Day_11

MySQL - PL - CURSORS

EMP

EMPNO	ENAME	SAL	DEPTNO
1	A	5000	1
2	B	6000	1
3	C	7000	1
4	D	9000	2
5	E	8000	2

- present in all RDBMS, some DBMS, and some of the front-end s/w
- cursor is a type of a variable
- cursor can store multiple rows
- cursor is similar to 2-D array

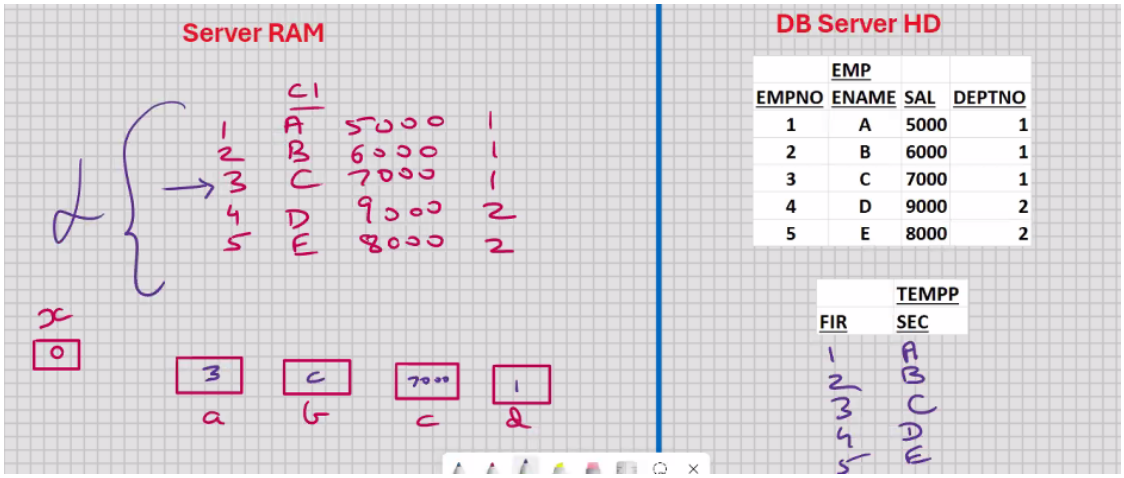
- use for storing multiple rows
- used for processing multiple rows
- used handling multiple rows

- used for storing data temporarily

```
delimiter //
create procedure abc()
begin
    declare a int;
    declare b varchar(15);
    declare c int;
    declare d int;
    declare x int default 0;
    declare c1 cursor for select * from emp;  ← cursor  declaration/Defination. At this point cursor does not contain data.
    open c1;   ←This will open the cursor, it will execute the select statement, it populate the cursor c1
    while x < 5 do
        fetch c1 into a, b, c, d;
        /*processing, set hra = c*0.4, etc*/
        insert into temp values(a, b);
        set x = x + 1;
    end while;
    close c1; ← THIS WILL CLOSE THE CURSOR C1 AND FREE THE RAM
end; //
delimiter ;
```

```
delimiter //
create procedure abc()
begin
        declare a int;
        declare b varchar(15);
        declare c int;
        declare d int;
        declare x int default 0;
        declare c1 cursor for select * from emp;
        open c1;
        while x < 5 do
                fetch c1 into a, b, c, d;
                /* processing, e.g. set hra = c*0.4, etc. */
                insert into tempp values(a, b);
                set x = x+1;
        end while;
        close c1;
end; //
delimiter ;
```

- cursor has to be declared AFTER all the variables



- cursor is based on SELECT statement
- the SELECT statement on which the cursor is based could be anything, e.g. select col1, col2, ..., coln, WHERE clause, ORDER BY clause, GROUP BY clause,etc.
- you can have cursor based on join, sub-queries, set operators, view, etc.
- you can have cursor based on computed, column, expression, functions etc.

