Oracle SQL Tutorial: Get Started Quickly

https://www.udemy.com/blog/oraclesql-tutorial/

This Oracle/SQL tutorial offers a quick way to learn how to use SQL to interact with an Oracle database. Newcomers to programming often consider learning a new programming language to be a difficult process. The best way to learn a new language is to start with the basics and then move to more advanced topics. If you're interested in demystifying the concepts relating to programming, an introductory course designed to teach non-programmers the basics is a good place to start.

Let's get started!

What is the Relationship Between Oracle and SQL?

To understand Oracle SQL, you first need to understand SQL. SQL is an acronym that stands for Structured Query Language, which is a type of programming language. Like all programming languages, SQL has two distinct characteristics: (1) it is comprised of a set of instructions and (2) it is used to solve problems. The instructions you write in SQL are called statements. Programming languages such as C++ and Java are considered general purposes languages because they can solve a variety of different problems. SQL is designed to focus on one—managing data that is stored in a relational database management system or RDMS.

There are numerous flavors of SQL that differ in how they implement the programming language. The list includes DB2, MS SQL Server, MySQL, PostgreSQL, Informix, and of course Oracle. Oracle was the first of these implementations to be offered for commercial purposes (it was initially sold to government agencies). Today, Oracle SQL is considered the standard implementation for interacting with RDBMS. Oracle 11g is the most recent version of Oracle that was released in 2007. Once you learn the basics in this tutorial, you can then extend your skills to include database administration. A comprehensive course in Oracle 11g, which includes the basics for administering an Oracle 11g database, is an excellent choice.

Key Concepts

Before jumping into using the Oracle implementation of SQL, there are some important concepts you should understand because you will undoubtedly come across them when using the language. These concepts are common to other query languages, but some have specific uses in Oracle.

Database – A collection of data that is organized in a specific order.

Table – An object that stores data or information in columns and rows organized in a database.

Datatype – The kind of data stored in a table or column. Oracle includes the following basic datatypes: char, varchar, number, date, and long.

Schema – The structure of an entire database (tables and fields), and how they relate to one another.

Command/Query – An operation that you perform on a database. The most basic operations you can perform on an Oracle database include the following:

- SELECT – Retrieve data
- CREATE – Create a table
- INSERT – Add data
- UPDATE – Replace a row value with the value identified in the SQL expression
- DELETE – Remove rows from a table
- DROP – Remove a table

Query Statement – A series of elements that are executed against a database to perform a specific command.

If you're interested in learning more about these concepts, you could do so in under a week by taking a quick course in SQL.

Querying an Oracle Database with SQL

Learning a new programming language is a lot like learning a new spoken language. You first have to learn the rules of the language. In programming terms, the rules are called syntax. It's important that you follow the rules to prevent errors during processing.

The most common SQL command is to query a database for specific data. You use the SELECT statement to do this:

```
SELECT * FROM Customers;
```

This is what you need to know about this statement:

- *SELECT* is the operation that instructs SQL that we want to extract data from the database.
- The asterisk is a shortcut for referring to all columns in the table. If you want to query a single column, you would use the name of the column instead of an asterisk.
- *FROM* Customers indicates where SQL should get the data.
- The semicolon at the end of the statement signals SQL the conclusion of the statement.
- SQL reserved words such as *SELECT* and *FROM* are not case sensitive, but common practice is to use all capital letters for readability.

SQL commands have specific purposes, but they have a similar structure. Here are more examples:

Example Query 1: CREATE

CREATE TABLE Movies

(

MovieID int,

Title varchar(255)

Genre varchar(255)

Rating varchar(255)

);

This is what you need to know about this query:

- *CREATE TABLE Movies* is the operation that indicates the name of the new table we want created.
- The information between the open and close parentheses identifies the columns in the new table and their datatypes.

Example Query 2: INSERT

INSERT INTO Movies (MovieID, Title, Genre, Rating) VALUES ('0001', 'Matrix Reloaded', 'Science Fiction', 'PG');

This is what you need to know about this query:

- *INSERT INTO Movies (MovieID, Title, Genre, Rating)* is the operation that informs SQL of the data we want to add. The data in the first set of parentheses indicate the columns we want to populate.
- *VALUES ('0001', 'Matrix Reloaded', 'Science Fiction', 'PG')* indicates the list of values we want for each column.

Example Query 3: UPDATE

UPDATE Orders SET CustomerName= 'Jane Done' WHERE CustomerName= 'John Doe';

This is what you need to know about this query:

- *UPDATE Orders* is the operation that instructs SQL which table we want to update.
- *SET CustomerName= 'Jane Doe'* indicates the updated data
- *WHERE CustomerName= 'John Doe'* indicates the data we want to update.

