KARNAVATI UNIVERSITY | UIT

Unitedworld Institute Of Technology

B.Tech. Computer Science & Engineering

Semester : 3rd

Introduction To Database Management System
Course Code: 71203002003

Unit – 3 : STRUCTURED QUERY LANGUAGE - SQL AND PL/SQL

Prepared by:
Mr. Utsav Kapadiya
Assistant Professor (UIT)

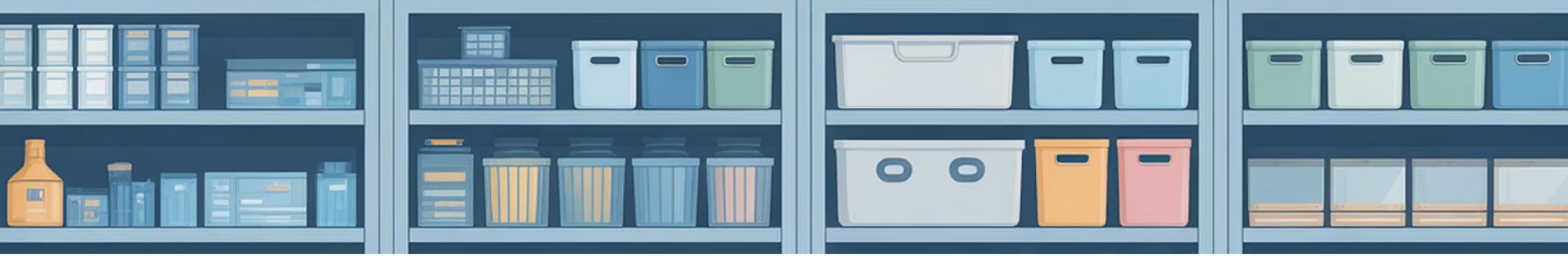
PL/SQL Basics: Data Types, Variables, Literals, and Constants

Master the fundamental building blocks of PL/SQL programming. This guide covers essential concepts that form the foundation of database programming: data types for organizing information, variables for storing temporary data, literals for fixed values, and constants for unchangeable values. Whether you're writing your first PL/SQL block or refining your skills, understanding these core elements is crucial for effective database development.

Common PL/SQL Data Types

A data type defines what kind of data a variable can store—numbers, text, dates, and more. Choosing the right data type ensures data integrity and optimal performance in your PL/SQL programs.

Category	Data Type	Description	Example
Numeric	NUMBER(p,s)	Stores numbers with precision and scale	NUMBER(5,2) → 999.99
	BINARY_INTEGER	Whole numbers (used for counting)	10, -20
Character	CHAR(n)	Fixed-length string	'A', 'YES'
	VARCHAR2(n)	Variable-length string	'John'
Date/Time	DATE	Stores date and time	01-JAN-2025
Boolean	BOOLEAN	TRUE / FALSE / NULL	flag BOOLEAN := TRUE;
Large Objects	CLOB, BLOB	Large text/binary data	File storage
Reference	%TYPE, %ROWTYPE	Uses type from a table/variable	student.name%TYPE



Understanding Data Types Through Analogy

CHAR: Fixed-Size Box

Think of CHAR as a rigid container with a set size. Whether you store 'A' or 'YES', it always occupies the same space—like a safety deposit box that's always the same size regardless of what you put inside.

VARCHAR2: Flexible Box

VARCHAR2 is like an expandable bag that adjusts to fit your content. Store 'Hi' or 'Hello World'—it only uses the space it needs, making it memory-efficient for variable-length text.

NUMBER: Numeric ontainer

NUMBER is specialized for mathematical data. Like a calculator drawer, it's designed specifically for storing and manipulating numeric values with precision and scale control.

Data Type Usage in Action

```
DECLARE  v_id       NUMBER(5);  v_name     VARCHAR2(30);  v_marks
NUMBER(5,2);  v_dob      DATE;  v_isPassed BOOLEAN := TRUE;BEGIN
v_id := 101;  v_name := 'Aarav';  v_marks := 89.75;  v_dob :=
'12-FEB-2004';    DBMS_OUTPUT.PUT_LINE(    'Student: ' || v_name
||     ' Marks: ' || v_marks  );END;/
```

Code Breakdown

This example demonstrates proper data type selection for different kinds of information:

