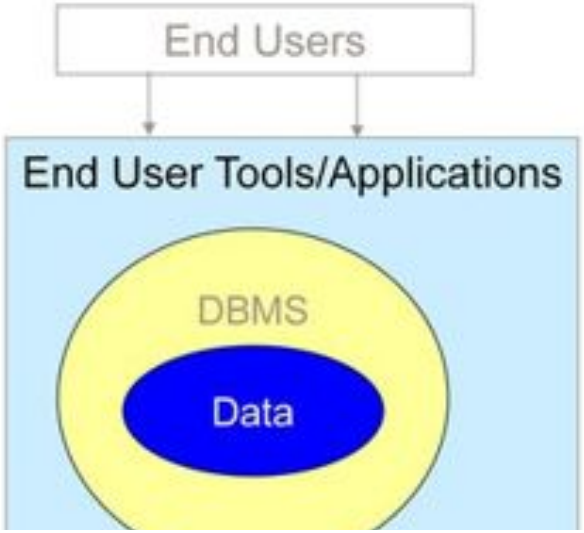
What is Data?

- Data (plural of the word datum) is a factual information used as a basis for reasoning, discussion, or calculation
- Data may be numerical data which may be integers or floating point numbers, and non-numerical data such as characters, date etc.
- Data by itself normally doesn't have a meaning associated with it.
- e.g:- Jack
  01-jan-71
  15-jun-05
  50000

1.1: Introduction to Database

Defining Database, DBMS & Schema

- Database: It is a set of inter-related data
- DBMS: It is a software that manages the data
- Schema: It is a set of structures and relationships, which meet a specific need



Introduction to Database:

- Alogically coherent collection of related data ("information") with inherent meaning, built for a certain application, and representing a subset of the "real-world". For eg: Customer database in bank, Employee Details The software that
- manages the database is known as "Database Management System" or "DBMS". Hence DBMS can be described as "a computer-based record keeping system which consists of software for processing a collection of interrelated data". The general purpose of a DBMS is to provide for the definition, storage, and management of data in a centralized area that can be shared by many users A set of structures and relationships that meet a specific need is called as a "schema".
-

1.2: Features of DBMS

Characteristics of DBMS

- Given below are the characteristics of DBMS:
  - Control of Data Redundancy
    - Traditionally, same data is stored in a number of places
    - Gives rise to data redundancy and its disadvantages
    - DBMS helps in removing data redundancies by providing means of data- integration.
  - Sharing of Data
    - DBMS allows many applications to share the data.
  - Maintenance of Integrity
    - DBMS maintains the correctness, consistency, and interrelationship of data with respect to the application, which uses the data.

1.3: The Data Models

Relational Model

- The Relational model:
  - The Relational model developed out of the work done by Dr. E. F. Codd at IBM in the late 1960s. He was looking for ways to solve the problems with the existing models.
  - At the core of the Relational model is the concept of a “table” (also called a “relation”), which stores all data.
  - Each “table” is made up of:
    - “records” (i.e. horizontal rows that are also known as “tuples”), and
    - “fields” (i.e. vertical columns that are also known as “attributes”)

Relational Model

- The Relational model:
  - Examples of RDBMS:
    - Oracle
    - Informix
    - Sybase
  - Because of lack of linkages, the Relational model is easier to understand and implement.

Student Table	
Scode	Sname
S1	A
S2	B

Course Table	
Ccode	Cname
C1	Physics
C2	Chemistry
C3	Maths
C4	Biology

Marks Table		
Ccode	Scode	Marks
C1	S1	65
C2	S1	78
C3	S1	83
C4	S1	85
C3	S2	83
C4	S2	85

DML Operations

1.3: The Data Models

Relational Model - Possibilities

- Possibilities in a Relational model:
  - INSERT
    - Inserting a “course record” or “student record” poses no problems because tables are separate.
  - UPDATE
    - Update can be done only to a particular table.
  - DELETE
    - Deleting any record affects only a particular table.

1.4: Relational DBMS

Relational Tables - Properties

- Properties of Relational Data Entities:
  - Tables must satisfy the following properties to be classified as relational:
    - Entries of attributes should be single-valued.
    - Entries of attributes should be of the same kind.
    - No two rows should be identical.
    - The order of attributes is unimportant.
    - The order of rows is unimportant.
    - Every column can be uniquely identified.

1.4: Relational DBMS

Data Integrity

- "Data Integrity" is the assurance that data is consistent, correct, and accessible throughout the database.
- Some of the important types of integrities are:
  - Entity Integrity:
    - It ensures that no "records" are duplicated, and that no "attributes" that make up the primary key are NULL.
    - It is one of the properties that is necessary to ensure the consistency of the database.

1.4: Relational DBMS

Data Integrity

- Foreign Key and Referential Integrity
  - The Referential Integrity rule: If a Foreign key in table A refers to the Primary key in table B, then every value of the Foreign key in table A must be null or must be available in table B.
- Unique Constraint:
  - It is a single field or combination of fields that uniquely defines a tuple or row.
  - It ensures that every value in the specified key is unique.
  - A table can have any number of unique constraints, with at most one unique constraint defined as a Primary key.
  - A unique constraint can contain NULL value.

Data Integrity

- Column Constraint:
  - It specifies restrictions on the values that can be taken by a column.

DEPT table		
Deptno	Dname	Loc
10	Accounting	New York
20	Research	Dallas

EMP table				
Empno	Empname	Job	Mgr	Deptno
7369	Smith	Clerk	7902	20
7499	Allen	Salesman	7839	30

2.1: SQL

What is SQL?

- SQL:
  - SQL stands for Structured Query Language.
  - SQL is used to communicate with a database.
  - Statements are used to perform tasks such as update data on a database, or retrieve data from a database.
  - Benefits of SQL are:
    - It is a Non-Procedural Language.
    - It is a language for all users.
    - It is a unified language.

2.1 SQL

What can SQL do?

- SQL
  - allows you to access a database.
  - can execute queries against a database.
  - can retrieve data from a database.
  - can insert new records into a database.
  - can delete records from a database.
  - can update records in a database.

2.2: SQL

Rules for SQL statements

- Rules for SQL statements:
  - SQL keywords are not case sensitive. However, normally all commands (SELECT, UPDATE, etc) are upper-cased.
  - “Variable” and “parameter” names are displayed as lower-case.
  - New-line characters are ignored in SQL.
  - Many DBMS systems terminate SQL statements with a semi-colon character.
  - “Character strings” and “date values” are enclosed in single quotation marks while using them in WHERE clause or otherwise.

