PL/SQL Tutorial

[](https://training.javatpoint.com/)

PL/SQL tutorial provides basic and advanced concepts of SQL. Our PL/SQL tutorial is designed for beginners and professionals.

PL/SQL is a block structured language that can have multiple blocks in it.

Our PL/SQL tutorial includes all topics of PL/SQL language such as conditional statements, loops, arrays, string, exceptions, collections, records, triggers, functions, procedures, cursors etc. There are also given PL/SQL interview questions and quizzes to help you better understand the PL/SQL language.

SQL stands for Structured Query Language i.e. used to perform operations on the records stored in database such as inserting records, updating records, deleting records, creating, modifying and dropping tables, views etc.

What is PL/SQL

PL/SQL is a block structured language. The programs of PL/SQL are logical blocks that can contain any number of nested sub-blocks. Pl/SQL stands for "Procedural Language extension of SQL" that is used in Oracle. PL/SQL is integrated with Oracle database (since version 7). The functionalities of PL/SQL usually extended after each release of Oracle database. Although PL/SQL is closely integrated with SQL language, yet it adds some programming constraints that are not available in SQL.

PL/SQL Functionalities

PL/SQL includes procedural language elements like conditions and loops. It allows declaration of constants and variables, procedures and functions, types and variable of those types and triggers. It can support Array and handle exceptions (runtime errors). After the implementation of version 8 of Oracle database have included features associated with object orientation. You can create PL/SQL units like procedures, functions, packages, types and triggers, etc. which are stored in the database for reuse by applications.

With PL/SQL, you can use SQL statements to manipulate Oracle data and flow of control statements to process the data.

The PL/SQL is known for its combination of data manipulating power of SQL with data processing power of procedural languages. It inherits the robustness, security, and portability of the Oracle Database.

PL/SQL is not case sensitive so you are free to use lower case letters or upper case letters except within string and character literals. A line of PL/SQL text contains groups of characters known as lexical units. It can be classified as follows:

* Delimeters
* Identifiers
* Literals
* Comments

# PL/SQL Variables

A variable is a meaningful name which facilitates a programmer to store data temporarily during the execution of code. It helps you to manipulate data in PL/SQL programs. It is nothing except a name given to a storage area. Each variable in the PL/SQL has a specific data type which defines the size and layout of the variable's memory.

A variable should not exceed 30 characters. Its letter optionally followed by more letters, dollar signs, numerals, underscore etc.

#### **1. It needs to declare the variable first in the declaration section of a PL/SQL block before using it.**

#### **2. By default, variable names are not case sensitive. A reserved PL/SQL keyword cannot be used as a variable name.**

## How to declare variable in PL/SQL

You must declare the PL/SQL variable in the declaration section or in a package as a global variable. After the declaration, PL/SQL allocates memory for the variable's value and the storage location is identified by the variable name.

**Syntax for declaring variable:**

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

Following is the syntax for declaring variable:

1. variable\_name [CONSTANT] datatype [NOT NULL] [:= | **DEFAULT** initial\_value]

Here, variable\_name is a valid identifier in PL/SQL and datatype must be valid PL/SQL data type. A data type with size, scale or precision limit is called a constrained declaration. The constrained declaration needs less memory than unconstrained declaration.

**Example:**

Radius Number := 5;

Date\_of\_birth date;

### **Declaration Restrictions:**

In PL/SQL while declaring the variable some restrictions hold.

* Forward references are not allowed i.e. you must declare a constant or variable before referencing it in another statement even if it is a declarative statement.  
  val number := Total - 200;  
  Total number := 1000;  
  The first declaration is illegal because the TOTAL variable must be declared before using it in an assignment expression.
* Variables belonging to the same datatype cannot be declared in the same statement.  
  N1, N2, N3 Number;  
  It is an illegal declaration.

## Naming rules for PL/SQL variables

The variable in PL/SQL must follow some naming rules like other programming languages.

* The variable\_name should not exceed 30 characters.
* Variable name should not be the same as the table table's column of that block.
* The name of the variable must begin with ASCII letter. The PL/SQL is not case sensitive so it could be either lowercase or uppercase. For example: v\_data and V\_DATA refer to the same variables.
* You should make your variable easy to read and understand, after the first character, it may be any number, underscore (\_) or dollar sign ($).
* NOT NULL is an optional specification on the variable.

## Initializing Variables in PL/SQL

Evertime you declare a variable, PL/SQL defines a default value NULL to it. If you want to initialize a variable with other value than NULL value, you can do so during the declaration, by using any one of the following methods.

* The DEFAULT keyword
* The assignment operator

1. counter binary\_integer := 0;
2. greetings varchar2(20) **DEFAULT** 'Hello JavaTpoint';

You can also specify NOT NULL constraint to avoid NULL value. If you specify the NOT NULL constraint, you must assign an initial value for that variable.

You must have a good programming skill to initialize variable properly otherwise, sometimes program would produce unexpected result.

## Example of initilizing variable

Let's take a simple example to explain it well:

1. **DECLARE**
2. a **integer** := 30;
3. b **integer** := 40;
4. c **integer**;
5. f **real**;
6. **BEGIN**
7. c := a + b;
8. dbms\_output.put\_line('Value of c: ' || c);
9. f := 100.0/3.0;
10. dbms\_output.put\_line('Value of f: ' || f);
11. **END**;

After the execution, this will produce the following result:

Value of c: 70

Value of f: 33.333333333333333333

PL/SQL procedure successfully completed.

## Variable Scope in PL/SQL:

PL/SQL allows nesting of blocks. A program block can contain another inner block. If you declare a variable within an inner block, it is not accessible to an outer block. There are two types of variable scope:

* Local Variable: Local variables are the inner block variables which are not accessible to outer blocks.
* Global Variable: Global variables are declared in outermost block.

## Example of Local and Global variables

Let's take an example to show the usage of Local and Global variables in its simple form:

1. **DECLARE**
2. -- Global variables
3. num1 number := 95;
4. num2 number := 85;
5. **BEGIN**
6. dbms\_output.put\_line('Outer Variable num1: ' || num1);
7. dbms\_output.put\_line('Outer Variable num2: ' || num2);
8. **DECLARE**
9. -- Local variables
10. num1 number := 195;
11. num2 number := 185;
12. **BEGIN**
13. dbms\_output.put\_line('Inner Variable num1: ' || num1);
14. dbms\_output.put\_line('Inner Variable num2: ' || num2);
15. **END**;
16. **END**;
17. /

After the execution, this will produce the following result:

Outer Variable num1: 95

Outer Variable num2: 85

Inner Variable num1: 195

Inner Variable num2: 185

PL/SQL procedure successfully completed.

### **Variable Attributes:**

When you declare a PL/SQL variable to hold the column values, it must be of correct data types and precision, otherwise error will occur on execution. Rather than hard coding the data type and precision of a variable. PL/SQL provides the facility to declare a variable without having to specify a particular data type using %TYPE and %ROWTYPE attributes. These two attributes allow us to specify a variable and have that variable data type be defined by a table/view column or a PL/SQL package variable.

A % sign servers as the attribute indicator. This method of declaring variables has an advantage as the user is not concerned with writing and maintaining code.

**Following are the types of Variable Attributes in PL/SQL.**

* **%TYPE:**

The %TYPE attribute is used to declare variables according to the already declared variable or database column. It is used when you are declaring an individual variable, not a record. The data type and precision of the variable declared using %TYPE attribute is the same as that of the column that is referred from a given table. This is particularly useful when declaring variables that will hold database values. When using the %TYPE keyword, the name of the columns and the table to which the variable will correspond must be known to the user. These are then prefixed with the variable name. If some previously declared variable is referred then prefix that variable name to the %TYPE attribute.

**The syntax for declaring a variable with %TYPE is:**

1. <var\_name> <tab\_name>.<column\_name>%TYPE;

Where <column\_name> is the column defined in the <tab\_name>.

**Consider a declaration.**

SALARY EMP.SAL % TYPE;

This declaration will declare a variable SALARY that has the same data type as column SAL of the EMP table.

**Example:**

1. **DECLARE**
2. SALARY EMP.SAL % TYPE;
3. ECODE EMP.empno % TYPE;
4. **BEGIN**
5. Ecode :=&Ecode;
6. **Select** SAL **into** SALARY **from** EMP **where** EMPNO = ECODE;
7. dbms\_output.put\_line('Salary of ' || ECODE || 'is = || salary');
8. **END**;

After the execution, this will produce the following result:

Enter value for ecode: 7499

Salary of 7499 is = 1600

PL/SQL procedure successfully completed.

* **%ROWTYPE:**

The %ROWTYPE attribute is used to declare a record type that represents a row in a table. The record can store an entire row or some specific data selected from the table. A column in a row and corresponding fields in a record have the same name and data types.

**The syntax for declaring a variable with %ROWTYPE is:**

1. <var\_name> <tab\_name>.ROW%TYPE;

Where <variable\_name> is the variable defined in the <tab\_name>.

**Consider a declaration.**

EMPLOYEE EMP. % ROW TYPE;

This declaration will declare a record named EMPLOYEE having fields with the same name and data types as that of columns in the EMP table. You can access the elements of EMPLOYEE record as

EMPLOYEE.SAL := 10000;

EMPLOYEE.ENAME := ‘KIRAN’;

**Example:**

1. **DECLARE**
2. EMPLOYEE EMP. % ROW TYPE;
3. **BEGIN**
4. EMPLOYEE.EMPNO := 2092;
5. 5   EMPLOYEE.ENAME := 'Sanju';
6. **Insert** **into** EMP **where** (EMPNO, ENAME) **Values** (employee.empno, employee.ename);
7. dbms\_output.put\_line('Row Inserted');
8. **END**;

After the execution, this will produce the following result:

Row Inserted

PL/SQL procedure successfully completed.

## Advantages:

* If you don’t know the data type at the time of declaration. The data type assigned to the associated variables will be determined dynamically at run time.
* If the data type of the variable you are referencing changes the %TYPE or %ROWTYPE variable changes at run time without having to rewrite variable declarations. For example: if the ENAME column of an EMP table is changed from a VARCHAR2(10) to VRACHAR2(15) then you don’t need to modify the PL/SQL code.

PL/SQL Constants

A constant is a value used in a PL/SQL block that remains unchanged throughout the program. It is a user-defined literal value. It can be declared and used instead of actual values.

Let's take an example to explain it well:

Suppose, you have to write a program which will increase the salary of the employees upto 30%, you can declare a constant and use it throughout the program. Next time if you want to increase the salary again you can change the value of constant than the actual value throughout the program.

**Syntax to declare a constant:**

[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

1. constant\_name CONSTANT datatype := VALUE;

* **Constant\_name:**it is the name of constant just like variable name. The constant word is a reserved word and its value does not change.
* **VALUE:**it is a value which is assigned to a constant when it is declared. It can not be assigned later.

Example of PL/SQL constant

Let's take an example to explain it well:

1. **DECLARE**
2. -- constant declaration
3. pi constant number := 3.141592654;
4. -- other declarations
5. radius number(5,2);
6. dia number(5,2);
7. circumference number(7, 2);
8. area number (10, 2);
9. **BEGIN**
10. -- processing
11. radius := 9.5;
12. dia := radius \* 2;
13. circumference := 2.0 \* pi \* radius;
14. area := pi \* radius \* radius;
15. -- output
16. dbms\_output.put\_line('Radius: ' || radius);
17. dbms\_output.put\_line('Diameter: ' || dia);
18. dbms\_output.put\_line('Circumference: ' || circumference);
19. dbms\_output.put\_line('Area: ' || area);
20. **END**;
21. /

After the execution of the above code at SQL prompt, it will produce the following result:.

1. Radius: 9.5
2. Diameter: 19
3. Circumference: 59.69
4. Area: 283.53
6. Pl/SQL **procedure** successfully completed.

PL/SQL Literals

Literals are the explicit numeric, character, string or boolean values which are not represented by an identifier. For example: TRUE, NULL, etc. are all literals of type boolean. PL/SQL literals are case-sensitive. There are following kinds of literals in PL/SQL:

* Numeric Literals
* Character Literals
* String Literals
* BOOLEAN Literals
* Date and Time Literals

Example of these different types of Literals:

|  |  |
| --- | --- |
| **Literals** | **Examples** |
| Numeric | 75125, 3568, 33.3333333 etc. |
| Character | 'A' '%' '9' ' ' 'z' '(' |
| String | Hello JavaTpoint! |
| Boolean | TRUE, FALSE, NULL etc. |
| Date and Time | '26-11-2002' , '2012-10-29 12:01:01' |

# PL/SQL If

PL/SQL supports the programming language features like conditional statements and iterative statements. Its programming constructs are similar to how you use in programming languages like Java and C++.

**Syntax for IF Statement:**

There are different syntaxes for the IF-THEN-ELSE statement.

**Syntax: (IF-THEN statement):**

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

1. IF condition
2. **THEN**
3. Statement: {It **is** executed **when** condition **is** **true**}
4. **END** IF;

This syntax is used when you want to execute statements only when condition is TRUE.

**Syntax: (IF-THEN-ELSE statement):**

1. IF condition
2. **THEN**
3. {...statements **to** **execute** **when** condition **is** **TRUE**...}
4. **ELSE**
5. {...statements **to** **execute** **when** condition **is** **FALSE**...}
6. **END** IF;

This syntax is used when you want to execute one set of statements when condition is TRUE or a different set of statements when condition is FALSE.

**Syntax: (IF-THEN-ELSIF statement):**

1. IF condition1
2. **THEN**
3. {...statements **to** **execute** **when** condition1 **is** **TRUE**...}
4. ELSIF condition2
5. **THEN**
6. {...statements **to** **execute** **when** condition2 **is** **TRUE**...}
7. **END** IF;

This syntax is used when you want to execute one set of statements when condition1 is TRUE or a different set of statements when condition2 is TRUE.

**Syntax: (IF-THEN-ELSIF-ELSE statement):**

1. IF condition1
2. **THEN**
3. {...statements **to** **execute** **when** condition1 **is** **TRUE**...}
4. ELSIF condition2
5. **THEN**
6. {...statements **to** **execute** **when** condition2 **is** **TRUE**...}
7. **ELSE**
8. {...statements **to** **execute** **when** both condition1 and condition2 are **FALSE**...}
9. **END** IF;

It is the most advance syntax and used if you want to execute one set of statements when condition1 is TRUE, a different set of statement when condition2 is TRUE or a different set of statements when both the condition1 and condition2 are FALSE.

#### **When a condition is found to be TRUE, the IF-THEN-ELSE statement will execute the corresponding code and not check the conditions any further.**

#### **If there no condition is met, the ELSE portion of the IF-THEN-ELSE statement will be executed.**

#### **ELSIF and ELSE portions are optional.**

## Example of PL/SQL If Statement

Let's take an example to see the whole concept:

1. **DECLARE**
2. a number(3) := 500;
3. **BEGIN**
4. -- check the boolean condition using if statement
5. IF( a < 20 ) **THEN**
6. -- if condition is true then print the following
7. dbms\_output.put\_line('a is less than 20 ' );
8. **ELSE**
9. dbms\_output.put\_line('a is not less than 20 ' );
10. **END** IF;
11. dbms\_output.put\_line('value of a is : ' || a);
12. **END**;

After the execution of the above code in SQL prompt, you will get the following result:

a is not less than 20

value of a is : 500

PL/SQL procedure successfully completed.

PL/SQL Case Statement

The PL/SQL CASE statement facilitates you to execute a sequence of satatements based on a selector. A selector can be anything such as variable, function or an expression that the CASE statement checks to a boolean value.

The CASE statement works like the IF statement, only using the keyword WHEN. A CASE statement is evaluated from top to bottom. If it get the condition TRUE, then the corresponding THEN calause is executed and the execution goes to the END CASE clause.

**Syntax for the CASE Statement:**

1. CASE [ expression ]
2. **WHEN** condition\_1 **THEN** result\_1
3. **WHEN** condition\_2 **THEN** result\_2
4. ...
5. **WHEN** condition\_n **THEN** result\_n
6. **ELSE** result
7. **END**

Example of PL/SQL case statement

Let's take an example to make it clear:

1. **DECLARE**
2. grade **char**(1) := 'A';
3. **BEGIN**
4. CASE grade
5. **when** 'A' **then** dbms\_output.put\_line('Excellent');
6. **when** 'B' **then** dbms\_output.put\_line('Very good');
7. **when** 'C' **then** dbms\_output.put\_line('Good');
8. **when** 'D' **then** dbms\_output.put\_line('Average');
9. **when** 'F' **then** dbms\_output.put\_line('Passed with Grace');
10. **else** dbms\_output.put\_line('Failed');
11. **END** CASE;
12. **END**;

After the execution of above code, you will get the following result:

Excellent

PL/SQL procedure successfully completed.

PL/SQL Loop

The PL/SQL loops are used to repeat the execution of one or more statements for specified number of times. These are also known as iterative control statements.

**Syntax for a basic loop:**

1. LOOP
2. **Sequence** **of** statements;
3. **END** LOOP;

Types of PL/SQL Loops

There are 4 types of PL/SQL Loops.

1. Basic Loop / Exit Loop
2. While Loop
3. For Loop
4. Cursor For Loop

PL/SQL Exit Loop (Basic Loop)

PL/SQL exit loop is used when a set of statements is to be executed at least once before the termination of the loop. There must be an EXIT condition specified in the loop, otherwise the loop will get into an infinite number of iterations. After the occurrence of EXIT condition, the process exits the loop.

**Syntax of basic loop:**

1. LOOP
2. **Sequence** **of** statements;
3. **END** LOOP;

**Syntax of exit loop:**

1. LOOP
2. statements;
3. EXIT;
4. {or EXIT **WHEN** condition;}
5. **END** LOOP;

Example of PL/SQL EXIT Loop

Let's take a simple example to explain it well:

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

1. **DECLARE**
2. i NUMBER := 1;
3. **BEGIN**
4. LOOP
5. EXIT **WHEN** i>10;
6. DBMS\_OUTPUT.PUT\_LINE(i);
7. i := i+1;
8. **END** LOOP;
9. **END**;

After the execution of the above code, you will get the following result:

1

2

3

4

5

6

7

8

9

10

Note: You must follow these steps while using PL/SQL Exit Loop.

* Initialize a variable before the loop body
* Increment the variable in the loop.
* You should use EXIT WHEN statement to exit from the Loop. Otherwise the EXIT statement without WHEN condition, the statements in the Loop is executed only once.

PL/SQL EXIT Loop Example 2

1. **DECLARE**
2. VAR1 NUMBER;
3. VAR2 NUMBER;
4. **BEGIN**
5. VAR1:=100;
6. VAR2:=1;
7. LOOP
8. DBMS\_OUTPUT.PUT\_LINE (VAR1\*VAR2);
9. IF (VAR2=10) **THEN**
10. EXIT;
11. **END** IF;
12. VAR2:=VAR2+1;
13. **END** LOOP;
14. **END**;

Output:

100

200

300

400

500

600

700

800

900

1000

PL/SQL While Loop

PL/SQL while loop is used when a set of statements has to be executed as long as a condition is true, the While loop is used. The condition is decided at the beginning of each iteration and continues until the condition becomes false.

**Syntax of while loop:**

1. WHILE <condition>
2. LOOP statements;
3. **END** LOOP;

Example of PL/SQL While Loop

Let's see a simple example of PL/SQL WHILE loop.

1. **DECLARE**
2. i **INTEGER** := 1;
3. **BEGIN**
4. WHILE i <= 10 LOOP
5. DBMS\_OUTPUT.PUT\_LINE(i);
6. i := i+1;
7. **END** LOOP;
8. **END**;

After the execution of the above code, you will get the following result:

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

1

2

3

4

5

6

7

8

9

10

Note: You must follow these steps while using PL/SQL WHILE Loop.

* Initialize a variable before the loop body.
* Increment the variable in the loop.
* You can use EXIT WHEN statements and EXIT statements in While loop but it is not done often.

PL/SQL WHILE Loop Example 2

1. **DECLARE**
2. VAR1 NUMBER;
3. VAR2 NUMBER;
4. **BEGIN**
5. VAR1:=200;
6. VAR2:=1;
7. WHILE (VAR2<=10)
8. LOOP
9. DBMS\_OUTPUT.PUT\_LINE (VAR1\*VAR2);
10. VAR2:=VAR2+1;
11. **END** LOOP;
12. **END**;

Output:

200

400

600

800

1000

1200

1400

1600

1800

2000

PL/SQL FOR Loop

PL/SQL for loop is used when when you want to execute a set of statements for a predetermined number of times. The loop is iterated between the start and end integer values. The counter is always incremented by 1 and once the counter reaches the value of end integer, the loop ends.

**Syntax of for loop:**

1. **FOR** counter IN initial\_value .. final\_value LOOP
2. LOOP statements;
3. **END** LOOP;

* initial\_value : Start integer value
* final\_value : End integer value

PL/SQL For Loop Example 1

Let's see a simple example of PL/SQL FOR loop.

1. **BEGIN**
2. **FOR** k IN 1..10 LOOP
3. -- note that k was not declared
4. DBMS\_OUTPUT.PUT\_LINE(k);
5. **END** LOOP;
6. **END**;

After the execution of the above code, you will get the following result:

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

1

2

3

4

5

6

7

8

9

10

Note: You must follow these steps while using PL/SQL WHILE Loop.

* You don't need to declare the counter variable explicitly because it is declared implicitly in the declaration section.
* The counter variable is incremented by 1 and does not need to be incremented explicitly.
* You can use EXIT WHEN statements and EXIT statements in FOR Loops but it is not done often.

PL/SQL For Loop Example 2

1. **DECLARE**
2. VAR1 NUMBER;
3. **BEGIN**
4. VAR1:=10;
5. **FOR** VAR2 IN 1..10
6. LOOP
7. DBMS\_OUTPUT.PUT\_LINE (VAR1\*VAR2);
8. **END** LOOP;
9. **END**;

Output:

10

20

30

40

50

60

70

80

90

100

PL/SQL For Loop REVERSE Example 3

Let's see an example of PL/SQL for loop where we are using REVERSE keyword.

1. **DECLARE**
2. VAR1 NUMBER;
3. **BEGIN**
4. VAR1:=10;
5. **FOR** VAR2 IN REVERSE 1..10
6. LOOP
7. DBMS\_OUTPUT.PUT\_LINE (VAR1\*VAR2);
8. **END** LOOP;
9. **END**;

Output:

100

90

80

70

60

50

40

30

20

10

# PL/SQL Continue Statement

The continue statement is used to exit the loop from the reminder if its body either conditionally or unconditionally and forces the next iteration of the loop to take place, skipping any codes in between.

The continue statement is not a keyword in Oracle 10g. It is a new feature encorporated in oracle 11g.

For example: If a continue statement exits a cursor FOR LOOP prematurely then it exits an inner loop and transfer control to the next iteration of an outer loop, the cursor closes (in this context, CONTINUE works like GOTO).

**Syntax:**

1. **continue**;

## Example of PL/SQL continue statement

Let's take an example of PL/SQL continue statement.

1. **DECLARE**
2. x NUMBER := 0;
3. **BEGIN**
4. LOOP -- After CONTINUE statement, control resumes here
5. DBMS\_OUTPUT.PUT\_LINE ('Inside loop:  x = ' || TO\_CHAR(x));
6. x := x + 1;
7. IF x < 3 **THEN**
8. **CONTINUE**;
9. **END** IF;
10. DBMS\_OUTPUT.PUT\_LINE
11. ('Inside loop, after CONTINUE:  x = ' || TO\_CHAR(x));
12. EXIT **WHEN** x = 5;
13. **END** LOOP;
15. DBMS\_OUTPUT.PUT\_LINE (' After loop:  x = ' || TO\_CHAR(x));
16. **END**;
17. /

After the execution of above code, you will get the following result:

Inside loop: x = 0

Inside loop: x = 1

Inside loop: x = 2

Inside loop, after CONTINUE: x = 3

Inside loop: x = 3

Inside loop, after CONTINUE: x = 4

Inside loop: x = 4

Inside loop, after CONTINUE: x = 5

After loop: x = 5

#### **Note: The continue statement is not supported in Oracle 10g. Oracle 11g supports this as a new feature.**

# PL/SQL GOTO Statement

In PL/SQL, GOTO statement makes you able to get an unconditional jump from the GOTO to a specific executable statement label in the same subprogram of the PL/SQL block.

Here the label declaration which contains the label\_name encapsulated within the << >> symbol and must be followed by at least one statement to execute.

**Syntax:**

1. **GOTO** label\_name;

Here the label declaration which contains the label\_name encapsulated within the << >> symbol and must be followed by at least one statement to execute.

1. **GOTO** label\_name;
2. ..
3. ..
4. <<label\_name>>
5. Statement;

Example of PL/SQL GOTO statement

Let's take an example of PL/SQL GOTO statement.

1. **DECLARE**
2. a number(2) := 30;
3. **BEGIN**
4. <<loopstart>>
5. -- while loop execution
6. WHILE a < 50 LOOP
7. dbms\_output.put\_line ('value of a: ' || a);
8. a := a + 1;
9. IF a = 35 **THEN**
10. a := a + 1;
11. **GOTO** loopstart;
12. **END** IF;
13. **END** LOOP;
14. **END**;
15. /

After the execution of above code, you will get the following result:

value of a: 30

value of a: 31

value of a: 32

value of a: 33

value of a: 34

value of a: 36

value of a: 37

value of a: 38

value of a: 39

value of a: 40

value of a: 41

value of a: 42

value of a: 43

value of a: 44

value of a: 45

value of a: 46

value of a: 47

value of a: 48

value of a: 49

Statement processed.

Restriction on GOTO statement

Following is a list of some restrictions imposed on GOTO statement.

* Cannot transfer control into an IF statement, CASE statement, LOOP statement or sub-block.
* Cannot transfer control from one IF statement clause to another or from one CASE statement WHEN clause to another.
* Cannot transfer control from an outer block into a sub-block.
* Cannot transfer control out of a subprogram.
* Cannot transfer control into an exception handler.

# PL/SQL Procedure

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

* **Header:** The header contains the name of the procedure and the parameters or variables passed to the procedure.
* **Body:** The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

## How to pass parameters in procedure:

When you want to create a procedure or function, you have to define parameters .There is three ways to pass parameters in procedure:

1. **IN parameters:**The IN parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or the function.
2. **OUT parameters:**The OUT parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.
3. **INOUT parameters:**The INOUT parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

#### **A procedure may or may not return any value.**

## PL/SQL Create Procedure

**Syntax for creating procedure:**

1. **CREATE** [OR REPLACE] **PROCEDURE** procedure\_name
2. [ (parameter [,parameter]) ]
3. **IS**
4. [declaration\_section]
5. **BEGIN**
6. executable\_section
7. [EXCEPTION
8. exception\_section]
9. **END** [procedure\_name];

## Create procedure example

In this example, we are going to insert record in user table. So you need to create user table first.

**Table creation:**

1. **create** **table** user(id number(10) **primary** **key**,**name** varchar2(100));

Now write the procedure code to insert record in user table.

