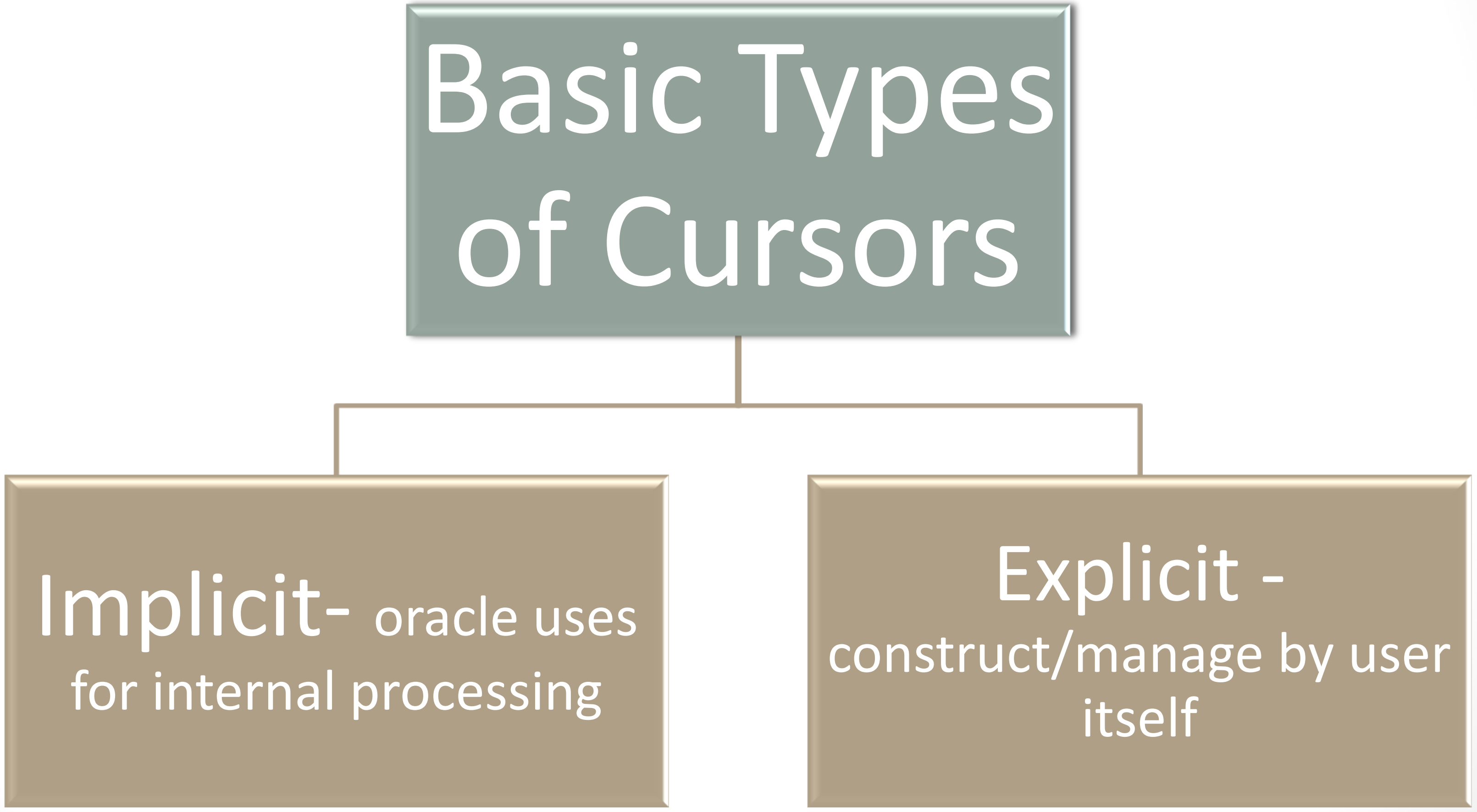
MySQL Cursor

By Prof. B.A.Khivsara

Cursor

- A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.
- To handle a result set inside a stored procedure, you use a cursor. A cursor allows you to iterate a set of rows returned by a query and process each row accordingly.

