- In a PL/SQL Statements like,

dbms_output.put_line(student_info_func);
This line displays the value returned by the function

Exception Handling: PL/SQL provides a feature to handle the Exceptions which occur in a PL/SQL Block known as exception Handling. Using Exception Handling we can test the code and avoid it from exiting abruptly.When an exception occurs, messages which explain its cause is received. PL/SQL Exception message consists of three parts.

1) Type of Exception
2) An ErrorCode
3) A message

By handling the exceptions we can ensure a PL/SQL block does not exit abruptly.

Exception Handling Structure

General Syntax for the exception section:

```
DECLARE
  Declaration part
BEGIN
  Exception part
EXCEPTION
WHEN excep1name THEN
   Error handling statements
WHEN excep2name THEN
   Error handling statements
WHEN Others THEN
  Error handling statements
END;
```

PL/SQLstatements in the Exception Block. When an exception is raised, a search for an appropriate exception handler in the exception section starts. For example in the above example, if the error raised is 'excep1name ', then the error is handled according to the statements under it. Since, it is not possible to determine all the possible runtime errors during testing for the code, the 'WHEN Others' exception is used to manage the exceptions that are not explicitly handled. Only one exception can be raised in a Block and the control does not return to the Execution Section after the error is handled.

```
DELCARE
  Declaration part
BEGIN
DECLARE
     Declaration part
```

```
BEGIN
    Execution part
EXCEPTION
    Exception part
END;
EXCEPTION
  Exception part
END;
```

If there are nested PL/SQL blocks as in the above case, if the exception is raised in the inner block it should be handled in the exception block of the inner PL/SQL block else the control moves to the Exception block of the next upper PL/SQL Block. If none of the blocks handle the exception the program ends abruptly with an error.

Types of Exception. There are 3 types of Exceptions.
1) Named System Exceptions
2) Unnamed System Exceptions
3) User-defined Exceptions

Named System Exceptions
System exceptions are automatically raised by Oracle, when a program violates a RDBMS rule. There are some system exceptions which are raised frequently, so they are pre-defined and given a name in Oracle which are known as Named System Exceptions.

For example: NO_DATA_FOUND and ZERO_DIVIDE are called Named System exceptions.
Named system exceptions are:
1) Not Declared explicitly,
2) Raised implicitly when a predefined Oracle error occurs,
3) Caught by referencing the standard name within an exception-handling routine.

For Example: Suppose a NO_DATA_FOUND exception is raised in a proc, we can write a code to handle the exception as given below.

```
BEGIN
  Execution part
EXCEPTION
WHEN NO_DATA_FOUND THEN
 dbms_output.put_line (' Using SELECT...INTO did not get any row.');
END;
```

Unnamed System Exceptions

Those system exception for which oracle does not provide a name is known as unnamed system exception. These exceptions do not occur frequently. These Exceptions have a code and an associated message.

There are two ways to handle unnamed system exceptions:

1. Using the WHEN OTHERS exception handler, or
2. By associating the exception code to a name and using it as a named exception.

We can assign a name to unnamed system exceptions using a Pragma called EXCEPTION_INIT.

EXCEPTION_INIT will associate a predefined Oracle error number to a programmer defined exception name.

Steps to be followed to use unnamed system exceptions are

They are raised implicitly.

If they are not handled in WHEN others they must be handled explicitly.

To handle the exception explicitly, they must be declared using Pragma EXCEPTION_INIT as given above and handled referencing the user-defined exception name in the exception section.

The general syntax to declare unnamed system exception using EXCEPTION_INIT is:

```
DECLARE
  excep_name EXCEPTION;
  PRAGMA
  EXCEPTION_INIT (excep_name, Err_code);
BEGIN
Execution part
EXCEPTION
 WHEN excep_name THEN
    handle the exception
END;
```

User-defined Exceptions

Apart from system exceptions we can explicitly define exceptions based on business rules. These are known as user-defined exceptions.

Steps to be followed to use user-defined exceptions:

They should be explicitly declared in the declaration section.

They should be explicitly raised in the Execution Section.

They should be handled by referencing the user-defined exception name in the exception section.

Triggers

Definition: A trigger is a PL/SQL block structure which is fired when a DML

statements like Insert, Delete, Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed.

A database triggers has three parts-

1. **Triggering event**(That causes the trigger to be executed)
2. **Condition**(must be satisfied for trigger execution to proceed)
3. **Action**(specify the action to be taken when the trigger executes).