2.3: SQL

Standard SQL statement groups

- Given below are the standard SQL statement groups:

Groups	Statements	Description
DQL	SELECT	DATA QUERY LANGUAGE – It is used to get data from the database and impose ordering upon it.
DML	DELETE INSERT UPDATE MERGE	DATA MANIPULATION LANGUAGE – It is used to change database data.
DDL	DROP TRUNCATE CREATE ALTER	DATA DEFINITION LANGUAGE – It is used to manipulate database structures and definitions.
TCL	COMMIT ROLLBACK SAVEPOINT	TCL statements are used to manage the transactions.
DCL (Rights)	REVOKE GRANT	They are used to remove and provide access rights to database objects.

The Select Statement and Syntax

- The SELECT command is used to retrieve rows from a single table or multiple Tables or Views.
  - A query may retrieve information from specified columns or from all of the columns in the Table.
  - It helps to select the required data from the table.

```
SELECT [ALL | DISTINCT] { * | col_name,...}
FROM table_name alias,...
      [ WHERE expr1 ]
      [ CONNECT BY expr2 [ START WITH expr3 ] ]
      [ GROUP BY expr4 ] [ HAVING expr5 ]
      [ UNION | INTERSECT | MINUS SELECT ... ]
      [ ORDER BY expr | ASC | DESC ];
```

3.1: The SELECT Statement

Selecting Columns

- Displays all the columns from the student_master table

```
SELECT *
        FROM student_master;
```

- Displays selected columns from the student_master table

```
SELECT student_code, student_name
        FROM student_master;
```

3.2: SELECT statement Clauses

The WHERE clause

- The WHERE clause is used to specify the criteria for selection.
  - For example: displays the selected columns from the student_master table based on the condition being satisfied

```
SELECT  student_code,  student_name, student_dob
            FROM student_master
          WHERE dept_code = 10;
```

3.2: SELECT statement Clauses

The AS clause

- The AS clause is used to specify an alternate colum heading.
  - For example: displays the selected columns from the student_master table based on the condition being satisfied. Observe the column heading

```
SELECT student_dob as "Date of Birth"
            FROM student_master
            WHERE dept_code = 10;

 -- quotes are required when the column heading contains a space
```

```
SELECT student_dob   "Date of Birth"
            FROM student_master
            WHERE dept_code = 10;

-- AS keyword is optional
```

3.2: SELECT statement Clauses

Character Strings and Dates

- Are enclosed in single quotation marks
- Character values are case sensitive
- Date values are format sensitive

```
SELECT  student_code, student_dob
        FROM  student_master
        WHERE student_name = 'Sunil' ;
```

3.3: SELECT statement Clauses

Mathematical, Comparison & Logical Operators

- Mathematical Operators:
  - Examples: +, -, *, /
- Comparison Operators:

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or Equal to
<	Less than
<=	Less than or Equal to
<>, !=, or ^=	Not Equal to

- Logical Operators:
  - Examples: AND, OR, NOT

Other Comparison Operators

Other Comparison operators	Description
[NOT] BETWEEN x AND y	Allows user to express a range. For example: Searching for numbers BETWEEN 5 and 10. The optional NOT would be used when searching for numbers that are NOT BETWEEN 5 AND 10.
[NOT] IN(x,y,…)	Is similar to the OR logical operator. Can search for records which meet at least one condition contained within the parentheses. For example: Pubid IN (1, 4, 5), only books with a publisher id of 1, 4, or 5 will be returned. The optional NOT keyword instructs Oracle to return books not published by Publisher 1, 4, or 5.

3.3: SELECT statement Clauses

Other Comparison Operators

Other Comparison operators	Description
[NOT] LIKE	Can be used when searching for patterns if you are not certain how something is spelt. For example: title LIKE 'TH%'. Using the optional NOT indicates that records that do contain the specified pattern should not be included in the results.
IS[NOT]NULL	Allows user to search for records which do not have an entry in the specified field. For example: Shipdate IS NULL. If you include the optional NOT, it would find the records that do not have an entry in the field. For example: Shipdate IS NOT NULL.

3.3: SELECT statement Clauses

BETWEEN … AND Operator

- The BETWEEN … AND operator finds values in a specified range:

```
SELECT staff_code,staff_name
      FROM  staff_master
      WHERE  staff_dob  BETWEEN '01-Jan-1980'
         AND  '31-Jan-1980';
```

3.3: SELECT statement Clauses

IN Operator

- The IN operator matches a value in a specified list.
  - The List must be in parentheses.
  - The Values must be separated by commas.

```
SELECT dept_code
    FROM  department_master
    WHERE  dept_name IN ( 'Computer Science', 'Mechanics');
```

3.3: SELECT statement Clauses

LIKE Operator

- The LIKE operator performs pattern searches.
  - The LIKE operator is used with wildcard characters.
  - Underscore (_) for exactly one character in the indicated position
  - Percent sign (%) to represent any number of characters

```
SELECT book_code,book_name
       FROM  book_master
       WHERE  book_pub_author LIKE '%Kanetkar%' ;
```

3.3: SELECT statement Clauses

||Operator (Concatenation)

- The || operator performs concatenation.
  - between a string literal and a column name.
  - between two column names
  - between string literal and a pseudocolumn

```
SELECT 'Hello' ||   student_name
      FROM  student_master

-- only single quotes not double
```

```
SELECT student_code || '    ' ||   student_name
      FROM  student_master
```

```
SELECT 'Today is ' ||   sysdate
      FROM  dual
```

3.3: SELECT statement Clauses

Logical Operators

- Logical operators are used to combine conditions.
  - Logical operators are NOT, AND, OR.
    - NOT reverses meaning.
    - AND both conditions must be true.
    - OR at least one condition must be true.
  - Use of AND operator

```
SELECT staff_code,staff_name,staff_sal
      FROM  staff_master
      WHERE  dept_code = 10
      AND  staff_dob > '01-Jan-1945';
```

3.3: SELECT statement Clauses

Using AND or OR Clause

- Use of OR operator:

```
SELECT  book_code
    FROM  book_master
    WHERE  book_pub_author LIKE '%Kanetkar%'
    OR  book_name LIKE '%Pointers%';
```

