

# Database Management Systems

Sumayyea Salahuddin (Lecturer)  
Dept. of Computer Systems Eng.  
UET Peshawar

# Objectives


- Advance SQL (Part IV)
  - Prepared Statement
  - General Purpose Stored Procedure
  - Cursors
  - Resources

# Prepared Statement

- **Prepared statement** or parameterized statement is a feature used to execute the same or similar database statements repeatedly with high efficiency

- Example: 

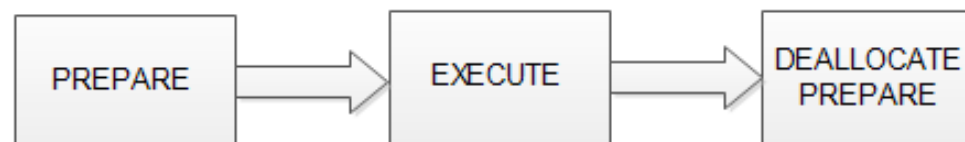
```
1 SELECT *  
2 FROM products  
3 WHERE productCode = ?
```

 This question mark is used as a placeholder.
- When MySQL executes example query with different product code values, it does not have to parse the query fully. As a result, this helps MySQL execute the query faster, especially when MySQL executes the query multiple times. Because the prepared statement uses placeholders (?), this helps avoid many variants of SQL injection hence make your application more secure.

# Prepared Statement (Cont.)

- Consists of three steps:
  1. PREPARE – Prepares statement for execution
  2. EXECUTE – Executes a prepared statement preparing by a PREPARE statement
  3. DEALLOCATE PREPARE – Releases a prepared statement

The following diagram illustrates how to use the prepared statement:



# Example

- Consider the product\_master table as given in sam\_data2:

```
mysql> select * from product_master;
```

product_no	product_desc	profit	unit	qoh	reorder_lvl	sell_price	cost_price
P00001	1.44 Floppies	5.0	piece	100	20	525	500
P03453	Monitors	6.0	piece	10	3	12000	11200
P06734	Mouse	5.0	piece	20	5	1050	500
P07865	1.22 Floppies	5.0	piece	100	20	525	500
P07868	Keyboards	2.0	piece	10	3	3150	3050
P07885	CD Drive	2.5	piece	10	3	5450	5100
P07965	540 HDD	4.0	piece	10	3	8400	8000
P07975	1.44 Drive	5.0	piece	10	3	1050	1000
P08865	1.22 Drive	5.0	piece	2	3	1050	1000

9 rows in set (0.00 sec)

We'll use this table to retrieve various products on demand from clients.

# Example (Cont.)

```
mysql> set @str = "Select product_no, product_desc from product_master where product_no = ?";  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> prepare stmt from @str;  
Query OK, 0 rows affected (0.00 sec)  
Statement prepared
```

```
mysql> set @pn = "P03453";  
Query OK, 0 rows affected (0.00 sec)
```

Client requesting  
product P03453

```
mysql> execute stmt using @pn;
```

```
-----+-----  
| product_no | product_desc |  
+-----+-----+  
| P03453     | Monitors     |  
+-----+-----+  
1 row in set (0.00 sec)
```

```
mysql> set @pn = "P00001";  
Query OK, 0 rows affected (0.00 sec)
```

Client requesting  
product P00001

```
mysql> execute stmt using @pn;  
-----+-----+  
| product_no | product_desc |  
+-----+-----+  
| P00001     | 1.44 Floppies |  
+-----+-----+  
1 row in set (0.00 sec)
```

Query is written once,  
executed twice.

```
mysql> deallocate prepare stmt;  
Query OK, 0 rows affected (0.00 sec)
```

# Example (Cont.)

- First we used the PREPARE statement to prepare a statement for execution. We used the SELECT statement to query product data from the products table based on a specified product code. We used a question mark (?) as a placeholder for the product code.
- Next, we declared a product code variable @pn and set its value to "P03453".
- Then, we used the EXECUTE statement to execute the prepared statement with the product code variable @pn.
- Next, we declared again product code variable @pn and set its value to "P00001".
- Finally, we used the DEALLOCATE PREPARE to release the prepared statement.

