PRATHYUSHA ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

for

CS3481-DATABASE MANAGEMENT SYSTEMS LABORATORY

(Regulation 2021, IV Semester)

ACADEMIC YEAR: 2022 – 2023

(Even Semester)

PRATHYUSHA ENGINEERING COLLEGE

PREPARED BY

B.Gunasundari,

Assistant Professor / CSE

PRATHYUSHA ENGINEERING COLLEGE

VISION

To emerge as a premier technical, engineering and management institution in the country by imparting quality education and thus facilitate our students to blossom in to dynamic professional so that they play a vital role for the progress of the nation and for a peaceful co-existence of our fellow human being.

MISSION

Prathyusha Engineering College will strive to emerge as a premier Institution in the country by

- To provide state of art infrastructure facilities
- Imparting quality education and training through qualified, experienced and committed members of the faculty
- Empowering the youth by providing professional leadership
- Developing centers of excellence in frontiers areas of Engineering, Technology and Management
- Networking with Industry, Corporate and Research Organizations
- Promoting Institute-Industry partnership for the peace and prosperity of the nation

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION

Our Vision is to build a strong teaching & research environment in the field of computer science and engineering for developing a team of young dynamic computer science engineers, researchers, future entrepreneurs who are adaptive to respond to the challenges of 21st century. Our commitment lies in producing disciplined human individuals, capable of contributing solutions to solve problems faced by our society.

MISSION

- To provide a quality undergraduate and graduate education in both the theoretical and applied foundations of computer science and engineering.
- To train the students to effectively apply this education to solve real-world problems, thus amplifying their potential for lifelong high-quality careers and gives them a competitive advantage in the ever-changing and challenging global work environment of the 21st century.
- To initiate collaborative real-world industrial projects with industries and academic institutions to inculcate facilities in the arena of Research & Development
- To prepare them with an understanding of their professional and ethical responsibilities

PRATHYUSHA ENGINEERING COLLEGE

PROGRAMME EDUCATIONAL OBJECTIVES

PEO-1: To train the graduates to be excellent in computing profession by updating technical skill-sets and applying new ideas as the technology evolves.

PEO-2: To enable the graduates to excel in professional career and /or higher education by acquiring knowledge in mathematical, computing and engineering principles.

PEO-3: To enable the graduates, to be competent to grasp, analyze, design, and create new products and solutions for the real time problems that are technically advanced economically feasible and socially acceptable

PEO- 4: To enable the graduates to pursue a productive career as a member of multi-disciplinary and cross-functional teams, with an appreciation for the value of ethic and cultural diversity and an ability to relate engineering issues to broader social context.

PROGRAMME OUTCOMES AND PROGRAMME SPECIFIC OUTCOMES

1. An ability to apply knowledge of computing, mathematics, science and engineering fundamentals appropriate to the discipline.
2. An ability to analyze a problem, and identify and formulate the computing requirements appropriate to its solution.
3. An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs with appropriate consideration for public health and safety, cultural, societal and environmental considerations.
4. An ability to design and conduct experiments, as well as to analyze and interpret data.
5. An ability to use current techniques, skills, and modern tools necessary for computing practice.
6. An ability to analyze the local and global impact of computing on individuals, organizations, and society.
7. Knowledge of contemporary issues.
8. An understanding of professional, ethical, legal, security and social issues and responsibilities.
9. An ability to function effectively individually and on teams, including diverse and multidisciplinary, to accomplish a common goal.
10. An ability to communicate effectively with a range of audiences.
11. Recognition of the need for and an ability to engage in continuing professional development.
12. An understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects.

PRATHYUSHA ENGINEERING COLLEGE

PROGRAMME SPECIFIC OBJECTIVES (PSO's)

A graduate of the Computer Science and Engineering Program will able,

PSO1: To Analyze, Design and Develop computer programs / Applications in the areas related to Web-Technologies, Networking, Algorithms, Cloud Computing, Data analytics, Computer Vision, Cyber-Security and Intelligent Systems for efficient design of Computer-based and Mobile-based systems of varying complexity.

PSO2: To use modern software tools (like NS2, MATLAB, OpenCV, etc.) for designing, simulating, analyzing and generating experimental results for real-time problems and case studies

PSO3: To Apply Software Engineering practices and strategies for developing Projects related to emerging technologies.

PRATHYUSHA ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDEX

S. NO	NAME OF EXPERIMENT	PAGE NO
1	DDL AND DML	1
2	REFERENTIAL INTEGRITY	9
3	'WHERE' CLAUSE AND AGGREGATE FUNCTIONS	11
4	SUB QUERIES AND SIMPLE JOIN OPERATIONS	15
5	JOINS	18
6	FUNCTIONS AND STORED PROCEDURES	22
7	DCL AND TCL COMMANDS	24
8	TRIGGERS	26
9	VIEW AND INDEX	27
10	XML DATABASE	28
11	NOSQL DATABASE TOOLS	31
12	EMPLOYEE MANAGEMENT SYSTEM	37
13	BANK MANAGEMENT SYSTEM	42
ADDITIONAL EXPERIMENTS		
14	PL/SQL PROCEDURE TO DISPLAY FIBONACCI SERIES	43
15	PL/SQL PROCEDURE TO DISPLAY STUDENT INFORMATION USING ARRAY	44

PRATHYUSHA ENGINEERING COLLEGE

Exp. No.1

DDL and DML commands.

Aim :

To Create a database table, add constraints (primary key, unique, check, Not null), insert rows,update and delete rows using SQL DDL and DML commands.

Procedure:

1. Create database, show databases and use database command:

syntax:
CREATE DATABASE databasename;
Eg.
CREATE DATABASE testDB;
Show databases;
Use testDB;

The Create Table Command: - it defines each column of the table uniquely. Each column has minimum of three attributes, a name , data type and size.

Syntax:

Create table <table name> (<col1> <datatype>(<size>),<col2> <datatype><size>)); Ex:create table emp(empno INT(4) primary key, ename char(10));

2. Modifying the structure of tables. a) Add new columns

Syntax:
Alter table <tablename> add(<new col><datatype(size),<new col>datatype(size));
Ex:alter table emp add(sal INT(7));

3. Dropping a column from a table.

Syntax:
Alter table <tablename> drop column <col>;
Ex:alter table emp drop column sal;

4. Modifying existing columns.

Syntax:
Alter table <tablename>modify <col><newdatatype>(<newsize>);
Ex:alter table emp modify ename varchar(15);

5. Renaming the tables

Syntax:
Rename <oldtable> to <new table>;
Ex:rename emp to emp1;

6. truncating the tables.

Syntax:
Truncate table <tablename>;
Ex:trunc table emp1;

PRATHYUSHA ENGINEERING COLLEGE

7. Destroying tables.

Syntax:

Drop table <tablename>;

Ex:drop table emp;

CREATION OF TABLE:

SYNTAX:

create table<tablename>(column1 datatype,column2 datatype...);

EXAMPLE:

create table std(sno INT(5),sname varchar(20),age INT(5),sdob date,sm1 INT(4,2),sm2 INT(4,2),sm3 INT(4,4));

Table created.

insert into std values(101,'AAA',16,'03-jul-88',80,90,98);

1 row created.

insert into std values(102,'BBB',18,'04-aug-89',88,98,90);

1 row created.

OUTPUT:

Select * from std;

SNO	SNAME	AGE	SDOB	SM1	SM2	SM3
101	AAA	16	03-jul-88	80	90	98
102	BBB	18	04-aug-89	88	98	90

ALTER TABLE WITH ADD:

create table student(id INT(5),name varchar(10),game varchar(20));

Table created.

insert into student values(1,'mercy','cricket');

1 row created.

SYNTAX:

alter table<tablename>add(col1 datatype,col2 datatype..);

EXAMPLE:

alter table student add(age INT(4));

insert into student values(2,'sharmi','tennis',19);

OUTPUT:

ALTER: select * from student;

ID NAME GAME

1 Mercy Cricket

PRATHYUSHA ENGINEERING COLLEGE

ADD: select * from student;
ID NAME GAME AGE
1 Mercy cricket
2 Sharmi Tennis 19

ALTER TABLE WITH MODIFY:
SYNTAX:
Alter table<tablename>modify col1 datatype;
EXAMPLE:

alter table student modify id INT(6);
desc command:
desc student;

NAME NULL? TYPE
Id INT(6)
Name Varchar(20)
Game Varchar(25)
Age INT(4)

DROP:

SYNTAX: drop table<tablename>;
EXAMPLE:
drop table student;
Table dropped.

TRUNCATE TABLE
SYNTAX: TRUNCATE TABLE <TABLE NAME>;
Example: Truncate table stud;

DESC
Example: desc emp;
Name Null? Type
--------------------------------- --------
EmpNo NOT NULL INT(5)
EName VarChar(15)
Job NOT NULL Char(10)
DeptNo NOT NULL INT(3)
PHONE_NO INT (10)

CONSTRAINTS:
Create table tablename (column_name1 data_ type constraints, column_name2 data_ type constraints …)

Example:
Create table Emp (EmpNo INT(5), EName VarChar(15), Job Char(10) constraint un unique, DeptNo INT(3) CONSTRAINT FKey2 REFERENCES DEPT(DeptNo));

PRATHYUSHA ENGINEERING COLLEGE

Create table stud (sname varchar(20) not null, rollno INT(10) not null,dob date not null);

DOMAIN INTEGRITY

Example: Create table cust(custid INT(6) not null, name char(10));
Alter table cust modify (name not null);

CHECK CONSTRAINT

Example: Create table student (regno INT (6), mark INT (3) constraint b check (mark >=0 and mark <=100)); Alter table student add constraint b2 check (length(regno<=4));

ENTITY INTEGRITY

a) Unique key constraint
Example: Create table cust(custid INT(6) constraint unique, name char(10)); Alter table cust add(constraint c unique(custid));
b) Primary Key Constraint
Example: Create table stud(regno INT(6) constraint primary key, name char(20));

Queries:

Q1. Create a table called EMP with the following structure.
Name Type
---------- ----------------------
EMPNO INT(6)
ENAME VARCHAR(20)
JOB VARCHAR(10)
DEPTNO INT(3)
SAL INT(7,2)
Allow NULL for all columns except ename and job.
Ans:
create table emp(empno INT(6),ename varchar(20)not null,job varchar(10) not null, deptno INT(3),sal INT(7,2));
Table created.

Q2: Add a column experience to the emp table.
experience numeric null allowed.
Ans:
alter table emp add(experience INT(2)); Table altered.

Q3: Modify the column width of the job field of emp table.
Ans: alter table emp modify(job varchar(12)); Table altered.
lter table emp modify(job varchar(13));
Table altered.

Q4: Create dept table with the following structure.
Name Type
------------ ---------------------
DEPTNO INT(2)
DNAME VARCHAR(10)

PRATHYUSHA ENGINEERING COLLEGE

LOC VARCHAR(10)
Deptno as the primarykey

Ans:
create table dept(deptno INT(2) primary key,dname varchar(10),loc varchar(10));
Table created.

Q5: create the emp1 table with ename and empno, add constraints to check the empno value while entering (i.e) empno > 100.
Ans:
create table emp1(ename varchar(10),empno INT(6) constraint check(empno>100));
Table created.

Q6: drop a column experience to the emp table.
Ans:
alter table emp drop column experience; Table altered.

Q7: Truncate the emp table and drop the dept table
Ans:
truncate table emp; Table truncated.

DML COMMANDS

DML commands are the most frequently used mysql commands and is used to query and manipulate the existing database objects. Some of the commands are Insert, Select, Update, Delete.

Insert Command : This is used to add one or more rows to a table. The values are separated by commas and the data types char and date are enclosed in apostrophes. The values must be entered in the same order as they are defined.

Select Commands : It is used to retrieve information from the table. It is generally referred to as querying the table. We can either display all columns in a table or only specify column from the table.

Update Command :It is used to alter the column values in a table. A single column may be updated or more than one column could be updated.

Delete command :After inserting row in a table we can also delete them if required. The delete command consists of a from clause followed by an optional where clause.

Q1: Insert a single record into dept table.
Ans:
insert into dept values (1,'IT','Tholudur');
1 row created.

PRATHYUSHA ENGINEERING COLLEGE

Q2: Insert more than a record into emp table using a single insert command.
Ans: insert into emp values(&empno,'&ename','&job',&deptno,&sal);
Enter value for empno: 1
Enter value for ename: Mathi
Enter value for job: AP
Enter value for deptno: 1
Enter value for sal: 10000
old 1: insert into emp values(&empno,'&ename','&job',&deptno,&sal)
new 1: insert into emp values(1,'Mathi','AP',1,10000)
1 row created.
/
Enter value for empno: 2
Enter value for ename: Arjun
Enter value for job: ASP
Enter value for deptno: 2
Enter value for sal: 12000
old 1: insert into emp values(&empno,'&ename','&job',&deptno,&sal)
new 1: insert into emp values(2,'Arjun','ASP',2,12000)
1 row created.
/
Enter value for empno: 3
Enter value for ename: Gugan
Enter value for job: ASP
Enter value for deptno: 1
Enter value for sal: 12000
old 1: insert into emp values(&empno,'&ename','&job',&deptno,&sal)
new 1: insert into emp values(3,'Gugan','ASP',1,12000)
1 row created.

PRATHYUSHA ENGINEERING COLLEGE

Q3: Update the emp table to set the salary of all employees to Rs15000/- who are working as ASP
Ans: select * from emp;

EMPNO ENAME JOB DEPTNO SAL
---------- -------------------- ------------- ---------- ----------
1 Mathi AP 1 10000
2 Arjun ASP 2 12000
3 Gugan ASP 1 12000

update emp set sal=15000 where job='ASP'; 2 rows updated.
select * from emp;
EMPNO ENAME JOB DEPTNO SAL
---------- -------------------- ------------- ---------- ----------
1 Mathi AP 1 10000
2 Arjun ASP 2 15000
3 Gugan ASP 1 15000
Q4: Create a pseudo table employee with the same structure as the table emp and insert rows into the table using select clauses.
Ans: create table employee as select * from emp;

Table created.
SHOW COLUMNS FROM `employee`;
Name Null? Type
--------------------------------------- -------- ----------------------------
EMPNO INT(6)
ENAME NOT NULL VARCHAR(20)
JOB NOT NULL VARCHAR(13)
DEPTNO INT(3)
SAL INT(7,2)

Q5: select employee name, job from the emp table
Ans: select ename, job from emp;
ENAME JOB
-------------------- -------------
Mathi AP
Arjun ASP
Gugan ASP
Karthik Prof
Akalya AP
suresh lect
6 rows selected.

Q6: Delete only those who are working as lecturer
Ans: select * from emp;
EMPNO ENAME JOB DEPTNO SAL
---------- ------------------- ------------- ---------- ----------
1 Mathi AP 1 10000
2 Arjun ASP 2 15000
3 Gugan ASP 1 15000
4 Karthik Prof 2 30000
5 Akalya AP 1 10000
6 suresh lect 1 8000
6 rows selected.
delete from emp where job='lect';
1 row deleted.
select * from emp;
EMPNO ENAME JOB DEPTNO SAL
--------- ------------------- ------------- ---------- ----------

1 Mathi AP 1 10000
2 Arjun ASP 2 15000
3 Gugan ASP 1 15000
4 Karthik Prof 2 30000
5 Akalya AP 1 10000

Q7: List the records in the emp table orderby salary in ascending order.
Ans: select * from emp order by sal;
EMPNO ENAME JOB DEPTNO SAL
---------- ------------------- ------------- ---------- ----------

PRATHYUSHA ENGINEERING COLLEGE

1 Mathi AP 1 10000
5 Akalya AP 1 10000
2 Arjun ASP 2 15000
3 Gugan ASP 1 15000
4 Karthik Prof 2 30000

Q8: List the records in the emp table orderby salary in descending order.
Ans: select * from emp order by sal desc;
EMPNO ENAME JOB DEPTNO SAL
---------- ------------------- ------------- ---------- ----------
4 Karthik Prof 2 30000
2 Arjun ASP 2 15000
3 Gugan ASP 1 15000
1 Mathi AP 1 10000
5 Akalya AP 1 10000

Q9: Display only those employees whose deptno is 30.
Solution: Use SELECT FROM WHERE syntax.
Ans: select * from emp where deptno=1;
EMPNO ENAME JOB DEPTNO SAL
---------- ------------------- ------------- ---------- ----------
1 Mathi AP 1 10000
3 Gugan ASP 1 15000
5 Akalya AP 1 10000

Q10: Display deptno from the table employee avoiding the duplicated values.
Solution:
1. Use SELECT FROM syntax.
2.Select should include distinct clause for the deptno.
Ans: select distinct deptno from emp;
DEPTNO

1
2

Result:
Thus the table created and constraints (primary key, unique, check, Not null) are added. Rows are inserted,updated and deleted using SQL DDL and DML commands.

PRATHYUSHA ENGINEERING COLLEGE

Exp. no. 2

REFERENTIAL INTEGRITY

Aim:

To create a set of tables, add foreign key constraints and incorporate referential integrity.

Procedure:

Constraints are the business Rules which are enforced on the data being storedin a table are called *Constraints*

TYPES OF CONSTRAINTS:

1. Not null
2. Unique
3. Check
4. Primary key
5. Foreign key/references

1. NOT NULL:

a) Not null constraint at column level.

Syntax:

<col><datatype>(size)not null

SQL > create table emp(e_id varchar(5) NOT NULL,e_name varchar(10), e_design varchar(10),dept varchar(10),mgr varchar(10),salary number(10));

2. UNIQUE :

Unique constraint at column level.

Syntax: <col><datatype>(size)unique

Ex:-

SQL > create table depositor(customer_name varchar(10),acc_no number(15) UNIQUE, brach_name
varchar(10));

Unique constraint at table level:

Syntax:

Create table tablename(col=format,col=format,unique(<col1>,<col2>));

Ex:-

SQL > create table depositor1(customer_name varchar(10),acc_no number(15), brach_name varchar(10),UNIQUE(acc_no));

3. PRIMARY KEY:

Primary key constraint at column level

Syntax:

<col><datatype>(size)primary key;

Ex:-

PRATHYUSHA ENGINEERING COLLEGE

SQL> create table customer(customer_id number (5) PRIMARY KEY, customer_name varchar(10),customer_street varchar(10),brach_name varchar(10));
Primary key constraint at table level.
Syntax:
Create table tablename(col=format,col=format primary key(col1>,<col2>);
Ex:-
SQL > create table customer1(customer_id number (5),customer_name varchar(10),customer_street varchar(10),brach_name varchar(10),PRIMARY KEY(customer_id));

4. CHECK:

Check constraint constraint at column level.
Syntax: <col><datatype>(size) check(<logical expression>)
Ex:-
create table loan(loan_no varchar(10),customer_name varchar(10), balance number (10) CHECK(balance>1000));
Check constraint constraint at table level.
Syntax: check(<logical expression>)
Ex:-
create table loan1(loan_no varchar(10),customer_name varchar(10), balance number (10), CHECK(balance>1000));

5. FOREIGN KEY:

Foreign key constraint at column level.
Syntax:
Column_name Datatype(size) REFERENCES parent_table_name (parent_column_name)
Ex:-
CREATE TABLE books (book_id NUMBER(3), book_title VARCHAR2(30), book_price NUMBER(3), book_author_id NUMBER(3) REFERENCES author(author_id));
Foreign key constraint at table level

Syntax:
CONSTRAINT constraint_name FOREIGN KEY(child_table_column) REFERENCES Parent_table_name(parent_table_column)
Ex:-
CREATE TABLE books (book_id NUMBER(3) CONSTRAINT bok_bi_pk PRIMARY KEY, book_title VARCHAR2(30), book_price NUMBER(3), book_author_id NUMBER(3), CONSTRAINT bok_ai_fk FOREIGN KEY (book_author_id) REFERENCES author(author_id));

PRATHYUSHA ENGINEERING COLLEGE

Result:
Thus the table is created and foreign key constraints and incorporate referential integrity are added.

Exp. No.3:

'WHERE' CLAUSE AND AGGREGATE FUNCTIONS

Aim:

To Query the database tables using different 'where' clause conditions and also implement aggregate functions.

Procedure:

Where clause:

The SQL **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it returns a specific value from the table. You should use the WHERE clause to filter the records and fetching only the necessary records.

The WHERE clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc., which we would examine in the subsequent chapters.

Syntax

The basic syntax of the SELECT statement with the WHERE clause is as shown below.

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

You can specify a condition using the comparison or logical operators like >, <, =, **LIKE, NOT**, etc. The following examples would make this concept clear.

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

PRATHYUSHA ENGINEERING COLLEGE

The following code is an example which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 −

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000;
```

This would produce the following result −

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
```

```
| 7 | Muffy    | 10000.00 |
+----+----------+----------+
```

The following query is an example, which would fetch the ID, Name and Salary fields from the CUSTOMERS table for a customer with the name **Hardik**.
Here, it is important to note that all the strings should be given inside single quotes ('').
Whereas, numeric values should be given without any quote as in the above example.

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE NAME = 'Hardik';
```

This would produce the following result −

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  5 | Hardik   |  8500.00 |
+----+----------+----------+
```

The SQL **AND** & **OR** operators are used to combine multiple conditions to narrow data in an SQL statement. These two operators are called as the conjunctive operators.
These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

The AND Operator

The **AND** operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

Syntax

The basic syntax of the AND operator with a WHERE clause is as follows −

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] AND [condition2]...AND [conditionN];
```

You can combine N number of conditions using the AND operator. For an action to be taken by the SQL statement, whether it be a transaction or a query, all conditions separated by the AND must be TRUE.

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is an example, which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 and the age is less than 25 years −

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 AND age < 25;
```

PRATHYUSHA ENGINEERING COLLEGE

This would produce the following result −

```
+----+-------+----------+
| ID | NAME  | SALARY   |
+----+-------+----------+
|  6 | Komal |  4500.00 |
|  7 | Muffy | 10000.00 |
+----+-------+----------+
```

The OR Operator

The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

Syntax

The basic syntax of the OR operator with a WHERE clause is as follows −

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] OR [condition2]...OR [conditionN]
```

You can combine N number of conditions using the OR operator. For an action to be taken by the SQL statement, whether it be a transaction or query, the only any ONE of the conditions separated by the OR must be TRUE.

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code block hasa query, which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 OR the age is less than 25 years.

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 OR age < 25;
```

This would produce the following result −

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  3 | kaushik  |  2000.00 |
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

PRATHYUSHA ENGINEERING COLLEGE

Aggregative functions:
In addition to simply retrieving data, we often want to perform some computation or summarization. SQL allows the use of arithmetic expressions. We now consider a powerful class of constructs for computing aggregate values such as MIN and SUM.
1. Count: COUNT following by a column name returns the count of tuple in that column. If DISTINCT keyword is used then it will return only the count of unique tuple in the column. Otherwise, it will return count of all the tuples (including duplicates) count (*) indicates all the tuples of the column.
Syntax: COUNT (Column name)
Example: SELECT COUNT (Sal) FROM emp;

2. SUM: SUM followed by a column name returns the sum of all the values in that column.
Syntax: SUM (Column name)
Example: SELECT SUM (Sal) From emp;
3. AVG: AVG followed by a column name returns the average value of that column values.
Syntax: AVG (n1, n2...)
Example: Select AVG (10, 15, 30) FROM DUAL;
4. MAX: MAX followed by a column name returns the maximum value of that column.
Syntax: MAX (Column name)
Example: SELECT MAX (Sal) FROM emp;
SQL> select deptno, max(sal) from emp group by deptno;
DEPTNO MAX (SAL)
SQL> select deptno, max (sal) from emp group by deptno having max(sal)<3000;
DEPTNO MAX(SAL)
30 2850
5. MIN: MIN followed by column name returns the minimum value of that column.
Syntax: MIN (Column name)
Example: SELECT MIN (Sal) FROM emp;
SQL>select deptno,min(sal) from emp group by deptno having min(sal)>1000;
DEPTNO MIN (SAL)
10 1300

Result:
Thus the database tables are queried using different 'where' clause conditions and also implementaggregate functions.