3.3: SELECT statement Clauses

Using NOT Clause

- The NOT operator finds rows that do not satisfy a condition.
  - For example: List staff members working in depts other than 10 & 20.

```
SELECT staff_code,staff_name
      FROM  staff_master
      WHERE  dept_code NOT IN ( 10,20 );
```

3.3: SELECT statement Clauses

Treatment of NULL Values

- NULL is the absence of data.
- Treatment of this scenario requires use of IS NULL operator.

```
SQL>SELECT student_code
            FROM student_master
            WHERE dept_code IS NULL;
```

Operator Precedence

- Operator precedence is decided in the following order:

Levels	Operators
1	* (Multiply), / (Division), % (Modulo)
2	+ (Positive), - (Negative), + (Add), (+ Concatenate), - (Subtract), & (Bitwise AND)
3	=, >, <, >=, <=, <>, !=, !>, !< (Comparison operators)
4	NOT
5	OR
6	AND
7	ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
8	= (Assignment)

Lab 4

3.4: SELECT statement Clauses

The DISTINCT clause

- The SQL DISTINCT clause is used to eliminate duplicate rows.
  - For example: Displays student codes from student_marks tables. the student codes are displayed without duplication

```
SELECT DISTINCT student_code
    FROM student_marks;
```

3.5: SELECT statement Clauses

The ORDER BY clause

- The ORDER BY clause presents data in a sorted order.
  - It uses an “ascending order” by default.
  - You can use the DESC keyword to change the default sort order.
  - It can process a maximum of 255 columns.
- In an ascending order, the values will be listed in the following sequence:
  - Numeric values
  - Character values
  - NULL values
- In a descending order, the sequence is reversed.

3.5: SELECT statement Clauses

Sorting Data

- The output of the SELECT statement can be sorted using ORDER BY clause
  - ASC : Ascending order, default
  - DESC : Descending order
- Display student details from student_master table sorted on student_code in descending order.

```
SELECT Student_Code,Student_Name,Dept_Code, Student_dob
    FROM Student_Master
    ORDER BY Student_Code DESC ;
```

3.5: SELECT statement Clauses

Sorting Data

- Sorting data on multiple columns

```
SELECT Student_Code,Student_Name, Dept_Code,Student_dob
    FROM Student_Master
    ORDER BY Student_Code,Dept_Code;
```

LAB 5

4.1: Introduction to Functions

The Group Functions

- The Group functions are built-in SQL functions that operate on “groups of rows”, and return one value for the entire group.
- The results are also based on groups of rows.
- For Example, Group function called “SUM” will help you find the total marks, even if the database stores only individual subject marks.

Syntax : GROUP BY & HAVING clause

- Syntax

```
SELECT      [column, ] aggregate function(column), ……
FROM        table
[WHERE      condition]
[GROUP BY column]
[HAVING     condition]
[ORDER BY  column] ;
```

Listing of Group Functions

- Given below is a list of Group functions supported by SQL:

Function	Value returned
SUM (expr)	Sum value of expr, ignoring NULL values.
AVG (expr)	Average value of expr, ignoring NULL values.
COUNT (expr)	Number of rows where expr evaluates to something other than NULL. COUNT(*) counts all selected rows, including duplicates and rows with NULLs.
MIN (expr)	Minimum value of expr.
MAX (expr)	Maximum value of expr.

Aggregate (Group) Functions supported by SQL (contd.):

SUM(COL_NAME | EXPRESSION)

- SUMreturns the total of values present inaparticular “column” or a “number of columns” that are linked together in the expression. All the columns, which form the argument to SUM, must be numeric only. To find the sum of one subject for all students following query is used:

```
SELECT SUM(subject1)
        FROM student_marks;
```

- To find the yearly compensation of staff from department 20, the following query is used:

```
SELECT SUM(12*staff_sal)
        FROM staff_master
        WHERE dept_code= 20;
```

AVG(COL_NAME | EXPRESSION)

- AVGis similarto SUM. AVG returns the average of a NUMBER of values. The restrictions, which apply on SUM also, apply on AVG. To find the average salary the staff, the following query is used:

```
SELECT AVG(sal)
        FROM staff_master;
```

- To find the average yearly compensation of staff from department 20, the following query is used:

```
SELECT AVG(12*staff_sal)
        FROM staff_master
        WHERE dept_code = 20;
```

Aggregate (Group) Functions supported by SQL (contd.):

COUNT(*)

COUNT returns thenumber of rows.

To find thetotal numberof staff members,the following query is used:

```
SELECT COUNT(*)
        FROM staff_master;
```

- It is possible to restrict therows for the operation of COUNT.

- To find the total number of staff members in department 10, the following query is used:

```
SELECT COUNT(*)
        FROM staff_master
        WHERE dept_code = 10 ;
```

- To find the total number of staff members hired after '01-JAN-02', the following query is used:

```
SELECT COUNT(*)
        FROM staff_master
        WHERE hiredate>'01-JAN-02' ;
```

- When an aggregate function is used in a SELECT statement, column names cannot be usedin SELECT unless GROUP BY clause is used.

MIN(COL_NAME | EXPRESSION)

- MIN returns the lowest of the values from the column. MIN accepts columns, which are NON-NUMERIC too.
- To find the minimum salary paid to a staff member, the following query is used:

```
SELECT MIN(staff_sal)
        FROM staff_master;
```

- To list the staff member who alphabetically heads the list, the following query is used:

```
SELECT MIN(staff_name)
        FROM staff_master;
```

MAX(COL_NAME | EXPRESSION)

- MAXis the reverseof MIN. MAX returns the maximum value from among the list of values. To find the maximum marks for a student in subject2, the following query is used:

```
SELECT MAX(subject2)
        FROM student_marks;
```

Examples of using Group Functions

- Example 1: Display the total number of records from
- student_marks.

```
SELECT COUNT( * )
        FROM Student_Marks;
```

- Example 2: Display average marks from each subject.

```
SELECT  AVG(Student_sub1), AVG(Student_sub2), AVG(Student_sub3)
        FROM Student_Marks;
```

4.2 : Using the GROUP BY & HAVING clause

The GROUP BY clause

- GROUP BY clause is used along with the Group functions to retrieve data that is grouped according to one or more columns.
  - For example: Displays the average staff salary based on every department. The values are grouped based on dept_code

```
SELECT Dept_Code, AVG(Staff_sal)
    FROM Staff_Master
    GROUP BY  Dept_Code;
```

- the other columns on which the grouping is done
-

1. The GROUP BY clause should contain all the columns in the SELECT list, except those used along with the Group functions. Only the column names which have been used in GROUP BY clause and aggregate columns can be used in SELECT clause When an Aggregate (Group) function is used in a SELECT statement, the
2. column names cannot be used in SELECT, unless GROUP BY clause is used. Grouping can be done on multiple columns, as well.
3.

