|  |
| --- |
|  |
|

**LAB # 01**

|  |
| --- |
| **Introduction to Database Systems and Foundation Statements of SQL** |

**Objectives:**

* Install and set up Oracle Database and SQL Developer.
* Understand the concept of databases and their different types (Relational, Hierarchical, Network, Object-Oriented).
* Get familiar with basic SQL statements: SELECT, FROM, and WHERE.
* Learn to use basic comparison operators: =, >, <, >=, <=, <>.
* Execute simple queries on sample schemas (e.g., HR) to retrieve, and filter data.
* Develop foundational skills to write basic SQL queries for future labs.

|  |
| --- |
| **Data Vs Information Vs Knowledge** |

In the world of computing and databases, the terms Data and Information are often used interchangeably, but they have distinct technical meanings.

**Data** refers to raw, unprocessed facts, figures, and symbols. It is the atomic unit of meaning; individual observations that, on their own, may lack context.

* *Example:* Numbers like 24, 95, 101, or strings like "Ali", "Red". Without context, we do not know if 24 is an age, a temperature, or a quantity.

**Information** is data that has been processed, organized, structured, or presented in a given context so as to make it useful. It provides answers to "who", "what", "where", and "when".

* *Example:* "Ali is 24 years old and scored 95 in the exam." Here, the raw data has been processed into a meaningful statement.

**Knowledge** is information that has been further processed through experience, analysis, understanding, and reasoning. It represents insights, conclusions, or decisions derived from information. Knowledge answers “how” and “why”, and it enables action, prediction, and decision-making.

In computing and database systems, knowledge often emerges when information is analyzed over time, patterns are identified, and rules or expertise are formed.

* *Example:* From the information *“Ali is 24 years old and scored 95 in the exam”*, we can derive knowledge such as:  
  *“Ali is a high-performing student and is likely to succeed in advanced courses.”*

Here, past experience, evaluation criteria, and reasoning are applied to the information to form knowledge.

Flow: Data → Information → Knowledge

|  |
| --- |
| **Database** |

A Database is a systematic collection of data. It is an organized body of related information, stored in a way that allows for efficient retrieval, insertion, and updating.

Think of a database as an electronic filing cabinet. Before computers, we used paper files (File Processing System). A computerized database solves many of the problems associated with paper files, such as redundancy (duplicate data) and inconsistency.

|  |
| --- |
| **The Database Management System (DBMS)** |

A **Database Management System (DBMS)** is the software that interacts with end-users, applications, and the database itself to capture and analyze the data. It serves as an interface between the user and the physical data.

Analogy: If the *Database* is the library full of books, the *DBMS* is the librarian who locates the books, ensures they are organized, manages who can borrow them, and puts them back safely.