Types of Cursor in Oracle


Basic Types of Cursors
Implicit- oracle uses for internal processing
Explicit - construct/manage by user itself

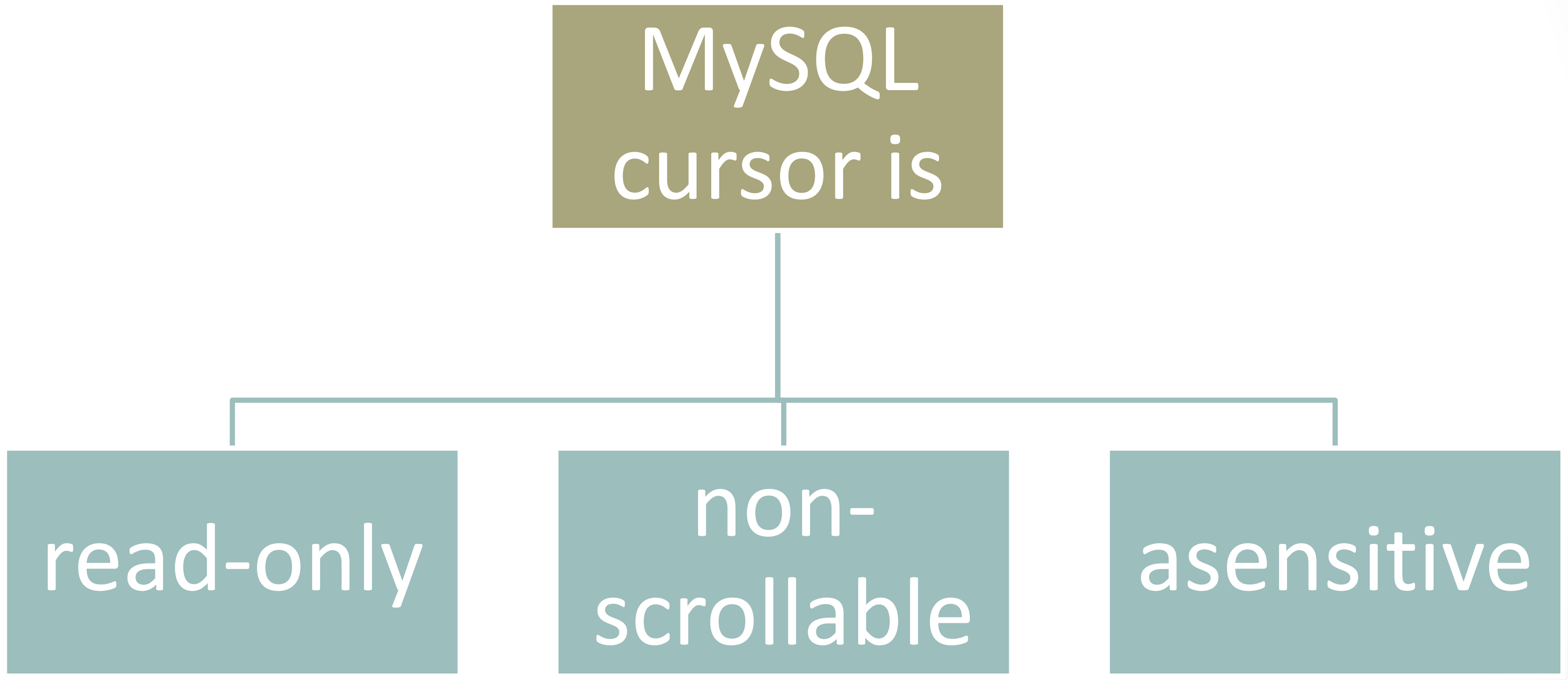
Implicit Cursors

- Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement.
- Programmers cannot control the implicit cursors and the information in it.
- Implicit Cursor is associated with following DML Statements
  - **INSERT,**
  - **UPDATE and**
  - **DELETE**
- In PL/SQL, you can refer to the most recent implicit cursor as the **SQL cursor**, which always has attributes such as
  - **%FOUND,**
  - **%ISOPEN,**
  - **%NOTFOUND**, and
  - **%ROWCOUNT**.

