



G H Raisoni College of Engineering
& Management, Wagholi, Pune



Department of Computer Engineering

D-19

Lab Manual
(2022-23)

Class: TY Computer

Term: V

UCOL301: Database Management Systems
LAB

Faculty Name: Mrs. Sunita Nandgave



G H Raisoni College of Engineering
& Management, Wagholi, Pune



Department of Computer Engineering

Database Management Systems (UCOP301)

Course Outcome

CO1	Analyze an information storage problem and derived an information model expression in the form of Entity relation diagram and design appropriate data model for it.
CO2	Demonstrate SQL queries to perform CRUD (Create, Retrieve, Update, Delete) operations on database and perform inferential analysis of data model
CO3	Identify features of database management systems and Relational database and Understand functional dependencies and various normalization forms
CO4	Perform basic transaction processing and management and ensure database security, integrity and concurrency control

List of Experiment

Sr. No.	Name of Experiment	CO Mapped
1	Draw E-R diagram and convert entities and relationships to relation table for a given scenario. a. Two assignments shall be carried out i.e. consider two different scenarios (eg. bank, college, Employee, Hotel etc.)	CO1
2	To perform following SQL activity: a) Creating a database b) Creating Tables (With and Without Constraints) c) Inserting Record in table	CO1, CO2
3	To Perform the following: a. Viewing all databases, Viewing all Tables in a Database, Updating/Deleting Records in a Table	CO2, CO3
4	To Perform the following SQL query on database: a. Altering a Table, Dropping/Truncating/Renaming Tables, Backing up / Restoring a Database	CO2, CO3
5	For a given set of relation schemes, create tables and perform the following Simple Queries: Simple Queries with Aggregate functions, Queries with Aggregate functions (group by and having clause), Queries involving- Date Functions, String Functions , Math Functions	CO2, CO3
6	To perform SQL query that demonstrate Join Queries- Inner Join, Outer Join, Left join, Right Join	CO2, CO3

7	To perform SQL query that demonstrate following: Search conditions, Summary queries, Sub- queries, Subqueries- With IN clause, With EXISTS clause	CO2, CO3
8	To perform SQL query for extracting data from more than one table using SQL concept	CO2, CO3
9	To perform SQL query to understand the concepts: Transaction, ROLL BACK, COMMIT & CHECK POINTS.	CO2, CO3
	Content Beyond Syllabus	
10	Open ended practical on NOSQL	CO2, CO3
11	Cassandra case study	CO2, CO3

Experiment No. - 1

AIM: Draw E-R diagram and convert entities and relationships to relation table for a given scenario.

Two assignments shall be carried out i.e. consider two different scenarios (eg. bank, college, Employee, Hotel etc)

Theory:

ER diagrams are visual tools that are helpful to represent the ER model. Peter Chen proposed ER Diagram in 1971 to create a uniform convention that can be used for relational databases and networks. He aimed to use an ER model as a conceptual modeling approach.

An entity-relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is an object, a component of data. An entity set is a collection of similar entities. These entities can have attributes that define their properties. By defining the entities, and their attributes, and showing the relationships between them, an ER diagram illustrates the logical structure of databases.

Facts about ER diagram are as follows:

- ER model allows you to draw Database Design
- It is an easy to use graphical tool for modeling data
- Widely used in Database Design
- It is a GUI representation of the logical structure of a Database
- It helps you to identifies the entities which exist in a system and the relationships between those entities
- ER diagrams are used to sketch out the design of a database

ER Diagrams Symbols & Notations:

Entity Relationship Diagram Symbols & Notations mainly contains three basic symbols which are rectangle, oval and diamond to represent relationships between elements, entities and attributes. There are some sub-elements which are based on main elements in ERD Diagram. ER Diagram is a visual representation of data that describes how data is related to each other using different ERD Symbols and Notations.

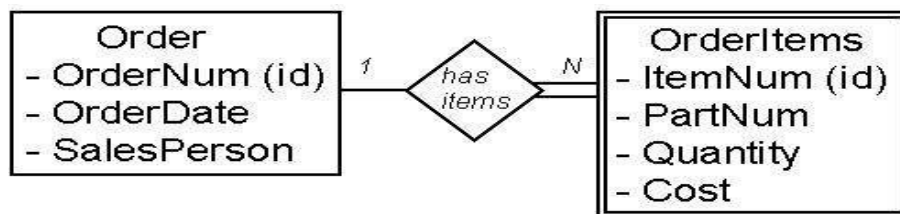
Following are the main components and its symbols in ER Diagrams:

- **Rectangles:** This Entity Relationship Diagram symbol represents entity types
- **Ellipses :** Symbol represent attributes
- **Diamonds:** This symbol represents relationship types

- **Lines:** It links attributes to entity types and entity types with other relationship types
- **Primary key:** attributes are underlined
- **Double Ellipses:** Represent multi-valued attributes



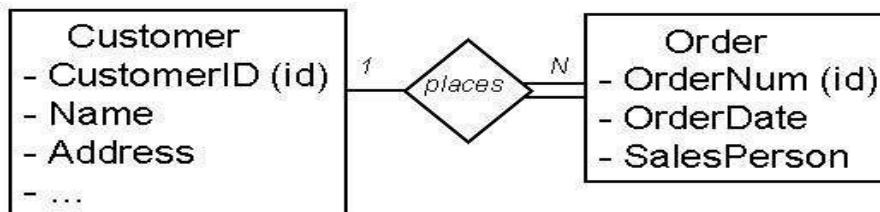
Figure 1. ER Diagram Symbols



Chen Notation

- ORDER** (OrderNum (key), OrderDate, SalesPerson)
ORDERITEMS (OrderNum (key)(fk) , ItemNum (key), PartNum, Quantity, Cost)
- In the above example, in the ORDERITEMS Relation: OrderNum is the Foreign Key and OrderNum plus ItemNum is the Composite Key.

Chen Notation



In the ORDER Relation: OrderNum is the *Key*.

Representing Relationships

- 1:1 Relationships.** The key of one relation is stored in the second relation. Look at example queries to determine which key is queried most often.
- 1:N Relationships.**

Parent - Relation on the "1" side. **Child** - Relation on the "Many" side.

d. Represent each Entity as a relation.

Copy the key of the parent into the child relation.

e. **CUSTOMER** (CustomerID (key), Name, Address, ...)

ORDER (OrderNum (key), OrderDate, SalesPerson, CustomerID (fk))

Schema for Company Database:

EMPLOYEE (SSN, Name, Address, Sex, Salary, SuperSSN, DNo)

DEPARTMENT (DNo, DName, MgrSSN, MgrStartDate)

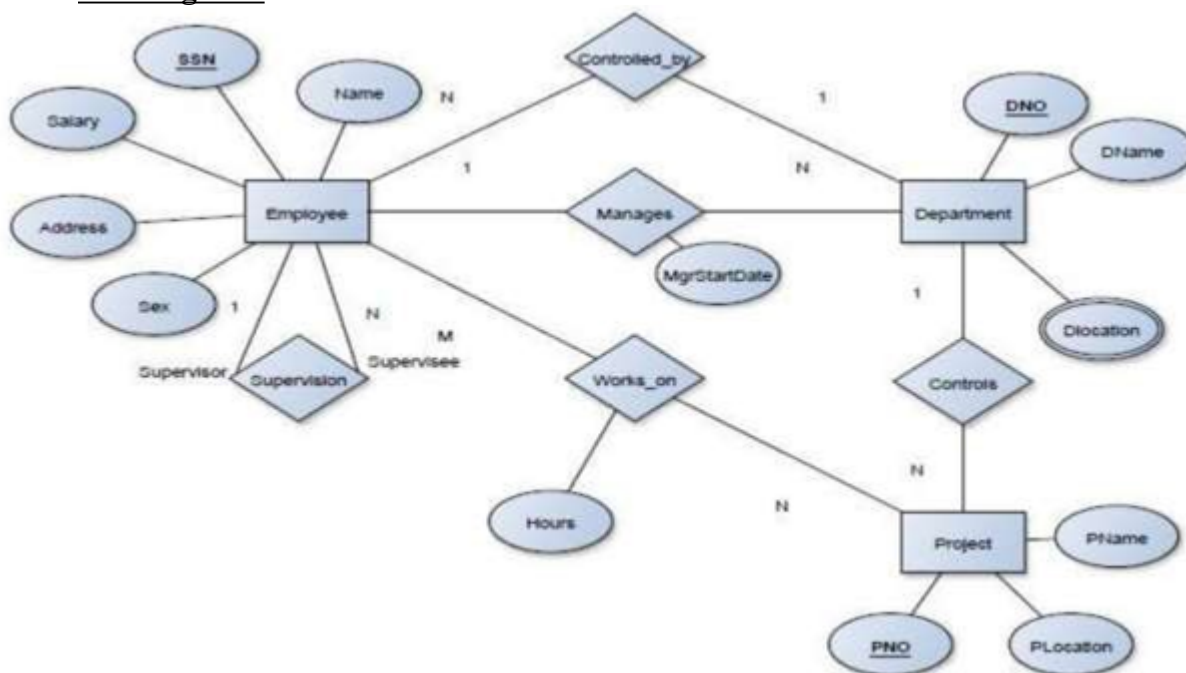
DLOCATION (DNo, DLoc)

PROJECT (PNo, PName, PLocation,

DNo) WORKS_ON (SSN, PNo,

Hours)

ER-Diagram:



Conclusion: In this experiment, we have studied history, facts about ER Diagram, necessity of ER diagram along with Entity Relationships diagram notations and their use to draw the design of a database system.

