A.Sri Sailesh Reddy ==>MYSQL(SAKILA.DB)= <https://dev.mysql.com/doc/sakila/en/>

**1.view:**

* View is a virtual table based on the result-set of an SQL statement.
* view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

Advantages of View:

To restrict data access

To make complex queries easy

To provide data independence

To present different views of the same data.

* Create view :

CREATE VIEW view\_name AS

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

* Replace view:

CREATE OR REPLACE VIEW view\_name AS

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

* Drop view:

DROP VIEW view\_name;

**2.Stored procedures**:

* Stored procedure is a prepared SQL code that you can save so the code can be reused over and over again.
* So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.
* Create procedure:

CREATE PROCEDURE `name` ()

BEGIN

Query….

END

**3.Stored Function:**

* The function parameter may contain only the IN parameter but can't allow specifying this parameter, while the procedure can allow IN, OUT, INOUT parameters.
* The stored function can return only a single value defined in the function header.
* Create function:

CREATE FUNCTION `new\_function` ()

RETURNS INTEGER

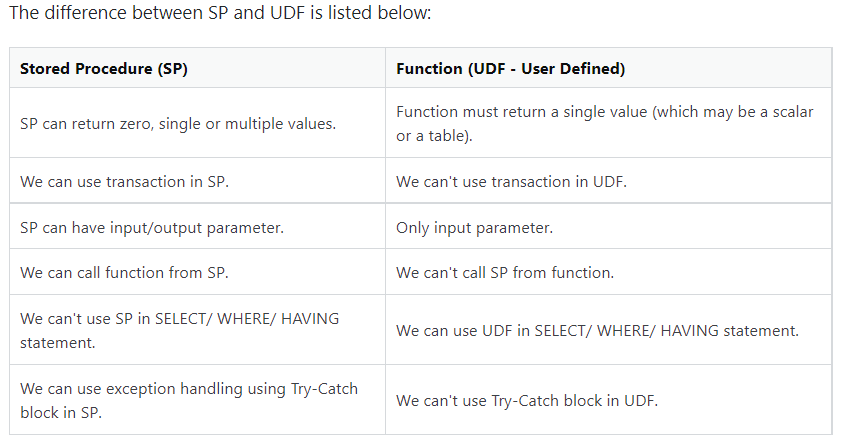
BEGIN

Query….

RETURN 1;

END

* **Difference between view and sp:**
* A view represents a **virtual** table. You can join multiple tables in a view and use the view to present the data as if the data were coming from a single table.
* A stored procedure uses parameters to do a function... whether it is updating and inserting data, or returning single values or data sets.
* **Difference between sp and udf:**



**4.Triggers:**

* Triggers are stored programs, which are automatically executed or fired when some events occur.
* Triggers are, in fact, written to be executed in response to any of the following events.
* A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
* A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
* A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).
* Create trigger:

CREATE TRIGGER trigger\_name

     (AFTER | BEFORE) (INSERT | UPDATE | DELETE)

          ON table\_name FOR EACH ROW

          BEGIN

         --variable declarations

         --trigger code

         END;

* SHOW TRIGGERS;

* Drop trigger:

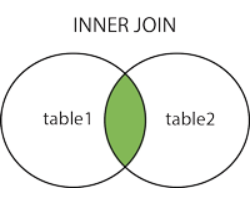
DROP TRIGGER scheme.trigger\_name;

DROP TRIGGER IF EXISTS employeedb.before\_update\_salaries;

* **The BEFORE INSERT trigger syntax parameter can be explained as below:**
* First, we will specify the name of the trigger that we want to create. It should be unique within the schema.
* Second, we will specify the trigger action time, which should be BEFORE INSERT. This trigger will be invoked before each row modifications occur on the table.
* Third, we will specify the name of a table to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.
* Finally, we will specify the statement for execution when the trigger is activated.
* **The AFTER INSERT trigger syntax parameter can be explained as below:**
* First, we will specify the name of the trigger that we want to create. It should be unique within the schema.
* Second, we will specify the trigger action time, which should be AFTER INSERT clause to invoke the trigger.
* Third, we will specify the name of a table to which the trigger is associated. It must be written after the ON keyword. If we did not specify the table name, a trigger would not exist.
* Finally, we will specify the trigger body that contains one or more statements for execution when the trigger is activated.

**5.Joints:**

* JOINS are used with SELECT statement. It is used to retrieve data from multiple tables. It is performed whenever you need to fetch records from two or more tables.
* There are three types of joins:
* MySQL INNER JOIN (or sometimes called simple join)
* MySQL LEFT OUTER JOIN (or sometimes called LEFT JOIN)
* MySQL RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)
* **INNER JOINT:**
* The [MySQL INNER JOIN](https://www.javatpoint.com/mysql-inner-join) is used to return all rows from multiple tables where the join condition is satisfied. It is the most common type of join.



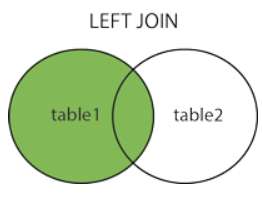
SYNTAX: SELECT columns

FROM table1

INNER JOIN table2

ON table1.column = table2.column;

* **LEFT OUTER JOIN**
* The LEFT OUTER JOIN returns all rows from the left hand table specified in the ON condition and only those rows from the other table where the join condition is fulfilled.



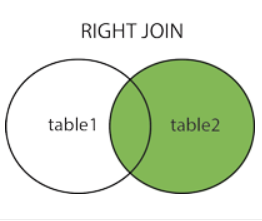
SYNTAX: SELECT columns

FROM table1

LEFT [OUTER] JOIN table2

ON table1.column = table2.column;

* **RIGHT OUTER JOIN**
* The MySQL Right Outer Join returns all rows from the RIGHT-hand table specified in the ON condition and only those rows from the other table where he join condition is fulfilled.

****

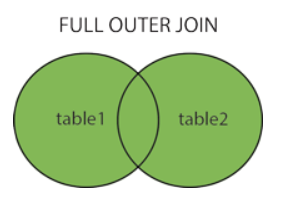
SYNTAX: SELECT columns

FROM table1

RIGHT [OUTER] JOIN table2

ON table1.column = table2.column;

* **OUTER JOIN**
* The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

****

* **CROSS JOIN**
* The CROSS JOIN keyword is always used with the SELECT statement and must be written after the FROM clause. The following syntax fetches all records from both joining tables:

SYNTAX :SELECT column-lists

FROM table1

CROSS JOIN table2;

* Examples for joins:
* -- Inner join

SELECT first\_name, last\_name, actor\_id, city.city, country.country

FROM actor

INNER JOIN city

INNER JOIN country

ON actor\_id = city.city\_id

AND country.country\_id ;

* --Outer join

select \* from actor,city where last\_name = "GUINESS";

* -- Left join

SELECT first\_name, last\_name, actor\_id, city, country

FROM actor

LEFT JOIN city

ON actor\_id = city.city\_id

LEFT JOIN country ON actor\_id = country.country\_id ;

* -- Right join

SELECT first\_name, last\_name, actor\_id, city, country

FROM actor

RIGHT JOIN city

ON actor\_id = city.city\_id

RIGHT JOIN country ON actor\_id = country.country\_id ;

* -- Cross join

SELECT \*

FROM actor

CROSS JOIN city

CROSS JOIN country;

* Difference between Lift join and Right join:

