MySQL PL/SQL control statements

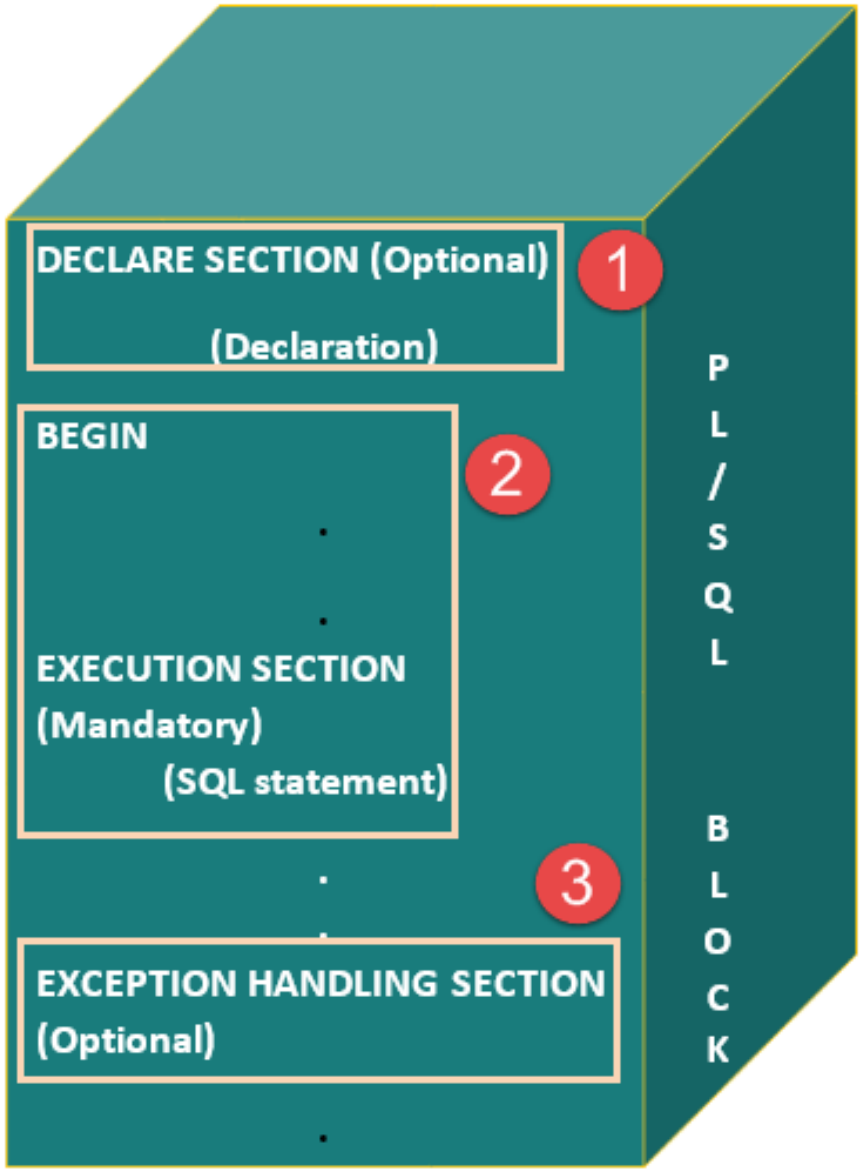
By Prof. B.A.Khivsara

PL/SQL Introduction

- PL/SQL is a combination of SQL along with the procedural features of programming languages.
- Basic Syntax of PL/SQL which is a **block-structured** language; this means that the PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub-parts
- Every PL/SQL statement ends with a semicolon (;).
- Following is the basic structure of a PL/SQL block −

```
DECLARE <declarations section>
 BEGIN <executable command(s)>
EXCEPTION <exception handling>
END;
```

