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).

```
DECLARE
  -- Declaration of variables, types, etc.
BEGIN
  -- Executable statements
EXCEPTION
  -- Exception handling (optional)
END;
/
```

The `DECLARE` section is optional and used to define variables, constants, cursors, and user-defined types.

Variables are declared with a name, data type, and optional initial value:

```
variable_name data_type := initial_value;
```

The `BEGIN` section contains the executable statements. This is where the main logic of the PL/SQL block resides. Statements are executed sequentially.

The `EXCEPTION` section is optional and provides 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

`NUMBER`	Numeric data type for storing integers and floating-point numbers. `NUMBER(p,s)` where `p` is precision and `s` is scale.
`VARCHAR2(size)`	Variable-length character string with a maximum size specified in bytes.
`DATE`	Stores date and time values.
`BOOLEAN`	Stores logical values: `TRUE`, `FALSE`, or `NULL`.
`CLOB`	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

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;
```

Common Exceptions

`NO_DATA_FOUND`	Raised when a `SELECT` statement returns no rows.
`TOO_MANY_ROWS`	Raised when a `SELECT INTO` statement returns more than one row.
`INVALID_CURSOR`	Raised when an invalid cursor operation is performed.
`ZERO_DIVIDE`	Raised when an attempt is made to divide by zero.
`DUP_VAL_ON_INDEX`	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.

Stored Procedures and Functions

Stored Procedures

A stored procedure is a named PL/SQL block that can accept input parameters and return output parameters. It is stored in the database and can be executed by name.

```
CREATE [OR REPLACE] PROCEDURE procedure_name (
  parameter1 IN data_type,
  parameter2 OUT data_type
) AS
  -- Declaration section
BEGIN
  -- Executable statements
EXCEPTION
  -- Exception handling
END;
/
```

Parameter Modes

- `IN`: The parameter is passed to the procedure.
- `OUT`: The parameter is returned from the procedure.
- `IN OUT`: The parameter is passed to the procedure and can be returned with a modified value.

Functions

A function is a named PL/SQL block that returns a single value. It is stored in the database and can be called from SQL statements or other PL/SQL blocks.

```
CREATE [OR REPLACE] FUNCTION function_name (
  parameter1 IN data_type
) RETURN data_type AS
  -- Declaration section
BEGIN
  -- Executable statements
  RETURN value;
EXCEPTION
  -- Exception handling
END;
/
```

Functions must contain a `RETURN` statement.

Functions can be called in SQL queries:

```
SELECT function_name(column1) FROM table_name;
```

Calling Stored Procedures and Functions

Calling a Stored Procedure

```
DECLARE
  output_variable data_type;
BEGIN

procedure_name(input_value, output_variable);
  -- Use output_variable
END;
/
```

Calling a Function

```
DECLARE
  return_value data_type;
BEGIN
  return_value := function_name(input_value);
  -- Use return_value
END;
/
```

Or directly in a SQL query:

```
SELECT function_name(column1) FROM table_name;
```