AISSMS

COLLEGE OF ENGINEERING

Approved by AICTE, New Delhi, Recognized by
Govt. of Maharashtra, Affiliated to Savitribai Phule Pune University
and recognized 2(f) and 12(B) by UGC (Id.No. PU / PN/ Engg. / 093 (1992)
Accredited by NAAC with 'A+' Grade

DEPARTMENT OF COMPUTER ENGINEERING

YEAR 2021-22

TERM - 5

PRACTICAL ASSIGNMENT

DBMS LABORATORY

CLASS - TE COMPUTER ENGG.(SHIFT A)

NAME - PRATIK PINGALE

ROLL NO. - 19C0056

BATCH - C

FACULTY – SONALI NALAMWAR

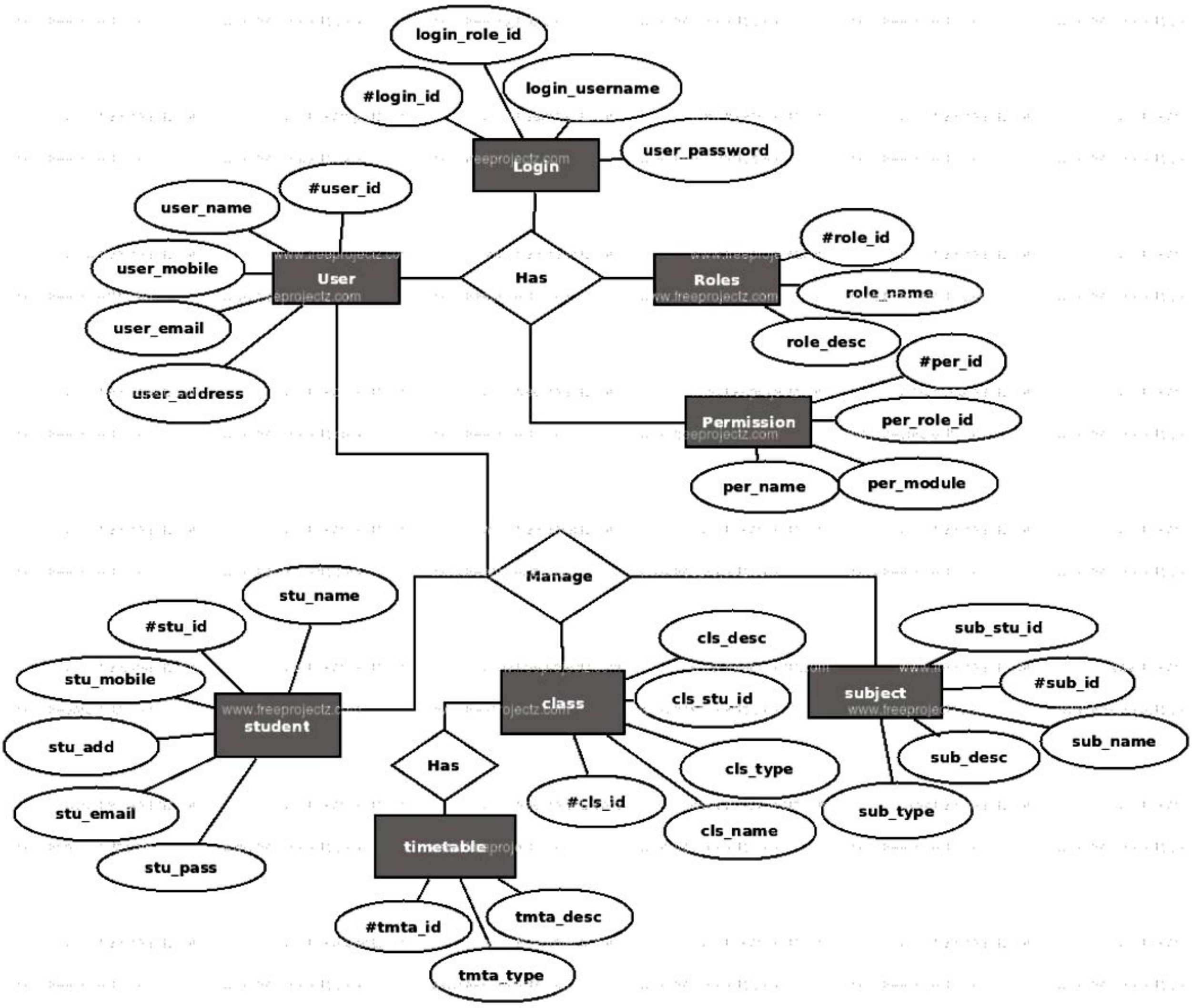
Pratik Pingale 19COD056

DBMSL ASSIGNMENT -1

AIM : Decide a case study and formulate / prepose a conceptual design using ER Diagram.

Connert ER diagrams into relational table. and normalize Relational data model.

Relational



ER Diagram For Timetable Management System

Conclusion: Thus from this experiment we learned to create a design of a database using ER diagram and created a relational table from it.

Pratik Pingale 19CODS6

PAGE :
DATE : / /

DBMSL ASSIGNMENT - 2

AIM : Design SQL DDL commands which uses SQL objects such as Table, View, Index, Sequence Synonym, etc

THEORY :

SQL Commands can be divided in 3 types:

1) Data defination language (DDL)
2) Data Manipulation Language (DML)
3) Data Control Language (DCL)

a) Data Definition Language (DDL)
Data Defination Language contains the commands which allows user to define data & their relationship to other type of data. It is used to design or create data table, dictionary & files within database.

* Commands used in DDL.

1) CREATE :
To create the database instance.

2) ALTER :
To alter structure of database

3) RENAME :
To rename the database instance

PAGE :
DATE : / /

4) DROP :

Drop table table_name

5) INSERT :

Insert into table_name [clm_names ...]
values (values ...).

6) SELECT :

Selects [column_names ...] from table
table_name [where ...]

7) UPDATE :

update table_name set column_name
new_column [where condition].

8) VIEW :

create view customer as
select C_name, city
from customer_table

9) INDEX :

create index indx_name
on Persons (last_name);

PAGE :
DATE : / /

* Part - 2.

1) Set operation :-

a) union : returns all distinct rows selected by either query.

b) unionall : returns all selected rows either query including duplicates.

c) intersect : returns those rows which are common to both queries.

d) minus : returns rows which are in 1st query and not in 2nd including common

2) SQL Operators :

a) Arithmetic operators : +, −, /, *, %

b) Comparison operators : >, <, >=, <=, ==, !=

c) Logical operators : AND, OR, NOT, NOR.

d) Bitwise operators : &, |, ^, !

3) Aggregate function :

Performs a calculation on a set of values and return a single value

a) Min() :- smallest value of a column.

b) Max() : returns largest of given column

c) Sum() : returns sum of all the values in the column.

d) Avg() : returns average of all the value from the column.

PAGE :
DATE : / /

CONCLUSION : In this experiment we learned about basic SQL DML, DCL, DDL Queries. along with writing basic PL/SQL blocks. and using them in practical usage in application.

```
┌─proxzima@proxzima in ~ on ☁  pratik@convin.ai took 51ms [▮ 100%]
└─☣ mycli -d proxzima

(none) > use dbmsl;
You are now connected to database "dbmsl" as user "proxzima"
Time: 0.001s

dbmsl > /* PART : 1
     → 1> DDL - Data Definition Language
     → 2> DML - Data Manipulation Language
     → 3> DQl - Data Query Language */;
Query OK, 0 rows affected
Time: 0.001s

dbmsl > CREATE TABLE library (ISBN VARCHAR(50) PRIMARY KEY, name VARCHAR(255), author VARCHAR(255), publis
     → her VARCHAR(255), yearOfPublication INTEGER, cost FLOAT NOT NULL );
Query OK, 0 rows affected
Time: 0.014s

dbmsl > INSERT INTO library VALUES ('9874133548', 'Only Time Will Tell', 'Jeffery Archer', 'PAN', 2011, 56
     → 2.45), ('9784157165', 'Midnights Children', 'Salman Rushdie', 'VINTAGE', 2013, 452.25), ('94571548
     → 12', 'Revolution 2020', 'Jeffery Archer', 'RUPA', 2011, 842.75);
Query OK, 3 rows affected
Time: 0.005s

dbmsl > INSERT INTO library VALUES ('9421548465', 'The Old Man And His God', 'Sudha Murty', 'PENGUIN', 200
     → 6, 170.45);
Query OK, 1 row affected
Time: 0.004s

dbmsl > SELECT * from library;
```

ISBN	name	author	publisher	yearOfPublication	cost
9421548465	The Old Man And His God	Sudha Murty	PENGUIN	2006	170.45
9457154812	Revolution 2020	Jeffery Archer	RUPA	2011	842.75
9784157165	Midnights Children	Salman Rushdie	VINTAGE	2013	452.25
9874133548	Only Time Will Tell	Jeffery Archer	PAN	2011	562.45

```
4 rows in set
Time: 0.015s

dbmsl > ALTER TABLE library ADD Department VARCHAR(255);
You're about to run a destructive command.
Do you want to proceed? (y/n): y
Your call!
Query OK, 0 rows affected
Time: 0.010s

dbmsl > SELECT * from library;
```

ISBN	name	author	publisher	yearOfPublication	cost	Department
9421548465	The Old Man And His God	Sudha Murty	PENGUIN	2006	170.45	<null>
9457154812	Revolution 2020	Jeffery Archer	RUPA	2011	842.75	<null>
9784157165	Midnights Children	Salman Rushdie	VINTAGE	2013	452.25	<null>
9874133548	Only Time Will Tell	Jeffery Archer	PAN	2011	562.45	<null>

```
4 rows in set
Time: 0.009s

dbmsl > ALTER TABLE library DROP COLUMN Department;
You're about to run a destructive command.
Do you want to proceed? (y/n): y
Your call!
Query OK, 0 rows affected
Time: 0.023s

dbmsl > SELECT * from library;
```

ISBN	name	author	publisher	yearOfPublication	cost
9421548465	The Old Man And His God	Sudha Murty	PENGUIN	2006	170.45
9457154812	Revolution 2020	Jeffery Archer	RUPA	2011	842.75

9784157165	Midnights Children	Salman Rushdie	VINTAGE	2013	452.25
9874133548	Only Time Will Tell	Jeffery Archer	PAN	2011	562.45

4 rows in set
Time: 0.010s

```
dbmsl > UPDATE library SET cost=325.25 where cost=452.25;
Query OK, 1 row affected
Time: 0.003s

dbmsl > SELECT distinct cost from library;
```

cost
170.45
842.75
325.25
562.45

4 rows in set
Time: 0.009s

```
dbmsl > SELECT * FROM library ORDER BY name;
```

ISBN	name	author	publisher	yearOfPublication	cost
9784157165	Midnights Children	Salman Rushdie	VINTAGE	2013	325.25
9874133548	Only Time Will Tell	Jeffery Archer	PAN	2011	562.45
9457154812	Revolution 2020	Jeffery Archer	RUPA	2011	842.75
9421548465	The Old Man And His God	Sudha Murty	PENGUIN	2006	170.45

4 rows in set
Time: 0.010s

```
dbmsl > SELECT * FROM library ORDER BY name DESC;
```

ISBN	name	author	publisher	yearOfPublication	cost
9421548465	The Old Man And His God	Sudha Murty	PENGUIN	2006	170.45
9457154812	Revolution 2020	Jeffery Archer	RUPA	2011	842.75
9874133548	Only Time Will Tell	Jeffery Archer	PAN	2011	562.45
9784157165	Midnights Children	Salman Rushdie	VINTAGE	2013	325.25

4 rows in set
Time: 0.011s

```
dbmsl > SELECT * FROM library WHERE name LIKE 'Onl%';
```

ISBN	name	author	publisher	yearOfPublication	cost
9874133548	Only Time Will Tell	Jeffery Archer	PAN	2011	562.45

1 row in set
Time: 0.009s

```
dbmsl > UPDATE library SET author='Chetan Bhagat' where name='Revolution 2020';
Query OK, 1 row affected
Time: 0.003s

dbmsl > SELECT * FROM library;
```

ISBN	name	author	publisher	yearOfPublication	cost
9421548465	The Old Man And His God	Sudha Murty	PENGUIN	2006	170.45
9457154812	Revolution 2020	Chetan Bhagat	RUPA	2011	842.75
9784157165	Midnights Children	Salman Rushdie	VINTAGE	2013	325.25
9874133548	Only Time Will Tell	Jeffery Archer	PAN	2011	562.45

4 rows in set
Time: 0.010s

```
dbmsl > SELECT * FROM library WHERE cost BETWEEN 200 and 700;
```

ISBN	name	author	publisher	yearOfPublication	cost
9784157165	Midnights Children	Salman Rushdie	VINTAGE	2013	325.25
9874133548	Only Time Will Tell	Jeffery Archer	PAN	2011	562.45

2 rows *in* set
Time: 0.009s

```
dbmsl > /* PART : 2
     → 1> Set Operations
     → 2> SQL Operators
     → 3> Aggeregate Functions */;
Query OK, 0 rows affected
Time: 0.001s

dbmsl > CREATE TABLE employee (e_no int NOT NULL primary key comment 'primary key', e_name VARCHAR(255) UNIQUE CO
     → MMENT 'employee name', address VARCHAR(255) COMMENT 'employee address', basic_salary FLOAT NOT NULL comme
     → nt 'employee salary', job_status VARCHAR(255) COMMENT 'job status') default charset utf8 comment '';
Query OK, 0 rows affected
Time: 0.016s

dbmsl > INSERT INTO employee VALUES (5, 'A. Ghosh', 'Kharagpur', 4200.00, 'Professor'), (10, 'G. Bhakta', 'Midnap
     → ur', 2700.00, 'Research fellow'), (1, 'P. Sen', 'Calcutta', 5000.00, 'Professor'), (7, 'D. Kundu', 'Khara
     → gpur', 8000.00, 'Director'), (3, 'S. Prasad', 'Calcutta', 5300.00, 'Professor'), (4, 'P. Gupta', 'Midnapu
     → r', 3000.00, 'Research fellow'), (11, 'G.S. Bose', 'Kharagpur', 2000.00, 'Office Asst'), (12, 'L. sen', '
     → Calcutta', 1500.00, 'Office Asst'), (13, 'K. Singh', 'Midnapur', 1700.00, 'Office Asst');
Query OK, 9 rows affected
Time: 0.005s

dbmsl > SELECT * FROM employee;
```

e_no	e_name	address	basic_salary	job_status
1	P. Sen	Calcutta	5000.0	Professor
3	S. Prasad	Calcutta	5300.0	Professor
4	P. Gupta	Midnapur	3000.0	Research fellow
5	A. Ghosh	Kharagpur	4200.0	Professor
7	D. Kundu	Kharagpur	8000.0	Director
10	G. Bhakta	Midnapur	2700.0	Research fellow
11	G.S. Bose	Kharagpur	2000.0	Office Asst
12	L. sen	Calcutta	1500.0	Office Asst
13	K. Singh	Midnapur	1700.0	Office Asst

```
9 rows in set
Time: 0.004s

dbmsl > CREATE TABLE project (p_no VARCHAR(12) NOT NULL primary key comment 'primary key', p_name VARCHAR(255) CO
     → MMENT 'project name', nos_of_staff INTEGER COMMENT 'number of staff' ) default charset utf8 comment '';
Query OK, 0 rows affected
Time: 0.022s

dbmsl > INSERT INTO project VALUES ('CS110', 'DBMS', 4), ('CS220', 'DDBMS', 3), ('MS005', 'MIS', 5), ('MS001', 'C
     → AD/VLSI', 4);
Query OK, 4 rows affected
Time: 0.004s

dbmsl > SELECT * FROM project;
```

p_no	p_name	nos_of_staff
CS110	DBMS	4
CS220	DDBMS	3
MS001	CAD/VLSI	4
MS005	MIS	5

```
4 rows in set
Time: 0.008s

dbmsl > SELECT e_name FROM employee WHERE basic_salary > 4000;
```

e_name
P. Sen
S. Prasad

A. Ghosh
D. Kundu

4 rows *in* set
Time: 0.008s

```
dbmsl > SELECT e_name FROM employee WHERE basic_salary > 4000 AND job_status = 'Professor';
```

e_name
P. Sen
S. Prasad
A. Ghosh

3 rows *in* set
Time: 0.008s

```
dbmsl > SELECT e_name, e_no FROM employee ORDER BY e_no DESC;
```

e_name	e_no
K. Singh	13
L. sen	12
G.S. Bose	11
G. Bhakta	10
D. Kundu	7
A. Ghosh	5
P. Gupta	4
S. Prasad	3
P. Sen	1

9 rows *in* set
Time: 0.009s

```
dbmsl > SELECT e_name FROM employee WHERE job_status LIKE 'Research%';
```

e_name
P. Gupta
G. Bhakta

2 rows *in* set
Time: 0.007s

```
dbmsl > SELECT e_name FROM employee WHERE MOD(e_no, 2) = 1;
```

e_name
A. Ghosh
D. Kundu
G.S. Bose
K. Singh
P. Sen
S. Prasad

6 rows *in* set
Time: 0.008s

```
dbmsl > SELECT SUM(basic_salary) AS "Total Salary" FROM employee;
```

Total Salary
33400.0

1 row *in* set
Time: 0.007s

```
dbmsl > SELECT SUM(basic_salary) AS "Total Salary" FROM employee WHERE job_status = 'Office Asst';
```

Total Salary
5200.0

```
1 row in set
Time: 0.007s

dbmsl > SELECT job_status, COUNT(*) As "No of employees" FROM employee GROUP BY job_status;
```

job_status	No of employees
Professor	3
Research fellow	2
Director	1
Office Asst	3

```
4 rows in set
Time: 0.008s

dbmsl > SELECT AVG(basic_salary) As "Average Salary" FROM employee WHERE job_status in ('Professor', 'Research fe
     → llow');
```

Average Salary
4040.0

```
1 row in set
Time: 0.008s

dbmsl > SELECT e_name FROM employee WHERE job_status in (SELECT job_status FROM employee WHERE e_name = 'P. Gupta
     → ');
```

e_name
P. Gupta
G. Bhakta

```
2 rows in set
Time: 0.010s

dbmsl >
```

Pratik Pingale 19CO056

PAGE :
DATE : / /

DBMSL ASSIGNMENT - 3

AIM : Use SQL DML queries to perform Join, sub-Query, views on a database.

THEORY :

1) Join :

A join is meant for combining columns from one table or one or more tables by using values common to each.

2) Inner Join :

Displays value that are common in both tables.

3) Left Join :

Displays all rows from left table and mating values from right table.

4) Right Join :

Displays all rows from right table and matching values from left side.

5) Full Join :

Displays combined result of both left and right join.

→ Select student.name, stud-cource.cource
from student
[inner | left | right full] joind stud-cource
on student.id = stud-cource.student.rollno.