4.2 : Using the GROUP BY & HAVING clause

The HAVING clause

- HAVING clause is used to filter data based on the Group functions.
  - HAVING clause is similar to WHERE condition. However, it is used with Group functions.
- Group functions cannot be used in WHERE clause. However, they can be used in HAVING clause.

4.2 : Using the GROUP BY & HAVING clause

Examples – GROUP BY and HAVING clause

- For example: Display all department numbers having more than five employees.

```
SELECT Department_Code, Count(*)
        FROM Staff_Master
        GROUP BY Department_Code
        HAVING Count(*)> 5;
```

The HAVING clause (contd.):

- Tofind outAverage, Maximum, Minimum salary of departments, where average salary is greater than 2000.

```
SELECT dept_code,AVG(staff_sal),MIN(staff_sal),
MAX(staff_sal) FROM staff_master GROUP BY
dept_code HAVING AVG(staff_sal) > 2000;
```

- To find out average salary of all staff members who belong to department 10 or 20 and their average salary is greater than10000

```
SELECT design_code,dept_code,avg(staff_sal)
FROM staff_master where dept_code in(10,20)

GROUP BY design_code,dept_code
HAVING avg(staff_sal) >10000;
```

Quick Guidelines

- All individual columns included in the SELECT clause other than group functions must be specified in the GROUP BY clause.
- Any column other than selected column can also be placed in GROUP BY clause.
- By default rows are sorted by ascending order of the column included in the GROUP BY list.
- WHERE clause specifies the rows to be considered for grouping.

Quick Guidelines

- Suppose your SELECT statement contains a HAVING clause. Then write your query such that the WHERE clause does most of the work (removing undesired rows) instead of the HAVING clause doing the work of removing undesired rows.
- Use the GROUP BY clause only with an Aggregate function, and not otherwise.
  - Since in other cases, you can accomplish the same end result by using the DISTINCT option instead, and it is faster.

LAB 6

5.2: Types of Single Row Functions

Number Functions

- Number functions accept “numeric data” as argument, and returns “numeric values”.

TRUNC(arg,n)	Returns a number “arg“ truncated to a “n” number of decimal places.
ROUND (arg,n)	Returns “arg” rounded to “n” decimal places. If “n” is omitted, then “arg” is rounded as an integer.
CEIL (arg)	Returns the smallest integer greater than or equal to “arg”.
FLOOR (arg)	Returns the largest integer less than or equal to “arg”.
ABS (arg)	Returns the absolute value of “arg”.
POWER (arg, n)	Returns the argument “arg” raised to the n^{th} power.
SQRT (arg)	Returns the square root of “arg”.
SIGN (arg)	Returns –1, 0, or +1 according to “arg” which is negative, zero, or positive respectively.
MOD (arg1, arg2)	Returns the remainder obtained by dividing “arg1” by “arg2”.

5.3: Types of Single Row Functions

Character Functions

- Character functions accept "character data" as argument, and returns "character" or "number" values.

LOWER (arg)	Converts alphabetic character values to lowercase.
UPPER (arg)	Converts alphabetic character values to uppercase.
INITCAP (arg)	Capitalizes first letter of each word in the argument string.
CONCAT (arg1, arg2)	Concatenates the character strings "arg1" and "arg2".
SUBSTR (arg, pos, n)	Extracts a substring from "arg", "n" characters long, and starting at position "pos".
LTRIM (arg)	Removes any leading blanks in the string "arg".
RTRIM (arg)	Removes any trailing blanks in the string "arg".
LENGTH (arg)	Returns the number of characters in the string "arg".
REPLACE (arg, str1, str2)	Returns the string "arg" with all occurrences of the string "str1" replaced by "str2".
LPAD (arg, n, ch)	Pads the string "arg" on the left with the character "ch", to a total width of "n" characters.
RPAD (arg, n, ch)	Pads the string "arg" on the right with the character "ch", to a total width of "n" characters.
INSTR(string, pattern[, start [,occurrence]])	It returns the location of a character in a string.

1. Current Date/Time

- `CURRENT_DATE` → Returns current date.
- `CURRENT_TIME` → Returns current time (with or without time zone).
- `CURRENT_TIMESTAMP` / `NOW()` → Returns current date & time (with time zone).
- `LOCALTIMESTAMP` / `LOCALTIME` → Current timestamp/time (without time zone).
- `STATEMENT_TIMESTAMP()` → Time when the current SQL statement started.
- `CLOCK_TIMESTAMP()` → Current system clock time (can differ within a transaction).
- `TRANSACTION_TIMESTAMP()` → Alias for `NOW()`.

```
1  SELECT current_date, current_time, now(), localtime,
2  clock_timestamp()
3
```

Data Output | Explain | Messages | Notifications

	current_date date	current_time time with time zone	now timestamp with time zone	localtime time without time zone	clock_timestamp timestamp with time zone
1	2025-08-16	14:02:37.358125+05:30	2025-08-16 14:02:37.358125+05:...	14:02:37.358125	2025-08-16 14:02:37.358492+05:.

2. Extracting Parts of Date/Time

- `EXTRACT(field FROM source)` → Extracts specific part.

  Example:

  ```sql
  SELECT EXTRACT(YEAR FROM CURRENT_DATE);
  ```

  Common fields:

  - YEAR
  - MONTH
  - DAY
  - HOUR
  - MINUTE
  - SECOND
  - DOW (day of week, 0=Sunday)
  - DOY (day of year)
  - WEEK
  - QUARTER
  - CENTURY

3. Truncating Dates

- `DATE_TRUNC('field', source)` → Truncate to given precision.

```sql
SELECT DATE_TRUNC('month', NOW());
-- First day of current month
```

---

4. Interval Functions

- `AGE(timestamp1, timestamp2)` → Returns interval difference.
- `AGE(timestamp)` → Difference from current date.
- Arithmetic with intervals:

```sql
SELECT NOW() + INTERVAL '7 days'; -- 7 days later
SELECT NOW() - INTERVAL '1 year'; -- 1 year earlier
```

5. Conversion & Formatting

- `TO_CHAR(date/time, format)` → Format into string.

```sql
SELECT TO_CHAR(NOW(), 'YYYY-MM-DD HH24:MI:SS');
```

- `TO_DATE(string, format)` → Convert string to date.
- `TO_TIMESTAMP(string, format)` → Convert string to timestamp.

