

# **STUDENT DATABASE MANAGEMENT SYSTEM**

## **A PROJECT REPORT**

*Submitted by*

**MATHI TARUN [RA2011047010086]**

**SAKILI AJAY [ RA2011047010114]**

**DOMA PAVAN SRINIVAS [RA2011047010091]**

*Under the guidance of*

**DR. S. SELVAKUMARASAMY**

(Assistant Professor, Department of Computational Intelligence)

*in partial fulfillment for the award of the*

*degree of*

**BACHELOR OF TECHNOLOGY**

in

**COMPUTATIONAL INTELLIGENCE**

of

**FACULTY OF ENGINEERING AND TECHNOLOGY**



S.R.M. Nagar, Kattankulathur, Kancheepuram District

**JUNE 2022**

# **SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

(Under Section 3 of UGC Act, 1956)

## **BONAFIDE CERTIFICATE**

Certified that this project report titled “**STUDENT DATABASE MANAGEMENT SYSTEM**” is the bonafide work of “**MATHI TARUN [RA2011047010086, SAKILI AJAY [ RA2011047010114], DOMA PAVAN SRINIVAS [RA2011047010091]**”, who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

### **SIGNATURE**

**DR. S.  
SELVAKUMARASAMY  
GUIDE**  
Assistant Professor  
Dept. of Computational  
Intelligence

### **SIGNATURE**

**Dr. ANNIE UTHRA  
HEAD OF THE DEPARTMENT**  
Dept. of Computational Intelligence

Signature of the External Examiner

Signature of the Internal  
Examiner

## **ACKNOWLEDGEMENTS**

I would like to express my deepest gratitude to my guide, **Dr. S. SELVAKUMARASAMY**, his valuable guidance, consistent encouragement, personal caring, timely help and providing me with an excellent atmosphere for doing the project. All through the work, in spite of his busy schedule, he has extended cheerful and cordial support to me for completing this project work.

**MATHI TARUN**  
**SAKILI AJAY**  
**DOMA PAVAN SRINIVAS**

# CHAPTER 1

## INTRODUCTION ABOUT THE PLATFORMS WORKED

### 1.1 MySQL

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company. MySQL is becoming so popular because of many good reasons:

- MySQL is released under an open-source license. So you have nothing to pay to use it.
- MySQL is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc.
- MySQL works very quickly and works well even with large data sets.
- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).
- MySQL is very friendly to PHP, the most appreciated language for web development.
- MySQL is customizable. The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

## **ABSTRACT**

An organized and systematic office solution is essential for all universities and organizations. There are many departments of administration for the maintenance of college information and student databases in any institution. All these departments provide various records regarding students. Most of these track records need to maintain information about the students. This information could be the general details like student name, address, performance, attendance etc or specific information related to departments like collection of data.

All the modules in college administration are interdependent. They are maintained manually. So they need to be automated and centralized as, Information from one module will be needed by other modules. For example when a student needs his course completion certificate it needs to check many details about the student like his name, reg number, year of study, exams he attended and many other details. So it needs to contact all the modules that are office, department and examination and result of students.

Our work is useful for easy user interface. We are planning to utilize the powerful database management, data retrieval and data manipulation. We will provide more ease for managing the data than manually maintaining in the documents. Our work is useful for saving valuable time and reduces the huge paperwork.

## INTRODUCTION

Student Management System deals with all kind of student details, academic related reports, college details, course details, curriculum, batch details and other resource related details too. It tracks all the details of a student from the day one to the end of his course which can be used for all reporting purpose, tracking of attendance, progress in the course, completed semesters years, coming semester year curriculum details, exam details, project or any other assignment details, nal exam result etc.

Our design can facilitate us to explore all the activities happening in the college, even we can get to know which faculty is assigned to which course, the current status of a student, attendance percentage of a student and upcoming requirements of a student. The student management system is an automated version of manual Student Management System. It can handle all details about a student. The details include college details, subject details, student personnel details, academic details, exam details etc.

In case of manual system they need a lot of time, manpower etc. Here almost all work is computerized. So the accuracy is maintained. Maintaining backup is very easy. It can do with in a few minutes. Our system has two type of accessing modes, administrator and user. Student management system is managed by an administrator. It is the job of the administrator to insert update and monitor the whole process. When a user log in to the system. He/she would only view details of the student. He/she can't perform any changes. Our system has seven modules, they are administrator, student, course, department, exam, attendance, and section. These modules and its attributes with entity relationship module presented in the ER diagram section.

# DESIGN

## UML Design

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the software system and its components. It is a graphical language, which provides a vocabulary and set of semantics and rules. The UML focuses on the conceptual and physical representation of the system. It captures the decisions and understandings about systems that must be constructed. It is used to understand, design, configure, maintain, and control information about the systems.

The UML is a language for:

- Visualizing
- Specifying
- Constructing
- Documenting

### Visualizing

Through UML we see or visualize an existing system and ultimately we visualize how the system is going to be after implementation. Unless we think, we cannot implement. UML helps to visualize, how the components of the system communicate and interact with each other.

### Specifying

Specifying means building, models that are precise, unambiguous and complete UML addresses the specification of all the important analysis design, implementation decisions that must be made in developing and deploying a software system.

### Constructing

UML models can be directly connected to a variety of programming language through mapping a model from UML to a programming language like JAVA or C++ or VB. Forward Engineering and Reverse Engineering is possible through UML.

### Documenting

The Deliverables of a project apart from coding are some Artifacts, which are critical in controlling, measuring and communicating about a system during its developing

requirements, architecture, desire, source code, project plans, tests, prototypes, releases, etc...

## **4.2 UML Approach**

### **UML Diagram**

A diagram is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices and arcs. You draw a diagram to visualize a system from a different perspective, so a diagram is a projection into a system. For all but most trivial systems, a diagram represents an elided view of the elements that make up a system. The same element may appear in all diagrams, only a few diagrams, or in no diagrams at all. In theory, a diagram may contain any combination of things and relationships. In practice, however, a small number of common combinations arise, which are consistent with the five most useful views that comprise the architecture of a software-intensive system. For this reason, the UML includes nine such diagrams:

1. Class diagram
2. Object diagram
3. Use case diagram
4. Sequence diagram
5. Collaboration diagram
6. State chart diagram
7. Activity diagram
8. Component diagram
9. Deployment diagram

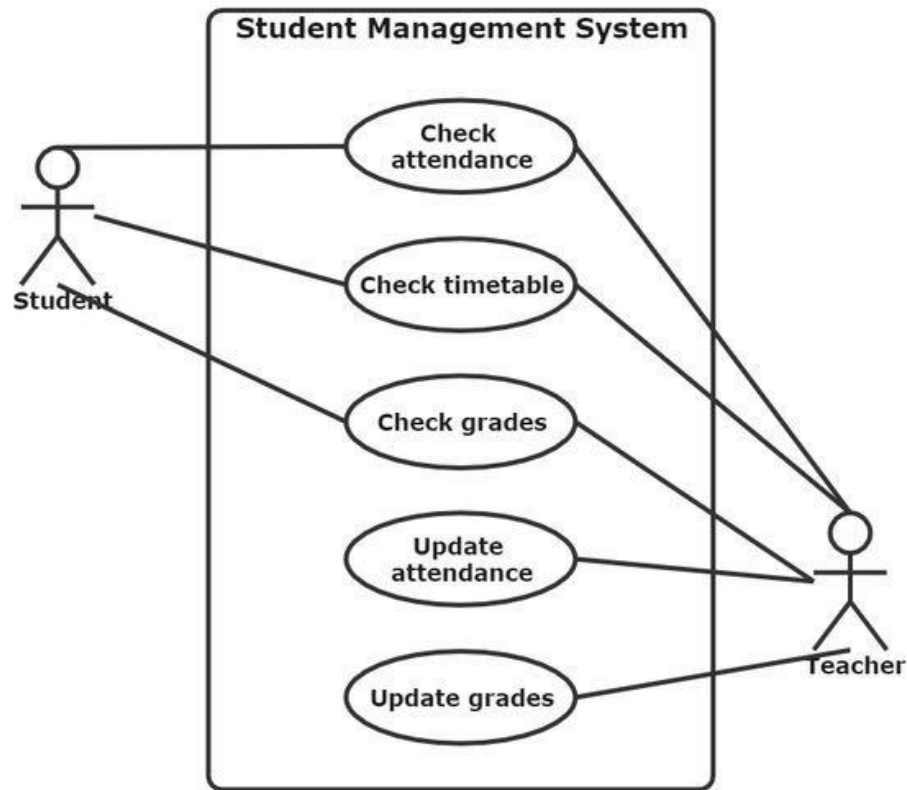
### **USE CASE DIAGRAM:**

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases.