PAGE :
DATE : / /

★ Sub Query
- Query within another query
- select name, location from database
where Roll-no in
(select Roll-no from student
where section = 'A');

★ View
- Virtual table based on the result set of an SQL statement
- create view product as
select p-name, price
from products
where Price > (select Avg(Price) from product);

PAGE :
DATE : / /

CONCLUSION : In this experiment we learned about basic SQL DML, DCL, DDL Queries. along with writing basic PL/SQL blocks. and using them in practical usage in application.

```
┌─proxzima@proxzima in ~ on ☁  pratik@convin.ai took 2ms [🔋 100%]
└─☣ mycli -d proxzima

(none) > USE dbmsl;
You are now connected to database "dbmsl" as user "proxzima"
Time: 0.000s

dbmsl > CREATE TABLE dept(deptno int NOT NULL primary key, dname VARCHAR(20) NOT NULL, location VARCHAR(20) NOT N
     → ULL ) default charset utf8 comment '';
Query OK, 0 rows affected
Time: 0.012s

dbmsl > INSERT INTO dept VALUES (1, 'HR', 'Mumbai'), (2, 'Management', 'Mumbai'), (3, 'Core', 'Pune');
Query OK, 3 rows affected
Time: 0.004s

dbmsl > SELECT * FROM dept;
```

deptno	dname	location
1	HR	Mumbai
2	Management	Mumbai
3	Core	Pune

```
3 rows in set
Time: 0.005s

dbmsl > CREATE TABLE emp(empno int NOT NULL primary key, ename VARCHAR(20) NOT NULL, job VARCHAR(20) NOT NULL, hi
     → redate DATETIME NOT NULL, sal int, deptno int) default charset utf8 comment '';
Query OK, 0 rows affected
Time: 0.012s

dbmsl > ALTER TABLE emp ADD FOREIGN KEY (deptno) REFERENCES dept(deptno);
You're about to run a destructive command.
Do you want to proceed? (y/n): y
Your call!
Query OK, 0 rows affected
Time: 0.024s

dbmsl > DESC emp;
```

Field	Type	Null	Key	Default	Extra
empno	int	NO	PRI	<null>	
ename	varchar(20)	NO		<null>	
job	varchar(20)	NO		<null>	
hiredate	datetime	NO		<null>	
sal	int	YES		<null>	
deptno	int	YES	MUL	<null>	

```
6 rows in set
Time: 0.006s

dbmsl > ALTER Table emp ADD mobileno int;
You're about to run a destructive command.
Do you want to proceed? (y/n): y
Your call!
Query OK, 0 rows affected
Time: 0.012s

dbmsl > INSERT INTO emp VALUES (1, 'Alex', 'Manager', '2010-04-11', 10000, 1, 99842744), (2, 'Bruce', 'Employee',
     →  '2012-06-12', 5500, 2, 546486156), (3, 'Alpha', 'Employee', '2016-06-23', 7500, 3, 18645184), (5, 'Loro'
     → , 'Manager', '2019-09-30', 4000, 2, 875138453);
Query OK, 4 rows affected
Time: 0.003s

dbmsl > INSERT INTO emp VALUES (7369, 'Sam', 'Employee', '2012-06-15', 60000, 1, 55421666);
Query OK, 1 row affected
Time: 0.003s

dbmsl > SELECT * FROM emp;
```

empno	ename	job	hiredate	sal	deptno	mobileno
1	Alex	Manager	2010-04-11 00:00:00	10000	1	99842744

2	Bruce	Employee	2012-06-12 00:00:00	5500	2	546486156
3	Alpha	Employee	2016-06-23 00:00:00	7500	3	18645184
5	Loro	Manager	2019-09-30 00:00:00	4000	2	875138453
7369	Sam	Employee	2012-06-15 00:00:00	60000	1	55421666

5 rows in set
Time: 0.010s

```
dbmsl > DELETE FROM emp WHERE empno = 7369;
You're about to run a destructive command.
Do you want to proceed? (y/n): y
Your call!
Query OK, 1 row affected
Time: 0.004s
```

```
dbmsl > SELECT * FROM emp;
```

empno	ename	job	hiredate	sal	deptno	mobileno
1	Alex	Manager	2010-04-11 00:00:00	10000	1	99842744
2	Bruce	Employee	2012-06-12 00:00:00	5500	2	546486156
3	Alpha	Employee	2016-06-23 00:00:00	7500	3	18645184
5	Loro	Manager	2019-09-30 00:00:00	4000	2	875138453

4 rows in set
Time: 0.010s

```
dbmsl > SELECT * from emp join dept WHERE emp.deptno=dept.deptno;
```

empno	ename	job	hiredate	sal	deptno	mobileno	deptno	dname	location
1	Alex	Manager	2010-04-11 00:00:00	10000	1	99842744	1	HR	Mumbai
2	Bruce	Employee	2012-06-12 00:00:00	5500	2	546486156	2	Management	Mumbai
5	Loro	Manager	2019-09-30 00:00:00	4000	2	875138453	2	Management	Mumbai
3	Alpha	Employee	2016-06-23 00:00:00	7500	3	18645184	3	Core	Pune

4 rows in set
Time: 0.012s

```
dbmsl > SELECT * from emp inner join dept on emp.deptno=dept.deptno;
```

empno	ename	job	hiredate	sal	deptno	mobileno	deptno	dname	location
1	Alex	Manager	2010-04-11 00:00:00	10000	1	99842744	1	HR	Mumbai
2	Bruce	Employee	2012-06-12 00:00:00	5500	2	546486156	2	Management	Mumbai
5	Loro	Manager	2019-09-30 00:00:00	4000	2	875138453	2	Management	Mumbai
3	Alpha	Employee	2016-06-23 00:00:00	7500	3	18645184	3	Core	Pune

4 rows in set
Time: 0.011s

```
dbmsl > SELECT * from emp left join dept on emp.deptno=dept.deptno;
```

empno	ename	job	hiredate	sal	deptno	mobileno	deptno	dname	location
1	Alex	Manager	2010-04-11 00:00:00	10000	1	99842744	1	HR	Mumbai
2	Bruce	Employee	2012-06-12 00:00:00	5500	2	546486156	2	Management	Mumbai
3	Alpha	Employee	2016-06-23 00:00:00	7500	3	18645184	3	Core	Pune
5	Loro	Manager	2019-09-30 00:00:00	4000	2	875138453	2	Management	Mumbai

4 rows in set
Time: 0.011s

```
dbmsl > SELECT * from emp right join dept on emp.deptno=dept.deptno;
```

empno	ename	job	hiredate	sal	deptno	mobileno	deptno	dname	location
1	Alex	Manager	2010-04-11 00:00:00	10000	1	99842744	1	HR	Mumbai
2	Bruce	Employee	2012-06-12 00:00:00	5500	2	546486156	2	Management	Mumbai

2	Bruce	Employee	2012-06-12 00:00:00	9500	2	546486156	2	Management	Mumbai
5	Loro	Manager	2019-09-30 00:00:00	4000	2	875138453	2	Management	Mumbai
3	Alpha	Employee	2016-06-23 00:00:00	7500	3	18645184	3	Core	Pune

4 rows *in* set
Time: 0.011s

```
dbmsl > CREATE VIEW Details AS SELECT emp.empno, emp.ename, emp.hiredate, dept.dname, dept.location FROM emp, dep
     → t WHERE emp.deptno=dept.deptno;
Query OK, 0 rows affected
Time: 0.006s

dbmsl > SELECT * FROM `Details`;
```

empno	ename	hiredate	dname	location
1	Alex	2010-04-11 00:00:00	HR	Mumbai
2	Bruce	2012-06-12 00:00:00	Management	Mumbai
5	Loro	2019-09-30 00:00:00	Management	Mumbai
3	Alpha	2016-06-23 00:00:00	Core	Pune

4 rows *in* set
Time: 0.009s

```
dbmsl > CREATE OR REPLACE VIEW Details AS SELECT emp.empno, emp.ename, emp.hiredate, emp.mobileno, dept.dname, de
     → pt.location FROM emp, dept WHERE emp.deptno=dept.deptno;
Query OK, 0 rows affected
Time: 0.005s

dbmsl > SELECT * FROM `Details`;
```

empno	ename	hiredate	mobileno	dname	location
1	Alex	2010-04-11 00:00:00	99842744	HR	Mumbai
2	Bruce	2012-06-12 00:00:00	546486156	Management	Mumbai
5	Loro	2019-09-30 00:00:00	875138453	Management	Mumbai
3	Alpha	2016-06-23 00:00:00	18645184	Core	Pune

```
4 rows in set
Time: 0.009s

dbmsl > DROP VIEW `Details`;
You're about to run a destructive command.
Do you want to proceed? (y/n): y
Your call!
Query OK, 0 rows affected
Time: 0.004s
```

Pratik Pingale 19CO056

PAGE :
DATE : / /

DBMSL ASSIGNMENT- 4

AIM : Use of control structure and exception handeling in PL/SQL

1) Borrower (Roll_no, Name, issue, book_name, status,
2) Fine (Roll_no, Date, Amount).
   - check issue dates
   - Fine according to days.
   - Changing status after submission.
   - Use exception handeling.

THEORY :

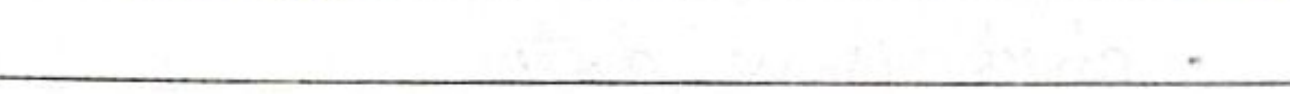
- PL/SQL is combination of SQL along with procedure feature of programming language.
- Basic Structure of PLSQL which is block structured language this means the PL/SQL program are divided and written in logical block of code.
- Each block consist of three subparts.

* Basic Structure.

DECLARE < declaration Section>
BEGIN < executable commands>
EXCEPTION < exception handeling>
END.

PAGE :
DATE : / /

PL/SQL block types



1) Anonymous block:
- These PL/SQL blocks don't have names assigned to them.
- Need to be used in same section as they are stored as database objects.
- No complicated setup steps are required as not stored in database.

2) Normal block:
- These PL/SQL blocks do have unique names assigned to them.
- They are stored as database object in server
- They can be reffered or used outside a given section.

★ Exception block.

- Exeptions are errors occuring during program execution in PL/SQL
- To catch such errors, exception block is provided.

- There are 2 types of exception :

1) System - defined exception :
   These are pre-defined exceptions in PL/SQL

2) User - defined exception :
   PL/SQL allows used to create their own exceptions which can be raised explicitly

PAGE :
DATE : / /

CONCLUSION : In this experiment we learned about basic SQL DML, DCL, DDL Queries. along with writing basic PL/SQL blocks. and using them in practical usage in application.

```
┌─proxzima@proxzima in ~ on ☁  pratik@convin.ai took 1h54m1s [🔋 38%]
└─$ mycli -d proxzima

(none) > USE dbmsl;
You are now connected to database "dbmsl" as user "proxzima"
Time: 0.001s

dbmsl > CREATE TABLE Borrower(Roll_no INT PRIMARY KEY NOT NULL, Name VARCHAR(20), DateofIssue DATE, NameofBook VARC
      → HAR(50), Status VARCHAR(10));
Query OK, 0 rows affected
Time: 0.016s

dbmsl > INSERT INTO Borrower VALUES (1, 'Pratik', '2021-11-01', 'Revolution 2020', 'I'), (2, 'Alex', '2021-10-12',
      → 'Midnights Children', 'I'), (3, 'Jhon', '2021-09-03', 'Only Time Will Tell', 'I');
Query OK, 3 rows affected
Time: 0.006s

dbmsl > SELECT * FROM Borrower;
```

Roll_no	Name	DateofIssue	NameofBook	Status
1	Pratik	2021-11-01	Revolution 2020	I
2	Alex	2021-10-12	Midnights Children	I
3	Jhon	2021-09-03	Only Time Will Tell	I

```
3 rows in set
Time: 0.004s

dbmsl > CREATE TABLE Fine (Roll_no INT PRIMARY KEY NOT NULL, Date DATE, Amt INT);
Query OK, 0 rows affected
Time: 0.014s

dbmsl > DROP PROCEDURE IF EXISTS LIBRARY;
      → DELIMITER $$
      → CREATE PROCEDURE LIBRARY(
      →     roll int,
      →     name_of_book varchar(50)
      → )
      → BEGIN
      →     DECLARE diff_date INT;
      →     DECLARE p_date DATE;
      →     DECLARE i_date DATE;
      →     DECLARE fine_amt FLOAT(5);
      →
      →     DECLARE EXIT HANDLER FOR SQLEXCEPTION
      →     BEGIN
      →         SELECT 'An error has occurred, the stored procedure was terminated';
      →     END;
      →
      →     SET p_date:=SYSDATE();
      →     SELECT DateofIssue INTO i_date FROM Borrower WHERE Roll_no=roll;
      →     SET diff_date:=DATEDIFF(p_date, i_date);
      →
      →     SELECT Roll_no AS 'Roll no', Name AS 'Name', DateofIssue AS 'Date of Issue', NameofBook AS 'Name of Boo
      → k', diff_date AS 'Days since last issued' FROM Borrower WHERE Roll_no=roll;
      →
      →     IF (15 ⩽ diff_date AND diff_date ⩽ 30) THEN
      →         SET fine_amt:=diff_date*5;
      →     ELSE
      →         SET fine_amt:=(diff_date-30)*50 + 15*5;
      →     END IF;
      →
      →     UPDATE Borrower SET status:='R' WHERE Roll_no=roll;
      →
      →     IF (fine_amt>0) THEN
      →         INSERT INTO Fine VALUES (roll, p_date, fine_amt);
      →     END IF;
      →
      →     SELECT * FROM Fine;
      → END$$
      →
      → DELIMITER ;
Query OK, 0 rows affected
Time: 0.005s

Changed delimiter to $$
Time: 0.000s

Query OK, 0 rows affected
Time: 0.012s

Changed delimiter to ;
```

```
Time: 0.000s

dbmsl > CALL LIBRARY(1, 'Revolution 2020');
```

Roll no	Name	Date of Issue	Name of Book	Days since last issued
1	Pratik	2021-11-01	Revolution 2020	5

```
1 row in set
Time: 0.009s
```

Roll_no	Date	Amt

```
0 rows in set
Time: 0.007s

dbmsl > CALL LIBRARY(2, 'Midnights Children');
```

Roll no	Name	Date of Issue	Name of Book	Days since last issued
2	Alex	2021-10-12	Midnights Children	25

```
1 row in set
Time: 0.007s
```

Roll_no	Date	Amt
2	2021-11-06	125

```
1 row in set
Time: 0.005s

dbmsl > CALL LIBRARY(3, 'Only Time Will Tell');
```

Roll no	Name	Date of Issue	Name of Book	Days since last issued
3	Jhon	2021-09-03	Only Time Will Tell	64

```
1 row in set
Time: 0.009s
```

Roll_no	Date	Amt
2	2021-11-06	125
3	2021-11-06	1775

```
2 rows in set
Time: 0.003s

dbmsl > SELECT * FROM Borrower;
```

Roll_no	Name	DateofIssue	NameofBook	Status
1	Pratik	2021-11-01	Revolution 2020	R
2	Alex	2021-10-12	Midnights Children	R
3	Jhon	2021-09-03	Only Time Will Tell	R

```
3 rows in set
Time: 0.010s

dbmsl >
```

```
┌─proxzima@proxzima in ~ on ☁  pratik@convin.ai took 3ms [🔋 68%]
└─☣ mycli -d proxzima

(none) > USE dbmsl;
You are now connected to database "dbmsl" as user "proxzima"
Time: 0.001s

dbmsl > CREATE TABLE circleArea (radius INT, area FLOAT);
Query OK, 0 rows affected
Time: 0.016s

dbmsl > DROP PROCEDURE IF EXISTS calcCircleArea;
      → DELIMITER $$
      → CREATE PROCEDURE calcCircleArea(
      →     IN radius INT
      → )
      → BEGIN
      →     DECLARE area FLOAT(5);
      →     DECLARE pi FLOAT(2) DEFAULT 3.14;
      →
      →     IF 5⩽radius AND radius⩽9 THEN
      →         SET area:=pi*POWER(radius, 2);
      →         INSERT INTO circleArea VALUES(radius, area);
      →     ELSE
      →         SELECT CONCAT('Enter valid radius: ', radius) AS 'ERROR';
      →     END IF;
      → END$$
      →
      → DELIMITER ;
Query OK, 0 rows affected
Time: 0.005s

Changed delimiter to $$
Time: 0.000s

Query OK, 0 rows affected
Time: 0.010s

Changed delimiter to ;
Time: 0.000s

dbmsl > CALL calcCircleArea(5);
Query OK, 1 row affected
Time: 0.004s

dbmsl > CALL calcCircleArea(7);
Query OK, 1 row affected
Time: 0.004s

dbmsl > CALL calcCircleArea(9);
Query OK, 1 row affected
Time: 0.003s

dbmsl > SELECT * FROM circleArea;
+--------+--------+
| radius | area   |
```

5	78.5
7	153.86
9	254.34

3 rows *in* set
Time: 0.014s

```
dbmsl > CALL calcCircleArea(4);
```

ERROR
Enter valid radius: 4

1 row *in* set
Time: 0.007s

```
dbmsl > CALL calcCircleArea(10);
```

ERROR
Enter valid radius: 10

1 row *in* set
Time: 0.008s

```
dbmsl >
```

Pratik Pingale 190056.

PAGE :
DATE : / /

DBMSL ASSIGNMENT- 5

AIM : Write PL/SQL Stored Procedure & Stored Function

Write a stored procedure that places students according to their marks.

- 990 <= marks <= 1500 is distinction.
- 900 <= marks <= 989 is first class
- 825 <= marks <= 899 is second class

THEORY :

1) Stored Procedure:

- A stored procedure is a segment of stored declarative SQL statement stored inside the database catalogue.

- A stored procedure can be invoked by trigger other stored programs such as JAVA, C++, PHP.

- When stored procedure calls itself its called recursive S.P.

2) Stored Function:

- A stored function is a special type of stored program that returns a single value.

- We use stored function to encapsulate common formulas.

or basiness rules that may be reusable among SQL statements.

