Q Programming

I Variables

Ways to declare variables in MySQL.

1. Using Set Statement

"Set @x = 10"

2. Using Declare Statement

Declare x INT Default 10; (Inside SP)

3. 'Select Into' Statement

Select <col_name> INTO <var_name>
From <tab_name> Where <condition>;

II Stored Procedures

- Set of SQL Statements that are stored in database server & can be executed as a single unit.
- Allows to encapsulate business logic & reuse it in multiple applications or queries.

Example :-

```
Create Procedure <proc_name>
Begin
_____
_____
End //
```

SP

System Defined — User Defined

- sp_rename : rename a database object
- sp_changeowner : change owner of database object
- sp_help : provide details on any db object
- sp_helpdb : " " of dbs defined in SQL Server.
- sp_depends : details of DB objects that depend on specific dB object.

✱ Syntax to Execute Stored Procedure

→ Call <proc_name>(); # MySQL

→ Exec/Execute <proc_name>; # SQL Server

✱ Parameters

IN (Pass I/P values) — OUT (return O/P values) — INOUT

```
Create Procedure getOfficeByCountry (IN countryName
VARCHAR(255))
BEGIN
    Select * from Offices
    where Country = countryName;
END //
Delimiter ;
Call getOfficeByCountry ('UK');
```

IN

```
Delimiter //
Create Procedure getOrderCountByStatus (IN orderStatus
VARCHAR(255), OUT total INT)
BEGIN
    Select count(orderNumber) INTO total
    From Orders
    where status = orderStatus;
END //

Call getOrderCountByStatus ('Shipped', @total);
Select @total;
```

OUT

```
Delimiter //
Create Procedure setCounter (INOUT counter IN
IN step int)
Begin
    Set counter = counter + step;
End //

Set @myCounter = 1;
Call setCounter(@myCounter, 2);
Call setCounter(@myCounter, 3);
Select @myCounter;
```

IN OUT

⭐ Local Variable Demo

```
Create Procedure getTotalOrders()
Begin
    Declare totalOrders INT Default 0;
    Select Count(*) INTO totalOrders
    From Orders;
    Select TotalOrders;
End //
Call getTotalOrders();
```

⭐ Listing Stored Procedures

```
Select * From information_schema.routines
Where routine_type = 'PROCEDURE'
AND routine_schema = '<db_name>';
```

(III) Stored Functions

- Contains sequence of SQL commands & returns a value

```
Delimiter //
Create Function customerLevel ( credit Decimal(10,2))
Returns Varchar(20)
Deterministic
Begin
    Declare customerLevel Varchar(20);
    If credit > 5000 Then
        Set customerLevel = 'Platinum';
    ElseIf (credit <= 5000 and credit >= 1000) Then
        Set customerLevel = 'Gold';
    Else
        Set customerLevel = 'Silver';
    End if;

    Return customerLevel;
End //

Delimiter ;

Select customerName, creditLimit, customerLevel (
creditLimit) From Customers
Order By CustomerName;
```

Deterministic

- return same O/P given the same I/P
- can be optimized by DB engine, & their results can be cached for faster query execution

Non-Deterministic

- return a diff O/P for same I/P depending on context in which it is executed.
- eg random number generators, current date/time functions.

Stored Procedure	Functions
- Set of pre compiled SQL statements which will executes when we call SP.	- Function is reusable piece of code which will get executed when we call it.
- will create Execution Plan	- will not have Execution Plan
- return always integer data type using OUT	- return any data type
- accept I/P & O/P params	- will accept only I/P params
☆ Can call fn in stored procedure	- can't call stored procedure in function

(R) TCL Commands

- Used to manage transactions in database.
- Changes made by DML statements.

Implicit Transactions

performed by system by default. "Auto commit" transac
DDL & DML

Explicit Transactions

used by user by using TCL commands.

eg:
```
Start Transaction
Update table1 Set col1 = 'value1' Where id = 1;
SavePoint my-savepoint;
Update ___ ___ ___
___ ___ ___ ___ ___
Rollback to my-savepoint;
Update ___ ___ ___
Commit;
```

** Note: DDL Statements (Create, Alter, Drop) cannot be rolled back.
Server Shutdown, Replication errors.

(5) DCL Commands

- used to enforce security in a multi-user database environment.
- control access to the database

(1) Grant → used to give priviliges to a user or group of users for a specific database or table.

```
Grant All Privileges on dB_name.*
To 'user'@'localhost' Identified By 'password';
```

(2) Revoke → remove priviliges from user or group of users.

```
Revoke All Priviliges ON db_name.*
From 'user'@'localhost';
```

(3) Create User →

```
Create User 'user'@'localhost' Identified By 'psw';
```

Grant Select

⑦ Cursors (Used to handle result set row by row)

- Memory location for storing database tables.
- Temporary work area allotted.
- Can hold more than one row but can process only one row at a time.