Pl/SQL Block structure



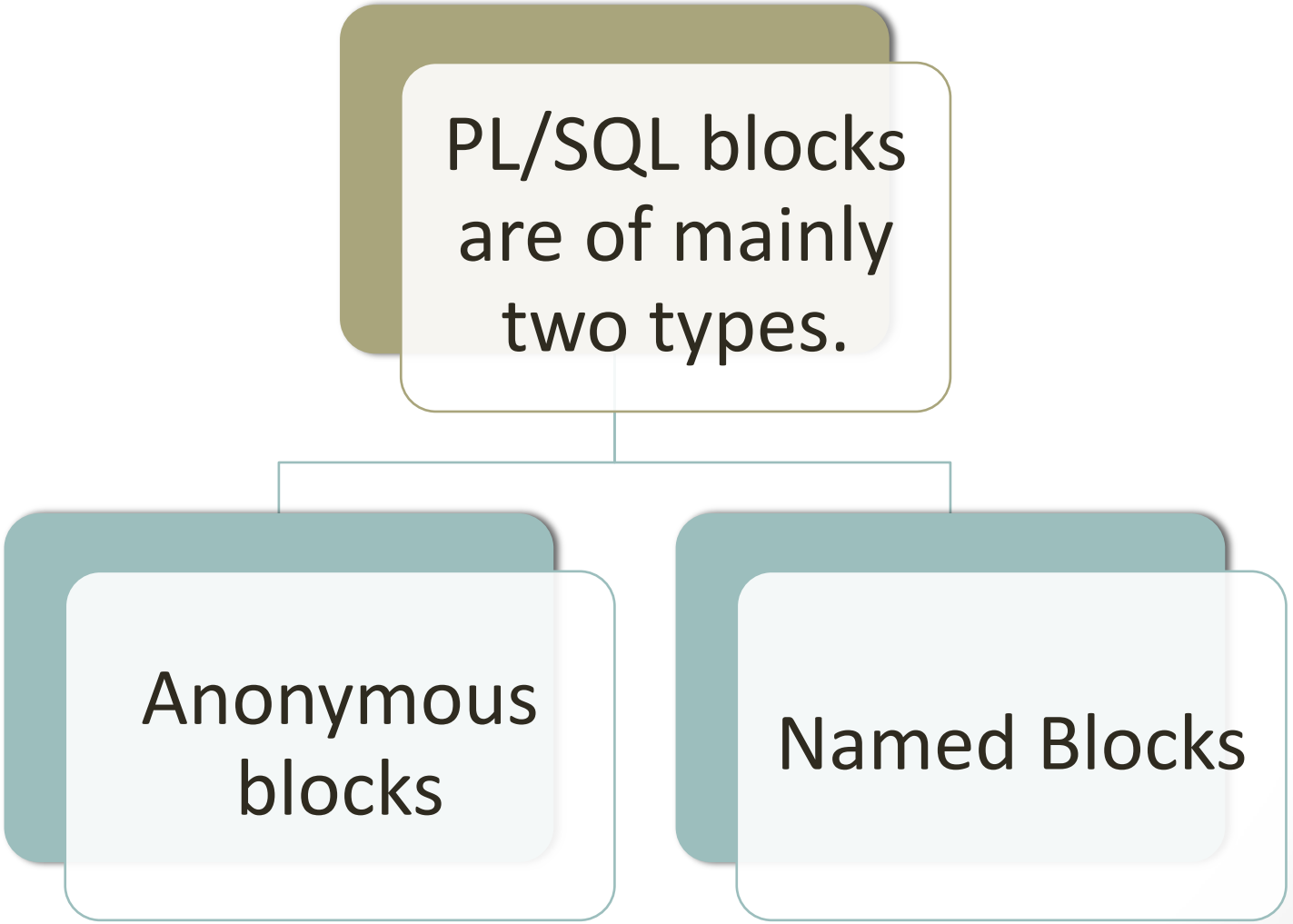
Pl/SQL Block structure Explanation

Sections	Description
Declarations	• This section starts with the keyword DECLARE. • It is an optional section and defines all variables, cursors, and other elements to be used in the program.
Executable Commands	• This section is enclosed between the keywords BEGIN and END and it is a mandatory section. • It consists of the executable PL/SQL statements of the program. • It should have at least one executable line of code.
Exception Handling	• This section starts with the keyword EXCEPTION. • This optional section contains exception(s) that handle errors in the program.