☆ Syntax

create function func_name ([Param1 . . . .])
Return datatype [characteristics]
Begin
< declarative statement >;
END ;

PAGE :
DATE : / /

CONCLUSION : In this experiment we learned about basic SQL DML, DCL, DDL Queries. along with writing basic PL/SQL blocks. and using them in practical usage in application.

```
┌─proxzima@proxzima in ~ via  v3.9.5 on ☁  pratik@convin.ai took 17m20s [▮ 100%]
└─$ mycli -d proxzima

(none) > USE dbmsl;
You are now connected to database "dbmsl" as user "proxzima"
Time: 0.001s

dbmsl > CREATE TABLE Stud_Marks (Roll INT PRIMARY KEY NOT NULL, name VARCHAR(255), total_marks INT);
Query OK, 0 rows affected
Time: 0.010s

dbmsl > INSERT INTO Stud_Marks VALUES (101, 'Alex', -46), (102, 'Pratik', 1300), (103, 'John', 683), (104,
      → 'Varun', 342), (105, 'Sam', 927), (106, 'Mohan', 767), (107, 'Rohan', 1847), (108, 'Andrew', 850);
Query OK, 8 rows affected
Time: 0.007s

dbmsl > CREATE TABLE Result (Roll INT PRIMARY KEY NOT NULL, name VARCHAR(255), class VARCHAR(50));
Query OK, 0 rows affected
Time: 0.011s

dbmsl > SELECT * FROM `Stud_Marks`;
```

Roll	name	total_marks
101	Alex	-46
102	Pratik	1300
103	John	683
104	Varun	342
105	Sam	927
106	Mohan	767
107	Rohan	1847
108	Andrew	850

```
8 rows in set
Time: 0.014s

dbmsl > DROP FUNCTION IF EXISTS final_Result;
      → DELIMITER $$
      → CREATE FUNCTION final_Result(
      →     marks INT
      → )
      → RETURNS VARCHAR(50)
      → DETERMINISTIC
      → BEGIN
      →     DECLARE class VARCHAR(255);
      →
      →     IF    (990 < marks and marks < 1500) THEN
      →         SET class:='Distincton';
      →     ELSEIF (890 < marks AND marks < 989) THEN
      →         SET class:='First Class';
      →     ELSEIF (825 < marks AND marks < 889) THEN
      →         SET class:='Higher Second Class';
      →     ELSEIF (750 < marks AND marks < 824) THEN
      →         SET class:='Second Class';
      →     ELSEIF (650 < marks AND marks < 749) THEN
      →         SET class:='Passed';
      →     ELSEIF   (0 < marks AND marks < 649) THEN
      →         SET class:='Fail';
      →     ELSE
      →         SET class:='Incorrect Marks';
      →     END IF;
      →
      →     RETURN (class);
      → END$$
      →
      → DELIMITER ;
Query OK, 0 rows affected
Time: 0.004s

Changed delimiter to $$
Time: 0.000s

Query OK, 0 rows affected
```

```
Time: 0.007s

Changed delimiter to ;
Time: 0.000s

dbmsl > DROP PROCEDURE IF EXISTS proc_Grade;
    -> DELIMITER $$
    -> CREATE PROCEDURE proc_Grade(
    ->     roll INT
    -> )
    -> BEGIN
    ->     DECLARE studName VARCHAR(50);
    ->     DECLARE marks INT;
    ->     DECLARE class VARCHAR(255);
    ->
    ->     DECLARE EXIT HANDLER FOR NOT FOUND
    ->     BEGIN
    ->         SELECT CONCAT('An error has occurred. Roll No ', roll, ' not found.') AS 'Error';
    ->     END;
    ->
    ->     SELECT sm.name, sm.total_marks INTO studName, marks FROM Stud_Marks sm WHERE sm.Roll=roll;
    ->
    ->     SET class:=final_Result(marks);
    ->
    ->     INSERT INTO Result VALUES (roll, studName, class);
    ->
    ->     SELECT sm.Roll AS 'Roll no', sm.name AS 'Name', class AS 'Class' FROM Stud_Marks sm WHERE sm.Ro
    -> ll=roll;
    -> END$$
    ->
    -> DELIMITER ;
Query OK, 0 rows affected
Time: 0.004s

Changed delimiter to $$
Time: 0.000s

Query OK, 0 rows affected
Time: 0.007s

Changed delimiter to ;
Time: 0.000s

dbmsl > CALL proc_Grade(101);
+---------+------+-----------------+
| Roll no | Name | Class           |
+---------+------+-----------------+
| 101     | Alex | Incorrect Marks |
+---------+------+-----------------+
1 row in set
Time: 0.011s

dbmsl > CALL proc_Grade(102);
+---------+--------+-------------+
| Roll no | Name   | Class       |
+---------+--------+-------------+
| 102     | Pratik | Distincton  |
+---------+--------+-------------+
1 row in set
Time: 0.006s

dbmsl > CALL proc_Grade(103);
+---------+------+--------+
| Roll no | Name | Class  |
+---------+------+--------+
| 103     | John | Passed |
+---------+------+--------+
1 row in set
Time: 0.009s

dbmsl > CALL proc_Grade(104);
+---------+--------+-------+
| Roll no | Name   | Class |
+---------+--------+-------+
| 104     | Varun  | Fail  |
+---------+--------+-------+
1 row in set
Time: 0.009s

dbmsl > CALL proc_Grade(105);
+---------+------+-------------+
| Roll no | Name | Class       |
+---------+------+-------------+
| 105     | Sam  | First Class |
+---------+------+-------------+
1 row in set
```

Time: 0.008s

```
dbmsl > CALL proc_Grade(106);
```

Roll no	Name	Class
106	Mohan	Second Class

1 row in set
Time: 0.005s

```
dbmsl > CALL proc_Grade(107);
```

Roll no	Name	Class
107	Rohan	Incorrect Marks

1 row in set
Time: 0.007s

```
dbmsl > CALL proc_Grade(108);
```

Roll no	Name	Class
108	Andrew	Higher Second Class

1 row in set
Time: 0.005s

```
dbmsl > CALL proc_Grade(109);
```

Error
An error has occurred. Roll No 109 not found.

1 row in set
Time: 0.008s

```
dbmsl > SELECT * FROM `Result`;
```

Roll	name	class
101	Alex	Incorrect Marks
102	Pratik	Distincton
103	John	Passed
104	Varun	Fail
105	Sam	First Class
106	Mohan	Second Class
107	Rohan	Incorrect Marks
108	Andrew	Higher Second Class

8 rows in set
Time: 0.009s

Pratik Pingale 19CO056

PAGE :
DATE : / /

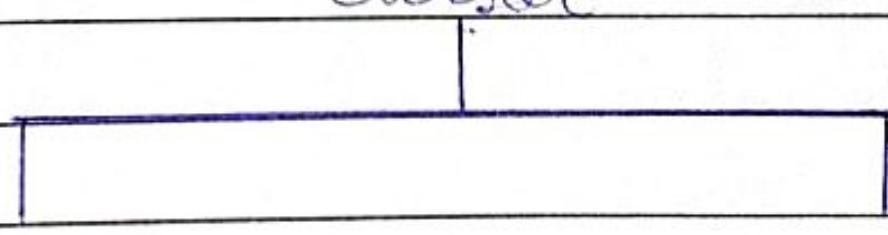
DBMSL ASSIGNMENT - 6

AIM : Write PL/SQL blocks of coding using parameter cursor that will merge the data available in newly created table Roll-no. with the Lates available in the table Roll.no of the data in the First table already exist in the second table.

Theory :

A cursor is a pointer in the content area. PL/SQL controls the cursor area through a cursor.

A cursor hold the rows one or more returned by SQL statement. The set of rows the cursor holds is active set. It could refer to a program to ~~set~~ fetch and process the rows returned by SQL statement. one at a time



→ Opening cursor:

It allocates the memory for the cursor and make it ready for fetching the rows returned by the SQL statement into it.

→ Fetching cursor:

Fetching of the cursor invalue accessing one row at a time.

Eg: . Fetch c customer

→ Declaring cursor:

Declaring the cursor that discuss i.e cursor with name and associates select statement.

→ Closing cursor:

closing the cursor or releasing the allated memory.

Eg: close c-customer.

PAGE :
DATE : / /

CONCLUSION : In this experiment we learned about basic SQL DML, DCL, DDL Queries. along with writing basic PL/SQL blocks. and using them in practical usage in application.

```
┌─proxzima@proxzima in ~ via  v3.9.5 took 108ms [ 98%]
└─$ mycli -d proxzima

(none) > USE dbmsl;
You are now connected to database "dbmsl" as user "proxzima"
Time: 0.001s

dbmsl > CREATE TABLE old_class(roll INT PRIMARY KEY, name VARCHAR(50));
Query OK, 0 rows affected
Time: 0.013s

dbmsl > INSERT INTO old_class VALUES (101, 'Alex'), (102, 'Pratik'), (103, 'John'), (104, 'Varun'), (105,
     → 'Sam'), (106, 'Mohan'), (107, 'Rohan'), (108, 'Andrew');
Query OK, 8 rows affected
Time: 0.005s

dbmsl > SELECT * FROM old_class;
+------+--------+
| roll | name   |
+------+--------+
| 101  | Alex   |
| 102  | Pratik |
| 103  | John   |
| 104  | Varun  |
| 105  | Sam    |
| 106  | Mohan  |
| 107  | Rohan  |
| 108  | Andrew |
+------+--------+
8 rows in set
Time: 0.008s

dbmsl > CREATE TABLE new_class(roll INT PRIMARY KEY, name VARCHAR(50));
Query OK, 0 rows affected
Time: 0.012s

dbmsl > INSERT INTO new_class VALUES (101, 'Alex'), (103, 'John'), (107, 'Rohan');
Query OK, 3 rows affected
Time: 0.004s

dbmsl > SELECT * FROM new_class;
+------+--------+
| roll | name   |
+------+--------+
| 101  | Alex   |
| 103  | John   |
| 107  | Rohan  |
+------+--------+
3 rows in set
Time: 0.008s

dbmsl > DROP PROCEDURE IF EXISTS copyTableOldInNew;
Query OK, 0 rows affected
Time: 0.004s

dbmsl > DELIMITER $$
Changed delimiter to $$
Time: 0.000s

dbmsl > CREATE PROCEDURE copyTableOldInNew()
     → BEGIN
     →   DECLARE o_roll INT DEFAULT 0;
     →   DECLARE o_name varchar(50);
     →   DECLARE done INT DEFAULT FALSE;
     →
     →   DECLARE tcursor cursor for SELECT roll, name FROM old_class;
     →   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
     →   OPEN tcursor;
     →   read_loop: LOOP
```

```
      →     FETCH tcursor into o_roll, o_name;
      →     if done then
      →       LEAVE read_loop;
      →     END IF;
      →     IF (SELECT 1 from new_class WHERE roll=o_roll) THEN
      →         SELECT CONCAT('Name: ', o_name) AS 'EXISTS IN DATABASE!! SKIPPING!!';
      →     ELSE
      →         INSERT INTO new_class(roll, name) values(o_roll, o_name);
      →     END IF;
      →   END LOOP;
      →   CLOSE tcursor;
      → END$$
Query OK, 0 rows affected
Time: 0.009s

dbmsl > DELIMITER ;
Changed delimiter to ;
Time: 0.000s

dbmsl > CALL copyTableOldInNew();
```

EXISTS IN DATABASE!! SKIPPING!!
Name: Alex

```
1 row in set
Time: 0.008s
```

EXISTS IN DATABASE!! SKIPPING!!
Name: John

```
1 row in set
Time: 0.007s
```

EXISTS IN DATABASE!! SKIPPING!!
Name: Rohan

```
1 row in set
Time: 0.007s

dbmsl > SELECT * FROM old_class;
```

roll	name
101	Alex
102	Pratik
103	John
104	Varun
105	Sam
106	Mohan
107	Rohan
108	Andrew

```
8 rows in set
Time: 0.008s

dbmsl > SELECT * FROM new_class;
```

roll	name
101	Alex
102	Pratik
103	John
104	Varun
105	Sam
106	Mohan
107	Rohan

Pratik Pingale 19C0056.

PAGE :
DATE : / /

DBMSL ASSIGNMENT - 7

AIM: Use database trigger [row, statement, before, after].

THEORY:

- Triggers are stored programs executed automatically when some event occurs.

- Triggers are injects written to be executed in response to an :

1) DML
2) DDL
3) Database operation

★ Benifits of Trigger

1) Generating some derived column values automatically
2) Enforcing referential integrity.
3) Event logging and storing information on table access.
4) Auditing
5) Synchronous replication of tables
6) Impressing security authorization.

PAGE :
DATE : / /

7) Preventing invalued transaction

* Creating triggers

Create or replace trigger trigger_name
<Before / After / Instead of>
<Insert / OR / update / delete>
[of Column-name]
on table_name
when condition

Declare
<declaration statement>
Begin
<executable statement>
Exception
<exception handeling>
END;

PAGE :
DATE : / /

CONCLUSION : In this experiment we learned about basic SQL DML, DCL, DDL Queries. along with writing basic PL/SQL blocks. and using them in practical usage in application.

```
┌─proxzima@proxzima in ~ via  v3.9.5 took 2ms [▮ 100%]
└─* mycli -d proxzima

(none) > USE dbmsl;
You are now connected to database "dbmsl" as user "proxzima"
Time: 0.001s

dbmsl > CREATE TABLE library (ISBN VARCHAR(50) PRIMARY KEY, name VARCHAR(255), author VARCHAR(255), publis
     → her VARCHAR(255), yearOfPublication INTEGER, cost FLOAT NOT NULL );
Query OK, 0 rows affected
Time: 0.012s

dbmsl > INSERT INTO library VALUES ('9874133548', 'Only Time Will Tell', 'Jeffery Archer', 'PAN', 2011, 56
     → 2.45), ('9784157165', 'Midnights Children', 'Salman Rushdie', 'VINTAGE', 2013, 452.25), ('94571548
     → 12', 'Revolution 2020', 'Jeffery Archer', 'RUPA', 2011, 842.75);
Query OK, 3 rows affected
Time: 0.005s

dbmsl > SELECT * FROM library;
```

ISBN	name	author	publisher	yearOfPublication	cost
9457154812	Revolution 2020	Jeffery Archer	RUPA	2011	842.75
9784157165	Midnights Children	Salman Rushdie	VINTAGE	2013	452.25
9874133548	Only Time Will Tell	Jeffery Archer	PAN	2011	562.45

```
3 rows in set
Time: 0.015s

dbmsl > CREATE TABLE library_audit (ISBN VARCHAR(50) PRIMARY KEY, name VARCHAR(255), author VARCHAR(255),
     → publisher VARCHAR(255), yearOfPublication INTEGER, cost FLOAT NOT NULL, trigger_type VARCHAR(50));
Query OK, 0 rows affected
Time: 0.016s

dbmsl > SELECT * FROM library_audit;
```

ISBN	name	author	publisher	yearOfPublication	cost	trigger_type

```
0 rows in set
Time: 0.009s

dbmsl > DELIMITER $$
Changed delimiter to $$
Time: 0.000s

dbmsl > CREATE TRIGGER update_library
     →     BEFORE UPDATE ON library
     →     FOR EACH ROW
     → BEGIN
     →     INSERT INTO library_audit VALUES (old.ISBN, old.name, old.author, old.publisher, old.yearOfPub
     → lication, old.cost, CONCAT('UPDATED: ', old.cost, ' → ', new.cost));
     → END $$
Query OK, 0 rows affected
Time: 0.009s

dbmsl > CREATE TRIGGER delete_library
     →     BEFORE DELETE ON library
     →     FOR EACH ROW
     → BEGIN
     →     INSERT INTO library_audit VALUES (old.ISBN, old.name, old.author, old.publisher, old.yearOfPub
     → lication, old.cost, 'DELETED');
     → END $$
Query OK, 0 rows affected
Time: 0.008s

dbmsl > DELIMITER ;
Changed delimiter to ;
Time: 0.000s

dbmsl > SELECT * FROM library;
```

ISBN	name	author	publisher	yearOfPublication	cost
9457154812	Revolution 2020	Jeffery Archer	RUPA	2011	842.75
9784157165	Midnights Children	Salman Rushdie	VINTAGE	2013	452.25
9874133548	Only Time Will Tell	Jeffery Archer	PAN	2011	562.45

```
3 rows in set
Time: 0.011s

dbmsl > UPDATE library SET cost=325.25 where ISBN=9457154812;
Query OK, 1 row affected
Time: 0.003s

dbmsl > SELECT * FROM library;
```

ISBN	name	author	publisher	yearOfPublication	cost
9457154812	Revolution 2020	Jeffery Archer	RUPA	2011	325.25
9784157165	Midnights Children	Salman Rushdie	VINTAGE	2013	452.25
9874133548	Only Time Will Tell	Jeffery Archer	PAN	2011	562.45

3 rows *in* set
Time: 0.009s

```
dbmsl > SELECT * FROM library_audit;
```

ISBN	name	author	publisher	yearOfPublication	cost	trigger_type
9457154812	Revolution 2020	Jeffery Archer	RUPA	2011	842.75	UPDATED: 842.75 → 325.25

1 row *in* set
Time: 0.009s

```
dbmsl > DELETE FROM library WHERE ISBN=9874133548;
Query OK, 1 row affected
Time: 0.004s

dbmsl > SELECT * FROM library;
```

ISBN	name	author	publisher	yearOfPublication	cost
9457154812	Revolution 2020	Jeffery Archer	RUPA	2011	325.25
9784157165	Midnights Children	Salman Rushdie	VINTAGE	2013	452.25

2 rows *in* set
Time: 0.009s

```
dbmsl > SELECT * FROM library_audit;
```

ISBN	name	author	publisher	yearOfPublication	cost	trigger_type
9457154812	Revolution 2020	Jeffery Archer	RUPA	2011	842.75	UPDATED: 842.75 → 325.25
9874133548	Only Time Will Tell	Jeffery Archer	PAN	2011	562.45	DELETED

2 rows *in* set
Time: 0.010s

Pratik Pingale 19C0056

DBMSL ASSIGNMENT - A8

AIM: Write a program to implement MYSQL / Oracle database connectivity with any frontend language to implement database navigation operations (CRUD).

THEORY:

* Java Database Connectivity (JDBC).

- It is an Application Programming Interface (API) for the programming language Java
- It defines how a client may access any kind of tabular data especially relational database.
- The JDBC classes are contained in the Java package java.sql and javax.sql
- JDBC helps you to write java application that manage these 3 programming activities:

1) Connect to a data source like a database.
2) Send update or query statement to the database.
3) Retrieve and process the results retrieved from the data base in answer to your query.

☆ Open Database Connectivity (ODBC):

- An ODBC driver uses 'this' interface by Microsoft that allows application to access data in database management System.

- ODBC uses 'SQL' as a standard for accessing the data.

- An Open Database Connectivity uses an ODBC interface for interaction.

- The ODBC driver manager loads and unloads ODBC driver on behalf of an application.

- It also processes ODBC function calls or passes them to an ODBC driver and resolves ODBC version conflicts.