**Procedure Code:**

1. **create** or replace **procedure** "INSERTUSER"
2. (id IN NUMBER,
3. **name** IN VARCHAR2)
4. **is**
5. **begin**
6. **insert** **into** user **values**(id,**name**);
7. **end**;
8. /

Output:

Procedure created.

## PL/SQL program to call procedure

Let's see the code to call above created procedure.

1. **BEGIN**
2. insertuser(101,'Rahul');
3. dbms\_output.put\_line('record inserted successfully');
4. **END**;
5. /

Now, see the "USER" table, you will see one record is inserted.

|  |  |
| --- | --- |
| **ID** | **Name** |
| 101 | Rahul |

## PL/SQL Drop Procedure

**Syntax for drop procedure**

1. **DROP** **PROCEDURE** procedure\_name;

## Example of drop procedure

1. **DROP** **PROCEDURE** pro1;

# PL/SQL Function

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.

**Syntax to create a function:**

1. **CREATE** [OR REPLACE] **FUNCTION** function\_name [parameters]
2. [(parameter\_name [IN | **OUT** | IN **OUT**] type [, ...])]
3. **RETURN** return\_datatype
4. {**IS** | **AS**}
5. **BEGIN**
6. < function\_body >
7. **END** [function\_name];

**Here:**

* **Function\_name:** specifies the name of the function.
* **[OR REPLACE]** option allows modifying an existing function.
* The **optional parameter list** contains name, mode and types of the parameters.
* **IN** represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.

### **The function must contain a return statement.**

* RETURN clause specifies that data type you are going to return from the function.
* Function\_body contains the executable part.
* The AS keyword is used instead of the IS keyword for creating a standalone function.

## PL/SQL Function Example

Let's see a simple example to **create a function**.

1. **create** or replace **function** adder(n1 in number, n2 in number)
2. **return** number
3. **is**
4. n3 number(8);
5. **begin**
6. n3 :=n1+n2;
7. **return** n3;
8. **end**;
9. /

Now write another program to **call the function**.

1. **DECLARE**
2. n3 number(2);
3. **BEGIN**
4. n3 := adder(11,22);
5. dbms\_output.put\_line('Addition is: ' || n3);
6. **END**;
7. /

**Output:**

Addition is: 33

Statement processed.

0.05 seconds

## Another PL/SQL Function Example

Let's take an example to demonstrate Declaring, Defining and Invoking a simple PL/SQL function which will compute and return the maximum of two values.

1. **DECLARE**
2. a number;
3. b number;
4. c number;
5. **FUNCTION** findMax(x IN number, y IN number)
6. **RETURN** number
7. **IS**
8. z number;
9. **BEGIN**
10. IF x > y **THEN**
11. z:= x;
12. **ELSE**
13. Z:= y;
14. **END** IF;
16. **RETURN** z;
17. **END**;
18. **BEGIN**
19. a:= 23;
20. b:= 45;
22. c := findMax(a, b);
23. dbms\_output.put\_line(' Maximum of (23,45): ' || c);
24. **END**;
25. /

**Output:**

Maximum of (23,45): 45

Statement processed.

0.02 seconds

## PL/SQL function example using table

Let's take a customer table. This example illustrates creating and calling a standalone function. This function will return the total number of CUSTOMERS in the customers table.

#### **Create customers table and have records in it.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Customers** | | | |
| **Id** | **Name** | **Department** | **Salary** |
| 1 | alex | web developer | 35000 |
| 2 | ricky | program developer | 45000 |
| 3 | mohan | web designer | 35000 |
| 4 | dilshad | database manager | 44000 |

**Create Function:**

1. **CREATE** OR REPLACE **FUNCTION** totalCustomers
2. **RETURN** number **IS**
3. total number(2) := 0;
4. **BEGIN**
5. **SELECT** count(\*) **into** total
6. **FROM** customers;
7. **RETURN** total;
8. **END**;
9. /

After the execution of above code, you will get the following result.

Function created.

**Calling PL/SQL Function:**

While creating a function, you have to give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. Once the function is called, the program control is transferred to the called function.

After the successful completion of the defined task, the call function returns program control back to the main program.

To call a function you have to pass the required parameters along with function name and if function returns a value then you can store returned value. Following program calls the function totalCustomers from an anonymous block:

1. **DECLARE**
2. c number(2);
3. **BEGIN**
4. c := totalCustomers();
5. dbms\_output.put\_line('Total no. of Customers: ' || c);
6. **END**;
7. /

After the execution of above code in SQL prompt, you will get the following result.

Total no. of Customers: 4

PL/SQL procedure successfully completed.

## PL/SQL Recursive Function

You already know that a program or a subprogram can call another subprogram. When a subprogram calls itself, it is called recursive call and the process is known as recursion.

## Example to calculate the factorial of a number

Let's take an example to calculate the factorial of a number. This example calculates the factorial of a given number by calling itself recursively.

1. **DECLARE**
2. num number;
3. factorial number;
5. **FUNCTION** fact(x number)
6. **RETURN** number
7. **IS**
8. f number;
9. **BEGIN**
10. IF x=0 **THEN**
11. f := 1;
12. **ELSE**
13. f := x \* fact(x-1);
14. **END** IF;
15. **RETURN** f;
16. **END**;
18. **BEGIN**
19. num:= 6;
20. factorial := fact(num);
21. dbms\_output.put\_line(' Factorial '|| num || ' is ' || factorial);
22. **END**;
23. /

After the execution of above code at SQL prompt, it produces the following result.

Factorial 6 is 720

PL/SQL procedure successfully completed.

## PL/SQL Drop Function

**Syntax for removing your created function:**

If you want to remove your created function from the database, you should use the following syntax.

1. **DROP** **FUNCTION** function\_name;

PL/SQL Cursor

When an SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. It contains all information needed for processing the statement. In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it.

A cursor is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors:

* Implicit Cursors
* Explicit Cursors

1) PL/SQL Implicit Cursors

The implicit cursors are automatically generated by Oracle while an SQL statement is executed, if you don't use an explicit cursor for the statement.

These are created by default to process the statements when DML statements like INSERT, UPDATE, DELETE etc. are executed.

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

Orcale provides some attributes known as Implicit cursor's attributes to check the status of DML operations. Some of them are: %FOUND, %NOTFOUND, %ROWCOUNT and %ISOPEN.

**For example:**When you execute the SQL statements like INSERT, UPDATE, DELETE then the cursor attributes tell whether any rows are affected and how many have been affected. If you run a SELECT INTO statement in PL/SQL block, the implicit cursor attribute can be used to find out whether any row has been returned by the SELECT statement. It will return an error if there no data is selected.

The following table soecifies the status of the cursor with each of its attribute.

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| %FOUND | Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect at least one row or more rows or a SELECT INTO statement returned one or more rows. Otherwise it returns FALSE. |
| %NOTFOUND | Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect no row, or a SELECT INTO statement return no rows. Otherwise it returns FALSE. It is a just opposite of %FOUND. |
| %ISOPEN | It always returns FALSE for implicit cursors, because the SQL cursor is automatically closed after executing its associated SQL statements. |
| %ROWCOUNT | It returns the number of rows affected by DML statements like INSERT, DELETE, and UPDATE or returned by a SELECT INTO statement. |

PL/SQL Implicit Cursor Example

**Create customers table and have records:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **NAME** | **AGE** | **ADDRESS** | **SALARY** |
| 1 | Ramesh | 23 | Allahabad | 20000 |
| 2 | Suresh | 22 | Kanpur | 22000 |
| 3 | Mahesh | 24 | Ghaziabad | 24000 |
| 4 | Chandan | 25 | Noida | 26000 |
| 5 | Alex | 21 | Paris | 28000 |
| 6 | Sunita | 20 | Delhi | 30000 |

Let's execute the following program to update the table and increase salary of each customer by 5000. Here, SQL%ROWCOUNT attribute is used to determine the number of rows affected:

**Create procedure:**

1. **DECLARE**
2. total\_rows number(2);
3. **BEGIN**
4. **UPDATE**  customers
5. **SET** salary = salary + 5000;
6. IF sql%notfound **THEN**
7. dbms\_output.put\_line('no customers updated');
8. ELSIF sql%found **THEN**
9. total\_rows := sql%rowcount;
10. dbms\_output.put\_line( total\_rows || ' customers updated ');
11. **END** IF;
12. **END**;
13. /

Output:

6 customers updated

PL/SQL procedure successfully completed.

Now, if you check the records in customer table, you will find that the rows are updated.

1. **select** \* **from** customers;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **NAME** | **AGE** | **ADDRESS** | **SALARY** |
| 1 | Ramesh | 23 | Allahabad | 25000 |
| 2 | Suresh | 22 | Kanpur | 27000 |
| 3 | Mahesh | 24 | Ghaziabad | 29000 |
| 4 | Chandan | 25 | Noida | 31000 |
| 5 | Alex | 21 | Paris | 33000 |
| 6 | Sunita | 20 | Delhi | 35000 |

2) PL/SQL Explicit Cursors

The Explicit cursors are defined by the programmers to gain more control over the context area. These cursors should be defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.

Following is the syntax to create an explicit cursor:

Syntax of explicit cursor

Following is the syntax to create an explicit cursor:

1. **CURSOR** cursor\_name **IS** select\_statement;;

Steps:

You must follow these steps while working with an explicit cursor.

1. Declare the cursor to initialize in the memory.
2. Open the cursor to allocate memory.
3. Fetch the cursor to retrieve data.
4. Close the cursor to release allocated memory.

1) Declare the cursor:

It defines the cursor with a name and the associated SELECT statement.

**Syntax for explicit cursor decleration**

1. **CURSOR** **name** **IS**
2. **SELECT** statement;

2) Open the cursor:

It is used to allocate memory for the cursor and make it easy to fetch the rows returned by the SQL statements into it.

**Syntax for cursor open:**

1. **OPEN** cursor\_name;

3) Fetch the cursor:

It is used to access one row at a time. You can fetch rows from the above-opened cursor as follows:

**Syntax for cursor fetch:**

1. **FETCH** cursor\_name **INTO** variable\_list;

4) Close the cursor:

It is used to release the allocated memory. The following syntax is used to close the above-opened cursors.

**Syntax for cursor close:**

1. **Close** cursor\_name;

PL/SQL Explicit Cursor Example

Explicit cursors are defined by programmers to gain more control over the context area. It is defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.

Let's take an example to demonstrate the use of explicit cursor. In this example, we are using the already created CUSTOMERS table.

**Create customers table and have records:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **NAME** | **AGE** | **ADDRESS** | **SALARY** |
| 1 | Ramesh | 23 | Allahabad | 20000 |
| 2 | Suresh | 22 | Kanpur | 22000 |
| 3 | Mahesh | 24 | Ghaziabad | 24000 |
| 4 | Chandan | 25 | Noida | 26000 |
| 5 | Alex | 21 | Paris | 28000 |
| 6 | Sunita | 20 | Delhi | 30000 |

**Create procedure:**

Execute the following program to retrieve the customer name and address.

1. **DECLARE**
2. c\_id customers.id%type;
3. c\_name customers.**name**%type;
4. c\_addr customers.address%type;
5. **CURSOR** c\_customers **is**
6. **SELECT** id, **name**, address **FROM** customers;
7. **BEGIN**
8. **OPEN** c\_customers;
9. LOOP
10. **FETCH** c\_customers **into** c\_id, c\_name, c\_addr;
11. EXIT **WHEN** c\_customers%notfound;
12. dbms\_output.put\_line(c\_id || ' ' || c\_name || ' ' || c\_addr);
13. **END** LOOP;
14. **CLOSE** c\_customers;
15. **END**;
16. /

Output:

1 Ramesh Allahabad

2 Suresh Kanpur

3 Mahesh Ghaziabad

4 Chandan Noida

5 Alex Paris

6 Sunita Delhi

PL/SQL procedure successfully completed.

# PL/SQL Exception Handling

## What is Exception

An error occurs during the program execution is called Exception in PL/SQL.

PL/SQL facilitates programmers to catch such conditions using exception block in the program and an appropriate action is taken against the error condition.

There are two type of exceptions:

* System-defined Exceptions
* User-defined Exceptions

## PL/SQL Exception Handling

**Syntax for exception handling:**

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

Following is a general syntax for exception handling:

1. **DECLARE**
2. <declarations **section**>
3. **BEGIN**
4. <executable command(s)>
5. EXCEPTION
6. <exception handling goes here >
7. **WHEN** exception1 **THEN**
8. exception1-handling-statements
9. **WHEN** exception2  **THEN**
10. exception2-handling-statements
11. **WHEN** exception3 **THEN**
12. exception3-handling-statements
13. ........
14. **WHEN** others **THEN**
15. exception3-handling-statements
16. **END**;

## Example of exception handling

Let's take a simple example to demonstrate the concept of exception handling. Here we are using the already created CUSTOMERS table.

SELECT\* FROM COUSTOMERS;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **NAME** | **AGE** | **ADDRESS** | **SALARY** |
| 1 | Ramesh | 23 | Allahabad | 20000 |
| 2 | Suresh | 22 | Kanpur | 22000 |
| 3 | Mahesh | 24 | Ghaziabad | 24000 |
| 4 | Chandan | 25 | Noida | 26000 |
| 5 | Alex | 21 | Paris | 28000 |
| 6 | Sunita | 20 | Delhi | 30000 |

1. **DECLARE**
2. c\_id customers.id%type := 8;
3. c\_name  customers.**name**%type;
4. c\_addr customers.address%type;
5. **BEGIN**
6. **SELECT**  **name**, address **INTO**  c\_name, c\_addr
7. **FROM** customers
8. **WHERE** id = c\_id;
9. DBMS\_OUTPUT.PUT\_LINE ('Name: '||  c\_name);
10. DBMS\_OUTPUT.PUT\_LINE ('Address: ' || c\_addr);
11. EXCEPTION
12. **WHEN** no\_data\_found **THEN**
13. dbms\_output.put\_line('No such customer!');
14. **WHEN** others **THEN**
15. dbms\_output.put\_line('Error!');
16. **END**;
17. /

After the execution of above code at SQL Prompt, it produces the following result:

No such customer!

