY clause and is applied after the data has been grouped.

The HAVING clause was introduced in SQL because the WHERE clause cannot be used with aggregate functions.

Syntax:
```
SELECT column1, column2, aggregate_fun (column)
FROM table_name
GROUP BY column1, column2
HAVING condition;
```

E.g:
```
SELECT city, AVG (Amount) AS AVG_AMT
FROM customers
GROUP BY city
HAVING AVG (Amount) > 2000;
```

city	AVG_AMT
Dhule	2150.00
Shirpur	2800.00

Joins :

The JOIN clause is used to combine rows from two or more tables based on a related column between them. It allows us to retrieve data from multiple tables as if they were a single table. Using JOIN helps in organizing data stored in different tables and returning a meaningful dataset.

Types of Joins :

There are different types of joins used to combine data from multiple tables. They are as follows:

1. INNER JOIN
2. LEFT JOIN
3. RIGHT JOIN
4. FULL JOIN
5. CROSS JOIN

Assume we have created a table with the names customers and orders for use in joins as follows:

Table - customers

ID	Name	Age	city
1	Pankaj	33	Dhule
2	Atharv	29	Pune
3	Rahul	45	Shirpur
4	Ananda	27	Shirpur
5	Harshal	29	Dhule
6	Abhijit	38	Shirpur

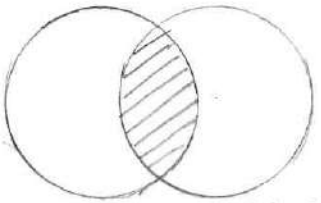
Table - orders

OID	Date	Cust-ID	Amount
101	2026-05-30	2	2700.00
102	2026-06-04	1	1800.00
103	2026-06-23	2	3500.00
104	2026-07-10	6	2400.00
105	2026-07-18	9	1500.00

① INNER JOIN :

The INNER JOIN keyword in SQL selects only the rows that have matching values in both tables. If there is no match, the row is not included in the result.

Syntax:
```
SELECT column1, column2, ---
FROM table1 INNER JOIN table2
ON table1.col_name = table2.col_name;
```



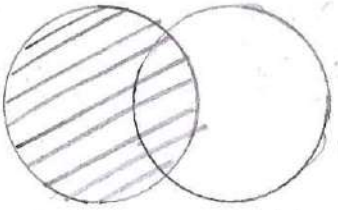
* e.g: SELECT customers.ID, customers.Name, Orders.Amount
FROM customers INNER JOIN Orders
ON customers.ID = Orders.cust_ID;

ID	Name	Amount
1	Pankaj	1800.00
2	Atharv	2700.00
2	Atharv	3500.00
6	Abhijit	2400.00

(2) LEFT ~~JOIN~~ JOIN:

The LEFT JOIN keyword returns all rows from the left table and the matched rows from the right table. If no match exists, NULL values are returned for columns from the right table.

Syntax: SELECT column1, column2, ...
FROM table1 LEFT JOIN table2
ON table1.col_name = table2.col_name;



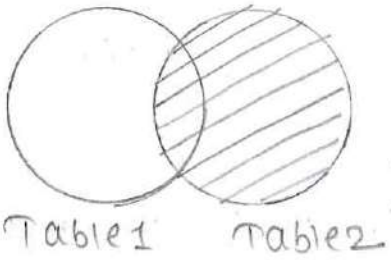
e.g: SELECT Customers.ID, Customers.Name, Orders.Amount
FROM ~~table1~~ customers LEFT JOIN Orders
ON customers.ID = orders.cust_ID;

ID	Name	Amount
1	Pankaj	1800.00
2	Atharv	2700.00
2	Atharv	3500.00
3	Rahul	NULL
4	Ananda	NULL
5	Harshal	NULL
6	Abhijit	2400.00

③ RIGHT JOIN :

The RIGHT JOIN keyword returns all rows from the right table, and the matched rows from the left table. If there is no match, NULL values are returned from the left table.

