Unit -4

INTRODUCTION : PL / SQL

SQL V/S PL/SQL

- **SQL** is a Structured Query Language used to issue a single query or execute a single insert/update/delete.
- **PL/SQL** is a procedural language used to create applications.
- **SQL** is used to write queries, DDL **and** DML statements.
- **PL/SQL** is used to write program blocks, functions, procedures triggers, **and** packages.

SQL V/S PL/SQL

- *SQL may be considered as the source of data for our reports, web pages and screens.*
- *PL/SQL can be considered as the application language similar to Java or PHP.*
- *SQL is a data oriented language used to select and manipulate sets of data.*
- *PL/SQL is a procedural language used to create applications.*

SQL V/S PL/SQL

- *SQL is executed one statement at a time.*
- *PL/SQL is executed as a block of code.*

- *SQL can be embedded within a PL/SQL program.*
- *But PL/SQL can't be embedded within a SQL statement.*

PL/SQL BLOCK STRUCTURE

- A **PL/SQL block** is defined by the keywords DECLARE, BEGIN, EXCEPTION, and END.
- These keywords divide the **block** into a declarative part, an executable part, and an exception-handling part.
- The declaration section is optional and may be used to define and initialize constants and variables.
- PL/SQL blocks contain three sections
  1. Declare section
  2. Executable section and
  3. Exception-handling section.

DECLARE (Optional)

Declaration of Variable, Constants.

BEGIN

PL/SQL Executable Statements.

EXCEPTION (Optional)

PL/SQL Exception Handler Block.

END;

PL/SQL BLOCK STRUCTURE

- PL/SQL block has the following structure:

```
DECLARE
    Declaration statements
BEGIN
    Executable statements
EXCETION
    Exception-handling statements
END ;
```

EXAMPLE

```
set serveroutput on;
Declare


begin
        dbms_output.put_line('kamani college');
        dbms_output.put_line('BCA Department');


end;
```

EXAMPLE

```
declare
        x number(3);
        y number(3);
begin
            x:=10;
            y:=20;
      dbms_output.put_line(x+y);
  end;
```

PL/SQL WITH TABLE

- create table emp(id number (3),name varchar2(10),salary number(10))
- insert into emp values(4,'dhanak',4000)
- select *from emp
- alter table emp ADD (sal_update number(10))

- declare
- x number(3);
- begin
- x:=10;
- UPDATE EMP SET SAL_UPDATE=SALARY * X WHERE ID=1;
- END;

VARIABLE ,

BASIC DATA TYPE,

CONDITIONS LOOP

THE PL/SQL DELIMITERS

Delimiter	Description
+, -, *, /	Addition, subtraction/negation, multiplication, division
%	Attribute indicator
'	Character string delimiter
.	Component selector
(,)	Expression or list delimiter
:	Host variable indicator
,	Item separator
"	Quoted identifier delimiter
=	Relational operator
@	Remote access indicator
;	Statement terminator
:=	Assignment operator
=>	Association operator
||	Concatenation operator
**	Exponentiation operator

**	Exponentiation operator
<<, >>	Label delimiter (begin and end)
/*, */	Multi-line comment delimiter (begin and end)
--	Single-line comment indicator
..	Range operator
<, >, <=, >=	Relational operators
<>, '=, ~=, ^=	Different versions of NOT EQUAL

THE PL/SQL COMMENTS

- The PL/SQL supports single-line and multi-line comments.
- All characters available inside any comment are ignored by the PL/SQL compiler.

- The PL/SQL single-line comments start with the delimiter -- (double hyphen) and multi-line comments are enclosed by /* and */.

```
DECLARE
   -- variable declaration
   message  varchar2(20):= 'Hello, World!';
BEGIN
   /*
   *  PL/SQL executable statement(s)
   */
   dbms_output.put_line(message);
END;
/
```

PL/SQL BASIC DATA TYPES

S.No	Date Type & Description
1	**Numeric** Numeric values on which arithmetic operations are performed.
2	**Character** Alphanumeric values that represent single characters or strings of characters.
3	**Boolean** Logical values on which logical operations are performed.
4	**Datetime** Dates and times.