Example Query 4: DELETE

DELETE FROM Contractors WHERE ContractorName= 'Smith Contracting, Inc.';

This is what you need to know about this query:

- *DELETE FROM Contractors* is the operation that indicates where we want to make a deletion.
- *WHERE ContractorName= 'Smith Contracting, Inc.'* indicates the data we want to delete.

Example Query 5: DROP

DROP TABLE Movies;

This is what you need to know about this query:

- *DROP TABLE Movies* is the operation that indicates which table we want to delete.

These are just the basics of using SQL with an Oracle RDBMS. The query language includes other elements that extend basic SQL functions. You are now ready to put your knowledge to work and take a course that teaches the basics of developing a local database.

PHP For Loop: A Lesson in Repetition

https://www.udemy.com/blog/php-for-loop/

PHP: Hypertext Preprocessor (PHP) is a server-side scripting language that is interpreted by a web server, and then optionally displayed in HTML. It basically runs as a program inside of a website. PHP programs do not compile the same as general purpose programming languages, but use similar logic, such as loops that change program logic.

Adding Loops to Your PHP Programs

Computer programs rarely contain statements that run on a continuous path from beginning to end. The more common scenario is that the program makes stops along the path to evaluate conditions and execute statements according to those conditions. PHP supports four types of loop statements:

- for – Repeat for a specific number of times
- while – Repeat until the condition is true
- do…while – Same as While, but executes at least once
- foreach – Repeat elements in an array

For this tutorial the concentration is the for loop. If the PHP language is new to you, you may want to consider learning PHP from scratch first to grasp the basics of the language.

The for loop syntax looks like this:

```
for (declaration; condition; action)
{
Statement to execute;
}
```

As you can see, there are two basic steps for creating a for loop in PHP:

1. Start the loop with the for keyword followed by three parameters (declaration, condition, and action) enclosed in parentheses:

- **Declaration** – Expression that declares the variable that you want to loop and its starting value. Add a semicolon at the end of the expression.
- **Condition** – Expression that sets the condition that you want to use to test against the variable. The loop will continue until this condition is no longer true. Add a semicolon at the end of the expression.
- **Action** – Expression that specifies how the value of the variable is updated. You can use the increment operator (++) to increase the value or the decrement operator (- -) to decrease the value. This expression is written with the operator immediately following the variable, no spaces.

2. Next, you add the statement that you want to execute, enclosed in curly brackets.

Example 1: Simple Loop

This example uses the classic “Hello world!” program to demonstrate how a PHP for loop works.

```
<?php
for ($num=1; $num<=5; $num++)
{
echo “ Hello world!””<br/>;
}
?>
```

In English language terms, this block of code states to start a variable called **num** at **1**, continue the loop as long as the value of **num** is less than or equal to **5**, and at the end of each loop increase the value of **num** by one. For each loop print “Hello world!” on the screen.

Here’s the output:

Hello world!

Hello world!

Hello world!

Hello world!

Hello world!

For each loop there is an output of “Hello world!” to the screen.

Example 2: Nested for Loops

The snippet of code in example 1 contains a single for loop. You can also create nested loops of two or more for loops that add more complexity to your code. This example contains two for loops to create a mathematical table you might recognize from your childhood. You can learn more interesting techniques in a PHP for beginners course.

```
<?php
    echo “<table>”;
    for ($x=1; $num<=10; $x++)
{
    echo “<tr>;
    for ($y=1; $y<=10; $y++)
    {
        echo “<td>”;
```

```
        echo $x*$y;
        echo "</td>";
    }
    echo "</tr>";
}
echo "</table>";
?>
```

Here's the output:

1 2 3 4 5 6 7 8 9 10

2 4 6 8 10 12 14 16 18 20

3 6 9 12 15 18 21 24 27 30

4 8 12 16 20 24 28 32 36 40

5 10 15 20 25 30 35 40 45 50

6 12 18 24 30 36 42 48 54 60

7 14 21 28 35 42 49 56 63 70

8 16 24 32 40 48 56 64 72 80

9 18 27 36 45 54 63 72 81 90

10 20 30 40 50 60 70 80 90 100

You might recognize the output as a multiplication table. This nested for loop consists of two for loops: one that generates the outer 1-10 values (the x variable) and one that generates the inner values (the y variable). The "td" and "tr" in the example are HTML elements that are responsible for creating the table structure (better looking than in this post). Interested in a lesson about these and more HTML elements, try an HTML workshop course.

The examples presented in this tutorial should help you understand how PHP for loops work. As mentioned at the start of this tutorial, the PHP language was developed as a scripting tool to accommodate web development. If you're interested in learning more details about creating PHP programs for websites, you may want to consider a course that teaches the basics of PHP web development.

