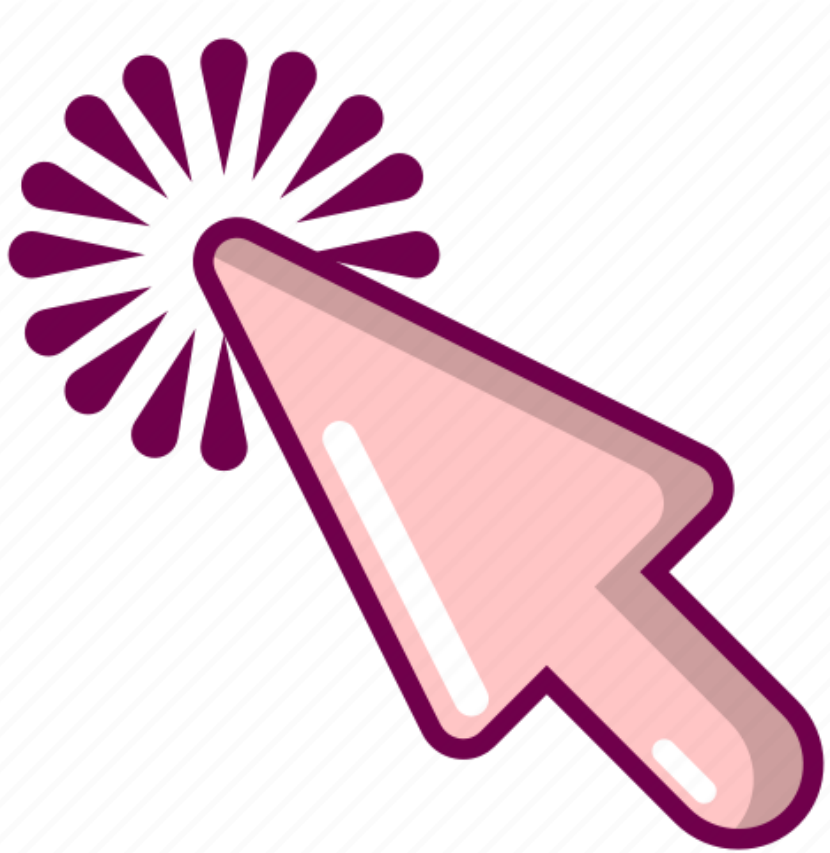
Session-13

Recap

- Features of PL/SQL
- MySQLProcedures
- MySQL Functions
- Conditional Statements
- MySQL Loops

MySQL CURSORS



Introduction to MySQL Cursor

- 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 individually.

- In simple, a cursor is a mechanism by which you can assign a name to a SELECT statement.

- MySQL cursor is:
  - **Read-only**: Not updatable
  - **Non-scrollable**: Can be traversed only in one direction and cannot skip rows
  - **Asensitive**: The server may or may not make a copy of its result table

- You can use MySQL cursors in stored procedures, stored functions, and triggers.

Working with MySQL cursor

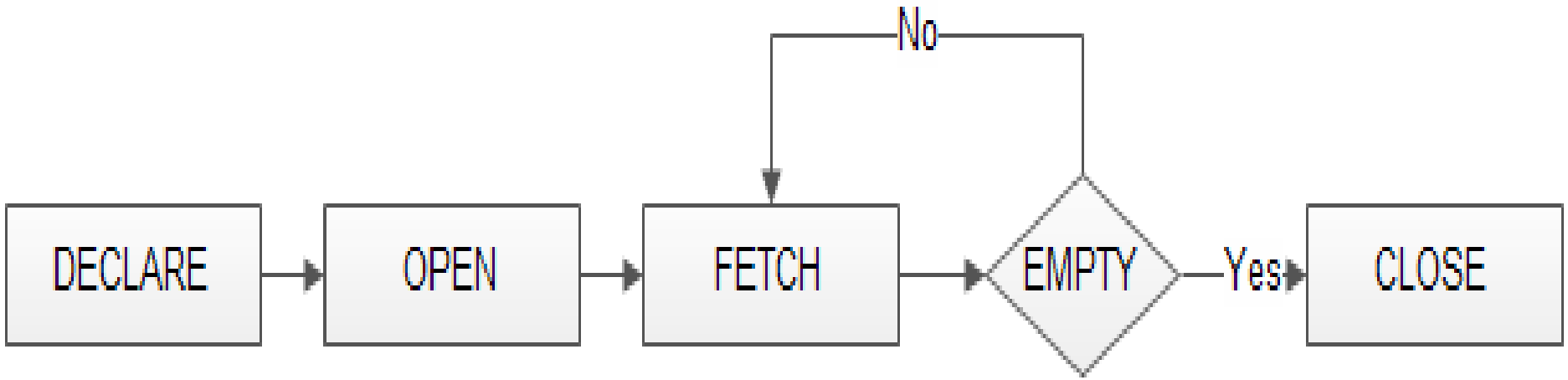
- To create a cursor, we need to perform the following steps.