The 'Hello World' Example

```
DECLARE
    msg varchar2(20):= 'Hello, World!';
 BEGIN
    dbms_output.put_line(message);
 END; /
```

Types of PL/SQL block



Anonymous blocks: Unnamed

Anonymous blocks are PL/SQL blocks which do not have any names assigned to them.

They need to be created and used in the same session because they will not be stored in the server as a database objects.

Since they need not to store in the database, they need no compilation steps.

They are written and executed directly, and compilation and execution happen in a single process.

Anonymous blocks: Unnamed

Below are few more characteristics of Anonymous blocks.

- These blocks don't have any reference name specified for them.
- These blocks start with the keyword 'DECLARE' or 'BEGIN'.
- These blocks can have all three sections of the block, in which execution section is mandatory, the other two sections are optional.

Named blocks:

Named blocks are having a specific and unique name for them.

They are stored as the database objects in the server.

Since they are available as database objects, they can be referred to or used as long as it is present in the server.

The compilation process for named blocks happens separately while creating them as a database objects.

Named blocks:

Below are few more characteristics of Named blocks.

- These blocks can be called from other blocks.
- The block structure is same as an anonymous block, except it will never start with the keyword 'DECLARE'. Instead, it will start with the keyword 'CREATE' which instruct the compiler to create it as a database object.
- These blocks can be nested within other blocks. It can also contain nested blocks.

Named blocks are basically of two types:

- Procedure
- Function

Unnamed block Examples

Not possible in MySQL but possible with oracle SQL

Unnamed block Examples-SQL

```
SQL>  declare                                   // For loop
  2   a number:=1;
  3   begin
  4   for a in 1..10 loop
  5   dbms_output.put_line(a);
  6   end loop;
  7   end;

SQL>  declare                                   //Simple loop
  2   a number:=1;
  3   begin
  4  loop
  5   dbms_output.put_line(a);
  6  a:=a+1;
  7  exit when a>10;
  8  end loop;
  9  end;
```

Unnamed block Examples-SQL

```
SQL>  declare                                   //While loop
  2   a number:=1;
  3   begin
  4  while a<11 loop
  5    dbms_output.put_line(a);
  6   a:=a+1;
  7  end loop;
  8  end;

SQL>  declare                                   // if-else
  2  a number(4);
  3  begin
  4  for a in 5..15 loop
  5  if mod(a,5)=0 then
  6  dbms_output.put_line(a);
  7  else
  8  dbms_output.put_line('value'||a);
  9  end if;
 10  end loop;
 11  end;
```

Unnamed block Examples-MySQL

We can use stored procedure instead of unnamed block in MySQL

Stored Procedure Syntax

CREATE PROCEDURE sp_name ([proc_parameter: [IN | OUT | INOUT] param_name data_type])