PHP If Else Tutorial: Extending the If Condition

https://www.udemy.com/blog/php-if-else/

In programming, a statement that executes only when a particular expression is true is called a conditional statement. In PHP, you have several options for adding conditional logic to your programs. When you need your code to test conditions, and then execute a statement based on that result, an if else statement is recommended. This tutorial introduces the PHP if else statement, examines how to write the code in PHP, and provides examples for demonstration and practice. If you are just beginning to learn PHP, you may want to start with a course to learn PHP fundamentals from scratch.

The PHP if else syntax looks like this:

```
if (condition)
    {
    Code to execute if the condition returns true;
    }
else
    {
Code to execute if the condition returns false;
    }
```

Examining an if else Statement in PHP

The logic for an if else statement starts with the if keyword that is followed by a condition, which is enclosed in parentheses.

The next part begins with an open curly bracket that indicates that a block of code is coming up. In this case, the block of code is a statement to execute if the condition returns true. A semicolon is added at the end to indicate the end of the expression. A closed curly bracket is added to signal the end of the code block (this is optional).

Up to this point the if else statement looks just like an if statement. The difference in the two is that the if statement contains an action for the program to take if the statement returns false. After the code block, the else keyword indicates the start of logic for a false return for the condition. This requires another code block, so another open curly bracket is added.

The else code block simply contains a statement to execute if the conditional expression returns false. The statement concludes with a semicolon, and a closed curly bracket ends the block of code.

The formatting in the syntax may have caught your attention. While it is not required and does not affect the operation of the program, indenting the execution statements so that the keywords and execution statements stand out aids in readability. Anyone reading the code can

easily see where each part begins. You can learn more PHP syntax in a course that teaches PHP programming for beginners.

Let's take a look at an example.

Example 1: Basic PHP if else Statement

```
<?php
 $num=50;

if($num == 50)

 {
 echo " Hello Universe!";
 }
else
 {
 echo "Hello World!";
 }
?>
```

Output:

Hello Universe!

The output is "Hello Universe" because the condition is true (equal to 50). If the value of num is changed to 5, the output would be "Hello World! since the condition is now false. This is a basic example of a PHP if else statement. What do you do if your program requires more conditional statements (think multiple choice)? You add more PHP if else statements, of course!

Example 2: Nested PHP if else statements

When your PHP program needs to test more than one condition, one option is to use nested if else statements. Adding multiple if else statements in a PHP program is particularly useful when you need to test several conditions such as with a login process. Check out a course in creating a PHP login script to learn how to do this.

Nesting if else statements adds a bit more complexity to your program. Let's review the logic before we dive into an example.

```
if (condition 1)
    {
    Code to execute condition 1 returns true;
    }
else
    if (condition 2)
        {
        Code to execute if condition 2 returns true;
```

```
      }
else
   {
   Code to execute if all conditions returns false;
   }
```

The difference between this syntax and the basic if else is the additional condition and code to execute if the condition is true. There is also an additional else keyword added to connect the additional code block.

Here's an example:

```
if (totalHoursWorked >= 40)
   {
   $bonus = 100;
   }
else
   if ($totalHoursWorked >= 50)
      {
      $bonus = 200;
      }
else
   {
   $bonus = 50;
   }
```

This example includes two checks for the number of hours an employee has worked, and awards a bonus accordingly. While you are able to omit the open and close brackets, they help keep the code organized as complexity grows, as in this example. You also have the option of writing the additional condition using the elseif keyword.

This example contains only two conditions. As a best practice, try to avoid stringing more than three if else statements. At that point you should consider using a switch statement, which functions like an if statement, but uses a format that is considered more readable for code with multiple check statements. A Learning fundamentals of PHP course that includes conditional statements such as if else and switch will help to advance your PHP knowledge.

Java If Else: An Exercise in Flow Control

https://www.udemy.com/blog/java-if-else-2/

A Java program executes sequentially unless you add code that breaks up this natural flow, and makes it more dynamic. For example, a section of your program may execute code that prints a specific message if the user is younger than 65 years old , and present a different message if they are over 65. You would handle this in your program by adding decision-making logic in the form of control flow or conditional statements. For this tutorial we are going to explore how to control the flow of Java programs using if else statements. This concept is considered a basic concept of the Java language. If you want to enhance your knowledge to include other core concepts, a course in Java fundamentals is recommended.

Before we jump into learning about the if else statement, let's start with the concept on which it is built–the if statement.

An if statement looks like this:

```
int Age = 70;
if (Age > 65)
    {
    System.out.println {"You are ready for retirement!");
    }
```

This snippet of code contains an initialization of a variable **Age** that is set to the number 70. The condition tests for when **Age** is greater than the number 65. If this statement returns true, the message, "You are ready for retirement!" is printed on the screen. If the condition is not met, then the program ignores the code block and continues program execution. Since our variable is currently set to a number that is greater than the 65, the message would be displayed.

