**Data** is raw, unprocessed facts and figures without any context. It can be in the form of numbers, text, images, or symbols.

**Example:**

* 1001, Alice, 22
* 200, "Laptop", "Electronics"

This data alone doesn’t provide much meaning until it is structured and processed.

**Information** is processed, meaningful data that helps in decision-making. It is the result of analyzing, organizing, and interpreting raw data.

**Example:**

* **Data:** 1001, Alice, 22, CS101
* **Information:** "Alice (Student ID: 1001) is enrolled in the CS101 course."

Information provides context and value to data, making it useful.

A **database** is an organized collection of data that allows easy access, retrieval, storage, and management. It is designed to store large amounts of structured or unstructured data efficiently.

**Example:**  
A **student database** might have tables like:

A screenshot of a computer

AI-generated content may be incorrect.

A database stores this data systematically, allowing queries and modifications.

Let's take an example using **SQL** to see how **raw data** is transformed into **useful information** in a database.

**Step 1: Creating the Database and Tables**

We can create a **Student Management System** database with two tables: **Students** and **Courses**.

-- Create Students Table // CREATE TABLE table \_name (column1 datatype constraints, column2 datatype constraints, column3 datatype constraints…..);

# **CREATE TABLE Students (StudentID INT PRIMARY KEY, Name VARCHAR2(100),Age INT, CourseID INT);**

-- Create Courses Table // CREATE TABLE table \_name (column1 datatype constraints, column2 datatype constraints, column3 datatype constraints…..);

#**CREATE TABLE Courses (CourseID INT PRIMARY KEY, CourseName VARCHAR2(100));**

**Step 2: Inserting Raw Data**

Let us insert **raw data** into the tables.

-- Insert Courses // INSERT INTO table\_ name(Name, Age, CourseID) VALUES ('Alice Johnson', 22, 1);

INSERT INTO Students (Name, Age, CourseID) VALUES ('Bob Smith', 23, 2);

INSERT INTO Students (Name, Age, CourseID) VALUES ('Charlie Brown', 21, 1);

INSERT INTO Courses (CourseName) VALUES ('Database Systems');

INSERT INTO Courses (CourseName) VALUES ('Computer Networks');

-- Insert Students

INSERT INTO Students (Name, Age, CourseID) VALUES ('Alice Johnson', 22, 1);

INSERT INTO Students (Name, Age, CourseID) VALUES ('Bob Smith', 23, 2);

INSERT INTO Students (Name, Age, CourseID) VALUES ('Charlie Brown', 21, 1);

**Step 3: Retrieving Raw Data**

A simple SELECT statement retrieves **raw data**, but it is not meaningful.

// SELECT\*FROM table\_name;

SELECT \* FROM Students;

**output**

A screenshot of a computer

AI-generated content may be incorrect.

**What is SQL**

* SQL (**Structured Query Language**) is a programming language used to **manage, manipulate, and query relational databases**.
* It is used with database management systems like **Oracle, MySQL, PostgreSQL, SQL Server, and SQLite**.

**Types of SQL Commands**

* DDL (Data Definition Language) – Defines database structure.
* DML (Data Manipulation Language) – Manages table data.
* DQL (Data Query Language) – Retrieves data.
* DCL (Data Control Language) – Controls access.
* TCL (Transaction Control Language) – Manages transactions.

**DDL (Data Definition Language)**

* DDL commands are used to **define and modify database structures**, such as tables, indexes, and schemas.

**Common DDL Commands:**

**A screenshot of a computer

AI-generated content may be incorrect.**

1. **Creating a Table (CREATE) //** CREATE TABLE table \_name (column1 datatype constraints, column2 datatype constraints, column3 datatype constraints…..);

CREATE TABLE Studentss (

StudentID INT PRIMARY KEY,

Name VARCHAR2(100),

Age INT

);

1. **Modifying a Table (ALTER)**

The ALTER TABLE statement is used to modify an existing table structure. You can:

* Add, modify, or drop columns
* Rename columns or tables
* Add or remove constraints

**ADD a New Column //** ALTER TABLE table\_name ADD column\_name data\_type [constraint];

ALTER TABLE Students A

DD Email VARCHAR2(150); // Adds a new column Email to the Students table.

**MODIFY an Existing Column //** ALTER TABLE table\_name

MODIFY column\_name new\_data\_type [constraint];

ALTER TABLE Students MODIFY Age NUMBER (3); // Changes the Age column to allow numbers up to 3 digits.

**RENAME a Column //** ALTER TABLE table\_name

RENAME COLUMN old\_column\_name TO new\_column\_name;

ALTER TABLE Students

RENAME COLUMN Name TO FullName; // Renames Name to FullName.

**RENAME a Table //** ALTER TABLE old\_table\_name

RENAME TO new\_table\_name;

ALTER TABLE Students

RENAME TO StudentRecords; // Renames Students table to StudentRecords.

**DROP (Delete) a Column //** ALTER TABLE table\_name

DROP COLUMN column\_name;

ALTER TABLE Students

DROP COLUMN Email; // Removes the Email column from Students.

1. **Deleting a Table (DROP)**

The DROP TABLE statement is used to permanently delete a table along with all its data, structure, and associated constraints. This action cannot be rolled back, so use it with caution. // DROP TABLE table\_name;

DROP TABLE Students;

1. **Deleting All Data Without Removing Structure (TRUNCATE)**

The TRUNCATE TABLE command is used to remove all records from a table while keeping the table structure intact. It is faster than DELETE and cannot be rolled back.

// TRUNCATE TABLE table\_name;

TRUNCATE TABLE Students;

**PROGRAM 1**

**1. Execute DDL statements**

1. Create a table Student with fields (RollNo, Name, Course, Year).
2. Alter table.
3. Drop table.
4. Truncate table.

Write necessary query statements.

# **To Create table**: create table stud(Rollno int primary key,Name varchar(60),Course varchar(50),year int);

# **To insert data:**

insert into stud values(1,'Ammu','MCA',2025);

insert into stud values(2,'Anu','MCA',2025);

insert into stud values(3,'abhi','MCA',2025);

insert into stud values(4,'Milan','MCA',2025);

insert into stud values(5,'Dileesh','MCA',2025);

# **To display table:**

SELECT \* FROM stud;

# **To Truncate table:**

truncate table stud;

#**To drop table:**

drop table stud;

**output**

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

**Data Manipulation Language (DML)**

Data Manipulation Language (DML) consists of SQL commands used to insert, update, delete, and retrieve data in a database. These commands allow users to manipulate data stored in tables.