---

6. Casting

- `DATE '2025-08-16'` → Explicit date.
- `'2025-08-16'::DATE` → String to date.
- `'2025-08-16 10:30'::TIMESTAMP` → String to timestamp.

5.5: Types of Single Row Functions

Conversion Functions

- Conversion functions facilitate the conversion of values from one datatype to another.

TO_CHAR (arg,fmt)	Converts a number or date “arg” to a specific character format.
TO_DATE (arg,fmt)	Converts a date value stored as string to date datatype
TO_NUMBER (arg)	Converts a number stored as a character to number datatype.

PostgreSQL Conversion Functions

1. `TO_CHAR`

- Converts **date/time** or **number** to a **string** in a specified format.

📌 Examples:

```sql
-- Date to string
SELECT TO_CHAR(NOW(), 'YYYY-MM-DD HH24:MI:SS');
-- Output: '2025-08-16 22:45:30'

-- Number to string
SELECT TO_CHAR(12345.678, '99999.99');
-- Output: '12345.68'
```

2. `TO_NUMBER`

- Converts a **string** to a **numeric value** based on a given format.

📌 Examples:

```sql
-- String to number
SELECT TO_NUMBER('12,345.67', '99G999D99');
-- Output: 12345.67

SELECT TO_NUMBER('20250816', '99999999');
-- Output: 20250816
```

(`G` = group separator, `D` = decimal separator)

3. `TO_DATE`

- Converts a **string** to a **date** using the specified format.

📌 Examples:

```sql
-- String to date
SELECT TO_DATE('16-08-2025', 'DD-MM-YYYY');
-- Output: 2025-08-16

-- Another example
SELECT TO_DATE('2025 Aug 16', 'YYYY Mon DD');
-- Output: 2025-08-16
```

4. `TO_TIMESTAMP`

- Converts a **string** to a **timestamp**.

📌 Example:

sql

```sql
SELECT TO_TIMESTAMP('16-08-2025 10:30:15', 'DD-MM-YYYY HH24:MI:SS');
-- Output: 2025-08-16 10:30:15
```

LAB 7

5.6: Types of Single Row Functions

The Case Function

- Case() function
  - Conditional evaluation by doing work of an IF-THEN-ELSE statement
  - Syntax

```
CASE expr when compare_expr1 then return_expr1
                [when compare_exprn then return_exprn
ELSE else_expr]
END
```

  - Example

```
SELECT  staff_code, staff_name,
CASE dept_code WHEN 10 then 'Ten' ELSE 'Other' END
      FROM staff_master;
```

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ...
    ELSE resultN
END
```

- Evaluates conditions in order.
- Returns the result for the **first matching condition**.
- If no condition matches, the `ELSE` value is returned.
- If there's no `ELSE` and nothing matches, it returns `NULL`.

6.1: Joins

What are Joins?

- If we require data from more than one table in the database, then a join is used.
  - Tables are joined on columns, which have the same “data type” and “data width” in the tables.
  - The JOIN operator specifies how to relate tables in the query.
    - When you join two tables a Cartesian product is formed, by default.
  - Oracle supports
    - Oracle Proprietary
    - SQL: 1999 Compliant Joins

6.1: Joins

Types of Joins

- Given below is a list of JOINs supported by Oracle:

Oracle Proprietary Joins	SQL: 1999 Compliant Joins
Cartesian Product	Cross Joins
Equijoin	Inner Joins (Natural Joins)
Outer-join	Left, Right, Full outer joins
Non-equijoin	Join on
Self-join	Join Using

6.1.1: Oracle Proprietary Joins

Cartesian Joins

- A Cartesian product is a product of all the rows of all the tables in the query.
- A Cartesian product is formed when the join condition is omitted or it is invalid
- To avoid having Cartesian product always include a valid join condition
- Example

```
SELECT Student_Name, Dept_Name
        FROM  Student_Master, Department_Master;
```

6.1.1: Oracle Proprietary Joins

EquiJoin

- In an Equijoin, the WHERE statement compares two columns from two tables with the equivalence operator "=".
- This JOIN returns all rows from both tables, where there is a match.
- Syntax :

```
SELECT <col1>, <col2>,…
    FROM <table1>,<table2>
    Where <table1>.<col1>=<table2>.<col2>
    [AND <condition>] [ORDER BY <col1>, <col2>,…]
```

6.1.1: Oracle Proprietary Joins

EquiJoin - Example

- Example 1: To display student code and name along with the
- department name to which they belong

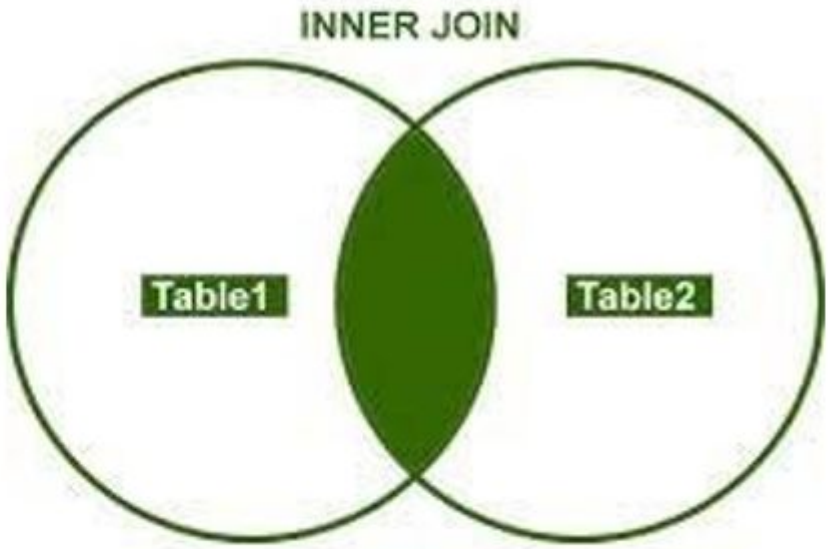
```
SELECT Student_Code,Student_name,Dept_name
 FROM Student_Master ,Department_Master
 WHERE Student_Master.Dept_code =Department_Master.Dept_code;
```

- Example 2: To display student and staff name along with the department name to which they belong