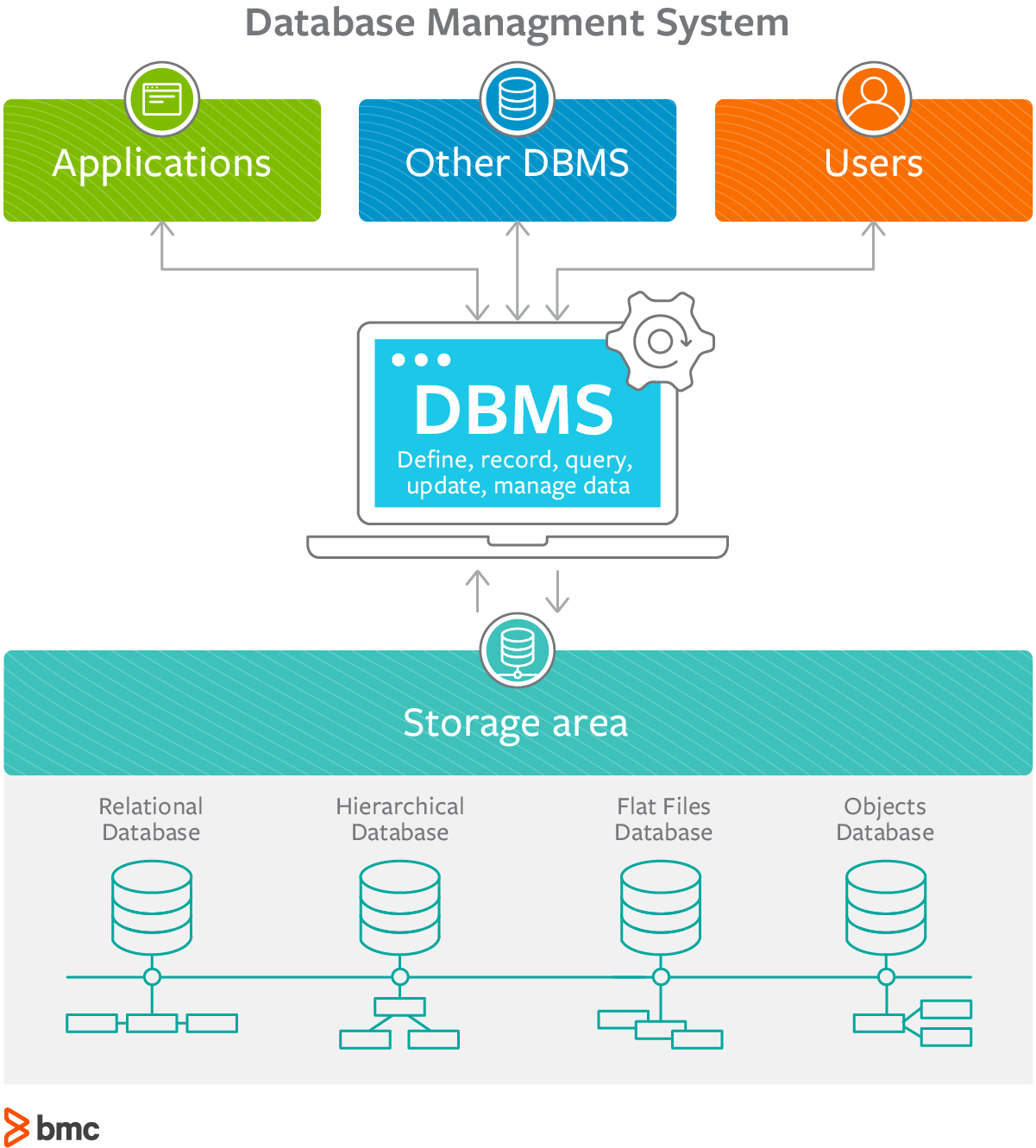
* Examples: MySQL, Oracle Database, Microsoft SQL Server, PostgreSQL.

|  |
| --- |
| **The Database System** |

The term **Database System** refers to the collective environment. It typically consists of four components:

1. **Users:** Programmers, Administrators, and End-users.
2. **Database Applications:** The programs users interact with (e.g., a website or a banking app).
3. **The DBMS:** The software managing the data.
4. **The Database:** The actual stored data.

Equation: Database System = Database + DBMS + Application Programs + Users.



*Figure 1: The Database Management System*

|  |
| --- |
| **Introduction to SQL** |

SQL stands for Structured Query Language; it lets you access and manipulate databases.

What Can SQL do?

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

|  |
| --- |
| **Basic SQL Syntax and Rules** |

Before writing code, you must understand the grammar of SQL (Structured Query Language):

1. **Semicolons (;):** Every SQL statement *must* end with a semicolon. It tells the server "I am done typing this command, execute it now."
2. **Case Insensitivity:** SQL keywords (SELECT, create, FROM) are not case-sensitive. select is the same as SELECT. However, it is a standard convention to write KEYWORDS IN UPPERCASE and identifiers (table names, columns) in lowercase.
3. **Quotes:** String values must be enclosed in single quotes 'value'.
4. **Whitespace:** Extra spaces and newlines are ignored, allowing you to format code for readability.