# General Purpose Stored Procedure

DELIMITER \$\$

```
CREATE PROCEDURE update_tbl(tbl_name varchar(30), tbl_col varchar(30), tbl_val varchar(50),  
tbl_id varchar(30), tbl_id_valie varchar(50))
```

```
begin
```

```
-- STEPT 1: Set variables to store arguments
```

```
set @tn = tbl_name;
```

```
set @tc = tbl_col;
```

```
set @tv = tbl_val;
```

```
set @ti = tbl_id;
```

```
set @tid = tbl_id_valie;
```

- Optional Step.
- Used for clarity.
- Variable tbl\_col etc. can be used directly.

```
-- STEPT 2: Write query using set variables
```

```
set @q = concat('update ', @tn, ' set ', @tc, ' = ', @tv, ' where ', @ti, ' = ', @tid, '');
```

```
-- STEPT 3: Prepare statement (As query is dynamically generated, so its need to be prepared)
```

```
prepare stmt from @q;
```

```
-- STEPT 4: Execute statement prepared in Step 3
```

```
execute stmt;
```

```
-- STEPT 5: Free space by deallocating prepare statement
```

```
deallocate prepare stmt;
```

```
end
```

General Purpose  
SP uses Prepared  
Statement for  
generalization.



# Output

```
mysql> select * from department;
```

dep_ID	dep_Name
10	Computer Systems Engineering
20	Mining Engineering
25	Civil Engineering
40	Chemical Engineering
50	Mechanical Engineering
60	Electrical Engineering

```
6 rows in set (0.00 sec)
```

```
mysql> call update_tbl("department", "dep_Name", "Industrial Engineering", "dep_ID", 20);  
Query OK, 0 rows affected (0.04 sec)
```

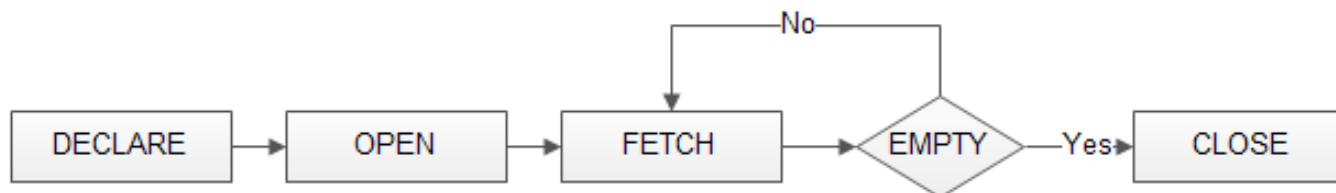
```
mysql> select * from department;
```

dep_ID	dep_Name
10	Computer Systems Engineering
20	Industrial Engineering
25	Civil Engineering
40	Chemical Engineering
50	Mechanical Engineering
60	Electrical Engineering

```
6 rows in set (0.00 sec)
```

# Cursor

- To handle a result set inside a stored procedure, cursor is used. **A cursor allows you to iterate a set of rows returned by a query and process each row accordingly.**
- It can be used in any stored program.
- There are following steps involved in cursor creation:



- Each of discussed next.

# Using Cursor

## 1) Declare Cursor:

```
1 DECLARE cursor_name CURSOR FOR SELECT_statement;
```

- The cursor declaration must be after any variable declaration.
- If you declare a cursor before variables declaration, MySQL will issue an error.
- A cursor must always be associated with a SELECT statement.

## 2) Open Cursor:

```
1 OPEN cursor_name;
```

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

## 3) Fetch Cursor:

```
1 FETCH cursor_name INTO variables list;
```

- Then, 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.

# Using Cursor (Cont.)

## 4) Check Cursor:

- Check if there is any row available before fetching it.

## 5) Close Cursor: ``` 1 CLOSE cursor_name; ```

- Use CLOSE statement to deactivate the cursor and release the memory associated with it.

## 6) Declare Continue Handler: ``` 1 DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1; ```

- Cursors need to declare a NOT FOUND handler to handle the situation when it could not find any row.
- As each time FETCH statement is called, 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.
- This handler is used to handle that condition.

# Cursor Example 1 – Build Email List of Employees

```
DELIMITER $$
```

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `build_email_list`(INOUT email_list varchar(4000))  
BEGIN
```

```
DECLARE v_finished INTEGER DEFAULT 0;  
DECLARE v_email varchar(100) DEFAULT "";
```

```
-- declare cursor for employee email  
DECLARE email_cursor CURSOR FOR  
SELECT email FROM client_master;
```

1. Declare Cursor

```
-- declare NOT FOUND handler  
DECLARE CONTINUE HANDLER  
FOR NOT FOUND SET v_finished = 1;
```

6. Declare Continue Handler

