In *Oracle SQL*\Plus*, variables created with the **DEFINE** command are **always treated as substitution variables*** and are **always typed as character strings**. SQL\Plus does **not** assign numeric, date, or other data types to DEFINE variables.

◆ **Key idea**

DEFINE creates a *text substitution* variable. SQL*Plus simply replaces &varname with the text you defined before sending the statement to Oracle.

◆ **Example**

sql

DEFINE myvar = 123

SELECT &myvar + 10 FROM dual;

Even though 123 looks numeric, SQL*Plus still treats it as **text** and substitutes it directly into the SQL statement. Oracle then interprets the resulting SQL.

◆ **How SQL*Plus variables differ**

Feature	DEFINE	ACCEPT	Bind Variables (VARIABLE)
Type	Always text	Can prompt user; still text	Real Oracle datatypes
Scope	SQL*Plus only	SQL*Plus only	Stored in SQL engine
Usage	&var or &&var	&var	:var

◆ **If you need typed variables**

Use **bind variables** instead:

sql

VARIABLE num NUMBER

EXEC :num := 123;

SELECT :num + 10 FROM dual;

◆ **Summary**

- DEFINE variables = **text substitution only**
- They have **no datatype**

- Oracle assigns meaning only after substitution

If you want, I can show you how to force numeric or date behavior using DEFINE, or how to convert DEFINE variables inside SQL.

additional pl sql statements from oracle 10G to 19c

Below is a clean, benchmark-friendly guide to **what PL/SQL gained from Oracle 10g → 11g → 12c → 18c → 19c**, grounded in the search results you triggered. The search results focus heavily on 19c new features, so I've woven those in with well-established PL/SQL evolution across versions. Citations appear where the search results explicitly mention features.

📘 Evolution of PL/SQL Features from Oracle 10g → 19c

A modular, layered breakdown you can use as a reference or benchmarking matrix.

🟦 Oracle 10g → 11g

Major PL/SQL Enhancements

- **Native compilation improvements** (no external C compiler needed)
- **PL/SQL function result cache**
- **Simple integer** datatype (faster than NUMBER)
- **Fine-grained dependency tracking**
- **Trigger enhancements** (compound triggers introduced in 11g)
- **CONTINUE** statement added
- **Improved error handling** (e.g., PRAGMA EXCEPTION_INIT flexibility)

🟩 Oracle 11g → 12c

Key PL/SQL Additions

- **WITH clause enhancements** (PL/SQL functions in WITH)
- **ACCESSIBLE BY** clause for encapsulation
- **UTL_CALL_STACK** for better stack inspection
- **PL/SQL in CDB/PDB architecture**
- **Implicit statement results** (returning result sets without ref cursors)
- **BEQUEATH CURRENT_USER** for invoker-rights views

- **Extended VARCHAR2, NVARCHAR2, RAW to 32K**
- **Function inlining** (PRAGMA INLINE)

🟧 **Oracle 12c → 18c**

Notable PL/SQL Improvements

- **Qualified expressions** (constructor-like syntax for records/collections)
- **New JSON support** (PL/SQL object types for JSON)
- **Improved SQL/PLSQL interoperability**
- **Better optimizer feedback for PL/SQL**
- **Enhanced security** (e.g., stronger definer/invoker rights controls)

🟥 **Oracle 18c → 19c**

Search results highlight several 19c PL/SQL-related features. The most important PL/SQL-specific additions are:

1. SQL Macros (19c)

A major new feature allowing reusable SQL fragments. Documented in the PL/SQL Language Reference changes.

Two types:

- **SCALAR SQL macros**
- **TABLE SQL macros**

Example:

sql

```
CREATE FUNCTION add_tax(x NUMBER)
  RETURN VARCHAR2 SQL_MACRO IS
BEGIN
  RETURN 'x * 1.07';
END;
```

2. Polymorphic Table Functions (PTFs) – simplified USING clause

19c allows PTF implementation methods in the same package without requiring USING clause.

3. General PL/SQL improvements in 19c

- Better integration with **In-Memory Column Store** (from 19c new features list)
- Performance and diagnosability enhancements
- More stable long-term support (19c is the terminal release of the 12c family)

🧩 **Summary Table (10g → 19c)**

Version	Major PL/SQL Additions
10g	Native compilation revamp, improved collections, early bulk processing
11g	Result cache, compound triggers, CONTINUE, simple integer
12c	32K VARCHAR2, ACCESSIBLE BY, implicit results, UTL_CALL_STACK, inlining
18c	Qualified expressions, JSON object types, improved interoperability
19c	SQL Macros, simplified PTF USING clause, performance/diagnosability improvements

Sources: