## ****Day 2: SQL - SELECT Statements and Filtering Data****

### ****Objective:****

Learn how to query data using SELECT and apply filtering techniques.

### ****Session Flow & Content****

**9:45 AM – 11:00 AM: SELECT Basics**

#### ****Concepts Covered:****

* Basic SELECT queries
* SELECT \* vs SELECT column list
* WHERE clause filtering
* ORDER BY clause sorting
* DISTINCT to remove duplicates

**Code Examples:**

SELECT \* FROM Employee;

SELECT Name, Department FROM Employee WHERE Salary > 50000 ORDER BY Salary DESC;

SELECT DISTINCT Department FROM Employee;

**Use Case 2: Payroll Querying**

* Display all employees with salary > 60K
* List distinct departments for workforce planning

**11:00 AM – 11:15 AM: Tea/Coffee Break**

**11:15 AM – 1:00 PM: Logical & Comparison Operators**

#### ****Operators Covered:****

* Comparison: =, <>, >, <, >=, <=
* Logical: AND, OR, NOT

**Code Examples:**

SELECT \* FROM Employee WHERE Department = 'IT' AND Salary >= 70000;

**1:00 PM – 2:00 PM: Lunch Break**

**2:00 PM – 3:30 PM: LIKE, BETWEEN, IN**

#### ****Advanced Filtering:****

* **LIKE:** Pattern-based filtering
* **BETWEEN:** Range filtering
* **IN:** Multi-value filtering

**Code Examples:**

SELECT \* FROM Employee WHERE Name LIKE 'A%';

SELECT \* FROM Employee WHERE Salary BETWEEN 40000 AND 60000;

SELECT \* FROM Employee WHERE Department IN ('HR', 'IT');

**Use Case 3: Filter Reports for Business Decisions**

* Identify IT employees within mid-salary range
* Filter employee names starting with a specific letter

**3:30 PM – 3:45 PM: Tea/Coffee Break**

**3:45 PM – 5:30 PM: Lab Practice on Filtering & Sorting**

**Lab Task:**

* List all employees not in HR department
* Show employees with joining dates after Jan 2022
* Use IN, BETWEEN, and LIKE in practice

**5:30 PM – 6:00 PM: Recap & Live Quiz**

* Recap of SELECT, WHERE, ORDER BY
* Kahoot/Forms quiz

## Coming Next:

Would you like me to continue writing detailed content for **Day 3 to Day 5** and **Week 2: PL/SQL Developer Training** in the same structure?

Let me know if you'd like changes to trainer script style, examples, or activity complexity.

**🔍 Interesting and Rarely Known SQL Concepts to Add**

**✅ 1. NULL Is Not Equal to NULL**

In SQL, NULL = NULL returns **FALSE**, not TRUE.

💡 Why?  
Because NULL represents "unknown", and comparing two unknowns doesn't confirm equality.

✅ **Use this instead**:

sql

CopyEdit

SELECT \* FROM Employee WHERE Salary IS NULL;

**Trainer Note:** Use this to explain pitfalls in filtering missing values. Many developers make this mistake in WHERE clauses.

**✅ 2. WHERE vs HAVING – Misused Often**

Most devs use HAVING like WHERE, but **HAVING is only for aggregated data** (i.e., used with GROUP BY).

sql

CopyEdit

-- WRONG:

SELECT \* FROM Employee HAVING Salary > 50000;

-- RIGHT:

SELECT Department, AVG(Salary) FROM Employee

GROUP BY Department

HAVING AVG(Salary) > 50000;

**✅ 3. SQL Is Case-Insensitive (But Not Always!)**

While keywords like SELECT, WHERE, etc. are case-insensitive, **column and string values are not** in all databases.

For example:

sql

CopyEdit

SELECT \* FROM Users WHERE Name = 'John';

-- Might not return 'john' or 'JOHN' in some DBs.

✅ Tip: Use UPPER() or LOWER() to normalize case:

sql

CopyEdit

WHERE UPPER(Name) = 'JOHN'

**✅ 4. You Can Use CASE in SELECT for Conditional Columns**

You can simulate IF-ELSE logic inside SELECT:

sql

CopyEdit

SELECT Name,

CASE

WHEN Salary >= 70000 THEN 'High Earner'

ELSE 'Regular'

END AS Salary\_Bracket

FROM Employee;

**Trainer Tip:** Show this in live coding to label patient risk level (e.g., HIGH, MEDIUM, LOW) based on symptoms or age.