Use case diagrams are formally included in two modeling languages defined by the OMG: the Unified Modeling Language (UML) and the Systems Modeling Language (SysML).

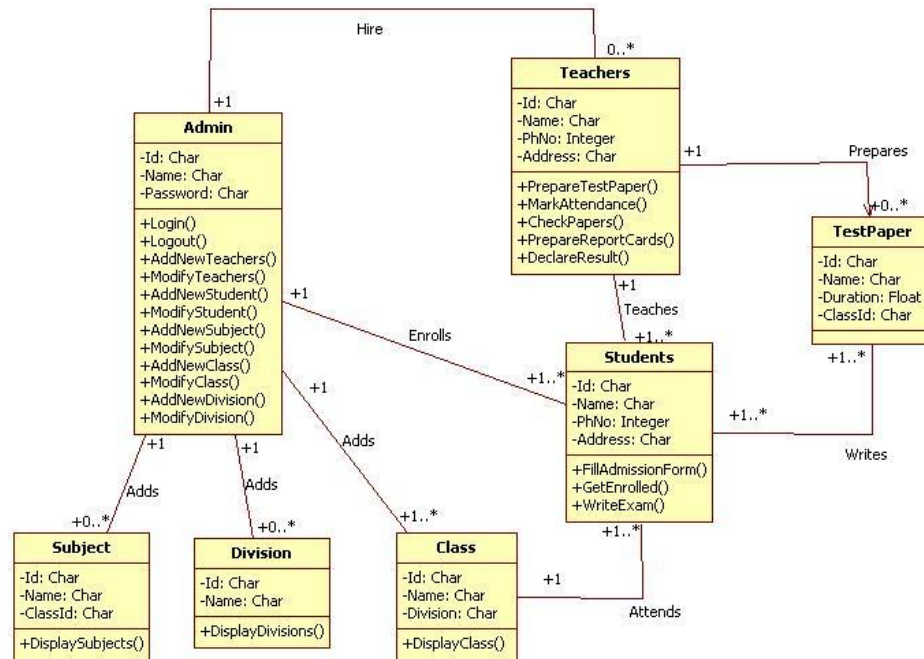


**Use case diagram of our project:**



### **Class Diagram:**

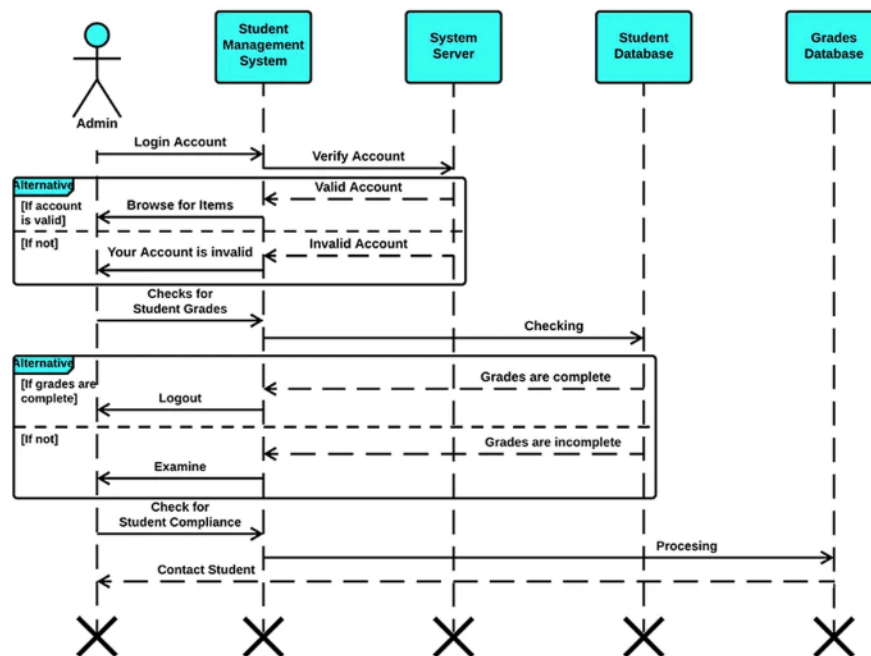
A Class is a category or group of things that has similar attributes and common behavior. A Rectangle is the icon that represents the class it is divided into three areas. The upper most area contains the name, the middle; area contains the attributes and the lowest areas show the operations. Class diagrams provides the representation that developers work from. Class diagrams help on the analysis side, too.



<http://umldiagramtutorial.blogspot.in/>

## Sequence diagram:

A **Sequence Diagram** is an interaction diagram that emphasizes the time ordering of messages; a collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages. Sequence diagrams and collaboration diagrams are isomorphic, meaning that you can take one and transform it into the other.



# 1. EXECUTION OF BASIC COMMANDS IN SQL

16.03.22

**AIM:** To create a table and execute the basic commands in SQL.

## **COMMANDS:**

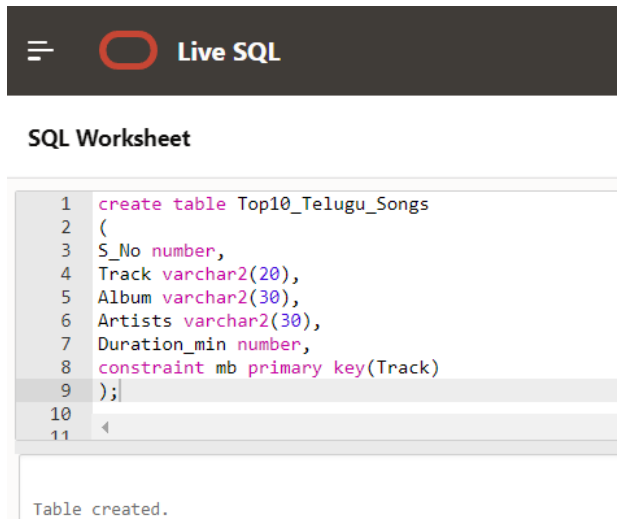
**1. CREATE:** It is used to create a new table in the database with attributes and its datatypes.

**SYNTAX:** create table table\_name (column\_name datatypes[,....]);

### **Code:**

```
create table Top10_Telugu_Songs
(
S_No number,
Track varchar2(20),
Album varchar2(30),
Artists varchar2(30),
Duration_min number,
constraint mb primary key(Track)
);
```

### **Output:**



The screenshot shows a web-based SQL editor interface. At the top, there is a dark header bar with a hamburger menu icon, a red circular logo, and the text "Live SQL". Below the header, the title "SQL Worksheet" is displayed. The main area contains a text editor with the following SQL code: 

```
1 create table Top10_Telugu_Songs
2 (
3 S_No number,
4 Track varchar2(20),
5 Album varchar2(30),
6 Artists varchar2(30),
7 Duration_min number,
8 constraint mb primary key(Track)
9 );
10
11
```

 Below the code editor, a status message "Table created." is visible.