**Common DML Commands**

**A screenshot of a computer

AI-generated content may be incorrect.**

**INSERT – Add New Records //** INSERT INTO table\_name (column1, column2, ...) VALUES (value1, value2, ...);

INSERT INTO Students (StudentID, Name, Age) VALUES (1, 'Alice Johnson',22);

**//** Inserts a new student record into the students table.

**UPDATE – Modify Existing Records //** UPDATE table\_name SET column1 = value1, column2 = value2 WHERE condition;

UPDATE Students SET Age = 23 WHERE StudentID = 1; **//** Updates Alice's age to 23.

**DELETE – Remove Specific Records //** DELETE FROM table\_name WHERE condition;

DELETE FROM Students WHERE StudentID = 1;// Deletes the record where StudentID = 1.

**2. Execute DML statements**

1. Create table Employee (EmployeeId, Name, Department, salary)
2. Also create another table Department (DepartmentId, DeptName, HeadOfDepartment)
3. Insert a minimum of 5 rows.
4. Set Primary Key and Foreign Key constraints.
5. Display the records.
6. Update a record.
7. Delete a record.

**#** **To create table**

create table emp2(empid int,name varchar(50),dept varchar(20),sal int,deptid int);

create table dept(deptid int,deptname varchar(30),hod varchar(50));

**#** **To alter table**

alter table dept add primary key(deptid);

alter table dept add constraint emp2 foreign key (deptid) references dept(deptid);

**#To insert data**

insert into dept values(101,'HR','Sumit Singh');

insert into dept values(102,'SALES','DIGVIJAY');

insert into dept values(103,'PROGRAMMING','AJIN');

insert into dept values(104,'ANALATYCS','MALAVAZHI SREENIVASAN');

insert into dept values(105,'ADVERTISEMENT','MANIDEEP');

insert into emp2 values(1,'ANU','HR',40000,101);

insert into emp2 values(1,'AMMU','HR',40000,101);

insert into emp2 values(1,'ABHI','PROGRAMMING',45000,103);

insert into emp2 values(4,'MILAN','PROGRAMMING',45000,103);

insert into emp2 values(5,'DILESSH','PROGRAMMING',45000,103);

**#To update table**

update emp2 set empid=2 where name='AMMU';

update emp2 set empid=3 where name='Abhi';

update emp2 set empid=3 where name='ABHI';

**#To display table**

select \*from dept;

select \*from emp2;

**# To delete elements**

delete from dept where deptid=105;

select \*from dept;

**#To save changes**

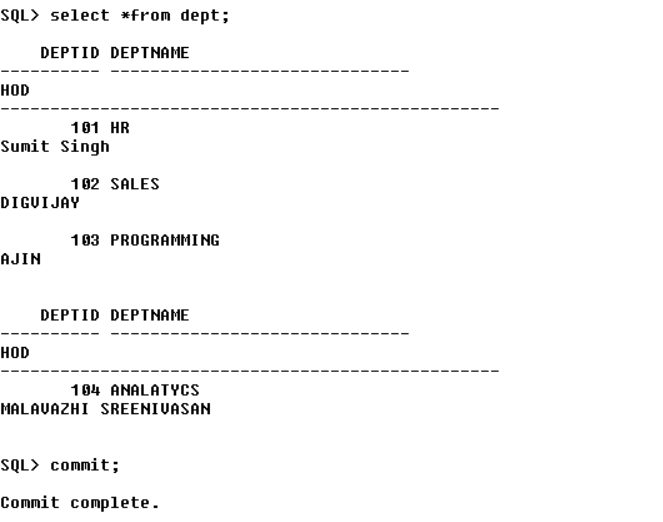
commit;

**output**

**A screenshot of a computer program

AI-generated content may be incorrect.A screenshot of a computer program

AI-generated content may be incorrect.**

A screenshot of a computer

AI-generated content may be incorrect.

**DQL (Data Query Language)**

DQL is used for retrieving data from the database.

**Common DQL Commands**

A screenshot of a computer

AI-generated content may be incorrect.

**SELECT Statement – Retrieve Data**

The SELECT statement is used to fetch data from one or more tables.

SELECT StudentID, Name, Age FROM Students; // Retrieves the StudentID, Name, and Age columns from the Students table.

**Retrieving All Data (SELECT \*) //** Instead of specifying individual columns, use \* to fetch all columns.

SELECT \* FROM Students; // Retrieves all columns from the Students table.

**Filtering Data (WHERE)** // The WHERE clause is used to filter records based on conditions.

SELECT \* FROM Students WHERE Age > 21; // Retrieves students whose age is greater than 21.

**Sorting Data (ORDER BY) //** The ORDER BY clause is used to sort query results in ascending (ASC) or descending (DESC) order.

//SELECT column1, column2 FROM table\_name ORDER BY column\_name ASC|DESC;

SELECT \* FROM Students ORDER BY Age DESC; // Retrieves student names and ages, sorted from oldest to youngest.

**Using DISTINCT to Remove Duplicates //** The DISTINCT keyword removes duplicate values from the result set.

SELECT DISTINCT Age FROM Students; // Retrieves unique ages from the Students table.

**Using GROUP BY for Aggregation //** The GROUP BY clause is used with aggregate functions (COUNT, SUM, AVG, MIN, MAX) to group data**.**

SELECT Age, COUNT(\*) AS TotalStudents FROM Students GROUP BY Age; // Counts the number of students for each unique age.

**Data Control Language (DML)**

**Data Control Language (DCL)** consists of SQL commands that control access to data within a database. It primarily deals with **permissions and privileges** to ensure database security.

**GRANT** – Gives specific privileges to users or roles.

**REVOKE** – Removes specific privileges from users or roles.

GRANT privilege\_name ON table\_name TO user\_name;

GRANT SELECT, INSERT ON Employees TO John; /// This grants John permission to SELECT and INSERT records in the Employees table.

REVOKE privilege\_name ON table\_name FROM user\_name;

REVOKE INSERT ON Employees FROM John;// This removes INSERT permission from John but keeps other privileges intact.