CONCLUSION: Hence in this program we learned the importance and usage of ODBC as well as JDBC with a front end language like python and implemented it.

```python
import mysql.connector
from tabulate import tabulate


def showDatabase():
    print("→ SHOW DATABASES")

    mycursor.execute("SHOW DATABASES")
    print(tabulate(mycursor,
        headers=['Database'],
        tablefmt='fancy_grid')
    )

    mycursor.execute("USE dbmsl")


def createTable():
    print("\n\n→ CREATE TABLE")

    mycursor.execute("DROP TABLE library")
    mycursor.execute("CREATE TABLE library (ISBN VARCHAR(50) PRIMARY KEY, name VARCHAR(255), author
      VARCHAR(255), publisher VARCHAR(255), yearOfPublication INTEGER, cost FLOAT NOT NULL )")

    mycursor.execute("DESC library")
    print(tabulate(mycursor,
        headers=['Field', 'Type', 'Null', 'Key', 'Default', 'Extra'],
        tablefmt='fancy_grid')
    )


def insertInTable():
    print("\n\n→ INSERT INTO: library")

    sql = "INSERT INTO library VALUES (%s, %s, %s, %s, %s, %s)"
    val = [
        ('9874133548', 'Only Time Will Tell', 'Jeffery Archer', 'PAN', 2011, 562.45),
        ('9784157165', 'Midnights Children', 'Salman Rushdie', 'VINTAGE', 2013, 452.25),
        ('9457154812', 'Revolution 2020', 'Jeffery Archer', 'RUPA', 2011, 842.75),
        ('9421548465', 'The Old Man And His God', 'Sudha Murty', 'PENGUIN', 2006, 170.45)
    ]

    mycursor.executemany(sql, val)
    mydb.commit()

    mycursor.execute("SELECT * FROM library")

    print(tabulate(mycursor,
        headers=['ISBN', 'name', 'author', 'publisher', 'yearOfPublication', 'cost'],
        tablefmt='fancy_grid')
    )


def updateInTable():
    print("\n\n→ UPDATE TABLE: SET author for Revolution 2020")

    sql = "UPDATE library SET author=%s where name=%s"
    val = ('Chetan Bhagat', 'Revolution 2020')

    mycursor.execute(sql, val)
    mydb.commit()

    mycursor.execute("SELECT * FROM library")

    print(tabulate(mycursor,
        headers=['ISBN', 'name', 'author', 'publisher', 'yearOfPublication', 'cost'],
        tablefmt='fancy_grid')
    )


def deleteInTable():
    print("\n\n→ DELETE FROM TABLE: library WHERE ISBN is 9784157165")

    sql = "DELETE FROM library WHERE ISBN=%s"
    val = ('9784157165',)

    mycursor.execute(sql, val)
    mydb.commit()
```

```
    mycursor.execute("SELECT * FROM library")

    print(tabulate(mycursor,
        headers=['ISBN', 'name', 'author', 'publisher', 'yearOfPublication', 'cost'],
        tablefmt='fancy_grid')
    )


if __name__ == "__main__":
    mydb = mysql.connector.connect(
        host="localhost",
        user="proxzima",
        password="password"
    )

    if mydb.is_connected():
        print("→ MySQL connection is connected to Python.\n\n")

    mycursor = mydb.cursor()
    showDatabase()
    createTable()
    insertInTable()
    updateInTable()
    deleteInTable()
    print("\n\n→ Disconnecting MySQL connection")
    mycursor.close()
    mydb.close()
```

OUTPUT

```
┌─proxzima@proxzima in ~/Documents/AISSMS/SEM5/Practicals/DBMSL via  v3.9.5 took 181ms [ 70%]
└─ python file.py
→ MySQL connection is connected to Python.
```

→ SHOW DATABASES

Database
dbmsl
information_schema
mysql
performance_schema
sys

→ CREATE TABLE

Field	Type	Null	Key	Default	Extra
ISBN	varchar(50)	NO	PRI		
name	varchar(255)	YES			
author	varchar(255)	YES			
publisher	varchar(255)	YES			
yearOfPublication	int	YES			
cost	float	NO			

→ INSERT INTO: library

ISBN	name	author	publisher	yearOfPublication	cost
9421548465	The Old Man And His God	Sudha Murty	PENGUIN	2006	170.45
9457154812	Revolution 2020	Jeffery Archer	RUPA	2011	842.75
9784157165	Midnights Children	Salman Rushdie	VINTAGE	2013	452.25
9874133548	Only Time Will Tell	Jeffery Archer	PAN	2011	562.45

→ UPDATE TABLE: SET author *for* Revolution 2020

ISBN	name	author	publisher	yearOfPublication	cost
9421548465	The Old Man And His God	Sudha Murty	PENGUIN	2006	170.45
9457154812	Revolution 2020	Chetan Bhagat	RUPA	2011	842.75
9784157165	Midnights Children	Salman Rushdie	VINTAGE	2013	452.25
9874133548	Only Time Will Tell	Jeffery Archer	PAN	2011	562.45

→ DELETE FROM TABLE: library WHERE ISBN *is* 9784157165

ISBN	name	author	publisher	yearOfPublication	cost
9421548465	The Old Man And His God	Sudha Murty	PENGUIN	2006	170.45
9457154812	Revolution 2020	Chetan Bhagat	RUPA	2011	842.75
9874133548	Only Time Will Tell	Jeffery Archer	PAN	2011	562.45

→ Disconnecting MySQL connection

Bratik Pingale 19COD56

PAGE :
DATE : / /

DBMSL ASSIGNMENT - 1

AIM : Design and develop Mongo DB Queries using CRUD operations.

THEORY :

Mongo DB:

It is a cross platform object oriented database that provide high performance, high availability & easy scalability. Mongo DB works on concept of collection & document.

Database :

It is a physical container for collection Each database gets its own set of file on the file system. A mongDB server has more than one database.

Collection :

This is relative to table. It is a group of multiple documents. A collection exist in a database. Collection doesn't enfore a schema.

Document : A de

A document is a set of key value pairs having dynamic schema, i.e all documents need not be similar. Also similer fields can have different data

PAGE :
DATE : / /

★ Why use Mongo DB:

1) Document oriented storage
2) Index any attribute
3) Replication and high availability
4) Auto-sharding
5) Rich Queries.
6) Fast in place updates

★ Mongo DB Queries:

1) use dbs : Creates new 'dbs' if it doesn't exist else return existing 'dbs'.

→ use database-name ;

2) show dbs : show all the databases one at a time in room.

→ show dbs

3) dropDatabase() : db.dropDatabase() drops the database with name 'db'.

→ database_name.dropDatabase()

4) createCollection() : db.createCollection() creates a collection for a database db.
We can provide extra parameters as function argument

→ db.createCollection("c_name");

5) insert() : data is inserted using insert() function
It can insert only one or many values at a time.
Using insertOne() or insertMany() is recommended.

→ db.c_name.insert({ key: data });

6) insert One() : Insert only one document in database

→ db.cname.insertOne({key: data}).

7) insertMany() : Inserts more than one documents in database

→ db.cname.insertMany([{key1: v1}, {k2: v2}])

8) find(): To query data from a Mong DB Collection, find() command is used.

→ db.cname.find()

9) findOne(): Returns only one document from the database

→ db.cname.findOne()

10) update(): This methods updates values in a Collection.

→ db.cname.update(selection_criteria, updated_data)

11) save(): Replaces existing documents with updated values

→ db.c-name.save(orig-data, new-data).

PAGE :
DATE : / /

14) updateOne(): updates only one of document matching given filter.

→ db.cname.updateOne(filter, update)

15) updateMany(): updates all the documents matching given filter

→ db.cname.updateMany(filter, update)

16) remove(): removes document from collection matching a criteria.

→ db.cname.remove(deletion-criteria).

* Operators in Mongo DB.

1) AND : queries document using $and keyword.

→ db.cname.find({ $and: [{k1:v1}, {k2:v2}] })

2) OR : queries document using $or keyword

→ db.cname.find({ $or: [{k1:v1}, {k2:v2}] })

3) NOR : queries document using $nor keyword

→ db.cname.find({ $nor: [{k1:v1}, {k2:v2}] })

4) NOT : queries document using $not keyword

→ db.cname.find({ $not: {k1:v1} }).

PAGE :
DATE : / /

CONCLUSION : In this experiment we learned about various functions in MongoDB and implemented them in application for practical use.

Pratik Pingale
Roll No: 19CO056

Assignment - 1

```
┌─proxzima@proxzima in ~ took 3ms [ 39%]
└─☣ mongosh
Current Mongosh Log ID: 6167d891bf93dd17b7578ee6
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Using MongoDB:      5.0.3
Using Mongosh:      1.1.0

------

test> db
test
test> show dbs
admin    41 kB
config  111 kB
test> use dbms-prac
switched to db dbms-prac
dbms-prac> db.stats()
{
  db: 'dbms-prac',
  collections: 0,
  views: 0,
  objects: 0,
  avgObjSize: 0,
  dataSize: 0,
  storageSize: 0,
  totalSize: 0,
  indexes: 0,
  indexSize: 0,
  scaleFactor: 1,
  fileSize: 0,
  fsUsedSize: 0,
  fsTotalSize: 0,
  ok: 1
}
dbms-prac> show dbs
admin    41 kB
config  111 kB
dbms-prac> db.createCollection("collection")
{ ok: 1 }
dbms-prac> show collections
collection
dbms-prac> db.collection.drop()
true
dbms-prac> show collections

dbms-prac> db.createCollection("collection")
{ ok: 1 }
dbms-prac> db.empDetails.insert(
... {
..... First_Name: "Radhika",
..... Last_Name: "Sharma",
..... Date_Of_Birth: "1995-09-26",
..... e_mail: "radhika_sharma.123@gmail.com",
..... phone: "9848022338"
..... }
... )
DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("6167db000143152a11dafcf0") }
}
dbms-prac> db.empDetails.insert([
... {
..... First_Name: "Alex",
..... Last_Name: "Ron",
..... Date_Of_Birth: "1996-04-26",
```

```
..... e_mail: "alex123@gmail.com",
..... phone: "9784514545"
..... },
... {
..... First_Name: "James",
..... Last_Name: "Ford",
..... Date_Of_Birth: "1998-01-14",
..... e_mail: "fordjames123@gmail.com",
..... phone: "9413575512"
..... }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("6167dd040143152a11dafcf1"),
    '1': ObjectId("6167dd040143152a11dafcf2")
  }
}
dbms-prac> db.empDetails.insertOne(
... {
..... First_Name: "Rachel",
..... Last_Name: "Christopher",
..... Date_Of_Birth: "1990-02-16",
..... e_mail: "Rachel_Christopher.123@gmail.com",
..... phone: "9000054321"
..... }
... )
{
  acknowledged: true,
  insertedId: ObjectId("6167ddd20143152a11dafcf3")
}
dbms-prac> db.empDetails.insertMany([
... {
..... First_Name: "Fathima",
..... Last_Name: "Sheik",
..... Date_Of_Birth: "1990-02-16",
..... e_mail: "Fathima_Sheik.123@gmail.com",
..... phone: "9000054321"
..... },
... {
..... First_Name: "Gaurav",
..... Last_Name: "Dalvi",
..... Date_Of_Birth: "2000-10-24",
..... e_mail: "dalvi2000@gmail.com",
..... phone: "9474121846"
..... }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("6167de270143152a11dafcf4"),
    '1': ObjectId("6167de270143152a11dafcf5")
  }
}
dbms-prac> db.empDetails.find()
[
  {
    _id: ObjectId("6167db000143152a11dafcf0"),
    First_Name: 'Radhika',
    Last_Name: 'Sharma',
    Date_Of_Birth: '1995-09-26',
    e_mail: 'radhika_sharma.123@gmail.com',
    phone: '9848022338'
  },
  {
    _id: ObjectId("6167dd040143152a11dafcf1"),
    First_Name: 'Alex',
    Last_Name: 'Ron',
    Date_Of_Birth: '1996-04-26',
    e_mail: 'alex123@gmail.com',
    phone: '9784514545'
  },
  {
    _id: ObjectId("6167dd040143152a11dafcf2"),
    First_Name: 'James',
    Last_Name: 'Ford',
    Date_Of_Birth: '1998-01-14',
    e_mail: 'fordjames123@gmail.com',
```

```
    phone: '9413575512'
  },
  {
    _id: ObjectId("6167ddd20143152a11dafcf3"),
    First_Name: 'Rachel',
    Last_Name: 'Christopher',
    Date_Of_Birth: '1990-02-16',
    e_mail: 'Rachel_Christopher.123@gmail.com',
    phone: '9000054321'
  },
  {
    _id: ObjectId("6167de270143152a11dafcf4"),
    First_Name: 'Fathima',
    Last_Name: 'Sheik',
    Date_Of_Birth: '1990-02-16',
    e_mail: 'Fathima_Sheik.123@gmail.com',
    phone: '9000054321'
  },
  {
    _id: ObjectId("6167de270143152a11dafcf5"),
    First_Name: 'Gaurav',
    Last_Name: 'Dalvi',
    Date_Of_Birth: '2000-10-24',
    e_mail: 'dalvi2000@gmail.com',
    phone: '9474121846'
  }
]
dbms-prac> db.empDetails.findOne({First_Name: "Alex"})
{
  _id: ObjectId("6167dd040143152a11dafcf1"),
  First_Name: 'Alex',
  Last_Name: 'Ron',
  Date_Of_Birth: '1996-04-26',
  e_mail: 'alex123@gmail.com',
  phone: '9784514545'
}
dbms-prac> db.empDetails.findOne({First_Name: {$in: ["Gaurav", "Rachel"]}})
{
  _id: ObjectId("6167ddd20143152a11dafcf3"),
  First_Name: 'Rachel',
  Last_Name: 'Christopher',
  Date_Of_Birth: '1990-02-16',
  e_mail: 'Rachel_Christopher.123@gmail.com',
  phone: '9000054321'
}
dbms-prac> db.empDetails.find({First_Name: {$in: ["Gaurav", "Rachel"]}})
[
  {
    _id: ObjectId("6167ddd20143152a11dafcf3"),
    First_Name: 'Rachel',
    Last_Name: 'Christopher',
    Date_Of_Birth: '1990-02-16',
    e_mail: 'Rachel_Christopher.123@gmail.com',
    phone: '9000054321'
  },
  {
    _id: ObjectId("6167de270143152a11dafcf5"),
    First_Name: 'Gaurav',
    Last_Name: 'Dalvi',
    Date_Of_Birth: '2000-10-24',
    e_mail: 'dalvi2000@gmail.com',
    phone: '9474121846'
  }
]
dbms-prac> db.empDetails.find({Date_Of_Birth: {$gt: "1995-01-01"}})
[
  {
    _id: ObjectId("6167db000143152a11dafcf0"),
    First_Name: 'Radhika',
    Last_Name: 'Sharma',
    Date_Of_Birth: '1995-09-26',
    e_mail: 'radhika_sharma.123@gmail.com',
    phone: '9848022338'
  },
  {
    _id: ObjectId("6167dd040143152a11dafcf1"),
    First_Name: 'Alex',
```

```
    Last_Name: 'Ron',
    Date_Of_Birth: '1996-04-26',
    e_mail: 'alex123@gmail.com',
    phone: '9784514545'
  },
  {
    _id: ObjectId("6167dd040143152a11dafcf2"),
    First_Name: 'James',
    Last_Name: 'Ford',
    Date_Of_Birth: '1998-01-14',
    e_mail: 'fordjames123@gmail.com',
    phone: '9413575512'
  },
  {
    _id: ObjectId("6167de270143152a11dafcf5"),
    First_Name: 'Gaurav',
    Last_Name: 'Dalvi',
    Date_Of_Birth: '2000-10-24',
    e_mail: 'dalvi2000@gmail.com',
    phone: '9474121846'
  }
]
dbms-prac> db.empDetails.find({$and: [{Date_Of_Birth: {$gt: "1995-01-01"}}, {First_Name: "Alex"}]})
[
  {
    _id: ObjectId("6167dd040143152a11dafcf1"),
    First_Name: 'Alex',
    Last_Name: 'Ron',
    Date_Of_Birth: '1996-04-26',
    e_mail: 'alex123@gmail.com',
    phone: '9784514545'
  }
]
dbms-prac> db.empDetails.find({$or: [{Date_Of_Birth: {$gt: "1995-01-01"}}, {First_Name: "Alex"}]})
[
  {
    _id: ObjectId("6167db000143152a11dafcf0"),
    First_Name: 'Radhika',
    Last_Name: 'Sharma',
    Date_Of_Birth: '1995-09-26',
    e_mail: 'radhika_sharma.123@gmail.com',
    phone: '9848022338'
  },
  {
    _id: ObjectId("6167dd040143152a11dafcf1"),
    First_Name: 'Alex',
    Last_Name: 'Ron',
    Date_Of_Birth: '1996-04-26',
    e_mail: 'alex123@gmail.com',
    phone: '9784514545'
  },
  {
    _id: ObjectId("6167dd040143152a11dafcf2"),
    First_Name: 'James',
    Last_Name: 'Ford',
    Date_Of_Birth: '1998-01-14',
    e_mail: 'fordjames123@gmail.com',
    phone: '9413575512'
  },
  {
    _id: ObjectId("6167de270143152a11dafcf5"),
    First_Name: 'Gaurav',
    Last_Name: 'Dalvi',
    Date_Of_Birth: '2000-10-24',
    e_mail: 'dalvi2000@gmail.com',
    phone: '9474121846'
  }
]
dbms-prac> db.empDetails.update({First_Name: "Alex"}, {$set:{e_mail: "newalex123@gmail.com"}})
DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany, or bulkWrite.
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
dbms-prac> db.empDetails.findOne({First_Name: "Alex"})
```

```
{
  _id: ObjectId("6167dd040143152a11dafcf1"),
  First_Name: 'Alex',
  Last_Name: 'Ron',
  Date_Of_Birth: '1996-04-26',
  e_mail: 'newalex123@gmail.com',
  phone: '9784514545'
}
dbms-prac> db.empDetails.updateOne({First_Name: "Rachel"}, {$set:{e_mail: "newrachel123@gmail.com"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
dbms-prac> db.empDetails.findOne({First_Name: "Rachel"})
{
  _id: ObjectId("6167ddd20143152a11dafcf3"),
  First_Name: 'Rachel',
  Last_Name: 'Christopher',
  Date_Of_Birth: '1990-02-16',
  e_mail: 'newrachel123@gmail.com',
  phone: '9000054321'
}
dbms-prac> db.empDetails.updateMany({Date_Of_Birth: {$gt: "1995-01-01"}}, {$set: {Date_Of_Birth:
"1995-01-01"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 4,
  modifiedCount: 4,
  upsertedCount: 0
}
dbms-prac> db.empDetails.find()
[
  {
    _id: ObjectId("6167db000143152a11dafcf0"),
    First_Name: 'Radhika',
    Last_Name: 'Sharma',
    Date_Of_Birth: '1995-01-01',
    e_mail: 'radhika_sharma.123@gmail.com',
    phone: '9848022338'
  },
  {
    _id: ObjectId("6167dd040143152a11dafcf1"),
    First_Name: 'Alex',
    Last_Name: 'Ron',
    Date_Of_Birth: '1995-01-01',
    e_mail: 'newalex123@gmail.com',
    phone: '9784514545'
  },
  {
    _id: ObjectId("6167dd040143152a11dafcf2"),
    First_Name: 'James',
    Last_Name: 'Ford',
    Date_Of_Birth: '1995-01-01',
    e_mail: 'fordjames123@gmail.com',
    phone: '9413575512'
  },
  {
    _id: ObjectId("6167ddd20143152a11dafcf3"),
    First_Name: 'Rachel',
    Last_Name: 'Christopher',
    Date_Of_Birth: '1990-02-16',
    e_mail: 'newrachel123@gmail.com',
    phone: '9000054321'
  },
  {
    _id: ObjectId("6167de270143152a11dafcf4"),
    First_Name: 'Fathima',
    Last_Name: 'Sheik',
    Date_Of_Birth: '1990-02-16',
    e_mail: 'Fathima_Sheik.123@gmail.com',
    phone: '9000054321'
  },
  {
    _id: ObjectId("6167de270143152a11dafcf5"),
```

```
    First_Name: 'Gaurav',
    Last_Name: 'Dalvi',
    Date_Of_Birth: '1995-01-01',
    e_mail: 'dalvi2000@gmail.com',
    phone: '9474121846'
  }
]
dbms-prac> db.empDetails.deleteOne({First_Name: "Alex"})
{ acknowledged: true, deletedCount: 1 }
dbms-prac> db.empDetails.findOne({First_Name: "Alex"})
null
dbms-prac> db.empDetails.deleteMany({Date_Of_Birth: {$gte: "1995-01-01"}})
{ acknowledged: true, deletedCount: 3 }
dbms-prac> db.empDetails.find()
[
  {
    _id: ObjectId("6167ddd20143152a11dafcf3"),
    First_Name: 'Rachel',
    Last_Name: 'Christopher',
    Date_Of_Birth: '1990-02-16',
    e_mail: 'newrachel123@gmail.com',
    phone: '9000054321'
  },
  {
    _id: ObjectId("6167de270143152a11dafcf4"),
    First_Name: 'Fathima',
    Last_Name: 'Sheik',
    Date_Of_Birth: '1990-02-16',
    e_mail: 'Fathima_Sheik.123@gmail.com',
    phone: '9000054321'
  }
]
dbms-prac>
```

Pratik Pingale 19CO056.

