**Predefined PL/SQL Datatypes**

Predefined PL/SQL data types are grouped into composite, LOB, reference, and scalar type categories.

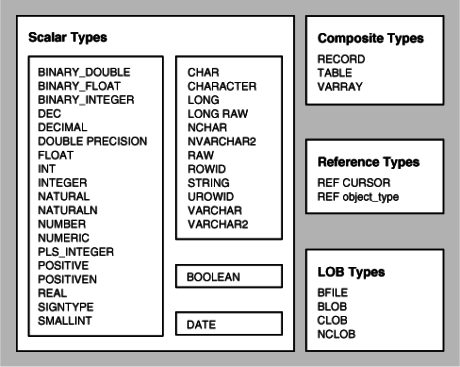
* A composite type has internal components that can be manipulated individually, such as the elements of an array, record, or table.
* A LOB type holds values, called lob locators, that specify the location of large objects, such as text blocks or graphic images, that are stored separately from other database data. LOB types include BFILE, BLOB, CLOB, and NCLOB.
* A reference type holds values, called pointers, that designate other program items. These types include REF CURSORS and REFs to object types.
* A scalar type has no internal components. It holds a single value, such as a number or character string. The scalar types fall into four families, which store number, character, Boolean, and date/time data. The scalar families with their datatypes are:

**1. PL/SQL Number Types**  
BINARY\_DOUBLE, BINARY\_FLOAT, BINARY\_INTEGER, DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INT, INTEGER, NATURAL, NATURALN, NUMBER, NUMERIC, PLS\_INTEGER, POSITIVE, POSITIVEN, REAL, SIGNTYPE, SMALLINT

**2. PL/SQL Character and String Types and PL/SQL National Character Types**  
CHAR, CHARACTER, LONG, LONG RAW, NCHAR, NVARCHAR2, RAW, ROWID, STRING, UROWID, VARCHAR, VARCHAR2 Note that the LONG and LONG RAW data types are supported only for backward compatibility Information.

**3. PL/SQL Boolean Types**  
BOOLEAN

**4. PL/SQL Date, Time, and Interval Types**  
DATE, TIMESTAMP, TIMESTAMP WITH TIMEZONE, TIMESTAMP WITH LOCAL TIMEZONE, INTERVAL YEAR TO MONTH, INTERVAL DAY TO SECOND

[](https://cdn.intellipaat.com/wp-content/uploads/2015/09/data-type.png)  
***Go for this in-depth job-oriented***[***PL/SQL Training***](https://intellipaat.com/pl-sql-online-training/)***Course now!***

**PL/SQL Subtypes**  
Each **PL/SQL base type** specifies a set of values and a set of operations applicable to items of that type. Subtypes specify the same set of operations as their base type, but only a subset of its values. A subtype does not introduce a new type; rather, it places an optional constraint on its base type.  
**Subtypes** can increase reliability, provide compatibility with ANSI/ISO types, and improve readability by indicating the intended use of constants and variables. PL/SQL predefines several subtypes in package STANDARD. For example, PL/SQL predefines the subtypes CHARACTER and INTEGER as follows:

SUBTYPE CHARACTER IS CHAR;

SUBTYPE INTEGER IS NUMBER(38,0); -- allows only whole numbers

The subtype CHARACTER specifies the same set of values as its base type CHAR, so CHARACTER is an unconstrained subtype. But, the subtype INTEGER specifies only a subset of the values of its base type NUMBER, so INTEGER is a constrained subtype.

* **Defining Subtypes**

You can define your own subtypes in the declarative part of any PL/SQL block, subprogram, or package using the syntax

SUBTYPE subtype\_name IS base\_type[(constraint)] [NOT NULL];

where subtype\_name is a type specifier used in subsequent declarations, base\_type is any scalar or user-defined PL/SQL datatype, and constraint applies only to base types that can specify precision and scale or maximum size.

* **Using Subtypes**

After you define a subtype, you can declare items of that type. In the following example, you declare a variable of type Counter. Notice how the subtype name indicates the intended use of the variable.

DECLARE

SUBTYPE Counter IS NATURAL;

rows Counter;

***Check out the top***[***PL/SQL Interview Questions***](https://intellipaat.com/blog/interview-question/pl-sql-interview-questions/)***to learn what is expected from PL/SQL professionals!***

**Converting PL/SQL Datatypes**  
Sometimes it is necessary to convert a value from one datatype to another. For example, to use a DATE value in a report, you must convert it to a character string. PL/SQL supports both explicit and implicit (automatic) datatype conversion.

* **Explicit Conversion**

Using explicit conversions, particularly when passing parameters to subprograms, can avoid unexpected errors or wrong results. For example, the TO\_CHAR function lets you specify the format for a[DATE](https://intellipaat.com/blog/tutorial/sql-tutorial/sql-formatting/) value, rather than relying on language settings in the database. Including an arithmetic expression among strings being concatenated with the || operator can produce an error unless you put parentheses or a call to TO\_CHAR around the entire arithmetic expression.

* **Implicit Conversion**

When it makes sense, PL/SQL can convert the data type of a value implicitly. For example, you can pass a numeric literal to a subprogram that expects a string value, and the subprogram receives the string representation of the number.