**3. Create a table and execute DCL statements (grant, revoke).**

**Transactional Control Language (TCL)**

* TCL (Transactional Control Language) commands are used to manage transactions in a database.
* TCL (Transactional Control Language) is a part of SQL used to control database transactions.
* A **transaction** is a group of operations that should either be **completely executed** or **completely undone** to maintain data accuracy.

For example, imagine transferring money from one bank account to another:

1. **Withdraw $100 from Account A**
2. **Deposit $100 into Account B**

If the system crashes after step 1 but before step 2, the money disappears! TCL commands **prevent such problems** by ensuring that either both steps complete successfully or none at all.

**Common TCL Commands:**

1. **COMMIT**
2. **ROLLBACK**
3. **SAVEPOINT**

**COMMIT (Save Changes)**

* When you insert, update, or delete records, the changes are not saved until you use COMMIT.
* Once committed, changes cannot be undone.

INSERT INTO students (id, name) VALUES (1, 'John'); //Insert data

COMMIT; //Save changes permanently

**ROLLBACK (Undo Changes)**

* If something goes wrong before COMMIT, we can undo the changes using ROLLBACK.

INSERT INTO students (id, name) VALUES (2, 'Jane'); // Insert data

ROLLBACK; // Undo the insertion

**SAVEPOINT (Partial Undo)**

SAVEPOINT creates a checkpoint in a transaction. We can ROLLBACK to a savepoint instead of undoing everything.

INSERT INTO students (id, name) VALUES (3, 'Alice');

SAVEPOINT sp1; //Savepoint created

INSERT INTO students (id, name) VALUES (4, 'Bob');

SAVEPOINT sp2; //Another savepoint

ROLLBACK TO sp1; //Undo Bob's insert, but keep Alice's

COMMIT; // Save Alice's data99

**4**.**Create a table and execute TCL statements (commit, rollback, savepoint).**

SQL> create table student22 (rollno int,name varchar(20),course varchar(20));

Table created.

SQL> insert into student22 values(1,'Aiswarya','MCA');

1 row created.

SQL> insert into student22 values(2,'Alen','MCA');

1 row created.

SQL> savepoint a;

Savepoint created.

SQL> select \* from student22;

ROLLNO NAME COURSE

---------- -------------------- --------------------

1 Aiswarya MCA

2 Alen MCA

SQL> insert into student22 values(3,'Anakha','MCA');

1 row created.

SQL> insert into student22 values(4,'Aneeta','MCA');

1 row created.

SQL> select \* from student22;

ROLLNO NAME COURSE

---------- -------------------- --------------------

1 Aiswarya MCA

2 Alen MCA

3 Anakha MCA

4 Aneeta MCA

SQL> rollback to a;

Rollback complete.

SQL> select \* from student22;

ROLLNO NAME COURSE

---------- -------------------- --------------------

1 Aiswarya MCA

2 Alen MCA

SQL> commit;

Commit complete.

SQL> insert into student22 values(5,'Anit','MCA');

1 row created.

SQL> select \* from student22;

ROLLNO NAME COURSE

---------- -------------------- --------------------

1 Aiswarya MCA

2 Alen MCA

5 Anit MCA

SQL> rollback;

Rollback complete.

SQL> select \* from student22;

ROLLNO NAME COURSE

---------- -------------------- --------------------

1 Aiswarya MCA

* 1. Alen MCA

**5. Consider the CUSTOMERS table having the following records**

**A screenshot of a table

AI-generated content may be incorrect.**

1. **Display details of Customers whose ID is 4. (Simple select)**

SELECT \* FROM CUSTOMERS WHERE ID = 4; // This query returns **all details** of the customer with ID = 4.

1. **Display distinct hometowns of customers. (DISTINCT)**

SELECT DISTINCT hometown FROM CUSTOMERS; //

1. SELECT DISTINCT hometown → DISTINCT ensures that only unique (non-duplicate) hometown values are returned.
2. FROM CUSTOMERS → Specifies the table from which data is retrieved.

This query lists all unique hometowns of customers, ensuring that duplicate hometowns appear only once.

1. Display the total number of Salary of customers. (COUNT)

SELECT COUNT(Salary) FROM CUSTOMERS; //.. This query returns the total number of customers who have a salary recorded

**QUERY**

SQL> select \* from customer where id=4;

ID NAME AGE HOMETOWN SALARY

---------- -------------------- ---------- -------------------- ----------

4 Chaitali 25 Mumbai 6500

SQL> select distinct hometown from customer;

HOMETOWN

--------------------

Bhopal

Ahmedabad

Delhi

Kota

Mumbai

MP

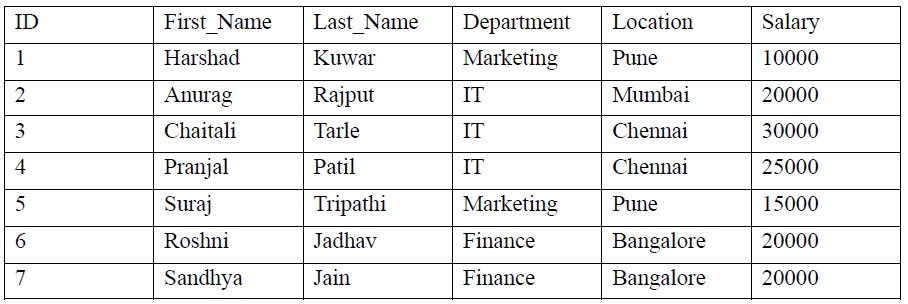
Indore

7 rows selected.

SQL> select count(salary) from customer;

COUNT(SALARY)\ || 10

**6**. **Consider we have an employee table created in the database with the following data:**



1. Display details of employees from employee tables in which the department of the employee is IT and the location is Chennai. (Where, And)

SELECT \* FROM employee

WHERE Department = 'IT' AND Location = 'Chennai';

* SELECT \* FROM employee → Selects all columns (\* means all columns) from the employee table.
* WHERE Department = 'IT' → Filters rows where the Department column is "IT".
* AND Location = 'Chennai' → Further filters rows where the Location column is "Chennai".
* Only employees who work in the IT department **AND** are located in Chennai will be displayed.

