**DAY-9**

1. **Go through the concept on Order by clause and store procedures.**

ANS: The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default.(ASC)

To sort the records in descending order, use the DESC keyword.

SELECT column1, column2, ...

FROM table\_name

ORDER BY column1, column2, ... ASC|DESC

**STORED PROCEDURES:**

A procedure (often called a stored procedure) is a collection of pre-compiled SQL statements stored inside the database. It is a subroutine or a subprogram in the regular computing language.

A procedure always contains a name, parameter lists, and SQL statements.

1. A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.
2. 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.
3. You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.
4. Stored procedures are reusable and transparent to any applications.
5. A procedure is always secure.

**CREATE** **PROCEDURE** procedure\_name [[IN | **OUT** | INOUT] parameter\_name datatype [, parameter datatype]) ]

**BEGIN**

    Declaration\_section

    Executable\_section

**END** &&

MySQL procedure parameter has one of three modes:

IN parameter

It is the default mode. It takes a parameter as input, such as an attribute. When we define it, the calling program has to pass an argument to the stored procedure. This parameter's value is always protected.

OUT parameters

It is used to pass a parameter as output. Its value can be changed inside the stored procedure, and the changed (new) value is passed back to the calling program. It is noted that a procedure cannot access the OUT parameter's initial value when it starts.

INOUT parameters

It is a combination of IN and OUT parameters. It means the calling program can pass the argument, and the procedure can modify the INOUT parameter, and then passes the new value back to the calling program.

Example:-

CREATE SCHEMA `priyanka` ;

CREATE PROCEDURE `details` ()

BEGIN

select Id,name,manager\_id,date\_of\_birth

from employeee;

END

USE `priyanka`;

DROP procedure IF EXISTS `details`

DELIMITER $$

USE `priyanka`$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `details`()

BEGIN

select Id,name,manager\_id,date\_of\_birth

from employee;

END$$

DELIMITER ;

1. **Write a store procedure to calculate the total marks of each student**

CREATE TABLE `priyanka`.`student\_sheet` (

`Id` INT NOT NULL,

`name` VARCHAR(45) NULL,

`SUB\_1` INT NULL,

`SUB\_2` INT NULL,

PRIMARY KEY (`Id`));

insert into student\_sheet(101,’priya’,85,80);

insert into student\_sheet(102,’jiya’,95,67);

insert into student\_sheet(103,’diya’,62,78);

CREATE PROCEDURE `student`()

BEGIN

select sum(SUB\_1+SUB\_2) AS total\_marks

from student\_sheet

group by(Id);

END

USE `priyanka`;

DROP procedure IF EXISTS `priyanka`.`student`;

DELIMITER $$

USE `priyanka`$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `student`()

BEGIN

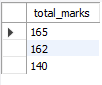
select sum(SUB\_1+SUB\_2) AS total\_marks

from student\_sheet

group by(Id);

END$$

DELIMITER ;



1. **Write a query which displays the list of students in ascending order based on there name along with the highest to lowest marks scored in different subject by each student.**

ANS :

CREATE DEFINER=`root`@`localhost` PROCEDURE `student`()

BEGIN

select name,SUB\_1,SUB\_2

from student\_sheet

group by(Id)

order by SUB\_1 DESC, SUB\_2 DESC, name ASC;

END

USE `priyanka`;

DROP procedure IF EXISTS `student`;

USE `priyanka`;

DROP procedure IF EXISTS `priyanka`.`student`;

DELIMITER $$

USE `priyanka`$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `student`()

BEGIN

select name,SUB\_1,SUB\_2

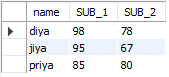
from student\_sheet

order by name, SUB\_1 DESC, SUB\_2 DESC;

END$$

DELIMITER ;

call priyanka.student();



1. **Add constrains in such a way that if a primary key related record is deleted then the corresponding record containing the foreign key should be deleted.**

ANS: **// table1(stu\_sheet)**

CREATE TABLE `priyanka`.`stu\_sheet` (

`Id` INT NOT NULL,

`name` VARCHAR(45) NULL,

`SUB\_1` INT NULL,

`SUB\_2` INT NULL,

PRIMARY KEY (`Id`));

insert into priyanka.stu\_sheet(ID,name,SUB\_1,SUB\_2) values (1,'priya',85,80);

insert into priyanka.stu\_sheet(ID,name,SUB\_1,SUB\_2) values (2,'jiya',95,67);

insert into priyanka.stu\_sheet(ID,name,SUB\_1,SUB\_2) values (3,'diya',62,78);

**// table2(stu\_records)**

CREATE TABLE `priyanka`.`stu\_records` (

`Id` INT NOT NULL,

`name` VARCHAR(45) NULL,

`mentor` VARCHAR(45) NULL,

`s\_id` INT NULL,

PRIMARY KEY (`Id`));

insert into priyanka.stu\_records (Id,name,mentor,s\_id) values (101,'priya','pinal',1);

insert into priyanka.stu\_records (Id,name,mentor,s\_id) values (102,'jiya','viral',2);

insert into priyanka.stu\_records (Id,name,mentor,s\_id) values (103,'diya','kavita',3);

ALTER TABLE priyanka.stu\_records

add constraint stu\_rec\_1

FOREIGN KEY(s\_id)

REFERENCES stu\_sheet(Id) ON delete CASCADE;

delete from priyanka.stu\_sheet

where Id=3;

select \* from priyanka.stu\_records; select \* from priyanka.stu\_sheet;



1. **Add constrains in such away that if a primary key related record is deleted then the corresponding record containing the foreign key should be update**

**ANS:**

ALTER TABLE priyanka.stu\_records

add constraint

FOREIGN KEY(s\_id)

REFERENCES stu\_sheet(Id) ON update CASCADE;

update priyanka.stu\_sheet

set Id=6

where Id=2;

select \* from priyanka.stu\_sheet select \* from priyanka.stu\_records