PRATHYUSHA ENGINEERING COLLEGE

Exp. No.4

SUB QUERIES AND SIMPLE JOIN OPERATIONS

Aim:

To Query the database tables and explore sub queries and simple join operations.

Procedure:

sub queries:

Table 1:

Create table department (deptname varchar(20), building varchar (15), budget INT(12,2), primary key(deptname));

Table created.

Insert into department values('ECE','block1',70000);
Insert into department values('CSE','block2',70000);
Insert into department values('EEE','block3',70000);

Table 2:

create table course (courseid varchar(8) primary key, title varchar(50), deptname varchar(20), credits INT(2), foreign key (deptname) references department) ;

Table created.

Insert into course values('cs101','python','CSE',4);
Insert into course values('cs102','java','CSE',4);
Insert into course values('ec102','Electronics circuits','ECE',3);
Insert into course values('ec202','Microprocessor','ECE',3);

Table 3:

create table instructor (ID varchar(5), name varchar(20) not null, deptname varchar(20), salary INT(8,2), primary key (ID), foreign key (deptname) references department);

insert into instructor values ('1002', 'Sumanth', 'ECE', 66000);
insert into instructor values ('1001', 'Sumitha', 'CSE', 56000);
insert into instructor values ('1007', 'Malar', 'CSE', 96000);
insert into instructor values ('1004', 'Mani', 'ECE', 36000);

Table 4:

Create table section (courseid varchar(8), secid varchar(8), semester varchar(6), year INT(4), building varchar(15) , roomnumner varchar(7), timeslot varchar(4), **primary key** (*courseid*, *secid*, *semester*, *year*), **foreign key** (*courseid*) **references** *course*);

Insert into section values('cs101','A','odd',2017,'block2','111','11');
Insert into section values('cs101','A','even',2018,'block2','222','5');
Insert into section values('cs102','A','even',2016,'block2','77','8');
Insert into section values('ec202','A','even',2016,'block1','47','8');
Insert into section values('ec202','B','odd',2017,'block1','47','8');

PRATHYUSHA ENGINEERING COLLEGE

Insert into section values('ec202','C','even',2018,'block1','47','8');

Table 5:

create table teaches (ID varchar(5), courseid varchar(8), secid varchar(8), semester varchar(6), year INT(4), primary key (ID, courseid, secid, semester, year), foreign key (ID) references instructor, foreign key (courseid, secid, semester, year) references section);

Insert into teaches values('1001', 'cs101','A','odd',2017);
Insert into teaches values('1007', 'cs102','A','even',2016);
Insert into teaches values('1002', 'ec202','A','even',2016);

Questions:

- Find the names of all departments with instructor, and remove duplicates

select distinct dept_name from instructor;

- To find all instructors in CSE with salary > 80000
  select name from instructor where dept_name = 'Comp. Sci.' and salary > 80000;
- Find the Cartesian product instructor X teaches
  select *from instructor, teaches;
- Find the course ID, semester, year and title of each course offered by the Comp. Sci. department

  select section.courseid, semester, year, title from section, course where section.courseid = course.courseid and deptname = 'CSE';
- List the names of instructors along with the course ID of the courses that they taught.

  select name, courseid from instructor, teaches where instructor.ID = teaches.ID;

select name, courseid from instructor natural join teaches;

- Find the names of all instructors who have a higher salary than some instructor in 'Comp. Sci'.
  select distinct T. name from instructor as T, instructor as S where T.salary > S.salary and S.deptname = 'CSE'
  - Find the names of all instructors whose name includes the substring "dar".

select name from instructor where name like '%dar%';

- List in alphabetic order the names of all instructors

select distinct name from instructor order by name;

- Find the names of all instructors with salary between
  $90,000 and $100,000 (that is, □ $90,000 and □ $100,000)

select name from instructor where salary between 90000 and 100000;

PRATHYUSHA ENGINEERING COLLEGE

nested quries:

Q1: Display all employee names and salary whose salary is greater than minimum salary of the company and job title starts with _M‘.
Solution:
1. Use select from clause.
2. Use like operator to match job and in select clause to get the result.
Ans: select ename,sal from emp where sal>(select min(sal) from emp where job like 'A%');
ENAME SAL
-------------------- ----------
Arjun 12000
Gugan 20000
Karthik 15000
Q2: Issue a query to find all the employees who work in the same job as Arjun.
Ans: select * from emp;
EMPNO ENAME JOB DEPTNO SAL
---------- ------------------- ---------- ---------- ----------
1 Mathi AP 1 10000
2 Arjun ASP 2 12000
3 Gugan ASP 2 20000
4 Karthik AP 1 15000

select ename from emp where job=(select job from emp where ename='Arjun');
ENAME

Arjun
Gugan

SIMPLE JOIN:
Join operations – Example

- Relation *course*

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

- Relation *prereq*

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

SQL> SELECT * FROM *course* **join** *prereq on course.course_id=prereq.course_id;*

Result:
Thus the database tables is queried with sub queries and simple join operations.

Exp. No.5

JOINS

PRATHYUSHA ENGINEERING COLLEGE

Aim:

To Query the database tables and explore natural, equi and outer joins.

Procedure:

Join is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.

We will briefly describe various join types in the following sections.

Join

Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol **θ**.

Notation

$R1 \bowtie_{\theta} R2$

R1 and R2 are relations having attributes (A1, A2, .., An) and (B1, B2,.. ,Bn) such that the attributes don't have anything in common, that is $R1 \cap R2 = \Phi$.

Theta join can use all kinds of comparison operators.

Student		
SID	**Name**	**Std**
101	Alex	10
102	Maria	11

Subjects	
Class	**Subject**
10	Math
10	English
11	Music
11	Sports

Student_Detail −

STUDENT $\bowtie_{Student.Std = Subject.Class}$ SUBJECT

Student_detail				
SID	**Name**	**Std**	**Class**	**Subject**
101	Alex	10	10	Math
101	Alex	10	10	English
102	Maria	11	11	Music

PRATHYUSHA ENGINEERING COLLEGE

102	Maria	11	11	Sports

Equijoin

When Theta join uses only **equality** comparison operator, it is said to be equijoin. The above example corresponds to equijoin.

Natural Join (⋈)

Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain.

Natural join acts on those matching attributes where the values of attributes in both the relations are same.

Courses		
CID	**Course**	**Dept**
CS01	Database	CS
ME01	Mechanics	ME
EE01	Electronics	EE

HoD	
Dept	**Head**
CS	Alex
ME	Maya
EE	Mira

Courses ⋈ HoD			
Dept	**CID**	**Course**	**Head**
CS	CS01	Database	Alex
ME	ME01	Mechanics	Maya
EE	EE01	Electronics	Mira

Outer Joins

Theta Join, Equijoin, and Natural Join are called inner joins. An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation. Therefore, we need to use outer joins to include all the tuples from the participating relations in the resulting relation. There are three kinds of outer joins − left outer join, right outer join, and full outer join.