Syntax : SELECT column1, column2, ...
FROM table1 RIGHT JOIN table2
ON table1.col_name = table2.col_name;



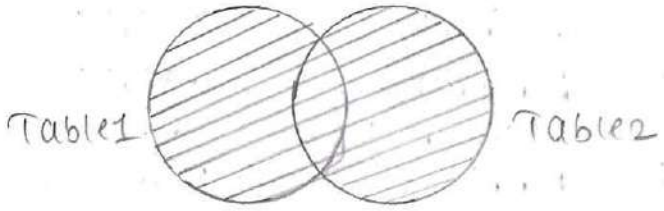
e.g. : SELECT customers.ID, customers.Name, Orders.Amount
FROM customers RIGHT JOIN orders
ON customers.ID = orders.cust_ID;

ID	Name	Amount
1	pankaj	1800.00
2	Atharv	2700.00
2	Atharv	3500.00
6	Abhijit	2400.00
NULL	NULL	1500.00

④ FULL JOIN :

The FULL JOIN keyword returns all rows from both tables, with NULL where there is no match. Some databases use FULL OUTER JOIN.

syntax : SELECT column1, column2, ...
FROM table1 FULL JOIN table2
ON table1.col_name = table2.col_name;



e.g: SELECT customers.ID, customers.Name, orders.Amount
FROM customers FULL JOIN orders
ON customers.ID = orders.cust-ID;

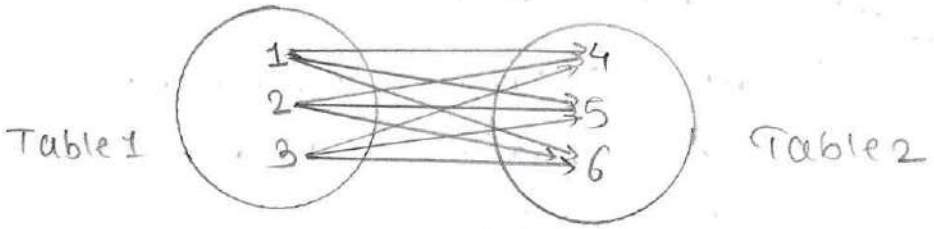
ID	Name	Amount
1	Pankaj	1800.00
2	Atharv	2700.00
2	Atharv	3500.00
3	Rahul	NULL
4	Ananda	NULL
5	Harshal	NULL
6	Abhijit	2400.00
NULL	NULL	1500.00

(5) CROSS JOIN:

A CROSS JOIN returns the cartesian product of two tables. A cartesian product means every row from the first table is combined with every row from the second table, producing all possible row combinations.

E.g: If Table1 has 3 rows and Table2 has 4 rows, the result of a CROSS JOIN will have $3 \times 4 = 12$ rows.

Syntax: SELECT column_name(s)
FROM table1 CROSS JOIN table2;



e.g: SELECT customers.Name, Orders.OID
FROM customers CROSS JOIN orders;

Nested Queries :

A Nested query, is a query within another query. It is also known as Inner query or subquery and the query containing it is the outer query. The inner query executes first, and its result is used by the outer query.

The outer query can contain the SELECT, INSERT, UPDATE and DELETE statements. We can use the subquery as a column expression, as a condition in SQL clauses, and with operators like =, >, <, >=, <=, IN, BETWEEN, etc.

Rules for writing nested queries :

- Nested queries must be enclosed with Paranthesis.
- Nested queries can be nested within another nested query.
- A nested query must contain the SELECT query and the FROM clause always.
- A nested query can return a single value, a single row, a single column, or a whole table. They are called scalar nested queries.