Implicit Cursor
- created by default when select statement executes.

Explicit Cursor
- Created by user.
- We can fetch multiple rows from table & store in memory area.

⭐ Steps to create Explicit Cursor

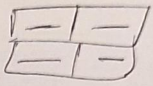
1. **Declare** → Defines name of cursor & associates with a Select statement.
   `<cursor_name> CURSOR for <Select statement>`

2. **Open** → initializes result set for the cursor
   `Open <cursor_name>`

3. **Fetch** → getting data from cursor.
   Retrieves one row at a time from the active set. Generally done inside a loop.
   `Fetch <cursor_name> INTO <variable_name>`

4. **Close** → closes the cursor. Releases all memory used by cursor.
   `Close <cursor_name>`

```
Delimiter $$
Create Procedure pro_cursor(pno int)
Begin
    declare cur int default 0;
    declare v1 numeric(11,2);
    declare v2 varchar(20);
    declare cur1 cursor for select sal, ename
    from emp where deptno = pno; Set cur=1;
    open cur1;
    GetCUR : loop
        fetch cur1 into v1, v2;
        if cur=1 then Leave GETCUR;
        endif;
        Select concat("sal = ", v1, " ename = ", v2);
        end loop;
    close cur1;
End $$
```



(U) Conditional & Looping

(I) Conditional

- control flow of execution of SQL statements

```
If cond Then
   Set statement1;
Else
   Set statement2;
End if;
```

```
If <cond> Then
   Set <statement1>;
ElseIf <cond> Then
   Set <statement2>;
Else
   Set <statement3>;
Endif;
```

Case Statement

Simple

```
Case country
   When 'USA' Then
      Set <stat1>;
   When 'CAN' Then
      Set <stat2>;
   Else
      Set <stat3>;
End Case;
```

Searched

```
Case
   When country = 'USA' Then
      Set <stat1>;
   When country = 'CAN' Then
      Set <stat2>;
   Else
      Set <stat3>;
End Case;
```

(*) Can be used in :-

a) Data Manipulation (Filter, Insert, Update, Delete)
b) SPs & SFs
c) Triggers
d) Views

Note** → IfNull (col_name, 'default value')

→ If (col > 10, 'A O/P', 'B O/P')

II) Looping

execute block of code repeatedly.

1) While Loop

```
While (x<=5) Do
    Set x = x+1;
    If (x mod 2 <>0) Then
        Set str = concat(str,x,' ');
    End if;
End While;
```

---

2) Repeat Loop ← ~~do-while~~

```
Repeat
    Set x=x+1;
    If (x mod 2<>0) Then
        Set str = concat(str, x,' ');
    End If;
Until x >5
END Repeat;
```

---

3) For Loop

```
loop_label : Loop If x>5 Then Leave loop_label;
             End If;
             Set x=x+1;
             If (x mod 2 <> 0) Then Iterate loop_label;
             Else
                 Set str= Concat(str, x, ' ');
             End if;
             End loop;
```

(V) Triggers

- special type of stored procedures that are automatically fired/executed when a DDL or DML command statement related with trigger is executed.
- Helps to preserve data integrity.

Trigger Events

Insert	Update	Delete
Before / After	Before / After	Before / After
(New)	(Old, New)	(Old)

☆ Syntax:

```
Create Trigger <Trigger Name>
Before/After <event> ON <Table Name>
For Each ROW
Begin
    Statements
End
```

☆ Examples:

① Create a Trigger to Insert Into Retired Table whenever Delete happens on Employees Table

```
Create Trigger trig_hr_del
Before Delete On Emp_ret
For each row
Begin
    Insert into retired values(old.first_name);
End $$
```

Emp

Delete

(2) Create Trigger trig_redw
Before Update On emp_rot
For each row
Begin
if new.salary < old.salary then
signal sqlstate '45000' set message_text = "_ ";
end if;
End $$