You are not limited to a single if statement. The following example shows what Java code with multiple if statements looks like.

Example 1: Multiple if Statements

```
public static void main (String[ ] args)
{
int Age = 18;
if (Age <  65)
    {
    System.out.println {"You are too young to retire!");
    }
if (Age >= 65)
    {
    System.out.println {You are ready for retirement!");
    }
)
```

This example contains two tests and possible outputs:

- If **Age** is less than 65, print “You are too young to retire!” to the screen.
- If **Age** is greater than or equal to 65, print “You are ready for retirement!”

The program starts with the first condition to test if it is true, and executes the output for only one of the conditions, depending on the value of **Age.**

In the if statement, conditions that return false values are ignored and program execution continues. If you want Java to do some more decision making, you can add if else logic.

Like the if statement, if else tests for specific conditions. However, it goes a step further and gives the program something to do in both true and false cases.

The if else statement syntax looks like this:

```
if (condition)
   {
   Statement to execute if condition is true;
   }
else
   {
   Statement to execute if condition is false;
   }
```

The first half mimics the syntax for the if statement. The if keyword indicates a change in program flow is about to change, which is followed by a condition (Boolean expression) that is enclosed in parentheses. You then create your first code block that includes the statement that you want to execute if the condition returns true (all enclosed in brackets). The second code block is similar to the first, but contains a statement that executes if the condition returns false (again all in brackets). The code blocks can contain any expression that returns a Boolean (true or false) value. The expressions always ends with a semicolon.

Formatting tips:

- Extra whitespace is ignored in Java programs, so you can add indentations as needed to aid readability. You can learn more interesting information in an introduction to Java programming course.
- Curly brackets that enclose the statement to execute are optional when there is only a single statement. However, it is good programming practice to add them to make the program easy to read.

Example 2: Basic if else Statement

This example contains an if else statement that prints a classic message on the screen.

```
int num = 4;
if (num >=3)
    {
    System.out.println(“Hello World!”);
    }
else
    {
    System.out.println(“Sorry. No printout!”);
    }
```

This code creates a variable called **num** and gives it a starting value of 1. The program continues the loop as long as the value of **num** is less than 5. At the end of each the loop, the program increases the value of **num** by one. For each loop the program prints “Hello world!”

Here’s the output:

Hello world!

Hello world!

Hello world!

Hello world!

Hello world!

The words “Hello world!” are printed to the screen five times.

Example 3: Compound Statements

Example 3 contains a single statement within a code block. Java also supports compound statements within a single code block.

This is what the syntax looks like:

```
if (condition)
    {
    Statement 1 to execute;
    Statement 2 to execute;
    Statement 3 to execute;
    }
else
    {
```

```
    Statement 4 to execute;
    }
```

Here's an example:

```
public static void main(String[] args)
{
if (num < 0)
    {
    System.out,println("Your value is a negative number.";
    System.out.println("Try another number:");
    num = keyboard.nextInt ( );
    }
else
    {
    System.out.println("You did not enter a negative number. Good job!");
    }
}
```

This example does the following:

- The variable **num** is being evaluated.
- The condition checks if the **num** variable is less than zero.
- If the condition returns true, the program prints "Your value is a negative number. Try another number." to the screen.
- The "num = keyboard.nextInt ( );" statement gets the next value of **num,** and the condition is evaluated again with the new value.
- If the condition returns false, the program prints "You did not enter a negative number. Good job!" to the screen.

Best practices for if else statements with compound statements:

- Make sure all of your statements are enclosed in brackets. As mentioned previously, this aids readability.
- Position the opening and closing brackets for a code block, with appropriate indentation, on a separate line. This adds a bit more coding, but aids readability.

Example 4: Saving Time with Boolean Operators

The examples so far include less than (<) and greater than or equal to (>=) operators to test the conditions, and for good reason. These relational operators work best to test a single condition. If your if else logic requires testing multiple conditions (also called a compound condition), Boolean operators called short-circuiting operators are ideal to use instead of coding a string of conditions (that could become messy!). The short-circuiting operators get their name from how quickly they can evaluate conditions. For example, in some cases Java does not have to evaluate the second condition. In Java, && (Logical AND) and II (Logical OR) are short-circuit operators.

The following table shows when each operator returns a Boolean (true or false) value.