Begin

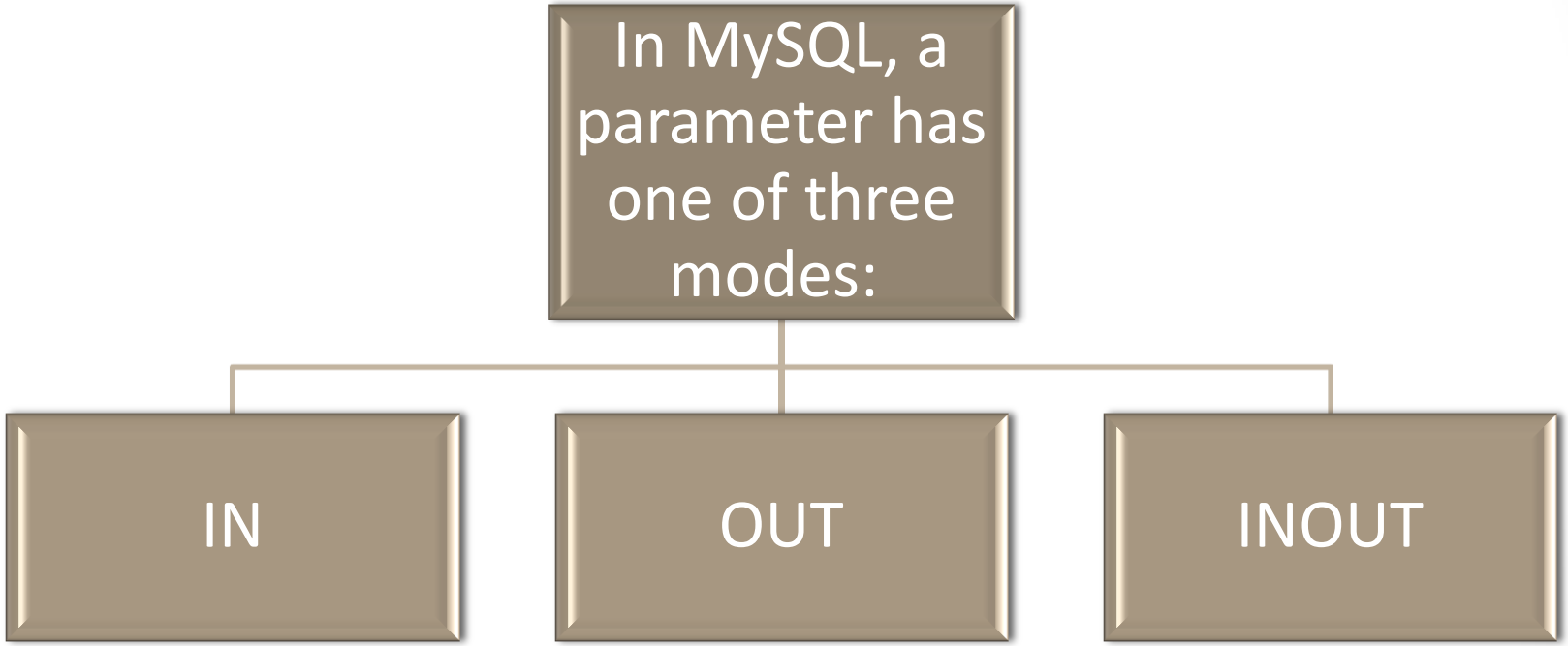
<Declare variable_name data_type;>

<Control Statements/loops>

SQL executable statements;

End

Stored Procedure- Parameters



Stored Procedure- Parameters

IN – is the default mode. When you define an IN parameter in a stored procedure, the calling program has to pass an argument to the stored procedure.

OUT – the value of an OUT parameter can be changed inside the stored procedure and its new value is passed back to the calling program

INOUT – an INOUT parameter is the combination of IN and OUT parameters. It means that the calling program may pass the argument, and the stored procedure can modify the INOUT parameter and pass the new value back to the calling program.

Without parameter Example

```
Mysql> DELIMITER  //
Mysql> CREATE PROCEDURE Allstud()
          BEGIN
          SELECT *  FROM stud;
          END
         //
Mysql> DELIMITER ;
Mysql> call Allstud();
```

The IN parameter example

```
Mysql> DELIMITER  //
Mysql> CREATE PROCEDURE Allstud(IN SName VARCHAR(25))
          BEGIN
          SELECT *  FROM stud where Name=SName;
          END
         //
Mysql> DELIMITER ;
Mysql> call Allstud('Reena');
```

The IN parameter example(more than one IN parameters)

```
Mysql> DELIMITER  //
Mysql> CREATE PROCEDURE Allstud(IN Rno1 int(3),SName
VARCHAR(25))
            BEGIN
             Update stud set Name=Sname where Rno=Rno1;
            END
          //
Mysql> DELIMITER ;
Mysql> call Allstud(1,‘Kritika’);
```

The IN parameter example(more than one IN parameters)

```
Mysql> DELIMITER  //
Mysql> CREATE PROCEDURE Allstud(IN Rno1 int(3),SName
VARCHAR(25))
          BEGIN
         insert into stud values(Rno1,Sname);
          END
        //
Mysql> DELIMITER ;
Mysql> call Allstud(2,‘Seema’);
```

The OUT parameter example

```
Mysql> DELIMITER  //
Mysql> CREATE PROCEDURE Allstud(OUT SName VARCHAR)
          BEGIN
          SELECT Name into SName FROM stud where Rno=1;
          END
         //
Mysql> DELIMITER ;
Mysql> call Allstud();
```