PRATHYUSHA ENGINEERING COLLEGE

Left Outer Join(R ⟕ S)

All the tuples from the Left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made NULL.

Left	
A	**B**
100	Database
101	Mechanics
102	Electronics

Right	
A	**B**
100	Alex
102	Maya
104	Mira

Courses ⟕ HoD			
A	**B**	**C**	**D**
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya

PRATHYUSHA ENGINEERING COLLEGE

Right Outer Join: (R ⟖ S)

All the tuples from the Right relation, S, are included in the resulting relation. If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL.

Courses ⟖ HoD			
A	**B**	**C**	**D**
100	Database	100	Alex
102	Electronics	102	Maya
---	---	104	Mira

Full Outer Join: (R ⟗ S)

All the tuples from both participating relations are included in the resulting relation. If there are no matching tuples for both relations, their respective unmatched attributes are made NULL.

Courses ⟗ HoD			
A	**B**	**C**	**D**
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya
---	---	104	Mira

Result:

Thus the database tables queried and explore natural, equi and outer joins.

PRATHYUSHA ENGINEERING COLLEGE

Exp. No:6

FUNCTIONS AND STORED PROCEDURES

Aim:

To Write user defined functions and stored procedures in SQL.

Procedure:

At the schema level, subprogram is a **standalone subprogram**. It is created with the CREATE PROCEDURE or the CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement.
A subprogram created inside a package is a **packaged subprogram**. It is stored in the database and can be deleted only when the package is deleted with the DROP PACKAGE statement. We will discuss packages in the chapter **'PL/SQL - Packages'**.
PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms −

- **Functions** − These subprograms return a single value; mainly used to compute and return a value.
- **Procedures** − These subprograms do not return a value directly; mainly used to perform an action.

Function:

```
CREATE OR REPLACE FUNCTION totalemployee return INT as
  total INT;
  begin
  SELECT count(*) into total from emp22;
 Return total;
 end;
 /
```

Function created.

```
declare
 a INT:=0;
 begin
 a:=totalemployee();
 dbms_output.put_line('Total employees are '||a);
 end;
 /
```

Total employees are 2
PL/SQL procedure successfully completed.

<u>Procedure example:</u>

<u>Program 1:</u>

```
set serveroutput on;
create or replace procedure p1 as
 begin
 dbms_output.put_line('welcome');
 end;
 /
```

Procedure created.

PRATHYUSHA ENGINEERING COLLEGE

```
 execute p1;
welcome
PL/SQL procedure successfully completed.
```

Program 2:

```
 create PROCEDURE findMin(x IN INT, y IN INT, z OUT INT) IS
  BEGIN
  IF x < y THEN
     z:= x;
    ELSE
       z:= y;
    END IF;
 END;
  /
Procedure created.

 declare
   a INT;
   b INT;
   c INT;
   begin
   a:=23;
  b:=4;
 findMin(a,b,c);
  dbms_output.put_line('Minimum value is:'||c);
  end;
   /
Minimum value is:4
PL/SQL procedure successfully completed.
```

Program 3:

```
set serveroutput on;
 create table emp22(id INT,name varchar(20),designation varchar(20),salary INT);
Table created.
 insert into emp22 values(3,'john','manager',100000);

1 row created.
 insert into emp22 values(3,'jagan','hr',400000);
1 row created.
```

PRATHYUSHA ENGINEERING COLLEGE

Result:
Thus the user defined functions and stored procedures in SQL are created and output is verified.

Exp. No.7

DCL AND TCL COMMANDS

Aim:

To execute complex transactions and realize DCL and TCL commands.

Procedure:

DCL COMMANDS

The DCL language is used for controlling the access to the table and hence securing the database. DCL is used to provide certain privileges to a particular user. Privileges are rights to be allocated.The privilege commands are namely, Grant and Revoke.The various privileges that can be granted or revoked are, Select Insert Delete Update References Execute All.

GRANT COMMAND: It is used to create users and grant access to the database. It requires database administrator (DBA) privilege, except that a user can change their password. A user can grant access to their database objects to other users.

REVOKE COMMAND: Using this command , the DBA can revoke the granted database privileges from the user.

TCL COMMAND

COMMIT: command is used to save the Records.
ROLL BACK: command is used to undo the Records.
SAVE POINT command is used to undo the Records in a particular transaction.

Queries:
Tables Used: Consider the following tables namely “DEPARTMENTS” and “EMPLOYEES”
Their schemas are as follows , Departments (dept _no , dept_ name , dept_location);
Employees (emp_id , emp_name , emp_salary);

Q1: Develop a query to grant all privileges of employees table into departments table
Ans: Grant all on employees to departments;
Grant succeeded.

Q2: Develop a query to grant some privileges of employees table into departments table
Ans: Grant select, update , insert on departments to departments with grant option;
Grant succeeded.

Q3: Develop a query to revoke all privileges of employees table from departments table
Ans: Revoke all on employees from departments; Revoke succeeded.

Q4: Develop a query to revoke some privileges of employees table from departments table
Ans: Revoke select, update , insert on departments from departments;
Revoke succeeded.

Q5: Write a query to implement the save point
Ans: SAVEPOINT S1;
Savepoint created.

PRATHYUSHA ENGINEERING COLLEGE

select * from emp;
EMPNO ENAME JOB DEPTNO SAL
---------- -------------------- ------------- ---------- ----------
1 Mathi AP 1 10000
2 Arjun ASP 2 15000
3 Gugan ASP 1 15000
4 Karthik Prof 2 30000
INSERT INTO EMP VALUES(5,'Akalya','AP',1,10000); 1 row created.
select * from emp;
EMPNO ENAME JOB DEPTNO SAL
---------- -------------------- ------------- ---------- ----------
1 Mathi AP 1 10000
2 Arjun ASP 2 15000
3 Gugan ASP 1 15000
4 Karthik Prof 2 30000
5 Akalya AP 1 10000

Q6: Write a query to implement the rollback
Ans: rollback s1; select * from emp;
EMPNO ENAME JOB DEPTNO SAL
---------- -------------------- ------------- ---------- ----------
1 Mathi AP 1 10000
2 Arjun ASP 2 15000
3 Gugan ASP 1 15000
4 Karthik Prof 2 30000

Q6: Write a query to implement the commit
Ans: COMMIT;
Commit complete.

Result:

Thus the DCL and TCL commands are executed.

PRATHYUSHA ENGINEERING COLLEGE

Exp. No:8

TRIGGERS

Aim:

To Write SQL Triggers for insert, delete, and update operations in a database table.

Procedure:

Trigger :

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events –

- A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Table creation:

Create table employee22(empid INT,empname varchar(20),empdept varchar(20),salary INT);

Example:

```
CREATE OR REPLACE TRIGGER display_salary
BEFORE DELETE OR UPDATE ON employee22
FOR EACH ROW
DECLARE
  sal_diff INT;
BEGIN
  sal_diff := :NEW.salary  - :OLD.salary;
  dbms_output.put_line('Old salary: ' || :OLD.salary);
  dbms_output.put_line('New salary: ' || :NEW.salary);
  dbms_output.put_line('Salary difference: ' || sal_diff);
END;
```

Trigger to ensure that no employee of age less than 25 can be inserted in the database.

```
CREATE TRIGGER  Check_age  BEFORE INSERT ON employee
FOR EACH ROW
BEGIN
IF NEW.age < 25 THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'ERROR:
      AGE MUST BE ATLEAST 25 YEARS!';
END IF;
END;
/
```

Result:

Thus the SQL Triggers for insert, delete, and update operations in a database table is created and executed successfully.

Exp. No: 9

VIEW AND INDEX

PRATHYUSHA ENGINEERING COLLEGE

Aim:

To Create View and index for database tables with a large number of records.

Procedure:

View:

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables, or another view. To create a view, a user must have the appropriate system privilege according to the specific implementation.