Update

(3) # Create a trigger to update balance in
Account table whenever wd(withdrawal), dposit
happens on trans table

delimiter $$
Create Trigger trig_bal
Before Insert on trans
For Each Row
Begin
if new.dep is not null then
update account set
balance = balance + new.dep where accno = new.accno;
else
update account set balance = balance - new.wd
where accno = new.accno;
endif;
End $$

Insert

(W) Window Fn & CTE

(1) Window Fn Intro

→ Select count(*) over(), ename from emp;
Returns a count column along with ename values

→ Select min(sal) over(), max(sal) over(), sum(sal) over(), avg(sal) over() from emp;
Same result provided in all rows

Running Summation

10 → 10
20 → 30
30 → 60
40 → 100

Select ename, sum(sal) over (order by sal)
From emp;

Group By

Groups the rows based on the values in one or more columns and aggregates data(or) values in each group.

Partition Over

Divides result set into partitions based on values in 1 or more columns & apply a window fn to each partition separately

(*) In over clause, we can pass 2 parameters,
Order By & Partition

(*) Ranking Fns

1) Row_Number() → row_number() over (partition by category order by amount desc)
Returns a number with each row in group.

2) rank() → gives rank & leaves gap

3) dense_rank() → gives rank & doesn't leave gaps

4) ntile() → splits the rows equally
Select ntile(5) Over(Order By deptno)

⭐ Comparing Fns

1) lead() → returns data from next row
lead(sal, 5) over(order by sal)
2) lag() → preceding nth row
lag(sal, 5) over(order by sal)
3) first_value(col) → 1st record
4) last_value(col) → last record
5) nth_value(col, n) → no. of record wants to be chosen

⭐ Frame

over(order by sal range between unbounded preceding and unbounded following)

1. Range Between 3 Preceding AND 3 Following
2. Range Between 100 Preceding AND Current Row
3. Range Between Current Row AND Unbounded Following
4. Range Between 100 Preceding AND Unbounded Following

II CTE

With CTE as (select * from emp) Select * from cte;

Recursive CTE :-

```
With Recursive Factorial(n, result) AS (
  Select 1,1
  Union ALL
  Select n+1, (n+1)*result From factorial
  Where n<10
)
Select Result from factorial where n=10;
```

(n+1)*result → previous row
Where n<10 → condition
Select Result ... where n=10; → Final value

Factorial.

n	result
1	1
2	2
3	6
4	24
10	⋮

(X) Temporary Table & MySQL Dump

- Created & exists only for duration of single database session or connection.
- Useful when you need to store & manipulate data temporarily.
- Syntax:

```
Create Temporary Table temp_table(
id INT, name VARCHAR(50), age INT);
```

Temporary Tables	Views
• physical tables that store data	• Virtual tables that do not store any data
• exist for duration of a single session or connection	• Remain in DB until explicitly dropped
• Used to store & manipulate data temporarily	• Way to access data from one or more tables in a structured & convenient way

(☆) Exporting Database

```
mysqldump -u root -p hr > exphr.sql
```

Importing Database

```
mysqldump -u root -p ramu < exphr.sql
```

(Y) Functions Miscellaneous

(1) Date_Format (Date, Format)

%D → day of month + (th/nd) eg 28th
%d → day of month eg 21
%w → day count in week
%W → day Name
%j → day count in Year
%M → full month
%b → 3 letter of month
%y → 2 digit of year

%m → month as number
%Y → 4 digit of year

(2) Extract function

- extract(year from CURDATE())
- extract(month from CURDATE())

(3) Conversion Functions

- TO_CHAR
- TO_DATE
- TO_TIMESTAMP

```
Select STR_TO_DATE("10-MAY-2022", "%d-%M-%Y");
       STR_To_DATE("10-JAN-22", "%d-%b-%y");
```

(4) General Functions

- NullIf(A, B) → If A=B Then Null Else A
- NVL(A, B) → If A IS Null Then B Else A
- NVL2(A,B,C) → If A IS Null Then C Else B
- COALESCE(A,B,C) → If A NOT NULL Then A,
  B is NULL Then B,
  (A,B) is NULL Then C

(2) SQL Optimization

(1) General Tips for Query Optimization

a) Use Column Names Instead of * in Select (27%)

b) Avoid Having Clause in Select Statements. Instead Use Where if Possible (31%)

c) Eliminate Unnecessary Distinct Columns (80%)

d) Un-nest Sub Queries (60%)