1. Display details of employees from employee tables in which the department of the employee is IT or the location is Chennai. (Where, OR)

SELECT \* FROM employee

WHERE Department = 'IT' OR Location = 'Chennai';

* SELECT \* FROM employee → Selects all columns from the employee table.
* WHERE Department = 'IT' → Filters rows where the Department column is "IT".
* OR Location = 'Chennai' → Filters rows where the Location column is "Chennai".
* This will return employees who are **either** in the IT department **OR** located in Chennai (or both).

1. Display First\_Name and Location number under the temporary column names 'EMP FN' and ‘Address’, respectively. (AS)

**Sql Aliases**

* It is used to give a table , or a coloumn in a table a temporary name.
* Aliases are often used to make coloumn names more readable
* An Aliases can only exist for the dyaration of that Query
* An Aliases is created with the “**As**” Keyword

SELECT First\_Name AS "EMP FN", Location AS "Address" FROM employee;

* SELECT First\_Name AS "EMP FN" → Selects the First\_Name column but displays it with a new temporary name (alias) **"EMP FN"**.
* Location AS "Address" → Selects the Location column but renames it as **"Address"**.
* FROM employee → Fetches the data from the employee table.
* Instead of showing column names as First\_Name and Location, it will show EMP FN and Address.

1. List records of employees whose names start with “S”.(Like)

**Sql Like Operator**

* + It is used in a where clause to search for a specified pattern in a coloumn.
  + A wild Character is used to substitute one or more character in a string
  + There are two wild cards often used in conjection with like operator
  + % represents 0,1 or Multiple.
  + Represent 1 , single character.

SELECT \* FROM employee WHERE First\_Name LIKE 'S%';

* SELECT \* FROM employee → Selects all columns from the employee table.
* WHERE First\_Name LIKE 'S%':
* LIKE is used for pattern matching.
* 'S%' means the First\_Name should start with **"S"** (% represents any number of characters after "S").
* This will return all employees whose First\_Name starts with the letter "S" (e.g., Suraj, Sandhya).

1. Add the salary of employees for each city. (Groupby) Show that Location whose total salary of employees is more than 10000. (Having)

SELECT Location, SUM(Salary) AS Total\_Salary

FROM employee

GROUP BY Location

HAVING SUM(Salary) > 10000;

1. SELECT Location, SUM(Salary) AS Total\_Salary:
   * Selects the Location column.
   * Calculates the **total salary** for each location using SUM(Salary).
   * Displays the result in a new column named **Total\_Salary**.
2. FROM employee → Fetches data from the employee table.
3. GROUP BY Location:
   * Groups the employees by their Location (e.g., Pune, Mumbai, Bangalore, Chennai).
   * SUM(Salary) is calculated separately for each group (location).
4. HAVING SUM(Salary) > 10000:
   * **HAVING** is like WHERE but works on grouped data.
   * Filters out locations where the total salary is **greater than 10,000**.
5. Displays only those locations where the sum of all employee salaries is **more than 10,000**.
6. Sort the records in the ascending order of the Employee names stored in the employee table.(Orderby)

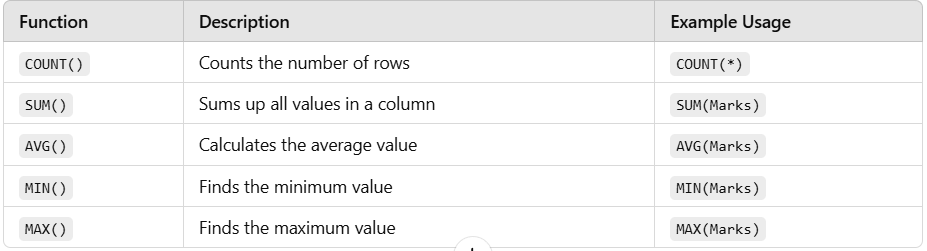
SELECT \* FROM employee

ORDER BY First\_Name ASC;

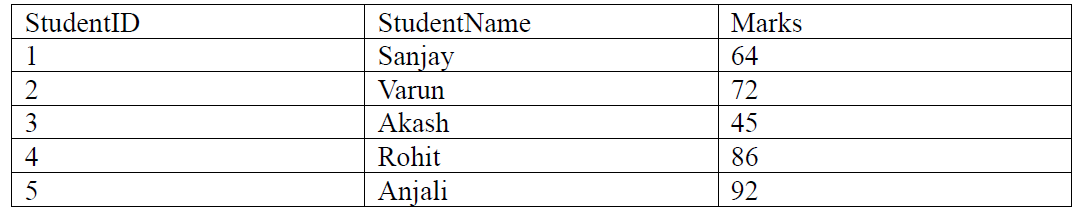
* SELECT \* FROM employee → Selects all columns from the employee table.
* ORDER BY First\_Name:
* Orders the results based on the First\_Name column.
* ASC (Ascending order):
* This means names will be sorted **A → Z** (e.g., Anurag, Chaitali, Harshad, Pranjal, Roshmi, Sandhya, Suraj).
* Displays all employees in alphabetical order based on First\_Name.

**7.Aggregate functions**

**Aggregate functions** in SQL perform calculations on multiple rows of data and return a single value.



A close-up of a box

AI-generated content may be incorrect.

1. Write a query to count the number of students scoring marks > 75 from the Students table.
2. Write a query to calculate the average marks of all students from the Students table.
3. Write a query to retrieve the minimum marks out of all students from the Students table.
4. Write a query to retrieve the maximum marks out of all students from the Students table.

**Count the number of students scoring marks > 75**

SELECT COUNT(\*) AS StudentsAbove75

FROM Students

WHERE Marks > 75;

**SELECT COUNT(\*)** → The COUNT(\*) function counts the total number of rows that satisfy the condition.

**AS StudentsAbove75** → This renames the column in the output for better readability.

**FROM Students** → Fetch data from the table named Students.

**WHERE Marks > 75** → Filters only those rows where Marks is greater than 75.

**Calculate the average marks of all students**

SELECT AVG(Marks) AS AverageMarks

FROM Students;

**SELECT AVG(Marks)** → The AVG() function calculates the average of the Marks column.

**AS AverageMarks** → Renames the output column as AverageMarks for clarity.

**FROM Students** → Fetch data from the Students table.

**Retrieve the minimum marks from all students**

SELECT MIN(Marks) AS MinimumMarks

FROM Students;

**SELECT MIN(Marks)** → The MIN() function finds the lowest value in the Marks column.

**AS MinimumMarks** → Renames the output column as MinimumMarks.

**FROM Students** → Fetch data from the Students table.

**Retrieve the maximum marks from all students**

SELECT MAX(Marks) AS MaximumMarks

FROM Students;

**SELECT MAX(Marks)** → The MAX() function finds the highest value in the Marks column.

**AS MaximumMarks** → Renames the output column as MaximumMarks.

**FROM Students** → Fetch data from the Students table.

SQL> create table student22a (student\_id int primary key,student\_name varchar(20

),marks float);

Table created.