PAGE :
DATE : / /

DBMSL ASSIGNMENT - 2.

AIM : Design & develop mongo DB queries using aggeration & indexing

THEORY :

Indexing : Indexing supports the efficient resolution of queries. Without indexes, MongoDB must scan every document of selection to select one. which in inefficient.

Indexes are special data structure, that store small portion of data set in an easy to traverse form. by storing values of specific fields or set of fields. ordered by value.

* create_Index() method:

→ db.c_name.create_Index({key:1})
- here key is the field name.
- multiple fields can be passed.

* dropIndex()
- drops a single index

→ db.c_name.dropIndex({key:1})

PAGE :
DATE : / /

★ dropIndexes()

- multiple arguments can be passed as arguments to drop a key.

→ db.c_name.dropIndexes({K1:1, K2:-1})

★ getIndex()

- Returns description of all indexes in an collection.

→ db.c_name.getIndexes()

★ Aggregations :

Process data records and return computed results. Groups multiple values together. Similar to count(*) with GROUP BY of SQL.

→ aggregate().

- aggeration operation groups values from a document

→ db.c_name.aggerate(aggration_operation)

– List of aggerate operation :

1) $sum
2) $avg
3) $min
4) $max
5) $push
6) $addToSet
7) $first
8) $last

PAGE :
DATE : / /

☆ Pipeline Concept :

- Pipeline in mongo means using output of first command as input of second.
- There is a set of possible stages and each of those is taken as a set of documents as an input and produce result.

→ Possible stages :

1) $project : select specific field.

2) $match : reduces amount of document for next stage.

3) $group : Aggeration

4) $sort : Sorts the document.

5) $skip : skips document by a given amount

6) $limit : limits amount of document to look

7) $unwind : unwind the documents that use arrays.

PAGE :
DATE : / /

CONCLUSION : In this experiment we learned about various functions in MongoDB and implemented them in application for practical use.

Pratik Pingale
Roll No: 19CO056

Assignment - 2

```
┌─proxzima@proxzima in ~ took 54ms [ 42%]
└─☣ mongosh
Current Mongosh Log ID: 616c3218fe228d3ff35e567b
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Using MongoDB:      5.0.3
Using Mongosh:      1.1.0

------

test> show dbs
admin        41 kB
config     36.9 kB
dbms-prac  81.9 kB
local      8.19 kB
test> use dbms-prac
switched to db dbms-prac
dbms-prac> show collections
collection
empDetails
dbms-prac> db.indexAggregate.insertMany([
... {
..... First_Name: 'Radhika',
..... Last_Name: 'Sharma',
..... Date_Of_Birth: '1995-09-26',
..... e_mail: 'radhika_sharma.123@gmail.com',
..... phone: '9848022338'
..... },
... {
..... First_Name: 'Alex',
..... Last_Name: 'Ron',
..... Date_Of_Birth: '1996-04-26',
..... e_mail: 'alex123@gmail.com',
..... phone: '9784514545'
..... },
... {
..... First_Name: 'James',
..... Last_Name: 'Ford',
..... Date_Of_Birth: '1998-01-14',
..... e_mail: 'fordjames123@gmail.com',
..... phone: '9413575512'
..... },
... {
..... First_Name: 'Rachel',
..... Last_Name: 'Christopher',
..... Date_Of_Birth: '1990-02-16',
..... e_mail: 'Rachel_Christopher.123@gmail.com',
..... phone: '9000054321'
..... },
... {
..... First_Name: 'Fathima',
..... Last_Name: 'Sheik',
..... Date_Of_Birth: '1990-02-16',
..... e_mail: 'Fathima_Sheik.123@gmail.com',
..... phone: '9000054321'
..... },
... {
..... First_Name: 'Gaurav',
..... Last_Name: 'Dalvi',
..... Date_Of_Birth: '2000-10-24',
..... e_mail: 'dalvi2000@gmail.com',
..... phone: '9474121846'
..... }
```

```
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("616c335b1a765adeb7084274"),
    '1': ObjectId("616c335b1a765adeb7084275"),
    '2': ObjectId("616c335b1a765adeb7084276"),
    '3': ObjectId("616c335b1a765adeb7084277"),
    '4': ObjectId("616c335b1a765adeb7084278"),
    '5': ObjectId("616c335b1a765adeb7084279")
  }
}
dbms-prac> db.indexAggregate.find()
[
  {
    _id: ObjectId("616c335b1a765adeb7084274"),
    First_Name: 'Radhika',
    Last_Name: 'Sharma',
    Date_Of_Birth: '1995-09-26',
    e_mail: 'radhika_sharma.123@gmail.com',
    phone: '9848022338'
  },
  {
    _id: ObjectId("616c335b1a765adeb7084275"),
    First_Name: 'Alex',
    Last_Name: 'Ron',
    Date_Of_Birth: '1996-04-26',
    e_mail: 'alex123@gmail.com',
    phone: '9784514545'
  },
  {
    _id: ObjectId("616c335b1a765adeb7084276"),
    First_Name: 'James',
    Last_Name: 'Ford',
    Date_Of_Birth: '1998-01-14',
    e_mail: 'fordjames123@gmail.com',
    phone: '9413575512'
  },
  {
    _id: ObjectId("616c335b1a765adeb7084277"),
    First_Name: 'Rachel',
    Last_Name: 'Christopher',
    Date_Of_Birth: '1990-02-16',
    e_mail: 'Rachel_Christopher.123@gmail.com',
    phone: '9000054321'
  },
  {
    _id: ObjectId("616c335b1a765adeb7084278"),
    First_Name: 'Fathima',
    Last_Name: 'Sheik',
    Date_Of_Birth: '1990-02-16',
    e_mail: 'Fathima_Sheik.123@gmail.com',
    phone: '9000054321'
  },
  {
    _id: ObjectId("616c335b1a765adeb7084279"),
    First_Name: 'Gaurav',
    Last_Name: 'Dalvi',
    Date_Of_Birth: '2000-10-24',
    e_mail: 'dalvi2000@gmail.com',
    phone: '9474121846'
  }
]
dbms-prac> db.indexAggregate.createIndex({"First_Name": 1})
First_Name_1
dbms-prac> db.indexAggregate.dropIndex({"First_Name": 1})
{ nIndexesWas: 2, ok: 1 }
dbms-prac> db.indexAggregate.createIndex({"First_Name": 1, "phone": -1})
First_Name_1_phone_-1
dbms-prac> db.indexAggregate.dropIndexes({"First_Name": 1, "phone": -1})
{ nIndexesWas: 2, ok: 1 }
dbms-prac> db.indexAggregate.createIndex({"First_Name": 1, "phone": -1})
First_Name_1_phone_-1
dbms-prac> db.indexAggregate.getIndexes()
```

```
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  {
    v: 2,
    key: { First_Name: 1, phone: -1 },
    name: 'First_Name_1_phone_-1'
  }
]
dbms-prac> db.indexAggregate.aggregate([{$group: {_id: "$First_Name", phone: {$min: "$phone"}}}])
[
  { _id: 'Radhika', phone: '9848022338' },
  { _id: 'James', phone: '9413575512' },
  { _id: 'Fathima', phone: '9000054321' },
  { _id: 'Rachel', phone: '9000054321' },
  { _id: 'Gaurav', phone: '9474121846' },
  { _id: 'Alex', phone: '9784514545' }
]
dbms-prac> db.indexAggregate.find().limit(2)
[
  {
    _id: ObjectId("616c335b1a765adeb7084274"),
    First_Name: 'Radhika',
    Last_Name: 'Sharma',
    Date_Of_Birth: '1995-09-26',
    e_mail: 'radhika_sharma.123@gmail.com',
    phone: '9848022338'
  },
  {
    _id: ObjectId("616c335b1a765adeb7084275"),
    First_Name: 'Alex',
    Last_Name: 'Ron',
    Date_Of_Birth: '1996-04-26',
    e_mail: 'alex123@gmail.com',
    phone: '9784514545'
  }
]
dbms-prac> db.indexAggregate.find().limit(5).skip(2)
[
  {
    _id: ObjectId("616c335b1a765adeb7084276"),
    First_Name: 'James',
    Last_Name: 'Ford',
    Date_Of_Birth: '1998-01-14',
    e_mail: 'fordjames123@gmail.com',
    phone: '9413575512'
  },
  {
    _id: ObjectId("616c335b1a765adeb7084277"),
    First_Name: 'Rachel',
    Last_Name: 'Christopher',
    Date_Of_Birth: '1990-02-16',
    e_mail: 'Rachel_Christopher.123@gmail.com',
    phone: '9000054321'
  },
  {
    _id: ObjectId("616c335b1a765adeb7084278"),
    First_Name: 'Fathima',
    Last_Name: 'Sheik',
    Date_Of_Birth: '1990-02-16',
    e_mail: 'Fathima_Sheik.123@gmail.com',
    phone: '9000054321'
  },
  {
    _id: ObjectId("616c335b1a765adeb7084279"),
    First_Name: 'Gaurav',
    Last_Name: 'Dalvi',
    Date_Of_Birth: '2000-10-24',
    e_mail: 'dalvi2000@gmail.com',
    phone: '9474121846'
  }
]
dbms-prac> db.indexAggregate.find().sort({"phone": -1})
[
  {
    _id: ObjectId("616c335b1a765adeb7084274"),
```

```
    First_Name: 'Radhika',
    Last_Name: 'Sharma',
    Date_Of_Birth: '1995-09-26',
    e_mail: 'radhika_sharma.123@gmail.com',
    phone: '9848022338'
  },
  {
    _id: ObjectId("616c335b1a765adeb7084275"),
    First_Name: 'Alex',
    Last_Name: 'Ron',
    Date_Of_Birth: '1996-04-26',
    e_mail: 'alex123@gmail.com',
    phone: '9784514545'
  },
  {
    _id: ObjectId("616c335b1a765adeb7084279"),
    First_Name: 'Gaurav',
    Last_Name: 'Dalvi',
    Date_Of_Birth: '2000-10-24',
    e_mail: 'dalvi2000@gmail.com',
    phone: '9474121846'
  },
  {
    _id: ObjectId("616c335b1a765adeb7084276"),
    First_Name: 'James',
    Last_Name: 'Ford',
    Date_Of_Birth: '1998-01-14',
    e_mail: 'fordjames123@gmail.com',
    phone: '9413575512'
  },
  {
    _id: ObjectId("616c335b1a765adeb7084277"),
    First_Name: 'Rachel',
    Last_Name: 'Christopher',
    Date_Of_Birth: '1990-02-16',
    e_mail: 'Rachel_Christopher.123@gmail.com',
    phone: '9000054321'
  },
  {
    _id: ObjectId("616c335b1a765adeb7084278"),
    First_Name: 'Fathima',
    Last_Name: 'Sheik',
    Date_Of_Birth: '1990-02-16',
    e_mail: 'Fathima_Sheik.123@gmail.com',
    phone: '9000054321'
  }
]
dbms-prac>
```

Pratik Pingale 19C0056

PAGE :
DATE : / /

DBMSL ASSIGNMENT - 3

AIM : Implement map reduce operation with suitable example using MongoDB.

THEORY :

1) Map reduce is a data processing paradigm for condidering large volume of data into useful aggerated result mongo DB uses Map Reduce commands.

2) MapReduce is used for processing large amount of data.

3) The availibily of MapReduce has been the reason for Hadoop's Succes & at the same time a major factor in limiting adaptions.

4) The map reduce function first quires the Collection, map the result document to emit key value pairs which is then reduced based on keys.

5) Map is a javascript function that maps a value with a key and returns a key value pair.

6) Reduce is a javascript function that maps a reduced all the document having source key.

PAGE :
DATE : / /

7) Query specifies the optional selection criteria

8) Sort specifies the optional sort criteria.

9) Limit specifies the optional max numbers of document to be returned.

PAGE :
DATE : / /

CONCLUSION : In this experiment we learned about various functions in MongoDB and implemented them in application for practical use.

```
┌─proxzima@proxzima in ~ via  v3.9.5 took 25ms [ 100%]
└─ mongosh
Current Mongosh Log ID: 61a90c9287b706e03bf08a0c
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
Using MongoDB:      5.0.4
Using Mongosh:      1.1.5

test> db
test

test> show dbs
admin        41 kB
config     36.9 kB
dbms-prac   143 kB
local      73.7 kB

test> use dbms-prac
switched to db dbms-prac

dbms-prac> show dbs
admin        41 kB
config     36.9 kB
dbms-prac   143 kB
local      73.7 kB

dbms-prac> db.stats()
{
  db: 'dbms-prac',
  collections: 3,
  views: 0,
  objects: 8,
  avgObjSize: 156.5,
  dataSize: 1252,
  storageSize: 61440,
  freeStorageSize: 16384,
  indexes: 4,
  indexSize: 81920,
  indexFreeStorageSize: 16384,
  totalSize: 143360,
  totalFreeStorageSize: 32768,
  scaleFactor: 1,
  fsUsedSize: 171861905408,
  fsTotalSize: 494024871936,
  ok: 1
}

dbms-prac> db.getCollectionInfos()
[
  {
    name: 'indexAggregate',
    type: 'collection',
    options: {},
    info: {
      readOnly: false,
      uuid: UUID("3093988a-bde8-4133-b442-8d075dd573ae")
    },
    idIndex: { v: 2, key: { _id: 1 }, name: '_id_' }
  },
  {
    name: 'collection',
    type: 'collection',
    options: {},
    info: {
      readOnly: false,
      uuid: UUID("da804109-2558-4565-8fe2-d17f643de0e7")
    },
    idIndex: { v: 2, key: { _id: 1 }, name: '_id_' }
  },
  {
    name: 'empDetails',
    type: 'collection',
    options: {},
    info: {
      readOnly: false,
```

```
      uuid: UUID("e48338a3-77b8-4687-9be7-de21d8bb5fc4")
    },
    idIndex: { v: 2, key: { _id: 1 }, name: '_id_' }
  }
]

dbms-prac> db.createCollection("student")
{ ok: 1 }

dbms-prac> db.student.insertMany([
... {
..... roll_no:101,
..... name:"Aishwarya",
..... marks:47,
..... fees_paid:25000
..... },
... {
..... roll_no:101,
..... name:"Aishwarya",
..... marks:48,
..... fees_paid:35000
..... },
... {
..... roll_no:102,
..... name:"Gita",
..... marks:45,
..... fees_paid:30000
..... },
... {
..... roll_no:102,
..... name:"Gita",
..... marks:43,
..... fees_paid:20000
..... },
... {
..... roll_no:103,
..... name:"Kavita",
..... marks:45,
..... fees_paid:30000
..... },
... {
..... roll_no:103,
..... name:"Kavita",
..... marks:44,
..... fees_paid:25000
..... },
... {
..... roll_no:104,
..... name:"Shweta",
..... marks:40,
..... fees_paid:35000
..... },
... {
..... roll_no:104,
..... name:"Shweta",
..... marks:42,
..... fees_paid:25000
..... },
... {
..... roll_no:105,
..... name:"Manali",
..... marks:43,
..... fees_paid:15000
..... },
... {
..... roll_no:105,
..... name:"Manali",
..... marks:44,
..... fees_paid:35000
..... }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("61a91473d837aac5beacc126"),
    '1': ObjectId("61a91473d837aac5beacc127"),
    '2': ObjectId("61a91473d837aac5beacc128"),
    '3': ObjectId("61a91473d837aac5beacc129"),
    '4': ObjectId("61a91473d837aac5beacc12a"),
    '5': ObjectId("61a91473d837aac5beacc12b"),
    '6': ObjectId("61a91473d837aac5beacc12c"),
```

```
    '7': ObjectId("61a91473d837aac5beacc12d"),
    '8': ObjectId("61a91473d837aac5beacc12e"),
    '9': ObjectId("61a91473d837aac5beacc12f")
  }
}

dbms-prac> db.student.mapReduce(
... function() {
..... emit(this.roll_no, this.marks);
..... },
... function(key, value) {
..... return Array.sum(value)
..... },
... {
..... out: "total_marks"
..... }
... )
{ result: 'total_marks', ok: 1 }

dbms-prac> db.total_marks.find()
[
  { _id: 103, value: 89 },
  { _id: 101, value: 95 },
  { _id: 104, value: 82 },
  { _id: 102, value: 88 },
  { _id: 105, value: 87 }
]

dbms-prac> db.student.mapReduce(
... function() {
..... emit(this.name, this.fees_paid);
..... },
... function(key, value) {
..... return Array.sum(value)
..... },
... {
..... out: "total_fees_paid"
..... }
... )
{ result: 'total_fees_paid', ok: 1 }

dbms-prac> db.total_fees_paid.find()
[
  { _id: 'Aishwarya', value: 60000 },
  { _id: 'Kavita', value: 55000 },
  { _id: 'Gita', value: 50000 },
  { _id: 'Shweta', value: 60000 },
  { _id: 'Manali', value: 50000 }
]
```

Pratik Pingale 19CO056.

DBMSL ASSIGNMENT-B4

AIM: Write a program to implement Mongo DB database connectivity with any front end language to implement database navigation operations (CRUD).

THEORY :

* Python :
- Python is an interpreted high-level general purpose programming language.
- Its design philosophy emphasizes code readability with its use of significant indentation.
- Its language construct as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

* MongoDB:
- MongoDB is a source-available cross-platform document-oriented database program.
- Classified as a No-SQL database program, MongDB uses JSON like document with optional schema.
- MongoDB is developed by mongoDB Inc. and licenced under server side Public Licence.

★ pyMongo :

- pyMongo is a python distribution containing tools for working with MongoDB.

- It is official MongoDB python driver for python.

- We access MongoDB data by creating a socket connection with MongoDB server.

→ Installation

- Pip is a python package management tool used to install packages.

: Pip install pymongo

→ Usage

- In python program use follow code to use the pymongo package.

: import pymongo

- Create a MongDB client

: myclient = pymong.MongoClient("mongodb://localhost/")

- Create or use database

: mydb = myclient['database name']

- Create or access a collection

: mycollection = mydb['collection-name'].

- Further operation on mycollection is similar to that of mongoDB.