CREATE TABLE **EMPLOYEE** (EMPLOYEE_NAME VARCHAR(10), EMPLOYEE_NO INT(8), DEPT_NAME VARCHAR(10), DEPT_NO INT (5),DATE_OF_JOIN DATE);

Table created.

CREATE VIEW

SYNTAX FOR CREATION OF VIEW

CREATE [OR REPLACE] [FORCE] VIEW viewname [(column-name, column-name)] AS Query [with check option];

Include all not null attribute.

CREATION OF VIEW

CREATE VIEW **EMPVIEW** AS SELECT EMPLOYEE_NAME, EMPLOYEE_NO, DEPT_NAME, DEPT_NO, DATE_OF_JOIN FROM **EMPLOYEE**;

View Created.

DISPLAY VIEW:

SELECT * FROM EMPVIEW;

EMPLOYEE_N	EMPLOYEE_NO	DEPT_NAME	DEPT_NO
RAVI	124	ECE	89
VIJAY	345	CSE	21
RAJ	98	IT	22
GIRI	100	CSE	67

INSERT INTO EMPVIEW VALUES ('SRI', 120,'CSE', 67);

1 ROW CREATED.

DROP VIEW EMPVIEW;

view dropped

CREATE OR REPLACE VIEW EMP_TOTSAL AS SELECT EMPNO "EID", ENAME "NAME", SALARY "SAL" FROM EMPL;

JOIN VIEW:

EXAMPLE-5:

CREATE OR REPLACE VIEW DEPT_EMP_VIEW AS SELECT A.EMPNO, A.ENAME,A.DEPTNO, B.DNAME, B.LOC FROM EMPL A, DEPMT B WHERE A.DEPTNO=B.DEPTNO;

CREATE INDEX

CREATE INDEX idx_lastname ON Persons (LastName);

ALTER INDEX

ALTER INDEX <index name> ON <table name> (<column(s)>);

DROP INDEX

DROP INDEX *index_name*;

Result:

Thus the View and index for database tables with are created.

Exp. No:10

PRATHYUSHA ENGINEERING COLLEGE

XML DATABASE

Aim:

To Create an XML database and validate it using XML schema

Procedure:

XML database is a data persistence software system used for storing the huge amount of information in XML format. It provides a secure place to store XML documents.

You can query your stored data by using XQuery, export and serialize into desired format. XML databases are usually associated with document-oriented databases.

Types of XML databases

There are two types of XML databases.

1. XML-enabled database
2. Native XML database (NXD)

XML-enable Database

XML-enable database works just like a relational database. It is like an extension provided for the conversion of XML documents. In this database, data is stored in table, in the form of rows and columns.

Native XML Database

Native XML database is used to store large amount of data. Instead of table format, Native XML database is based on container format. You can query data by XPath expressions.

Native XML database is preferred over XML-enable database because it is highly capable to store, maintain and query XML documents.

Let's take an example of XML database:

```
<?xml version="1.0"?>
<contact-info>
  <contact1>
    <name>Vimal Jaiswal</name>
    <company>SSSIT.org</company>
    <phone>(0120) 4256464</phone>
  </contact1>
  <contact2>
    <name>Mahesh Sharma </name>
    <company>SSSIT.org</company>
    <phone>09990449935</phone>
  </contact2>
</contact-info>
```

In the above example, a table named contacts is created and holds the contacts (contact1 and contact2). Each one contains 3 entities name, company and phone.

XML Validation

A well formed XML document can be validated against DTD or Schema.

A well-formed XML document is an XML document with correct syntax. It is very necessary to know about valid XML document before knowing XML validation.

Valid XML document

It must be well formed (satisfy all the basic syntax condition)

It should be behave according to predefined DTD or XML schema

XML DTD

PRATHYUSHA ENGINEERING COLLEGE

A DTD defines the legal elements of an XML document
In simple words we can say that a DTD defines the document structure with a list of legal elements and attributes.
XML schema is a XML based alternative to DTD.
Actually DTD and XML schema both are used to form a well formed XML document.
We should avoid errors in XML documents because they will stop the XML programs.

XML schema

It is defined as an XML language
Uses namespaces to allow for reuses of existing definitions
It supports a large number of built in data types and definition of derived data types

Checking Validation

An XML document is called "well-formed" if it contains the correct syntax. A well-formed and valid XML document is one which have been validated against Schema.
Visit **http://www.xmlvalidation.com** to validate the XML file against schema or DTD.

XML Schema Example

Let's create a schema file.
employee.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.javatpoint.com"
xmlns="http://www.javatpoint.com"
elementFormDefault="qualified">

<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="email" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
1.        </xs:element>
2.
3.        </xs:schema>
```

Let's see the xml file using XML schema or XSD file.
employee.xml

```
<?xml version="1.0"?>
<employee
xmlns="http://www.javatpoint.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.javatpoint.com employee.xsd">

  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```

Description of XML Schema

PRATHYUSHA ENGINEERING COLLEGE

<xs:element name="employee"> : It defines the element name employee.
<xs:complexType> : It defines that the element 'employee' is complex type.
<xs:sequence> : It defines that the complex type is a sequence of elements.
<xs:element name="firstname" type="xs:string"/> : It defines that the element 'firstname' is of string/text type.
<xs:element name="lastname" type="xs:string"/> : It defines that the element 'lastname' is of string/text type.
<xs:element name="email" type="xs:string"/> : It defines that the element 'email' is of string/text type.

XML Schema Data types

There are two types of data types in XML schema.

1. simpleType
2. complexType

simpleType

The simpleType allows you to have text-based elements. It contains less attributes, child elements, and cannot be left empty.

complexType

The complexType allows you to hold multiple attributes and elements. It can contain additional sub elements and can be left empty.

Result:

Thus an XML database and validating using XML schema is completed.

PRATHYUSHA ENGINEERING COLLEGE

Exp. No.:11

NOSQL DATABASE TOOLS

Aim:

To Create Document, column and graph based data using NOSQL database tools.

Procedure:

MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

Database

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.
The following table shows the relationship of RDBMS terminology with MongoDB.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by MongoDB itself)
Database Server and Client	
mysqld/Oracle	mongod
mysql/sqlplus	mongo

Sample Document

Following example shows the document structure of a blog site, which is simply a comma separated key value pair.

```
{
  _id: ObjectId(7df78ad8902c)
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
```

PRATHYUSHA ENGINEERING COLLEGE

```
  by: 'tutorials point',
  url: 'http://www.tutorialspoint.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100,
  comments: [
    {
      user:'user1',
      message: 'My first comment',
      dateCreated: new Date(2011,1,20,2,15),
      like: 0
    },
    {
      user:'user2',
      message: 'My second comments',
      dateCreated: new Date(2011,1,25,7,45),
      like: 5
    }
  ]
}
```

_id is a 12 bytes hexadecimal number which assures the uniqueness of every document. You can provide _id while inserting the document. If you don't provide then MongoDB provides a unique id for every document. These 12 bytes first 4 bytes for the current timestamp, next 3 bytes for machine id, next 2 bytes for process id of MongoDB server and remaining 3 bytes are simple incremental VALUE.

The use Command

MongoDB **use DATABASE_NAME** is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

Syntax

Basic syntax of **use DATABASE** statement is as follows −

use DATABASE_NAME

Example

If you want to use a database with name **<mydb>**, then **use DATABASE** statement would be as follows −

```
>use mydb
switched to db mydb
```

To check your currently selected database, use the command **db**

>db

mydb

If you want to check your databases list, use the command **show dbs**.

>show dbs

local 0.78125GB

test 0.23012GB

Your created database (mydb) is not present in list. To display database, you need to insert at least one document into it.

```
>db.movie.insert({"name":"tutorials point"})
>show dbs
local     0.78125GB
mydb      0.23012GB
```


PRATHYUSHA ENGINEERING COLLEGE

```
test      0.23012GB
```

In MongoDB default database is test. If you didn't create any database, then collections will be stored in test database.