Original:

```
Select * From products P
Where P.prodid = ( Select
S.prodid From Sales S
Where S.cust id=1009 AND
S.quantity_sold=1 );
```

Improved:

```
Select p.* From Products p,
Sales S
Where p.prod_id=s.prod_id
AND s.cust_id = 10099
AND s.quantity_sold=1;
```

e) Use IN operator Instead of OR (73%)

f) Use Exists Instead of Distinct when using table joins that involve tables having one-to-many relationships (60%)

g) Use Union All in place of Union

h) Avoid Using or in join conditions instead Use Union All

i) Remove any Redundant Mathematics

② SQL Tuning

a) Using Where Clause

b) Use Table Joins Rather than Multiple Queries

c) Use Case Statement Rather than " "

d) Select
```
Count(Case when Sal<=2000 Then 1 Else Null End) Low,
Count(Case when Sal <4000 Then 1 Else Null End) Med,
Count(Case when Sal >4000 Then 1 Else Null End) high,
From Emp;
```

d) Avoid Table Scans

e) Use Partitioned Tables

③ Access Path

- Ways in which data is retrieved from the database

Table Scan ↙ Index Scan ↓ Index Lookup →

** explain format = tree select * from sal;
(Table Scan)
explain format = tree select deptno from dept;
(Index Scan)
explain format = tree select * from salgrade
where grade = 1;
(Index Lookup)

④ Indexes

- Can be created or Altered

→ Create Index dat_idx ON Emp(hiredate);

→ Alter Table Emp Add Index Ename_Idx (Ename);

- Show index from test.emp;

Q) See Indexes of tables in a database?

```
Select distinct Table_Name, Column_Name,
Index_Name From Information_Schema.Statistics
where Table_Schema = 'test'
and table_name = 'emp';
```

(5) Invisible Indexes

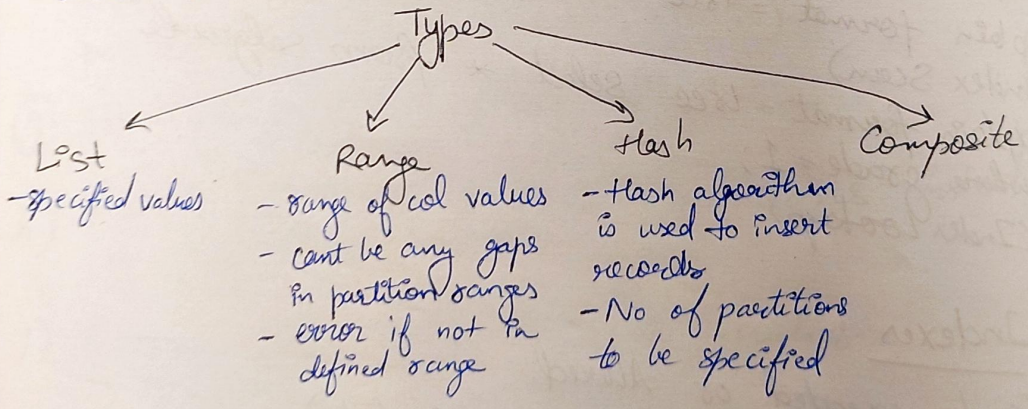
- Useful to temporarily hide an index from query optimizer.

```
Select Index_Name, IS_VISIBLE, From Information_Sch
Statistics Where Table_Schema = 'Test'
And Table_Name = 'EMP';
```

```
Alter Table Emp Alter Index Em_Idx INVISIBLE;
```

(6) Partition Tables

- Reduces amount of I/O's by dividing table into smaller subsets.
- After partitions are defined, DDL statements can access & manipulate individual partitions rather than entire tables or indexes.



Range Partition

```
Create Table empsal (empno INT, sal INT)
Partition By Range (sal)
( Partition Sal2000 Values less than (2000),
  Partition Sal4000 Values less than (4000),
  Partition Sal6000 Values less than MaxValue );
```

List Partition

```
partition by list (category) (
    Partition p_elect VALUES IN ('Electronics'),
    Partition p_cloth VALUES IN ('Clothing'),
    Partition p_books VALUES IN ('Books'),
    Partition p_other VALUES IN (Default)
);
```