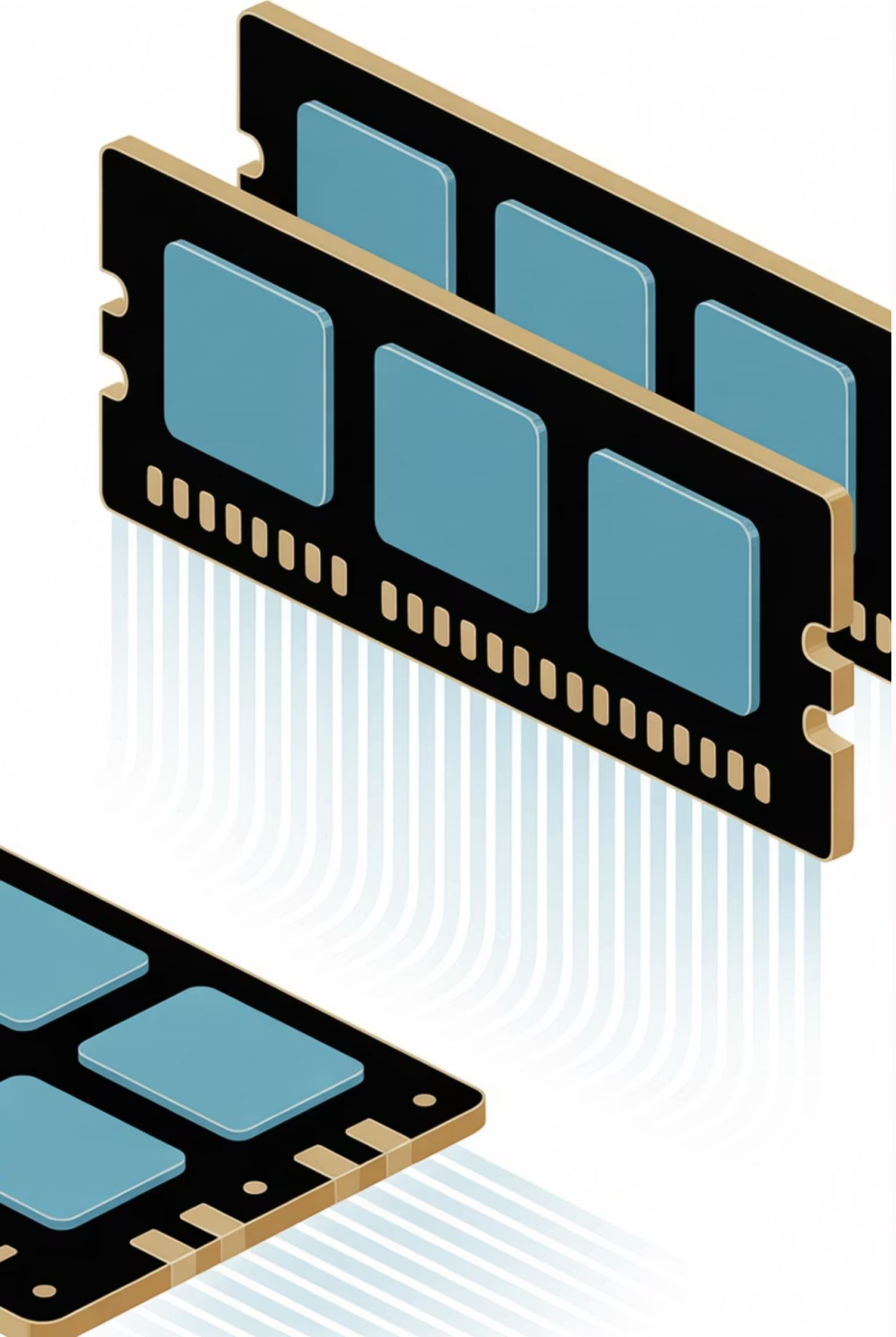
v_id: NUMBER(5) for student ID (up to 99999)

v_name: VARCHAR2(30) for flexible name storage

v_marks: NUMBER(5,2) for decimal precision (like 89.75)

v_dob: DATE for birthdate tracking

v_isPassed: BOOLEAN for pass/fail status



Variables in PL/SQL

Named Memory Location

A variable is a container in memory with a specific name that temporarily stores data while your program executes. Think of it as a labeled storage space.

Mutable Values

Unlike constants, variables can be updated throughout program execution. Assign, modify, and reassign values as your business logic requires.

Temporary Storage

Variables exist only during program execution. Once your PL/SQL block completes, the variable values are discarded unless explicitly saved.

Rules for Declaring Variables

Follow these essential naming conventions and rules to write clean, error-free PL/SQL code:

1

Start with a Letter

Variable names must begin with a letter (A–Z or a–z). Numbers and special characters cannot be the first character.

2

Alphanumeric Characters

After the first letter, you can include letters, digits (0–9), or underscores (_). Keep names descriptive but concise.

3

Avoid Reserved Keywords

Never use PL/SQL keywords like SELECT, BEGIN, END, IF, or LOOP as variable names—this will cause syntax errors.

4

Declare in DECLARE Section

All variables must be declared in the DECLARE section before the BEGIN keyword. This ensures proper scope and initialization.