The IN and OUT parameter example

```
Mysql> DELIMITER  //
Mysql> CREATE PROCEDURE Allstud(IN Rno1 int,OUT SName
VARCHAR)
        BEGIN
        SELECT Name into SName FROM stud where Rno=RNo1;
        END
       //
Mysql> DELIMITER ;
Mysql> call Allstud(2,@SName);
Mysql> select @Sname'
```

The IN and OUT parameter example

```
Mysql> DELIMITER  //
Mysql> CREATE PROCEDURE Allstud(IN SNAME varchar,OUT
RNno int)
            BEGIN
            SELECT Rno into Rno1 FROM stud where Name=SName;
            END
          //
Mysql> DELIMITER ;
Mysql> call Allstud('Reena',@Rno1);
Mysql> select @Rno1;
```

The INOUT parameter example

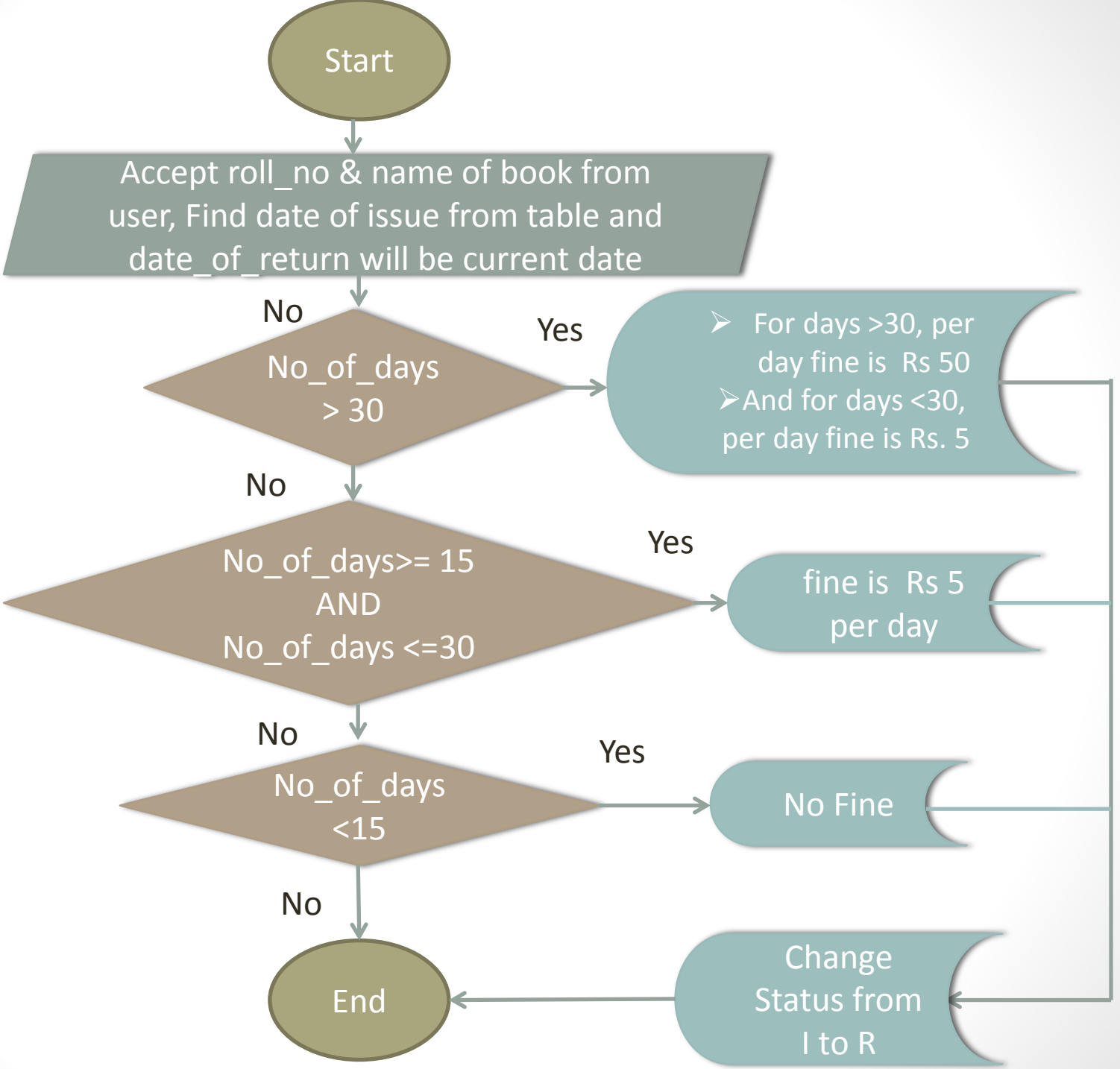
```
Mysql> DELIMITER $
Mysql> CREATE PROCEDURE set_counter(INOUT count INT(4))
        BEGIN
        SET count = count + 10;
        END
        $
Mysql> DELIMITER ;
Mysql> SET @counter = 1;
Mysql> CALL set_counter(@counter);
Mysql> SELECT @counter;
```