SQL> insert into student22a values(1,'Sanjay',64);

1 row created.

SQL> insert into student22a values(2,'Varun',72);

1 row created.

SQL> insert into student22a values(3,'Akash',45);

1 row created.

SQL> insert into student22a values(4,'Rohit',86);

1 row created.

SQL> insert into student22a values(5,'Anjali',92);

1 row created.

SQL> insert into student22a values(6,'Ammu',95);

1 row created.

SQL> insert into student22a values(7,'Rahul',80);

1 row created.

SQL> insert into student22a values(8,'Prathibha',62);

1 row created.

SQL> insert into student22a values(9,'Ravi',75);

1 row created.

SQL> insert into student22a values(10,'Pulkit',76);

1 row created.

SQL> select \* from student22a;

STUDENT\_ID STUDENT\_NAME MARKS

---------- -------------------- ----------

1 Sanjay 64

2 Varun 72

3 Akash 45

4 Rohit 86

5 Anjali 92

6 Ammu 95

7 Rahul 80

8 Prathibha 62

9 Ravi 75

10 Pulkit 76

10 rows selected.

SQL> select count(\*) from student22a where marks>75;

COUNT(\*)

----------

5

SQL> select avg(marks) as average\_marks from student22a;

AVERAGE\_MARKS

-------------

74.7

SQL> select min(marks) as minimum\_marks from student22a;

MINIMUM\_MARKS

-------------

45

SQL> select max(marks) as maximum\_marks from student22a;

MAXIMUM\_MARKS

-------------

95

**What is a Subquery?**

A **subquery** is a query inside another query. It is also called an **inner query** or **nested query** because it is executed first, and its result is used by the outer query.

**Syntax:**

SELECT column\_name(s)

FROM table\_name

WHERE column\_name = (SELECT column\_name FROM another\_table WHERE condition);

**Tables to be used:**

Sailors (sid: integer, sname: string, rating: integer, age: real)

Boats (bid: integer, bname: string, color: string)

Reserves (sid: integer, bid: integer,day: date)

A table of numbers and names

AI-generated content may be incorrect.A table with numbers and words

AI-generated content may be incorrect.

**A screenshot of a computer

AI-generated content may be incorrect.**

**Example 1: Basic Subquery**

**Find the names of sailors who have reserved boat number 101.**

**Tables Given:**

* **Sailors** (sid, sname, rating, age)
* **Reserves** (sid, bid, day)

🔹 **SQL Query using Subquery:**

SELECT sname FROM Sailors WHERE sid IN (SELECT sid FROM Reserves WHERE bid = 101);

Explanation

SELECT sname FROM Sailors WHERE sid IN (...) //outer query

* SELECT sname → Selects the **name** of sailors.
* FROM Sailors → Retrieves data from the **Sailors** table.
* WHERE sid IN (...) → Checks if a sailor’s sid is in the result of the **subquery**.

(SELECT sid FROM Reserves WHERE bid = 101); //subquery

* SELECT sid → Fetches the sid (Sailor ID) from the Reserves table.
* FROM Reserves → Retrieves data from the **Reserves** table.
* WHERE bid = 101 → Filters only those records where bid (Boat ID) is 101.
* This query **returns the sid of sailors who reserved boat 101**.

The inner query finds the sid of sailors who have reserved boat 101.

The outer query fetches the sname of these sailors.

**Example 2: Subquery with DISTINCT**

**Find distinct names of sailors.**

🔹 **SQL Query:**

SELECT DISTINCT sname FROM Sailors;

* DISTINCT removes duplicates, ensuring unique names of sailors.

**Example 3: Subquery in ORDER BY Clause**

**Find the names of sailors who have reserved a red boat, sorted by age.**

🔹 **SQL Query:**

SELECT sname

FROM Sailors

WHERE sid IN (SELECT sid FROM Reserves WHERE bid IN

(SELECT bid FROM Boats WHERE color = 'red'))

ORDER BY age;

**1. Outer Query: Selecting Sailor Names**

SELECT sname FROM Sailors WHERE sid IN (...);

* SELECT sname → Selects **sailor names** from the Sailors table.
* FROM Sailors → Retrieves data from the Sailors table.
* WHERE sid IN (...) → Filters sailors whose sid appears in the subquery results.

**2. First Subquery: Finding Sailors Who Reserved Red Boats**

SELECT sid FROM Reserves WHERE bid IN (...);

* SELECT sid → Fetches **sailor IDs** (sid) from the Reserves table.
* FROM Reserves → Retrieves data from Reserves (which stores reservation records).
* WHERE bid IN (...) → Filters only the reservations of boats with **red color** (using another subquery).

**3. Second (Innermost) Subquery: Finding Red Boats**

SELECT bid FROM Boats WHERE color = 'red';

* SELECT bid → Fetches **boat IDs** (bid) from the Boats table.
* FROM Boats → Retrieves data from the Boats table.
* WHERE color = 'red' → Filters **only red boats**.

**4. ORDER BY Clause**

ORDER BY age;// Sorts the final list of sailors **by age in ascending order**.

1. **Innermost Query:** Finds bid of red boats.
2. **Middle Query:** Finds sid of sailors who reserved these boats.
3. **Outer Query:** Fetches their sname and sorts them by age.

**Example 4: Subquery with GROUP BY**

**Find sailors who have reserved at least one boat.**

🔹 **SQL Query:**

SELECT sname FROM Sailors WHERE sid IN (SELECT DISTINCT sid FROM Reserves);

* The **inner query** fetches distinct sid from Reserves.
* The **outer query** finds corresponding sname.