Syntax:
```
SELECT column1, column2, ---
FROM table1, table2, ---
WHERE column_name OPERATOR (SELECT col1, col2, ---
                            FROM table
                            WHERE condition);
```

Sample Table (Employee)

EmpID	Name	Department	Salary
1	Amit	HR	25000
2	Neha	IT	40000
3	Rahul	Sales	30000
4	Priya	IT	50000
5	kiran	HR	28000

Example 1 : Employees with salary greater than average salary.

```
SELECT Name, Salary
FROM Employee
WHERE Salary > (SELECT AVG(Salary)
                FROM Employee);
```

(outer query: SELECT, FROM, WHERE; inner query: SELECT AVG(Salary) FROM Employee)

Name	Salary
Neha	40000
Priya	50000

Example 2 : Employees working in IT Department.

```
SELECT Name
FROM Employee
WHERE Department = (SELECT Department
                    FROM Employee
                    WHERE Name = 'Neha');
```

Name
Neha
priya

Advantages of Nested Queries :

- makes complex queries easier.
- Improves readability.
- Useful for comparisions and filtering data.
- Can replace temporary tables in many cases.

Disadvantages :

- Can be slower for large databases.
- Complex nested queries are harder to debug.

3.3 operators :

Operators are special symbols or keywords used to perform operations on data stored in database tables. They are mainly used in SQL statements such as SELECT, UPDATE, DELETE and WHERE clauses to filter, compare and manipulate data.

SQL operators help users perform calculations, compare values, combine conditions and search for specific patterns in a database.

Types of SQL operators:

1. Relational operators
2. Arithmetic operators
3. Logical operators
4. Set operators

1. Relational operators:

comparison operators are used to compare two values.. They return either TRUE or FALSE.

Operator	meaning
=	Equal to
!= or <>	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

e.g 1:
```
SELECT *
FROM Employee
WHERE Salary > 25000;
```

e.g 2:
```
SELECT *
FROM Employee
WHERE Dept <> 'HR';
```

2. Arithmetic operators:

Arithmetic operators are used to perform mathematical calculations on numeric data.

Operator	meaning	Example
+	Addition	salary + 1000
-	Subtraction	salary - 500
*	Multiplication	salary * 2
/	Division	salary / 2
%	Modulus (Remainder)	salary % 2

e.g:
```
SELECT Name, Salary, Salary + 5000 AS Increased_Salary
FROM Employee;
```

③ Logical operators :

Logical operators are used to combine multiple conditions in a query.

operator	meaning
AND	All conditions must be true
OR	At least one condition must be true
NOT	Reverses the condition

e.g :
```
SELECT *
FROM Employee
WHERE Department = 'IT' AND Salary > 35000;
```

④ Set operators :

Set operators are used to combine results from multiple SELECT statements.

operator	meaning
UNION	Combines results and removes duplicates
UNION ALL	Combines all results including duplicates
INTERSECT	Returns common records
MINUS / EXCEPT	Returns records from first query not in second

e.g :
```
SELECT Name FROM HR_Employee
UNION
SELECT Name FROM IT_Employee;
```

3.4 Functions :

Functions are predefined commands provided by the DBMS to perform specific operations on data. Functions help users manipulate, calculate, format and retrieve data from database tables easily.

Functions take input values, process them and return an output value. SQL functions reduce the complexity of queries and improve efficiency while working with databases.

Types of functions :

① Numeric functions
② Date and Time functions
③ String functions
④ Aggregate functions

① Numeric functions :

Numeric functions perform mathematical operations.

Function	Description	Example	Output
ABS()	Returns absolute value	ABS(-15)	15
CEIL() / CEILING()	Returns next highest integer	CEIL(4.3)	5
FLOOR()	Returns next lowest integer	FLOOR(4.9)	4
ROUND()	Rounds a number	ROUND(12.456,2)	12.46
TRUNC()	Removes decimal part	TRUNC(12.456,2)	12.45
MOD()	Returns remainder	MOD(17,5)	2
POWER()	Returns power value	POWER(2,4)	16
SQRT()	Returns square root	SQRT(64)	8
SIGN()	Returns sign of number	SIGN(-20)	-1
EXP()	Returns exponential value	EXP(1)	2.718
LOG()	Returns logarithm value	LOG(10)	1
PI()	Returns value of pi	PI()	3.14159
RAND()	Generates random number	RAND()	Random value
GREATEST()	Returns largest value	GREATEST(10,20,30)	30
LEAST()	Returns smallest value	LEAST(10,20,30)	10

② Date and Time functions :

Date and Time functions are used to perform operations on date and time values stored in database tables. These functions help users retrieve the current date and time, extract parts of dates, format dates and perform calculations involving dates and time.