Experiment No. - 2

AIM: To perform following SQL activity:

a) Creating a database

b) Creating Tables (With and Without Constraints)

c) Inserting Record in table

Theory:

CONSTRAINTS:

Integrity Constraints are a mechanism to prevent invalid data entry into the table to maintain data consistency. The whole purpose of constraints is to maintain the data integrity during the various transactions like update/delete/insert on a table.

Types of constraints:

- NOT NULL
- UNIQUE
- DEFAULT
- CHECK
- Key Constraints – PRIMARY KEY, FOREIGN KEY

NOT NULL:

NOT NULL constraint makes sure that a column does not hold NULL value. When we don't provide value for a particular column while inserting a record into a table, it takes NULL value by default. By specifying the NULL constraint, we can be sure that a particular column(s) cannot have NULL values.

UNIQUE:

UNIQUE Constraint enforces a column or set of columns to have unique values. If a column has a unique constraint, it means that a particular column cannot have duplicate values in a table.

DEFAULT:

The DEFAULT constraint provides a default value to a column when there is no value provided while inserting a record into a table.

CHECK:

This constraint is used for specifying a range of values for a particular column of a table. When this constraint is being set on a column, it ensures that the specified column must have the value falling in the specified range.

The primary key uniquely identifies each record in a table. It must have unique values and cannot contain nulls. In the below example the ROLL_NO field is marked as the primary key, which means the ROLL_NO field cannot have duplicate and null values.

FOREIGN KEY:

Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

- a) Creating a database:

```
Mysql> create databse a1;
```

Output:

Database created

- b) To view all databases

```
Mysql> show databases;
```

Output:

a1

System_init

- c) Use created database

```
Mysql> use a1;
```

Output:

database changed

- d) Create table without constraint

```
Mysql> create table student (rollno integer(5), name varchar(30));
```

Output:

Query Ok, 0 rows affected 0 rows updated.

- e) Table creation with constraints:

General syntax:

```
Create table tablename (column_name1 data_type constraints, column_name2 data_type constraints ...);
```

Example:

```
Create table Emp ( EmpNo number(5), EName VarChar(15), Job Char(10) constraint unique, DeptNo number(3) CONSTRAINT FKey2 REFERENCES DEPT(DeptNo));
```

```
Create table stud (sname varchar2(20) not null, rollno number(10) not null, dob date notnull);
```

DOMAIN INTEGRITY

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

```
Alter table cust modify (name not null);
```

CHECK CONSTRAINT

Example: Create table student (regno number (6), mark number (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 number(6) constraint unique, name char(10)); Altertable cust add(constraint c unique(custid));

b) Primary Key Constraint

Example: Create table stud(regno number(6) constraint primary key, name char(20));

Queries:

Q1. Create a table called EMP with the following structure.

Name	Type
------	------

EMPNO	NUMBER(6)
-------	-----------

ENAME	
-------	--

JOB	VARCHAR2(20)
-----	--------------

DEPTNO	NUMBER(3)
--------	-----------

SAL	NUMBER(7,2)
-----	-------------

Allow NULL for all columns except ename and job.

1. Understand create table syntax.

2. Use the create table syntax to create the said tables.

3. Create primary key constraints for each table as understood from logical table structure.

SQL> create table emp(empno number(6),ename varchar2(20)not null,job varchar2(10)not null,deptno number(3), sal number(7,2));

Table created.

Q2: Add a column experience to the emp table.

experience numeric null allowed.

Solution:

1. Learn to alter table syntax.

2. Define the new column and its data type.

3. Use the altered table syntax.

Ans: SQL> alter table emp add(experience number(2));Table altered.

Q3: Modify the column width of the job field of the emp table.

Solution:

1. Use the altered table syntax.

2. Modify the column width and its data type.

Ans: SQL> alter table emp modify(job varchar2(12));

Table altered.

SQL> alter table emp modify(job varchar(13));

Table altered.

Q4: Create a dept table with the following structure.

Name Type

DEPT NUMBER(2)
DNAME VARCHAR2(10)
LOC VARCHAR2(10)
Deptno the primary key

Solution:

1. Understand create table syntax.
2. Decide the name of the table.
3. Decide the name of each column and its data type.
4. Use the create table syntax to create the said tables.
5. Create primary key constraint for each table as understand from logical table structure. Ans:

```
SQL> create table dept(deptno number(2) primary key,dname  
varchar2(10),loc varchar2(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.

Solution:

1. Learn alter table syntax.
2. Define the new constraint [columns name type]
3. Use the alter table syntax for adding constraints.

Ans:

```
SQL> create table emp1(ename varchar2(10),empno number(6) constraint  
check(empno>100));
```

Table created.

Q6: drop a column experience to the emp table.

Solution:

1. Learn to alter table syntax. Use the alter table syntax to drop the column.

Ans:

```
SQL> alter table emp drop column experience; Table altered.
```

Q7: Truncate the emp table and drop the dept table

Solution:

1. Learn drop, and truncate table syntax.

Ans: SQL> truncate table emp; Table truncated.

OUTPUT:

SQL> create table emp1(emp_no number(6),ename varchar2(20) not null,job varchar2(10) not null,dept_no number(3),sal number(7,2));

Table created.

SQL> desc emp1;

Name	Null?	Type
EMP_NO		NUMBER(6)
ENAME		NOT NULL VARCHAR2(20)
JOB		NOT NULL VARCHAR2(10)
DEPT_NO		NUMBER(3)
SAL		NUMBER(7,2)

SQL> alter table emp1 add(experience number(5));

Table altered.

SQL> desc emp1;

Name	Null?	Type
EMP_NO		NUMBER(6)
ENAME		NOT NULL VARCHAR2(20)
JOB		NOT NULL VARCHAR2(10)
DEPT_NO		NUMBER(3)
SAL		NUMBER(7,2)
EXPERIENCE		NUMBER(5)

SQL> alter table emp1 modify(job varchar2(12));

Table altered.

SQL> desc emp1;

Name	Null?	Type
EMP_NO		NUMBER(6)
ENAME		NOT NULL VARCHAR2(20)
JOB		NOT NULL VARCHAR2(12)
DEPT_NO		NUMBER(3)
SAL		NUMBER(7,2)
EXPERIENCE		NUMBER(5)

```
SQL> create table dept1(dept_no number(12) primary key,d_name varchar2(10),loc
varchar2(10));
Table created.
```

```
SQL> desc dept1;
```

Name	Null?	Type
DEPT_NO		NOT NULL NUMBER(12)
D_NAME		VARCHAR2(10)
LOC		VARCHAR2(10)

```
SQL> alter table emp1 modify(emp_no number(6) check (emp_no>100));
Table altered.
```

```
SQL> desc emp1;
```

Name	Null?	Type
EMP_NO		NUMBER(6)
ENAME		NOT NULL VARCHAR2(20)
JOB		NOT NULL VARCHAR2(12)
DEPT_NO		NUMBER(3)
SAL		NUMBER(7,2)
EXPERIENCE		NUMBER(5)

```
SQL> insert into emp1 values(120,'shiv','manager',13,30000,10);
```

```
1 row created.
```

```
SQL> select * from emp1;
```

EMP_NO	ENAME	JOB	DEPT_NO	SAL	EXPERIENCE
120	shiv	manager	13	30000	10

```
SQL> alter table emp1 drop(experience);
Table altered.
```

```
SQL> desc emp1;
```

Name	Null?	Type
EMP_NO		NUMBER(6)
ENAME		NOT NULL VARCHAR2(20)
JOB		NOT NULL VARCHAR2(12)
DEPT_NO		NUMBER(3)
SAL		NUMBER(7,2)