Variable Declaration Syntax

General Format

```
variable_name datatype
[DEFAULT value]   [NOT NULL];
```

The syntax is straightforward: provide a name, specify the data type, and optionally set a default value or enforce NOT NULL constraints.

Syntax Components

variable_name: Descriptive identifier following naming rules
datatype: Defines what kind of data can be stored
DEFAULT value: Optional initial value assignment
NOT NULL: Constraint requiring a value at all times

Variable Declaration Example

```
DECLARE  emp_id     NUMBER := 101;  emp_name   VARCHAR2(20) := 'Riya';  emp_salary NUMBER DEFAULT 50000;BEGIN
DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp_id);  DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_name);
DBMS_OUTPUT.PUT_LINE('Salary: ' || emp_salary);END;/
```

Variable Initialization

Variables can be assigned initial values during declaration using the `:=` operator or the `DEFAULT` keyword. Both approaches achieve the same result.

Temporary Storage

These variables exist only during program execution. They store employee data temporarily for processing and display operations.

String Concatenation

The `||` operator concatenates strings and variables for output. This is essential for building meaningful messages and reports.

Using %TYPE for Smart Declarations

Inherit Data Types

Instead of manually specifying data types, use %TYPE to automatically inherit the data type from a table column. This creates a dynamic link between your variable and the database schema.

Key Advantages

- Automatic synchronization with database changes
- Reduced maintenance overhead
- Guaranteed type compatibility
- Fewer errors from type mismatches

Example Code

```
DECLARE  v_name student.name%TYPE;BEGIN  SELECT name   INTO
v_name   FROM student   WHERE id = 101;    DBMS_OUTPUT.PUT_LINE(
'Student Name: ' || v_name  );END;/
```

If the student.name column changes from VARCHAR2(20) to VARCHAR2(50), your v_name variable automatically adapts—no code changes needed!

Literals in PL/SQL

Definition

A literal is a fixed value that appears directly in your code. Unlike variables, literals never change during program execution—they represent constant, hard-coded values.

Where You Find Literals

Literals appear throughout PL/SQL code in assignments, comparisons, calculations, and output statements. They're the raw data values your program works with—numbers like 100, strings like 'Hello', or dates like '12-FEB-2025'.

Types of Literals

Numeric Literals

Examples: 100, -45.7, 3.14159

Whole numbers or decimal values used directly in calculations and assignments. No quotes needed.

Character Literals

Examples: 'John', 'A', 'Hello World'

String values enclosed in single quotes. Can be single characters or longer text strings.

Date Literals

Examples: '12-FEB-2025', '01-JAN-2024'

Must be enclosed in quotes and formatted according to your database's date format settings.

Boolean Literals

Examples: TRUE, FALSE, NULL

Logical values used in conditional statements and flags. No quotes required for these keywords.

Literals in Action

```
DECLARE  v_total NUMBER := 500;  v_bonus NUMBER := 50;BEGIN
DBMS_OUTPUT.PUT_LINE(    'Total Marks: ' || v_total + v_bonus  );
DBMS_OUTPUT.PUT_LINE(    'Result Date: ' || '12-FEB-2025'  );END;/
```

Identifying Literals

In this example, the following are literals:

`500` - numeric literal

`50` - numeric literal

`'12-FEB-2025'` - date literal

These values are fixed and cannot be modified during execution. They're hard-coded directly into the program logic.

Constants in PL/SQL

What Makes Constants Special

Similar to Variables

Constants are declared just like variables with a name and data type. They occupy memory and can be used in expressions and calculations throughout your code.

One-Time Assignment

The critical difference: constants must be assigned a value during declaration. Once set, this value is locked forever for the duration of program execution.

Immutable Values

Any attempt to modify a constant after declaration results in a compilation error. This immutability protects critical values from accidental changes.

Constant Declaration Syntax

Format

```
CONSTANT_NAME CONSTANT   datatype := value;
```

The `CONSTANT` keyword signals to PL/SQL that this value cannot be changed. The assignment operator `:=` must be used at declaration time.