CONTROL STRUCTURE

THREE TYPE OF CONTROL STRUCTURE

- [1]Conditional Control
  - *1) IF…THEN…ENDIF*
  - *2) IF…THEN..ELSE…ENDIF*
  - *3) IF…THEN…ELSIF…ENDIF*
  - *4) CASE…ENDCASE*
- [2] Iterative Control
  - [1]Basic LOOP
  - [2]While..LOOP
  - [3]FOR…LOOP
- [3]Sequential Control
  - [1]GOTO Statement

CONDITIONAL CONTROL

- PL/SQL allows the use of an IF statement to control the execution of a block of code.
- PL/SQL has four conditional or selection statement available for decision making:
  - *1) IF…THEN…ENDIF*
  - *2) IF…THEN..ELSE…ENDIF*
  - *3) IF…THEN…ELSIF…ENDIF*
  - *4) CASE…ENDCASE*

1) IF…THEN…ENDIF

- A simple IF statement performs action statement if the result of the condition is TRUE.
- If the condition is FALSE no action is performed , and the program continues with the next statement in the block.

- Syntax:
  - If <condition>THEN

    <action>

    END IF

- EXAMPLE :

```
declare
        no number(3);
begin
        no:=20;
        if (no<50)then
                dbms_output.put_line(' Number is smaller ');
        end if;
end;
/
```

2) IF...THEN..ELSE...ENDIF

- It is an extension of the simple IF statement .
- It provides action statement for the TRUE outcome as well as for the FALSE outcome.

- Syntax:
  - If <condition>THEN

    <action>

    ELSE

    <some other action>;

    END IF;

EXAMPLE:

```
declare
        no number(3);
begin
        no:=20;
        if (no < 70)then
                dbms_output.put_line('smaller');
        else
                dbms_output.put_line('bigest');
        end if;
End;
/
```

3) IF…THEN…ELSIF…ENDIF

- It is an extension to the previous statement .
- When you have many alternatives/option, you can use previously explained statement but the ELSIF alternative is more efficient than the other two.
- Syntax:
  - If <condition>THEN

    <action>

    ELSIF<condition Action>THEN

    <some other action>

    ELSE

    <some other action>;

    END IF;

EXAMPLE

```
declare
        x number(3);
        y number(3);
begin
        x:=200;
        y:=100;
        if (x=y) then
                dbms_output.put_line('equal');
        elsif (x > y)then
                dbms_output.put_line('bigest');
            else
                dbms_output.put_line('smaller');
        end if;
end;
/
```

[4] CASE…ENDCASE

- Like the **IF** statement, the **CASE statement** selects one sequence of statements to execute.
- However, to select the sequence, the **CASE** statement uses a selector rather than multiple Boolean expressions

- Syntax
- CASE selector
  - WHEN 'value1' THEN S1;
  - WHEN 'value2' THEN S2;
  - WHEN 'value3' THEN S3;
  - …
  - ELSE Sn; -- default case

  END CASE;

EX.

```
DECLARE
   grade char(1) := 'B';
BEGIN
   CASE grade
      when 'A' then dbms_output.put_line('Excellent');
      when 'B' then dbms_output.put_line('Very good');
      when 'C' then dbms_output.put_line('Well done');
      when 'D' then dbms_output.put_line('You passed');
      when 'F' then dbms_output.put_line('Better try again');
      else
            dbms_output.put_line('No such grade');
   END CASE;
END;
/
```

ITERATIVE CONTROL &

LOOPING STRUCRURE

ITERATIVE CONTROL

- Iterative control statement perform one or more statements repeatedly , either a certain number of times or until a condition is meet. There three forms of iterative structures:
  - 1)Basic Loop.
  - 2)While...Loop.
  - 3)For...Loop.

1)Basic Loop.

- A basic loop is a loop that is performed repeatedly. once a loop is entered all statement in the loop are performed.

- Syntax:
  - Loop
    - <statement>
    - Exit [when <condition>];
    - Increment statement;
    - END LOOP

EX.

```
declare
        i number(3):=1;
begin
        loop
                exit when(i>=10);
                        dbms_output.put_line(i);
                i:=i+1;
        end loop;
end;
/
```

2)While…Loop.