```
SQL> truncate table emp1;
```

Table truncated.

SQL> desc emp1;

Name	Null?	Type

EMP_NO		NUMBER(6)
ENAME		NOT NULL VARCHAR2(20)
JOB		NOT NULL VARCHAR2(12)
DEPT_NO		NUMBER(3)
SAL		NUMBER(7,2)

SQL> select * from emp1;
no rows selected

SQL> drop table emp1;
Table dropped.

SQL> create table dept(deptno number(5) constraint dept_deptno_pk primary key, dname
varchar2(20),loc varchar2(20));
Table created.

SQL> desc dept;

Name	Null?	Type

DEPTNO		NOT NULL NUMBER(5)
DNAME		VARCHAR2(20)
LOC		VARCHAR2(20)

SQL> create table emp(empno number(5),ename varchar2(25) not null,job varchar2(15),mgr
number(5),joindate date,salary number(7,2),comm number(7,2),deptno number(7) constraint
emp_deptno_fk references dept(deptno));

Table created.

SQL> desc emp;

Name	Null?	Type

EMPNO		NUMBER(5)
ENAME		NOT NULL VARCHAR2(25)
JOB		VARCHAR2(15)
MGR		NUMBER(5)
JOINDATE		DATE
SALARY		NUMBER(7,2)
COMM		NUMBER(7,2)
DEPTNO		NUMBER(7)

SQL> insert into dept values(12,'computer','pune');

1 row created.

SQL> select * from dept;

DEPTNO	DNAME	LOC
12	computer	pune

SQL> insert into emp values(23,'shiv','manager',55,'12sep2019',30000,55,12); 1 row created.

SQL> select * from emp;

EMPNO	ENAME	JOB	MGR	JOINDATE	SALARY	COMM	DEPTNO
23	shiv	manager	55	12-SEP-19	30000	55	12

SQL> create table emp1(emp_no number(6),ename varchar2(20) not null,job varchar2(10) not null,dept_no number(3),sal number(7,2));
Table created.

SQL> desc emp1;

Name	Null?	Type
EMP_NO		NUMBER(6)
ENAME		NOT NULL VARCHAR2(20)
JOB		NOT NULL VARCHAR2(10)
DEPT_NO		NUMBER(3)
SAL		NUMBER(7,2)

SQL> alter table emp1 add(experience number(5));
Table altered.

SQL> desc emp1;

Name	Null?	Type
EMP_NO		NUMBER(6)
ENAME		NOT NULL VARCHAR2(20)
JOB		NOT NULL VARCHAR2(10)
DEPT_NO		NUMBER(3)
SAL		NUMBER(7,2)
EXPERIENCE		NUMBER(5)

SQL> alter table emp1 modify(job varchar2(12));
Table altered.

SQL> desc emp1;

Name	Null?	Type
------	-------	------

EMP_NO		NUMBER(6)
ENAME	NOT NULL	VARCHAR2(20)
JOB	NOT NULL	VARCHAR2(12)
DEPT_NO		NUMBER(3)
SAL		NUMBER(7,2)
EXPERIENCE		NUMBER(5)

SQL> create table dept1(dept_no number(12) primary key,d_name varchar2(10),loc varchar2(10));

Table created.

SQL> desc dept1;

Name	Null?	Type
------	-------	------

DEPT_NO	NOT NULL	NUMBER(12)
D_NAME		VARCHAR2(10)
LOC		VARCHAR2(10)

SQL> alter table emp1 modify(emp_no number(6) check (emp_no>100));

Table altered.

SQL> desc emp1;

Name	Null?	Type
------	-------	------

EMP_NO		NUMBER(6)
ENAME	NOT NULL	VARCHAR2(20)
JOB	NOT NULL	VARCHAR2(12)
DEPT_NO		NUMBER(3)
SAL		NUMBER(7,2)
EXPERIENCE		NUMBER(5)

SQL> insert into emp1 values(120,'shiv','manager',13,30000,10);

1 row created.

SQL> select * from emp1;

EMP_NO	ENAME	JOB	DEPT_NO	SAL	EXPERIENCE
--------	-------	-----	---------	-----	------------

120	shiv	manager	13	30000	10
-----	------	---------	----	-------	----

SQL> alter table emp1 drop(experience);

Table altered.

SQL> desc emp1;

Name	Null?	Type
EMP_NO		NUMBER(6)
ENAME	NOT NULL	VARCHAR2(20)
JOB	NOT NULL	VARCHAR2(12)
DEPT_NO		NUMBER(3)
SAL		NUMBER(7,2)

SQL> truncate table emp1;

Table truncated.

SQL> desc emp1;

Name	Null?	Type
EMP_NO		NUMBER(6)
ENAME	NOT NULL	VARCHAR2(20)
JOB	NOT NULL	VARCHAR2(12)
DEPT_NO		NUMBER(3)
SAL		NUMBER(7,2)

SQL> select * from emp1;

no rows selected

SQL> drop table emp1;

Table dropped.

SQL> desc emp1;

ERROR:

ORA-04043: object emp1 does not exist

SQL> create table dept(deptno number(5) constraint dept_deptno_pk primary key,dname varchar2(20),loc varchar2(20));

Table created.

SQL> desc dept;

Name	Null?	Type
DEPTNO	NOT NULL	NUMBER(5)
DNAME		VARCHAR2(20)
LOC		VARCHAR2(20)

SQL> create table emp(empno number(5),ename varchar2(25) not null,job varchar2(15),mgr number(5),joindate date,salary number(7,2),comm number(7,2),deptno number(7) constraint emp_deptno_fk references dept(deptno));

Table created.


```
SQL> desc emp;
```

Name	Null?	Type
EMPNO		NUMBER(5)
ENAME	NOT NULL	VARCHAR2(25)
JOB		VARCHAR2(15)
MGR		NUMBER(5)
JOINDATE		DATE
SALARY		NUMBER(7,2)
COMM		NUMBER(7,2)
DEPTNO		NUMBER(7)

```
SQL> insert into dept values(12,'computer','pune');
```

```
1 row created.
```

```
SQL> select * from dept;
```

DEPTNO	DNAME	LOC
12	computer	pune

```
SQL> insert into emp values(23,'shiv','manager',55,'12sep2019',30000,55,12);
```

```
1 row created.
```

```
SQL> select * from emp;
```

EMPNO	ENAME	JOB	MGR	JOINDATE	SALARY	COMM	DEPTNO
23	shiv	manager	55	12-SEP-19	30000	55	12

Conclusion: In this experiment, we have studied the concept of constraint and executed all constraints such as NULL, NOTNULL, PRIMARY KEY, UNIQUE, CHECK, DEFAULT, and REFERENCES. Also created tables with and without constraint and inserted records into it.

Experiment No-3

Aim: To Perform the following: Viewing all databases, Viewing all Tables in a Database, Updating/Deleting Records in a Table

Theory:

View:

- A database view is a virtual table or logical table which is defined as a SQL SELECT query with joins. Because a database view is similar to a database table, which consists of rows and columns, so you can query data against it. Most database management systems, including MySQL, allows you to update data in the underlying tables through the database view with some prerequisites.
- A database view is dynamic because it is not related to the physical schema. The database system stores database views as a SQL SELECT statement with joins. When the data of the tables changes, the view reflects that changes as well.

Advantages of database view:

- A database view allows you to simplify complex queries: a database view is defined by an SQL statement that associates with many underlying tables. You can use a database view to hide the complexity of underlying tables to the end-users and external applications. Through a database view, you only have to use simple SQL statements instead of complex ones with many joins.
- A database view helps limit data access to specific users: You may not want a subset of sensitive data that can be queryable by all users. You can use database views to expose only non-sensitive data to a specific group of users.
- A database view provides an extra security layer. Security is a vital part of any relational database management system. Database views provide extra security for a database management system. A database view allows you to create only a read-only view to expose read-only data to specific users. Users can only retrieve data in the read-only view but cannot update it.
- A database view enables computed columns. A database table should not have calculated columns however a database view should.
- Database view enables backward compatibility. Suppose you have a central database,

which many applications are using it. One day you decided to redesign the database to adapt to the new business requirements. You remove some tables and create several new tables, and you don't want the changes to affect other applications. In this scenario, you can create database views with the same schema as the legacy tables that you have removed.

Disadvantages of database view:

Besides the advantages above, there are several disadvantages of using database views:

- Performance: querying data from a database view can be slow especially if the view is created based on other views.
- Tables dependency: you create a view based on underlying tables of the database. Whenever you change the structure of those tables that view associates with, you have to change the view as well.

DDL COMMAND ON VIEW:

1. CREATE : Syntax : CREATE VIEW view_name AS SELECT column_name(s)
FROM table_name WHERE condition;

Example: CREATE VIEW Emp_View AS Select * From emp;

```
mysql> select * from emp;
```

```
+-----+-----+-----+-----+
| empid | emp_name | salary | Dept      |
+-----+-----+-----+-----+
| 1 | XYZ   | 10000 | COMPUTER |
| 2 | PQR   | 12000 | CIVIL    |
| 3 | LMN   | 15000 | E&TC     |
| 4 | ABC   | 25000 | COMPUTER |
+-----+-----+-----+-----+
```

4 rows in set (0.00 sec)