PL/SQL procedure successfully completed.

The above program should show the name and address of a customer as result whose ID is given. But there is no customer with ID value 8 in our database, so the program raises the run-time exception NO\_DATA\_FOUND, which is captured in EXCEPTION block.

#### **Note: You get the result "No such customer" because the customer\_id used in the above example is 8 and there is no cutomer having id value 8 in that table.**

If you use the id defined in the above table (i.e. 1 to 6), you will get a certain result. For a demo example: here, we are using the id 5.

1. **DECLARE**
2. c\_id customers.id%type := 5;
3. c\_name  customers.**name**%type;
4. c\_addr customers.address%type;
5. **BEGIN**
6. **SELECT**  **name**, address **INTO**  c\_name, c\_addr
7. **FROM** customers
8. **WHERE** id = c\_id;
9. DBMS\_OUTPUT.PUT\_LINE ('Name: '||  c\_name);
10. DBMS\_OUTPUT.PUT\_LINE ('Address: ' || c\_addr);
11. EXCEPTION
12. **WHEN** no\_data\_found **THEN**
13. dbms\_output.put\_line('No such customer!');
14. **WHEN** others **THEN**
15. dbms\_output.put\_line('Error!');
16. **END**;
17. /

After the execution of above code at SQL prompt, you will get the following result:

Name: alex

Address: paris

PL/SQL procedure successfully completed.

## Raising Exceptions

In the case of any internal database error, exceptions are raised by the database server automatically. But it can also be raised explicitly by programmer by using command RAISE.

**Syntax for raising an exception:**

1. **DECLARE**
2. exception\_name EXCEPTION;
3. **BEGIN**
4. IF condition **THEN**
5. RAISE exception\_name;
6. **END** IF;
7. EXCEPTION
8. **WHEN** exception\_name **THEN**
9. statement;
10. **END**;

## PL/SQL User-defined Exceptions

PL/SQL facilitates their users to define their own exceptions according to the need of the program. A user-defined exception can be raised explicitly, using either a RAISE statement or the procedure DBMS\_STANDARD.RAISE\_APPLICATION\_ERROR.

**Syntax for user define exceptions**

1. **DECLARE**
2. my-exception EXCEPTION;

## PL/SQL Pre-defined Exceptions

There are many pre-defined exception in PL/SQL which are executed when any database rule is violated by the programs.

**For example:**NO\_DATA\_FOUND is a pre-defined exception which is raised when a SELECT INTO statement returns no rows.

Following is a list of some important pre-defined exceptions:

|  |  |  |  |
| --- | --- | --- | --- |
| **Exception** | **Oracle Error** | **SQL Code** | **Description** |
| ACCESS\_INTO\_NULL | 06530 | -6530 | It is raised when a NULL object is automatically assigned a value. |
| CASE\_NOT\_FOUND | 06592 | -6592 | It is raised when none of the choices in the "WHEN" clauses of a CASE statement is selected, and there is no else clause. |
| COLLECTION\_IS\_NULL | 06531 | -6531 | It is raised when a program attempts to apply collection methods other than exists to an uninitialized nested table or varray, or the program attempts to assign values to the elements of an uninitialized nested table or varray. |
| DUP\_VAL\_ON\_INDEX | 00001 | -1 | It is raised when duplicate values are attempted to be stored in a column with unique index. |
| INVALID\_CURSOR | 01001 | -1001 | It is raised when attempts are made to make a cursor operation that is not allowed, such as closing an unopened cursor. |
| INVALID\_NUMBER | 01722 | -1722 | It is raised when the conversion of a character string into a number fails because the string does not represent a valid number. |
| LOGIN\_DENIED | 01017 | -1017 | It is raised when s program attempts to log on to the database with an invalid username or password. |
| NO\_DATA\_FOUND | 01403 | +100 | It is raised when a select into statement returns no rows. |
| NOT\_LOGGED\_ON | 01012 | -1012 | It is raised when a database call is issued without being connected to the database. |
| PROGRAM\_ERROR | 06501 | -6501 | It is raised when PL/SQL has an internal problem. |
| ROWTYPE\_MISMATCH | 06504 | -6504 | It is raised when a cursor fetches value in a variable having incompatible data type. |
| SELF\_IS\_NULL | 30625 | -30625 | It is raised when a member method is invoked, but the instance of the object type was not initialized. |
| STORAGE\_ERROR | 06500 | -6500 | It is raised when PL/SQL ran out of memory or memory was corrupted. |
| TOO\_MANY\_ROWS | 01422 | -1422 | It is raised when a SELECT INTO statement returns more than one row. |
| VALUE\_ERROR | 06502 | -6502 | It is raised when an arithmetic, conversion, truncation, or size-constraint error occurs. |
| ZERO\_DIVIDE | 01476 | 1476 | It is raised when an attempt is made to divide a number by zero. |

PL/SQL Trigger

Trigger is invoked by Oracle engine automatically whenever a specified event occurs.Trigger is stored into database and invoked repeatedly, when specific condition match.

Triggers are stored programs, which are automatically executed or fired when some event occurs.

Triggers are written to be executed in response to any of the following events.

* A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
* A database definition (DDL) statement (CREATE, ALTER, or DROP).
* A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

Play Video[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

Advantages of Triggers

These are the following advantages of Triggers:

* Trigger generates some derived column values automatically
* Enforces referential integrity
* Event logging and storing information on table access
* Auditing
* Synchronous replication of tables
* Imposing security authorizations
* Preventing invalid transactions

Creating a trigger:

**Syntax for creating trigger:**

1. **CREATE** [OR REPLACE ] **TRIGGER** trigger\_name
2. {BEFORE | **AFTER** | **INSTEAD** **OF** }
3. {**INSERT** [OR] | **UPDATE** [OR] | **DELETE**}
4. [**OF** col\_name]
5. **ON** table\_name
6. [REFERENCING OLD **AS** o NEW **AS** n]
7. [**FOR** EACH ROW]
8. **WHEN** (condition)
9. **DECLARE**
10. Declaration-statements
11. **BEGIN**
12. Executable-statements
13. EXCEPTION
14. Exception-handling-statements
15. **END**;

**Here,**

* CREATE [OR REPLACE] TRIGGER trigger\_name: It creates or replaces an existing trigger with the trigger\_name.
* {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
* {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
* [OF col\_name]: This specifies the column name that would be updated.
* [ON table\_name]: This specifies the name of the table associated with the trigger.
* [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
* [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
* WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

PL/SQL Trigger Example

Let's take a simple example to demonstrate the trigger. In this example, we are using the following CUSTOMERS table:

**Create table and have records:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **NAME** | **AGE** | **ADDRESS** | **SALARY** |
| 1 | Ramesh | 23 | Allahabad | 20000 |
| 2 | Suresh | 22 | Kanpur | 22000 |
| 3 | Mahesh | 24 | Ghaziabad | 24000 |
| 4 | Chandan | 25 | Noida | 26000 |
| 5 | Alex | 21 | Paris | 28000 |
| 6 | Sunita | 20 | Delhi | 30000 |

**Create trigger:**

Let's take a program to create a row level trigger for the CUSTOMERS table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:

1. **CREATE** OR REPLACE **TRIGGER** display\_salary\_changes
2. BEFORE **DELETE** OR **INSERT** OR **UPDATE** **ON** customers
3. **FOR** EACH ROW
4. **WHEN** (NEW.ID > 0)
5. **DECLARE**
6. sal\_diff number;
7. **BEGIN**
8. sal\_diff := :NEW.salary  - :OLD.salary;
9. dbms\_output.put\_line('Old salary: ' || :OLD.salary);
10. dbms\_output.put\_line('New salary: ' || :NEW.salary);
11. dbms\_output.put\_line('Salary difference: ' || sal\_diff);
12. **END**;
13. /

After the execution of the above code at SQL Prompt, it produces the following result.

Trigger created.

**Check the salary difference by procedure:**

Use the following code to get the old salary, new salary and salary difference after the trigger created.

1. **DECLARE**
2. total\_rows number(2);
3. **BEGIN**
4. **UPDATE**  customers
5. **SET** salary = salary + 5000;
6. IF sql%notfound **THEN**
7. dbms\_output.put\_line('no customers updated');
8. ELSIF sql%found **THEN**
9. total\_rows := sql%rowcount;
10. dbms\_output.put\_line( total\_rows || ' customers updated ');
11. **END** IF;
12. **END**;
13. /

Output:

Old salary: 20000

New salary: 25000

Salary difference: 5000

Old salary: 22000

New salary: 27000

Salary difference: 5000

Old salary: 24000

New salary: 29000

Salary difference: 5000

Old salary: 26000

New salary: 31000

Salary difference: 5000

Old salary: 28000

New salary: 33000

Salary difference: 5000

Old salary: 30000

New salary: 35000

Salary difference: 5000

6 customers updated

**Note:** As many times you executed this code, the old and new both salary is incremented by 5000 and hence the salary difference is always 5000.

After the execution of above code again, you will get the following result.

Old salary: 25000

New salary: 30000

Salary difference: 5000

Old salary: 27000

New salary: 32000

Salary difference: 5000

Old salary: 29000

New salary: 34000

Salary difference: 5000

Old salary: 31000

New salary: 36000

Salary difference: 5000

Old salary: 33000

New salary: 38000

Salary difference: 5000

Old salary: 35000

New salary: 40000

Salary difference: 5000

6 customers updated

Important Points

Following are the two very important point and should be noted carefully.

* OLD and NEW references are used for record level triggers these are not avialable for table level triggers.
* If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.

# PL/SQL Interview Questions

PL/SQL is an advance version of SQL. There are given top list of PL/SQL interview questions with answer.

### **1) What is PL/SQL?**

PL/SQL stands for procedural language extension to SQL. It supports procedural features of programming language and SQL both. It was developed by Oracle Corporation in early of 90's to enhance the capabilities of SQL.

### **2) What is the purpose of using PL/SQL?**

PL/SQL is an extension of SQL. While SQL is non-procedural, PL/SQL is a procedural language designed by Oracle. It is invented to overcome the limitations of SQL.

### **3) What are the most important characteristics of PL/SQL?**

A list of some notable characteristics:

[](https://campaign.adpushup.com/get-started/?utm_source=banner&utm_campaign=growth_hack)

* PL/SQL is a block-structured language.
* It is portable to all environments that support Oracle.
* PL/SQL is integrated with the Oracle data dictionary.
* Stored procedures help better sharing of application.

### **4) What is PL/SQL table? Why it is used?**

Objects of type tables are called PL/SQL tables that are modeled as database table. We can also say that PL/SQL tables are a way to providing arrays. Arrays are like temporary tables in memory that are processed very quickly. PL/SQL tables are used to move bulk data. They simplifies moving collections of data.

### **5) What are the datatypes available in PL/SQL?**

There are two types of datatypes in PL/SQL:

1. **Scalar datatypes** Example are NUMBER, VARCHAR2, DATE, CHAR, LONG, BOOLEAN etc.
2. **Composite datatypes** Example are RECORD, TABLE etc.

### **6) What is the basic structure of PL/SQL?**

PL/SQL uses BLOCK structure as its basic structure. Each PL/SQL program consists of SQL and PL/SQL statement which form a PL/SQL block.

PL/SQL block contains 3 sections.

1. The Declaration Section (optional)
2. The Execution Section (mandatory)
3. The Exception handling Section (optional)

### **7) What is the difference between FUNCTION, PROCEDURE AND PACKAGE in PL/SQL?**

**Function**: The main purpose of a PL/SQL function is generally to compute and return a single value. A function has a return type in its specification and must return a value specified in that type.

**Procedure**: A procedure does not have a return type and should not return any value but it can have a return statement that simply stops its execution and returns to the caller. A procedure is used to return multiple values otherwise it is generally similar to a function.

**Package**: A package is schema object which groups logically related PL/SQL types , items and subprograms. You can also say that it is a group of functions, procedure, variables and record type statement. It provides modularity, due to this facility it aids application development. It is used to hide information from unauthorized users.

### **8) What is exception? What are the types of exceptions?**

Exception is an error handling part of PL/SQL. There are two type of exceptions: pre\_defined exception and user\_defined exception.

### **9) How to write a single statement that concatenates the words ?Hello? and ?World? and assign it in a variable named Greeting?**

Greeting := 'Hello' || 'World';

### **10) Does PL/SQL support CREATE command?**

No. PL/SQL doesn't support the data definition commands like CREATE.

### **11) Write a unique difference between a function and a stored procedure.**

A function returns a value while a stored procedure doesn?t return a value.

### **12) How exception is different from error?**

Whenever an Error occurs Exception arises. Error is a bug whereas exception is a warning or error condition.

### **13) What is the main reason behind using an index?**

Faster access of data blocks in the table.

### **14) What are PL/SQL exceptions? Tell me any three.**

1. Too\_many\_rows
2. No\_Data\_Found
3. Value\_error
4. Zero\_error etc.

### **15) How do you declare a user-defined exception?**

You can declare the User defined exceptions under the DECLARE section, with the keyword EXCEPTION.