```
OPEN email_cursor;
```

```
get_email: LOOP
```

2. Open Cursor

# Cursor Example 1 – Build Email List of Employees (Cont.)

get\_email: LOOP

3. Fetch Cursor

Loop

FETCH email\_cursor INTO v\_email;

IF v\_finished = 1 THEN  
LEAVE get\_email;  
END IF;

4. Check  
Cursor Flag

-- build email list  
SET email\_list = CONCAT(v\_email,";",email\_list);  
  
END LOOP get\_email;

CLOSE email\_cursor;

5. Close Cursor

END

# Output

```
mysql> select * from client_master;
```

client_no	name	dob	city	state	pincode	balance_due	balance_flag	email
0001	Sadiq	1984-04-12	Peshawar	KPK	400054	15000.00	0	sadiq@yahoo.com
0002	Zeeshan	1980-09-12	Lahore	Punjab	780001	0.00	0	Zeeshan@yahoo.com
0003	Abbas	1985-02-02	Peshawar	KPK	400054	5000.00	0	abbas@yahoo.com
0004	Samina	1974-10-01	Peshawar	KPK	400054	0.00	0	Samina@gmail.com
0005	Ali	1966-08-14	Karachi	Sindh	100001	2000.00	0	Ali@gmail.com
0006	Semeen	1956-06-22	Peshawar	KPK	400054	0.00	0	Semeen@ymail.com

```
6 rows in set (0.00 sec)
```

```
mysql> set @email_list = "";
```

```
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> call build_email_list(@email_list);
```

```
Query OK, 0 rows affected (0.02 sec)
```

```
mysql> select @email_list;
```

@email_list
Semeen@ymail.com;Ali@gmail.com;Samina@gmail.com;abbas@yahoo.com;Zeeshan@yahoo.com;sadiq@yahoo.com;

```
1 row in set (0.00 sec)
```

# Cursor Example 2 – Fetch Table Attributes into Variable

```
DELIMITER $$
CREATE PROCEDURE cursor_proc()
BEGIN
    declare id int(5);
    declare n varchar(50);

    -- STEPT 1: As cursor will reach end of table, this flag will set to be true
    declare exit_loop boolean;

    -- STEP 2: Declare cursor
    declare department_cursor cursor for select dep_ID, dep_Name from department;

    -- STEP 3: Set exit_loop flag to true if there are no more rows
    declare continue handler for not found set exit_loop = true;

    -- STEP 4: Open cursor
    open department_cursor;
```



# Cursor Example 2 – Fetch Table Attributes into Variable (Cont.)

```
-- start loop
department_loop: loop

    -- STEP 5: Read data into variables
    fetch department_cursor into id, n;

    -- check if the exit_loop flag has been set by mysql,
    -- STEP 6: Close the cursor and exit the loop if it has.
    if exit_loop then
        close department_cursor;
        leave department_loop;
    end if;

    select id, n;

end loop department_loop;

END
```

# Advantages of Using Cursors

- No need to write business logic after fetching data from your code logic.
- Gives better flexibility for operating even on the single column of the row. Manipulation become always easy.
- Saves from the simplex join structure.
- Easy to maintain the business logic of your application at one place.

# Disadvantages of Using Cursors

- Slows down stored procedure or function performance in cause of large record set in cursor.
- Debugging of your business logic become tough.
- Hard to manage.
- Need extra care in cause of the implementation of replication using binary log.
- Need to consider locking of the database.

# Summary

- Discussed and implemented prepared statement
- Discussed and implemented general purpose store procedure based on prepared statement
- Discussed and implemented cursor in database

# Resources

- Books

- 1) MySQL Stored Procedure Programming, by Guy Harrison with Steven Feuerstein
- 2) MySQL in a Nutshell, by Russell Dyer
- 3) Web Database Applications with PHP and MySQL, by Hugh Williams and David Lane
- 4) MySQL, by Paul DuBois
- 5) High Performance MySQL, by Jeremy Zawodny and Derek Balling
- 6) MySQL Cookbook, by Paul DuBois
- 7) Pro MySQL, by Michael Krukenberg and Jay Pipes
- 8) MySQL Design and Tuning, by Robert D. Schneider
- 9) SQL in a Nutshell, by Kevin Kline, et al.
- 10) Learning SQL, by Alan Beaulieu

# Resources (Cont.)

- Internet Resources
  - 1) MySQL - Start at <http://www.mysql.com>
  - 2) MySQL Developer Zone - <http://dev.mysql.com/>
  - 3) MySQL Online Documentation - <http://dev.mysql.com/doc/>
  - 4) MySQL Forums - <http://www.planetmysql.org/>
  - 5) MySQL Stored Routines Library -  
<http://savannah.nongnu.org/projects/mysql-sr-lib/>