Trigger Syntax:

```
CREATE [OR REPLACE ] TRIGGER name_of_trigger
 {BEFORE | AFTER | INSTEAD OF }
 {INSERT [OR] | UPDATE [OR] | DELETE}
 [OF name_of_col]
 ON table_name
 [REFERENCING OLD AS O NEW AS N]
 [FOR EACH ROW]
 WHEN (condition)
 BEGIN
   --- sql statements --
 END;
```

- **CREATE [OR REPLACE] TRIGGER name_of_trigger –** In PL/SQL to creates a trigger with the given name or overwrites an existing trigger with the same name we use this clause.

- **{BEFORE | AFTER | INSTEAD OF } -** This clause indicates at what time should the trigger get fired. i.e for example: before or after updating a table. INSTEAD OF is used to create a trigger on a view. Before and after cannot be used to create a trigger on a view.
- **{INSERT [OR] | UPDATE [OR] | DELETE} -** This clause determines the triggering event. More than one triggering events can be used together separated by OR keyword. The trigger gets fired at all the specified triggering event.
- **[OF name_of_col] -** This clause is used with update triggers. This clause is used when you want to trigger an event only when a specific column is updated.
- **[ON table_name] -** This clause identifies the name of the table or view to which the trigger is associated.
- **[REFERENCING OLD AS O NEW AS N] -** This clause is used to reference the old and new values of the data being changed. By default, you reference the values as :old.column_name or :new.column_name. The reference names can also be changed from old (or new) to any other user-defined name. You cannot reference old values when inserting a record, or new values when deleting a record, because they do not exist.
- **[FOR EACH ROW] -** It is used to determine whether a trigger must fire

when each row gets affected (i.e. a Row Level Trigger) or just once when the entire sql statement is executed(i.e. a statement level Trigger).

- WHEN (condition) – It is valid only for row level triggers. The trigger is fired only for rows that satisfy the condition specified.

For Example: The classes of a student changes constantly. It is important to maintain the history of the classes of the students.

Types of PL/SQL Triggers

There are two types of triggers based on the level on which it is triggered

1) **Row level trigger -** An event is triggered for each row updated, inserted or deleted.

2) **Statement level trigger -** An event is triggered for each sql statement executed.

PL/SQL Trigger Execution Hierarchy

The following hierarchy is followed when a trigger is fired.

1) Firstly BEFORE statement trigger fires.

2) Next BEFORE row level trigger fires, once for each row affected.

3) Then AFTER row level trigger fires once for each affected row. This event will alternates between BEFORE and AFTER row level triggers.

4) Finally the AFTER statement level trigger fires.

Important Points:

- PL/SQL Block consists of the Declaration section, the Execution section and the Exception Handling section.
- We must have to write the "**SET Serveroutput ON**" command when we start PL/SQL.
- If you use a EXIT statement without WHEN condition, the statements in the loop is executed only once.
- A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the active set.
- A procedure may or may not return any value.
- The major difference between a procedure and a function is that, a function must always return a value, but a procedure may or may not return a value.
- A trigger is triggered automatically when an associated DML statement is executed.

Practice Questions

Objective type questions:

Q1. Which one is not the part of PL/SQL

a) Declare b) BEGIN

c) Start d) End

Q2. PL/SQL is developed by

a) IBM b) ORACLE

c) Microsoft d) none of these

Q3. Which word must be used along with select statement.

a) Goto b) Into

c) Do d) all

Q4. How many types of cursors are there.

a) 2 b)4

c) 5 d) 1

Q5. The work of % FOUND attribute is just opposite to.

a) %CURSOR b) % NOT COUNT

c) %NOT FOUND d) % FOUND COUNT

Very Short answer type questions.

Q1. What is PL/SQL

Q2. How many parts are there in PL/SQL block.

Q3. Why we use Declare in PL/SQL.

Q4. What is the use of & in PL/SQL.

Q5. Where we declare Variables in PL/SQL

Q6. How select statement is used in PL/SQL

Q7. What is the use of exception block.

Q8. Give types of variable in PL/SQL.

Q9. What is trigger.

Q10. How we use triggers.

.

Short answer type questions:

Q1. Differentiate between %TYPE and %ROWTYPE

Q2. What is the use of EXIT statement in PL/SQL.

Q3. What is before trigger.

Q4. Differentiate between implicit and explicit cursor

Q5. Write syntax of for Loop.

Essay type questions:

Q1. What is cursor? What is the use of cursor? Explain explicit cursor with example.

Q2. Explain different types of database triggers with example.

Q3. What is exception? Explain different types of exceptions.

Q4. Explain different types of loop in PL/SQL.

Q5. What is function? How it is different from Procedure? Explain syntax for functions and procedure.

Answers key for objective questions

Q1: c Q2: b Q3: b Q4: a Q5: c