CONCLUSION: Hence in this we learned the importance and usage of MongoDB. We implemented a program to connect any frontend language with MongoDB and further performed CRUD operation.

```python
import pymongo
from tabulate import tabulate
from pprint import pprint


def showDatabase():
    print("→ SHOW DATABASES")

    print(tabulate({
            'Database': myclient.list_database_names()
        },
        headers='keys',
        tablefmt='fancy_grid')
    )

    return myclient["dbms-prac"]


def createAndInsertInCollection():
    print("\n\n→ CREATE COLLECTION: empDetails")

    if 'empDetails' in mydb.list_collection_names():
        mydb['empDetails'].drop()

    myCollection = mydb['empDetails']

    myData = [
        {
            "First_Name": "Fathima",
            "Last_Name": "Sheik",
            "Date_Of_Birth": "1990-02-16",
            "e_mail": "Fathima_Sheik.123@gmail.com",
            "phone": "9000054321"
        },
        {
            "First_Name": "Gaurav",
            "Last_Name": "Dalvi",
            "Date_Of_Birth": "2000-10-24",
            "e_mail": "dalvi2000@gmail.com",
            "phone": "9474121846"
        },
        {
            "First_Name": "Rachel",
            "Last_Name": "Christopher",
            "Date_Of_Birth": "1990-02-16",
            "e_mail": "Rachel_Christopher.123@gmail.com",
            "phone": "9000054321"
        },
        {
            "First_Name": "Alex",
            "Last_Name": "Ron",
            "Date_Of_Birth": "1996-04-26",
            "e_mail": "alex123@gmail.com",
            "phone": "9784514545"
        },
        {
            "First_Name": "James",
            "Last_Name": "Ford",
            "Date_Of_Birth": "1998-01-14",
            "e_mail": "fordjames123@gmail.com",
            "phone": "9413575512"
        },
        {
            "First_Name": "Radhika",
            "Last_Name": "Sharma",
            "Date_Of_Birth": "1995-09-26",
            "e_mail": "radhika_sharma.123@gmail.com",
            "phone": "9848022338"
        }
    ]

    myCollection.insert_many(myData)
    print(tabulate({
            'Collections': mydb.list_collection_names()
        },
        headers='keys',
        tablefmt='fancy_grid')
```

```
    print("\n\n→ INSERT INTO: empDetails")

    cursor = myCollection.find({})
    for document in cursor:
        pprint(document)

    return myCollection


def updateInCollection():
    print("\n\n→ UPDATE Collection: SET new Date_Of_Birth to 1995-01-01 if $it > 1995-01-01")

    myquery = {
        "Date_Of_Birth": {
            "$gt": "1995-01-01"
        }
    }

    newvalues = {
        "$set": {
            "Date_Of_Birth": "1995-01-01"
        }
    }

    myCollection.update_many(myquery, newvalues)

    cursor = myCollection.find({})
    for document in cursor:
        pprint(document)


def deleteInCollection():
    print("\n\n→ DELETE FROM Collection: empDetails Date_Of_Birth ⩾ 1995-01-01")

    myquery = {
        "Date_Of_Birth": {
            "$gte": "1995-01-01"
        }
    }

    myCollection.delete_many(myquery)

    cursor = myCollection.find({})
    for document in cursor:
        pprint(document)


if __name__ == "__main__":
    try:
        myclient = pymongo.MongoClient("mongodb://127.0.0.1:27017/")
        myclient.server_info()
        print("→ MongoDB is connected to Python.\n\n")
    except pymongo.errors.ServerSelectionTimeoutError as err:
        print("→ Connection error!!!")
        exit()

    mydb = showDatabase()
    myCollection = createAndInsertInCollection()
    updateInCollection()
    deleteInCollection()
    print("\n\n→ Disconnecting MongoDB connection")
    myclient.close()
```

OUTPUT

```
┌─proxzima@proxzima in ~/Documents/AISSMS/SEM5/Practicals/DBMSL via  v3.9.5 took 3ms [ 36%]
└─ python file.py
→ MongoDB is connected to Python.


→ SHOW DATABASES
```

Database
admin
config
dbms-prac

```
local
```

→ CREATE COLLECTION: empDetails

```
Collections
indexAggregate
library
total_fees_paid
student
total_marks
empDetails
collection
```

→ INSERT INTO: empDetails

```
{'Date_Of_Birth': '1990-02-16',
 'First_Name': 'Fathima',
 'Last_Name': 'Sheik',
 '_id': ObjectId('61a9a9e23b261cff5aba3566'),
 'e_mail': 'Fathima_Sheik.123@gmail.com',
 'phone': '9000054321'
}
{'Date_Of_Birth': '2000-10-24',
 'First_Name': 'Gaurav',
 'Last_Name': 'Dalvi',
 '_id': ObjectId('61a9a9e23b261cff5aba3567'),
 'e_mail': 'dalvi2000@gmail.com',
 'phone': '9474121846'
}
{'Date_Of_Birth': '1990-02-16',
 'First_Name': 'Rachel',
 'Last_Name': 'Christopher',
 '_id': ObjectId('61a9a9e23b261cff5aba3568'),
 'e_mail': 'Rachel_Christopher.123@gmail.com',
 'phone': '9000054321'}
{'Date_Of_Birth': '1996-04-26',
 'First_Name': 'Alex',
 'Last_Name': 'Ron',
 '_id': ObjectId('61a9a9e23b261cff5aba3569'),
 'e_mail': 'alex123@gmail.com',
 'phone': '9784514545'
}
{'Date_Of_Birth': '1998-01-14',
 'First_Name': 'James',
 'Last_Name': 'Ford',
 '_id': ObjectId('61a9a9e23b261cff5aba356a'),
 'e_mail': 'fordjames123@gmail.com',
 'phone': '9413575512'
}
{'Date_Of_Birth': '1995-09-26',
 'First_Name': 'Radhika',
 'Last_Name': 'Sharma',
 '_id': ObjectId('61a9a9e23b261cff5aba356b'),
 'e_mail': 'radhika_sharma.123@gmail.com',
 'phone': '9848022338'
}
```

→ UPDATE Collection: SET new Date_Of_Birth to 1995-01-01 *if* $it > 1995-01-01

```
{'Date_Of_Birth': '1990-02-16',
 'First_Name': 'Fathima',
 'Last_Name': 'Sheik',
 '_id': ObjectId('61a9a9e23b261cff5aba3566'),
 'e_mail': 'Fathima_Sheik.123@gmail.com',
 'phone': '9000054321'
}
{'Date_Of_Birth': '1995-01-01',
 'First_Name': 'Gaurav',
 'Last_Name': 'Dalvi',
 '_id': ObjectId('61a9a9e23b261cff5aba3567'),
 'e_mail': 'dalvi2000@gmail.com',
 'phone': '9474121846'
}
{'Date_Of_Birth': '1990-02-16',
```

```
 'First_Name': 'Rachel',
 'Last_Name': 'Christopher',
 '_id': ObjectId('61a9a9e23b261cff5aba3568'),
 'e_mail': 'Rachel_Christopher.123@gmail.com',
 'phone': '9000054321'
}
{'Date_Of_Birth': '1995-01-01',
 'First_Name': 'Alex',
 'Last_Name': 'Ron',
 '_id': ObjectId('61a9a9e23b261cff5aba3569'),
 'e_mail': 'alex123@gmail.com',
 'phone': '9784514545'
}
{'Date_Of_Birth': '1995-01-01',
 'First_Name': 'James',
 'Last_Name': 'Ford',
 '_id': ObjectId('61a9a9e23b261cff5aba356a'),
 'e_mail': 'fordjames123@gmail.com',
 'phone': '9413575512'
}
{'Date_Of_Birth': '1995-01-01',
 'First_Name': 'Radhika',
 'Last_Name': 'Sharma',
 '_id': ObjectId('61a9a9e23b261cff5aba356b'),
 'e_mail': 'radhika_sharma.123@gmail.com',
 'phone': '9848022338'
}


→ DELETE FROM Collection: empDetails Date_Of_Birth ⩾ 1995-01-01
{'Date_Of_Birth': '1990-02-16',
 'First_Name': 'Fathima',
 'Last_Name': 'Sheik',
 '_id': ObjectId('61a9a9e23b261cff5aba3566'),
 'e_mail': 'Fathima_Sheik.123@gmail.com',
 'phone': '9000054321'
}
{'Date_Of_Birth': '1990-02-16',
 'First_Name': 'Rachel',
 'Last_Name': 'Christopher',
 '_id': ObjectId('61a9a9e23b261cff5aba3568'),
 'e_mail': 'Rachel_Christopher.123@gmail.com',
 'phone': '9000054321'
}


→ Disconnecting MongoDB connection
```

TIME TABLE SCHEDULER

DBMS MINI PROJECT

CLASS – COMPUTER TE 1

GROUP MEMBERS –

19CO038 PRANAV KAKANI

19CO043 ANURAG KHADTARE

19CO056 PRATIK PINGALE

ABSTRACT AND INTRODUCTION

- The project is developed to automatically generate timetable and schedule classes without clashing with each other.
- Timetable scheduler is capable of auto-generating separate timetable for Teachers and Rooms in the Institute based on the Class timetable created/auto-generated by the user.
- This will allow User to create and modify timetables easily and have them hosted online so that they can be retrieved easily.
- We can easily modify or update timetable incase new classes are added.

CONSTRAINTS SATISFIED

Hard Constraints

- Unique class timing.
- Course students <= room seating capacity.
- Two classes don't have same room.
- Class timing for each teacher is unique.
- Teachers are allocated to their course accordingly.

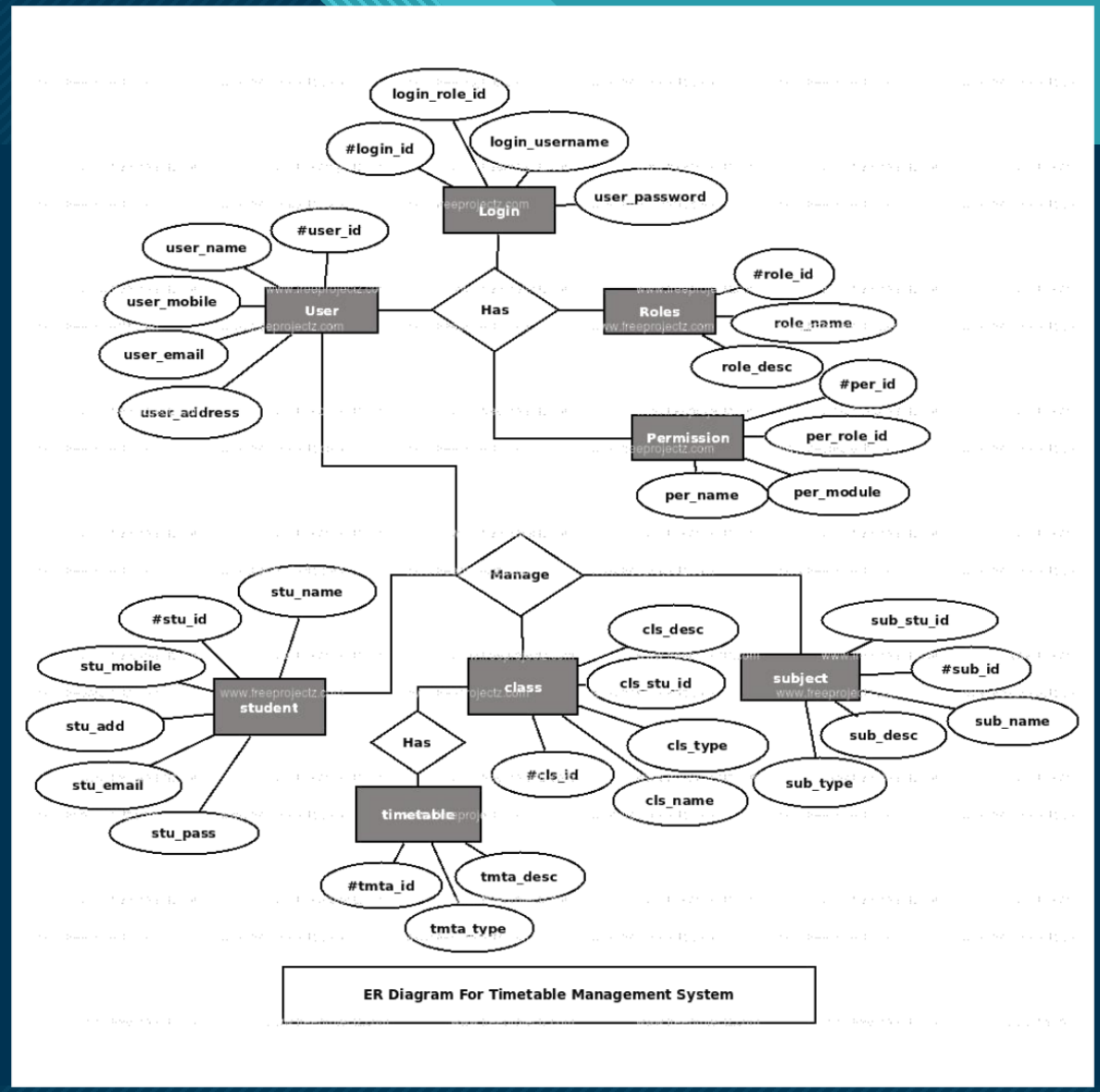
Soft Constraints

- classes are allotted according to section requirements.
- All courses are according to their department.
- Even distribution of course in a section per week.

SOFTWARE REQUIREMENTS:-

- OPERATING SYSTEM :- WINDOWS/LINUX
- SOFTWARE :- PYTHON = 3.9.5
  DJANGO = 3.2.9

SCHEMA


login_role_id
#login_id
login_username
user_password
Login
#user_id
user_name
user_mobile
User
user_email
user_address
Has
#role_id
Roles
role_name
role_desc
#per_id
Permission
per_role_id
per_name
per_module
Manage
stu_name
#stu_id
stu_mobile
student
stu_add
stu_email
stu_pass
Has
timetable
#tmta_id
tmta_desc
tmta_type
cls_desc
cls_stu_id
class
cls_type
#cls_id
cls_name
sub_stu_id
subject
#sub_id
sub_name
sub_desc
sub_type
ER Diagram For Timetable Management System

OPERATIONS ON THE SYSTEM

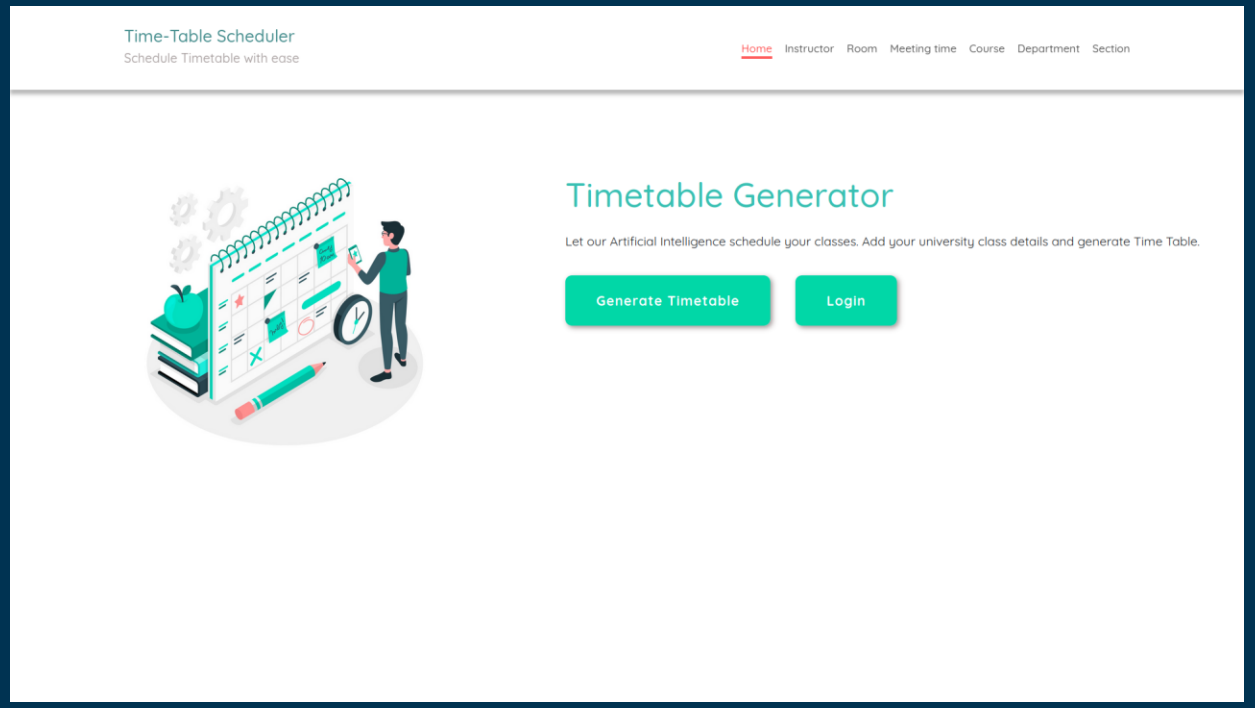
- THE USER WILL HAVE TO FEED THE FOLLOWING INFORMATION TO GET THE TIME TABLE :-

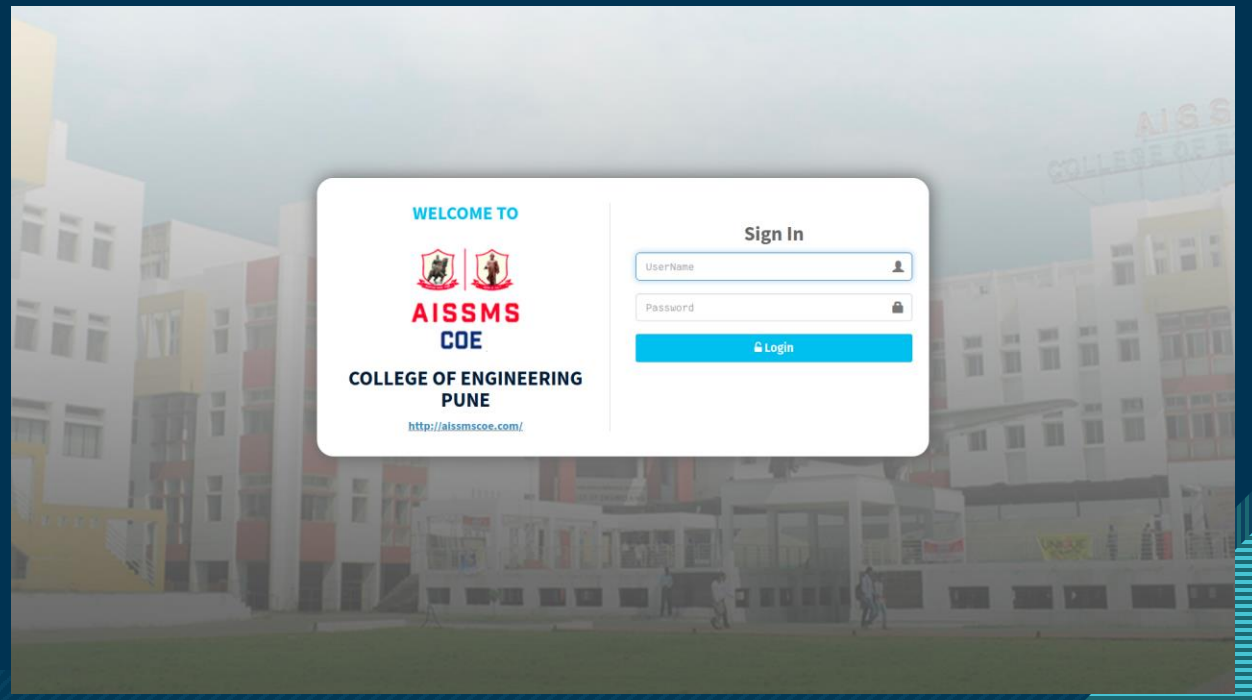
1. INSTRUCTOR ID AND NAME
2. ROOM NUMBER AND SEATING CAPACITY
3. MEETING TIME
4. NAME OF THE COURSE
5. DEPARTMENT NAME
6. SECTION NAME