**Example 5: Subquery for Finding Sailors with Two Reservations on the Same Day**

**Find the IDs and names of sailors who reserved two different boats on the same day.**

🔹 **SQL Query:**

SELECT sid, sname

FROM Sailors

WHERE sid IN (

SELECT sid FROM Reserves

GROUP BY sid, day

HAVING COUNT(DISTINCT bid) >= 2

);

1. **Inner Query:** Groups sid and day, counting unique bid.
2. **HAVING COUNT(DISTINCT bid) >= 2** ensures at least 2 reservations per day.
3. **Outer Query:** Retrieves sname of these sailors.

A **subquery** runs before the outer query and provides intermediate results.  
Used in **WHERE, FROM, HAVING, ORDER BY, and SELECT** clauses.  
Helps retrieve **filtered** or **aggregated** data efficiently.

SQL> create table sailors22 (sid int primary key,sname varchar(20),rating int,ag

e float);

Table created.

SQL> create table boats22 (bid int primary key,bname varchar(20),color varchar(2

0));

Table created.

SQL> create table reserves22 (sid int,bid int,day date,foreign key(bid) referenc

es boats22(bid),foreign key(sid) references sailors22(sid));

Table created.

SQL> insert into sailors22 values(22,'Dustin',7,45.0);

1 row created.

SQL> insert into sailors22 values(29,'Brutus',1,33.0);

1 row created.

SQL> insert into sailors22 values(31,'Lubber',8,55.5);

1 row created.

SQL> insert into sailors22 values(32,'Andy',8,25.5);

1 row created.

SQL> insert into sailors22 values(58,'Rusty',10,35.0);

1 row created.

SQL> insert into sailors22 values(64,'Horatio',7,35.0);

1 row created.

SQL> insert into sailors22 values(71,'Zorba',10,16.0);

1 row created.

SQL> insert into sailors22 values(74,'Horatio',9,35.0);

1 row created.

SQL> insert into sailors22 values(85,'Art',3,25.5);

1 row created.

SQL> insert into sailors22 values(95,'Bob',3,63.5);

1 row created.

SQL> select \* from sailors22;

SID SNAME RATING AGE

---------- -------------------- ---------- ----------

22 Dustin 7 45

29 Brutus 1 33

31 Lubber 8 55.5

32 Andy 8 25.5

58 Rusty 10 35

64 Horatio 7 35

71 Zorba 10 16

74 Horatio 9 35

85 Art 3 25.5

95 Bob 3 63.5

10 rows selected.

SQL> insert into boats22 values (101,'Interlake','blue');

1 row created.

SQL> insert into boats22 values (102,'Interlake','red');

1 row created.

SQL> insert into boats22 values (103,'Clipper','green');

1 row created.

SQL> insert into boats22 values (104,'Marine','red');

1 row created.

SQL> select \* from boats22;

BID BNAME COLOR

---------- -------------------- --------------------

101 Interlake blue

102 Interlake red

103 Clipper green

104 Marine red

SQL> insert into reserves22 values (22,101,'10-Oct-98');

1 row created.

SQL> insert into reserves22 values (22,102,'10-Oct-98');

1 row created.

SQL> insert into reserves22 values (22,103,'10-Aug-98');

1 row created.

SQL> insert into reserves22 values (22,104,'10-Jul-98');

1 row created.

SQL> insert into reserves22 values (31,102,'11-Oct-98');

1 row created.

SQL> insert into reserves22 values (31,103,'11-Jun-98');

1 row created.

SQL> insert into reserves22 values (31,104,'11-Dec-98');

1 row created.

SQL> insert into reserves22 values (64,101,'9-May-98');

1 row created.

SQL> insert into reserves22 values (64,102,'9-Aug-98');

1 row created.

SQL> insert into reserves22 values (74,103,'9-Aug-98');

1 row created.

SQL> select \* from reserves22;

SID BID DAY

---------- ---------- ---------

22 101 10-OCT-98

22 102 10-OCT-98

22 103 10-AUG-98

22 104 10-JUL-98

31 102 11-OCT-98

31 103 11-JUN-98

31 104 11-DEC-98

64 101 09-MAY-98

64 102 09-AUG-98

74 103 09-AUG-98

10 rows selected.

SQL> select distinct(sname) from sailors22;

SNAME

--------------------

Rusty

Lubber

Brutus

Andy

Art

Bob

Dustin

Zorba

Horatio

9 rows selected.

SQL> select \* from sailors22 where sid in (select sid from reserves22 where bid=

101);

SID SNAME RATING AGE

---------- -------------------- ---------- ----------

22 Dustin 7 45

64 Horatio 7 35

SQL> select sname from sailors22 where sid in (select sid from reserves22);

SNAME

--------------------

Dustin

Lubber

Horatio

Horatio

SQL> select sname,age from sailors22 where sid in (select sid from reserves22,bo

ats22 where reserves22.bid=boats22.bid and color='red') order by age;

SNAME AGE

-------------------- ----------

Horatio 35

Dustin 45

Lubber 55.5

SQL>

SQL> select distinct s.sid,s.sname from sailors22 s,reserves22 r1,reserves r2 wh

ere s.sid=r1.sid and s.sid=r2.sid and r1.day=r2.day and r1.bid<>r2.bid;

SID SNAME

---------- --------------------

22 Dustin

**9. Union, Intersect and Except**

Using the same table

1. Find the ids of sailors who have reserved a red boat or a green boat.
2. Find the names of sailors who have reserved boat 103.
3. Find the name and the age of the youngest sailor.
4. Find the names of sailors who have reserved all boats.

**UNION**

* UNION is used to combine the results of two SELECT statements.
* It returns distinct values — meaning duplicates will be removed.
* Both SELECT queries must have the same number of columns and the same data types in the same order.
  1. **SELECT sid FROM reserves22, boats22 WHERE reserves22.bid = boats22.bid AND color = 'red' UNION SELECT sid FROM reserves22, boats22 WHERE reserves22.bid = boats22.bid AND color = 'green';**

Here in the above question, we need two tables:

1. reserves22: tells you which sailor reserved which boat.
2. boats22: gives you the details of each boat like color. , So, to know **which sailor reserved a red or green boat**, we need to
3. Join the two tables on bid (boat ID).
4. Filter for color = 'red' or color = 'green'.
5. Get the sid (sailor ID).