Key Rules

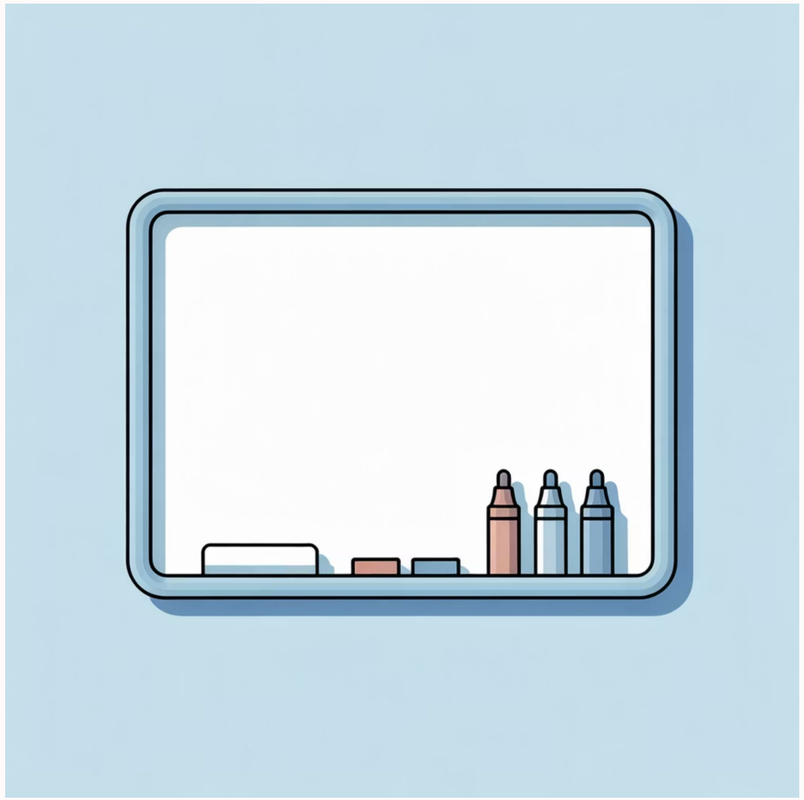
Must include the `CONSTANT` keyword

- Value assignment is mandatory at declaration
- Cannot be modified after initialization
- Use uppercase names by convention
- Perfect for configuration values and fixed business rules

Constants Example

```
DECLARE PI CONSTANT NUMBER := 3.14159; BONUS_RATE CONSTANT NUMBER := 0.10; salary NUMBER := 50000;BEGIN DBMS_OUTPUT.PUT_LINE(
'Bonus: ' || salary * BONUS_RATE );  -- PI := 3.14;
```

Variable vs. Constant Analogy



Variable = Whiteboard

A variable is like a whiteboard in a conference room. You can write information, erase it, and write something new as many times as needed. The content is temporary and flexible—perfect for values that change during program execution.

- Reusable and flexible
- Can be erased and rewritten
- Changes throughout program flow

Constant = Poster

A constant is like a printed poster on the wall. Once created and hung, the information is permanent and unchangeable. It stays the same no matter what happens—ideal for fixed values like mathematical constants or configuration settings.

- Fixed and permanent
- Cannot be modified once set
- Reliable throughout execution

Quick Reference Summary

Concept	Meaning	Example
Data Type	Defines the type of data a variable can store	NUMBER, VARCHAR2, DATE, BOOLEAN
Variable	Named memory location that can change during execution	v_name VARCHAR2(20);
Literal	Fixed, direct value written in code	'John', 500, '12-FEB-2025'
Constant	Fixed variable that cannot be changed after declaration	PI CONSTANT NUMBER := 3.14;

Understanding these four fundamental concepts is essential for writing effective PL/SQL code. Data types ensure data integrity, variables provide flexibility, literals offer direct values, and constants protect critical values from modification.

Real-Life Analogy Summary

Data Type = Container Type

Just as you need different containers for different items—a glass for water, a box for books, a jar for cookies—data types define what kind of information can be stored. Each type has specific characteristics and purposes.

Literal = Fixed Value

Think of writing "100" directly on a piece of paper—it's a concrete, unchanging value. Literals are the raw data you write directly into your code without storing them in variables first.

Variable = Reusable Container

Like a reusable water bottle that can be emptied and refilled throughout the day, variables store values that can be changed multiple times during program execution. Pour out the old, pour in the new.

Constant = Fixed Rule

Constants are like the law of gravity—unchangeable rules that remain the same no matter what. Once defined, constants like PI or tax rates stay fixed throughout your program's execution.

Putting It All Together

```
DECLARE  v_name VARCHAR2(20) := 'Aarav';  v_age NUMBER(3) := 20;
v_course CONSTANT VARCHAR2(10) := 'BCA';BEGIN
DBMS_OUTPUT.PUT_LINE('Name: ' || v_name);
DBMS_OUTPUT.PUT_LINE('Age: ' || v_age);
DBMS_OUTPUT.PUT_LINE('Course: ' || v_course);END;/
```

Expected Output:

```
Name: AaravAge: 20Course: BCA
```

Complete Example Breakdown

This compact example demonstrates all four concepts working together:

Data Types: VARCHAR2, NUMBER define what each variable stores
Variables: v_name and v_age can be modified later if needed

Constant: v_course is locked as 'BCA' and cannot change

Literals: 'Aarav', 20, and 'BCA' are fixed values in the code

This pattern forms the foundation of every PL/SQL program you'll write. Master these basics, and you're ready for more advanced database programming!

Thank You!