**Syntax:**

1. **<exception\_name>** EXCEPTION;

### **16) What are some predefined exceptions in PL/SQL?**

A list of predefined exceptions in PL/SQL:

* DUP\_VAL\_ON\_INDEX
* ZERO\_DIVIDE
* NO\_DATA\_FOUND
* TOO\_MANY\_ROWS
* CURSOR\_ALREADY\_OPEN
* INVALID\_NUMBER
* INVALID\_CURSOR
* PROGRAM\_ERROR
* TIMEOUT \_ON\_RESOURCE
* STORAGE\_ERROR
* LOGON\_DENIED
* VALUE\_ERROR
* etc.

### **17) What is a trigger in PL/SQL?**

A trigger is a PL/SQL program which is stored in the database. It is executed immediately before or after the execution of INSERT, UPDATE, and DELETE commands.

### **18) What is the maximum number of triggers, you can apply on a single table?**

12 triggers.

### **19) How many types of triggers exist in PL/SQL?**

There are 12 types of triggers in PL/SQL that contains the combination of BEFORE, AFTER, ROW, TABLE, INSERT, UPDATE, DELETE and ALL keywords.

* BEFORE ALL ROW INSERT
* AFTER ALL ROW INSERT
* BEFORE INSERT
* AFTER INSERT etc.

### **20) What is the difference between execution of triggers and stored procedures?**

A trigger is automatically executed without any action required by the user, while, a stored procedure is explicitly invoked by the user.

### **21) What happens when a trigger is associated to a view?**

When a trigger is associated to a view, the base table triggers are normally enabled.

### **22) What is the usage of WHEN clause in trigger?**

A WHEN clause specifies the condition that must be true for the trigger to be triggered.

### **23) How to disable a trigger name update\_salary?**

ALTER TRIGGER update\_salary DISABLE;

### **24) Which command is used to delete a trigger?**

DROP TRIGGER command.

### **25) what are the two virtual tables available at the time of database trigger execution?**

Table columns are referred as THEN.column\_name and NOW.column\_name.

For INSERT related triggers, NOW.column\_name values are available only.

For DELETE related triggers, THEN.column\_name values are available only.

For UPDATE related triggers, both Table columns are available.

### **26) What is stored Procedure?**

A stored procedure is a sequence of statement or a named PL/SQL block which performs one or more specific functions. It is similar to a procedure in other programming languages. It is stored in the database and can be repeatedly executed. It is stored as schema object. It can be nested, invoked and parameterized.

### **27) What are the different schemas objects that can be created using PL/SQL?**

* Stored procedures and functions
* Packages
* Triggers
* Cursors

### **28) What do you know by PL/SQL Cursors?**

Oracle uses workspaces to execute the SQL commands. When Oracle processes a SQL command, it opens an area in the memory called Private SQL Area. This area is identified by the cursor. It allows programmers to name this area and access it?s information.

### **29) What is the difference between the implicit and explicit cursors?**

Implicit cursor is implicitly declared by Oracle. This is a cursor to all the DDL and DML commands that return only one row.

Explicit cursor is created for queries returning multiple rows.

### **30) What will you get by the cursor attribute SQL%ROWCOUNT?**

The cursor attribute SQL%ROWCOUNT will return the number of rows that are processed by a SQL statement.

### **31) What will you get by the cursor attribute SQL%FOUND?**

It returns the Boolean value TRUE if at least one row was processed.

### **32) What will you get by the cursor attribute SQL%NOTFOUND?**

It returns the Boolean value TRUE if no rows were processed.

### **33) What do you understand by PL/SQL packages?**

A PL/SQL package can be specified as a file that groups functions, cursors, stored procedures, and variables in one place.

### **34) What are the two different parts of the PL/SQL packages?**

PL/SQL packages have the following two parts:

**Specification part:** It specifies the part where the interface to the application is defined.

**Body part:** This part specifies where the implementation of the specification is defined.

### **35) Which command is used to delete a package?**

The DROP PACKAGE command is used to delete a package.

### **36) How to execute a stored procedure?**

There are two way to execute a stored procedure.

From the SQL prompt, write EXECUTE or EXEC followed by procedure\_name.

1. **EXECUTE** or [**EXEC**] procedure\_name;

Simply use the procedure name

1. procedure\_name;

### **37) What are the advantages of stored procedure?**

Modularity, extensibility, reusability, Maintainability and one time compilation.

### **38) What are the cursor attributes used in PL/SQL?**

**%ISOPEN**: it checks whether the cursor is open or not.

**%ROWCOUNT**: returns the number of rows affected by DML operations: INSERT,DELETE,UPDATE,SELECT.

**%FOUND**: it checks whether cursor has fetched any row. If yes - TRUE.

**%NOTFOUND**: it checks whether cursor has fetched any row. If no - TRUE.

### **39) What is the difference between syntax error and runtime error?**

A syntax error can be easily detected by a PL/SQL compiler. For example: incorrect spelling etc. while, a runtime error is handled with the help of exception-handling section in a PL/SQL block. For example: SELECT INTO statement, which does not return any rows.

### **40) Explain the Commit statement.**

Following conditions are true for the Commit statement:

* Other users can see the data changes made by the transaction.
* The locks acquired by the transaction are released.
* The work done by the transaction becomes permanent.

### **41) Explain the Rollback statement?**

The Rollback statement is issued when the transaction ends. Following conditions are true for a Rollback statement:

* The work done in a transition is undone as if it was never issued.
* All locks acquired by transaction are released.

### **42) Explain the SAVEPOINT statement.**

With SAVEPOINT, only part of transaction can be undone.

### **43) What is mutating table error?**

Mutating table error is occurred when a trigger tries to update a row that it is currently using. It is fixed by using views or temporary tables.

### **44) What is consistency?**

Consistency simply means that each user sees the consistent view of the data.

Consider an example: there are two users A and B. A transfers money to B's account. Here the changes are updated in A's account (debit) but until it will be updated to B's account (credit), till then other users can't see the debit of A's account. After the debit of A and credit of B, one can see the updates. That?s consistency.

### **45) What is cursor and why it is required?**

A cursor is a **temporary work area** created in a system memory when an SQL statement is executed.

A cursor contains information on a select statement and the row of data accessed by it. This temporary work area stores the data retrieved from the database and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. Cursor are required to process rows individually for queries.

### **46) How many types of cursors are available in PL/SQL?**

There are two types of cursors in PL/SQL.

1. Implicit cursor, and
2. explicit cursor

# SQL Interview Questions and Answers



The following are the most popular and useful SQL interview questions and answers for fresher and experienced candidates. These questions are created specifically to familiarise you with the types of questions you might encounter during your SQL interview. According to our experiences, good interviewers rarely plan to ask any specific topic during the interview. Instead, questioning usually begins with a basic understanding of the subject, and based on your responses, further discussion happened.

### **1) What is SQL?**

SQL stands for the Structured Query Language. It is the standard language used to maintain the relational database and perform many different data manipulation operations on the data. SQL was initially invented in 1970. It is a database language used for database creation, deletion, fetching and modifying rows, etc. sometimes, it is pronounced as 'sequel.' We can also use it to handle organized data comprised of entities (variables) and relations between different entities of the data.

### **2) When SQL appeared?**

SQL first appeared in 1974. It is one of the most used languages for maintaining the relational database. In 1986, SQL became the standard of the American National Standards Institute (ANSI) and ISO (International Organization for Standardization) in 1987.

### **3) What are the usages of SQL?**

SQL is responsible for maintaining the relational data and the data structures present in the database. Some of the common usages are given below:

70.3M

1.1K

Hello Java Program for Beginners

* To execute queries against a database
* To retrieve data from a database
* To inserts records in a database
* To updates records in a database
* To delete records from a database
* To create new databases
* To create new tables in a database
* To create views in a database
* To perform complex operations on the database.

### **4) Does SQL support programming language features?**

SQL refers to the Standard Query Language. Therefore, it is true that SQL is a language but does not actually support the programming language. It is a common language that doesn't have a loop, conditional statements, and logical operations. It cannot be used for anything other than data manipulation. It is a command language to perform database operations. The primary purpose of SQL is to retrieve, manipulate, update, delete, and perform complex operations like joins on the data present in the database.

### **5) What are the subsets of SQL?**

The following are the four significant subsets of the SQL:

* **Data definition language (DDL):** It defines the data structure that consists of commands like CREATE, ALTER, DROP, etc.
* **Data manipulation language (DML):** It is used to manipulate existing data in the database. The commands in this category are SELECT, UPDATE, INSERT, etc.
* **Data control language (DCL):** It controls access to the data stored in the database. The commands in this category include GRANT and REVOKE.
* **Transaction Control Language (TCL):** It is used to deal with the transaction operations in the database. The commands in this category are COMMIT, ROLLBACK, SET TRANSACTION, SAVEPOINT, etc.

### **6) What is the purpose of DDL Language?**

DDL stands for Data definition language. It is the subset of a database that defines the data structure of the database when the database is created. **For example,** we can use the DDL commands to add, remove, or modify tables. It consists of the following commands: CREATE, ALTER and DELETE database objects such as schema, tables, indexes, view, sequence, etc.

**Example**

1. CREATE TABLE Students
2. (
3. Roll\_no INT,
4. Name VARCHAR(45),
5. Branch VARCHAR(30),
6. );

### **7) What is the purpose of DML Language?**

Data manipulation language makes the user able to retrieve and manipulate data in a relational database. The DML commands can only perform read-only operations on data. We can perform the following operations using DDL language:

* Insert data into the database through the INSERT command.
* Retrieve data from the database through the SELECT command.
* Update data in the database through the UPDATE command.
* Delete data from the database through the DELETE command.

**Example**

1. INSERT INTO Student VALUES (111, 'George', 'Computer Science')

### **8) What is the purpose of DCL Language?**

Data control language allows users to control access and permission management to the database. It is the subset of a database, which decides that what part of the database should be accessed by which user at what point of time. It includes two commands, GRANT and REVOKE.

**GRANT:** It enables system administrators to assign privileges and roles to the specific user accounts to perform specific tasks on the database.

**REVOKE:** It enables system administrators to revoke privileges and roles from the user accounts so that they cannot use the previously assigned permission on the database.

**Example**

1. GRANT \* ON mydb.Student TO javatpoint@localhsot;

### **9) What are tables and fields in the database?**

A table is a set of organized data in the form of rows and columns. It enables users to store and display records in the structure format. It is similar to worksheets in the spreadsheet application. Here rows refer to the tuples, representing the simple data item, and columns are the attribute of the data items present in a particular row. Columns can categorize as vertical, and Rows are horizontal.

Fields are the components to provide the structure for the table. It stores the same category of data in the same data type. A table contains a fixed number of columns but can have any number of rows known as the record. It is also called a column in the table of the database. It represents the attribute or characteristics of the entity in the record.

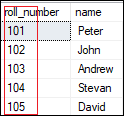
**Example**

**Table:** Student

**Field:** Stud\_rollno, Stud\_name, Date of Birth, Branch, etc.

### **10) What is a primary key?**

A primary key is a field or the combination of fields that uniquely identify each record in the table. It is one of a special kind of unique key. If the column contains a primary key, it cannot be null or empty. A table can have duplicate columns, but it cannot have more than one primary key. It always stores unique values into a column. **For example,** the ROLL Number can be treated as the primary key for a student in the university or college.



We can define a primary key into a student table as follows:

1. CREATE TABLE Student (
2. roll\_number INT PRIMARY KEY,
3. name VARCHAR(45),
4. );

To read more information, [click here](https://www.javatpoint.com/mysql-primary-key).

### **11) What is a foreign key?**

The foreign key is used to link one or more tables together. It is also known as the referencing key. A foreign key is specified as a key that is related to the primary key of another table. It means a foreign key field in one table refers to the primary key field of the other table. It identifies each row of another table uniquely that maintains the referential integrity. The primary key-foreign key relationship is a very crucial relationship as it maintains the ACID properties of the database sometimes. It also prevents actions that would destroy links between the child and parent tables.

We can define a foreign key into a table as follows:

1. CONSTRAINT constraint\_name]
2. FOREIGN KEY [foreign\_key\_name] (col\_name, ...)
3. REFERENCES parent\_tbl\_name (col\_name,...)

To read more information, [click here](https://www.javatpoint.com/mysql-foreign-key).

### **12) What is a unique key?**

A unique key is a single or combination of fields that ensure all values stores in the column will be unique. It means a column cannot stores duplicate values. This key provides uniqueness for the column or set of columns. **For example,** the email addresses and roll numbers of student's tables should be unique. It can accept a null value but only one null value per column. It ensures the integrity of the column or group of columns to store different values into a table.

We can define a foreign key into a table as follows:

1. CREATE TABLE table\_name(
2. col1 datatype,
3. col2 datatype UNIQUE,
4. ...
5. );

To read more information, [click here](https://www.javatpoint.com/mysql-unique-key).

### **13) What is the difference between a primary key and a unique key?**

The primary key and unique key both are essential constraints of the SQL. The main difference among them is that the primary key identifies each record in the table. In contrast, the unique key prevents duplicate entries in a column except for a NULL value. The following comparison chart explains it more clearly:

|  |  |
| --- | --- |
| **Primary Key** | **Unique Key** |
| The primary key act as a unique identifier for each record in the table. | The unique key is also a unique identifier for records when the primary key is not present in the table. |
| We cannot store NULL values in the primary key column. | We can store NULL value in the unique key column, but only one NULL is allowed. |
| We cannot change or delete the primary key column values. | We can modify the unique key column values. |

To read more information, [click here](https://www.javatpoint.com/primary-key-vs-unique-key).