No	Function	Query	Output
1)	CURRENT_DATE	SELECT CURRENT_DATE;	2026-05-22
2)	CURRENT_TIME	SELECT CURRENT_TIME;	11:30:25
3)	NOW()	SELECT NOW();	2026-05-22 11:30:25
4)	SYSDATE()	SELECT SYSDATE();	2026-05-22 11:30:25
5)	DAY()	SELECT DAY('2026-05-22');	22
6)	MONTH()	SELECT MONTH('2026-05-22');	5
7)	YEAR()	SELECT YEAR('2026-05-22');	2026
8)	HOUR()	SELECT HOUR('10:45:20');	10
9)	MINUTE()	SELECT MINUTE('10:45:20');	45
10)	SECOND()	SELECT SECOND('10:45:20');	20
11)	DATE_ADD()	SELECT DATE_ADD('2026-05-22',INTERVAL 10 DAY);	2026-06-01
12)	DATE_SUB()	SELECT DATE_SUB('2026-05-22',INTERVAL 5 DAY);	2026-05-17
13)	DATEDIFF()	SELECT DATEDIFF('2026-05-22', '2026-05-10');	12
14)	EXTRACT()	SELECT EXTRACT(YEAR FROM '2026-05-22');	2026
15)	DATE_FORMAT()	SELECT DATE_FORMAT('2026-05-22','%d-%m-%Y');	22-05-2026

③ String Functions:

String functions are predefined functions used to manipulate ~~between~~ character or text data stored in database tables.

Function	Description	Example Query	Result
UPPER()	Converts string to uppercase	SELECT UPPER(Name) FROM Employee;	AMIT
LOWER()	Converts string to lowercase	SELECT LOWER(Name) FROM Employee;	amit
LENGTH()	Returns length of string	SELECT LENGTH('Rahul');	5
CONCAT()	Combines strings	SELECT CONCAT(Name,'-',City) FROM Employee;	Amit-Mumbai
SUBSTRING() / SUBSTR()	Extracts part of string	SELECT SUBSTRING('Database',1,4);	Data
TRIM()	Removes spaces from both sides	SELECT TRIM(' SQL ');	SQL
LTRIM()	Removes left spaces	SELECT LTRIM(' SQL');	SQL
RTRIM()	Removes right spaces	SELECT RTRIM('SQL ');	SQL
REPLACE()	Replaces text in string	SELECT REPLACE('I like SQL', 'SQL', 'DBMS');	I like DBMS
INSTR()	Finds position of substring	SELECT INSTR('DATABASE','BASE');	5
LEFT()	Extracts characters from left	SELECT LEFT('Amit',2);	Am
RIGHT()	Extracts characters from right	SELECT RIGHT('Amit',2);	it
REVERSE()	Reverses string	SELECT REVERSE('Amit');	timA
ASCII()	Returns ASCII value of first character	SELECT ASCII('A');	65
CHAR()	Returns character for ASCII value	SELECT CHAR(65);	A
POSITION()	Finds position of substring	SELECT POSITION('a' IN 'Rahul');	2
LOCATE()	Finds location of substring	SELECT LOCATE('SQL','I Love SQL');	8
LPAD()	Pads string on left side	SELECT LPAD('SQL',6,'*');	***SQL
RPAD()	Pads string on right side	SELECT RPAD('SQL',6,'*');	SQL***
FORMAT()	Formats numbers as string	SELECT FORMAT(12345.678,2);	12,345.68
SPACE()	Returns spaces	SELECT CONCAT('Hello', SPACE(3), 'SQL');	Hello SQL
STRCMP()	Compares two strings	SELECT STRCMP('ABC','ABC');	0
REPEAT()	Repeats string multiple times	SELECT REPEAT('SQL ',3);	SQL SQL SQL
INSERT()	Inserts string at position	SELECT INSERT('HelloWorld',6,5,'SQL');	HelloSQL
MID()	Extracts substring	SELECT MID('Database',2,4);	atab
FIELD()	Returns index position	SELECT FIELD('B','A','B','C');	2

④ Aggregate Functions:

Aggregate functions used to perform calculations on a set of values and return a single result. They are mainly used with the SELECT statement and are very common in reports and data analysis. These are often used along with GROUP BY & HAVING Clauses.

