## CHEAT SHEETS

**PL/SQL** Cheatsheet A concise reference for PL/SQL, covering syntax, data types, control structures, and common operations for Oracle database programming.



# **PL/SQL Basics**

#### **Block Structure**

PL/SQL code is organized into blocks. A block can be anonymous or named (a stored procedure, function, or trigger).	NUMB ER	N a
DECLARE Declaration of variables, types, etc. BEGIN Executable statements	VARC HAR2( size )	a N r
EXCEPTION Exception handling (optional)	DAT	S
END; /	BOOL	5
The <b>DECLARE</b> section is optional and used to define variables, constants, cursors, and user-defined types.	CLO B	a
Variables are declared with a name, data type, and optional initial value:		
<pre>variable_name data_type := initial_value;</pre>		
The <b>BEGIN</b> section contains the executable statements. This is where the main logic of the PL/SQL block resides. Statements are executed sequentially.		
The <b>EXCEPTION</b> section is optional and provides		

The error handling. When an exception occurs, the control is transferred to this section.

Exception handlers are defined for specific exceptions or for all exceptions.

## Data Types

NUMB ER	Numeric data type for storing integers and floating-point numbers. NUMBER(p,s) where p is precision and s is scale.
VARC HAR2( size )	Variable-length character string with a maximum size specified in bytes.
DAT	Stores date and time values.
BOOL EAN	Stores logical values: TRUE , FALSE , or NULL .
CLO B	Character Large Object, for storing large amounts of text data (up to 4GB).

#### Variables and Constants

Variables are declared in the **DECLARE** section and are used to store data during the execution of the PL/SQL block.

variable\_name data\_type [:= initial\_value];

Constants are declared using the **CONSTANT** keyword. Their value cannot be changed after initialization.

constant\_name CONSTANT data\_type := value;

Referencing Database Columns:

variable\_name

table\_name.column\_name%TYPE;

This declares a variable with the same data type as a specified column in a database table.

## **Control Structures**

# Conditional Statements

Conditional Statements
IF-THEN-ELSE
IF condition THEN
Statements to execute if the
condition is true
[ELSIF condition THEN
Statements to execute if the
condition is true]
[ELSE
Statements to execute if all
conditions are false]
END IF;
CASE Statement
CASE selector
WHEN value1 THEN
Statements for value1
WHEN value2 THEN
Statements for value2
[ELSE
Default statements]

END CASE;

## Looping Structures

Basic LOOP	LOOP Statements to execute EXIT WHEN condition; END LOOP;
WHILE LOOP	WHILE condition LOOP Statements to execute while the condition is true END LOOP;
FOR LOOP	FOR index IN [REVERSE] lower_bound upper_bound LOOP Statements to execute END LOOP; The REVERSE keyword iterates in descending order.

#### Cursors

Cursors allow you to process rows returned by a SQL query one at a time.

#### **Explicit Cursors**

DECLARE

CURSOR cursor\_name IS

SELECT column1, column2 FROM

table\_name WHERE condition;

record\_name cursor\_name%ROWTYPE;

#### BEGIN

OPEN cursor\_name;

LOOP

FETCH cursor\_name INTO record\_name;

EXIT WHEN cursor\_name%NOTFOUND;

-- Process record\_name

END LOOP;

CLOSE cursor\_name;

END;

## **Cursor Attributes**

- **%FOUND**: Boolean attribute that is TRUE if a fetch returns a row, FALSE otherwise.
- **%NOTFOUND**: Boolean attribute that is TRUE if a fetch does not return a row, FALSE otherwise.
- **%ISOPEN** : Boolean attribute that is TRUE if the cursor is open, FALSE otherwise.
- %ROWCOUNT : Number of rows fetched from the cursor.

## **Exception Handling**

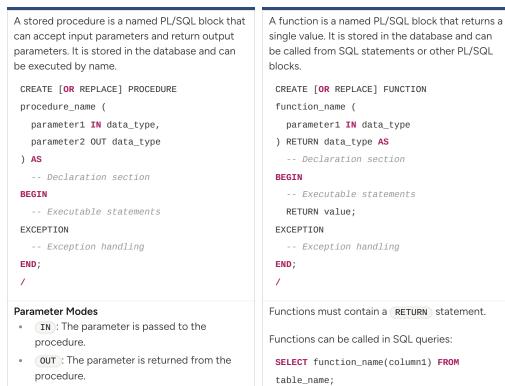
```
Exception Handling Block
 The EXCEPTION section handles errors that
 occur during the execution of the BEGIN
 section.
  EXCEPTION
    WHEN exception_name THEN
       -- Handle the exception
    WHEN OTHERS THEN
       -- Handle any other exception
  END:
 Predefined Exceptions: NO_DATA_FOUND ,
 TOO_MANY_ROWS , INVALID_CURSOR ,
 ZERO_DIVIDE , etc.
 User-Defined Exceptions: Can be declared and
 raised explicitly using RAISE statement.
  DECLARE
    my_exception EXCEPTION;
  BEGIN
    IF condition THEN
      RAISE my_exception;
    END IF:
  EXCEPTION
    WHEN my_exception THEN
       -- Handle my_exception
```

END;

## **Stored Procedures and Functions**

#### Stored Procedures

#### Functions



• **IN OUT**: The parameter is passed to the procedure and can be returned with a modified value.

# Common Exceptions

NO_DATA _FOUND	Raised when a <b>SELECT</b> statement returns no rows.	
TOO_MAN Y_ROWS	Raised when a <b>SELECT INTO</b> statement returns more than one row.	
INVALID _CURSOR	Raised when an invalid cursor operation is performed.	
ZERO_DI VIDE	Raised when an attempt is made to divide by zero.	
DUP_VAL _ON_INDE X	Raised when attempting to insert a duplicate value into a unique index.	

## RAISE\_APPLICATION\_ERROR

Used to return user-defined error messages from a PL/SQL block to the calling environment.

RAISE\_APPLICATION\_ERROR (error\_number, message [, {TRUE | FALSE}]);

- error\_number : An integer between -20000 and -20999.
- message : The error message string (up to 2048 bytes).
- **TRUE**: Error is placed on the stack of previous errors.
- FALSE (default): Error replaces any previous errors.

# Calling Stored Procedures and Functions

Calling a Stored Procedure	DECLARE
Procedure	output_variable
	data_type;
	BEGIN
	procedure_name(input_va
	<pre>lue, output_variable);</pre>
	Use
	output_variable
	END;
	/
Calling a	
Function	DECLARE
	return_value
	data_type;
	BEGIN
	return_value :=
	function_name(input_val
	ue);
	Use return_value
	END;
	/
	Or directly in a SQL query:
	SELECT
	<pre>function_name(column1)</pre>
	<pre>FROM table_name;</pre>