  - DECLARE Cursor
  - OPEN Cursor
  - FETCH Cursor
  - CLOSE Cursor
  - Set up Handler for Cursor's NOT FOUND condition

Introduction to MySQL Cursor...

1. Declare Cursor	2. Open Cursor
DECLARE cursor_name CURSOR FOR select_statement;	OPEN cursor_name;
3. Fetch Cursor	**4. Close Cursor**
FETCH cursor_name INTO variables list;	CLOSE cursor_name;
5. NOT FOUND Handler	
DECLARE CONTINUE HANDLER FOR NOT FOUND [set_condition];	

Cursor Functioning



Example

- Cursor to get the list of Employee mail_ids

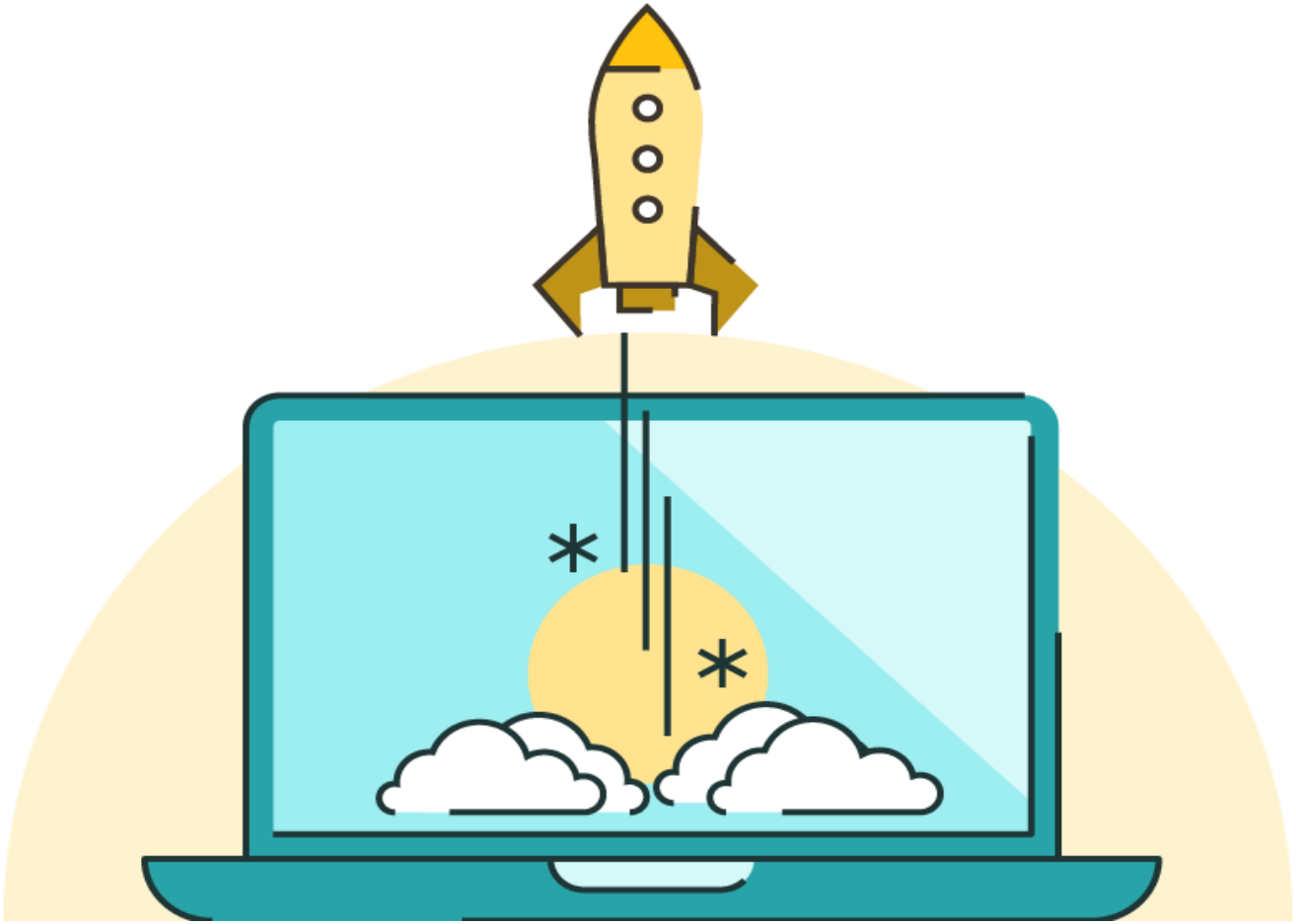
EID	ENAME	DEPT	DESG	MAID_ID
1	..	..	..	..
2	..	..	..	..