OPERATION :-

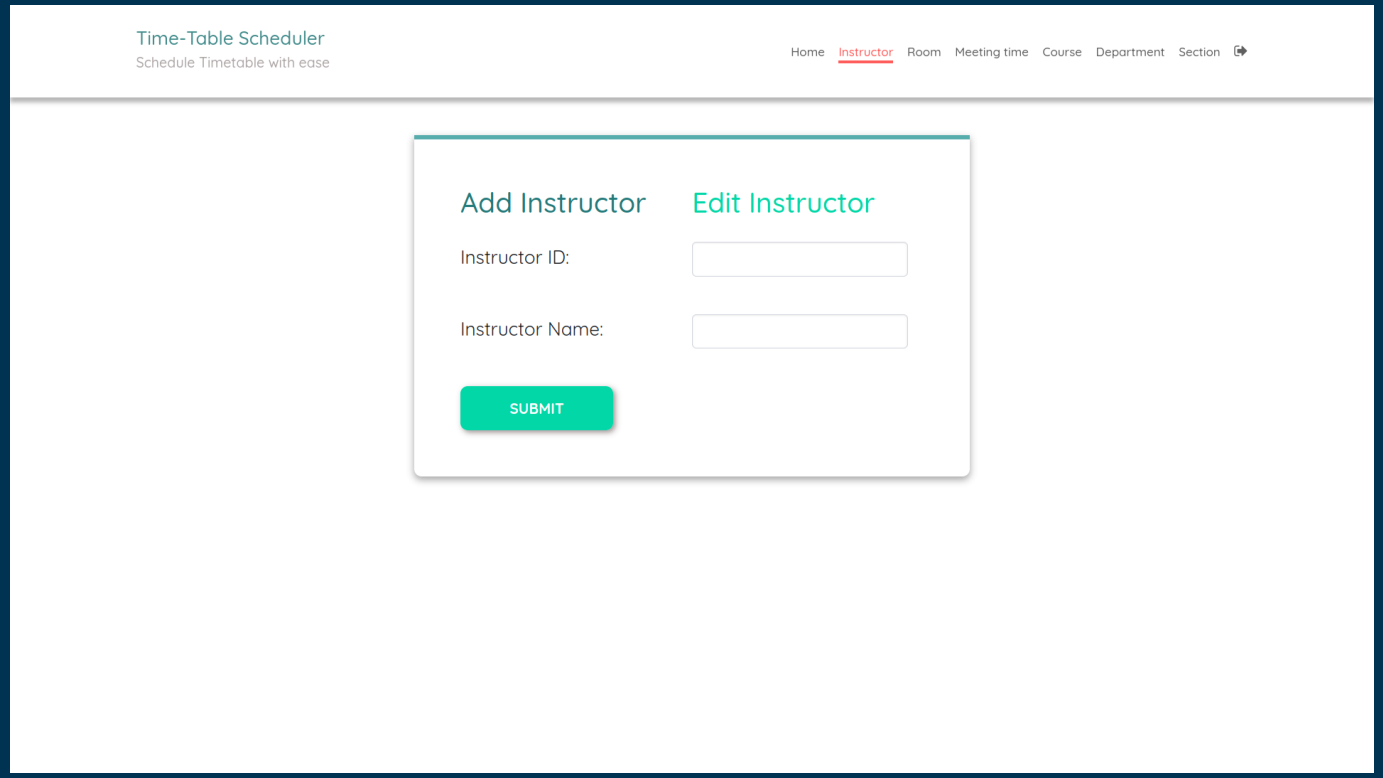
- AFTER FILLING THE INFORMATION IN THE GIVEN WINDOW AND SELECTING GIVEN OPTIONS THE USER WILL GET A OPTION TO GENERATE TIMETABLE.
- AND NOW AFTER FEW SECONDS YOU WILL SEE THE FINAL TIME TABLES ON THE SCREEN IN A ORGANIZED MANNER.
- THE TIMETABLES WILL BE ACCORDING TO SPECIFIC DEPARTMENT, INSTRUCTOR AND SECTION.
- NOW ACCORDING TO YOUR REQUIREMENTS YOU HAVE RECEIVED YOUR TIMETABLE BUT IF YOU WANT TO CHANGE OR UPDATE YOU CAN EASILY DO THAT.

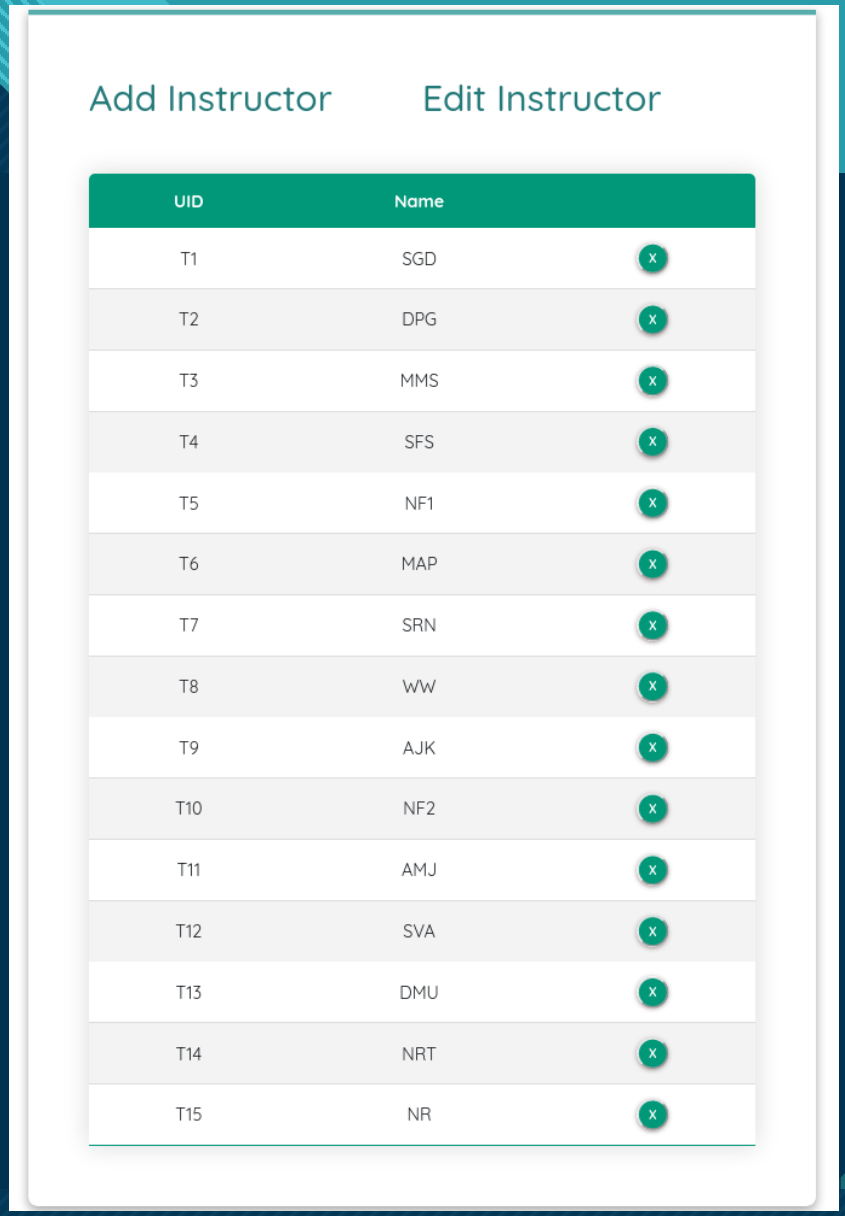
SCREENSHOTS :- LOGIN



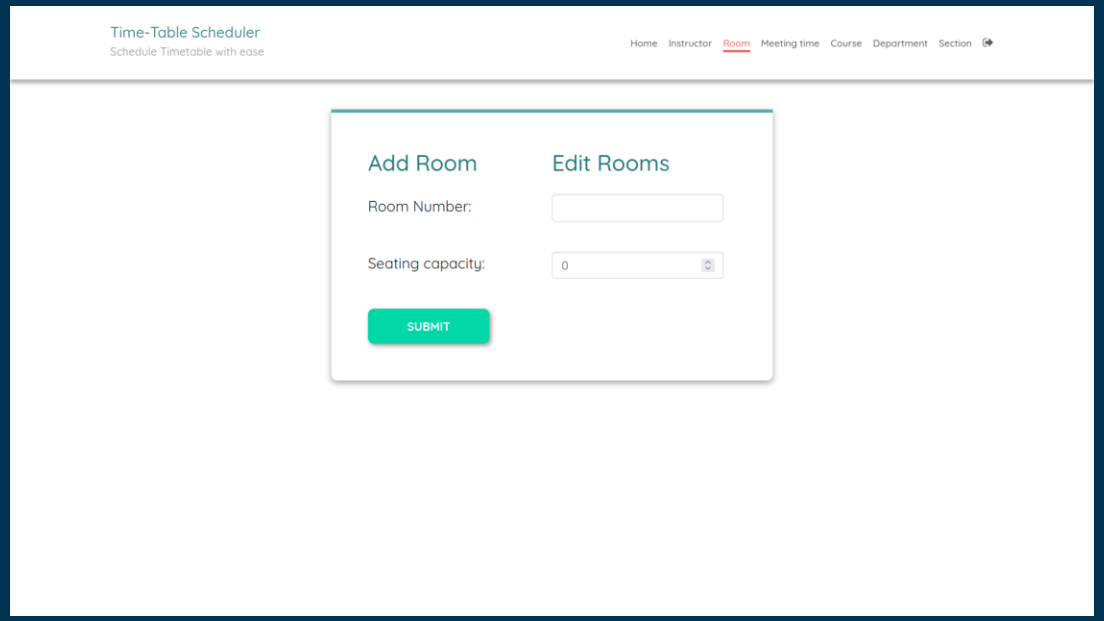


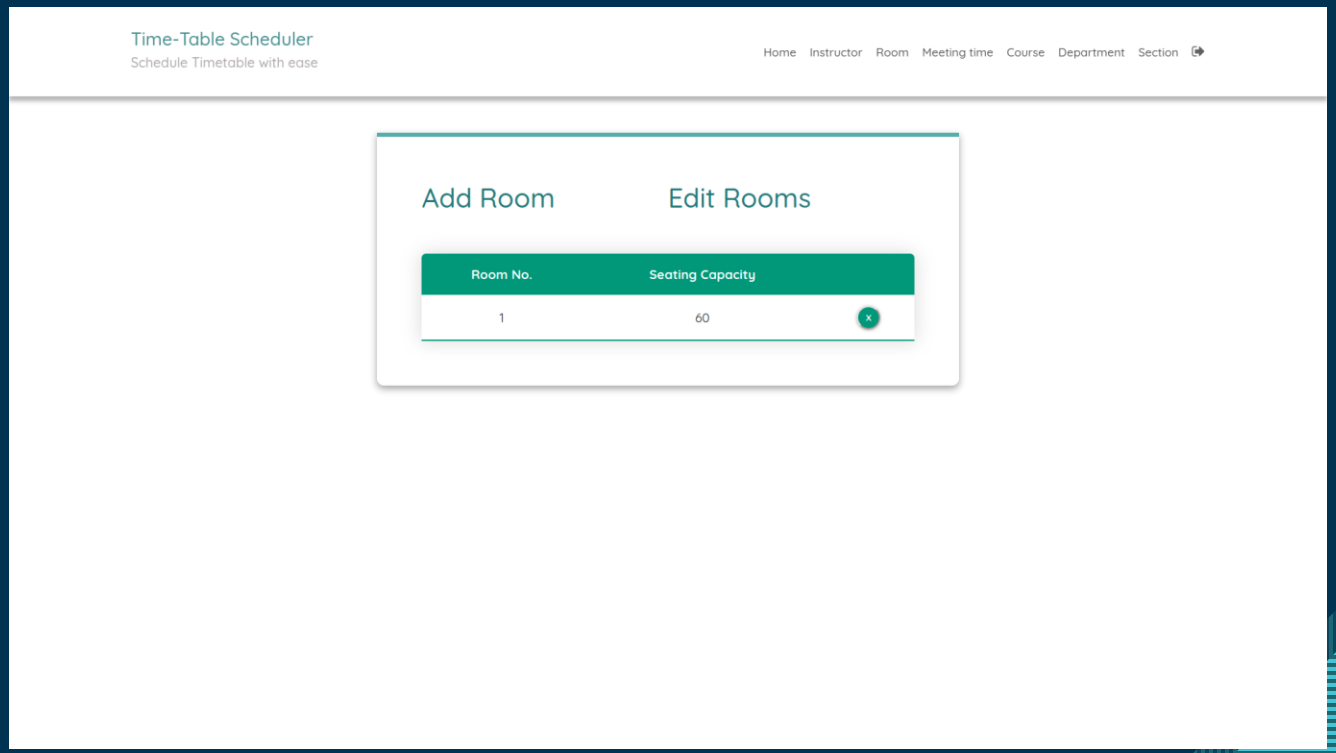
INSERTING INSTRUCTOR



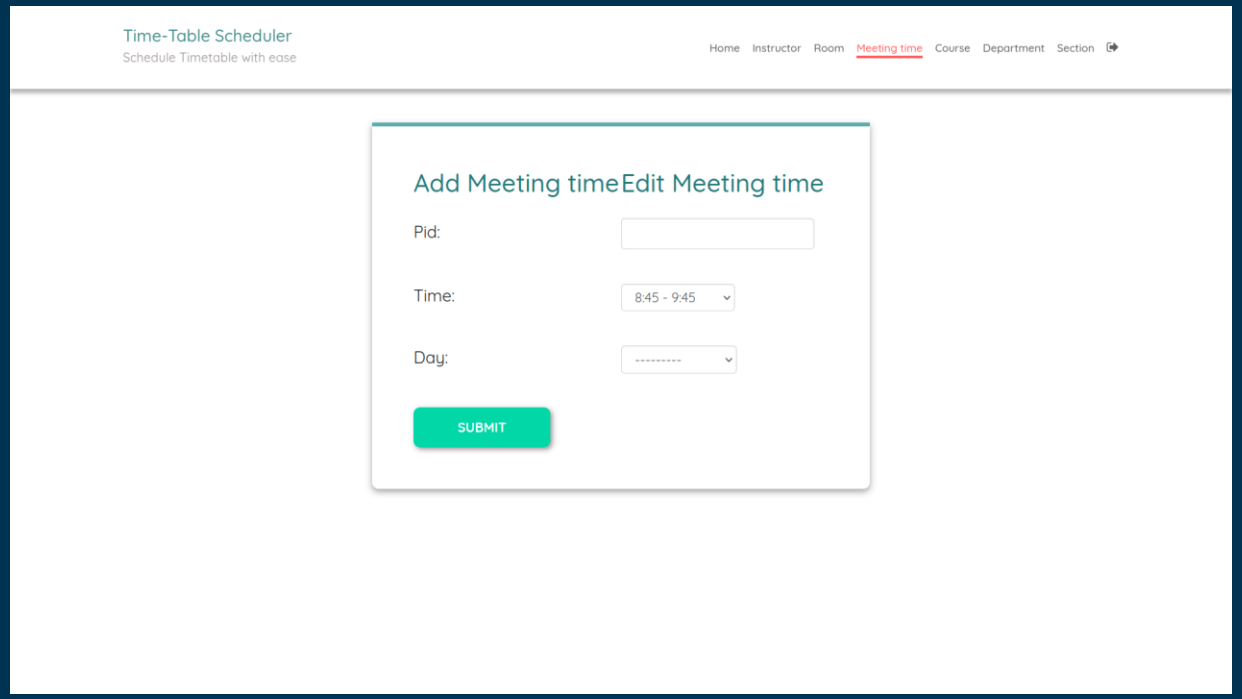


INSERTING ROOM





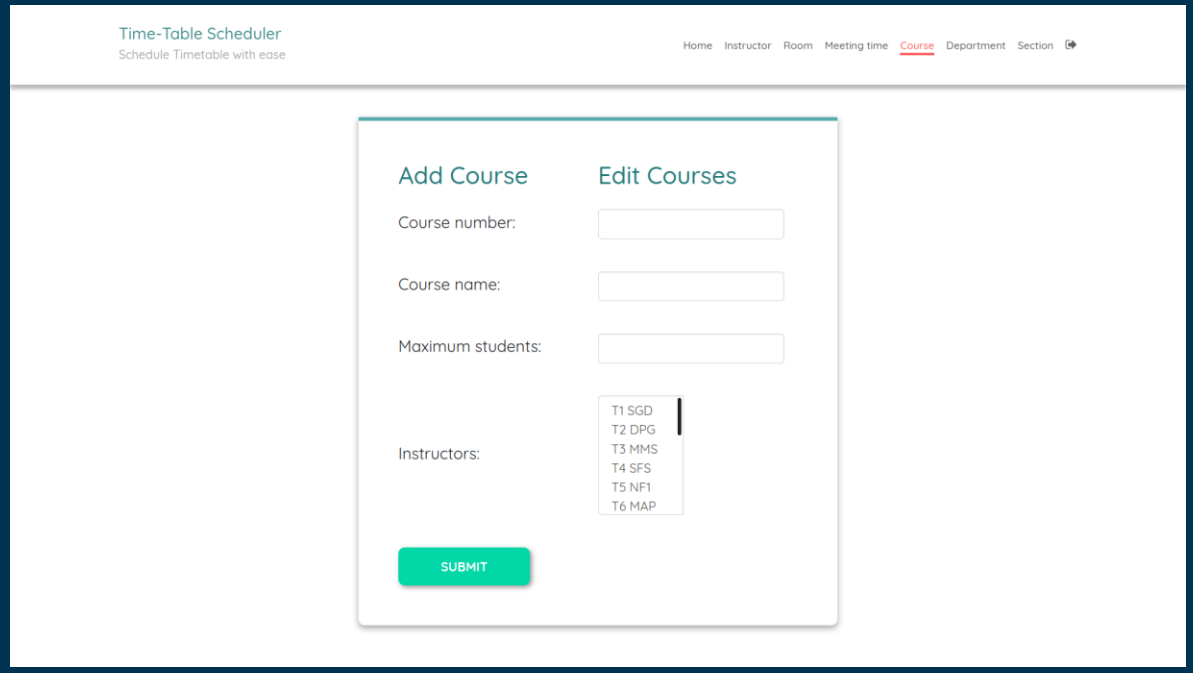
INSERTING MEETING TIME



Add Meeting time Edit Meeting time

PID	Day	Timing	
M1	Monday	8:45 - 9:45	x
M2	Monday	10:00 - 11:00	x
M3	Monday	11:00 - 12:00	x
M4	Monday	1:00 - 2:00	x
M5	Monday	2:15 - 3:15	x
T1	Tuesday	8:45 - 9:45	x
T2	Tuesday	10:00 - 11:00	x
T3	Tuesday	11:00 - 12:00	x
T4	Tuesday	1:00 - 2:00	x
T5	Tuesday	2:15 - 3:15	x
W1	Wednesday	8:45 - 9:45	x
W2	Wednesday	10:00 - 11:00	x
W3	Wednesday	11:00 - 12:00	x
Th1	Thursday	8:45 - 9:45	x
Th2	Thursday	10:00 - 11:00	x
Th3	Thursday	11:00 - 12:00	x
Th4	Thursday	1:00 - 2:00	x
F1	Friday	8:45 - 9:45	x
F2	Friday	10:00 - 11:00	x
F3	Friday	11:00 - 12:00	x
F4	Friday	1:00 - 2:00	x
W4	Wednesday	1:00 - 2:00	x