```
SELECT student_name,staff_name, dept_name
        FROM student_master, department_master,staff_master
        WHERE student_master.dept_code=department_master.dept_code
         and staff_master.dept_code=department_master.dept_code;
```

Equi Join

- Frequently, these type of JOIN involves PRIMARY and FOREIGN key complements. You can also use table aliases to qualify column names in the
- SELECT and Join Condition



6.1.1: Oracle Proprietary Joins

Non-EquiJoin

- A non-equi join is based on condition other than an equality operator
- Example: To display details of staff_members who receive salary in the range defined as per grade

```
SELECT  s.staff_name,s.staff_sal,sl.grade
      FROM staff_master s,salgrade sl
      WHERE staff_sal BETWEEN sl.losal and sl.hisal
```

🔄 What is an Outer Join?

- An **outer join** returns **all rows** from one (or both) tables, **even if there is no match** in the other table.
- Missing values are filled with `NULL`.

There are **3 types** of outer joins:

1. **LEFT OUTER JOIN (LEFT JOIN)**
   - Returns all rows from the **left table** and matching rows from the right.
   - If there's no match, the right table columns show `NULL`.
2. **RIGHT OUTER JOIN (RIGHT JOIN)**
   - Returns all rows from the **right table** and matching rows from the left.
   - If there's no match, the left table columns show `NULL`.
3. **FULL OUTER JOIN (FULL JOIN)**
   - Returns all rows from **both tables**.
   - If a row exists only in one table, the other side shows `NULL`.

🛠️ Examples with `users` & `devices` tables

1️⃣ LEFT JOIN

```sql
SELECT u.user_id, u.username, d.device_type, d.device_os
FROM users u
LEFT JOIN devices d ON u.user_id = d.user_id;
```

✔️ Shows **all users** (even those with no device).

📌 If a user has no device → device fields will be `NULL`.

2️⃣ RIGHT JOIN

sql

```sql
SELECT u.user_id, u.username, d.device_type, d.device_os
FROM users u
RIGHT JOIN devices d ON u.user_id = d.user_id;
```

✔️ Shows **all devices** (even those not linked to a user).

📌 If a device is not linked to a user → user fields will be `NULL`.

3️⃣ FULL OUTER JOIN

```sql
SELECT u.user_id, u.username, d.device_type, d.device_os
FROM users u
FULL OUTER JOIN devices d ON u.user_id = d.user_id;
```

✔️ Shows **all users** and **all devices**.

📌 If a user has no device → device fields `NULL`.

📌 If a device has no user → user fields `NULL`.

6.1.1: Oracle Proprietary Joins

Self Join

- In Self Join, two rows from the “same table” combine to form a “resultant row”.
  - It is possible to join a table to itself, as if they were two separate tables, by using aliases for table names.
  - This allows joining of rows in the same table.
- Example: To display staff member information along with their manager information

```
SELECT staff.staff_code, staff.staff_name,
       mgr.staff_code, mgr.staff_name
FROM staff_master staff, staff_master mgr
WHERE staff.mgr_code  = mgr.staff_code;
```

LAB 8

6.2: Subqueries

What is a SubQuery?

- A sub-query is a form of an SQL statement that appears inside another SQL statement.
  - It is also called as a “nested query”.
- The statement, which contains the sub-query, is called the “parent statement”.
- The “parent statement” uses the rows returned by the sub-query.

Subquery - Examples

- Example 1: To display name of students from “Mechanics” department.
  - Method 1:

```
SELECT Dept_Code
   FROM Department_Master
   WHERE Dept_name = 'Mechanics';
```

  - O/P : 40

```
SELECT student_code,student_name
FROM student_master
WHERE dept_code=40;
```

Subquery - Examples

- Example 1 (contd.):
  - Method 2: Using sub-query

```
SELECT student_code, student_name
        FROM student_master
       WHERE dept_code = (SELECT dept_code
                          FROM department_master
                          WHERE dept_name = 'Mechanics');
```

6.2: Subqueries

Where to use Subqueries?

- Subqueries can be used for the following purpose :
  - To insert records in a target table.
  - To create tables and insert records in the table created.
  - To update records in the target table.
  - To create views.
  - To provide values for conditions in the clauses, like WHERE, HAVING, IN, etc., which are used with SELECT, UPDATE and DELETE statements.

6.2: Subqueries

Comparison Operators for Subqueries

- Types of SubQueries
  - Single Row Subquery
  - Multiple Row Subquery.
- Some comparison operators for subqueries:

Operator	Description
IN	Equals to any member of
NOT IN	Not equal to any member of
*ANY	compare value to every value returned by sub-query using operator *
*ALL	compare value to all values returned by sub-query using operator *

Using Comparison Operators - Examples

- Example 1: To display all staff details of who earn salary least salary

```
SELECT staff_name, staff_code, staff_sal
    FROM staff_master
      WHERE staff_sal  = (SELECT MIN(staff_sal)
                          FROM staff_master) ;
```

- Example 2: To display staff details who earn salary greater than average salary earned in dept 10

```
SELECT staff_code,staff_sal
      FROM staff_master
      WHERE staff_sal > ANY(SELECT AVG(staff_sal)
                            FROM staff_master  GROUP BY dept_code);
```

7.1: Database Objects

Table

- Tables are objects, which store the user data.
- Use the CREATE TABLE statement to create a table, which is the basic structure to hold data.
- For example:

```
CREATE TABLE book_master
(book_code number,
book_name varchar2(50),
book_pub_year number,
book_pub_author varchar2(50));
```

Data Integrity: Integrity Constraint

- AnIntegrity Constraint is a declarative method of defining a rule for a column of atable.

Example of Data Integrity:

- Assume that youdefine an “Integrity Constraint” for the Staff_Sal column of the Staff_Master table. This Integrity Constraint enforces the rule that “no row in this table can contain a numeric value greater than 10,000 in this column”. If an
- INSERT or UPDATE statement attempts to violate this “Integrity Constraint”, Oracle rolls back the statement and returns an “information error” message.
-

7.2: Data Integrity

Types of Integrity Constraints

- Let us see the types of Data Integrity Constraints:
  - Nulls
  - Default
  - Unique Column Values
  - Primary Key Values
  - Check
  - Referential Integrity

7.3: Examples of CREATE TABLE

NOT NULL Constraint

- The user will not be allowed to enter null value.
- For Example:
  - A NULL value is different from a blank or a zero. It is used for a quantity that is "unknown".
  - A NULL value can be inserted into a column of any data type.