```
mysql> CREATE VIEW VEMP AS select * from
emp; Query OK, 0 rows affected (0.05 sec)
```

```
mysql> select * from VEMP;
```

```
+-----+-----+-----+-----+
| empid | emp_name | salary | Dept      |
+-----+-----+-----+-----+
| 1 | XYZ   | 10000 | COMPUTER |
| 2 | PQR   | 12000 | CIVIL    |
| 3 | LMN   | 15000 | E&TC     |
| 4 | ABC   | 25000 | COMPUTER |
+-----+-----+-----+-----+
```

+-----+-----+-----+-----+
4 rows in set (0.00 sec)

2. ALTER: Once a view is defined, you can modify it by using the ALTER VIEW statement. The syntax of the ALTER VIEW statement is similar to the CREATE VIEW statement except the CREATE keyword is replaced by the ALTER keyword

Syntax : ALTER VIEW view_name AS SELECT column_name(s)
FROM table_name WHERE condition;

Example: ALTER VIEW Emp_View AS Select * From emp;

3. DROP: Syntax: DROP VIEW [IF EXISTS] [database_name].[view_name];

Example: DROP VIEW emp_view; or DROP VIEW EMP.emp_view; //
EMP is Database name.

4. TRUNCATE: We can't do truncate on view.

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

Insert Command 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 columns 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 a 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.

Viewing all databases:

Show databases;

Viewing all tables

Show tables

Q1: Insert a single record into the dept table.

Ans: SQL> insert into dept values (1,'IT','Tholudur');1

row created.

Q2: Insert more than a record into emp table using a single insert command.

Ans: SQL> 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.

SQL> / 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.

SQL> / Enter value for empno: 3

Enter value for ename: Gungan

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,'Gungan','ASP',1,12000)

1 row created.

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

Ans: SQL> select * from emp;

EMPNO ENAME JOB DEPTNO

SAL

1	Mathi	AP	1	10000
2	Arjun	ASP	2	12000
3	Gugan	ASP	1	12000

SQL> update emp set sal=15000 where job='ASP'; 2 rows updated. SQL>

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: SQL> create table employee as select * from emp;

Table created.

SQL> desc employee;

Name Null? Type

-EMPNO	NUMBER(6)
ENAME	NOT NULL
VARCHAR2(20)	JOB NOT NULL
VARCHAR2(13)	DEPTNO
NUMBER(3)	
SAL	NUMBER(7,2)

Q5: select employee name, job from the emp table Ans:

SQL> select ename, job from emp;

ENAME JOB

Mathi	AP
Arjun	ASP
Gugan	ASP
Karthik	Prof
Akalya	AP
Suresh	lect

6 rows were selected.

Q6: Delete only those who are working as lecturersAns:

SQL> 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 were selected.

SQL> delete from emp where job='lect';1

row deleted.

SQL> 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 order by salary in ascending order.Ans:

SQL> select * from emp order by sal;

EMPNO	ENAME	JOB	DEPTNO	SAL
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 order by salary in descending order.

Ans: SQL> select * from emp order by sal desc;

EMPNO ENAME JOB DEPTNO SAL

4 Karthik Prof 2 30000

2 Arjun ASP 2 15000

3 Gungan 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: SQL> select * from emp where deptno=1;

EMPNO ENAME JOB DEPTNO SAL

-1 Mathi AP 1 10000

3 Gungan 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 a distinct clause for the deptno.

Ans: SQL> select distinct deptno from emp;

DEPTNO

1

2

Conclusion: In this experiment, we have studied the concept of Data Manipulation Language and implemented Insert, Select, Update, Delete. Also implemented queries to View all databases, all Tables in a Database.

Experiment No. - 4

Aim: To Perform the following SQL query on database:

Altering a Table, Dropping/Truncating/Renaming Tables, Backing up / Restoring a Database

Theory:

Structured Query Language(SQL) as we all know is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database. SQL uses certain commands like Create, Drop, Insert, etc.

Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database. DDL is a set of SQL commands used to create, modify, and delete database structures but not data. These commands are normally not used by a general user, who should be accessing the database via an application.

List of DDL commands:

- **CREATE:** This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).
- **DROP:** This command is used to delete objects from the database.
- **ALTER:** This is used to alter the structure of the database.
- **TRUNCATE:** This is used to remove all records from a table, including all spaces allocated for the records are removed.
- **RENAME:** This is used to rename an object existing in the database.

The SQL CREATE TABLE Statement

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

Syntax:

```
CREATE TABLE table_name (  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
    ....  
);
```

The SQL DROP TABLE Statement

The DROP TABLE statement is used to drop an existing table in a database.

Syntax:

```
DROP TABLE table_name;
```

SQL ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

To add a column in a table, use the following syntax:

Syntax:

```
ALTER TABLE table_name  
ADD column_name datatype;
```

ALTER TABLE - DROP COLUMN

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

Syntax:

```
ALTER TABLE table_name  
DROP COLUMN column_name;
```

DCL (Data Control Language):

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

List of DCL commands:

- **GRANT:** This command gives users access privileges to the database.
- **REVOKE:** This command withdraws the user's access privileges given by using the GRANT command. Though many resources claim there to be another category of SQL clauses

List of TCL commands:

- **COMMIT:** Commits a Transaction.
- **ROLLBACK:** Rollbacks a transaction in case of any error occurs.
- **SAVEPOINT:** Sets a savepoint within a transaction.

The SQL BACKUP DATABASE Statement

The BACKUP DATABASE statement is used in SQL Server to create a full back up of an existing SQL database.

Syntax:

```
BACKUP DATABASE databasename  
TO DISK = 'filepath';
```

BACKUP DATABASE Example

The following SQL statement creates a full back up of the existing database "testDB" to the D disk:

Example:

```
BACKUP DATABASE testDB  
TO DISK = 'D:\backups\testDB.bak';
```

Output:

1. The Create Table Command: - Defines each column of the table uniquely. Each column has a 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 number(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 number(7,2));

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>(<newsizesize>)); Ex:
alter table emp modify(ename varchar2(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;

7. Destroying tables.

Syntax:

Drop table <tablename>;

Ex: drop table emp;

Conclusion: In this experiment we studied Data Definition Language all structure related queries such as create, alter with add and modify option, rename, drop, truncate and backup database.

Experiment No. – 5

Aim: For a given set of relation schemes, create tables and perform the following Simple Queries:

Simple Queries with Aggregate functions, Queries with Aggregate functions (group by and having clause), Queries involving- Date Functions, String Functions, Math Functions.

Theory:

Aggregate functions perform a calculation on a set of values and return a single value. There are different types of aggregate functions such as min, max, sum, avg, count, etc.

Why use aggregate functions: From a business perspective, different organization levels have different information requirements. Top levels managers are usually interested in knowing whole figures and not necessary the individual details.

Aggregate functions allow us to easily produce summarized data from our database.

Aggregate Functions are all about

- Performing calculations on multiple rows
- Of a single column of a table
- And returning a single value.

IMPLEMENTATION OF DATA AND BUILT IN FUNCTIONS IN SQL

CHARACTER/STRING FUNCTION:

```
SQL> select upper('welcome') from dual;
```

```
-----
```

```
WELCOM
```

```
E
```

```
SQL> select upper('hai') from dual;
```

```
---
```

```
HA
```

```
I
```

```
SQL> select lower('HAI') from dual;
```

LOW

hai

SQL> select initcap('hello world') from dual;

INITCAP('Hello world')

Hello World

SQL> select ltrim(' hai') from dual;

LTR

hai

SQL> select rtrim('hai ')from dual;

RTR

hai

SQL> select rtrim(' hai ')from dual;

RTRIM('hai')

hai

SQL> select concat('GHRCEM',' Pune')from dual;

GHRCEM Pune

SQL> select length('SRM')from dual;

LENGTH('SRM')

12

```
SQL> select replace('SRM university', 'SRM','Anna')from dual;
```

```
-----
```

```
Anna university
```

```
SQL> select substr('SRM', 7,6)from dual;
```

```
SUBSTR
```

```
-----
```

```
lingam
```

```
SQL> select rpad('hai',3,'*')from dual;
```

```
RPAD('
```

```
-----
```

```
hai***
```

```
SQL> select lpad('hai',3,'*')from dual;
```

```
LPAD
```

```
-----
```

```
***hai
```

```
SQL> select replace('Dany','y','ie')from dual;
```

```
REPLACE
```

```
-----
```

```
Danie
```

```
SQL> select translate('cold','ld','ol')from dual;
```

```
TRANSL
```

```
-----
```

```
cool
```

SET OPERATORS

Set operations:

Union/ Intersect/ Except operations – These operations operate on relations, which must be compatible i.e. they must have the same no. of attributes with the same domain types.

Syntax:

(select query1) Union/ Intersect/Except (select query2)

Here set operations are applied to tuples in the results of multiple select queries. All these operations eliminate duplicate tuples from the result.