The Control Statement example

```
Mysql> DELIMITER  //
Mysql> CREATE PROCEDURE Allstud(IN Rno1 int(3))
            BEGIN
            Declare marks int;
            if (Rno1>5) then
            Set marks=70;
            Else
            Set marks=90;
           End if;
            Update stud set Mark=marks where Rno=Rno1; ;
            END
          //
Mysql> DELIMITER ;
Mysql> call Allstud(3);
Mysql> select * from stud;
```

Assignment

- Unnamed PL/SQL code block: Use of Control structure and Exception handling is mandatory. Write a PL/SQL block of code for the following requirements:-
- **Schema:**
  1. Borrower(Roll_no, Name, DateofIssue, NameofBook, Status)
  2. Fine(Roll_no,Date,Amt)
- **Accept roll_no & name of book from user**.
- Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day.
- If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.
- After submitting the book, status will change from I to R.
- If condition of fine is true, then details will be stored into fine table.


Start
Accept roll_no & name of book from user, Find date of issue from table and date_of_return will be current date
No
No_of_days > 30
Yes
For days >30, per day fine is Rs 50
And for days <30, per day fine is Rs. 5
No
No_of_days>= 15 AND No_of_days <=30
Yes
fine is Rs 5 per day
No
No_of_days <15
Yes
No Fine
No
End
Change Status from I to R

Assignment Required Functions- CURDATE()

- The CURDATE() function returns the current date
- .
- **Note:** This function returns the current date as a "YYYY-MM-DD" format

- **Example**
- Return current date:

- SELECT CURDATE();

Assignment Required Functions- DATEDIFF()

- The DATEDIFF() function returns the difference in days between two date values.

- **Syntax**
- DATEDIFF(*date1*, *date2*)

- **Example**
- Return the difference in days between two date values:
- SELECT DATEDIFF("2017-06-25", "2017-06-15");

Assignment Tables-before procure run

Borrower Table

Roll_no	Name	DateofIssue	NameofBook	Status
1	Amita	2017-06-25	Java	I
2	Sonakshi	2017-07-10	Networking	I
3	Nira	2017-05-22	MySQL	I
4	Jagdish	2017-06-10	DBMS	I
5	Jayashree	2017-07-05	MySQL	I
6	Kiran	2017-06-30	Java	I

Fine Table

Roll_no	Date	Amt

Assignment Tables-After procure run

Borrower Table

Rno	Name	DateofIssue	NameofBook	Status
1	Amita	2017-06-25	Java	I
2	Sonakshi	2017-07-10	Networking	I
3	Nira	2017-05-22	MySQL	I
4	Jagdish	2017-06-10	DBMS	R
5	Jayashree	2017-07-05	MySQL	I
6	Kiran	2017-06-30	Java	I

Fine Table

Roll_no	Date	Amt
4	2017-06-30	100

Stored Procedure Example in MySQL

To find difference in current date and issue date

```
Mysql> delimiter $
Mysql> Create procedure P1(In  rno1 int(3),name1 varchar(30))
       begin
          Declare i_date date;
          Declare diff int;
          select DateofIssue into i_date from stud where Rno=rno1
          and  NameofBook=name1;
          SELECT DATEDIFF(CURDATE(), i_date) into diff;
       End;
       $
Mysql>delimiter ;
Mysql> call p1(1,'DBMS');
```