Explicit Cursor

- Explicit cursors are programmer-defined cursors for gaining more control over the **context area**.
- An explicit cursor 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.
- Working with an explicit cursor includes the following steps –
  - ***Declaring the cursor for initializing the memory***
  - ***Opening the cursor for allocating the memory***
  - ***Fetching the cursor for retrieving the data***
  - ***Closing the cursor to release the allocated memory***

Types of Cursor in MySQL



Types of Cursor in MySQL

- **Read only**: you cannot update data in the underlying table through the cursor.
- **Non-scrollable**: you can only fetch rows in the order determined by the SELECT statement. You cannot fetch rows in the reversed order. In addition, you cannot skip rows or jump to a specific row in the result set.
- **Asensitive**: there are two kinds of cursors: asensitive cursor and insensitive cursor. An asensitive cursor points to the actual data, whereas an insensitive cursor uses a temporary copy of the data. An asensitive cursor performs faster than an insensitive cursor because it does not have to make a temporary copy of data. However, any change that made to the data from other connections will affect the data that is being used by an asensitive cursor, therefore, it is safer if you don't update the data that is being used by an asensitive cursor. MySQL cursor is asensitive.

Step for Using Cursor

- Declare cursor
- Open cursor
- Loop
- Fetch data from cursor
- Exit loop
- Close cursor

DECLARE statement

- Syntax

DECLARE cursor_name CURSOR FOR SELECT_statement;

- Explanation
  - The cursor declaration must be after any variable declaration.
  - A cursor must always be associated with a SELECT statement.

OPEN statement

- Syntax

  OPEN cursor_name;

- Explanation
  - The OPEN statement initializes the result set for the cursor, therefore, you must call the OPEN statement before fetching rows from the result set.

LOOP statement

- Syntax

```
[label_name :]  LOOP
    statement_list
END LOOP  [label_name]
```

- Explanation
  - The LOOP loop depends on the careful placement of the LEAVE statement to terminate iteration.

FETCH statement

- Syntax

  FETCH cursor_name INTO variables list;

- Explanation
  - Then, you use the FETCH statement to retrieve the next row pointed by the cursor and move the cursor to the next row in the result set.

CLOSE statement

- Syntax

CLOSE cursor_name;

- Explanation
  - Finally, you call the CLOSE statement to deactivate the cursor and release the memory associated with it

NOT FOUND handler

When working with MySQL cursor, you must also declare a NOT FOUND handler to handle the situation when the cursor could not find any row.

Because each time you call the FETCH statement, the cursor attempts to read the next row in the result set.

When the cursor reaches the end of the result set, it will not be able to get the data, and a condition is raised.

The handler is used to handle this condition.

NOT FOUND handler

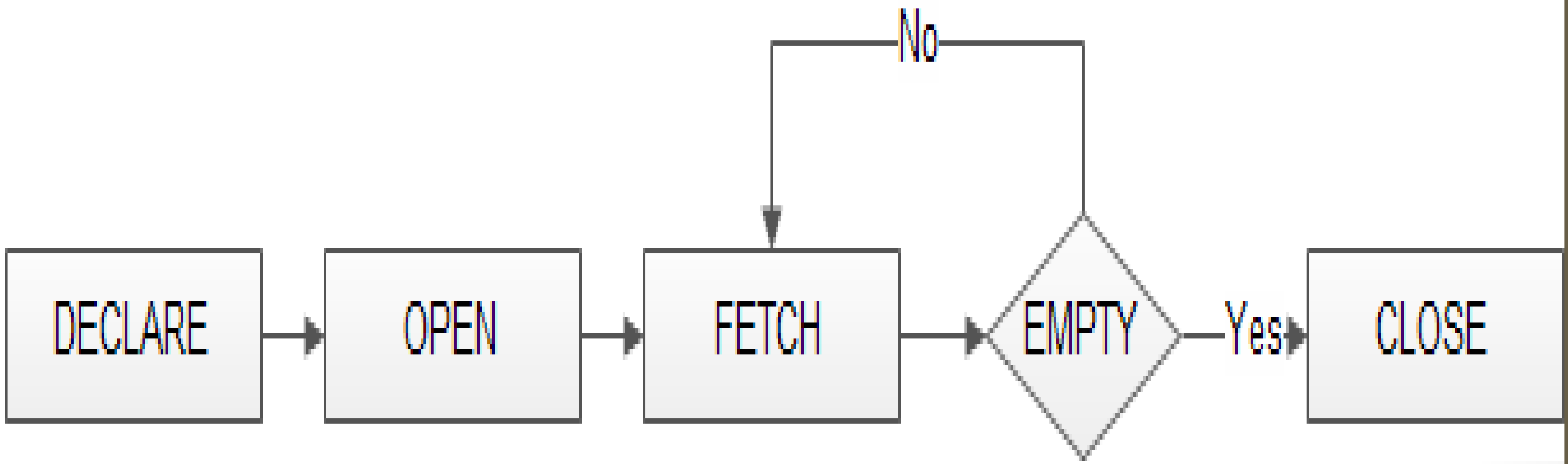
- Syntax

```
DECLARE CONTINUE HANDLER
FOR NOT FOUND
SET exit_loop = TRUE
```