Logical Operator	True when…	False when…
&&	Both conditions return true	• At least one condition returns false
II	At least one condition returns true	• Both conditions returns true • Both conditions return false

The syntax for a Java if else statement that contains a compound condition looks like this:

```
if ((condition 1) logical operator (condition 2))
    {
    Statement to execute if condition is true;
    }
else
   {
   Statement to execute if condition is false;
}
```

In addition to the placement of the logical operator between the conditions, you should also notice another difference right away. A set of parentheses is placed around each condition as well as the entire compound condition. These are the only two differences in comparison to a statement with a single condition.

Here's an example if else statement that contains a compound condition that uses the && operator:

```
if ((a < 5) && (b > 5))
    {
    System.out.println("This is true!");
    }
else
   {
   System.out.println ("This is false!");
   }
```

In this example, Java evaluates the first condition and then the second according to the following:

- First condition true, second condition true; statement returns true.
- First condition false, Java skips second condition; statement returns false.
- First condition true, second condition false; statement returns false.

If the statement returns true, Java prints “This is true!” on the screen. Otherwise, “This is false!” is printed on the screen.

In this next if else statement we use the II operator to combine conditions:

```
if ((a == 5) || (b == 10)
    {
    System.out.println(“This is true!”);
    }
else
    {
    System.out.println(“This is false!”);
}
```

In this example, the decision making works like this:

- First condition true, Java skips second condition; statement returns true.
- First condition false, second condition true; statement returns true.
- First condition false, second condition false; statement returns false.

If the statement returns true, Java prints “This is true!” on the screen. Otherwise, “This is false!” is printed on the screen.

Learning how operators work in Java is essential to programming in the language. You may want to consider a course in Java for complete beginners that teaches the core aspects of the programming language.

Example 5: Nested if else Statements

The examples so far have contained a single condition statement to test. If your program requires a test of multiple conditions, then nested if else statements are what you need. A nested statement is also referred to as a multibranch.

The syntax for nested if else statements:

```
if (condition)
    {
    Statement to execute if condition is true;
    }
    else if
        {
        Statement to execute if condition is true;
        }
    else if
        {
        Statement to execute if condition is true;
        }
else
```

```
   {
   Statement to execute if condition is false;
   }
```

The following example contains two else if statements.

```
public static void main (String args[ ])
  {
int score = 85;
if (score < 70)
  {
  System.out.println("You did not receive a passing score!");}
     else if (score = = 85)
         {
     System.out.println("You received a passing score!");
         }
     else if (score = = 100)
         {
         System.out.println ("You received a perfect score!"); }
else
   {
   System.out.printlin ("I don't know your score!");
   }
```

Here's the output:

You received a passing score!

In this example, the variable **score** is set to **30.** There are four conditions (tests), each with an output if the condition returns true. The final statement is the output if neither of the conditions returns true:

- Test 1: If **score** is less than 70. If true, "You did not receive a passing score!" is printed to the screen.
- Test 2: If **score** equals 85. If true, "You received a passing score! " is printed to the screen.
- Test 3: If **score** is equal to 100. If true, "You received a perfect score! " is printed to the screen.
- Test 4: If **score** does not return true for tests 1-3. If true, "I don't know your score!" is printed to the screen.

There is no limit to the number of else if statements you can add to a program. However, it is always a good idea to write out and test your logic to discover the most efficient method to achieve the goal of the program. Too many nested if else statements can easily become confusing. Creating a flow diagram of your program is one way to test your logic. This method of testing allows you to visualize the flow of your logic.

Closing Thoughts

This tutorial explained different ways to use if else logic in a Java program. Try modifying the examples or create new ones to practice what you have learned. The if else statement is just one type of condition that you can use in Java. You can also use the following control statements:

- Conditional statements: In addition to if and if else, there is also the switch statement that tests a condition against several statements.
- Repetition statements: Loops such as while, for, and do-while.
- Special control statements: These include the break, continue, and return statements.

The samples in this tutorial contain snippets of code. A course that enables you to learn to program in Java is recommended to give you hands-on experience with the programming language.

PL/SQL Tutorial: Expanding the Value of SQL

https://www.udemy.com/blog/plsql-tutorial/

In this lesson you can expect to learn all the basics of coding and running a PL/SQL program. There are examples throughout the lesson that you can use for understanding and as a guide to practice creating your own PL/SQL programs. This tutorial gives a brief history of Oracle SQL, the basis of PL/SQL. An introductory course in Oracle SQL is recommended if you want to learn more about the basics of the SQL language.

What is PL/SQL?

Oracle released Structured Query Language (SQL) for commercial use in the late 1970s, and in less than 10 years later it became the standard for querying data in a relational database management system (RDBMS). Oracle also developed MySQL You may want to participate in a MySQL training course to learn about this language too.

SQL is a powerful programming language, but if you need to construct more complex statements to manipulate your data, you need another tool. Oracle developed PL/SQL in the 1990s to extend the functionality of SQL to include procedural commands.