- While loop has a condition associated with the loop.
- The condition is evaluated and if the condition is true the statement inside the loop are executed.
- Ex.
- While<condition>
  - Loop
    - <loop body statement>
    - Increment statement;
  - End loop

EX.

```
declare
        i number(3):=1;
begin
        while(i<51)
          loop
                dbms_output.put_line(i);
          i:=i+1;
        end loop;
end;
/
```

3) For...Loop.

- We use FOR...LOOP if we want the iterations to occur a fixed number of times. The FOR...LOOP is executed for a range values.
- Syntax:
  - For<variable>IN <start range>....<end range>
  - Loop
    - <loop body statement>
  - End loop;

EX...(1)

```
DECLARE
   i number(1);
BEGIN
   -- outer_loop
      FOR i IN 1..3 LOOP
         dbms_output.put_line('i is: '| |  i );
      END loop ;
END;
/
```

EX...(2)

```
DECLARE
   i number(1);
   j number(1);
BEGIN
   -- outer_loop
   FOR i IN 1..3 LOOP
      -- inner_loop
      FOR j IN 1..3 LOOP
         dbms_output.put_line('i is: '|| i || ' and j is: ' || j);
      END loop ;
   END loop ;
END;
/
```

EX. (3)

```
declare
        i number(3):=1;
begin
            i:=100;
               for i in reverse 5..10
            loop
              dbms_output.put_line(i);
            end loop;
end;
/
```

SEQUENTIAL CONTROL
GOTO STATEMENT

- A **GOTO statement** with a label may be used to pass control to another part of the program.
- GOTO statement is the program are not very widely used and are not recommended.
- Syntax.
  - GOTO <lable>

EX.

```
declare
      x number(3):=10;
begin
       loop
              x:=x+3;
              if x > 20 then
                  goto stop;
             end if;
        end loop;
        <<stop>>
              dbms_output.put_line('OUTSIDE LOOP...');
end;
/
```

PROGRAM IN PL/SQL

PRO-1 NEXT VALUES GENERETE

```
DECLARE
   x number := 10;
BEGIN
   LOOP
      dbms_output.put_line(x);
      x := x + 10;
      IF x > 50 THEN
         exit;
      END IF;
   END LOOP;
   -- after exit, control resumes here
   dbms_output.put_line('After Exit x is: ' || x);
END;
/
```

- O /p :
- 10
- 20
- 30
- 40
- 50
- After Exit x is: 60 Statement processed.

PRO-2 FACTORIAL PROGRAM

```
declare
        i number(4):=1;
        n number(4):=5;
        f number(4):=1;
 begin
    for i in 1..n
    loop
        f:=f*i;
           Dbms_output.put_line('the factorial of '||i||' is:'||f);
     end loop;
end;
/
```

- **Output :**

- the factorial of 5 is:120
- Statement processed.