```
SELECT customerNumber id, contactLastname name FROM customers
UNION
SELECT employeeNumber id, firstname name FROM employees
Output:
id name
```

```
-----
103 Schmitt
112 King
114 Ferguson
119 Labrune
121 Bergulfsen
```

```
124 Nelson
125 Piestrzeniewicz
128 Keitel
129 Murphy
131 Lee
```

```
ubuntu@ubuntu-OptiPlex-380:~$ MySQL -u root -p
```

```
Enter password:
```

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

```
Your MySQL connection id is 36
```

```
Server version: 5.5.62-0ubuntu0.14.04.1 (Ubuntu)
```

```
Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.
```

```
Oracle is a registered trademark of Oracle Corporation and/or its
```

```
affiliates. Other names may be trademarks of their respective
owners.
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> show databases;
```

```
+-----+
| Database |
```

```

+-----+
| information_schema |
| anya              |

| book              |
| first             |
| info              |
| jn                 |
| job                |
| last              |
| MySQL              |

| performance_schema |
| se                  |
| sec                 |
| sece                |
| second             |
| suraj              |
| third              |
| universalbank      |
| viju                |
+-----+
18 rows in set (0.13 sec)

```

mysql> use mysql;

Reading table information for completion of table and column names You
can turn off this feature to get a quicker startup with -A

Database changed

mysql> create table product(id int(4),pname varchar(9) NOT NULL,pprice int(8) NOT
NULL,discount int(5));Query OK, 0 rows affected (0.08 sec)

mysql> desc product;

```

+-----+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(4) | YES  |     | NULL    |       |
| pname | varchar(9) | NO  |     | NULL    |       |
| pprice | int(8) | NO   |     | NULL    |       |
| discount | int(5) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)

```

mysql> insert into product VALUES(1,'AMIT',2000,100);
Query OK, 1 row affected (0.04 sec)

```
mysql> insert into product VALUES(2,'CORNFLAKES',800,50);
Query OK, 1 row affected, 1 warning (0.04 sec)
```

```
mysql> insert into product VALUES(3,'PARLE',90,5);
Query OK, 1 row affected (0.04 sec)
```

```
mysql> insert into product VALUES(4,'NESTLE',70,2);
Query OK, 1 row affected (0.04 sec)
```

```
mysql> insert into product VALUES(5,'CADBOURY',100,10);
Query OK, 1 row affected (0.04 sec)
```

```
mysql> insert into product VALUES(6,'AMUL',200,20);
Query OK, 1 row affected (0.03 sec)
```

```
mysql> insert into product VALUES(7,'JAM',70,8);
Query OK, 1 row affected (0.04 sec)
```

```
mysql> select * from product;
```

id	pname	pprice	discount
1	AMIT	2000	100
2	CORNFLAKE	800	50
3	PARLE	90	5
4	NESTLE	70	2
5	CADBOURY	100	10
6	AMUL	200	20
7	JAM	70	8

```
7 rows in set (0.00 sec)
```

```
mysql> select count(id) AS Numberofproducts FROM product;
```

Numberofproducts
7

```
1 row in set (0.00 sec)
```

```
mysql> select Sum(pprice) AS Totalprice FROM product;
```

Totalprice
3330

```
1 row in set (0.00 sec)
```

```
mysql> select Max(discount) AS Largestdiscount FROM product;
```

```
+-----+
| Largestdiscount |
+-----+
|          100 |
```

```
+-----+
1 row in set (0.00 sec)
```

```
mysql> select Min(pprice) AS Smallestprice FROM product;
```

```
+-----+
| Smallestprice |
+-----+
|           70 |
```

```
+-----+
1 row in set (0.00 sec)
```

```
mysql> select Avg(pprice) AS AveragePrice FROM product;
```

```
+-----+
| AveragePrice |
+-----+
|    475.7143 |
```

```
+-----+
1 row in set (0.00 sec)
```

Conclusion: In this experiment, we have studied aggregate functions and Transaction Control Language and implemented Count, Sum, Max, Min, Avg, Commit, Rollback, and Savepoint commands. Simple Queries with Aggregate functions, Queries with Aggregate functions (group by and having clause), Queries involving- Date Functions, String Functions, Math Functions.

Experiment No. - 6

Aim: To perform SQL query that demonstrate Join Queries- Inner Join, Outer Join, Left join, Right Join

Theory:

SQL JOIN: A JOIN clause is used to combine rows from two or more tables, based on a related column between them. Following are different types of join.

1. Equi-join/Inner join
2. Non-equi-join
3. Self-join
4. Outer join

Equi-join:

A join that is based on equalities is called equi-join. '=' operator is used in equi- join comparison. It retrieves rows from tables having a common column. It is also called simple join.

Non-equi-join:

A join that specifies the relationship between columns belonging to different tables by making use of the relational operators (<, >, <=, >=, !=) other than the '=' operator is called as non-equi-join

Self-join:

Joining a table to itself is known as self-join i.e. it joins one row in a table to another. It can compare each row of the table to itself and also with other rows of the same table.

Outer join:

An outer join returns all the rows returned by simple join or equi join as well as those rows from one table that do not match any row from the other table.

The symbol (+) represents outer join.

Implementation:

1. Equi join:

```
select e.empno, e.ename, e.dept, d.deptno, d.loc, d.dname from emp e, dept d
where e.dept=d.deptno;
```

Output:

EMPNO	ENAME	DEPT	DEPTNO	LOC	DNAME
1001	Nilesh Joshi	10	10	Fourth	Computer
1002	Avinash Pawar	30	30	Second	Electrical
1003	Amit Kumar	30	30	Second	Electrical
1005	Niraj Sharma		20	20	First Mechanical
1006	Pushkar Deshpande	30	30	Second	Electrical
1007	Sumit Patil	20	20	First	Mechanical
1008	ravi sawant	20	20	First	Mechanical

2. Non-equi-join:

Select e.ename, e.salary, s.grade from emp e, salgrade s
where e.salary>=s.losal and e.salary<=s.hisal;

Output:

ENAME	SALARY	GRADE
Amit Kumar	2000	3
Nilesh Joshi	2800	3
Avinash Pawar	5000	4
Pushkar Deshpande	6500	4

3. Self Join:

select worker.ename "employee", manager.ename "manager" from emp worker, empmanager
where worker.mgr=manager.empno;

Output:

Employee Manager	Avinash
Pawar Amit Kumar	Pushkar
Deshpande Amit Kumar	Niraj
Sharma	Amit
Kumar	
Sumit Patil	nitin kulkarni
Amit kumar	nitin kulkarni

DEPARTMENT

INSERT INTO DEPARTMENT

VALUES(&DNO,'&DNAME',&MGRSSN,&MGRSTARTDATE');SELECT * FROM

DEPARTMENT;

DNO DNAME	MGRSTAR TD
MGRSSN	-----

10-
AUG-

1 RESEARCH	111111	12
2 ACCOUNTS	222222	10-AUG-10
3 AI	333333	15-APR-12
4 NETWORKS	111111	18-MAY-14
5 BIGDATA	666666	21-JAN-10

5 rows were selected.

EMPLOYEE

```
INSERT INTO EMPLOYEE
VALUES('&SSN','&NAME','&ADDRESS','&SEX','&SALARY','&SUPERSSN',& DNO);
```

```
SELECT * FROM EMPLOYEE;
```

SSN	NAME	ADDRESS	SEX	SALARY	SUPERSSN	DNO
111111	RAJ	BENGALURU	M	700000		1
222222	RASHMI	MYSORE	F	400000	111111	2
333333	RAGAVI	TUMKUR	F	800000		3
444444	RAJESH	TUMKUR	M	650000	333333	3
555555	RAVEESH	BENGALURU	M	500000	333333	3
666666	SCOTT	ENGLAND	M	700000	444444	5
777777	NIGANTH	GUBBI	M	200000	222222	2
888888	RAMYA	GUBBI	F	400000	222222	3
999999	VIDYA	TUMKUR	F	650000	333333	3
100000	GEETHA	TUMKUR	F	800000		3

10 rows selected.

DLOCATION

```
INSERT INTO DLOCATION VALUES(&DNO,'&DLOC');
```

```
SELECT * FROM DLOCATION;
```

DNO	DLOC
1	MYSORE
1	TUMKUR
2	BENGALURU
3	GUBBI
5	DELHI

> BENGALURU

6 rows were selected.

PROJECT

INSERT INTO PROJECT

VALUES(&PNO,'&PNAME','&PLOCATION','&DNO');SELECT * FROM
PROJECT;

PNO	PNAME	PLOCATION	DNO
-----	-----	-----	-----
111	IOT	GUBBI	3
222	TEXTSPEECH	GUBBI	3
333	IPSECURITY	DELHI	4
444	TRAFICANAL	BENGALUR	5
555	CLOUDSEC	U	1

5 rows were selected.

WORKS_ON

INSERT INTO WORKS_ON VALUES('&SSN',&PNO,&HOURS);

SELECT * FROM WORKS_ON;

SSN	PNO	HOURS
-----	-----	-----
666666	333	4
666666	111	2
111111	222	3
555555	222	2
333333	111	4
444444	111	6
222222	111	2

8 rows were selected.

- > Make a list of all project numbers for projects that involve an employee whose last name is 'Scott', either as a worker or as a manager of the department that controls the project.