PL/SQL is a compound acronym that stands for Procedural Language/Structured Query Language. As you would probably expect, a procedural language involves giving the computer directions. In the case of PL/SQL, these directions are in the form of commands that are grouped in units called blocks (more information about these later). PL/SQL allows you to combine procedural constructs (such as IF, Else, Loop, and End Loop) with SQL statements.

The additional functionality you have with PL/SQL offers the following advantages:

- Since your code is encapsulated in blocks, you can easily store it in the database and reuse it in multiple programs.
- More control of your programs through conditional and control statements
- Better flow and more completeness with error handling.
- Portability. Once you write a PL/SQL program, you can run it on any Oracle database server regardless of the operating system.
- Improved performance by way of the PL/SQL engine that takes the multiple statements in a block and processes them at the same time.

How Does PL/SQL Work?

A PL/SQL program consists of one or more blocks. A block is a unit of code that stores both SQL and PL/SQL statements.

Let's look at a basic PL/SQL block using the classic "Hello World" program.

```
BEGIN
  DBMS_OUTPUT.PUT_LINE ('Hello World');
END;
```

The Hello World program contains two keywords that are required for every PL/SQL block: BEGIN and END.

The execution section of the program, defined by the BEGIN keyword, is the body and contains the PL/SQL and SQL statements to execute to accomplish the task. In this case we want to print "Hello World" to the screen. In the statement, **DBMS_OUTPUT.put_line** refers to the package name and procedure. Those concepts are discussed later. Each statement in this section must conclude with a semicolon.

The END keyword marks the end of the block. A semicolon at the end is required here too.

Optional Sections of a PL/SQL Program

In addition to an execution section, a block can also contain declaration and exception sections. These sections are defined by the DECLARE and EXCEPTION keywords, respectively.

Here's an example program that contains the required execution section and optional declaration and exception sections:

```
DECLARE
   Bonus NUMBER (8,2);
   Temp_Id NUMBER (6) = 100;
```

```
BEGIN
   SELECT salary * 0.10 INTO bonus
   FROM employees
   WHERE employee_id = emp_id;
EXCEPTION
   When NO_DATA_FOUND
   THEN
      Null;
END;
```

The DECLARE keyword at the very top of the block indicates the start of a declaration section. Here is where you list all the placeholders you plan to use to accomplish your task. A placeholder can be a variable with a value that can change in the block. In the example block, **Bonus** and **Temp_Id** are variables. Along with declaring their names, you can also define their data type and assign an initial value.

PL/SQL supports a variety of pre-defined data types. The most common are Varchar2, Char, Number, Date, and Long:

- Varchar2 *(maximum_length)* – Stores character data with variable lengths up to 32767 bytes (2000 bytes maximum inserted into a database column with a VARCHAR2 data type).
- Char [*maximum_lengthI)*] – Stores character data of a fixed length up to 32767 bytes (2000 bytes maximum inserted into a database column with a Char data type)..
- Number[(*precision, scale*)] – Stores numbers (fixed or floating-point) without any maximum. The precision parameter refers to the total number of numbers. The scale parameter refers to rounding.
- Date – Stores dates from and including January 1, 4712 B.C. to and including December 32. 4712 A.D.
- Long – Stores character strings with variable lengths. Similar to VARCHAR2, but has a maximum length of 32,760 bytes.

You can learn more about other data types that PL/SQL supports in an Oracle PL/SQL tutorial course.

In the example, both variables are defined with the NUMBER data type; only the Temp_Id is initialized with a starting value. If your block does not require any placeholders, you don't need this section. A semicolon is required at the end of a placeholder declaration. The execution section of this block starts with the keyword BEGIN and contains a SQL SELECT statement.

The EXCEPTION keyword means that the block contains error handling. The exception section of a PL/SQL block includes actions for the system to take when an error occurs. If you know that specific errors may occur in your block, adding an exception section is a good practice. This ensures that your program executes and terminates gracefully. An example of an exception is if your program includes a calculation that requires dividing two numbers, you might consider adding an exception for cases when the divisor is zero. In this case, you would be able to code

your error handling the ZERO_DIVIDE predefined exception that is available in PL/SQL. There are several of these available.

Here's an example of PL/SQL code that contains an exception section:

```
BEGIN
   DBMS_OUTPUT.PUT_LINE(1/0);
EXCEPTION
   When ZERO_DIVIDE
   THEN
      DBMS_OUTPUT.PUT_LINE('Division by zero');
END;
```

In the example, every time the user inputs a value that equates to division by zero, the program throws an error with the message, "Division by zero."

More About Blocks

The PL/SQL language supports anonymous and named blocks.