The dropDatabase() Method

MongoDB **db.dropDatabase()** command is used to drop a existing database.

Syntax

Basic syntax of **dropDatabase()** command is as follows −

```
db.dropDatabase()
```

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

Example

First, check the list of available databases by using the command, **show dbs**.

```
>show dbs
local     0.78125GB
mydb      0.23012GB
test      0.23012GB
>
```

If you want to delete new database **<mydb>**, then **dropDatabase()** command would be as follows −

```
>use mydb
switched to db mydb
>db.dropDatabase()
>{ "dropped" : "mydb", "ok" : 1 }
>
```

Now check list of databases.

```
>show dbs
local     0.78125GB
test      0.23012GB
>
```

The insert() Method

To insert data into MongoDB collection, you need to use MongoDB's **insert()** or **save()** method.

Syntax

The basic syntax of **insert()** command is as follows −

```
>db.COLLECTION_NAME.insert(document)
```

Example

```
> db.users.insert({
... _id : ObjectId("507f191e810c19729de860ea"),
... title: "MongoDB Overview",
... description: "MongoDB is no sql database",
... by: "tutorials point",
... url: "http://www.tutorialspoint.com",
... tags: ['mongodb', 'database', 'NoSQL'],
... likes: 100
... })
WriteResult({ "nInserted" : 1 })
```

PRATHYUSHA ENGINEERING COLLEGE

```
>
```

Here **mycol** is our collection name, as created in the previous chapter. If the collection doesn't exist in the database, then MongoDB will create this collection and then insert a document into it.
In the inserted document, if we don't specify the _id parameter, then MongoDB assigns a unique ObjectId for this document.
_id is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows −
_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer)
You can also pass an array of documents into the insert() method as shown below:.

```
> db.createCollection("post")
> db.post.insert([
	{
		title: "MongoDB Overview",
		description: "MongoDB is no SQL database",
		by: "tutorials point",
		url: "http://www.tutorialspoint.com",
		tags: ["mongodb", "database", "NoSQL"],
		likes: 100
	},
	{
	title: "NoSQL Database",
	description: "NoSQL database doesn't have tables",
	by: "tutorials point",
	url: "http://www.tutorialspoint.com",
	tags: ["mongodb", "database", "NoSQL"],
	likes: 20,
	comments: [
		{
			user:"user1",
			message: "My first comment",
			dateCreated: new Date(2013,11,10,2,35),
			like: 0
		}
	]
}
])
BulkWriteResult({
	"writeErrors" : [ ],
	"writeConcernErrors" : [ ],
	"nInserted" : 2,
	"nUpserted" : 0,
	"nMatched" : 0,
	"nModified" : 0,
	"nRemoved" : 0,
	"upserted" : [ ]
})
>
```

PRATHYUSHA ENGINEERING COLLEGE

To insert the document you can use **db.post.save(document)** also. If you don't specify **_id** in the document then **save()** method will work same as **insert()** method. If you specify _id then it will replace whole data of document containing _id as specified in save() method.

The insertOne() method

If you need to insert only one document into a collection you can use this method.

Syntax

The basic syntax of insert() command is as follows −

```
>db.COLLECTION_NAME.insertOne(document)
```

Example

Following example creates a new collection named empDetails and inserts a document using the insertOne() method.

```
> db.createCollection("empDetails")
{ "ok" : 1 }
> db.empDetails.insertOne(
	{
		First_Name: "Radhika",
		Last_Name: "Sharma",
		Date_Of_Birth: "1995-09-26",
		e_mail: "radhika_sharma.123@gmail.com",
		phone: "9848022338"
	})
{
	"acknowledged" : true,
	"insertedId" : ObjectId("5dd62b4070fb13eec3963bea")
}
>
```

The insertMany() method

You can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.

Example

Following example inserts three different documents into the empDetails collection using the insertMany() method.

```
> db.empDetails.insertMany(
	[
		{
			First_Name: "Radhika",
			Last_Name: "Sharma",
			Date_Of_Birth: "1995-09-26",
			e_mail: "radhika_sharma.123@gmail.com",
			phone: "9000012345"
		},
		{
			First_Name: "Rachel",
			Last_Name: "Christopher",
			Date_Of_Birth: "1990-02-16",
			e_mail: "Rachel_Christopher.123@gmail.com",
			phone: "9000054321"
		},
```

PRATHYUSHA ENGINEERING COLLEGE

```
            {
                        First_Name: "Fathima",
                        Last_Name: "Sheik",
                        Date_Of_Birth: "1990-02-16",
                        e_mail: "Fathima_Sheik.123@gmail.com",
                        phone: "9000054321"
            }
      ]
)
{
      "acknowledged" : true,
      "insertedIds" : [
            ObjectId("5dd631f270fb13eec3963bed"),
            ObjectId("5dd631f270fb13eec3963bee"),
            ObjectId("5dd631f270fb13eec3963bef")
      ]
}
>
```

PRATHYUSHA ENGINEERING COLLEGE

Result:
Thus the Document, column and graph based data using NOSQL database tools is created successfully.

Exp. No: 12

EMPLOYEE MANAGEMENT SYSTEM

Aim :

To implement the project for Employee management system.

Procedure:

Employee Management System is used to maintain the detail of employees of an organization. HR can view the details department wise. HR can delete the employees.HR can update the employees details. HR can add new employees.

Tables :

- **Employee**(empno INT primary key , empname varchar(20) , department varchar(20) , salary INT)

select *from employee;

EMPID	EMPNAME	EMPDEPT	SALARY
1	rekha	cse	30000
2	renu	cse	70000
3	elamathi	ece	90000
4	rahul	ece	79990

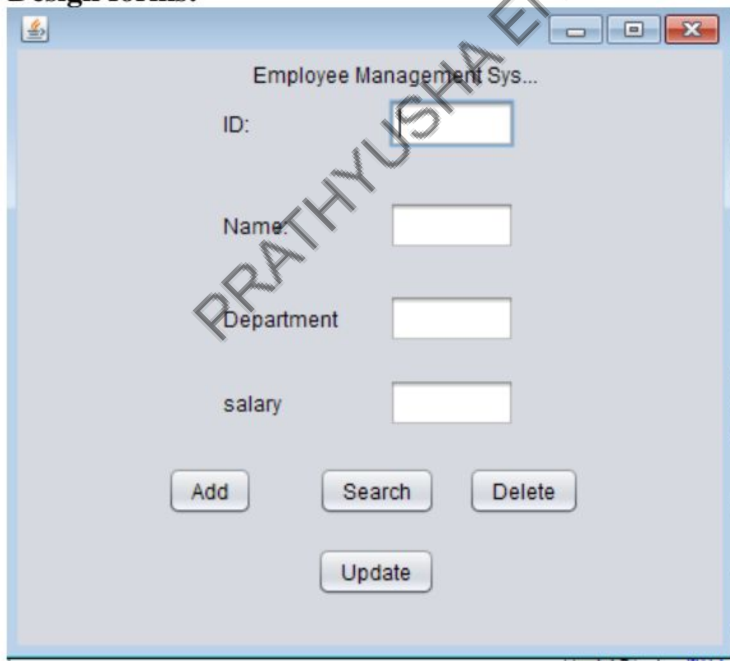
Net beans:

Add jar file.

Right click on package---select properties ----select libraries- add jar file(ojdbc7).

Right click on package --- select new----select JFrame forms.

Design forms:



PRATHYUSHA ENGINEERING COLLEGE

Procedure:

1. Code for components in GUI will come automatically.

2. Do oracle connectivity. (click on services----right click on database—select new connection—
   Select oracle thin driver----give service id as orcl.---give username and password.---click test connection.