**2. INSERT:** It is used to insert data into the row of a table.

**SYNTAX:** INSERT INTO TABLE\_NAME (col1, col2, col3,.... col N) VALUES (value1, value2, value3, .... valueN)

### **Code:**

```
insert into Top10_Telugu_Songs (S_No,Track,Album,Artists,Duration_min) values
(1,'Kalaavathi','Sarkaru Vaari Paata','Sid Sriram',4);
```

```
insert into Top10_Telugu_Songs (S_No,Track,Album,Artists,Duration_min) values (2,'Tillu Anna','DJ
Tillu','Ram',3);
```

```
insert into Top10_Telugu_Songs (S_No,Track,Album,Artists,Duration_min) values (3,'Oo Antava','Pushpa','Indravathi',4);
```

```
insert into Top10_Telugu_Songs (S_No,Track,Album,Artists,Duration_min) values (4,'Naa Kosam','Bangarraju','Sid Sriram',4);
```

```
insert into Top10_Telugu_Songs (S_No,Track,Album,Artists,Duration_min) values (5,'Ee Raathale','Radhe Shyam','Yuvan Shankar Raja',4);
```

```
insert into Top10_Telugu_Songs (S_No,Track,Album,Artists,Duration_min) values (6,'Saami Saami','Pushpa','Mounika',4);
```

```
insert into Top10_Telugu_Songs (S_No,Track,Album,Artists,Duration_min) values (7,'Jai Balayya','Akhand','Geetha Madhuri',4);
```

```
insert into Top10_Telugu_Songs (S_No,Track,Album,Artists,Duration_min) values (8,'Bangaara','Bangarraju','Madhu Priya',3);
```

```
insert into Top10_Telugu_Songs (S_No,Track,Album,Artists,Duration_min) values (9,'Srivalli','Pushpa','Sid Sriram',4);
```

```
insert into Top10_Telugu_Songs (S_No,Track,Album,Artists,Duration_min) values (10,'Atta Sudake','Khiladi','DSP',3);
```

## Output:

The screenshot shows an SQL Worksheet interface with a toolbar at the top containing 'Clear', 'Find', 'Actions', 'Save', and 'Run' buttons. The main area displays 10 SQL insert statements, each on a new line, numbered 13 to 22. The statements are: 13. insert into Top10\_Telugu\_Songs (S\_No,Track,Album,Artists,Duration\_min) values (1,'Kalaavathi','Sarkaru Vaari Paata','Sid Sriram',4); 14. insert into Top10\_Telugu\_Songs (S\_No,Track,Album,Artists,Duration\_min) values (2,'Tillu Anna','DJ Tillu','Ram',3); 15. insert into Top10\_Telugu\_Songs (S\_No,Track,Album,Artists,Duration\_min) values (3,'Oo Antava','Pushpa','Indravathi',4); 16. insert into Top10\_Telugu\_Songs (S\_No,Track,Album,Artists,Duration\_min) values (4,'Naa Kosam','Bangarraju','Sid Sriram',4); 17. insert into Top10\_Telugu\_Songs (S\_No,Track,Album,Artists,Duration\_min) values (5,'Ee Raathale','Radhe Shyam','Yuvan Shankar Raja',4); 18. insert into Top10\_Telugu\_Songs (S\_No,Track,Album,Artists,Duration\_min) values (6,'Saami Saami','Pushpa','Mounika',4); 19. insert into Top10\_Telugu\_Songs (S\_No,Track,Album,Artists,Duration\_min) values (7,'Jai Balayya','Akhand','Geetha Madhuri',4); 20. insert into Top10\_Telugu\_Songs (S\_No,Track,Album,Artists,Duration\_min) values (8,'Bangaara','Bangarraju','Madhu Priya',3); 21. insert into Top10\_Telugu\_Songs (S\_No,Track,Album,Artists,Duration\_min) values (9,'Srivalli','Pushpa','Sid Sriram',4); 22. insert into Top10\_Telugu\_Songs (S\_No,Track,Album,Artists,Duration\_min) values (10,'Atta Sudake','Khiladi','DSP',3); Below the statements, the execution results are shown, indicating that each statement successfully inserted one row (1 row(s) inserted. for each of the 10 statements).

**3. ALTER TABLE:** It is used to alter the structure of the table.

a.) **ADD:** To add a new column in the table

**SYNTAX:** ALTER TABLE table\_name ADD column\_name COLUMN-definition

**Code:**

```
alter table Top10_Telugu_Songs add (Music_Director varchar2(20));
```

## Output:

### SQL Worksheet

```
22 insert into Top10_Telugu_Songs (S_No,Track,Music_Director,Duration_min,
23
24 alter table Top10_Telugu_Songs add (Music_Director varchar2(20));|
25
26 alter table Top10_Telugu_Songs rename column Artists to Artist;
```

Table altered.

b.) **UPDATE:** It is command is used to update or modify the value of a column in the table.

**SYNTAX:** UPDATE table\_name SET [column\_name1= value1,...column\_nameN = valueN]  
[WHERE CONDITION]

## Code:

```
update Top10_Telugu_Songs set Music_Director = 'Thaman S' where Track = 'Kalaavathi';
update Top10_Telugu_Songs set Music_Director = 'Sricharan' where Track = 'Tillu Anna';
update Top10_Telugu_Songs set Music_Director = 'DSP' where Track = 'Oo Antava';
update Top10_Telugu_Songs set Music_Director = 'Anup Rubens' where Track = 'Naa Kosam';
update Top10_Telugu_Songs set Music_Director = 'Justin Prabhakaran' where Track = 'Ee Raathale';
update Top10_Telugu_Songs set Music_Director = 'DSP' where Track = 'Saami Saami';
update Top10_Telugu_Songs set Music_Director = 'Thaman S' where Track = 'Jai Balayya';
update Top10_Telugu_Songs set Music_Director = 'Anup Rubens' where Track = 'Bangaara';
update Top10_Telugu_Songs set Music_Director = 'DSP' where Track = 'Srivalli';
update Top10_Telugu_Songs set Music_Director = 'DSP' where Track = 'Atta Sudake';
```

## Output:

### SQL Worksheet

Clear

```
28 update Top10_Telugu_Songs set Music_Director = 'Thaman S' where Track = 'Kalaavathi';
29 update Top10_Telugu_Songs set Music_Director = 'Sricharan' where Track = 'Tillu Anna';
30 update Top10_Telugu_Songs set Music_Director = 'DSP' where Track = 'Oo Antava';
31 update Top10_Telugu_Songs set Music_Director = 'Anup Rubens' where Track = 'Naa Kosam';
32 update Top10_Telugu_Songs set Music_Director = 'Justin Prabhakaran' where Track = 'Ee Raathale';
33 update Top10_Telugu_Songs set Music_Director = 'DSP' where Track = 'Saami Saami';
34 update Top10_Telugu_Songs set Music_Director = 'Thaman S' where Track = 'Jai Balayya';
35 update Top10_Telugu_Songs set Music_Director = 'Anup Rubens' where Track = 'Bangaara';
36 update Top10_Telugu_Songs set Music_Director = 'DSP' where Track = 'Srivalli';
37 update Top10_Telugu_Songs set Music_Director = 'DSP' where Track = 'Atta Sudake';
38
39
```

1 row(s) updated.

1 row(s) updated.

1 row(s) updated.

**4. DESC:** It is used to describe the table.

**SYNTAX:** Desc tablename

**Code:**

```
desc Top10_Telugu_Songs
```

**Output:**

```
9 );
10
11 desc Top10_Telugu_Songs
12
```

TABLE TOP10\_TELUGU\_SONGS

Column	Null?	Type
S_NO	-	NUMBER
TRACK	NOT NULL	VARCHAR2(20)
ALBUM	-	VARCHAR2(30)
ARTISTS	-	VARCHAR2(30)
DURATION_MIN	-	NUMBER

[Download CSV](#)  
5 rows selected.

