✱ Views -

- considered as a virtual table
- SQL statement stored in database

✱ Creating views

```
CREATE VIEW view_name AS
  Select col1, col2 ....
  from table_name
  Where [condition];
```

Select * from view_name

→ multiple tables can be included

✱ With check option

```
Create view view_name as
  select statement
  with check optian;
```

→ used to ensure all updates & inserts satisfy the conditions in the view definition
if not satisfied returns error → error code 1369, check option failed

✱ MySQL Programming

→ Stored programs

① Procedures – Perform series of SQL statement based on certain inputs & conditions

② Functions - perform calculation based on given ilp & return the result

③ Triggers - perform additional task on execution of DML statements.

DATE: / /
PAGE NO.:

⇒ Why programming in DB
- to bundle several SQL statements together
- Centralized business logic
- Hide complexity & implementation details.

⇒ Sections of generic program
① Declaration
② Code execution
③ Exception handling

* Stored procedure
- used to execute bunch of SQL statements based on inputs (optional) & may or may not return any value
- can be nested
- Creating a procedure

```
delimiter //
create procedure procedure_name (parameters (optional))
Begin
    declarations &
    Executions
End //
Delimiter ;
```

* Why procedure?
→ increases the performance of the applications
→ reduces traffic between application & db server
→ reusable & secure

✱ Procedure Parameters

① IN parameters
- used to define parameters that expect the input from calling program
- default parameter type
- e.g.

```
Delimiter $$
create procedure get_student (IN var1 INT)
Begin
   logic
End
delimiter ;
CALL get_student(4);
```

② OUT parameters
- used to return the result to calling program.
-

```
delimiter $$
create procedure max_marks (OUT highest INT)
begin
   select max(marks) into highest from students
end $$
delimiter ;

call max_marks (@M);
select @m;
```

③ ✱ INOUT parameters -
- combination of IN & OUT parameters
- calling program may pass the argument & stored

procedure can modify it & pass the new value back to the calling program.

```
- delimeter $$
create procedure display_marks (INOUT var1 INT)
begin
    select marks into var1 from student where
    stud_id = var1;
end $$
delimiter ;

set @M = '3';
call display_marks (@M);
select @M;
```

* Flow control statements

① loop

```
[begin_label:] LOOP
        statement_list
    END LOOP [end_label].
```

- implements simple loop constructs
- enables repeated execution of statement_list
- statements are repeated until loop is terminated.

② While

```
[begin_label:] while search_condition DO
        statement_list
    END WHILE [end_label].
```

statement list within a while statement is repeated as long as the search_condition expression is true.

③ Repeat

```
[begin_label :] REPEAT
        statement_list
    UNTIL search_condition
    END REPEAT [end_label]
```

The statement list within a REPEAT statement is repeated until the search_condition expression is true. repeat enters the loop atleast once.

☆ Conditional statements.

① IF & IF-ELSE-THEN

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list]...
    [ELSE statement_list]
END IF
```

If a given search_condition evaluates to true, the corresponding THEN or ELSEIF clause statement_list executes. If no search_condition matches, the ELSE clause's statement_list executes.

Each statement_list consists of one or more SQL statements, an empty statement_list is not permitted.

⑥ CASE statement

```
CASE case_value
  WHEN when_value THEN statement_list
 [WHEN when_value THEN statement_list]...
 [ELSE statement_list]
END CASE
```

OR

```
CASE
  WHEN search_condition THEN statement_list
 [WHEN search_condition THEN statement_list]
 [ELSE statement_list]
END CASE.
```

* Loop constructs -

① ITERATE

iterate label

ITERATE can appear only within loop, repeat & while statements. ITERATE means 'start the loop again'

② LABEL

leave label.

Used to exit the flow control construct that has given label. If the label is for the outermost stored program block, leave exists the program

LEAVE can be used with BEGIN... END or loop constructs (loop, repeat, while)

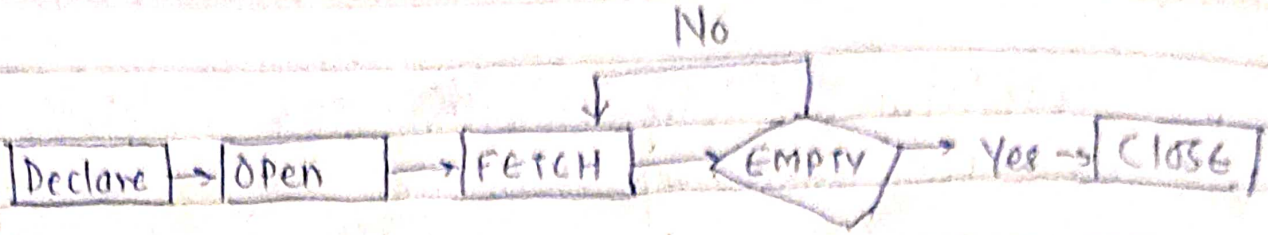
* CURSORS
- used to handle result set inside a stored procedure
- allows ~~cursor~~ user to iterate a set of rows returned by a query & process each row individually.

⇒ Types

① Read-only : ~~yo~~ cannot be used to update the data in underlying table through the cursor

② Non-scrollable - you can only fetch rows in the order determined by the select statement.
→ rows cannot be fetched in reverse order
→ rows cannot be skipped or cannot jump to a specific row in result set.

③ Asensitive & insensitive cursor :-
- An asensitive cursor ~~is insen~~ points to actual data
- Insensitive cursor uses a temporary copy of the data.
- An asensitive cursor performs faster than an insensitive cursor.
- Any change made to the data from other connections will ~~aft~~ affect the data that is being used an asensitive cursor.
- MySQL cursor is a sensitive.



→ declaring a cursor
→ be must be after any variable declaration
Declare cursor_name cursor for select_statement;

→ Opening a cursor
OPEN cursor_name;
initializes result set for the cursor

→ Fetch
used to ~~fetch~~ retrieve the next row pointed by the cursor & move the cursor to the next row in the result set
Fetch cursor_name INTO variables list

→ Close
deactivate the cursor & release the memory
CLOSE cursor_name;

⇒ When cursor reaches end of result set, it will not be able to get the data & a condition will be raised. Handler used to handle this condition.

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET
finished = 1;
```