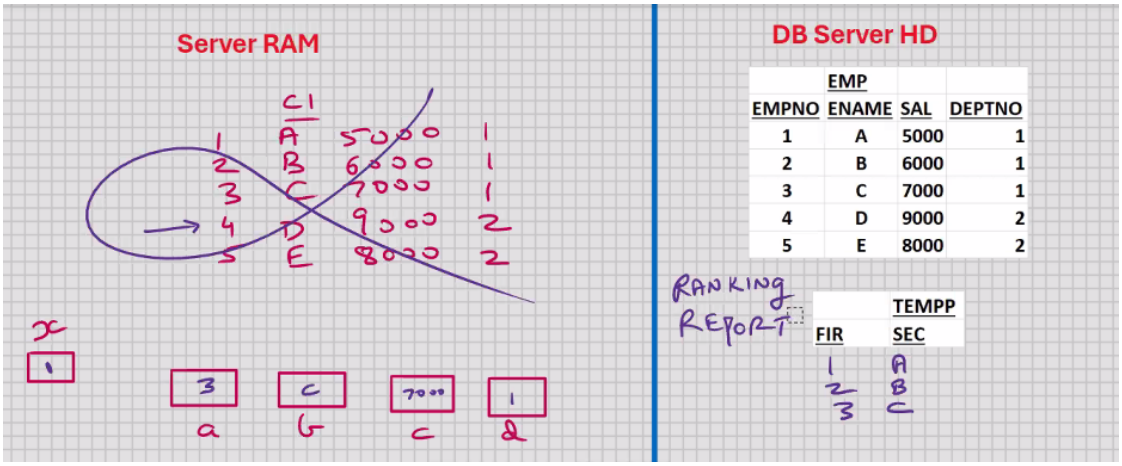
- cursor is a READ_ONLY variable
- data that is present inside the cursor, it cannot be manipulated
- you will have to fetch 1 row at a time into some intermediate variables, and do your processing with those variables
- you can only fetch sequentially(top to bottom)
- YOU CANNOT FETCH BACKWARDS IN MySQL CURSOR
- you can only fetch 1 row a time

- DO CALCULATION FOR ONLY CERTAIN ROWS

```
delimiter //
create procedure abc()
begin
    declare a int;
    declare b varchar(15);
    declare c int;
    declare d int;
    declare x int default 1;
    declare c1 cursor for select * from emp;  ← cursor  declaration/Defination. At this point cursor does not contain data.
    open c1;   ←This will open the cursor, it will execute the select statement, it populate the cursor c1
    while x < 4 do
        fetch c1 into a, b, c, d;
        /*processing, set hra = c*0.4, etc*/
        insert into temp values(a, b);
        set x = x + 1;
    end while;
    close c1; ← THIS WILL CLOSE THE CURSOR C1 AND FREE THE RAM
```

end; //

delimiter ;



delimiter //

create procedure abc()

begin

declare a int;

declare b varchar(15);

declare c int;

declare d int;

declare x int default 1;

declare c1 cursor for select * from emp; ← cursor declaration/Defination. At this point cursor does not contain data.

open c1; ←This will open the cursor, it will execute the select statement, it populate the cursor c1

while x < 11 do ← ERROR

fetch c1 into a, b, c, d;

/*processing, set hra = c*0.4, etc*/

insert into temp values(a, b);

set x = x + 1;

end while;

close c1; ← THIS WILL CLOSE THE CURSOR C1 AND FREE THE RAM

end; //

delimiter ;

- Declare a CONTINUE HANDLER for NOT FOUND event
- NOT FOUND is a cursor attribute; it returns a Boolean TRUE value if the last fetch was unsuccessful, and FALSE value if the last fetch was successful

```
begin
        declare a int;
        declare b varchar(15);
        declare c int;
        declare d int;
        declare y int default 0;
        declare c1 cursor for select * from emp;
        declare continue handler for not found set y = 1;
        open c1;
        cursor_c1_loop:loop
                fetch c1 into a, b, c, d;
                if y = 1 then
                        leave cursor_c1_loop;
                end if;
                /* processing, e.g. set hra = c*0.4, etc. */
                insert into tempp values(a, b);
        end loop cursor_c1_loop;
        close c1;
end; //
delimiter ;
```