### **14) What is a Database?**

A database is an organized collection of data that is structured into tables, rows, columns, and indexes. It helps the user to find the relevant information frequently. It is an electronic system that makes data access, data manipulation, data retrieval, data storing, and data management very easy. Almost every organization uses the database for storing the data due to its easily accessible and high operational ease. The database provides perfect access to data and lets us perform required tasks.

The following are the common features of a database:

* Manages large amounts of data
* Accurate
* Easy to update
* Security
* Data integrity
* Easy to research data

### **15) What is meant by DBMS?**

DBMS stands for Database Management System. It is a software program that primarily functions as an interface between the database and the end-user. It provides us the power such as managing the data, the database engine, and the database schema to facilitate the organization and manipulation of data using a simple query in almost no time. It is like a File Manager that manages data in a database rather than saving it in file systems. Without the database management system, it would be far more difficult for the user to access the database's data.

The following are the components of a DBMS:

* Software
* Data
* Procedures
* Database Languages
* Query Processor
* Database Manager
* Database Engine
* Reporting

### **16) What are the different types of database management systems?**

The database management systems can be categorized into several types. Some of the important lists are given below:

* Hierarchical databases (DBMS)
* Network databases (IDMS)
* Relational databases (RDBMS
* Object-oriented databases
* Document databases (Document DB)
* Graph databases
* ER model databases
* NoSQL databases

### **17) What is RDBMS?**

RDBMS stands for Relational Database Management System. It is a database management system based on a relational model. It facilitates you to manipulate the data stored in the tables by using relational operators. RDBMS stores the data into the collection of tables and links those tables using the relational operators easily whenever required. Examples of relational database management systems are Microsoft Access, MySQL, SQL Server, Oracle database, etc.

### **18) What is Normalization in a Database?**

Normalization is used to minimize redundancy and dependency by organizing fields and table of a database.

There are some rules of database normalization, which is commonly known as Normal From, and they are:

* First normal form(1NF)
* Second normal form(2NF)
* Third normal form(3NF)
* Boyce-Codd normal form(BCNF)

Using these steps, the redundancy, anomalies, inconsistency of the data in the database can be removed.

### **19) What is the primary use of Normalization?**

Normalization is mainly used to add, delete or modify a field that can be made in a single table. The primary use of Normalization is to remove redundancy and remove the insert, delete and update distractions. Normalization breaks the table into small partitions and then links them using different relationships to avoid the chances of redundancy.

### **20) What are the disadvantages of not performing database Normalization?**

The major disadvantages are:

The occurrence of redundant terms in the database causes the waste of space in the disk.

Due to redundant terms, inconsistency may also occur. If any change is made in the data of one table but not made in the same data of another table, then inconsistency will occur. This inconsistency will lead to the maintenance problem and effects the ACID properties as well.

### **21) What is an inconsistent dependency?**

An Inconsistent dependency refers to the difficulty of getting relevant data due to a missing or broken path to the data. It leads users to search the data in the wrong table, resulting in an error as an output.

### **22) What is Denormalization in a Database?**

Denormalization is a technique used by database administrators to optimize the efficiency of their database infrastructure. The denormalization concept is based on Normalization, which is defined as arranging a database into tables correctly for a particular purpose. This method allows us to add redundant data into a normalized database to alleviate issues with database queries that merge data from several tables into a single table. It adds redundant terms into the tables to avoid complex joins and many other complex operations.

Denormalization doesn't mean that normalization will not be done. It is an optimization strategy that takes place after the normalization process.

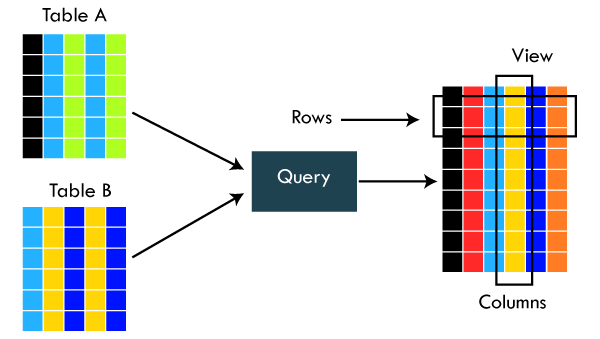
### **23) What are the different types of SQL operators?**

Operators are the special keywords or special characters reserved for performing particular operations. They are also used in SQL queries. We can primarily use these operators within the WHERE clause of SQL commands. It's a part of the command to filters data based on the specified condition. The SQL operators can be categorized into the following types:

* **Arithmetic operators:** These operators are used to perform mathematical operations on numerical data. The categories of this operators are addition (+), subtraction (-), multiplication (\*), division (/), remainder/modulus (%), etc.
* **Logical operators:** These operators evaluate the expressions and return their results in True or False. This operator includes ALL, AND, ANY, ISNULL, EXISTS, BETWEEN, IN, LIKE, NOT, OR, UNIQUE.
* **Comparison operators:** These operators are used to perform comparisons of two values and check whether they are the same or not. It includes equal to (=), not equal to (!= or <>), less than (<), greater than (>), less than or equal to (<=), greater than or equal to (>=), not less than (!<), not greater than (!>), etc.
* **Bitwise operators:** It is used to do bit manipulations between two expressions of integer type. It first performs conversion of integers into binary bits and then applied operators such as AND (& symbol), OR (|, ^), NOT (~), etc.
* **Compound operators:** These operators perform operations on a variable before setting the variable's result to the operation's result. It includes Add equals (+=), subtract equals (-=), multiply equals (\*=), divide equals (/=), modulo equals (%=), etc.
* **String operators:** These operators are primarily used to perform concatenation and pattern matching of strings. It includes + (String concatenation), += (String concatenation assignment), % (Wildcard), [] (Character(s) matches), [^] (Character(s) not to match), \_ (Wildcard match one character), etc.

### **24) What is a view in SQL?**

A view is a database object that has no values. It is a virtual table that contains a subset of data within a table. It looks like an actual table containing rows and columns, but it takes less space because it is not present physically. It is operated similarly to the base table but does not contain any data of its own. Its name is always unique. A view can have data from one or more tables. If any changes occur in the underlying table, the same changes reflected in the views also.



The primary use of a view is to implement the security mechanism. It is the searchable object where we can use a query to search the view as we use for the table. It only shows the data returned by the query that was declared when the view was created.

We can create a view by using the following syntax:

1. CREATE VIEW view\_name AS
2. SELECT column\_lists FROM table\_name
3. WHERE condition;

### **25) What is an Index in SQL?**

An index is a disc structure associated with a table or view that speeds up row retrieval. It reduces the cost of the query because the query's high cost will lead to a fall in its performance. It is used to increase the performance and allow faster retrieval of records from the table. Indexing reduces the number of data pages we need to visit to find a particular data page. It also has a unique value meaning that the index cannot be duplicated. An index creates an entry for each value which makes it faster to retrieve data.

**For example:** Suppose we have a book which carries the details of the countries. If you want to find out information about India, why will you go through every page of that book? You could directly go to the index. Then from the index, you can go to that particular page where all the information about India is given.

### **26) What are the different types of indexes in SQL?**

SQL indexes are nothing more than a technique of minimizing the query's cost. The higher the query's cost, the worse the query's performance. The following are the different types of Indexes supported in SQL:

* Unique Index
* Clustered Index
* Non-Clustered Index
* Bit-Map Index
* Normal Index
* Composite Index
* B-Tree Index
* Function-Based Index

### **27) What is the unique index?**

UNIQUE INDEX is used to enforce the uniqueness of values in single or multiple columns. We can create more than one unique index in a single table. For creating a unique index, the user has to check the data in the column because the unique indexes are used when any column of the table has unique values. This indexing does not allow the field to have duplicate values if the column is unique indexed. A unique index can be applied automatically when a primary key is defined.

We can create it by using the following syntax:

1. CREATE UNIQUE INDEX index\_name
2. ON table\_name (index\_column1, index\_column2,...);

**Example**

1. CREATE TABLE Employee(
2. ID int AUTO\_INCREMENT PRIMARY KEY,
3. Name varchar(45),
4. Phone varchar(15),
5. City varchar(25),
6. );

Suppose we want to make a Phone column as a unique index. We can do this like below:

1. CREATE UNIQUE INDEX index\_name\_phone ON Employee (Phone);

To read more information, [click here](https://www.javatpoint.com/mysql-unique-index).

### **28) What is clustered index in SQL?**

A clustered index is actually a table where the data for the rows are stored. It determines the order of the table data based on the key values that can sort in only one direction. Each table can have only one clustered index. It is the only index, which has been automatically created when the primary key is generated. If many data modifications needed to be done in the table, then clustered indexes are preferred.

To read more information, [click here](https://www.javatpoint.com/mysql-clustered-index).

### **29) What is the non-clustered index in SQL?**

The indexes other than PRIMARY indexes (clustered indexes) are called non-clustered indexes. We know that clustered indexes are created automatically when primary keys are generated, and non-clustered indexes are created when multiple joins conditions and various filters are used in the query. The non-clustered index and table data are both stored in different places. It cannot be able to alter the physical order of the table and maintains the logical order of data.

The purpose of creating a non-clustered index is for searching the data. Its best example is a book where the content is written in one place, and the index is at a different place. We can create 0 to 249 non-clustered indexes in each table. The non-clustered indexing improves the performance of the queries which use keys without assigning the primary key.

### **30) What are the differences between SQL, MySQL, and SQL Server?**

The following comparison chart explains their main differences:

|  |  |  |
| --- | --- | --- |
| **SQL** | **MySQL** | **SQL Server** |
| SQL or Structured Query Language is useful for managing our relational databases. It is used to query and operate the database. | MySQL is the popular database management system used for managing the relational database. It is a fast, scalable, and easy-to-use database. | SQL Server is an RDBMS database system mainly developed for the Windows system to store, retrieve, and access data requested by the developer. |
| SQL first appeared in 1974. | MySQL first appeared on May 23, 1995. | SQL Server first appeared on April 24, 1989. |
| SQL was developed by IBM Corporation. | MySQL was developed by Oracle Corporation. | SQL Server was developed by Microsoft Company. |
| SQL is a query language for managing databases. | MySQL is database software that uses SQL language to conduct with the database. | SQL Server is also a software that uses SQL language to conduct with the database. |
| SQL has no variables. | MySQL can use variables constraints and data types. | SQL Server can use variables constraints and data types. |
| SQL is a programming language, so that it does not get any updates. Its commands are always fixed and remain the same. | MySQL is software, so it gets frequent updation. | SQL Server is also software, so it gets frequent updation. |

### **31) What is the difference between SQL and PL/SQL?**

The following comparison chart explains their main differences:

|  |  |
| --- | --- |
| **SQL** | **PL/SQL** |
| SQL is a database structured query language used to communicate with relational databases. It was developed by IBM Corporations and first appeared in 1974. | PL/SQL or Procedural Language/Structured Query Language is a dialect of SQL used to enhance the capabilities of SQL. Oracle Corporation developed it in the early 90's. It uses SQL as its database language. |
| SQL is a declarative and data-oriented language. | PL/SQL is a procedural and application-oriented language. |
| SQL has no variables. | PL/SQL can use variables constraints and data types. |
| SQL can execute only a single query at a time. | PL/SQL can execute a whole block of code at once. |
| SQL query can be embedded in PL/SQL. | PL/SQL cannot be embedded in SQL as SQL does not support any programming language and keywords. |
| SQL can directly interact with the database server. | PL/SQL cannot directly interact with the database server. |
| SQL is like the source of data that we need to display. | PL/SQL provides a platform where SQL data will be shown. |

### **32) Is it possible to sort a column using a column alias?**

Yes. We can use the alias method in the ORDER BY instead of the WHERE clause for sorting a column.

### **33) What is the difference between clustered and non-clustered indexes in SQL?**

Indexing is a method to get the requested data very fast. There are mainly two types of indexes in SQL, clustered index and non-clustered index. The differences between these two indexes are very important from an SQL performance perspective. The following comparison chart explains their main differences:

|  |  |
| --- | --- |
| **Clustered Index** | **Non-Clustered Index** |
| A clustered index is a table or view where the data for the rows are stored. In a relational database, if the table column contains a primary key, MySQL automatically creates a clustered index named PRIMARY. | The indexes other than PRIMARY indexes (clustered indexes) are called non-clustered indexes. It has a structure separate from the data row. The non-clustered indexes are also known as secondary indexes. |
| Clustered indexes store the data information and the data itself. | Non-clustered indexes stores only the information, and then it will refer you to the data stored in clustered data. |
| There can only be one clustered index per table. | There can be one or more non-clustered indexes in a table. |
| A clustered index determines how data is stored physically in the table. Therefore, reading from a clustered index is faster. | It creates a logical ordering of data rows and uses pointers for accessing the physical data files. Therefore, reading from a clustered index is slower. |
| A clustered index always contains an index id of 0. | A non-clustered index always contains an index id>0. |

To read more information, [click here](https://www.javatpoint.com/mysql-clustered-vs-non-clustered-index).

### **34) What is the SQL query to display the current date?**

There is a built-in function in SQL called GetDate(), which is used to return the current timestamp.

### **35) Which are joins in SQL? Name the most commonly used SQL joins?**

SQL joins are used to retrieve data from multiple tables into a meaningful result set. It is performed whenever you need to fetch records from two or more tables. They are used with SELECT statement and join conditions.