**✅ 5. LIMIT vs ROWNUM vs FETCH FIRST**

Depending on DBs, the way to fetch "Top N Records" **varies**:

* **Oracle:** ROWNUM

sql

CopyEdit

SELECT \* FROM Employee WHERE ROWNUM <= 5;

* **MySQL/PostgreSQL:** LIMIT

sql

CopyEdit

SELECT \* FROM Employee LIMIT 5;

* **SQL Server / Standard SQL:**

sql

CopyEdit

SELECT \* FROM Employee FETCH FIRST 5 ROWS ONLY;

💡 **Ask learners** what DB they're most familiar with and do a comparative demo.

**✅ Bonus Use Case (Security Simulation)**

**Query:** Simulate user-based filtering.

sql

CopyEdit

SELECT \* FROM Patient\_Records

WHERE Department = CASE

WHEN :user\_role = 'Doctor' THEN 'Cardiology'

WHEN :user\_role = 'Admin' THEN Department

END;

🔒 This is a simulation of **row-level access control** that mimics real-world security requirements.

Would you like me to integrate these directly into your **Day 2 trainer script in Canvas**? I can insert them as **"Advanced Tips" blocks** right after key sections.

**🔍 Interesting Facts About PL/SQL**

**✅ 1. PL/SQL Has Boolean Data Type — Unlike SQL**

* Unlike SQL (which does **not** support BOOLEAN), **PL/SQL supports BOOLEAN natively**.
* You can use TRUE, FALSE, and NULL in PL/SQL, but **you cannot use BOOLEAN in SQL queries or DDL**.

plsql

CopyEdit

DECLARE

is\_valid BOOLEAN := TRUE;

BEGIN

IF is\_valid THEN

DBMS\_OUTPUT.PUT\_LINE('Valid!');

END IF;

END;

**✅ 2. Exception Section Is Optional But Powerful**

* Even if you **don’t include an EXCEPTION block**, your code will still compile.
* But **without exception handling**, you can’t gracefully handle run-time errors, which makes debugging harder.
* You can even define **user-defined exceptions** with custom messages.

**✅ 3. Nested Named Blocks Are Supported**

* You can define **PL/SQL blocks inside PL/SQL blocks**, and they can have **local scope** for variables and exceptions.

plsql

CopyEdit

DECLARE

v\_message VARCHAR2(50);

BEGIN

DECLARE

v\_message VARCHAR2(50) := 'Nested Block!';

BEGIN

DBMS\_OUTPUT.PUT\_LINE(v\_message);

END;

v\_message := 'Outer Block!';

DBMS\_OUTPUT.PUT\_LINE(v\_message);

END;

**✅ 4. You Can Use %TYPE and %ROWTYPE for Schema Independence**

* These are **attribute inheritance features** that make your code more **robust and maintainable**.

plsql

CopyEdit

DECLARE

v\_name EMPLOYEES.FIRST\_NAME%TYPE;

v\_row EMPLOYEES%ROWTYPE;

BEGIN

SELECT \* INTO v\_row FROM EMPLOYEES WHERE EMPLOYEE\_ID = 100;

v\_name := v\_row.FIRST\_NAME;

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

END;

**✅ 5. Cursors Can Be Used as Parameters**

* You can **pass cursors as parameters to procedures**, giving you a lot of flexibility for result set processing.

**✅ 6. PL/SQL Is Procedural but Still Runs on the Oracle Engine**

* Unlike SQL which is declarative, PL/SQL **executes line-by-line**, yet **runs inside the Oracle database engine** — which makes it faster than most external app-layer logic.
* You don't need to pull data out into Java/Python — you can process logic right inside the DB.

**✅ 7. Autonomous Transactions Can Be Created Using PRAGMA AUTONOMOUS\_TRANSACTION**

* You can commit or roll back operations **independently** of the main transaction, useful in logging errors or audits without affecting parent logic.

**✅ 8. You Can Call Java from PL/SQL**

* Oracle allows calling **Java stored procedures** from PL/SQL if your database has the Java engine enabled.

**✅ 9. Variables Declared in DECLARE Block Cannot Be Used Outside**

* PL/SQL supports **block-level scope**, and variables **cannot** leak outside the block they are declared in — just like local variables in modern programming languages.

**✅ 10. You Can Simulate Object-Oriented Features Using PL/SQL Packages**

* PL/SQL packages allow **encapsulation**, **overloading**, and **modularity**— making your logic more maintainable and structured, just like OOP languages.