**5. Select:** It is used to display the Created Table.

**Syntax:** select \*from table name

**Code:**

```
select *from Top10_Telugu_Songs
```

**Output:**

Live SQL
Feedback

**SQL Worksheet**
Clear
Find

```

37
38 select *from Top10_Telugu_Songs
39
40

```

S_NO	TRACK	ALBUM	ARTISTS	DURATION_MIN	MUSIC_DIRECTOR
1	Kalaavathi	Sarkaru Vaari Paata	Sid Sriram	4	Thaman S
2	Tillu Anna	DJ Tillu	Ram	3	Sricharan
3	Oo Antava	Pushpa	Indravathi	4	DSP
4	Naa Kosam	Bangarraju	Sid Sriram	4	Anup Rubens
5	Ee Raathale	Radhe Shyam	Yuvan Shankar Raja	4	Justin Prabhakaran
6	Saami Saami	Pushpa	Mounika	4	DSP
7	Jai Balayya	Akhanda	Geetha Madhuri	4	Thaman S
8	Bangaara	Bangarraju	Madhu Priya	3	Anup Rubens
9	Srivalli	Pushpa	Sid Sriram	4	DSP
10	Atta Sudake	Khiladi	DSP	3	DSP

[Download CSV](#)  
10 rows selected.

**Result:** Hence, from the above experiment we studied and executed basic commands in SQL on the created table.

## **2. ER Diagrams in Data Base Management System**

**23.03.22**

**Aim:** Some important ER diagrams in Data Base Management System.

### **Description:**

An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are entity set and relationship set.

**Rectangle:** Represents Entity sets

**Ellipses:** Attributes

**Diamonds:** Relationship Set

**Lines:** They link attributes to Entity Sets and Entity sets to Relationship Set

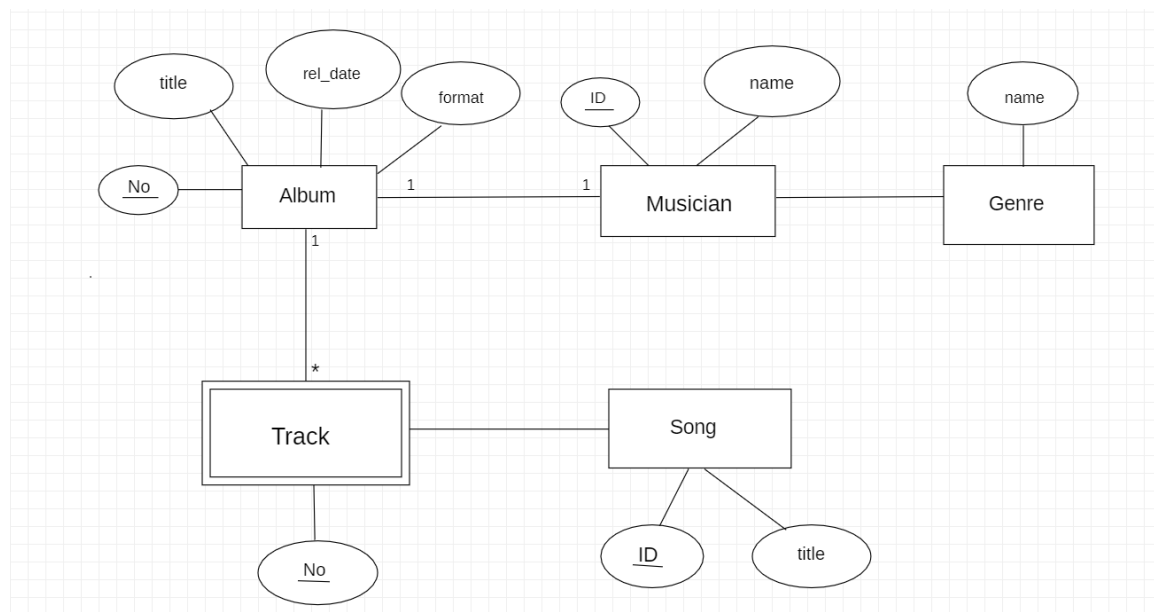
**Double Ellipses:** Multivalued Attributes

**Dashed Ellipses:** Derived Attributes

**Double Rectangles:** Weak Entity Sets

**Double Lines:** Total participation of an entity in a relationship set

### **Output:**





### **3. SQL Transactions commands**

**30.03.22**

#### **1)Execute and show Transactions commands in SQL.**

**Aim:** Some important Transactions commands in SQL.

**Description:**

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: **success** or **failure**.

**1. BEGIN TRANSACTION:** It indicates the start point of an explicit or local transaction.

**Syntax:**

BEGIN TRANSACTION transaction\_name ;

**2. SET TRANSACTION:** Places a name on a transaction.

**Syntax:**

SET TRANSACTION [ READ WRITE | READ ONLY];

**3. COMMIT:** If everything is in order with all statements within a single transaction, all changes are recorded together in the database is called **committed**. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

**Syntax:**

COMMIT;

**4. ROLLBACK:** If any error occurs with any of the SQL grouped statements, all changes need to be aborted. The process of reversing changes is called **rollback**. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

**Syntax:**

ROLLBACK;

**5. SAVEPOINT:** creates points within the groups of transactions in which to ROLLBACK.

A SAVEPOINT is a point in a transaction in which you can roll the transaction back to a certain point without rolling back the entire transaction.

### Syntax for Savepoint command:

SAVEPOINT SAVEPOINT\_NAME;

This command is used only in the creation of SAVEPOINT among all the transactions. In general ROLLBACK is used to undo a group of transactions.

### Syntax for rolling back to Savepoint command:

ROLLBACK TO SAVEPOINT\_NAME;

## Output:

#### SQL Worksheet

```
1 create table Student (  
2 Roll_No number primary key,  
3 Student_Name varchar(20),  
4 Address varchar(30),  
5 Mobile_Number number,  
6 Age number,  
7 Blood_Group varchar(10))  
8  
9 insert into Student (Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group) values (1,'Prince','AP',9848926526,18,'A+ve');  
10 insert into Student (Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group) values (2,'Charan','TS',8542682549,19,'O+ve');  
11 insert into Student (Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group) values (3,'Tarak','TN',7821060252,18,'O-ve');  
12 insert into Student (Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group) values (4,'Nani','AP',9000056485,19,'B+ve');  
13 insert into Student (Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group) values (5,'Chiru','AP',7825647502,18,'A-ve');  
14 insert into Student (Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group) values (6,'Vijay','TS',8210564852,18,'B+ve');  
15  
16 delete from Student where Roll_No=4;  
17  
18 select * from Student
```

ROLL_NO	STUDENT_NAME	ADDRESS	MOBILE_NUMBER	AGE	BLOOD_GROUP
1	Prince	AP	9848926526	18	A+ve
2	Charan	TS	8542682549	19	O+ve
3	Tarak	TN	7821060252	18	O-ve
5	Chiru	AP	7825647502	18	A-ve
6	Vijay	TS	8210564852	18	B+ve

[Download CSV](#)

```

1 create table Student (
2 Roll_No number primary key,
3 Student_Name varchar(20),
4 Address varchar(30),
5 Mobile_Number number,
6 Age number,
7 Blood_Group varchar(10))
8
9 insert into Student (Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group) values (1,'Prince','AP',9848926526,18,'A+ve');
10 insert into Student (Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group) values (2,'Charan','TS',8542682549,19,'O+ve');
11 insert into Student (Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group) values (3,'Tarak','TN',7821060252,18,'O-ve');
12 insert into Student (Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group) values (4,'Nani','AP',9000056485,19,'B+ve');
13 insert into Student(Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group) values (5,'Chiru','AP',7825647502,18,'A-ve');
14 insert into Student (Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group) values (6,'Vijay','TS',8210564852,18,'B+ve');
15
16 delete from Student where Roll_No=4;
17
18 select * from Student
19
20 commit
21
22 ROLLBACK;
23
24 select * from Student
25
26 SAVEPOINT;
27

```

**Result:** From the above Experiment, we understand the concept of SQL Transactions commands.

## **4. SQL BUILT-IN FUNCTIONS**

**06.04.22**

### **1)Execute and show BUILT-IN commands in SQL.**

**Aim:** some important BUILT-IN commands in SQL.

#### **Description:**

##### **1.The SQL CREATE TABLE Statement**

The CREATE TABLE statement is used to create a new table in a database.