Example

- Cursor to get the list of Employee mail_ids

```
drop procedure if exists get_mailList;
delimiter @
create procedure get_mailList(inout mail_idList varchar(500))
begin
    declare email varchar(30) default "";
    declare finished int default 0;

    declare curEmail cursor for select mail_id from employee where dept = 'CSE';

    declare continue handler for not found set finished = 1;

    open curEmail;

    getmail: Loop
            fetch curEmail into email;
            if finished = 1 then
                leave getmail;
            end if;
            set mail_idList = concat(email, " ; ", mail_idList);
            end loop getmail;

    close curEmail;

end @;
delimiter ;
```

MySQL Triggers



Introduction to Triggers

- A trigger is a stored program invoked automatically in response to an event that occurs in the associated table.

- MySQL supports triggers that are invoked in response to the INSERT, UPDATE or DELETE event.

- The SQL standard defines two types of triggers:
  - row-level triggers
  - statement-level triggers.

- MySQL supports only row-level triggers. It doesn't support statement-level triggers.

Syntax for Creating & Dropping Trigger

```
CREATE
        TRIGGER trigger_name
        trigger_time  trigger_event
        ON table_name FOR EACH ROW
        BEGIN
                trigger_body
        END
```

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

```
DROP TRIGGER trigger_name;
```

Types of Triggers

- There are 6 type of triggers that can be created.
  - Before Insert
  - After Insert
  - Before Update
  - After Update
  - Before Delete
  - After Delete

Value Modifiers

- The trigger body can access the values of the column being affected by the DML statement.
- To distinguish between the value of the columns BEFORE and AFTER the DML has fired, we use the **NEW** and **OLD** modifiers.

Trigger Event	OLD	NEW
INSERT	No	Yes
UPDATE	Yes	Yes
DELETE	Yes	No

Examples

- Simple triggers on EMPLOYEE table

```
AFTER INSERT
------------

create table employee_log (empid int, empname varchar(20), actiontype varchar(10), actiondate date);

drop trigger aft_emp_insert;
delimiter $$
create trigger aft_emp_insert after insert on employee for each row
begin
     insert into employee_log values(new.eid, new.ename, 'inserted', now());
end $$
delimiter ;


insert into employee values (9, 'Suma', 'ECM', 'HOD', 'suma@gmail.com');
select * from employee;
select * from employee_log;
```

Examples

- Simple triggers on EMPLOYEE table

```
BEFORE DELETE
-------------
drop trigger bef_emp_delete;
delimiter $$
create trigger bef_emp_delete before delete on employee for each row
begin
     insert into employee_log values(old.eid, old.ename, 'deleted', now());
end $$
delimiter ;

delete from employee where eid = 8;
```

Examples

- Simple triggers on EMPLOYEE table

```
/* BEFORE UPDATE on same Table */

drop trigger bef_emp_update;
delimiter $$
create trigger bef_emp_update before update on employee for each row
begin
    set new.ename = UPPER(new.ename);
end $$
delimiter ;

update employee set dept = 'ME' where eid = 8;
```

SUMMARY

- MyQSL Procedures
- MySQL Functions
- IF-THEN-ELSE Statements
- CASE Statements
- MySQL LOOPs
- MySQL Cursors
- MySQL Triggers


THANK
YOU