PRO.-3 ODD EVEN NUMBER

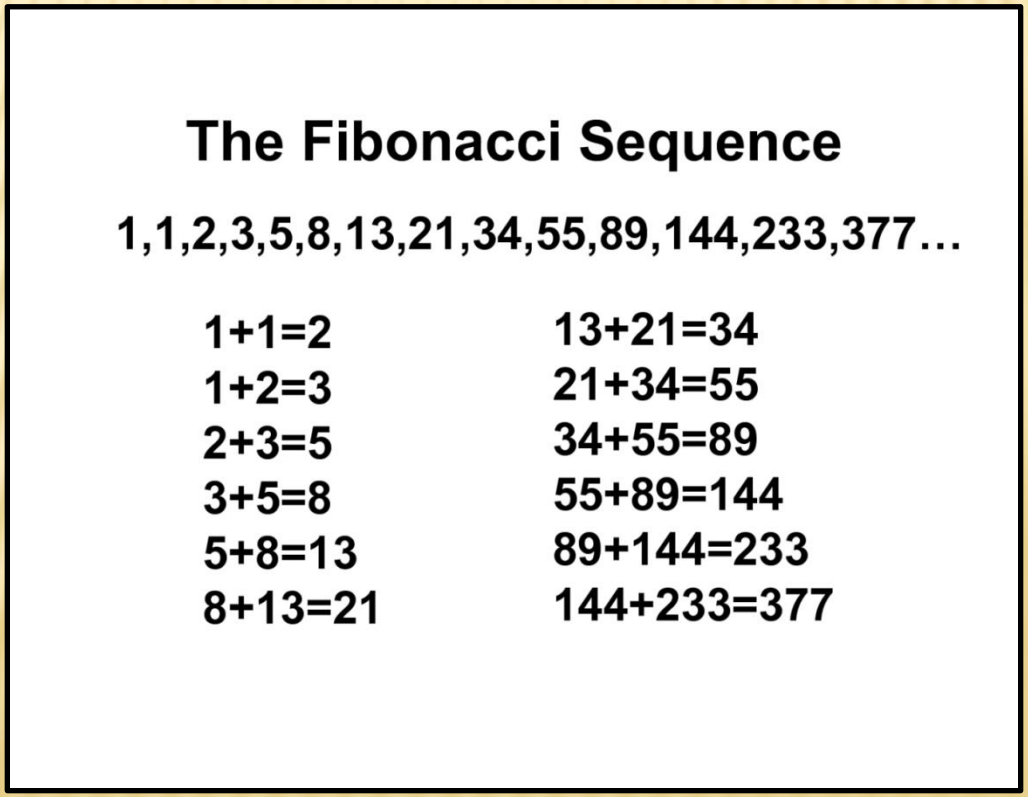
```
declare
BEGIN
    for i in 1..10
    loop
      if mod(i,2) = 0 then
         dbms_output.put_line(i||' is an even number');
      else
         dbms_output.put_line(i||' is an odd number');
     end if;
    end loop;
END;
/
```

- 1 is an odd number
- 2 is an even number
- 3 is an odd number
- 4 is an even number
- 5 is an odd number
- 6 is an even number
- .....
- Statement processed.

PRO-4 BLOCK TO GENERATE FIBONACCI SERIES.

```
declare
      a number:= 0 ;
       b number:= 1;
       c number;
begin
        dbms_output.put(a||' '||b||' ');
        for i in 3..10
      loop
           c := a + b;
              dbms_output.put(c||' ');
          a := b;
           b := c;
       end loop;
          dbms_output.put_line(' ');
end;
/
```

- Output:
- 0 1 1 2 3 5 8 13 21 34
- PL/SQL procedure successfully completed



PRO-5 FIND SUM AND AVERAGE OF THREE NUMBERS.

```
declare
        a number:=1;
        b number:=2;
        c number:=3;
        sm number;
        av number;
begin
        sm:=a+b+c;
        av:=sm/3;
            dbms_output.put_line('Sum = '||sm);
            dbms_output.put_line('Average = '||av);
end;
/
```

- **Output :**
- Sum = 6
- Average = 2
- PL/SQL procedure successfully completed.

PRO.6 FIND REVERSE OF A NUMBER

```
declare
        N number;
        S NUMBER := 0;
        R NUMBER;
        K number;
begin
        N := 1234;
        K := N;
        loop
            exit WHEN N = 0;
            S := S * 10;
           R := MOD(N,10);
           S := S + R;
         N := TRUNC(N/10);
        end loop;
              dbms_output.put_line('THE REVERSED DIGITS OF '||K||' = '||S);
end;
/
```

- **Output :**

- THE REVERSED DIGITS OF 1234 = 4321
- Statement processed.

EX.2 REVERSED NUMBER

```
declare
        i number(3):=1;
begin
         dbms_output.put_line('THE REVERSED DIGITS OF 5 6 7 8 9 10 is  ');

         i:=100;
           for i in reverse 5..10
         loop
           dbms_output.put_line(i);
         end loop;
end;
/
```

PRIME NUMBER

```
DECLARE
   i NUMBER(3);
   j NUMBER(3);
BEGIN
dbms_output.Put_line('The prime numbers are:');
          dbms_output.new_line;
   i := 2;
   LOOP
      j := 2;
      LOOP
         EXIT WHEN( ( MOD(i, j) = 0 )
                  OR ( j = i ) );
         j := j + 1;
      END LOOP;
      IF( j = i )THEN
        dbms_output.Put(i||'   ');

      END IF;
      i := i + 1;
      exit WHEN i = 50;
   END LOOP;
          dbms_output.new_line;
END;
/
```