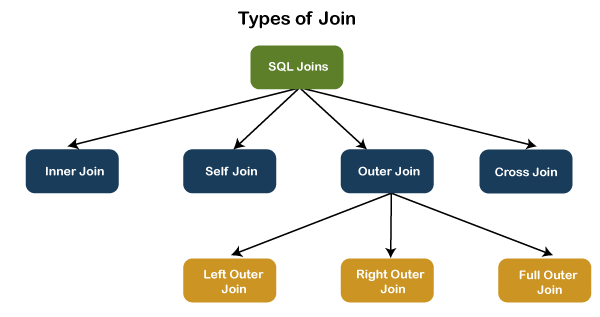
The following are the most commonly used joins in SQL:

* INNER JOIN
* LEFT OUTER JOIN
* RIGHT OUTER JOIN

### **36) What are the different types of joins in SQL?**

Joins are used to merge two tables or retrieve data from tables. It depends on the relationship between tables. According to the ANSI standard, the following are the different types of joins used in SQL:

* INNER JOIN
* SELF JOIN
* LEFT OUTER JOIN
* RIGHT OUTER JOIN
* FULL OUTER JOIN
* CROSS JOIN

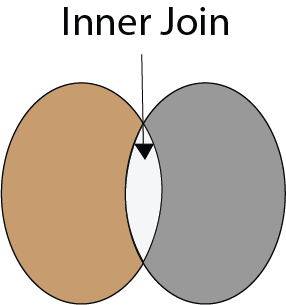


To read more information, [click here](https://www.javatpoint.com/sql-server-joins).

### **37) What is INNER JOIN in SQL?**

Inner join returns only those records from the tables that match the specified condition and hides other rows and columns. In simple words, it fetches rows when there is at least one match of rows between the tables is found. INNER JOIN keyword joins the matching records from two tables. It is assumed as a default join, so it is optional to use the INNER keyword with the query.

The below visual representation explain this join more clearly:



The following syntax illustrates the INNER JOIN:

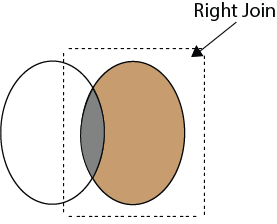
1. SELECT column\_lists
2. FROM table1
3. INNER JOIN table2 ON join\_condition1
4. INNER JOIN table3 ON join\_condition2
5. ...;

To read more information, [click here](https://www.javatpoint.com/mysql-inner-join).

### **38) What is the Right JOIN in SQL?**

The Right join is used to retrieve all rows from the right-hand table and only those rows from the other table that fulfilled the join condition. It returns all the rows from the right-hand side table even though there are no matches in the left-hand side table. If it finds unmatched records from the left side table, it returns a Null value. This join is also known as Right Outer Join.

The below visual representation explain this join more clearly:



The following syntax illustrates the RIGHT JOIN:

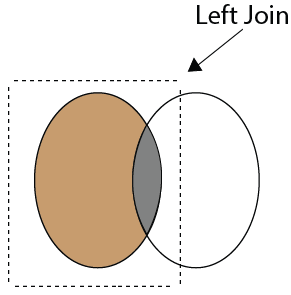
1. SELECT colum\_lists
2. FROM table1
3. RIGHT JOIN table2
4. ON join\_condition;

To read more information, [click here](https://www.javatpoint.com/sql-right-join).

### **39) What is Left Join in SQL?**

The Left Join is used to fetch all rows from the left-hand table and common records between the specified tables. It returns all the rows from the left-hand side table even though there are no matches on the right-hand side table. If it will not find any matching record from the right side table, then it returns null. This join can also be called a Left Outer Join.

The following visual representation explains it more clearly:



The following syntax illustrates the RIGHT JOIN:

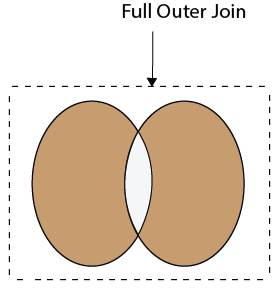
1. SELECT colum\_lists
2. FROM table1
3. LEFT JOIN table2
4. ON join\_condition;

To read more information, [click here](https://www.javatpoint.com/sql-left-join).

### **40) What is Full Join in SQL?**

The Full Join results from a combination of both left and right join that contains all the records from both tables. It fetches rows when there are matching rows in any one of the tables. This means it returns all the rows from the left-hand side table and all the rows from the right-hand side tables. If a match is not found, it puts NULL value. It is also known as FULL OUTER JOIN.

The following visual representation explains it more clearly:



The following syntax illustrates the FULL JOIN:

1. SELECT \* FROM table1
2. FULL OUTER JOIN table2
3. ON join\_condition;

To read more information, [click here](https://www.javatpoint.com/sql-full-join).

### **41) What is a "TRIGGER" in SQL?**

A trigger is a set of SQL statements that reside in a system catalog. It is a special type of stored procedure that is invoked automatically in response to an event. It allows us to execute a batch of code when an insert, update or delete command is run against a specific table because the trigger is the set of activated actions whenever DML commands are given to the system.

SQL triggers have two main components one is action, and another is an event. When certain actions are taken, an event occurs as a result of those actions.

We use the CREATE TRIGGER statement for creating a trigger in SQL. Here is the syntax:

1. CREATE TRIGGER trigger\_name
2. (AFTER | BEFORE) (INSERT | UPDATE | DELETE)
3. ON table\_name FOR EACH ROW
4. BEGIN
5. --variable declarations
6. --trigger code
7. END;

To read more information, [click here](https://www.javatpoint.com/mysql-trigger).

### **42) What is self-join and what is the requirement of self-join?**

A SELF JOIN is used to join a table with itself. This join can be performed using table aliases, which allow us to avoid repeating the same table name in a single sentence. It will throw an error if we use the same table name more than once in a single query without using table aliases.

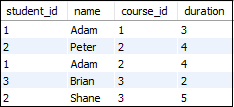
A SELF JOIN is required when we want to combine data with other data in the same table itself. It is often very useful to convert a hierarchical structure to a flat structure.

The following syntax illustrates the SELF JOIN:

1. SELECT column\_lists
2. FROM table1 AS T1, table1 AS T2
3. WHERE join\_conditions;

**Example**

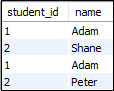
Suppose we have a table 'Student' having the following data:



If we want to get retrieve the student\_id and name from the table where student\_id is equal, and course\_id is not equal, it can be done by using the self-join:

1. SELECT  s1.student\_id, s1.name
2. FROM student AS s1, student s2
3. WHERE s1.student\_id=s2.student\_id
4. AND s1.course\_id**<>**s2.course\_id;

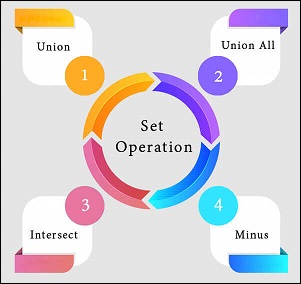
Here is the result:



To read more information, [click here](https://www.javatpoint.com/mysql-self-join).

### **43) What are the set operators in SQL?**

We use the set operators to merge data from one or more tables of the same kind. Although the set operators are like SQL joins, there is a significant distinction. SQL joins combine columns from separate tables, whereas SQL set operators combine rows from different queries. SQL queries that contain set operations are called compound queries. The set operators in SQL are categories into four different types:



**A. UNION:** It combines two or more results from multiple SELECT queries into a single result set. It has a default feature to remove the duplicate rows from the tables. The following syntax illustrates the Union operator:

1. SELECT columns FROM table1
2. UNION
3. SELECT columns FROM table2;

**B. UNION ALL:** This operator is similar to the Union operator, but it does not remove the duplicate rows from the output of the SELECT statements. The following syntax illustrates the UNION ALL operator:

1. SELECT columns FROM table1
2. UNION  ALL
3. SELECT columns FROM table2;

**C. INTERSECT:** This operator returns the common records from two or more SELECT statements. It always retrieves unique records and arranges them in ascending order by default. Here, the number of columns and data types should be the same. The following syntax illustrates the INTERSECT operator:

1. SELECT columns FROM table1
2. INTERSECT
3. SELECT columns FROM table2;

**D. MINUS:** This operator returns the records from the first query, which is not found in the second query. It does not return duplicate values. The following syntax illustrates the MINUS operator:

1. SELECT columns FROM table1
2. MINUS
3. SELECT columns FROM table2;

To read more information, [click here](https://www.javatpoint.com/dbms-sql-set-operation).

### **44) What is the difference between IN and BETWEEN operators?**

The following comparison chart explains their main differences:

|  |  |
| --- | --- |
| **BETWEEN Operator** | **IN Operator** |
| This operator is used to selects the range of data between two values. The values can be numbers, text, and dates as well. | It is a logical operator to determine whether or not a specific value exists within a set of values. This operator reduces the use of multiple OR conditions with the query. |
| It returns records whose column value lies in between the defined range. | It compares the specified column's value and returns the records when the match exists in the set of values. |
| The following syntax illustrates this operator: SELECT \* FROM table\_name WHERE column\_name BETWEEN 'value1' AND 'value2'; | The following syntax illustrates this operator: SELECT \* FROM table\_name WHERE column\_name IN ('value1','value 2'); |

### **45) What is a constraint? Tell me about its various levels.**

The constraint is used to specify the rule and regulations that allows or restricts what values/data will be stored in the table. It ensures data accuracy and integrity inside the table. It enforces us to store valid data and prevents us from storing irrelevant data. If any interruption occurs between the constraint and data action, the action is failed. Some of the most commonly used constraints are NOT NULL, PRIMARY KEY, FOREIGN KEY, AUTO\_INCREMENT, UNIQUE KEY, etc.

The following syntax illustrates us to create a constraint for a table:

1. CREATE TABLE table\_name (
2. column1 datatype constraint,
3. column2 datatype constraint,
4. .........
5. );

SQL categories the constraints into two levels:

**Column Level Constraints:** These constraints are only applied to a single column and limit the type of data that can be stored in that column.

**Table Level Constraints:** These constraints are applied to the entire table and limit the type of data that can be entered.

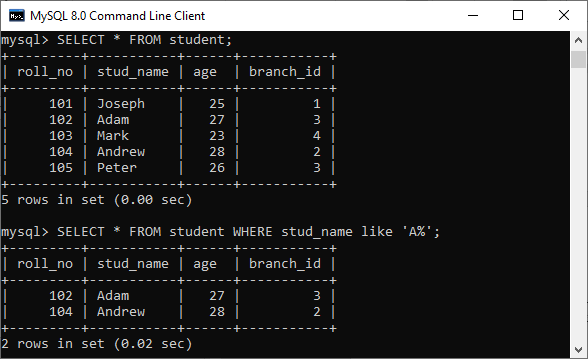
To read more information, [click here](https://www.javatpoint.com/mysql-constraints).

### **46) How to write an SQL query to find students' names start with 'A'?**

We can write the following query to get the student details whose name starts with A:

1. SELECT \* FROM student WHERE stud\_name like 'A%';

Here is the demo example where we have a table named student that contains two names starting with the 'A' character.

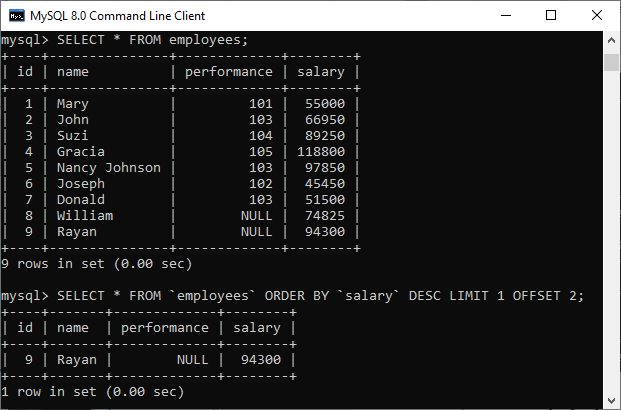


### **47) Write the SQL query to get the third maximum salary of an employee from a table named employees.**

The following query is the simplest way to get the third maximum salary of an employee:

1. SELECT \* FROM `employees` ORDER BY `salary` DESC LIMIT 1 OFFSET 2

Here is the demo example that shows how to get the third maximum salary of an employee.



The following are the alternative way to get the third-highest salary of an employee:

**A. Using LIMIT Keyword**

1. SELECT salary FROM employees
2. ORDER BY salary DESC
3. LIMIT 2, 1;

**B. Using Subquery**

1. SELECT salary
2. FROM
3. (SELECT salary
4. FROM employees
5. ORDER BY salary DESC
6. LIMIT 3) AS Temp
7. ORDER BY salary LIMIT 1;

**C. Using TOP Keyword**

1. SELECT TOP 1 salary
2. FROM
3. (SELECT DISTINCT TOP 3 salary
4. FROM employees
5. ORDER BY salary DESC) AS Temp
6. ORDER BY salary ASC;

### **48) What is the difference between DELETE and TRUNCATE statements in SQL?**

The main difference between them is that the delete statement deletes data without resetting a table's identity, whereas the truncate command resets a particular table's identity. The following comparison chart explains it more clearly:

|  |  |  |
| --- | --- | --- |
| **No.** | **DELETE** | **TRUNCATE** |
| 1) | The delete statement removes single or multiple rows from an existing table depending on the specified condition. | The truncate command deletes the whole contents of an existing table without the table itself. It preserves the table structure or schema. |
| 2) | DELETE is a **DML command.** | TRUNCATE is a **DML command.** |
| 3) | We **can use the WHERE** clause in the DELETE command. | We **cannot use the WHERE** clause with TRUNCATE. |
| 4) | DELETE statement is used **to delete a row** from a table. | TRUNCATE statement is used **to remove all the rows** from a table. |
| 5) | DELETE is **slower** because it maintained the log. | TRUNCATE statement is **faster** than DELETE statement as it deletes entire data at a time without maintaining transaction logs. |
| 6) | You **can roll back** data after using the DELETE statement. | It is **not possible to roll back** after using the TRUNCATE statement. |
| 7) | DELETE query **takes more space.** | TRUNCATE query **occupies less space.** |

To read more information, [click here](https://www.javatpoint.com/delete-vs-truncate-command).