Syntax: CREATE TABLE *table\_name* (  
    *column1*      *datatype*,  
    *column2*      *datatype*,  
    *column3* *datatype*,  
    ....  
);

##### **2. The SQL INSERT INTO Statement**

The INSERT INTO statement is used to insert new records in a table.

Syntax: INSERT INTO *table\_name* (*column1*, *column2*, *column3*,  
...)  
VALUES (*value1*, *value2*, *value3*, ...);

##### **3. The SQL COUNT(), AVG() and SUM() Functions**

###### **i) COUNT()**

Syntax: **SELECT COUNT(*column\_name*) FROM *table\_name*  
WHERE *condition*;**

###### **ii) AVG()**

Syntax : **SELECT AVG(*column\_name*) FROM *table\_name*  
WHERE *condition*;**

iii) Sum()

Syntax: **SELECT SUM(*column\_name*) FROM *table\_name***  
**WHERE *condition*;**

#### **4. The SQL MIN() and MAX() Functions**

i) MIN()

Syntax: **SELECT MIN(*column\_name*) FROM *table\_name***  
**WHERE *condition*;**

ii) Max()

syntax: **SELECT MAX(*column\_name*) FROM *table\_name***  
**WHERE *condition*;**

#### **5. The SQL BETWEEN**

Operator BETWEEN

Syntax: **SELECT *column\_name(s)* FROM *table\_name***  
**WHERE *column\_name* BETWEEN *value1* AND *value2*;**

#### **6. The SQL NOT BETWEEN operator**

Syntax: **SELECT \* FROM**

***column\_name***

**WHERE *column\_name* NOT BETWEEN *value1* AND *value2*;**

## SQL Worksheet

```
1 create table Employee(  
2     EmpId number primary key,  
3     Empname varchar(15));  
4  
5 insert into Employee (EmpId,Empname) values (203,'Tharun');  
6 insert into Employee (EmpId,Empname) values (420,'Ranga Sai');  
7 insert into Employee (EmpId,Empname) values (840,'Pavan');  
8 insert into Employee (EmpId,Empname) values (210,'Sandeep');  
9 insert into Employee (EmpId,Empname) values (350,'Chandan');  
10  
11 select * from Employee;  
12 select count(Empname) as no_of_Employees from Employee;  
13 select count(*) from Employee;  
14 select max(EmpId) from Employee;  
15 select min(EmpId) from Employee;  
16 select avg(EmpId) from Employee;  
17 select sum(EmpId) from Employee;  
18 select Empname,sqrt(EmpId) from Employee;  
19 select concat(EmpId,Empname) from Employee;  
20 select upper(Empname) from Employee;  
21 select lower(Empname) from Employee;  
22 select greatest(EmpId) from Employee;  
23 select * from Employee where EmpId between 203 and 420;  
  
24 select * from Employee where EmpId between 203 and 420 and EmpId not in (203,420);  
25 select * from Employee order by Empname;  
26 select * from Employee where Empname not between 'Tharun' and 'Pavan' order by Empname;  
27 select * from Employee where EmpId = 420;  
28 select * from Employee where EmpId > 420;  
29 select * from Employee where EmpId <> 420;  
30 select * from Employee where Empname = 'Tharun' and EmpId = 203;  
31 select * from Employee where Empname = 'Tharun' or EmpId = 420;  
32  
33  
34  
35  
36  
37
```

## SQL Worksheet

```
1 create table toy(  
2     toy_id integer,  
3     toy_name varchar(100),  
4     color varchar(20));  
5 insert into toy (toy_id,toy_name,color) values (1,'Tiger','orange');  
6 insert into toy(toy_id,toy_name) values (2,'Lion');  
7 insert into toy(toy_id,toy_name,color) values (3,'cheetah','pink');  
8 insert into toy(toy_id,toy_name,color) values (4,'bear','brown');  
9 select * from toy;  
10  
11 insert into toy(toy_id,toy_name ,color) values (8,'Elephant','black');  
12 exec savepoint after_six;  
13  
14 select * from toy;  
15 insert into toy ( toy_id, toy_name, color) values ( 9, 'Purple Ninja', 'purple' );  
16 select * from toy;  
17 rollback to after_six;  
18 select * from toy;  
19  
20
```

# OUTPUT:

My Session

4 rows selected.

Statement 7

SELECT COUNT(emname) FROM employee

COUNT(EMNAME)
4

Download CSV

Statement 8

SELECT COUNT(\*) FROM employee

COUNT(*)
4

Download CSV

Statement 9

SELECT MAX(empid)FROM employee

MAX(EMPID)
4

Download CSV

Statement 10

SELECT Min(empid)FROM employee

MIN(EMPID)
1

Download CSV

My Session

Download CSV

Statement 11

SELECT AVG(empid)FROM employee

AVG(EMPID)
2.5

Download CSV

Statement 12

SELECT sum(empid)FROM employee

SUM(EMPID)
10

Download CSV

Statement 13

SELECT emname, SQRT(empid) FROM employee

EMNAME	SQRT(EMPID)
name	1
name1	1.41421356237309504880168872420969807857
name2	1.73205080756887729352744634150587236694
name3	2

Download CSV

4 rows selected.

## My Session

Download CSV  
4 rows selected.

Statement 14



SELECT CONCAT(empid, emname) FROM employee

CONCAT(EMPID,EMNAME)

1name

2name1

3name2

4name3

Download CSV  
4 rows selected.

Statement 15



SELECT UPPER (emname) from employee

UPPER(EMNAME)

NAME

NAME1

NAME2

NAME3

Download CSV  
4 rows selected.

Statement 16



select lower(emname) from employee

## My Session

Statement 22



SELECT \* FROM employee WHERE emname NOT BETWEEN 'name' AND 'name1' ORDER BY emname

EMPID	EMNAME
3	name2
4	name3

Download CSV  
2 rows selected.

Statement 23



SELECT \* FROM employee WHERE empid = 4

EMPID	EMNAME
4	name3

Download CSV

Statement 24



SELECT \* FROM employee WHERE empid >=3

EMPID	EMNAME
3	name2
4	name3

Download CSV  
2 rows selected.



1name

Download CSV

My Session

Statement 29

SELECT \* FROM employee WHERE emname = 'name' or empid = 2

EMPID	EMNAME
1	name
2	name1

Download CSV

2 rows selected.

Statement 30

SELECT \* FROM employee WHERE empid > ANY (SELECT empid FROM employee WHERE empid > 3)

no data found

Statement 31

SELECT \* FROM employee WHERE empid > ANY (SELECT empid FROM employee WHERE empid > 1)

EMPID	EMNAME
3	name2
4	name3

Download CSV

2 rows selected.

**Result:** From the above, experiment we understand the concept of BUILT-IN COMMANDS in SQL.

## 5. SQL DDL & DML commands FOR Project

04.05.22

### Aim:

Some important DDL AND DML commands in SQL.

