**Sample Table: USER**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **NAME** | **AGE** | **ADDRESS** | **SALARY** |
| 1 | Ramesh | 23 | Allahabad | 20000 |
| 2 | Suresh | 22 | Kanpur | 22000 |
| 3 | Mahesh | 24 | Ghaziabad | 24000 |

1. **What is PL/SQL?**

* Pl/SQL stands for "Procedural Language extension of SQL" that is used in Oracle.
* PL SQL is a procedural language which has interactive SQL, as well as procedural programming language constructs like conditional branching and iteration.
* 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.

1. **Compare SQL & PL/SQL?**

|  |  |  |
| --- | --- | --- |
| **Criteria** | **SQL** | **PL/SQL** |
| What it is | Single query or command execution | Full programming language |
| What it comprises | Data source for reports, web pages | Application language to build, format and display report, web pages |
| Characteristic | Declarative in nature | Procedural in nature |
| Used for | Manipulating data | Creating applications |

1. **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.

1. **Types of Trigger based on Row/Table?**

**Row level Trigger**: The trigger would be executed for each row being affected

**Table level Trigger**: The trigger will execute just once when the SQL statement is executed

1. **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.

1. **Explain the uses of database trigger?**

* Audit data modifications.
* Log events transparently.
* Enforce complex business rules.
* Maintain replica tables
* Derive column values
* Implement Complex security authorizations

1. **How many triggers can be applied to a table?**

A maximum of 12 triggers can be applied to one table.

1. **Explain 3 basic parts of a trigger?**

* A triggering statement or event.
* A restriction
* An action

1. **How would you reference column values BEFORE and AFTER you have inserted and deleted triggers?**

Using the keyword “**new.column name**”, the triggers can reference column values by new collection. By using the keyword “**old.column name**”, they can reference column vaues by old collection.

1. **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.

1. **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.

1. **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.

1. **How to disable & delete a trigger name update\_salary?**

ALTER TRIGGER update\_salary DISABLE;

DROP TRIGGER update\_salary

1. **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.

1. **How to create a trigger?**

|  |  |
| --- | --- |
| Syntax | Example |
| 1. **CREATE** [OR REPLACE ] **TRIGGER** trigger\_name 2. {BEFORE | **AFTER** | **INSTEAD** **OF** } 3. {**INSERT** [OR] | **UPDATE** [OR] | **DELETE**} 4. [**OF** col\_name]   **ON** table\_name 5. [REFERENCING OLD **AS** o NEW **AS** n] 6. [**FOR** EACH ROW]   //Specifies it is a Row level Trigger 7. **WHEN** (condition) 8. **DECLARE** 9. Declaration-statements 10. **BEGIN** 11. Executable-statements 12. EXCEPTION 13. Exception-handling-statements 14. **END**; | E.g. This trigger is placed on Sample Table   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. / |

To execute this trigger we need to perform operation on Table using procedure: **Check the salary difference by 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**;  / | 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 |

**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.

Because we are performing update operation on Salary in table and trigger is set on Salary field.

1. **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.

1. **How to pass parameters in procedure?**

There is three ways to pass parameters in procedure:

* **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.
* **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.
* **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.

1. **How to create & run PL/SQL Procedure?**

|  |  |
| --- | --- |
| 1. Syntax 2. **CREATE** [OR REPLACE] **PROCEDURE** procedure\_name  [ (parameter [,parameter]) ] 3. **IS** 4. [declaration\_section] 5. **BEGIN** 6. executable\_section 7. [EXCEPTION 8. exception\_section] 9. **END** [procedure\_name]; | 1. Example 2. **create** or replace **procedure** "INSERTUSER" 3. (id IN NUMBER, 4. **name** IN VARCHAR2) 5. **is** 6. **begin** 7. **insert** **into** user (ID,NAME) **values** (id,**name**); 8. **end**; 9. / |

**PL/SQL program to call procedure:**

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

1. **How to drop PL/SQL Procedure?**

|  |  |
| --- | --- |
| 1. Syntax 2. **DROP** **PROCEDURE** procedure\_name; | 1. Example 2. **DROP** **PROCEDURE** INSERTUSER; |

1. **What is a 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.

1. **How to create a PL/SQL Function?**

|  |  |
| --- | --- |
| Syntax | Example |
| 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]; | 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 execute this 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 Example:**

|  |  |
| --- | --- |
| 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**      IF x > y **THEN**        z:= x;  **ELSE**        Z:= y;  **END** IF;  **RETURN** z;  **END**;  **BEGIN**     a:= 23;     b:= 45;     c := findMax(a, b);     dbms\_output.put\_line(' Maximum of (23,45): ' || c);  **END**;  / | Output:  Maximum of (23,45): 45  Statement processed.  0.02 seconds |

**Recursive Call**

|  |  |
| --- | --- |
| 1. **DECLARE** 2. num number; 3. factorial number; 5. **FUNCTION** fact(x number) 6. **RETURN** number 7. **IS**      f number;  **BEGIN**     IF x=0 **THEN**        f := 1;  **ELSE**        f := x \* fact(x-1);  //Recursive call to self  **END** IF;  **RETURN** f;  **END**;    **BEGIN**     num:= 6;     factorial := fact(num);     dbms\_output.put\_line(' Factorial '|| num || ' is ' || factorial);  **END**;  / | Output:  Factorial 6 is 720  PL/SQL procedure successfully completed. |

1. **How to drop a PL/SQL function?**

|  |  |
| --- | --- |
| **DROP** **FUNCTION** function\_name; | **DROP** **FUNCTION** adder; |

1. **What is a 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 refer 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. **What are 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.