### **49) What is the ACID property in a database?**

The ACID properties are meant for the transaction that goes through a different group of tasks. A transaction is a single logical order of data. It provides properties to maintain consistency before and after the transaction in a database. It also ensures that the data transactions are processed reliably in a database system.

The ACID property is an acronym for Atomicity, Consistency, Isolation, and Durability.

**Atomicity:** It ensures that all statements or operations within the transaction unit must be executed successfully. If one part of the transaction fails, the entire transaction fails, and the database state is left unchanged. Its main features are COMMIT, ROLLBACK, and AUTO-COMMIT.

**Consistency:** This property ensures that the data must meet all validation rules. In simple words, we can say that the database changes state only when a transaction will be committed successfully. It also protects data from crashes.

**Isolation:** This property guarantees that the concurrent property of execution in the transaction unit must be operated independently. It also ensures that statements are transparent to each other. The main goal of providing isolation is to control concurrency in a database.

**Durability:** This property guarantees that once a transaction has been committed, it persists permanently even if the system crashes, power loss, or failed.

To read more information, [click here](https://www.javatpoint.com/acid-properties-in-dbms).

### **50) Is a blank space or zero the same as a NULL value?**

No. The NULL value is not the same as zero or a blank space. The following points explain their main differences:

* A NULL value is a value, which is 'unavailable, unassigned, unknown or not applicable.' It would be used in the absence of any value. We can perform arithmetic operations on it. On the other hand, zero is a number, and a blank space is treated as a character.
* The NULL value can be treated as an unknown and missing value, but zero and blank spaces differ from the NULL value.
* We can compare a blank space or a zero to another blank space or a zero. On the other hand, one NULL may not be the same as another NULL. NULL indicates that no data has been provided or that no data exists.

### **51) What are functions and their usage in SQL?**

SQL functions are simple code snippets that are frequently used and re-used in database systems for data processing and manipulation. Functions are the measured values. It always performs a specific task. The following rules should be remembered while creating functions:

* A function should have a name, and the name cannot begin with a special character such as @, $, #, or other similar characters.
* Functions can only work with the SELECT statements.
* Every time a function is called, it compiles.
* Functions must return value or result.
* Functions are always used with input parameters.

SQL categories the functions into two types:

* **User-Defined Function:** Functions created by a user based on their needs are termed user-defined functions.
* **System Defined Function:** Functions whose definition is defined by the system are termed system-defined functions. They are built-in database functions.

SQL functions are used for the following purposes:

* To perform calculations on data
* To modify individual data items
* To manipulate the output
* To format dates and numbers
* To convert data types

### **52) What is meant by case manipulation functions? Explains its different types in SQL.**

Case manipulation functions are part of the character functions. It converts the data from the state in which it is already stored in the table to upper, lower, or mixed case. The conversion performed by this function can be used to format the output. We can use it in almost every part of the SQL statement. Case manipulation functions are mostly used when you need to search for data, and you don't have any idea that the data you are looking for is in lower case or upper case.

There are three case manipulation functions in SQL:

**LOWER:** This function is used to converts a given character into lowercase. The following example will return the 'STEPHEN' as 'stephen':

1. SELECT LOWER ('STEPHEN') AS Case\_Reault FROM dual;

#### **NOTE: Here, 'dual' is a dummy table.**

**UPPER:** This function is used to converts a given character into uppercase. The following example will return the 'stephen' as 'STEPHEN':

1. SELECT UPPER ('stephen') AS Case\_Reault FROM dual;

**INITCAP:** This function is used to converts given character values to uppercase for the initials of each word. It means every first letter of the word is converted into uppercase, and the rest is in lower case. The following example will return the 'hello stephen' as 'Hello Stephen':

1. SELECT INITCAP ('hello stephen') AS Case\_Reault FROM dual;

### **53) Explain character-manipulation functions? Explains its different types in SQL.**

Character-manipulation functions are used to change, extract, and alter the character string. When one or more characters and words are passed into the function, the function will perform its operation on those input strings and return the result.

The following are the character manipulation functions in SQL:

**A) CONCAT:** This function is used to join two or more values together. It always appends the second string into the end of the first string. For example:

**Input:** SELECT CONCAT ('Information-', 'technology') FROM DUAL;

**Output:** Information-technology

**B) SUBSTR:** It is used to return the portion of the string from a specified start point to an endpoint. For example:

**Input:** SELECT SUBSTR ('Database Management System', 9, 11) FROM DUAL;

**Output:** Management

**C) LENGTH:** This function returns the string's length in numerical value, including the blank spaces. For example:

**Input:** SELECT LENGTH ('Hello Javatpoint') FROM DUAL;

**Output:** 16

**D) INSTR:** This function finds the exact numeric position of a specified character or word in a given string. For example:

**Input:** SELECT INSTR ('Hello Javatpoint', 'Javatpoint');

**Output:** 7

**E) LPAD:** It returns the padding of the left-side character value for right-justified value. For example:

**Input:** SELECT LPAD ('200', 6,'\*');

**Output:** \*\*\*200

**F) RPAD:** It returns the padding of the right-side character value for left-justified value. For example:

**Input:** SELECT RPAD ('200', 6,'\*');

**Output:** 200\*\*\*

**G) TRIM:** This function is used to remove all the defined characters from the beginning, end, or both. It also trimmed extra spaces. For example:

**Input:** SELECT TRIM ('A' FROM 'ABCDCBA');

**Output:** BCDCB

**H) REPLACE:** This function is used to replace all occurrences of a word or portion of the string (substring) with the other specified string value. For example:

**Input:** SELECT REPLACE ( 'It is the best coffee at the famous coffee shop.', 'coffee', 'tea');

**Output:** It is the best tea at the famous tea shop.

### **54) What is the usage of the NVL() function?**

The NVL() function is used to convert the NULL value to the other value. The function returns the value of the second parameter if the first parameter is NULL. If the first parameter is anything other than NULL, it is left unchanged. This function is used in Oracle, not in SQL and MySQL. Instead of NVL() function, MySQL have IFNULL() and SQL Server have ISNULL() function.

### **55) Which function is used to return remainder in a division operator in SQL?**

The MOD function returns the remainder in a division operation.

### **56) What are the syntax and use of the COALESCE function?**

The COALESCE() function evaluates the arguments in sequence and returns the first NON-NULL value in a specified number of expressions. If it evaluates arguments as NULL or not found any NON-NULL value, it returns the NULL result.

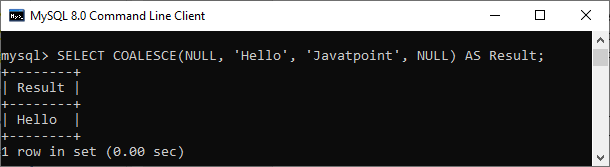
The syntax of COALESCE function is given below:

1. COALESCE (exp1, exp2, .... expn)

**Example:**

1. SELECT COALESCE(NULL, 'Hello', 'Javatpoint', NULL) AS Result;

This statement will return the following output:

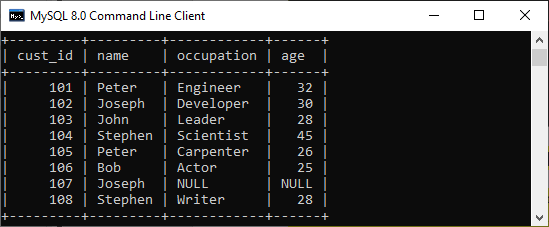


### **57) How do we use the DISTINCT statement? What is its use?**

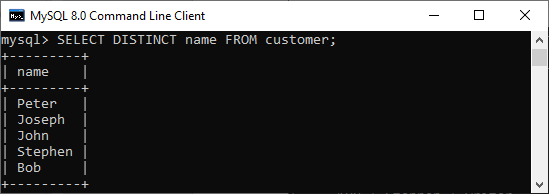
The DISTINCT keyword is used to ensure that the fetched value always has unique values. It does not allow to have duplicate values. The DISTINCT keyword is used with the SELECT statement and retrieves different values from the table's column. We can use it with the help of the following syntax:

1. SELECT DISTINCT column\_lists FROM table\_name WHERE [condition];

Suppose we have a table 'customer' containing eight records in which the name column has some duplicate values.



If we want to get the name column without any duplicate values, the DISTINCT keyword is required. Executing the below command will return a name column with unique values.



### **58) What is the default ordering of data using the ORDER BY clause? How could it be changed?**

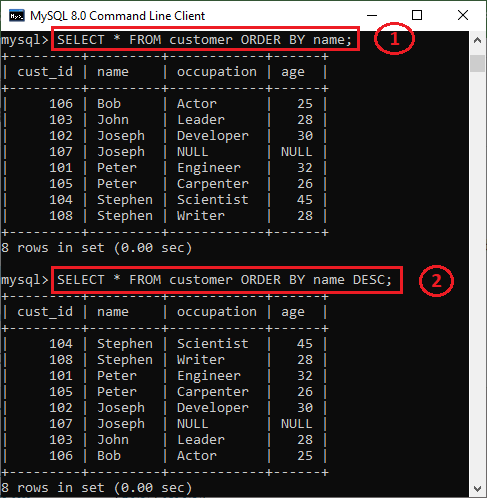
The ORDER BY clause is used to sort the table data either in ascending or descending order. By default, it will sort the table in ascending order. If we want to change its default behavior, we need to use the DESC keyword after the column name in the ORDER BY clause.

The syntax to do this is given below:

1. SELECT expressions FROM tables
2. WHERE conditions
3. ORDER BY expression [ASC | DESC];

We have taken a customer table in the previous example. Now, we will demonstrate the ORDER BY clause on them as well.

In the below output, we can see that the first query will sort the table data in ascending order based on the name column. However, if we run the second query by specifying the DESC keyword, the table's order is changed in descending order.



### **59) Is the following query returns the output?**

1. SELECT subject\_code, AVG (marks)
2. FROM Students
3. WHERE AVG(marks) **>** 70
4. GROUP BY subject\_code;

**Answer:** No. The above query does not return the output because we cannot use the WHERE clause to restrict the groups. We need to use the HAVING clause instead of the WHERE clause to get the correct output.

### **60) What is the difference between the WHERE and HAVING clauses?**

The main difference is that the WHERE clause is used to filter records before any groupings are established, whereas the HAVING clause is used to filter values from a group. The below comparison chart explains the most common differences:

|  |  |
| --- | --- |
| **WHERE** | **HAVING** |
| This clause is implemented in row operations. | This clause is implemented in column operations. |
| It does not allow to work with aggregate functions. | It can work with aggregate functions. |
| This clause can be used with the SELECT, UPDATE, and DELETE statements. | This clause can only be used with the SELECT statement. |

To know more differences, [click here](https://www.javatpoint.com/where-vs-having).

### **61) How many Aggregate functions are available in SQL?**

The aggregate function is used to determine and calculate several values in a table and return the result as a single number. For example, the average of all values, the sum of all values, and the maximum and minimum value among particular groupings of values.

The following syntax illustrates how to use aggregate functions:

1. function\_name (DISTINCT | ALL expression)

**SQL provides seven (7) aggregate functions, which are given below:**

* **AVG():** This function is used to returns the average value from specified columns.
* **COUNT():** This function is used to returns the number of table rows, including rows with null values.
* **MAX():** This function is used to returns the largest value among the group.
* **MIN():** This function is used to returns the smallest value among the group.
* **SUM():** This function is used to returns the total summed values(non-null) of the specified column.
* **FIRST()**: This function is used to returns the first value of an expression.
* **LAST()**: This function is used to returns the last value of an expression.

### **62) What is SQL Injection?**

SQL injection is a type of vulnerability in website and web app code that allows attackers to control back-end operations and access, retrieve, and destroy sensitive data from databases. In this technique, malicious SQL statements are inserted into a database entry field, and once they are performed, the database becomes vulnerable to an attacker. This technique is commonly used to access sensitive data and perform administrative activities on databases by exploiting data-driven applications. It is also known as **SQLi attack.**

Some common examples of SQL injection are:

* Accessing confidential data to modify an SQL query to get desired results.
* UNION attacks to steal data from different database tables.
* Examine the database to extract information regarding the version and structure of the database.

### **63) What is the difference between the RANK() and DENSE\_RANK() functions?**

The **RANK function** determines the rank for each row within your ordered partition in the result set. If the two rows are assigned the same rank, then the next number in the ranking will be its previous rank plus a number of duplicate numbers. For example, if we have three records at rank 4, the next rank listed would be ranked 7.

The **DENSE\_RANK** function assigns a unique rank for each row within a partition as per the specified column value without any gaps. It always specifies ranking in consecutive order. If the two rows are assigned the same rank, this function will assign it with the same rank, and the next rank being the next sequential number. For example, if we have 3 records at rank 4, the next rank listed would be ranked 5.

### **64) Is it possible to implicitly insert a row for the identity column?**

Yes. We can implicitly insert a row for the identity column. Here is an example of doing this:

1. SET IDENTITY\_INSERT TABLE1 ON
2. INSERT INTO demo\_table1 (id, name, branch)
3. SELECT id, name, branch FROM demo\_table2
4. SET IDENTITY\_INSERT OFF

### **65) What are SQL comments?**

Comments are explanations or annotations in SQL queries that are readable by programmers. It's used to make SQL statements easier to understand for humans. During the parsing of SQL code, it will be ignored. Comments can be written on a single line or across several lines.

* **Single Line Comments:** It starts with two consecutive hyphens (--).
* **Multi-line Comments:** It starts with /\* and ends with \*/.