Anonymous Blocks

The Hello World and Bonus programs above are examples of anonymous blocks. They are called "anonymous" because they do not have a header or name associated with them. This is a significant characteristic because it means that other blocks in the program cannot reference them. Another important feature of anonymous blocks is that they are not stored in the database; they are compiled each time the system loads them into memory. Activities that require one-time processing are good examples of when to use anonymous blocks.

Named Blocks

A named block is stored in the database and can be run over and over again. There are types of named blocks: function and procedure. The difference in the two is that a function always returns a value. A procedure doesn't explicitly return a value. You can pass parameters to a procedure that get modified and are returned back to the program that is calling the procedure. That is not required.

Here's the syntax for a basic named block:

```
CREATE PROCEDURE <named_block>
BEGIN
   <SQL or PL/SQL statement>;
END;
```

Package

A package is a block that organizes your procedures and functions and variables into cohesive units. For example, you may want to store all of your code that deals with employment in a package called HR. This type of package provides the block with a namespace.

Here's the syntax for a package:

```
CREATE OR REPLACE PACKAGE <package> AS
      <variable name and type declaration>;
BEGIN
    <SQL or PL/SQL statement>;
END;
```

Trigger

A chunk of code that is stored in the data dictionary and reacts to some kind of activity in the database. So, when some condition is true, this chunk of code will be executed and the end user might not be aware of it. For example, if you update the salary of some employees, you might want to store that in some special log file. Any time there is any update activity, that trigger gets executed and is written to the log file.

Here's the syntax for a trigger:

```
CREATE OR REPLACE
TRIGGER <trigger_name>
BEFORE OR AFTER
INSERT OR UPDATE OR DELETE
ON <table_name>
BEGIN
  <SQL or PL/SQL statement>;
END;
```

The sections above explained some basic information about how to program using PL/SQL.

Here are some basic tips for writing a PL/SQL program:

- Start with what it is you want to do
- Break up the different parts

Running a PL/SQL Program

Once you write a PL/SQL program, you can run it on any Oracle database server regardless of the operating system.

PL/SQL is run in a PL/SQL environment (compiler) that includes a PL/SQL engine that runs the PL/SQL block. The engine is responsible for separating the parts of the entire block and forwarding them to their respective executor (procedural statement executor or SQL statement

executor on an Oracle server). Once each part is executed, processing returns to the engine where the parts are integrated and the process complete.

The engine can exist on the client side (The PL parts executed on the client and the SQL parts go to the database server and back to the client). You can also create a PL/SQL program that is run inside the database server.

You have three options to use to run a PL/SQL block:

- Write the program in a text file and then execute on your SQL prompt.
- Use an Oracle tool such as Oracle SQL Developer.
- Use a third-party tool such as TOAD.

Conclusion

This tutorial dove right into coding with PL/SQL, and is a great start to learning PL/SQL. A good complement is a course that examines Oracle PL/SQL from scratch so that you can understand the minute details of the programming language.

Oracle vs. MySQL vs. SQL Server: A Comparison of Popular RDBMS

https://www.udemy.com/blog/oracle-vs-mysql-vs-sql-server/

Since their introduction in the 1980s, relational database management systems (RDBMS) have become the standard database type for a variety of industries. As their name implies, these systems are based on the relational model that organizes data into groups of tables referred to as relations. This post explores the history and features of three popular RDBMS: Oracle, MySQL, and SQL Server. The comparison should help you understand the differences between the systems, and if considering implementing a RDBMS, provide you with information that will help make a decision. If you are interested in learning more about how RDBMS work, there are many courses available. For example, an Oracle getting started course can introduce you to this platform and teach you detailed information about how it works.

Summary Feature Comparison

The following table includes information about the Oracle, MySQL, and SQL Server databases and how they compare.

Feature	Oracle	MySQL	SQL Server
Interface	GUI, SQL	SQL	GUI, SQL, Various
Language support	Many, including C, C#, C++, Java, Ruby, and Objective C	Many, including C, C#, C++, D, Java, Ruby, and Objective C	Java, Ruby, Python, VB, .Net, and PHP
Operating System	Windows, Linux, Solaris, HP-UX, OS X, z/OS, AIX	Windows, Linux, OS X, FreeBSD, Solaris	Windows
Licensing	Proprietary	Open source	Proprietary

Oracle

History

IBM was the first company to develop a RDBMS, however, Oracle Corporation made history in 1980 by releasing its RDBMS, Oracle, for commercial use. Just a few years later the company would release a version of its system for IBM computers. Since its exhibition to the RDBMS market, Oracle has consistently led the way. According to Gartner, Oracle owned nearly 50% of the RDBMS market in 2011. In addition to opening up the commercial market for RDBMS, the Oracle Corporation also was the first company to develop a commercial-level version of SQL that was designed to manipulate data in a RDBMS using (at that time) queries and joins.