Sample Table – Student

RollNo	Name	Department	marks
1	Athary	CS	85
2	Riya	CS	78
3	Rahul	IT	90
4	Neha	IT	88
5	Amit	CS	75

Function	Description	Query	Result
COUNT()	counts number of rows	SELECT COUNT(*)	5
SUM()	calculate total sum	SELECT SUM(marks)	416
AVG()	calculate average value	SELECT AVG(marks)	83.2
MAX()	find maximum value	SELECT MAX(marks)	90
MIN()	find minimum value	SELECT MIN(marks)	75

Aggregate functions with GROUP BY:

The GROUP BY clause is used to group rows with same values in a column.

e.g 1: Department-wise Average marks:

```
SELECT Department, AVG(marks)
FROM student
GROUP BY Department;
```

Department	AVG(marks)
CS	79.33
IT	89

Aggregate functions with HAVING:

The HAVING clause is used to apply conditions on groups.

e.g: Departments with Average marks > 80

```
SELECT Department, AVG(marks)
FROM student
GROUP BY Department
HAVING AVG(marks) > 80;
```

Department	AVG(marks)
IT	89

Difference between WHERE and HAVING:

Feature	WHERE	HAVING
Used with	Rows	Groups
Works with	SELECT, UPDATE, DELETE	GROUP BY
Aggregate functions	Not Allowed	Allowed

3.5 Views, Sequences, Indexes :

Views, sequences and Indexes are important database objects in SQL that help improve data management, security, performance and automation in database systems.

1] Views :

A view is a virtual table created from one or more existing tables. A view does not store data physically inside the database. Instead, it stores the SQL query used to generate the data dynamically. Whenever a user accesses a view, the database executes the stored query and displays the result. Views are used to simplify complex queries, improve security, and provide customized access to data.

Creating a view :

The CREATE VIEW statement is used to create a new view.

```
CREATE VIEW IT_Employee AS
SELECT Name, Salary
FROM Employee
WHERE Department = 'IT';
```

Display view Data :

```
SELECT *
FROM IT_Employees;
```

Updating views :

A view can be modified using the CREATE OR REPLACE VIEW statement.

Syntax: CREATE OR REPLACE VIEW view-name AS
SELECT columns
FROM table-name
WHERE condition;

e.g: CREATE OR REPLACE VIEW IT_Employees AS
SELECT Name, salary, Department
FROM Employee
WHERE salary > 30000;

Dropping Views:

The DROP VIEW statement is used to remove a view permanently from the database.

Syntax: DROP VIEW view_name;

e.g: DROP VIEW IT_Employees;

Difference Between Table and View:

Feature	Table	view
Stores Data	yes	NO
Physical storage	Required	Not required
Data source	Independent	Derived from tables
performance	Faster	Sometimes slower
Security	less flexible	more secure

2] Sequences:

A sequence is a database object used to generate unique numeric values automatically in a sequential order.

Sequences are mainly used for generating:

- Employee IDs
- Student IDs
- order Numbers
- Invoice Numbers

A sequence generates numbers independently of tables, which makes it very useful in multi-user database environments.

The sequence can:
- Start from a specified number
- Increase or decrease by a specified value
- Generate unique values
- Continue automatically without manual intervention

Syntax of creating sequence:

```
CREATE SEQUENCE Sequence_Name
START WITH Starting_value
INCREMENT BY increment_value
MINVALUE minimum_value
MAXVALUE maximum_value
CACHE cache_size
CYCLE;
```

e.g:
```
CREATE SEQUENCE Emp_Seq
START WITH 100
INCREMENT BY 1;
```

Generated values: 100, 101, 102, 103, ----

Using sequence in INSERT Statement:
sequences are commonly used for automatic ID generation.

e.g:
```
INSERT INTO Employee
VALUES (Emp_seq.NEXTVAL, 'Rahul', 'Sales');
```

Altering sequences:
The ALTER SEQUENCE statement is used to modify an existing sequence.