First SELECT: Red **boats**

SELECT sid FROM reserves22, boats22 WHERE reserves22.bid = boats22.bid AND color = 'red' /// it gives you all sailors (sid) who reserved **red boats**.

Second SELECT: Green boats

SELECT sid FROM reserves22, boats22 WHERE reserves22.bid = boats22.bid AND color = 'green' /// it gives you all sailors who reserved **green boats.**

**INTERSECT**

* INTERSECT is used to get only the common rows between two SELECT statements**.**
* It **removes duplicates** automatically.
* Both SELECT queries must have the **same number of columns** and same data types.

**2.SELECT sname FROM sailors22 INTERSECT SELECT sname FROM sailors22, reserves22 WHERE sailors22.sid = reserves22.sid AND bid = 103;**

SELECT sname FROM sailors22 **//** it gives you **all sailor names** in the sailors22 table.

SELECT sname FROM sailors22, reserves22 WHERE sailors22.sid = reserves22.sid AND bid = 103; **//** it does a join between sailors and reservations, and filters only where bid = 103 (boat ID 103). So, It shows the **names of sailors who reserved boat 103**.

**3. SELECT sname, age FROM sailors22 WHERE age IN (SELECT MIN(age) FROM sailors22);**

SELECT MIN(age) FROM sailors22; //

* This is a **subquery**, which means it runs **inside** the main query.
* MIN(age) finds the **smallest (minimum) age** in the table — i.e., the **youngest sailor's age**.

SELECT sname, age FROM sailors22 WHERE age IN (22);

It give name and age of **all sailors** whose age is the **minimum age** found

**4. select s.sname from sailors22 s where not exists ((select b.bid from boats22 b) minus (select r.bid from reserves22 r where r.sid=s.sid));**

SELECT b.bid FROM boats22 b /// lists all boat IDs.

SELECT r.bid FROM reserves22 r WHERE r.sid = s.sid/// This gives all boats that **sailor s has reserved**.

MINUS returns the **boats that are in the first list but not in the second list**.

10. **10. Aggregation Operators**

Using the same table

1. Find the name and the age of the youngest sailor.
2. Find the average age of sailors for each rating level.
3. Count the number of different sailor names.
4. Calculate the average age of all sailors.
5. SQL> select sname,age from sailors22 where age in (select min(age) from sailors22);

SNAME AGE

-------------------- ----------

Zorba 16

1. SQL> select rating,avg(age) as avg\_age from sailors22 group by rating;

RATING AVG\_AGE

---------- ----------

1 33

8 40.5

7 40

3 44.5

10 25.5

9 35

6 rows selected.

1. SQL> select count(distinct sname) from sailors22;

COUNT(DISTINCTSNAME)

--------------------

9

1. SQL> select avg(age) as avg\_age from sailors22;

AVG\_AGE

----------

36.9

**PL/SQL**

* PL/SQL = **Procedural Language + SQL**
* Developed by **Oracle** to add **programming constructs** to SQL (like loops, if statements).
* Used to **write procedures, functions, packages, and triggers**.

Basic Structure of PL/SQL Block

DECLARE

-- Variable declaration

BEGIN

-- Executable statements

EXCEPTION

-- Exception handling

END;

In PL/SQL, a program is structured as a **block** of code, but not all parts of the block are mandatory. A standard PL/SQL block is made up of three main sections:

DECLARE

-- Variable declarations (optional)

BEGIN

-- Executable statements (mandatory)

EXCEPTION

-- Error handling (optional)

END;

/

**1. DECLARE Block (Optional)**

This section is used to declare variables, constants, cursors, and user-defined types. If your code does not require any variable declarations, this section can be **omitted**.

Example without DECLARE:

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Hello, World!');

END;

**2. BEGIN...END Block (Mandatory)**

This is the **core** part of any PL/SQL block. It contains the actual executable code such as assignments, queries, and procedure calls. The BEGIN...END block is **required** in all PL/SQL programs.

**3. EXCEPTION Block (Optional)**

The EXCEPTION section is used to handle runtime errors such as division by zero or data not found. It is **not mandatory**, but it is highly recommended in production code for better error handling.

Example without EXCEPTION:

DECLARE

name VARCHAR2(50);

BEGIN

name := 'Farisamol';

DBMS\_OUTPUT.PUT\_LINE('Hello, ' || name);

END;

**Hello World Example**

**SET SERVEROUTPUT ON;**

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Hello, PL/SQL!');

END;

**Declaring Variables**

DECLARE

v\_name VARCHAR2(20);

v\_age NUMBER := 25;

BEGIN

v\_name := 'Farisamol';

DBMS\_OUTPUT.PUT\_LINE('Name: ' || v\_name || ', Age: ' || v\_age);

END;

**Trigger**

A **trigger** is like an **automatic action** that happens in a **database** when something specific happens to a table like inserting, updating, or deleting data.

( Imagine you have a **smart light** in your room that **automatically turns on** when you enter the room. In the same way, a **trigger in a database** automatically runs when a specific event (like INSERT) happens.).

**Basic Syntax**

CREATE TRIGGER trigger\_name

AFTER INSERT ON table\_name

FOR EACH ROW

BEGIN

-- Your SQL code here

END;

we can use a **trigger** to automatically **log** whenever someone is inserted into the student’s table.( Suppose we have a table called students and we want to log whenever a new student is added).

Example:

Step 1: Main Table

CREATE TABLE students ( id INT, name VARCHAR(50), age INT, course VARCHAR(50));

INSERT INTO Students VALUES (1, ‘Asha’ ,20, ‘B.Sc’);

INSERT INTO Students VALUES (2, ‘Ravi’ ,21, ‘B.Com’);

| **ID** | **Name** | **Age** | **Course** |
| --- | --- | --- | --- |
| 1 | Asha | 20 | B.Sc |
| 2 | Ravi | 21 | B.Com |

Now, we want to **keep a record (log)** every time a **new student is added** to the table

(Log :To **record an event** — like writing it into another table or printing a message.)