```
CREATE TABLE  student_master
(student_code  number(4) NOT NULL,
dept_code   number(4) CONSTRAINT dept_code_nn
                                NOT  NULL );
```

7.3: Examples of CREATE TABLE

DEFAULT clause

- If no value is given, then instead of using a “Not Null” constraint, it is sometimes useful to specify a default value for an attribute.

- For Example:
  - When a record is inserted the default value can be considered.

```
CREATE TABLE  staff_master(
Staff_Code number(8) PRIMARY KEY,
Staff_Name varchar2(50) NOT NULL,
Staff_dob date,
Hiredate date DEFAULT sysdate,
…..)
```

7.3: Examples of CREATE TABLE

UNIQUE constraint

- The keyword UNIQUE specifies that no two records can have the same attribute value for this column.

- For Example:

```
CREATE TABLE student_master
(student_code number(4),
 student_name varchar2(30) ,
CONSTRAINT stu_id_uk  UNIQUE(student_code )) ;
```

7.3: Examples of CREATE TABLE

PRIMARY KEY constraint

- The Primary Key constraint enables a unique identification of each record in a table.

- For Example:

```
CREATE TABLE Staff Master
(staff_code  number(6)
CONSTRAINT staff_id_pk PRIMARY KEY,
 staff_name varchar2(20)
  ……);
```

7.3: Examples of CREATE TABLE

CHECK constraint

- CHECK constraint allows users to restrict possible attribute values for a column to admissible ones.

- For Example:

```
CREATE TABLE staff_master
( staff_code number(2),
  staff_name  varchar2(20),
  staff_sal   number(10,2) CONSTRAINT staff_sal_min
                           CHECK (staff_sal >1000),
  .....) ;
```

7.3: Examples of CREATE TABLE

FOREIGN KEY constraint

- The FOREIGN KEY constraint specifies a “column” or a “list of columns” as a foreign key of the referencing table.
- The referencing table is called the “child-table”, and the referenced table is called “parent-table”.

- For Example:

```
CREATE TABLE student_master
(student_code number(6) ,
 dept_code number(4) CONSTRAINT stu_dept_fk
                REFERENCES department_master(dept_code),
 student_name varchar2(30) );
```

7.3: Examples of CREATE TABLE

Create new table based on existing table

- Constraints on an “old table” will not be applicable for a “new table”.

```
CREATE TABLE student_dept10 AS
SELECT student_code, student_name
FROM student_master WHERE  dept_code = 10
```

7.4: Examples of ALTER TABLE

ALTER Table

- Given below is an example of ALTER TABLE:

```
ALTER TABLE table_name
        [ADD (col_name col_datatype col_constraint ,...)]|
        [ADD (table_constraint)]|
        [DROP CONSTRAINT constraint_name]|
        [MODIFY existing_col_name new_col_datatype
                                        new_constraint new_default]
        [DROP COLUMN existing_col_name]
        [SET UNUSED COLUMN existing_col)name];
```

7.4: Examples of ALTER TABLE

ALTER Table – Add clause

- The “Add” keyword is used to add a column or constraint to an existing table.
  - For adding three more columns to the emp table, refer the following example:

```
ALTER TABLE Student_Master
ADD (last_name varchar2(25) );
```

7.4: Examples of ALTER TABLE

ALTER Table – Add clause

- For adding Referential Integrity on “mgr_code" column, refer the following example:

```
ALTER TABLE staff_master
  ADD CONSTRAINT FK FOREIGN KEY  (mgr_code) REFERENCES
staff_master(staff_code);
```

1. Change a column's data type

```sql
ALTER TABLE table_name
ALTER COLUMN column_name TYPE new_data_type;
```

Example:

```sql
ALTER TABLE staff_master
ALTER COLUMN staff_sal TYPE numeric(12,2);
```

2. Rename a column

```sql
ALTER TABLE table_name
RENAME COLUMN old_name TO new_name;
```

Example:

```sql
ALTER TABLE staff_master
RENAME COLUMN staff_sal TO salary;
```

3. Set or drop default value

```sql
ALTER TABLE table_name
ALTER COLUMN column_name SET DEFAULT some_value;

ALTER TABLE table_name
ALTER COLUMN column_name DROP DEFAULT;
```

---

4. Set or drop NOT NULL constraint

```sql
ALTER TABLE table_name
ALTER COLUMN column_name SET NOT NULL;

ALTER TABLE table_name
ALTER COLUMN column_name DROP NOT NULL;
```

5. Add or drop constraints

sql

```sql
ALTER TABLE table_name
ADD CONSTRAINT constraint_name CHECK (salary > 0);

ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

7.5: Database Objects

Drop a Table

- The DROP TABLE command is used to remove the definition of a table from the database.

- For Example:

```
DROP TABLE staff_master;

DROP TABLE Department_master
    CASCADE CONSTRAINTS;
```

7.5: Database Objects

Truncating a Table

- The TRUNCATE command is used to permanently remove the data from a table, keeping the table structure intact.
- For Example:

```
TRUNCATE TABLE staff_master ;
```

7.6: Index

Usage of Index

- Index is a database object that functions as a “performance-tuning” method for allowing faster retrieval of records.
- Index creates an entry for each value that appears in the indexed columns.
- The absence or presence of an Index does not require change in wording of any SQL statement.

7.6: Index

Usage of Index

- Syntax:

```
CREATE [UNIQUE] INDEX index_name
ON table_name(col_name1 [ASC|DESC],col_name2,.....)
```

7.6: Index

Creating an Index

- Example 1: A simple example of an Index is given below:

```
CREATE INDEX  staff_sal_index ON staff_master(staff_sal);
```

- Example 2: To allow only unique values in the field “ename”, the CREATE statement should appear as shown below:

```
CREATE UNIQUE INDEX  staff_ename_unindex
ON staff_master(staff_name );
```

7.6: Index

How are Indexes created?

- Indexes can be either created “automatically” or “manually”.
  - Automatically: A unique Index is automatically created when you define a PRIMARY KEY or UNIQUE constraint in a table definition.
  - Manually: A non-unique index can be created on columns by users in order to speed up access to the rows.