Features

The first “real” release of the Oracle RDBMS was Oracle 2. This system supported only basic SQL features, and it was written in an assembly language. The following year, and for the next 10 years or so, Oracle Corporation released updates to its flagship database. Probably one of the reasons the Oracle RDBMS has managed to remain at the top of mighty RDBMS is linked to its product updates that are closely tied to changes in the market. Database buzzwords such as “scalable”, “programmable”, “distributed”, and “portable” are also tied to Oracle release. For example, in 1985 support for a client-server model was added in anticipation of a growing acceptance of network communication. As the Internet paved the way for the Digital Era, the Oracle RDBMS was updated to include a native Java virtual machine (JVM).

Oracle Database 12c is the most recent release of the RDBMS, and it includes the following features:

- New data redaction to enhance security of sensitive data
- Introduction of Oracle Advanced Analytics platform
- New database handling for archiving Flash Data Archive (FDA)
- Support for integrating with operating system processor groups
- Support for data pump for database consolidation
- Several enhancements to Oracle Application Express, a rapid-development tool that allows users to develop Web apps using SQL and/or PL/SQL.
- Advanced network compression to enhance performance

If you’re interested in how you code with Oracle SQL, an introduction to Oracle SQL course can provide the basics of the language.

SQL Server

History

Microsoft SQL Server entered the RDBMS market as a serious competitor in the mid 1990s when Microsoft purchased it from Sybase, and then released version 7.0. The companies originally worked together to develop the platform to run on the IBM OS/2 platform. However, Microsoft eventually developed its own operating system (Windows NT), and wanted to work solo to create a database management for it. It would take several years for the Microsoft and Sybase to completely sever their ties. Sybase eventually changed it’s product name so that it would be completely different from the product sold to Microsoft. Microsoft SQL Server version 4.2 was the initial release.

Features

In 2000, Microsoft released SQL Server 2000. The release was a significant milestone for the company because it marked the first release of the product where the original Sybase code was completely replaced. In the same vein as Oracle Corporation, Microsoft has attempted to enhance SQL Server to keep up with changing technology. SQL Server 2005 is an example. The eXtensible Markup Language (XML) received stamp of approval from W3C and started gaining ground in the late 1990s. One of the major new features of SQL Server 2005 was support for XML data. Other notable features of the flagship product include the introduction of SQL Server Always On (data management technology to decrease user downtime), support for structured and semi-structured data, enhanced compression, and several add-ons to support other products on the market. SQL Server 2012 was proclaimed as the last release to include native support for OLE. A SQL Server 2012 essentials course can offer more information about this platform and how to use it.

SQL Server 2014 is the latest release of SQL Server and includes the following features:

- Introduction of In-Memory Online Traction Processing (OLTP), an embedded feature that allows sophisticated database management to enhance performance
- New solutions to handle disaster recovery
- Updated version of SQL Server Data Tools for Business Intelligence (SSDT BI)

MySQL

History

There are two unique aspects of MySQL in comparison to Oracle and SQL Server: it was not originally developed for commercial use and it is an open source database. The database platform was a happenstance as the individuals who developed it started out trying to use mSQL to interface with their database tables, and decided they needed a much more powerful interface. The initial phase of MySQL used an API leveraged from mSQL, enhancements that increased speed considerably, and other features that included the InnoDB storage engine, full text search, portability, and internationalization.

Another difference of the MySQL platform in comparison to the other two is that it is open source. The Digital Age spawned a movement in software development collaboration that has blossomed into a competitive market for databases and other software. According to market reports, there is an excess of 10 million installations of MySQL, which indicates it is quickly moving into the enterprise space.

The ownership of MySQL has transitioned from the product's humble beginnings. The two most notable acquisitions are (1) in 2008 when Sun Microsystems acquired MySQL AB, the company that created MySQL, and (2) in 2010 when Oracle acquired Sun Microsystems.

Features

Oracle and SQL Server are considered tools that favor users with large enterprise systems, while MySQL is considered a tool that appeals most often to individuals interested in managing databases associated with their websites. As with Oracle and SQL Server, MySQL has released updates to its software just about every year. The original version was developed in the mid 1990s. The most notable changes to MySQL was in 2010, the time of the last acquisition in 2010. The enhancements to this release (GA release 5.5) included semisynchronous replication, custom partitioning, improved support for SMP and updates to the InnoDB I/O subsystem. If you are just learning about MySQL, you may be interested in learning more details about it. A MySQL database for beginners course is a good place to start your education.

Conclusion

This comparison shows just how close the databases are in three key areas. Considering your unique situation is probably more relevant for deciding which one to implement than determining which one is best.