```
(SELECT DISTINCT PNO
FROM PROJECT P, DEPARTMENT D,
EMPLOYEE E WHERE P.DNO=D.DNO
AND
SSN=MGRSSN
AND
NAME='SCOTT')
```

```
UNION
(SELECT DISTINCT P.PNO
FROM PROJECT P, WORKS_ON W,
EMPLOYEE E WHERE P.PNO=W.PNO
AND
W.SSN=E.SSN
AND
NAME='SCOTT');
```

PNO

111

333

444

- > Show the resulting salaries if every employee working on the 'IoT' project is given a 10 percent raise.

```
SELECT FNAME, LNAME, 1.1*SALARY AS
INCR_SALFROM EMPLOYEE E, WORKS_ON W,
PROJECT P
WHERE E.SSN=W.SSN
ANDW.PNO=P.PNO AND
P.PNAME='IOT';
```

SSN	NAME SEX	ADDRESS	SALARY SUPERSSN	DNO
-----	-----		-----	-----

111111	RAJ	BENGALURU	M	700000		1
222222	RASHMI	MYSORE	F	440000	111111	2
333333	RAGAVI	TUMKUR	F	880000		3
444444	RAJESH	TUMKUR	M	715000	333333	3
555555	RAVEESH	BENGALURU	M	500000	333333	3
666666	SCOTT	ENGLAND	M	770000	444444	5
777777	NIGANTH	GUBBI	M	200000	222222	2
	RAMYA	GUBBI	F	400000	222222	3
888888						
999999	VIDYA	TUMKUR	F	650000	333333	3
100000	GEETHA	TUMKUR	F	800000		3

1. rows selected.

3. Find the sum of the salaries of all employees of the 'Accounts' department, as well as the maximum salary, the minimum salary, and the average salary in this department.

```

SELECT      SUM(SALARY),      MAX(SALARY),
            MIN(SALARY),AVG(SALARY) FROM EMPLOYEE E,
DEPARTMENT D
WHERE DNAME='ACCOUNTS'
      AND D.DNO=E.DNO;

SUM(SALARY) MAX(SALARY) MIN(SALARY) AVG(SALARY)

```

```

-----
40000  200000  320000
mysql> create database jn;
Query OK, 1 row affected (0.00 sec)

```

```

mysql> use jn;
Database changed
mysql> create table orders(orderID int NOT NULL,custID int NOT NULL,empID
VARCHAR(10) NOT NULL,PRIMARY KEY(orderID));
Query OK, 0 rows affected (0.08 sec)

```

```

mysql> DESC orders;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| orderID | int(11)   | NO   | PRI | NULL    |       |
| custID  | int(11)   | NO   |     | NULL    |       |
| empID   | varchar(10) | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

```

mysql> create table customers(custID int NOT NULL,cust_name VARCHAR(30) NOT
NULL,city varchar(20) NOT NULL);
Query OK, 0 rows affected (0.07 sec)

```

```

mysql> desc customers;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| custID | int(11)   | NO   |     | NULL    |       |

```

```

| cust_name | varchar(30) | NO   |     | NULL    |       |
| city      | varchar(20) | NO   |     | NULL    |       |

```

```

+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

```

mysql> INSERT INTO orders VALUES(101,1,'emp1');
Query OK, 1 row affected (0.04 sec)

```

```

mysql> INSERT INTO orders VALUES(103,2,'emp2');
Query OK, 1 row affected (0.03 sec)

```

```

mysql> INSERT INTO orders VALUES(136,3,'emp3');
Query OK, 1 row affected (0.02 sec)

```

```

mysql> INSERT INTO customers VALUES(5,'riya','delhi');

```

Query OK, 1 row affected (0.05 sec)

```
mysql> INSERT INTO customers VALUES(7,'raj','goa');
```

Query OK, 1 row affected (0.03 sec)

```
mysql> INSERT INTO customers VALUES(8,'priya','pune');
```

Query OK, 1 row affected (0.04 sec)

```
mysql> SELECT orders.orderID,customers.cust_name,customers.city FROM orders INNER  
JOIN customers ON orders.custID=customers.custID;  
Empty set (0.00 sec)
```

```
mysql> INSERT INTO customers VALUES(2,'puja','pune');
```

Query OK, 1 row affected (0.03 sec)

//INNER JOIN

```
mysql> SELECT orders.orderID,customers.cust_name,customers.city FROM orders INNER  
JOIN customers ON orders.custID=customers.custID;
```

orderID	cust_name	city
103	puja	pune

1 row in set (0.00 sec)

```
mysql> INSERT INTO customers VALUES(3,'divya','mumbai');
```

Query OK, 1 row affected (0.05 sec)

```
mysql> INSERT INTO customers VALUES(1,'jay','mumbai');
```

Query OK, 1 row affected (0.04 sec)

```
mysql> SELECT orders.orderID,customers.cust_name,customers.city FROM orders INNER  
JOIN customers ON orders.custID=customers.custID;
```

orderID	cust_name	city
103	puja	pune
136	divya	mumbai
101	jay	mumbai

3 rows in set (0.00 sec)

```
mysql> INSERT INTO orders VALUES(171,7,'emp7');
```

Query OK, 1 row affected (0.04 sec)

```
mysql> INSERT INTO orders VALUES(172,6,'emp8');
Query OK, 1 row affected (0.05 sec)
```

```
mysql> SELECT orders.orderID,customers.cust_name,customers.city FROM orders INNER
JOIN customers ON orders.custID=customers.custID;
```

```
+-----+-----+-----+
| orderID | cust_name | city |
+-----+-----+-----+
| 171 | raj | goa |
| 103 | puja | pune |
| 136 | divya | mumbai |
| 101 | jay | mumbai |
+-----+-----+-----+
4 rows in set (0.00 sec)
```

```
// LEFT JOIN
```

```
mysql> SELECT orders.orderID, customers.cust_name FROM orders LEFT JOIN customers
ON orders.custID=customers.custID;
```

```
+-----+-----+
| orderID | cust_name |
+-----+-----+
| 101 | jay |
| 103 | puja |
| 136 | divya |
| 171 | raj |
| 172 | NULL |
+-----+-----+
```

```
+-----+-----+
5 rows in set (0.00 sec)
```

```
mysql> select * from orders;
```

```
| orderID | custID | empID |
+-----+-----+-----+
| 101 | 1 | emp1 |
| 103 | 2 | emp2 |
| 136 | 3 | emp3 |
| 171 | 7 | emp7 |
| 172 | 6 | emp8 |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

```
mysql> SELECT orders.orderID,customers.cust_name,customers.city FROM orders LEFT JOIN
customers ON orders.custID=customers.custID;
```

```
+-----+-----+-----+
| orderID | cust_name | city |
+-----+-----+-----+
| 101 | jay | mumbai |
+-----+-----+-----+
```

	orderID	cust_name	city
	103	puja	pune
	136	divya	mumbai
	171	raj	goa
	172	NULL	NULL

5 rows in set (0.00 sec)

RIGHT JOIN

```
mysql> SELECT orders.orderID,customers.cust_name,customers.city FROM orders RIGHT JOIN customers ON orders.custID=customers.custID;
```

	orderID	cust_name	city
	NULL	riya	delhi
	171	raj	goa
	NULL	priya	pune
	103	puja	pune
	136	divya	mumbai
	101	jay	mumbai

6 rows in set (0.00 sec)

```
mysql> select * from customers;
```

	custID	cust_name	city
	5	riya	delhi
	7	raj	goa
	8	priya	pune
	2	puja	pune

	3	divya	mumbai
--	---	-------	--------

	1	jay	mumbai
--	---	-----	--------

6 rows in set (0.00 sec)

```
mysql> SELECT * FROM orders JOIN customers ON orders.custID=customers.custID;
```

	orderID	custID	empID	custID	cust_name	city
	171	7	emp7	7	raj	goa
	103	2	emp2	2	puja	pune
	136	3	emp3	3	divya	mumbai
	101	1	emp1	1	jay	mumbai

```
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

//EQUI JOIN

```
mysql> SELECT orders.orderID,customers.cust_name,customers.city FROM orders JOIN
customers ON orders.custID=customers.custID;
```