Oracle provides some attributes known as **Implicit cursors** attributes to check the status of DML operations. Some of them are: %FOUND, %NOTFOUND, %ROWCOUNT and %ISOPEN.

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.

|  |  |
| --- | --- |
| Example | Output |
| 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;         dbms\_output.put\_line( total\_rows || ' customers updated ');  **END** IF;  **END**;  / | 6 customers updated  PL/SQL procedure successfully completed. |

1. **What are 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.

|  |  |
| --- | --- |
| Example | Output |
| 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   **FETCH** c\_customers **into** c\_id, c\_name, c\_addr;        EXIT **WHEN** c\_customers%notfound;        dbms\_output.put\_line(c\_id || ' ' || c\_name || ' ' || c\_addr);  **END** LOOP;  **CLOSE** c\_customers;  **END**;  / | 1 Ramesh Allahabad  2 Suresh Kanpur  3 Mahesh Ghaziabad  4 Chandan Noida  5 Alex Paris  6 Sunita Delhi  PL/SQL procedure successfully completed. |

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.
5. **What does closing a cursor do?**

Closing a cursor clears the private SQL area as well as de-allocates memory.

1. **Differentiate between % ROWTYPE and TYPE RECORD?**

**% ROWTYPE** is used when a query returns an entire row of a table or view.

**TYPE RECORD,** on the other hand, is used when a query returns column of different tables or views.

Eg.  TYPE r\_emp is **RECORD** (sno smp.smpno%type,sname smp sname %type)

e\_rec smp **ROWTYPE**

Cursor c1 is select smpno,dept from smp;

e\_rec c1 %**ROWTYPE**

1. **Show the two PL/SQL cursor exceptions?**

Cursor\_Already\_Open

Invaid\_cursor

1. **Show the cursor attributes of PL/SQL?**

%**ISOPEN** : Checks if the cursor is open or not

%**ROWCOUNT** : The number of rows that are updated, deleted or fetched.

%**FOUND** : Checks if the cursor has fetched any row. It is true if rows are fetched

%**NOT** **FOUND** : Checks if the cursor has fetched any row. It is True if rows are not fetched.

1. **Explain PL/SQL Exception Handling?**

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

1. **Example of System Exceptions?**

|  |  |
| --- | --- |
| Example | Output |
| 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);   DBMS\_OUTPUT.PUT\_LINE ('Address: ' || c\_addr);  EXCEPTION  **WHEN** no\_data\_found **THEN**        dbms\_output.put\_line('No such customer!');  **WHEN** others **THEN**        dbms\_output.put\_line('Error!');  **END**;  / | No such customer!  PL/SQL procedure successfully completed.  There is no customer with ID value 8 in database, so the program raises the run-time exception NO\_DATA\_FOUND, which is captured in EXCEPTION block. |

1. **List some predefined exceptions?**

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

1. **What is the importance of SQLCODE and SQLERRM?**

**SQLCODE** returns the value of the number of error for the last encountered error whereas **SQLERRM** returns the message for the last error.

1. **Explain Raise\_application\_error?**

It is a procedure of package **DBMS\_STANDARD** that allows issuing of user\_defined error messages from database trigger or stored sub-program.

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**;
11. **Explain 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;
3. **Mention what are different methods to trace the PL/SQL code?**

Tracing code is a crucial technique to measure the code performance during the runtime. Different methods for tracing includes

* DBMS\_APPLICATION\_INFO
* DBMS\_TRACE
* DBMS\_SESSION and DBMS\_MONITOR
* trcsess and tkproof utilities

1. **Mention what PL/SQL package consists of?**

A PL/SQL package consists of

* PL/SQL table and record TYPE statements
* Procedures and Functions
* Cursors
* Variables ( tables, scalars, records, etc.) and constants
* Exception names and pragmas for relating an error number with an exception
* Cursors

1. **Mention what are the benefits of PL/SQL packages?**

* **Enforced Information Hiding:** It offers the liberty to choose whether to keep data private or public
* **Top-down design:** You can design the interface to the code hidden in the package before you actually implemented the modules themselves
* **Object persistence:** Objects declared in a package specification behaves like a global data for all PL/SQL objects in the application. You can modify the package in one module and then reference those changes to another module
* **Object oriented design:**The package gives developers strong hold over how the modules and data structures inside the package can be used
* **Guaranteeing transaction integrity:**It provides a level of transaction integrity
* **Performance improvement:**The RDBMS automatically tracks the validity of all program objects stored in the database and enhance the performance of packages.

1. **How would you convert date into Julian date format?**

We can use the J format string:

SQL > select to\_char(to\_date(‘29-Mar-2013’,’dd-mon-yyyy’),’J’) as julian from dual;

JULIAN

1. **What is the location of Pre\_defined\_functions?**

They are stored in the standard package called “**Functions, Procedures and Packages**”

1. **Differentiate between SGA and PGA?**

SGA stands for System Global Area whereas PGA stands for Program or Process Global Area. PGA is only allocated 10% RAM size, but SGA is given 40% RAM size.

1. **What are sequences?**

Sequences are used to generate sequence numbers without an overhead of locking. Its drawback is that the sequence number is lost if the transaction is rolled back.

1. **What packages are available to PL SQL developers?**

DBMS\_ series of packages, such as, DBMS\_PIPE, DBMS\_DDL, DBMS\_LOCK, DBMS\_ALERT, DBMS\_OUTPUT, DBMS\_JOB, DBMS\_UTILITY, DBMS\_SQL, DBMS\_TRANSACTION, UTL\_FILE.

1. **Explain mutating table error?**

It occurs when a trigger tries to update a row that it is currently using. It is fixed by using views or temporary tables, so database selects one and updates the other.