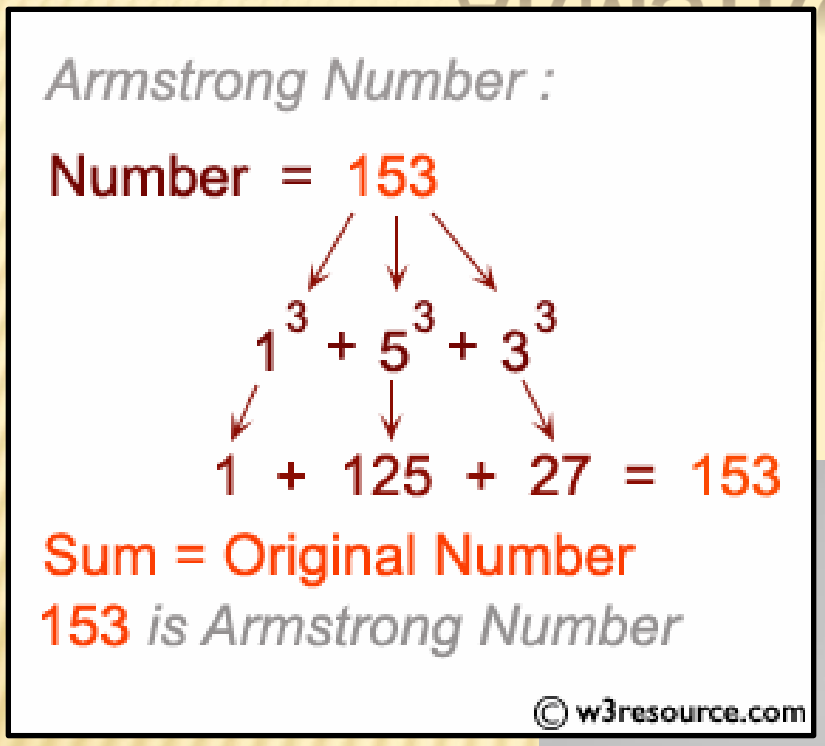
PRIME NUMBER

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

ARMSTRONG NUMBER

```
declare
   n number:=407;
   s number:=0;
   r number;
   len number;
   m number;

begin
   m:=n;

   len:=length(to_char(n));

   while n>0
   loop
      r:=mod(n,10);
      s:=s+power(r,len);
      n:=trunc(n/10);
   end loop;

   if m=s
   then
      dbms_output.put_line('armstrong number');
   else
      dbms_output.put_line('not armstrong number');
   end if;

end;
/
```

ARMSTRONG NUMBER




```
Armstrong Numbers between 1 and 1000
1
153
370
371
407
```

% TYPE AND % ROWTYPE

% TYPE

- →*%TYPE is used to **declare a field with** the same type as*
- → *that of a specified table's column:*

- *Ex.*
- declare
- no emp.id %type;
- nm emp.name %type;
- sal emp.salary %type;
- begin
- select id,name,salary into no,nm,sal from emp WHERE ROWNUM = 1;
- dbms_output.put_line(no);
- dbms_output.put_line(nm);
- dbms_output.put_line(sal);
- end;
- /

%ROWTYPE

- → *%ROWTYPE is used to **declare a record with** the same types as*
- → *found in the specified database table, view or cursor:*

- *Ex.*

```
declare
        myr emp %rowtype;
begin
        select id,name,salary into myr.id,myr.name,myr.salary from emp WHERE ROWNUM = 1;
        dbms_output.put_line('id is:'||myr.id);
        dbms_output.put_line('name is:'||myr.name);
        dbms_output.put_line('salary is:'||myr.salary);
end;
/
```

USING CURSOR (IMPLICIT, EXPLICIT)

WHAT IS CURSOR ?

- The oracle engine uses a work area for its internal processing in order to execute an SQL statement. This work area is call ***CURSOR***.
- You cannot use a SELECT INTO statement when the query returns/retrieve more than one row.
- Cursors are pointer to a memory area that maintain information returned from the query.
- The data stored in the cursor memory is call the 'ACTIVE DATA SET'.