```
+-----+-----+-----+
| orderID | cust_name | city |
+-----+-----+-----+
| 171 | raj | goa |
| 103 | puja | pune |
| 136 | divya | mumbai |
| 101 | jay | mumbai |
+-----+-----+-----+
```

4 rows in set (0.00 sec)

//NON-EQUI JOIN

```
mysql> SELECT orders.orderID,customers.cust_name,customers.city FROM orders JOIN
customers WHERE orders.custID BETWEEN 3 AND 10;
```

```
+-----+-----+-----+
| orderID | cust_name | city |
+-----+-----+-----+
| 136 | riya | delhi |
| 171 | riya | delhi |
| 172 | riya | delhi |
| 136 | raj | goa |
| 171 | raj | goa |
| 172 | raj | goa |
| 136 | priya | pune |
| 171 | priya | pune |
| 172 | priya | pune |
```

```
| 136 | puja | pune |
| 171 | puja | pune |
| 172 | puja | pune |
```

```
| 136 | divya | mumbai |
| 171 | divya | mumbai |
| 172 | divya | mumbai |
| 136 | jay | mumbai |
| 171 | jay | mumbai |
| 172 | jay | mumbai |
+-----+-----+-----+
```

18 rows in set (0.00 sec)

Conclusion: In this experiment, we have studied SQL queries for suitable database applications using SQL

DML Statements and implemented all types of Join.

Experiment No-7

AIM: To perform SQL query that demonstrate following:

Search conditions, Summary queries, Sub- queries, Subqueries- With IN clause, With EXISTS clause

Theory:

In SQL a Subquery can be simply defined as a query within another query. In other words we can say that a Subquery is a query that is embedded in WHERE clause of another SQL query. Important rules for Subqueries:

- You can place the Subquery in a number of SQL clauses: WHERE clause, HAVING clause, FROM clause. Subqueries can be used with SELECT, UPDATE, INSERT, DELETE statements along with expression operator. It could be equality operator or comparison operator such as =, >, <, <= and Like operator.
- A subquery is a query within another query. The outer query is called as **main query** and inner query is called as **subquery**.
- The subquery generally executes first when the subquery doesn't have any **co-relation** with the **main query**, when there is a co-relation the parser takes the decision **on the fly** on which query to execute **on precedence** and uses the output of the subquery accordingly.
- Subquery must be enclosed in parentheses.
- Subqueries are on the right side of the comparison operator.
- ORDER BY command **cannot** be used in a Subquery. GROUPBY command can be used to perform same function as ORDER BY command.
- Use single-row operators with singlerow Subqueries. Use multiple-row operators with multiple-row Subqueries.

Syntax: There is not any general syntax for Subqueries. However, Subqueries are seen to be used most frequently with SELECT statement as shown below:

```
SELECT column_name  
FROM table_name  
WHERE column_name expression operator  
( SELECT COLUMN_NAME from TABLE_NAME WHERE ... );
```

Sample Table: Database

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	Chennai	9988775566
Raj	102	Coimbatore	8877665544
Sasi	103	Madurai	7766553344
Ravi	104	Salem	8989898989
Sumathi	105	Kanchipuram	8989856868

Student

NAME	ROLL_NO	SECTION
Ravi	104	A

Sumathi105 B
Raj 102 A

Sample Queries:

- To display NAME, LOCATION, PHONE_NUMBER of the students from DATABASE table whose section is A
Select NAME, LOCATION, PHONE_NUMBER from DATABASE
WHERE ROLL_NO IN
(SELECT ROLL_NO from STUDENT where SECTION='A');
Explanation : First subquery executes “ SELECT ROLL_NO from STUDENT where SECTION='A' ” returns ROLL_NO from STUDENT table whose SECTION is ‘A’. Then outer-query executes it and return the NAME, LOCATION, PHONE_NUMBER from the DATABASE table of the student whose ROLL_NO is returned from inner subquery. Output:

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
------	---------	----------	--------------

Ravi	104	Salem	8989898989
------	-----	-------	------------

Raj	102	Coimbatore	8877665544
-----	-----	------------	------------

- Insert Query Example:

Table1: Student1

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
------	---------	----------	--------------

Ram	101	chennai	9988773344
Raju	102	coimbatore	9090909090
Ravi	103	salem	8989898989

Table2: Student2

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
------	---------	----------	--------------

Raj	111	chennai	8787878787
Sai	112	mumbai	6565656565
Sri	113	coimbatore	7878787878

- To insert Student2 into Student1 table:
INSERT INTO Student1 SELECT * FROM Student2;

- Output:

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
------	---------	----------	--------------

Ram	101	chennai	9988773344
Raju	102	coimbatore	9090909090
Ravi	103	salem	8989898989
Raj	111	chennai	8787878787
Sai	112	mumbai	6565656565
Sri	113	coimbatore	7878787878

- To delete students from Student2 table whose rollno is same as that in Student1 table and having location as chennai
DELETE FROM Student2

WHERE ROLL_NO IN (SELECT ROLL_NO
FROM Student1

WHERE LOCATION = 'chennai');

- Output:

1 row delete successfully.

- **Display Student2 table:**

NAMEROLL_NOLOCATIONPHONE_NUMBER

Sai 112 mumbai 6565656565

Sri 113 coimbatore 7878787878

- To update name of the students to geeks in Student2 table whose location is same as Raju, Ravi in Student1 table

UPDATE Student2

SET NAME='geeks'

WHERE LOCATION IN (SELECT LOCATION

FROM Student1

WHERE NAME IN ('Raju','Ravi'));

- Output:

1 row updated successfully.

- **Display Student2 table:**

NAMEROLL_NOLOCATIONPHONE_NUMBER

Sai 112 mumbai 6565656565

geeks 113 coimbatore 7878787878

Conclusion: In this experiment concept of subquery single row subquery, multiple row subquery and multiple column subqueries studied and implemented.

Experiment No-8

AIM: To perform SQL query for extracting data from more than one table using SQL concept

Theory:

-- create database

create database stud;

use stud;

```
-- create table and insert records for student
create table student(rollno int, sname varchar(30), dept varchar(10));
```

```
insert into student values (137, "shubham", "comp");
insert into student values (134, "sandeep", "IT");
insert into student values (130, "niyamat", "civil");
insert into student values (129, "faizan", "mech");
insert into student values (140, "vishal", "entc");
insert into student values (128, "himang", "mech");
insert into student values (131, "prachi", "comp");
insert into student values (139, "tanishka", "mech");
```

```
select * from student;
```

```
-- create table and insert records for department
create table department(deptid int, dept varchar(30), hod varchar(30));
```

```
insert into department values(1, "comp", "abc");
insert into department values(2, "IT", "pqr");
insert into department values(3, "civil", "mnq");
insert into department values(4, "mech", "xyz");
insert into department values(5, "entc", "dhr");
```

```
select * from department;
```

```
-- create table and insert records for coresubject
create table coresubject(subjectcode int, subjectname varchar(20), dept varchar(10));
```

```
insert into coresubject values(11, "OS", "comp");
insert into coresubject values(12, "CN", "comp");
insert into coresubject values(13, "OOP", "comp");
insert into coresubject values(21, "networking", "IT");
insert into coresubject values(31, "construction building", "civil");
insert into coresubject values(32, "maths-4", "civil");
insert into coresubject values(41, "thermodynamics", "mech");
insert into coresubject values(51, "digital communi", "entc");
```

```
select * from coresubject;
```

```
-- extracting data of the student who has core subject from his field
select student.rollno, student.sname, department.deptid, department.dept, coresubject.subjectcode,
coresubject.subjectname from student join department on student.dept = department.dept join
coresubject on department.dept = coresubject.dept
```

Conclusion: In this experiment we implemented SQL query for extracting data from more than one table using SQL concept

Experiment No. 9

AIM: To perform SQL query to understand the concepts: Transaction, ROLL BACK, COMMIT & CHECK POINTS.

Theory:

TCL (Transaction Control Language) :

Transaction Control Language commands are used to manage transactions in the database. These are used to manage the changes made by DML-statements. It also allows statements to be grouped together into logical transactions.

Examples of TCL commands –

COMMIT: Commit command is used to permanently save any transaction into the database.

ROLLBACK: This command restores the database to last committed state. It is also used with savepoint command to jump to a savepoint in a transaction.

SAVEPOINT: Savepoint command is used to temporarily save a transaction so that you can rollback to that point whenever necessary.

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 tableAns:

SQL> Grant all on employees to departments;

Grant succeeded.

Q2: Develop a query to grant some privileges of employees table into departments table

Ans: SQL> Grant select, update, insert on departments to departments with grantoption;
Grant succeeded.

Q3: Develop a query to revoke all privileges of employees table from departments tableAns:

SQL> Revoke all on employees from departments; Revoke succeeded.

Q4: Develop a query to revoke some privileges of employees table from departments tableAns:

SQL> Revoke select, update, insert on departments from departments;

Revoke succeeded.

Q5: Write a query to implement the save point

Ans: SQL> SAVEPOINT S1;

Savepoint created.

SQL> 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

SQL> INSERT INTO EMP VALUES(5,'Akalya','AP',1,10000); 1 row created.

SQL> 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:

SQL> rollback s1; SQL> 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: SQL> COMMIT;
Commit complete.

Example:

For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than Rs. 6,00,000.