### DDL Commands:

**CREATE:** It is used to create a new table in the database with attributes and its datatypes.

**Code:** create table SRM\_STUDENTS( STUD\_ID int,STUD\_NAME varchar(20),STUD\_DEPT varchar(20),STUD\_YEAR int);

**INSERT:** It is used to insert data into the row of a table.

**SYNTAX:** INSERT INTO TABLE\_NAME (col1, col2, col3,.... col N)  
VALUES (value1, value2, value3, .... valueN)

**ALTER TABLE:** It is used to alter the structure of the table.

**Code:** alter TABLE SRM\_STUDENTS ADD STUD\_DOB date;

**Drop:** A Drop statement in SQL removes a component from a relational database management system(RDBMS)

**Syntax:** DROP object object\_name

**Code:** DROP table SRM\_STUDENTS;

### Output:

```
1 create table SRM_STUDENTS( STUD_ID int,STUD_NAME varchar(20),STUD_DEPT varchar(20),STUD_YEAR int);
2 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
3 (115,'Ajay','AI',2);
4 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
5 (86,'THARUN','IT',3);
6 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
7 (91,'PAVAN','CSE',4);
8 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
9 (152,'SANGA','EEE',3);
10 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
11 (154,'SANDEEP','ECE',2);
12 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
13 (143,'RANGA SAI','AI',3);
14 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
15 (123,'CHARITH','EEE',4);
16 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
17 (111,'ABINAY','AI',2);
18 alter TABLE SRM_STUDENTS ADD STUD_DOB date;
19 DROP table SRM_STUDENTS;
20 |
```

## DML Commands:

**Select:** It is used to display the Created Table.

**Syntax:** select \*from table name

Code: select \* from SRM\_STUDENTS;

**INSERT:** It is used to insert data into the row of a table.

**SYNTAX:** INSERT INTO TABLE\_NAME (col1, col2, col3,... col N) VALUES (value1, value2, value3, .... valueN)

**UPDATE:** It is command is used to update or modify the value of a column in the table.

**SYNTAX:** UPDATE table\_name SET [column\_name1= value1,...column\_nameN = valueN] [WHERE CONDITION]

Code: UPDATE SRM\_STUDENTS SET (STUD\_NAME ="AJAY",STUD\_ID = 115) WHERE STUD\_ID= 86;

**DELETE:** Used when we specify the row that we want to remove or delete from the table or relation.

**Syntax:** DELETE FROM table\_name;

Code: DELETE from SRM\_STUDENTS where STUD\_ID = 86;

## Output:

```
1 create table SRM_STUDENTS( STUD_ID int,STUD_NAME varchar(20),STUD_DEPT varchar(20),STUD_YEAR int);
2 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
3 (115,'Ajay','AI',2);
4 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
5 (86,'THARUN','IT',3);
6 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
7 (91,'PAVAN','CSE',4);
8 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
9 (152,'SANGA','EEE',3);
10 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
11 (154,'SANDEEP','ECE',2);
12 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
13 (143,'RANGA SAI','AI',3);
14 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
15 (123,'CHARITH','EEE',4);
16 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
17 (111,'ABINAY','AI',2);
18 UPDATE SRM_STUDENTS SET (STUD_NAME ='AJAY',STUD_ID = 115) WHERE STUD_ID= 86;
19 DELETE from SRM_STUDENTS where STUD_ID = 86;
20 |
```

**Result:** FROM THE ABOVE EXPERIMENT, we observed some of the DDL & DML Commands in sql.

## 6. SQL Transactions commands FOR Project

04.05.22

### **Aim:**

Some important Transactions commands in SQL.

### **Description:**

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: **success** or **failure**.

**1. BEGIN TRANSACTION:** It indicates the start point of an explicit or local transaction.

**Syntax:** BEGIN TRANSACTION transaction name;

**2. SET TRANSACTION:** Places a name on a transaction.

**Syntax:** SET TRANSACTION [ READ WRITE | READ ONLY];

**3. COMMIT:** If everything is in order with all statements within a single transaction, all changes are recorded together in the database is called **committed**. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.

**Syntax:** COMMIT;

**4. ROLLBACK:** If any error occurs with any of the SQL grouped statements, all changes need to be aborted. The process of reversing changes is called **rollback**. This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

**Syntax:** ROLLBACK;

**5. SAVEPOINT:** creates points within the groups of transactions in which to ROLLBACK.

A SAVEPOINT is a point in a transaction in which you can roll the transaction back to a certain point without rolling back the entire transaction.

**Syntax:** SAVEPOINT SAVEPOINT\_NAME;

This command is used only in the creation of SAVEPOINT among all the transactions. In general ROLLBACK is used to undo a group of transactions.

**Syntax:** ROLLBACK TO SAVEPOINT\_NAME;

**Output:**

1	create table Student (	
2	Roll_No number primary key,	
3	Student_Name varchar(20),	
4	Address varchar(30),	
5	Mobile_Number number,	
6	Age number,	
7	Blood_Group varchar(10))	
8		
9	insert into Student (Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group)	values (1,'Prince','AP',9848926526,18,'A+ve');
10	insert into Student (Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group)	values (2,'Charan','TS',8542682549,19,'O+ve');
11	insert into Student (Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group)	values (3,'Tarak','TN',7821060252,18,'O-ve');
12	insert into Student (Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group)	values (4,'Nani','AP',9000056485,19,'B+ve');
13	insert into Student (Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group)	values (5,'Chiru','AP',7825647502,18,'A-ve');
14	insert into Student (Roll_No,Student_Name,Address,Mobile_Number,Age,Blood_Group)	values (6,'Vijay','TS',8210564852,18,'B+ve');
15		
16	delete from Student where Roll_No=4;	
17		
18	select * from Student	
19		
20	commit	
21		
22	ROLLBACK;	
23		
24	select * from Student	
25		
26	SAVEPOINT;	
27		

**RESULT:** FROM THE ABOVE EXPERIMENT, we observed some of the TCL Commands in sql.

## **7. SQL BUILT-IN FUNCTIONS FOR PROJECT**

**04.05.22**

### **Aim:**

Some important BUILT-IN commands in SQL.

### **Description:**

#### **1. The SQL CREATE TABLE Statement**

The **CREATE TABLE** statement is used to create a new table in a database.

**Syntax:** CREATE TABLE *table name* (  
    *column1 datatype,*  
    *column2 datatype,*  
    *column3 datatype,*  
    ....  
);

#### **2. The SQL INSERT INTO Statement**

The INSERT INTO statement is used to insert new records in a table.

**Syntax:** INSERT INTO *table name* (*column1, column2, column3, ...*)  
VALUES (*value1, value2, value3, ...*);

#### **3. The SQL COUNT(), AVG() and SUM() Functions**

##### **i) COUNT()**

Syntax: SELECT COUNT(*column name*) FROM *table name*  
WHERE *condition*;

##### **ii) AVG()**

Syntax: SELECT AVG(*column name*) FROM *table name*  
WHERE *condition*;

##### **iii) Sum()**

**Syntax:** SELECT SUM(*column name*) FROM *table name* WHERE *condition*;

#### **4. The SQL MIN() and MAX() Functions**

##### **i) MIN()**

**Syntax:** SELECT MIN(*column name*) FROM *table name* WHERE *condition*;

ii) Max()