TYPES OF CURSOR

- **(1)implicit cursor** (SQL cursor :open and managed by oracle)
- **(2)Explicit cursor** (user defined cursor : open and managed by user)

(1)IMPLICIT CURSOR

- It is a SQL cursor :open and managed by oracle engine internally.
- Implicit cursor using SELECT statement returning one row of data.
- The SQL cursor/implicit cursor four attributes:
  - SQL%found
  - SQL%notfound
  - SQL%rowcount
  - SQL%ISOPEN

Cursor Attribute	Cursor Variable	Description
%ISOPEN	SQL%ISOPEN	Oracle engine automatically open the cursor If cursor open **return TRUE** otherwise **return FALSE.**
%FOUND	SQL%FOUND	If SELECT statement return one or more rows or DML statement (INSERT, UPDATE, DELETE) affect one or more rows If affect **return TRUE** otherwise **return FALSE.** If not execute SELECT or DML statement **return NULL.**
%NOTFOUND	SQL%NOTFOUND	If SELECT INTO statement return no rows and fire no_data_found PL/SQL exception before you can check SQL%NOTFOUND. If not affect the row **return TRUE** otherwise **return FALSE.**
%ROWCOUNT	SQL%ROWCOUNT	Return the number of rows affected by a SELECT statement or DML statement (insert, update, delete). If not execute SELECT or DML statement **return NULL.**

EXAMPLE (IMPLICIT CURSOR)

```
declare
      aa emp%rowtype;
      cursor c1 is select *from emp;
begin
     open c1;
     if c1%found then
          dbms_output.put_line('cursor is open');
     else
          dbms_output.put_line('cursor is close');
   end if;
end;
```

o/p:

```
         cursor is close
      Statement processed.
```

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE emp SET salary = salary + 500;
  IF sql%notfound THEN
    dbms_output.put_line('no employee selected');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' employee selected ');
  END IF;
END;
/
```

(2)EXPLICIT CURSOR

- Explicit Cursor which are construct/manage by user itself call explicit cursor.
- For queries that return multiple rows , you have to explicitly create a cursor.
- Four action can be perform on explicit cursor:
  - Declare the cursor
  - Open the cursor
  - Fetch the data from cursor
  - Close the cursor

EXAMPLE (EXPLICIT CURSOR)