Stored Procedure Example in MySQL

- To change status from I to R

```
Mysql> delimiter $
Mysql> Create procedure P2(In  rno1 int(3),name1 varchar(30))
       begin
          Declare i_date date;
          Declare diff int;
          select DateofIssue into i_date from stud where Rno=rno1
          and  NameofBook=name1;
          SELECT DATEDIFF(CURDATE(), i_date) into diff;
          If  diff>15 then
                    Update stud
                     set status='R'
                    where Rno=rno1 and NameofBook=name1;
          End if;
       End;
       $
Mysql>delimiter ;
Mysql> call p2(1,'DBMS');
```

Stored Procedure Example in MySQL- To set fine amount between 15 and 30 days with status change

- Create procedure P3(In rno1 int(3),name1 varchar(30))
  begin
  Declare i_date date;
  Declare diff int;
  Declare fine_amt int;
  select Idate into i_date from stud where rno=rno1
  and name=name1;
  SELECT DATEDIFF(CURDATE(), i_date) into diff;
  If (diff>=15 and diff<=30)then
  SET fine_amt=diff*5;
  insert into fine values(rno1,CURDATE(),fine_amt);
  Update stud set status='R' where rno=rno1 and name=name1;
  End if;
  End;
  $
  call p3(1,'DBMS');

Stored Procedure Example in MySQL-To set fine amount between 15 and 30 days & > 30 days with status change

- Create procedure P3(In rno1 int(3),name1 varchar(30))
  begin
  Declare i_date date;
  Declare diff int;
  Declare fine_amt int;
  select Idate into i_date from stud where rno=rno1
  and name=name1;
  SELECT DATEDIFF(CURDATE(), i_date) into diff;
  If (diff>=15 and diff<=30)then
  SET fine_amt=diff*5;
  insert into fine values(rno1,CURDATE(),fine_amt);
  elseif (diff>30) then
  SET fine_amt=diff*50;
  insert into fine values(rno1,CURDATE(),fine_amt);
  End if;
  Update stud set status='R' where rno=rno1 and name=name1;
  End;

Exception handling

- **Declaring a handler**
- To declare a handler, you use the statement as follows:
  - DECLARE action HANDLER FOR condition_value statement;

- If a condition whose value matches the condition_value , MySQL will execute the statement and continue or exit the current code block based on the action .

- The action accepts one of the following values:
  1. CONTINUE : the execution of the enclosing code block (BEGIN … END) continues.
  2. EXIT : the execution of the enclosing code block, where the handler is declared, terminates.

Exception handling-simple example

```
Mysql>delimiter //
Mysql>Create procedure Eh()
begin
DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'Table not found';
SELECT * FROM abc;
end;
//
Mysql>delimiter ;
Mysql>Call Eh();
```

Exception handling

- Create procedure a2(In rno1 int(3),name1 varchar(30))
  begin
  Declare i_date date;
  Declare diff int;
  Declare fine_amt int;
  DECLARE EXIT HANDLER FOR SQLEXCEPTION SELECT 'Table not found';
  select ldate into i_date from **stud1** where rno=rno1 and name=name1;
  SELECT DATEDIFF(CURDATE(), i_date) into diff;
  If (diff>=15 and diff<=30)then
  SET fine_amt=diff*5;
  insert into fine values(rno1,CURDATE(),fine_amt);
  elseif (diff>30) then
  SET fine_amt=diff*50;
  insert into fine values(rno1,CURDATE(),fine_amt);
  End if;
  Update stud set status='R' where rno=rno1 and name=name1;
  End;

References

- https://dev.mysql.com/doc/refman/5.7/en/flow-control-statements.html
- http://www.guru99.com/blocks-pl-sql.html
- http://www.mysqltutorial.org/
- https://forums.mysql.com/read.php?98,358569
- http://www.mysqltutorial.org/mysql-error-handling-in-stored-procedures/