**Syntax:** SELECT MAX(*column name*) FROM *table name* WHERE *condition*;

## 5. The SQL BETWEEN Operator

### BETWEEN

**Syntax:** SELECT *column name(s)* FROM *table name* WHERE *column name* BETWEEN *value1* AND *value2*;

## 6.The SQL NOT BETWEEN operator

**Syntax:** SELECT \* FROM *column name* WHERE *column name* NOT BETWEEN *value1* AND *value2*;

```
1  -- BUILT IN FUNCTIONS
2  create table SRM_STUDENTS( STUD_ID int,STUD_NAME varchar(20),STUD_DEPT varchar(20),STUD_YEAR int);
3  insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
4  (115,"Ajay","AI",2);
5  insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
6  (86,"THARUN","IT",3);
7  insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
8  (91,"PAVAN","CSE",4);
9  insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
10 (152,"SANGA","EEE",3);
11 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
12 (154,"SANDEEP","ECE",2);
13 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
14 (143,"RANGA SAI","AI",3);
15 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
16 (123,"CHARITH","EEE",4);
```

```
17 insert into SRM_STUDENTS(STUD_ID,STUD_NAME,STUD_DEPT,STUD_YEAR) values
18 (111,"ABINAY","AI",2);
19 select * from SRM_STUDENTS;
20 select count(STUD_ID) from SRM_STUDENTS where STUD_ID>91;
21 select avg(STUD_YEAR) from SRM_STUDENTS;
22 select sum(STUD_ID) from SRM_STUDENTS;
23 select min(STUD_YEAR) from SRM_STUDENTS;
24 select max(STUD_ID) from SRM_STUDENTS;
25 select upper(STUD_NAME) from SRM_STUDENTS;
26 select lower(STUD_NAME) from SRM_STUDENTS;
27 select * from SRM_STUDENTS where STUD_ID BETWEEN 86 and 143;
28 select * from SRM_STUDENTS ORDER by STUD_NAME;
29 select * from SRM_STUDENTS where STUD_ID> 111;
30 select * from SRM_STUDENTS where STUD_ID <> 111;
31 select * from SRM_STUDENTS where STUD_ID < 111;
```

```
32 SELECT * from SRM_STUDENTS where STUD_NAME = "THARUN" and STUD_ID = 86;
33
34 -- TRANSACTIONS COMMANDS
35 delete from SRM_STUDENTS where STUD_ID = 86;
36 commit; -- COMMIT TRANSCATIONS COMMAND
37 select * from SRM_STUDENTS;
38 select * from SRM_STUDENTS ORDER by STUD_NAME;
39 delete from SRM_STUDENTS where STUD_ID = 86;
40 ROLLBACK;
41 select * from SRM_STUDENTS;
42 SAVEPOINT SP1;
43 delete from SRM_STUDENTS where STUD_ID = 86;
44 SAVEPOINT SP2;
45 ROLLBACK to SP1;
```

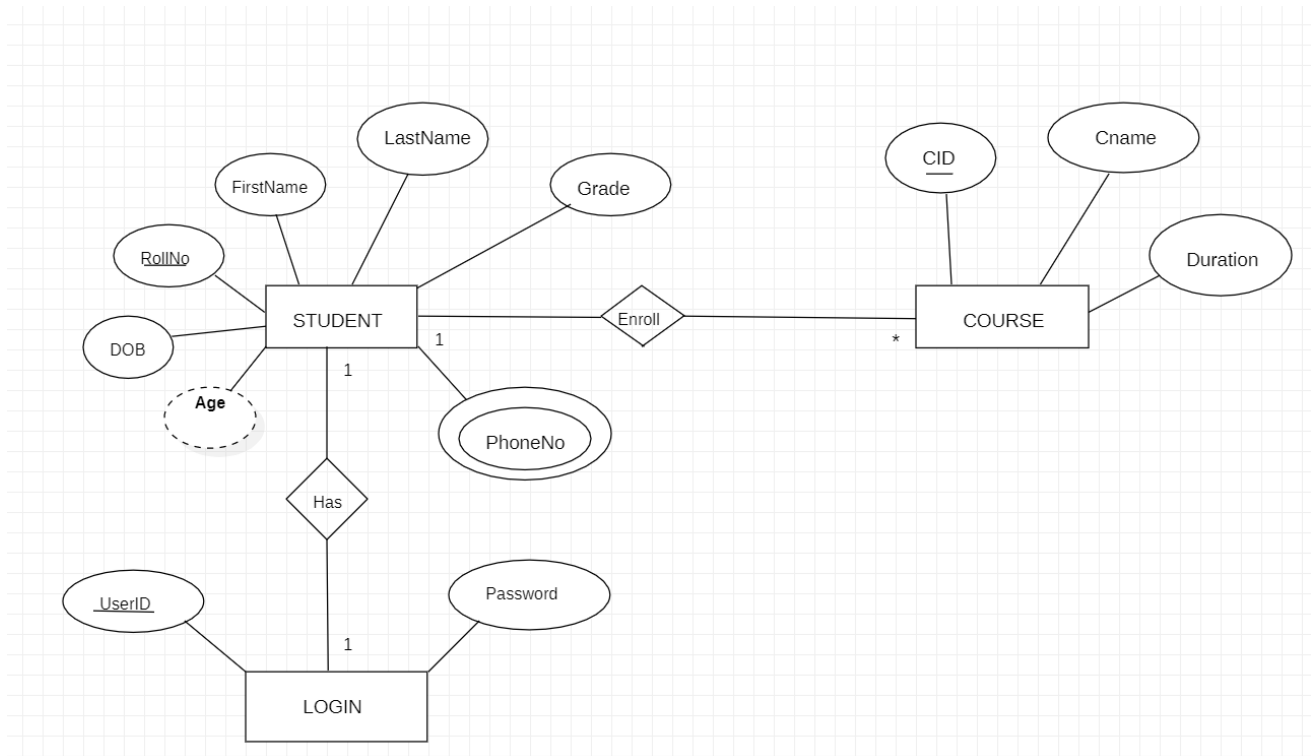
**RESULT:** FROM THE ABOVE EXPERIMENT, we observed some of the Build-In Commands in sql.



## 8. ER Diagrams in Data Base Management System

04.05.22

**Aim:** Some important ER diagrams in Data Base Management System.



**RESULT:** ER Diagram for Student Database Management System.

## 9. SQL SUB QUERIES/NESTED-QUERIES

08.05.22

**Aim:** Some important queries commands in SQL.

**Description:**

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that subqueries must follow –

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.

### i) Subqueries with the **SELECT** Statement

- Subqueries are most frequently used with the SELECT statement.
- `SELECT column_name [, column_name ] FROM table1 [, table2 ] WHERE column_name OPERATOR (SELECT column_name [, column_name ] FROM table1 [, table2 ] [WHERE]);`

### ii) Subqueries with the **INSERT** Statement

Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date or number functions.

The basic syntax is as follows.

```
INSERT INTO table_name [ (column1 [, column2 ]) ] SELECT [ *|column1 [, column2 ] FROM table1 [, table2 ] [ WHERE VALUE OPERATOR ];
```

### iii) Subqueries with the **UPDATE** Statement

The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.

The basic syntax is as follows.

```
UPDATE table SET column_name = new_value [ WHERE OPERATOR [ VALUE ] (SELECT COLUMN_NAME FROM TABLE_NAME) [ WHERE) ];
```

#### iv) Subqueries with the **DELETE** Statement

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

The basic syntax is as follows.

```
DELETE FROM TABLE_NAME [ WHERE OPERATOR [ VALUE ] (SELECT COLUMN_NAME FROM TABLE_NAME) [ WHERE) ];
```

### OUTPUT:

```
-- SQL SUBQUERIES
-- i) SELECT command in subqueries
select * from SRM_STUDENTS where STUD_ID = (select max(STUD_ID) from SRM_STUDENTS );
select * from SRM_STUDENTS where STUD_ID < (select max(STUD_ID) from SRM_STUDENTS );
select * from SRM_STUDENTS where STUD_ID > (select max(STUD_ID) from SRM_STUDENTS );
-- ii) Insert command in subqueries
create table Ai_A(Ai_A_ID int,Ai_A_NAME varchar(30),Ai_A_DEPT varchar(20),Ai_A_YEAR number);
insert into Ai_A values (105,"Shruthi","IT",3);
insert into Ai_A values (108,"Nishtha","AI",3);
insert into Ai_A values (124,"Saifeen","CSE",4);
insert into Ai_A values (153,"Leela","ECE",2);

select * from Ai_A;

create table Ai_B(Ai_B_ID int,Ai_B_NAME varchar(30),Ai_B_DEPT varchar(20),Ai_B_YEAR number);
insert into Ai_B select * from Ai_A where Ai_A_ID in (select Ai_A_ID from Ai_A where Ai_A_YEAR >2);
select * from Ai_B;
-- Delete subquery
select * from Ai_B;
delete from Ai_B where Ai_B_YEAR in (select Ai_B_YEAR from Ai_B where Ai_B_YEAR <5);
```

**RESULT:** FROM THE ABOVE EXPERIMENT, we observed some of the subqueries in sql.

## 10. SQL JOINS

03.06.22

**Aim:** Some important Joins commands in SQL.

### **Description:**

Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal to symbol.