3. Import javax.swing and java.sql
4. Double click on button in forms and type the coding

Coding:

```
private void deleteActionPerformed(java.awt.event.ActionEvent evt) {
try{
  Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:orcl","system","manager");

Statement stmt=con.createStatement();

String a=JOptionPane.showInputDialog(null,"Enter Employee id");
int temp=0;
ResultSet rs=stmt.executeQuery("select * from employee");
while(rs.next())
{
   if (rs.getString(1).equals(a))
   {
      temp=1;

         }
}

if (temp==1)
{
   String query="delete from employee where empid= " + a ;
   stmt.execute(query);
      JOptionPane.showMessageDialog (null,"employee record deleted");

}
else

     JOptionPane.showMessageDialog (null,"employee record not found");



con.close();

}

catch(SQLException ex)
      {
```

PRATHYUSHA ENGINEERING COLLEGE

```
        JOptionPane.showMessageDialog (null,ex);

       } catch (ClassNotFoundException ex) {

       }
    }

    private void salaryActionPerformed(java.awt.event.ActionEvent evt) {
       // TODO add your handling code here:
    }

    private void nameActionPerformed(java.awt.event.ActionEvent evt) {
       // TODO add your handling code here:
    }

    private void addbuttonActionPerformed(java.awt.event.ActionEvent evt) {
try
{

    Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:orcl","system","manager");

Statement stmt=con.createStatement();
String query="insert into employee values( "
       + id.getText()+", '"+name.getText()+"' ,'"+department.getText()+"' ,"
      +salary.getText()+")";

//String query ="insert into employee values(34,'mala','cse',234324)";
       stmt.execute(query);



       JOptionPane.showMessageDialog (null,"employee added");

       id.setText(null);
       name.setText(null);
       department.setText(null);
       salary.setText(null);

//con.close();

}

catch(SQLException ex)
       {
        JOptionPane.showMessageDialog (null,ex);

       } catch (ClassNotFoundException ex) {
```


PRATHYUSHA ENGINEERING COLLEGE

```
        }
    }

    private void addbuttonMouseClicked(java.awt.event.MouseEvent evt) {

    }
    private void searchActionPerformed(java.awt.event.ActionEvent evt) {
try{

        Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:orcl","system","manager");

Statement stmt=con.createStatement();
    String a=JOptionPane.showInputDialog(null,"Enter Employee id");

ResultSet rs=stmt.executeQuery("select * from employee");
while(rs.next())
{
    if (rs.getString(1).equals(a))
    {
        id.setText(rs.getString(1));
            name.setText(rs.getString(2));
        department.setText(rs.getString(3));
        salary.setText(rs.getString(4));

    }
}

    }
    catch(SQLException ex)
        {
         JOptionPane.showMessageDialog (null,ex);

        } catch (ClassNotFoundException ex) {

        }
    }
    private void updateActionPerformed(java.awt.event.ActionEvent evt) {
try
{
    Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:orcl","system","manager");
Statement stmt=con.createStatement();
String query="update employee set empid= " + id.getText()+",empname='"
        +name.getText()+"' , empdept='"+department.getText()+"' ,salary="
      +salary.getText()+"where empid= " + id.getText();
```

PRATHYUSHA ENGINEERING COLLEGE

```
//String query ="insert into employee values(34,'mala','cse',234324)";
        stmt.executeQuery(query);
        JOptionPane.showMessageDialog (null,"employee details updated");
        id.setText(null);
        name.setText(null);
        department.setText(null);
        salary.setText(null);
        con.close();
        }
catch(SQLException ex)
        {
         JOptionPane.showMessageDialog (null,ex);

        } catch (ClassNotFoundException ex) {
      }
    }
```

Result:

Thus the Employee project is completed.

PRATHYUSHA ENGINEERING COLLEGE

Exp. No.13:

CASE STUDY

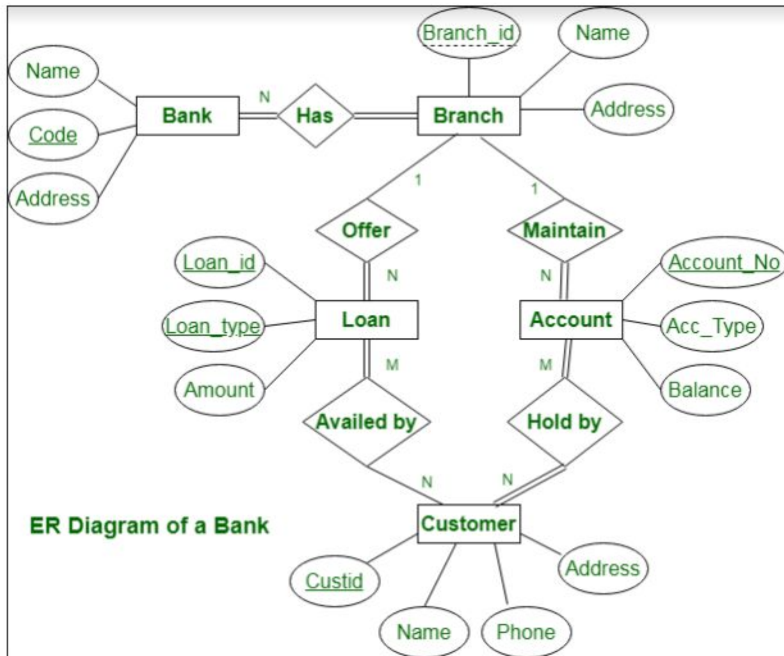
BANK MANAGEMENT SYSTEM

Aim:

To create bank management system using oracle.

Procedure:

1. Draw ER diagram



2. Create database tables
3. Design forms
4. Complete the connectivity.
5. Run the project

Result:

Thus the bank management system using oracle is created and executed successfully.

PRATHYUSHA ENGINEERING COLLEGE

ADDITIONAL EXPERIMENTS

EXP. NO.14

PL/SQL PROCEDURE TO DISPLAY FIBONACCI SERIES

AIM:

To write a PL/SQL procedure for displaying Fibonacci series.

ALGORITHM:

1. Start
2. Initialize variables first= 0, second=1.

3. Display the first and second variables values.
4. For i =1 to 5
   temp:=first+second;
   first := second;
   second := temp;
   display temp
5. stop

PROGRAM:

```
first number := 0;
second number := 1;
temp number;
n number := 5;
i number;
begin
  dbms_output.put_line('Series:');
  dbms_output.put_line(first);
   dbms_output.put_line(second);
    for i in 2..n
 loop
  temp:=first+second;
 first := second;
  second := temp;
  dbms_output.put_line(temp);
 end loop;
 end;
```

OUTPUT:

0 1 1 2 3 5

RESULT:

Thus the PL/SQL procedure for displaying Fibonacci series is executed successfully.

PRATHYUSHA ENGINEERING COLLEGE

EXP. NO.15

PL/SQL PROCEDURE TO DISPLAY STUDENT INFORMATION USING ARRAY

AIM:

To write a PL/SQL procedure for displaying student information using array.

ALGORITHM:

1. Start
2. Declare an array

3. Store student information in arrary.
4. Display student information using for loop.
5. End.

PROGRAM:

```
DECLARE
  type namesarray IS VARRAY(5) OF VARCHAR2(10);
  type grades IS VARRAY(5) OF INTEGER;
  names namesarray;
  marks grades;
  total integer;
BEGIN
  names := namesarray('Kavita', 'Pritam', 'Ayan', 'Rishav', 'Aziz');
  marks:= grades(98, 97, 78, 87, 92);
  total := names.count;
  dbms_output.put_line('Total '|| total || ' Students');
  FOR i in 1 .. total LOOP
    dbms_output.put_line('Student: ' || names(i) || '
    Marks: ' || marks(i));
  END LOOP;
END;
/
```

OUTPUT:

Total 5 Students
Student: Kavita Marks: 98
Student: Pritam Marks: 97
Student: Ayan Marks: 78
Student: Rishav Marks: 87
Student: Aziz Marks: 92

RESULT:

Thus the PL/SQL procedure for displaying student information using array is executed successfully.

PRATHYUSHA ENGINEERING COLLEGE