Creating an Index - Examples

- Example 1: The index shown below can be very useful when you query for comparing revenue - cost.

- Example 2:

```
CREATE INDEX sales_margin_index
    ON sales(revenue - cost )
```

```
CREATE INDEX uppercase_idx
    ON staff_master (UPPER(staff_name));
```

7.7 View

Usage of View

- A View can be thought of as a “stored query” or a “virtual table”, i.e. a logical table based on one or more tables.
  - A View can be used as if it is a table.
  - A View does not contain data.

Capgemini

Why do we use Views?

Views areused:

- To restrict data access. To make complex
- queries easy. To provide data
- independence. To present different views of
- the same data.

Features of Simple and Complex Views:

Features	Simple View	Complex View
Number of tables	One	One or more
Contains functions	No	Yes Yes
Contains groups of data	No	
DML operations through a View.	Yes	Not always

7.7 View

Usage of View

- Syntax

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view [(alias[, alias]...)]
AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

7.7: View

Creating a View

- Given below is an example of a simple View:

```
CREATE VIEW staff_view
    AS
    SELECT * FROM staff_master
    WHERE hiredate >'01-jan-82';
```

7.7 View

Creating a View

- Creating a Complex View:
  - As shown in the example given below, create a Complex View that contains group functions to display values from two tables.

```
CREATE VIEW dept_sum_vu
 (name, minsal, maxsal, avgsal)
AS SELECT dept.dept_name, MIN(staff.staff_sal),
          MAX(staff. staff_sal),AVG(staff. staff_sal)
  FROM   staff_master staff, department_master dept
  WHERE staff.dept_code = dept.dept_code
  GROUP BY dept.dept_name;
```

7.7 View

Rules for performing operation on View

- You can perform “DML operations” on simple Views.
- You cannot remove a row if the View contains the following:
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudocolumn ROWNUM keyword

7.8 Deleting Database Objects

Deleting Database Objects

- Use the following syntax for deleting database objects:

- Example 1:

Given below is an example of deleting a table:

```
DROP Obj_Type obj_name;
```

```
DROP TABLE student_marks;
```

8.1: Addition of Data into Tables

INSERT

- INSERT command:
  - INSERT is a DML command. It is used to add rows to a table.
  - In the simplest form of the command, the values for different columns in the row to be inserted have to be specified.
  - Alternatively, the rows can be generated from some other tables by using a SQL query language command.

8.1: Addition of Data into Tables

Inserting Rows into a Table

- Inserting by specifying values:
- Example: To insert a new record in the DEPT table

```
INSERT INTO table_name[(col_name1,col_name2,...)]
{VALUES (value1,value2,....) | query};
```

```
INSERT INTO Department_master
VALUES (10, 'Computer Science');
```

8.1: Addition of Data into Tables

Inserting Rows into a Table

- Inserting rows in a table from another table using Subquery:
- Example: The example given below assumes that a new_emp_table exists. You can use a subquery to insert rows from another table.

```
INSERT INTO new_staff_table
SELECT * FROM staff_master
WHERE staff_master.hiredate > '01-jan-82';
```

8.2: Deletion of Data from Tables

DELETE

- The DELETE command is used to delete one or more rows from a table.
  - The DELETE command removes all rows identified by the WHERE clause.

```
DELETE [FROM] {table_name | alias }
        [WHERE condition];
```

8.2: Deletion of Data from Tables

Deleting Rows from Table

- Example 1: If the WHERE clause is omitted, all rows will be deleted from the table.

- Example 2: If we want to delete all information about department 10 from the Emp

```
DELETE
    FROM staff_master;
```

```
DELETE
    FROM student_master
    WHERE  dept_code=10;
```

8.3: Modifying / Updating existing Data in a Table

UPDATE

- Use the UPDATE command to change single rows, groups of rows, or all rows in a table.
  - In all data modification statements, you can change the data in only “one table at a time”.

```
UPDATE table_name
SET   col_name = value|
      col_name = SELECT_statement_returning_single_value|
      (col_name,...) = SELECT_statement
[WHERE condition];
```

8.3: Modifying / Updating existing Data in a Table

Updating Rows from Table

- Example 1: To UPDATE the column “dname” of a row, where deptno is 10, give the following command:

```
UPDATE department_master
SET dept_name= ‘Information Technology’
WHERE dept_code=10;
```

8.3: Modifying / Updating existing Data in a Table

Updating Rows from Table

- Example 2: To UPDATE the subject marks details of a particular student, give the following command:

```
UPDATE student_marks
SET subject1= 80 , subject2= 70
        WHERE student_code=1005;
```

8.3: Modifying / Updating existing Data in a Table

Using a Subquery to do an Update

- For making salary of “Anil” equal to that of staff member 100006, use the following command:

```
UPDATE staff_master
SET staff_sal = (SELECT staff_sal FROM staff_master
                 WHERE staff_code = 100006 )
WHERE staff_name = 'Anil';
```

Transactions follow **ACID**:

1. **Atomicity** → all or nothing.
2. **Consistency** → data stays valid.
3. **Isolation** → transactions don't interfere with each other.
4. **Durability** → once committed, changes stay (even after crash).

🔄 Transaction Commands

1. Start a transaction

sql Copy Edit

```sql
BEGIN;
-- or in some databases: START TRANSACTION;
```

2. Perform SQL operations

sql Copy Edit

```sql
UPDATE accounts SET balance = balance - 500 WHERE account_id = 1;
UPDATE accounts SET balance = balance + 500 WHERE account_id = 2;
```

3. Save changes permanently

```sql
COMMIT;
```

4. Undo changes (if something goes wrong)

```sql
ROLLBACK;
```

Example: Bank Transfer

Imagine transferring ₹500 from **Account A → Account B**:

sql

Copy

```sql
BEGIN;

UPDATE accounts SET balance = balance - 500 WHERE account_id = 'A';
UPDATE accounts SET balance = balance + 500 WHERE account_id = 'B';

COMMIT;
```

- If both updates succeed → `COMMIT` makes it permanent.
- If second update fails (say, account doesn't exist) → you can `ROLLBACK` so no money is lost.

⚠️ Auto-Commit Mode

- By default, in PostgreSQL, MySQL, SQL Server, etc., each SQL statement is automatically committed.
- To group multiple statements, you explicitly use `BEGIN` and `COMMIT`.