FLEXIBLE:

```
delimiter //
create procedure abc(dd int, ss int)
begin
        ..........................;
        declare c1 cursor for select * from emp
        where deptno = dd and sal > ss;
        ..........................;
end; //
delimiter ;

call abc(1, 5000);
call abc(2, 6000);
```

- use of cursors:
  - used for storing/processing multiple rows
  - USED FOR LOCKING THE ROWS MANUALLY

mysql> select * from emp for update;

```
delimiter //
create procedure abc()
begin
        declare c1 cursor for select * from emp for update;
        open c1;
        close c1;
end; //
delimiter ;

mysql> call abc();
```

- LOCKS ARE AUTOMATICALLY RELEASED WHEN YOU ROLLBACK OR COMMIT

- You can pass parameters to procedure

Parameters are of 3 types:

IN (by default)

- Read-Only
- can pass a constant, variable, and expression
- call by value
- FASTEST in terms of processing speed

```
delimiter //
create procedure abc(in y int)
begin
    insert into tempp values(y, 'inside abc');
end; //
delimiter ;

delimiter //
create procedure pqr()
begin
    declare x in default 10;
    call abc(5);
    call abc(x);
    call abc(2*x+5);
end; //
delimiter ;

mysql> call pqr();
```

OUT

- Write Only
- can pass variables only

- call by reference
- procedure can return a value indirectly if you call by reference
- if you want to return a value and Security is a concern
- used on public network e.g. Internet
- MOST SECURE(e.g. username, password, OTP, etc.)

```
delimiter //
create procedure abc(out y int)
begin
    set y = 100;
end ; //
delimiter ;

delimiter //
create procedure pqr()
begin
    declare x in default 10;
    insert into tempp values(x, 'before abc');
    call abc(x);
    insert into tempp values(x, 'after abc');
end; //
delimiter ;

mysql> call pqr();
```

INOUT

- Read and Write
- can pass variables only

- call by reference
- procedure can return a value indirectly if you call by reference
- if you want to return a value and Security is not a concern
- used on local network e.g. Home, CDAC office
- MOST POWER and BEST FUNCTIONALITY(e.g. username, password, OTP, etc.)

```
delimiter //
create procedure abc(inout y int)
begin
    set y = y*y*y;
end ; //
delimiter ;

delimiter //
create procedure pqr()
begin
    declare x in default 10;
    insert into tempp values(x, 'before abc');
    call abc(x);
    insert into tempp values(x, 'after abc');
end; //
delimiter ;

mysql> call pqr();
```

MySQL - PL - STORED FUNCTIONS

STORED OBJECTS

- objects that are stored in the database
- e.g. CREATE ……. tables, indexes, views, stored procedures
- anything that you do with CREATE command is a stored object

---

STORED FUNCTIONS

- routine that return a value directly and compulsorily
- global functions
- stored in the database
- can be called in MySQL Command Line Client, MySQL WorkBench, JAva, MS .Net; can be called through any front-end s/w
- stored in the database in the COMPILED FORMAT
- hence the execution will be very fast
- hiding source code from end user
- within the function, all MySQL-PL statements allowed e.g. vars, cursors, IF statements, etc.
- stored procedure can call stored function
- stored function can call stored function
- one function can call another function
- function can call itself (known as Recursion)
- to make it flexible, you can pass parameters to a function
- OVERLOADING OF STORED FUNCTIONS is NOT ALLOWED, because it's a stored object; you can create 2 more functions with the same name even if the NUMBER of parameters passed is different or the DATATYPE of parameters passed is different
- IN parameters ONLY

Stored functions are of 2 Types:

- Deterministic

- Not Deterministic
- for the same input parameters, if the stored function returns the same result, it is considered deterministic, and otherwise the stored function is not deterministic
- you have to decide whether a stored function is deterministic or not
- if you declare it incorrectly, the stored function may produce an unexpected result, or the available optimization is not used which degrades the performance

mysql> call abc();

- unlike a stored procedure a stored function cannot be called by itself, because a function returns a value and that value has to be stored somewhere, and therefore the function has to equated with a variable, or it has to be a part of some expression