|  |
| --- |
| **Foundation Statements in SQL** |

|  |
| --- |
| **1. CONNECT Statement** |

To select a particular database to work with you issue the Connect statement as follows:

|  |
| --- |
| CONNECT *database\_name*; |

In this statement, following the connect keyword is the name of the database that you want to select.

|  |
| --- |
| **2. SELECT Statement** |

The SELECT statement is used to select data from a database. The data returned is stored in a result table, called the result-set. A SELECT indicates that we are merely reading information, as opposed to modifying it. What we are selecting is identified by an expression or column list immediately following the SELECT. The FROM statement specifies the name of the table or tables from which we are getting our data.

When you want to select **particular fields** available in the table, use the following syntax:

|  |
| --- |
| SELECT *column1, column2,* ... FROM *table\_name*; |

**Example:**

**Query Statement:** Show first name, last name, and salary of all employees.

**Query:**

|  |
| --- |
| SELECT first\_name, last\_name, salary FROM employees; |

It Selects data of these three columns from the Employees table

When you want to select **all the fields** available in the table, use the following syntax:

|  |
| --- |
| SELECT \*  FROM *table\_name*; |

**Example:**

|  |
| --- |
| SELECT \* FROM Employees; |

Selects all the employees records from the database and displays its columns.

|  |
| --- |
| **3. WHERE clause** |

The next thing we want to do is to start limiting, or filtering, the data we fetch from the database.

By adding a WHERE clause to the SELECT statement, we add one (or more) conditions that must be met by the selected data. This will limit the number of rows that answer the query and are fetched. In many cases, this is where most of the "action" of a query takes place.

In other words:

The WHERE clause is used to filter records.

The WHERE clause is used to extract only those records that fulfill a specified condition.

**Syntax:**

|  |
| --- |
| SELECT *column1, column2, ...*  FROM *table\_name*  WHERE *condition*; |
| **Note:** The WHERE clause is not only used in SELECT statement, it is also used in UPDATE, DELETE statement, etc.! (will learn in upcoming labs) |

**Example:** The following SQL statement selects all the employees from the FIRST\_NAME "Ellen", in the "employees" table:

|  |
| --- |
| SELECT *\**  FROM employees  WHERE FIRST\_NAME = 'Ellen'; |

If you want to get the opposite, the employees other than Ellen then query will be:

|  |
| --- |
| SELECT *\**  FROM employees  WHERE FIRST\_NAME <> 'Ellen'; |

Also you can use “**!=**” at the place of “**<>**”.

**Text Field & Numeric Fields**

Oracle requires single quotes around text values (most database systems will also allow double quotes).

However, numeric fields should not be enclosed in quotes.

**Syntax:**

|  |
| --- |
| SELECT *\**  FROM employees  WHERE EMPLOYEE\_ID = 103; |

|  |
| --- |
| **Operators in the WHERE clause** |

**1. SQL Comparison Operators**

|  |  |
| --- | --- |
| **Operator** | **Description** |
| **=** | Checks if the values of two operands are equal or not, if yes then condition becomes true. |
| **!=** | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. |
| **<>** | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. |
| **>** | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. |
| **<** | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. |
| **>=** | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. |
| **<=** | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. |

THE END!

Lab tasks:

Q1: Display all employees whose salary is between 3000 and 7000.

Q2: Display all employees whose first\_name starts with ‘A’.

Q3: Display all employees whose department\_id is 50 OR 80.

Q4: Display all employees whose salary is greater than 5000 AND department\_id is 60.

Q5: Display all employees whose first\_name does NOT start with ‘S’.

Q6: Display all employees whose hire\_date is between 01-JAN-2004 and 31-DEC-2008.

Q7: Display all employees whose job\_id is ‘IT\_PROG’ OR salary is greater than 8000.

Q8: Display all employees whose department\_id is NOT between 30 and 90.

Q9: Display all employees whose first\_name contains ‘an’.

Q10: Display all employees whose salary is less than 4000 AND first\_name ends with ‘n’.