DATE: / /
PAGE NO.:

★ Triggers

- named database object
- A Trigger is defined to activate when a statement inserts, updates or deletes rows in the associated table.
- speciat type of stored procedure that is invoked automatically in response to an event
- cannot be directly called directly like stored procedure.
- is automatically called
- Two types
  1. Row-level trigger
  2. Statement level trigger

* MySQL doesn't supports statement-level trigger. supports only row level triggers only

Why triggers -

- helps in enforcing business logic
- validate data before insertion & updation (selective)
- increases the performance
- reduces the client side code
- easy to maintain

limitations -

- difficult to troubleshoot from client side
- cannot use NOT NULL, UNIQUE, CHECK & FOREIGN KEY constraints for simple validations.

DATE: / /
PAGE NO.:

★ Creating a trigger

```
Create trigges trigger_name
    (AFTER | BEFORE) (INSERT | UPDATE | DELETE)
    ON table_name FOR EACH ROW
    BEGIN
    - variable declarations
    - trigger code
    END;
```

⇒ two keywords
OLD & NEW

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

★ Functions –

```
Delimiter $$
create function function_name (param 1, param2...)
Returns datatype
(NOT) DETERMINISTIC
BEGIN
        statements
END $$

Delimiter ;
```

✱	Simple View	Complex View
→ No. of tables	1 one	more than one
→ Use of group functions	No	Yes
→ DML operations	can be performed directly possible	not always cannot be applied directly
→ INSERT, UPDATE, DELETE operations		
→ NOT NULL columns	are not included	can be included
→ Group by, distinct	does not contains	contains
→ Association application	not needed	needed because of multiple tables)

✱	Procedure	Function
→ Returns	no, one or multiple values	single value
→ supported parameters	in, out, inout	only in
→ Can call	Function, Triggers	nothing
→ Exception handling	Yes	No
→ Used in select statement	Yes	No
→ Temporary tables	can be use	cannot be used
→ Used	For performing DL	for computations
→ Join clause	cannot be used in	can be used in join clause

★ Error handling & Exceptions

Declare {continue | exit} handler
for {SQLSTATE} sqlstate_code | MySQL error
code | condition_name}
handler action

① Handler type -
→ Exit : When an exit handler fires, the currently executing block is terminated
→ Continue : Execution continues with the statement following the one that caused the error to occur

② Handler condition (SQLSTATE, MySQL error code, named condition)
→ MySQL error code : Error number which is associated with each error
→ SQLSTATE : An ANSI-standard SQLSTATE code. will have same value regardless of the underlying database
→ Condition_name → Used to trap exception with user defined or inbuilt names.
e.g. SQLEXCEPTION, SQLWARNING & NOT FOUND

③ Handler action
- SET l_error = 1;

★ naming exceptions

Declare condition_name condition for $ SQLSTATE sqlstate_code | MySQL_error_code ;

e.g. Declare duplicate record CONDITION for SQLSTATE '23000' ;

↓ OR

1062

★

We may have implemented generic exception handling using following code

```
DECLARE continue HANDLER FOR sqlexception;
```

However to know what was the Mysql Error Code or Error Message we need extra code effort which is as following

```
delimiter #
DROP PROCEDURE p_diag_demo#
CREATE PROCEDURE p_diag_demo(p_deptno INT,p_dname
VARCHAR(20),p_loc VARCHAR(20))
BEGIN
        DECLARE err_code INT;
        DECLARE err_msg VARCHAR(2000);
        DECLARE raised_error INT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
        BEGIN
          -- Exception handler block
          GET DIAGNOSTICS CONDITION 1 err_code =
MYSQL_ERRNO, err_msg = MESSAGE_TEXT;

          SELECT raised_error,err_code,err_msg;
        END;

        INSERT INTO dept(deptno,dname,loc)
        VALUES(p_deptno,p_dname,p_loc);

END#

call p_diag_demo(30,'SALES','PUNE');
```

In this program the statement "GET DIAGNOSTICS" will try to interact with the memory area which is maintaining the
information regarding raised exception. In our code we are using this statement to fetch the information like

```
-- Exception category (raised_error = NUMBER)
-- MySQL Error Code (raised_error err_code =
MYSQL_ERRNO)
-- MySQL Error Message (err_msg = MESSAGE_TEXT)
```

Information which we can retrieve are as follows

```
NUMBER : Return Condition Area number
RETURNED_SQLSTATE : Return ANSI SQLSTATE code
which is related to cought exception
MESSAGE_TEXT      : Return Error Message which is
related to cought exception
MYSQL_ERRNO       : Return MySQL Error code which
is related to cought exception
```

Raising our own exception

```
SIGNAL SQLSTATE sqlstate_code|condition_name [SET
MESSAGE_TEXT=string_or_variable];
```