# **PL/SQL Training - Week 1: SQL Fundamentals (Trainer Manual)**

## ****Day 2: SQL - SELECT Statements, Filtering, and Operators (PL/SQL Context)****

**Training Duration:** 9:45 AM – 6:00 PM

### ****Objective:****

Introduce SELECT queries, filtering, sorting, and logical operations in the context of PL/SQL. Enable learners to extract data using anonymous PL/SQL blocks and apply business logic.

### ****Session Flow & Content****

**9:45 AM – 11:00 AM: SELECT Basics in PL/SQL**

#### ****Using SELECT INTO in PL/SQL****

The SELECT INTO statement is used to fetch data into PL/SQL variables.

##### **Syntax:**

SELECT column1 INTO variable\_name FROM table\_name WHERE condition;

##### **Example:**

DECLARE

v\_name VARCHAR2(50);

BEGIN

SELECT name INTO v\_name FROM Employee WHERE emp\_id = 101;

DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || v\_name);

END;

##### **Trainer Script:**

"Unlike standard SQL, in PL/SQL, you must store the result of SELECT into a variable. If the query returns more than one row, it raises TOO\_MANY\_ROWS error."

##### **Real-World Scenario:**

"A hospital admin system fetching a patient name based on patient ID inside a PL/SQL procedure."

**☕ 11:00 AM – 11:15 AM: Tea/Coffee Break**

**🕚 11:15 AM – 1:00 PM: Filtering with WHERE, Logical Operators in PL/SQL**

#### ****Comparison and Logical Operators in PL/SQL:****

Used inside SELECT or IF statements

* =, <>, >, <, >=, <=
* AND, OR, NOT

##### **Example with Conditional Logic:**

DECLARE

v\_salary NUMBER;

BEGIN

SELECT salary INTO v\_salary FROM Employee WHERE emp\_id = 102;

IF v\_salary > 50000 THEN

DBMS\_OUTPUT.PUT\_LINE('High salary employee');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Moderate salary employee');

END IF;

END;

##### **Use Case:**

"Determine if a doctor qualifies for bonus based on salary fetched via SELECT INTO."

**🍽️ 1:00 PM – 2:00 PM: Lunch Break**

**🕑 2:00 PM – 3:30 PM: LIKE, IN, BETWEEN in PL/SQL**

#### ****LIKE with SELECT INTO:****

DECLARE

v\_name VARCHAR2(50);

BEGIN

SELECT name INTO v\_name FROM Patient WHERE name LIKE 'S%';

DBMS\_OUTPUT.PUT\_LINE(v\_name);

END;

#### ****BETWEEN and IN:****

DECLARE

v\_count NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_count FROM Employee

WHERE salary BETWEEN 40000 AND 70000

AND department IN ('IT', 'HR');

DBMS\_OUTPUT.PUT\_LINE('Count: ' || v\_count);

END;

##### **Real Scenario: Hospital Filtering**

"Filter patients admitted between two dates using BETWEEN in PL/SQL."

DECLARE

v\_count NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_count FROM Patient

WHERE AdmissionDate BETWEEN TO\_DATE('01-JAN-2024','DD-MON-YYYY') AND TO\_DATE('31-MAR-2024','DD-MON-YYYY');

DBMS\_OUTPUT.PUT\_LINE('Admitted in Q1: ' || v\_count);

END;

**☕ 3:30 PM – 3:45 PM: Tea/Coffee Break**

**🕒 3:45 PM – 5:30 PM: Lab Practice - SELECT INTO and Conditional Logic**

### ****Lab Task:****

1. Retrieve employee name using SELECT INTO
2. Fetch and compare salary – print based on high or low salary
3. Use IN clause in SELECT INTO
4. Use BETWEEN to count admissions
5. Combine multiple filters inside a PL/SQL block

**🕔 5:30 PM – 6:00 PM: Recap & Case Study Discussion**

### ****Mini Case Study: Automated Bonus System in Hospital Payroll****

**Objective:** Automate bonus eligibility check using SELECT INTO and conditional logic

**Requirements:**

* Fetch salary using SELECT INTO
* Use conditional IF to determine eligibility
* Print bonus message

DECLARE

v\_salary NUMBER;

BEGIN

SELECT salary INTO v\_salary FROM Doctor WHERE doctor\_id = 201;

IF v\_salary > 90000 THEN

DBMS\_OUTPUT.PUT\_LINE('Eligible for 10% bonus');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Not eligible');

END IF;

END;

**Trainer Instructions:**

* Assign bonus-check logic to groups
* Demonstrate IF-ELSE structure and exception handling if needed