Syntax:
```
ALTER SEQUENCE sequence_name
INCREMENT BY value
MAXVALUE value
MINVALUE value
CYCLE | NOCYCLE;
```

e.g: ALTER SEQUENCE Emp_seq
INCREMENT BY 2
MAXVALUE 1000;

Generated values: 100, 102, 104, 106, ----

Dropping sequences:

The DROP SEQUENCE statement removes a sequence permanently from the database.

syntax: DROP SEQUENCE sequence_name;

e.g: DROP SEQUENCE Emp_seq;

Use of sequence in table:

Sequences are mainly used with tables to insert automatic values into primary key columns.

step-1] create Employee Table:

```
CREATE TABLE Employee (
    EmpID NUMBER PRIMARY KEY,
    Name VARCHAR2(50),
    Department VARCHAR2(30),
    Salary NUMBER
);
```

step-2] create sequence:

```
CREATE SEQUENCE Emp_seq
START WITH 100
INCREMENT BY 1;
```

step-3] Use sequence while inserting Data:

The NEXTVAL keyword is used to generate the next sequence value.

Insert Records using sequence:

```
INSERT INTO Employee
VALUES (Emp_seq.NEXTVAL, 'Amit', 'HR', 25000);

INSERT INTO Employee
VALUES (Emp_seq.NEXTVAL, 'Neha', 'IT', 40000);

INSERT INTO Employee
VALUES (Emp_seq.NEXTVAL, 'Rahul', 'Sales', 30000);
```

EmpID	Name	Department	Salary
100	Amit	HR	25000
101	Neha	IT	40000
102	Rahul	Sales	30000

3] Index:

An index in SQL is a database object used to improve the speed and performance of data retrieval operations on a table. Indexes work similarly to the index of book. Instead of searching every page one by one, the index helps locate information quickly.

Without an index, the database scans the entire table to find required data. With an index, the database can directly locate records faster.

Employee Table

EmpID	Name	Department	Salary
101	Amit	HR	25000
102	Neha	IT	40000
103	Rahul	Sales	30000
104	Priya	IT	50000

Creating Index:

The CREATE INDEX statement is used to create an index on one or more columns.

Syntax: CREATE INDEX index_name.
ON table_name (column_name);

e.g: CREATE INDEX idx_name
ON Employee (Name);

Using Index: SELECT *
FROM Employee
WHERE Name = 'Neha';

Explanation: The database uses the index to quickly locate records where Name = 'Neha'.

Dropping Index:

Index can be removed using the DROP INDEX statement.

Syntax: DROP INDEX index_name;
eg: DROP INDEX idx_name;

Types of Indexes in SQL:

Index Type	Description	Example Query	Purpose
Single Column Index	Index on one column	CREATE INDEX idx_name ON Employee(Name);	Fast search on one column
Composite Index	Index on multiple columns	CREATE INDEX idx_emp ON Employee(Name, Department);	Fast multi-column search
Unique Index	Prevents duplicate values	CREATE UNIQUE INDEX idx_email ON Employee(Email);	Maintain uniqueness
Clustered Index	Physically sorts table data	CREATE CLUSTERED INDEX idx_empid ON Employee(EmpID);	Faster retrieval
Non-Clustered Index	Separate index structure	CREATE NONCLUSTERED INDEX idx_dept ON Employee(Department);	Faster searching
Full-Text Index	Used for text searching	CREATE FULLTEXT INDEX idx_desc ON Products(Description);	Search large text
Bitmap Index	Uses bitmap structure	CREATE BITMAP INDEX idx_gender ON Employee(Gender);	Data warehouse queries
Function-Based Index	Index on function/expression	CREATE INDEX idx_uppername ON Employee(UPPER(Name));	Improve function queries