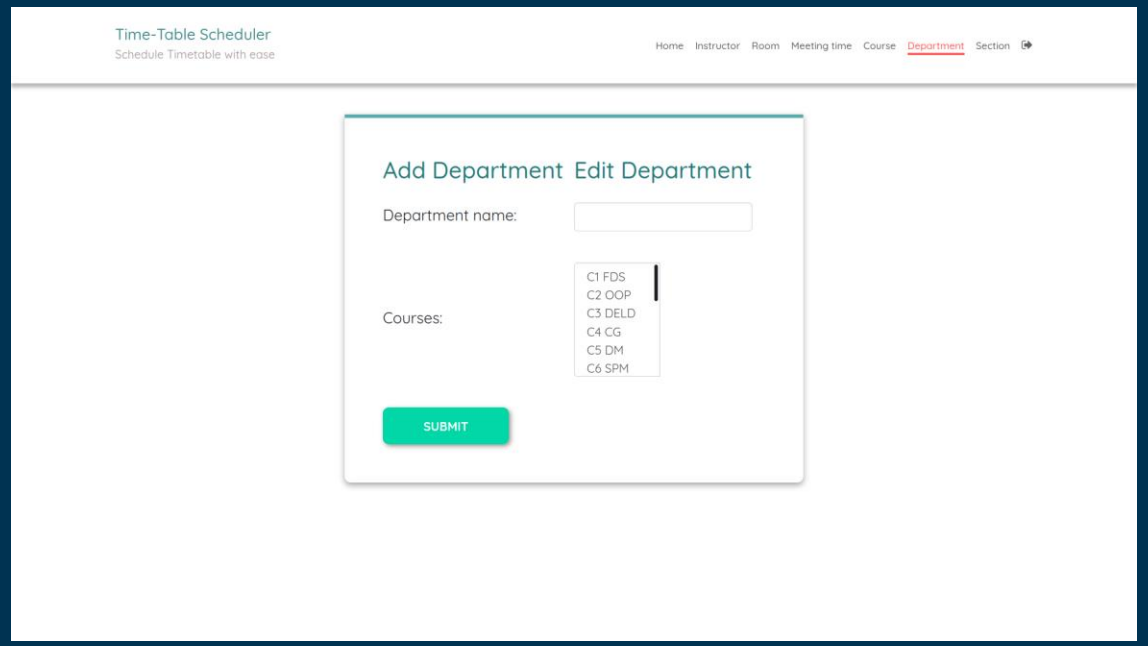
INSERTING COURSE

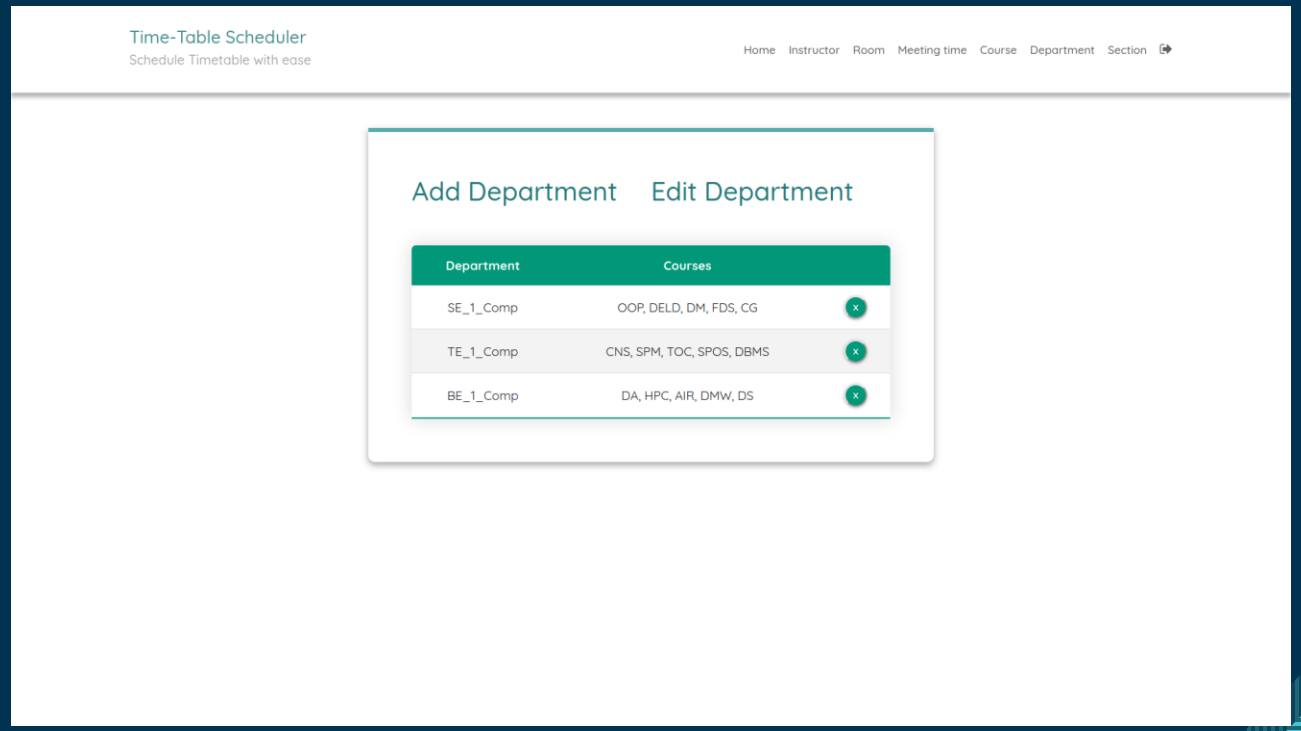


Add Course Edit Courses

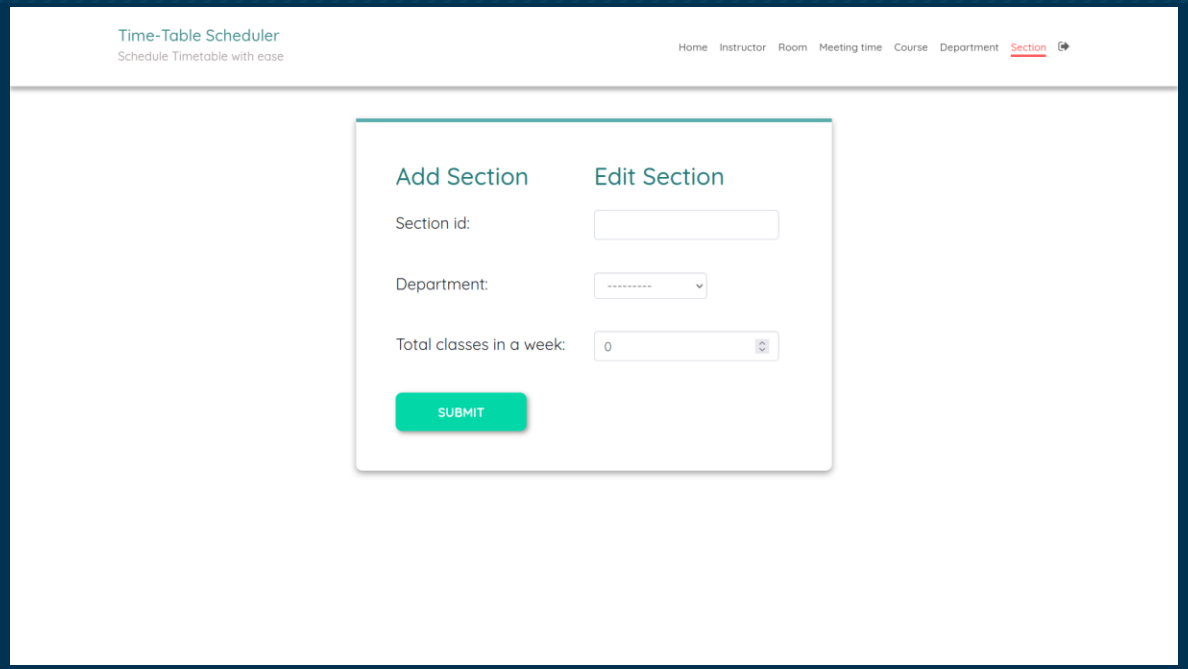
Course Code	Course Name	Max students	Instructors	
C1	FDS	60	SGD	X
C2	OOP	60	DPG	X
C3	DELD	60	MMS	X
C4	CG	60	SFS	X
C5	DM	60	NF1	X
C6	SPM	60	MAP	X
C7	DBMS	60	SRN	X
C8	TOC	60	WW	X
C9	CNS	60	AJK	X
C10	SPOS	60	NF2	X
C11	DA	60	AMJ	X
C12	HPC	60	SVA	X
C13	DMW	60	DMU	X
C14	DS	60	NRT	X
C15	AIR	60	NR	X

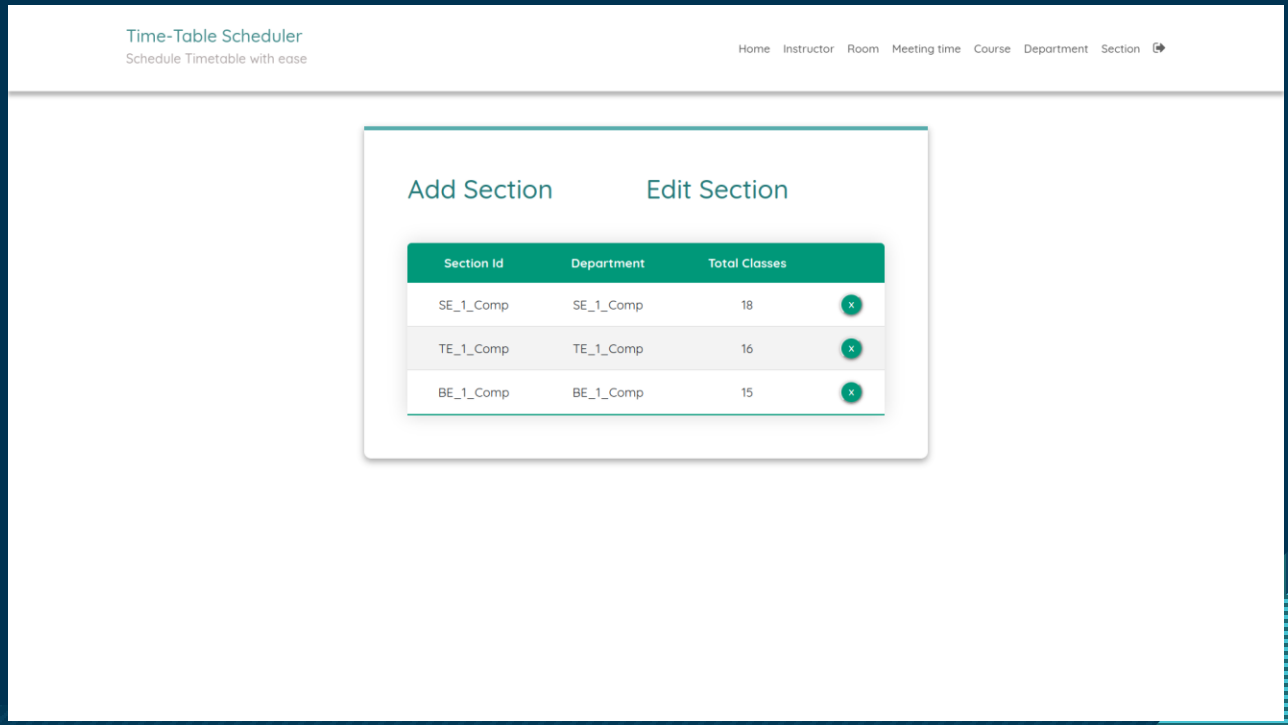
INSERTING DEPARTMENT



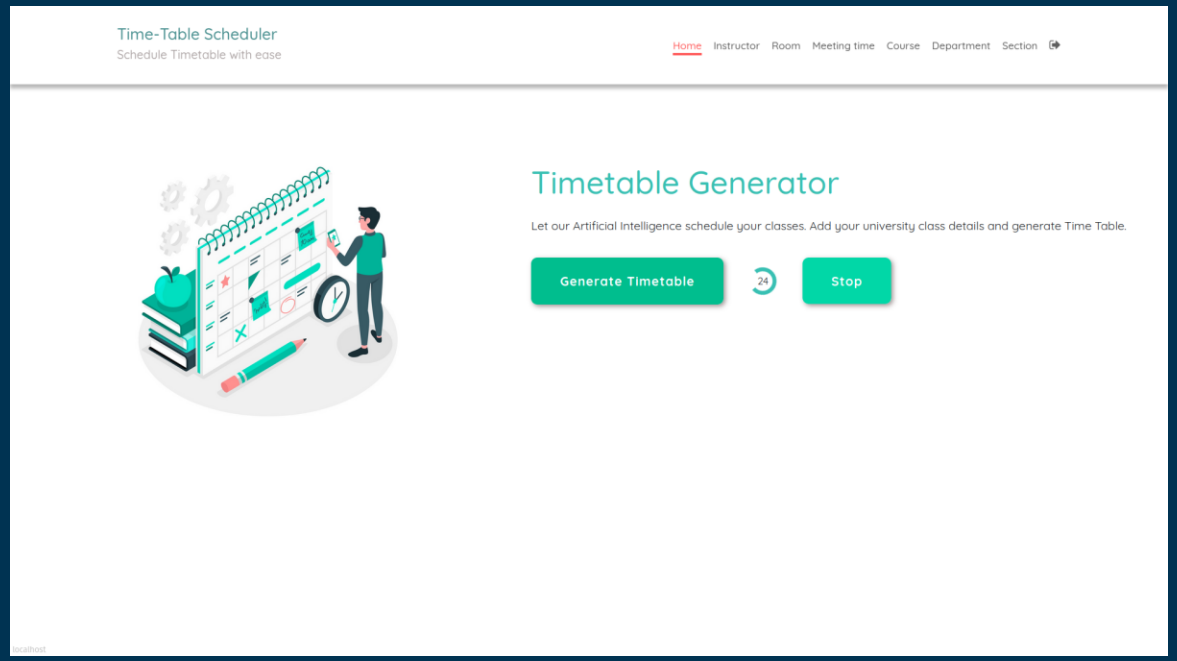


INSERTING SECTION





FINAL



Time-Table Scheduler

Schedule Timetable with ease

Home Instructor Room Meeting time Course Department Section

SE_1_Comp (SE_1_Comp)

Class #	8:45 - 9:45	10:00 - 11:00	11:00 - 12:00	1:00 - 2:00	2:15 - 3:15
Monday	FDS (SGD)	CG (SFS)		OOP (DPG)	DELD (MMS)
Tuesday	OOP (DPG)	CG (SFS)	DELD (MMS)		
Wednesday		FDS (SGD)	OOP (DPG)	DM (NF1)	
Thursday	DM (NF1)	DELD (MMS)	FDS (SGD)	CG (SFS)	
Friday	FDS (SGD)	OOP (DPG)	CG (SFS)	DM (NF1)	

TE_1_Comp (TE_1_Comp)

Class #	8:45 - 9:45	10:00 - 11:00	11:00 - 12:00	1:00 - 2:00	2:15 - 3:15
Monday	SPOS (NF2)	SPM (MAP)	DBMS (SRN)		
Tuesday		DBMS (SRN)	TOC (WW)	CNS (AJK)	SPM (MAP)
Wednesday	CNS (AJK)	DBMS (SRN)		TOC (WW)	
Thursday	SPOS (NF2)	CNS (AJK)		SPM (MAP)	
Friday		TOC (WW)	SPOS (NF2)	DBMS (SRN)	

BE_1_Comp (BE_1_Comp)

Class #	8:45 - 9:45	10:00 - 11:00	11:00 - 12:00	1:00 - 2:00	2:15 - 3:15
Monday	DA (AMJ)	DMW (DMU)	HPC (SVA)	AIR (NR)	DS (NRT)
Tuesday			DA (AMJ)		AIR (NR)
Wednesday		DS (NRT)	HPC (SVA)	DMW (DMU)	
Thursday			DMW (DMU)	DA (AMJ)	
Friday	HPC (SVA)	AIR (NR)		DS (NRT)	

CODE :-

https://github.com/PROxZIMA/TimetableScheduler

SYSTEM TESTING

- Software testing can be stated as the process of verifying and validating that software or application is bug-free, meets the technical requirements as guided by its design and development, and meets the user requirements effectively and efficiently with handling all the exceptional and boundary cases.
- **Unit Testing** is a software testing technique by means of which individual units of software i.e. group of computer program modules, usage procedures and operating procedures are tested to determine whether they are suitable for use or not. It is a testing method using which every independent modules are tested to determine if there are any issue by the developer himself.
- **Integration testing** is the process of testing the interface between two software units or module. It's focus on determining the correctness of the interface. The purpose of the integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

TYPES OF TESTING

- **White box** testing techniques analyze the internal structures the used data structures, internal design, code structure and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing.

- **Integration Test** Case differs from other test cases in the sense it focuses mainly on the interfaces & flow of data/information between the modules. Here priority is to be given for the integrating links rather than the unit functions which are already tested.

Advantages

- Faculty did not need to worry about time clashes.
- Authority now does not need to perform permutation and combination.
- Authority can concentrate on other things rather than wasting their time on preparing Time-Table.
- Substitution Management made easy.
- And one of the most important things, no more paperwork.

Future Work

- MORE FEATURES SUCH AS SCHEDULE PRINT FOR INDIVIDUAL FACULTY ETC. WOULD ALSO BE INVOLVED TO MAKE THIS MORE USEFUL AS A FINAL PRODUCT.
- IN FUTURE EXPORTING TIMETABLE IN VARIOUS FORMATS WILL BE AVAILABLE.
- FASTER PROCESSING OF SCHEDULE AND IMPROVEMENTS IN THE ALGORITHM

CONCLUSION

- IT IS COMPLICATED TASK THAT TO HANDLE MANY FACULTY'S AND ALLOCATING SUBJECTS FOR THEM AT A TIME PHYSICALLY. SO OUR PROPOSED SYSTEM WILL HELP TO OVERCOME THIS DISADVANTAGE. THUS WE CAN PRODUCE TIMETABLE FOR ANY NUMBER OF COURSES AND MULTIPLE SEMESTERS.
- THIS SYSTEM WILL HELP TO CREATE DYNAMIC PAGES SO THAT FOR IMPLEMENTING SUCH A SYSTEM WE CAN MAKE USE OF THE DIFFERENT TOOLS ARE WIDELY APPLICABLE AND FREE TO USE ALSO.

Thank You