- Explanation

Where finished is a variable to indicate that the cursor has reached the end of the result set. Notice that the handler declaration must appear after variable and cursor declaration inside the stored procedures.

MySQL cursor working



Cursor Example 1

```
CREATE PROCEDURE cursor_proc2()
 BEGIN
   DECLARE id   VARCHAR(3);
   DECLARE name1 VARCHAR(20);
 -- this flag will be set to true when cursor reaches end of table
   DECLARE exit_loop BOOLEAN;
   -- Declare the cursor
   DECLARE c1 CURSOR FOR   SELECT rno, name FROM stud;
   -- set exit_loop flag to true if there are no more rows
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET exit_loop = TRUE;
   -- open the cursor
 OPEN c1;
 -- start looping
   L1: LOOP
  -- read the id and name from next row into the variables
       FETCH  c1 INTO id, name1;
      select id,name1;
    -- check if the exit_loop flag has been set by mysql, close the cursor and exit the loop if it
has.
 IF exit_loop THEN
        CLOSE c1;
        LEAVE L1;
    END IF;
   END LOOP L1;
 END
```

Cursor Example 2

```
CREATE PROCEDURE sal_cur2()
 BEGIN
   DECLARE eno1 int(3);
   DECLARE sal1 int(20);

   DECLARE exit_loop BOOLEAN;
   DECLARE c1 CURSOR FOR SELECT eno, sal FROM emp;
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET exit_loop = TRUE;

   OPEN c1;

    emp_loop: LOOP
      FETCH  c1 INTO eno1,sal1;
     Select eno1,sal1;

     IF exit_loop THEN
        CLOSE c1;
        LEAVE emp_loop;
     END IF;

   END LOOP emp_loop;
 END
```

Cursor Example 3

```
CREATE PROCEDURE sal_cur2()
 BEGIN
   DECLARE eno1 int(3);
   DECLARE sal1 int(20);

   DECLARE exit_loop BOOLEAN;
   DECLARE c1 CURSOR FOR SELECT eno, sal FROM emp;
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET exit_loop = TRUE;

   OPEN c1
   emp_loop: LOOP
    FETCH  c1 INTO eno1,sal1;

     If sal1>4000 then
        Update emp set sal=sal1+5000 where eno=eno1;
     else
        Update emp set sal=sal1+1000 where eno=eno1;
    End If;

     IF exit_loop THEN
         CLOSE c1;
         LEAVE emp_loop;
     END IF;
   END LOOP emp_loop;
 END
```

Cursor Example 4

```
CREATE PROCEDURE sal_cur2()
 BEGIN
   DECLARE eno1 int(3);
   DECLARE sal1 int(20);

   DECLARE exit_loop BOOLEAN;
   DECLARE c1 CURSOR FOR SELECT eno, sal FROM emp;
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET exit_loop = TRUE;

   OPEN c1;
   emp_loop: LOOP
       FETCH  c1 INTO eno1,sal1;
            insert into emp1 values(eno1,sal1);

         IF exit_loop THEN
        CLOSE c1;
        LEAVE emp_loop;
     END IF;

   END LOOP emp_loop;
 END
```

Assignment

- Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall.
- If the data in the first table already exist in the second table then that data should be skipped.