There are different types of joins available in SQL –

- INNER JOIN – returns rows when there is a match in both tables.
- LEFT JOIN – returns all rows from the left table, even if there are no matches in the right table.
- RIGHT JOIN – returns all rows from the right table, even if there are no matches in the left table.
- FULL JOIN – returns rows when there is a match in one of the tables.
- SELF JOIN – is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- CARTESIAN JOIN – returns the Cartesian product of the sets of records from the two or more joined tables.

### **1.INNER JOIN:**

INNER JOIN Syntax

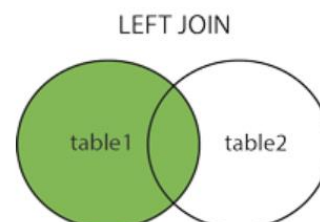
```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```



### **2. LEFT JOIN:**

LEFT JOIN Syntax

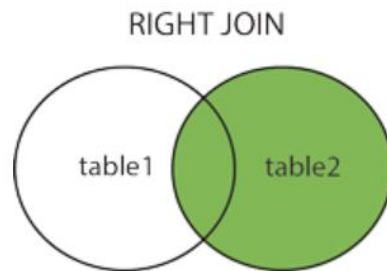
```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```



### 3.RIGHT JOIN:

#### RIGHT JOIN Syntax

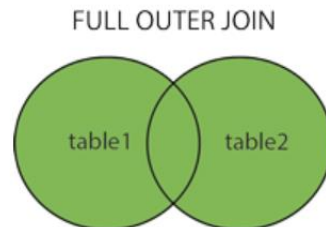
```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```



### 4.FULL JOIN:

#### FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```



### OUTPUT:

```
123 -- JOINS IN SQL
124 create table cricket(cricket_id int auto_increment,name varchar(20),primary key(cricket_id));
125 create table football(football_id int auto_increment,name varchar(20),primary key(football_id));
126 insert into cricket(name) values('stewart'),('michael'),('Johnson'),('Warner');
127 select * from cricket;
128 insert into football(name) values('kohli'),('rohith'),('Johnson'),('Warner');
129 select * from football;
130 -- The people who are part of both game??
131 select * from cricket as c inner join football as f on c.name = f.name;
132 -- we can select individual columns
133 select c.cricket_id ,c.name,f.football_id,f.name from cricket as c inner join football as f on c.name = f.name;
134 select c.cricket_id ,c.name,f.football_id,f.name from cricket as c left join football as f on c.name = f.name;
135 select c.cricket_id ,c.name,f.football_id,f.name from cricket as c right join football as f on c.name = f.name;
136
137
```

**RESULT:** From the above experiment, we studied sql joins.

## 11. SQL Triggers

23.06.22

**Aim:** To create and execute the Triggers

Triggers are 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).

**Syntax:**

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{ BEFORE | AFTER | INSTEAD OF }
{ INSERT [OR] | UPDATE [OR] | DELETE }
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
    Declaration-statements
BEGIN
    Executable-statements
EXCEPTION
    Exception-handling-statements
END;
```

## Code:

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
    sal_diff number;
BEGIN
    sal_diff := :NEW.salary - :OLD.salary;
    dbms_output.put_line('Old salary: ' || :OLD.salary);
    dbms_output.put_line('New salary: ' || :NEW.salary);
    dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

## Output:

```
133 CREATE OR REPLACE TRIGGER display_salary_changes
134 BEFORE DELETE OR INSERT OR UPDATE ON customers
135 FOR EACH ROW
136 WHEN (NEW.ID > 0)
137 DECLARE
138     sal_diff number;
139 BEGIN
140     sal_diff := :NEW.salary - :OLD.salary;
141     dbms_output.put_line('Old salary: ' || :OLD.salary);
142     dbms_output.put_line('New salary: ' || :NEW.salary);
143     dbms_output.put_line('Salary difference: ' || sal_diff);
144 END;
145 /

146
147 DECLARE
148     total_rows number(2);
149 BEGIN
150     UPDATE customers
151     SET salary = salary + 5000;
152     IF sql%notfound THEN
153         dbms_output.put_line('no customers updated');
154     ELSIF sql%found THEN
155         total_rows := sql%rowcount;
156         dbms_output.put_line( total_rows || ' customers updated ');
157     END IF;
158 END;
159 /
```

**Result:** From the above experiment we studied and executed TRIGGERS in sql successfully.

## 12. SQL EXCEPTION HANDLING

30.06.22

**Aim:** Some important Exception Handling commands in SQL.

### **Description:**

An exception is an error which disrupts the normal flow of program instructions. PL/SQL provides us the exception block which raises the exception thus helping the programmer to find out the fault and resolve it. There are two types of exceptions defined in PL/SQL

1. User defined exception.
2. System defined exceptions

**1.User defined exceptions:** This type of users can create their own exceptions according to the need and to raise these exceptions explicitly *raise* command is used.

```
-- Exception Handling in sql
-- i) user defined exceptions:
DECLARE
    x int:=&x; /*taking value at run time*/
    y int:=&y;
    div_r float;
    exp1 EXCEPTION;
    exp2 EXCEPTION;

BEGIN
    IF y=0 then
        raise exp1;
    ELSE IF y > x then
```



```

        raise exp2;
ELSE
    div_r:= x / y;
    dbms_output.put_line('the result is '||div_r);
END IF;
EXCEPTION
WHEN exp1 THEN
    dbms_output.put_line('Error');
    dbms_output.put_line('division by zero not allowed');

WHEN exp2 THEN
    dbms_output.put_line('Error');
    dbms_output.put_line('y is greater than x please check the input');
END;

```

**2. System defined exceptions:** These exceptions are predefined in PL/SQL which get raised WHEN certain **database rule is violated**.

System-defined exceptions are further divided into two categories:

1. Named system exceptions.
2. Unnamed system exceptions.

**1.NO\_DATA\_FOUND:** It is raised WHEN a SELECT INTO statement returns *no* rows. For eg:

```

BEGIN
    SELECT g_id into temp from geeks where g_name='Tharun';

exception
    WHEN no_data_found THEN
        dbms_output.put_line('ERROR');
        dbms_output.put_line('there is no name as');
        dbms_output.put_line('Tharun is not in table');
end;

```

**2. VALUE\_ERROR:** This error is raised WHEN a statement is executed that resulted in an arithmetic, numeric, string, conversion, or constraint error. This error mainly results from programmer error or invalid data input.

```

DECLARE
    temp number;

BEGIN
    SELECT g_name into temp from SRM_STUDENTS where STUD_NAME='Ajay';
    dbms_output.put_line('the stud_name is ' || temp);

EXCEPTION
    WHEN value_error THEN
        dbms_output.put_line('Error');
        dbms_output.put_line('Change data type of temp to varchar(20)');

END;

```

### 3. ZERO\_DIVIDE = raises exception WHEN dividing with zero

```

DECLARE
    a int:=10;
    b int:=0;
    answer int;

BEGIN
    answer:=a/b;
    dbms_output.put_line('the result after division is' || answer);

exception
    WHEN zero_divide THEN
        dbms_output.put_line('dividing by zero please check the values again');
        dbms_output.put_line('the value of a is ' || a);
        dbms_output.put_line('the value of b is ' || b);

END;

```

**Result:** From the above experiment we studied and executed Exception Handling in sql successfully.

## Conclusion

Course management system for Student Database were developed by using SQL. Student Management System can be used by educational institutions to maintain their student records easily. Achieving this objective is difficult using the manual system as the information is scattered, can be redundant, and collecting relevant information may be very time-consuming. All these problems are solved by this project.