```
declare
        e_nm emp.name%type;
        e_sl emp.salary%type;
       cursor c1 is select name,salary from emp;
begin
       open c1;
       fetch c1 into e_nm,e_sl;
                dbms_output.put_line('salary of '|| e_nm ||' is '||e_sl);
        fetch c1 into e_nm,e_sl;
                dbms_output.put_line('salary of '|| e_nm ||' is '||e_sl);
       fetch c1 into e_nm,e_sl;
                 dbms_output.put_line('salary of '|| e_nm ||' is '||e_sl);
end;
```

O/p: salary of aaa is 3000
salary of bbb is 2000
salary of bbb is 5000
is Statement processed.

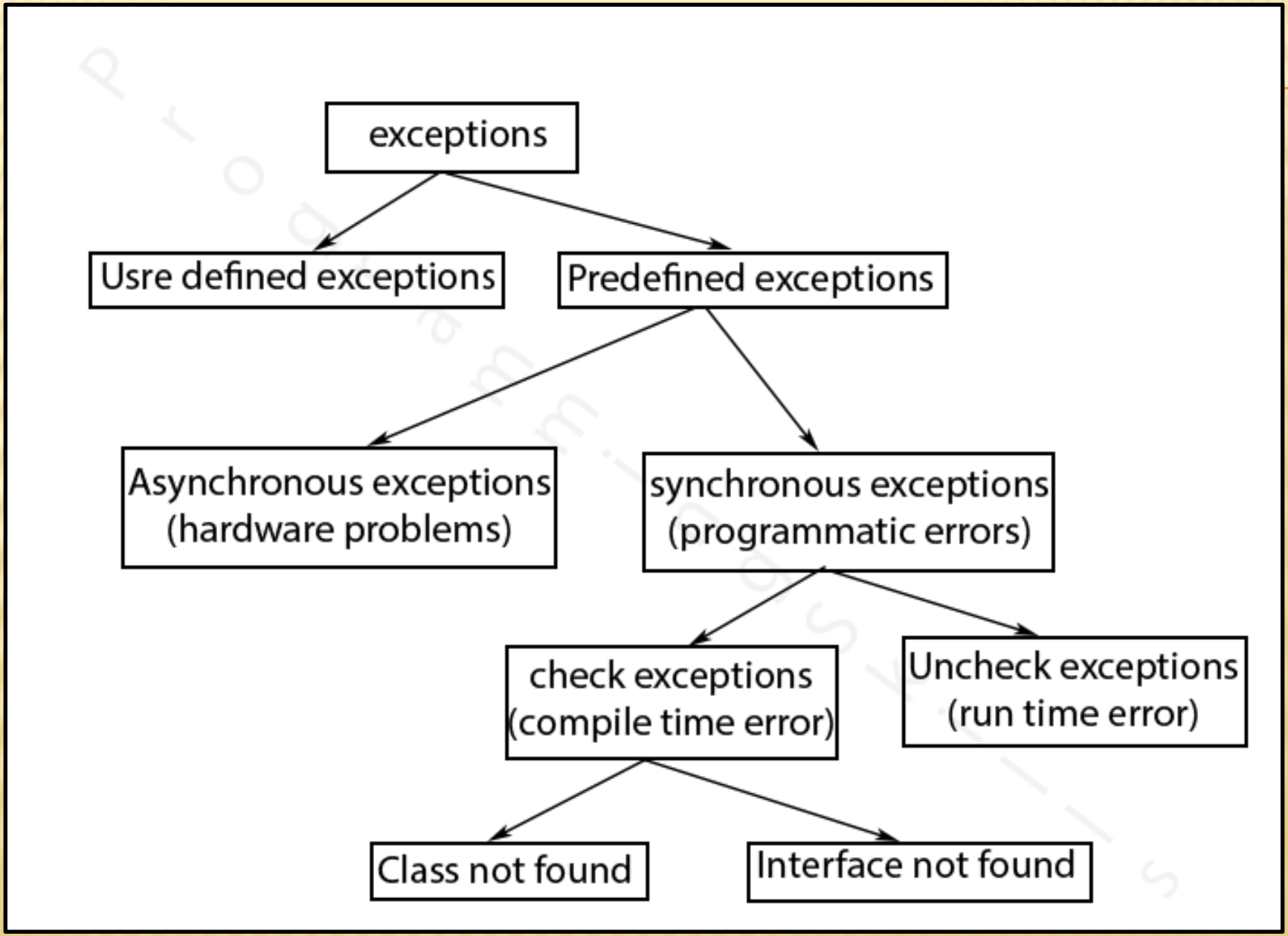
EXCEPTION HANDLING

EXCEPTION HANDLING IN PL/SQL

- An exception is an error condition during a program execution.
- PL/SQL supports programmers to catch such conditions using **EXCEPTION** block in the program and an appropriate action is taken against the error condition.
- There are two types of exceptions
  - System-defined exceptions
  - User-defined exceptions

TYPES OF EXCEPTION

- [1] pre-define exception
- [2] user-name exception
- [3] user-define exception
- [4] customize exception


exceptions
Usre defined exceptions
Predefined exceptions
Asynchronous exceptions
(hardware problems)
synchronous exceptions
(programmatic errors)
check exceptions
(compile time error)
Uncheck exceptions
(run time error)
Class not found
Interface not found

EXCEPTION HANDLING

syntex

```
DECLARE
   <declarations section>
BEGIN
   <executable command(s)>
EXCEPTION
   <exception handling goes here >
   WHEN exception1 THEN
      exception1-handling-statements
   WHEN exception2  THEN
      exception2-handling-statements
   WHEN exception3 THEN
      exception3-handling-statements
   ........
   WHEN others THEN
      exception3-handling-statements
END;
```

- *EX.* *EXCEPTION HANDLING*

```
declare
    e_name emp.name%type;
    e_salary emp.salary%type;
begin
    select name into e_name from emp;
exception
  when no_data_found then
       dbms_output.put_line('record does not exits');
  when too_many_rows then
       dbms_output.put_line('multiple rows retrieved');
  when others then
        dbms_output.put_line('errors in retrieval');
end;
```