Step 2: Log Table

CREATE TABLE student\_log ( log\_time TIMESTAMP, student\_name VARCHAR(50),

message VARCHAR(100));

Step 3: Trigger to log on insert

CREATE OR REPLACE TRIGGER log\_student\_insert

AFTER INSERT ON students

FOR EACH ROW

BEGIN

INSERT INTO student\_log (log\_time, student\_name, message) VALUES (SYSDATE, :NEW.name, 'New student added');

END;

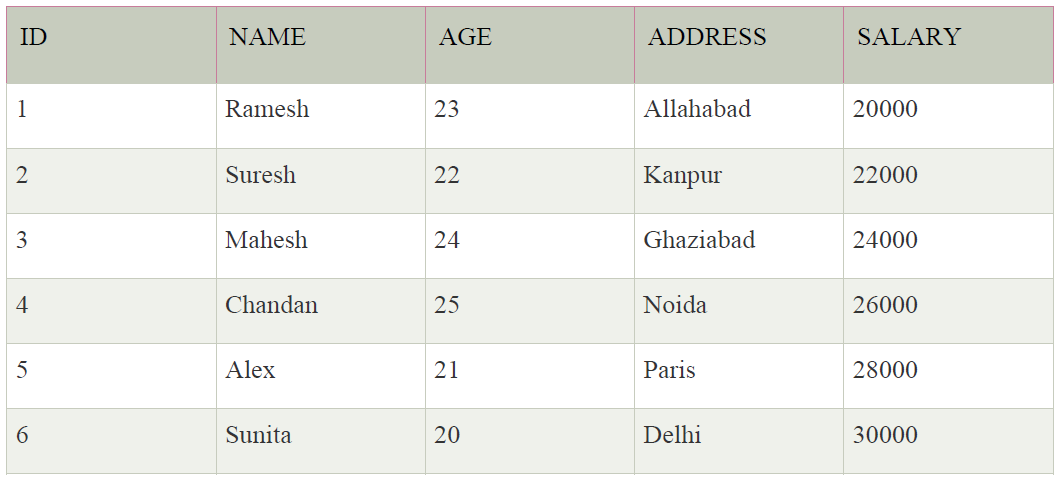
* Every time you run:

INSERT INTO students VALUES (3, 'Riya', 19, 'BBA');

* The trigger **automatically** adds this to student\_log:

| **log\_time** | **student\_name** | **message** |
| --- | --- | --- |
| 21-MAY-2025 11:00 | Riya | New student added |
| **What is SYSDATE in SQL?**  **SYSDATE** is a **built-in function** in SQL (especially in Oracle SQL) that returns the **current date and time** from the **system's clock** — that is, the **date and time right now** when the SQL statement runs. |  |  |

Program 17

17. Execution of trigger.

SQL> create table customers(Id int, Name varchar(20),Age int, Address varchar(20), Salary int);

(Table created.)

SQL> insert into customers values(1, 'Ramesh', 23, 'Allahabad', 20000);

SQL> insert into customers values(2,'Suresh',22,'Kanpur',22000);

SQL> insert into customers values(3,'Mahesh',24,'Ghaziabad',24000);

SQL> insert into customers values(4,'Chandan',25,'Noida',26000);

SQL> insert into customers values(5,'Alex',21,'Paris',28000);

SQL> insert into customers values(6,'Sunita',20,'Delhi',30000);

(Values Inserted)

To Create Trigger

SQL> SET SERVEROUTPUT ON;

SQL> CREATE OR REPLACE TRIGGER display\_salary\_changes

2

3 BEFORE DELETE OR INSERT OR UPDATE ON customers

4

5 FOR EACH ROW

6

7 WHEN (NEW.ID > 0)

8

9 DECLARE

10

11 sal\_diff number;

12

13 BEGIN

14

15 sal\_diff := :NEW.salary - :OLD.salary;

16

17 dbms\_output.put\_line('Old salary: ' || :OLD.salary);

18

19 dbms\_output.put\_line('New salary: ' || :NEW.salary);

20

21 dbms\_output.put\_line('Salary difference: ' || sal\_diff);

22

23 END;

24

25 /

To Display Salary Change,

SQL> DECLARE

2

3 total\_rows number(2); // You declare a variable named total\_rows.

it can store a number up to 2 digits (like 99).

This will store how many rows were updated.

4

5 BEGIN

6

7 UPDATE customers

8

9 SET salary = salary + 5000;

10

11 IF sql%notfound THEN

12

13 dbms\_output.put\_line('no customers updated');

14

15 ELSIF sql%found THEN

16

17 total\_rows := sql%rowcount;

18

19 dbms\_output.put\_line( total\_rows || ' customers updated ');

20

21 END IF;

22

23 END;

24

25 /

**Cursor**

A cursor in DBMS is like a pointer that allows you to go through the rows of a query result one by one.

Imagine you ran a SQL query, and it gave you a list of 10 student names.

Now, if you want to look at each student one at a time, maybe to check something or perform an action, a cursor helps you do exactly that.

Think of it like reading a book. You don’t read all the pages at once – you go one page at a time. A cursor lets you do this with data in a database.

Normally, SQL works with **sets** of data – it processes many rows at once. But sometimes, you need to:

* Process **each row individually**
* Perform **custom operations** for each row (like updating based on conditions)
* Use **loops** to go through the data step-by-step.

**Steps Involved in Using a Cursor:**

* Declare the Cursor

You define a cursor and write the SQL query it will use.

Example: “Give me all students who scored above 80.”

* Open the Cursor

This runs the SQL query and stores the result in the cursor.

* Fetch from the Cursor

One row at a time, you retrieve data from the result and process it.

Think: Fetch one row → Do something → Fetch next row → Repeat.

* Close the Cursor

Once all rows are processed, you close the cursor to free up memory.

**Types of Cursors:**

* **Implicit Cursor:**  
  Automatically used by the system when you run simple SQL commands (like SELECT INTO, INSERT, UPDATE, DELETE). You don’t have to write any cursor code.
* **Explicit Cursor:**  
  We need to write the code to declare, open, fetch, and close the cursor. This is used when you want more control over the row-by-row process.

DECLARE

c\_id customers.id%type;

c\_name customers.name%type;

c\_addr customers.address%type;

CURSOR c\_customers IS

SELECT id, name, address FROM customers;

BEGIN

OPEN c\_customers;

LOOP

FETCH c\_customers INTO c\_id, c\_name, c\_addr;

EXIT WHEN c\_customers%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(c\_id || ' ' || c\_name || ' ' || c\_addr);

END LOOP;

CLOSE c\_customers;

END;

/