---

```
delimiter //
create function abc()
returns int
deterministic
begin
    return 10;
end; //
delimiter ;

delimiter //
create procedure pqr()
begin
    declare x int;
    set x = abc();
    insert into tempp values(x, 'after abc');
```

```
end; //
delimiter ;
mysql> call pqr();
drop function abc;← dropping a function
```

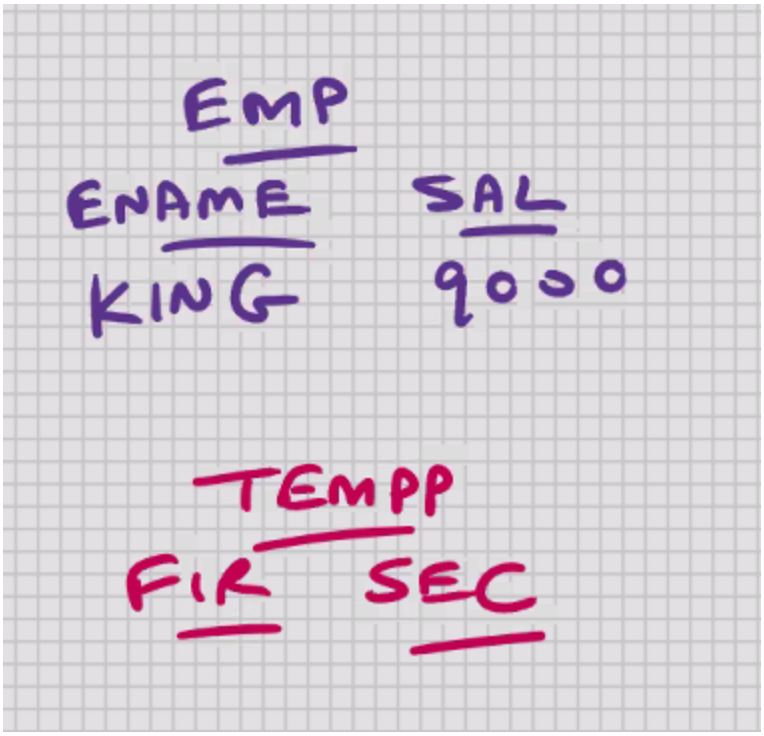
```
delimiter //
create function abc(y int)
returns int
deterministic
begin
    return y*y;
end;  //
delimiter ;

delimiter //
create procedure pqr();
begin
    declare x int;
    set x = abc(10);
    insert into tempp values(x, 'after abc');
end; //
delimiter ;
mysql> call pqr();
```

- STORED FUNCTION CAN BE CALLED IS SELECT STATEMENT
- STORED FUNCTION CAN BE CALLED IN DML COMMANDS ALSO

```
mysql> select abc(sal) from emp;
```

```
mysql> update emp set sal = abc(sal);
mysql> delete from emp where abc(sal) = 1000000;
```




```
delimiter //
create function abc(y int)
returns boolean
deterministic
begin
    if y>5000 then
        return TRUE;
    else
        return FALSE;
    end if;
end; //
delimiter ;
```

```
delimiter //
create procedure pqr()
begin
    declare x int;
    select sal into x from emp where ename = 'KING';
    if abc(x) then
        insert into tempp values(x, '> 5000');
    else
        insert into tempp values(x, '≤5000');
    end if;
end; //
delimiter ;
mysql> call pqr();
```

- function is normally used as a validation routine
- function normally returns a boolean TRUE or FALSE value, accordingly some future processing
- if function returns a Boolean value, then you can directly use the functions name as a condition for IF statement

---

```
To see which all functions are created:-

show function status;           <- shows all functions in all databases

show function status where db = 'cdacmumbai';

show function status where name like 'a%';
```

```
To view the source code of stored function:-

show create function abc;

To share the function with other users:-

root_mysql> grant execute on function cdacmumbai.abc to scott@localhost;

scott_mysql> select cdacmumbai.abc() from dual;

root_mysql> revoke execute on function cdacmumbai.abc from scott@localhost;
```