Create O_rollcall table and add rows in the table

-
  ```
  mysql> create table O_rollcall (rno int(3) primary key, name varchar(20),addr varchar(30));
  ```

-
  ```
  mysql> insert into O_rollcall values(1,'Amit','Nashik');
  mysql> insert into O_rollcall values(2,'Shital','Pune');
  mysql> insert into O_rollcall values(3,'Ranak','Manmad');
  ```

-
  ```
  mysql> select * from O_rollcall;
  +-------+-----------+-------------+
  | rno   | name      | addr        |
  +-------+-----------+-------------+
  |   1   | Amit      | Nashik      |
  |   2   | Shital    | Pune        |
  |   3   | Ranak     | Manmad      |
  +-------+-----------+-------------+
  ```

Create N_rollcall table and add rows in N_rollcall by O_rollcall table

- mysql> create table N_rollcall(rno int(3),name varchar(20), addr varchar(30));

- mysql> select * from N_rollcall;
  Empty set (0.00 sec)

- mysql> ***insert into N_rollcal select * from O_rollcall;***

- mysql> select * from N_rollcall;

```
+-------+-----------+-------------+
| rno  | name    | addr        |
+-------+-----------+-------------+
|   1  | Amit    | Nashik      |
|   2  | Shital  | Pune        |
|   3  | Ranak   | Manmad |
+-------+-----------+-------------+
```

Delete rows from N_rollcall table

```
mysql> delete from N_rollcall;
```

```
mysql> select * from N_rollcall;
Empty set (0.00 sec)
```

Simple procedure to insert rows in N_rollcall from O_rollcall if it is not exist in N_rollcall

```
MySQL> Delimiter //
MySQL>CREATE PROCEDURE new1(IN rno1 int(3))
 BEGIN
      If not exists (select * from N_rollcall where rno=rno1) then
         insert into N_rollcall
         select * from O_rollcall
         where rno=rno1;
     End If;
END
//

MySQL>Call new1(1)//
```

Simple procedure with Parameterized **cursor** to insert rows in N_rollcall from O_rollcall if that rno is not exist in N_rollcall

```
CREATE PROCEDURE newcur(IN rno1 int(3))
BEGIN
DECLARE c1 CURSOR FOR SELECT rno FROM O_rollcall where rno=rno1;

OPEN c1;
 FETCH  c1 INTO rno1;

 If not exists(select * from N_rollcall where rno=rno1) then
        insert into N_rollcall select * from O_rollcall where rno=rno1;
    End If;

 CLOSE c1;
  END
//

MySQL>Call newcur(2)//
```

Parameterized cursor to insert rows in N_rollcall from O_rollcall if rno> given rno1 is not exist in N_rollcall

```
CREATE PROCEDURE newcur1(IN rno1 int(3))
 BEGIN
   DECLARE rno2 int(3);
   DECLARE exit_loop BOOLEAN;
   DECLARE c1 CURSOR FOR SELECT rno FROM O_rollcall where rno>rno1;
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET exit_loop = TRUE;

 OPEN c1;
 emp_loop: LOOP
  FETCH  c1 INTO rno2;
    If not exists(select * from N_rollcall  where rno=rno2) then
       insert into N_rollcall select * from O_rollcall where  rno=rno2;
   End If;
    IF exit_loop THEN
        CLOSE c1;
        LEAVE emp_loop;
    END IF;
   END LOOP emp_loop;
 END
MySQL>Call newcur1(1)//
```

cursor to insert rows in N_rollcall from O_rollcall if that rno is not exist in N_rollcall

```
CREATE PROCEDURE newcur2()
 BEGIN
   DECLARE rno1 int(3);
   DECLARE exit_loop BOOLEAN;
   DECLARE c1 CURSOR FOR SELECT rno FROM O_rollcall ;
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET exit_loop = TRUE;

 OPEN c1;
  emp_loop: LOOP
  FETCH  c1 INTO rno1;
If not exists(select * from N_rollcall  where rno=rno1) then
   insert into N_rollcall  select * from O_rollcall where  rno=rno1;
End If;

If exit_loop THEN
    CLOSE c1;
    LEAVE emp_loop;
END IF;
```

References

Websites

- *http://www.mysqltutorial.org/mysql-cursor/*
- *http://www.way2tutorial.com/plsql/plsql_cursors.php*
- *http://www.databasejournal.com/features/mysql/mysql-cursors-and-loops.html*
- *http://www.brainbell.com/tutorials/MySQL/Working_With_Cursors.htm*
- *http://www.c-sharpcorner.com/topics/cursor-in-mysql*

Video

- *https://www.youtube.com/watch?v=9z6ouWK5_l0*