```
SELECT  DNO,
COUNT(SSN)FROM THE
EMPLOYEE

WHERE SALARY>600000 AND
      DNOIN(SELECT DNO

            FROM THE
            EMPLOYEEGROUP
            BY DNO
            HAVING COUNT(SSN)>5)

GROUP BY DNO ;
      DNO COUNT(SSN)
      -----
          3
```

ubuntu@ubuntu-OptiPlex-380:~\$ MySQL -u root -p

Enter password:

Welcome to the MySQL monitor. Commands end with; or \g.

Your MySQL connection id is 36

Server version: 5.5.62-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;

```
+-----+
| Database |
```



```
+-----+
| information_schema |
| anya              |
```

```
| book              |
| first             |
| info              |
| jn                |
| job               |
| last              |
| MySQL             |
```

```
| performance_schema |
| se                 |
| sec                |
| sece               |
| second             |
| suraj              |
| third              |
| universalbank      |
| viju               |
```

```
+-----+
18 rows in set (0.13 sec)
```

mysql> use mysql;

Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed

mysql> create table product(id int(4),pname varchar(9) NOT NULL,pprice int(8) NOT NULL,discount int(5));Query OK, 0 rows affected (0.08 sec)

mysql> desc product;

```
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | int(4) | YES  |     | NULL    |       |
| pname | varchar(9) | NO  |     | NULL    |       |
| pprice | int(8) | NO   |     | NULL    |       |
| discount | int(5) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

mysql> insert into product VALUES(1,'AMIT',2000,100);
Query OK, 1 row affected (0.04 sec)

```
mysql> insert into product VALUES(2,'CORNFLAKES',800,50);
Query OK, 1 row affected, 1 warning (0.04 sec)
```

```
mysql> insert into product VALUES(3,'PARLE',90,5);
Query OK, 1 row affected (0.04 sec)
```

```
mysql> insert into product VALUES(4,'NESTLE',70,2);
Query OK, 1 row affected (0.04 sec)
```

```
mysql> insert into product VALUES(5,'CADBOURY',100,10);
Query OK, 1 row affected (0.04 sec)
```

```
mysql> insert into product VALUES(6,'AMUL',200,20);
Query OK, 1 row affected (0.03 sec)
```

```
mysql> insert into product VALUES(7,'JAM',70,8);
Query OK, 1 row affected (0.04 sec)
```

```
mysql> select * from product;
```

```
+-----+-----+-----+-----+
| id | pname    | pprice | discount |
+-----+-----+-----+-----+
| 1 | AMIT     | 2000   | 100      |
| 2 | CORNFLAKE | 800    | 50       |
| 3 | PARLE    | 90     | 5        |
| 4 | NESTLE   | 70     | 2        |
| 5 | CADBOURY | 100    | 10       |
| 6 | AMUL     | 200    | 20       |
| 7 | JAM      | 70     | 8        |
+-----+-----+-----+-----+
```

```
7 rows in set (0.00 sec)
```

```
mysql> select count(id) AS Numberofproducts FROM product;
```

```
+-----+
| Numberofproducts |
+-----+
| 7                |
+-----+
```

```
1 row in set (0.00 sec)
```

```
mysql> select Sum(pprice) AS Totalprice FROM product;
```

```
+-----+
| Totalprice |
+-----+
| 3330      |
+-----+
```

```
1 row in set (0.00 sec)
```

```
mysql> select Max(discount) AS Largestdiscount FROM product;
```

```

+-----+
| Largestdiscount |
+-----+
|      100 |

```

```

+-----+
1 row in set (0.00 sec)

```

```
mysql> select Min(pprice) AS Smallestprice FROM product;
```

```

+-----+
| Smallestprice |
+-----+
|      70 |

```

```

+-----+
1 row in set (0.00 sec)

```

```
mysql> select Avg(pprice) AS AveragePrice FROM product;
```

```

+-----+
| AveragePrice |
+-----+
|  475.7143 |

```

```

+-----+
1 row in set (0.00 sec)

```

Conclusion: In this experiment performed Transaction Control Language such as commit, rollback, savepoint.

Content Beyond Syllabus

Open ended practical on NOSQL

Experiment No. - 10

AIM: Design and Develop MongoDB Queries using CRUD Operations.

Theory: MongoDB CRUD Operations

- Create Operations
- Read Operations
- Update Operations
- Delete Operations

Create Operations

Create or insert operations and add new documents to a collection. If the collection does not currently exist, insert operations will create the collection.

MongoDB provides the following methods to insert documents into a collection:

- `db.collection.insertOne()` *New in version 3.2*
- `db.collection.insertMany()` *New in version 3.2*

In MongoDB, insert operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

Read Operations

Read operations retrieve documents from a collection; i.e. queries a collection for documents. MongoDB provides the following methods to read documents from a collection:

- `db.collection.find()`

You can specify query filters or criteria that identify the documents to return.

Update Operations

Update operations modify existing documents in a collection. MongoDB provides the following methods to update documents of a collection:

- `db.collection.updateOne()` *New in version 3.2*
- `db.collection.updateMany()` *New in version 3.2*
- `db.collection.replaceOne()` *New in version 3.2*

In MongoDB, update operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

You can specify criteria, or filters, that identify the documents to update. These filters use the same syntax as reading operations.

For examples, see Update Documents.

Delete Operations

Delete operations remove documents from a collection. MongoDB provides the following methods to delete documents of a collection:

- `db.collection.deleteOne()` *New in version 3.2*
- `db.collection.deleteMany()` *New in version 3.2*

In MongoDB, delete operations target a single collection. All write operations in MongoDB are atomic on the level of a single document.

You can specify criteria, or filters, that identify the documents to remove. These filters use the same syntax as reading operations.

- 1.1. The Use Command:** MongoDB uses **DATABASE_NAME** is used to create a database. The command will create a new database; if it doesn't exist otherwise it will return the existing database.

Syntax: Use DATABASE_NAME;

Example: If you want to create a database with the name **<mydb>**, then **use the database** the 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, then 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
```

```
test 0.23012GB
```

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

2. MongoDB Drop Database:

2.1. The Drop Database: MongoDB **db.dropDatabase ()** command is used to drop an 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 the 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
```

3. Basic Operations with the Shell: We can use the four basic operations, create, read, update, and delete (CRUD) to manipulate and view data in the shell.

3.1. Create: The insert function adds a document to a collection. For example, suppose we want to store a blog post. First, we'll create a local variable called **post** which is a JavaScript object representing our document. It will have the keys "title", "content", and "date". (the date that it was published):

```
> post = {"title" : "My Blog Post",  
... "content": "Here's my blog post.",  
... "date" : new Date()}  
{  
  "title" : "My Blog Post",  
  "content" : "Here's my blog post.",  
  "date" : ISODate("2012-08-24T21:12:09.982Z")  
}
```

This object is a valid MongoDB document, so we can save it to the blog collection using the insert method:

```
> db.blog.insert(post)
```

- 3.2. Read:** Find and findOne can be used to query a collection. If we just want to see one document from a collection, we can use findOne.

```
> db.blog.findOne()
{
  "_id" : ObjectId("5037ee4a1084eb3ffef7228"),
  "title" : "My Blog Post",
  "content" : "Here's my blog post.",
  "date" : ISODate("2012-08-24T21:12:09.982Z")
}
```

Find and findOne can also be passed criteria in the form of a query document. This will restrict the documents matched by the query. The shell will automatically display up to 20 documents matching a find, but more can be fetched. See Chapter 4 for more information on querying.

- 3.3. Update:** If we would like to modify our post, we can use an update. The update takes (at least) two parameters: the first is the criteria to find which document to update, and the second is the new document. Suppose we decide to enable comments on the blog post we created earlier. We'll need to add an array of comments as the value for a new key in our document.

The first step is to modify the variable post and add a "comments" key:

```
> post.comments = []
[ ]
```

Then we perform the update, replacing the post titled "My Blog Post" with our new version of the document:

```
> db.blog.update({title : "My Blog Post"}, post)
```

Now the document has a "comments" key. If we call find again, we can see the new key:

```
> db.blog.find()
{
  "_id" : ObjectId("5037ee4a1084eb3ffef7228"),
  "title" : "My Blog Post",
  "content" : "Here's my blog post.",
  "date" : ISODate("2012-08-24T21:12:09.982Z"),
  "comments" : [ ]
}
```

- 3.4. Delete:** Remove permanently deletes documents from the database. Called with no parameters, it removes all documents from a collection. It can also take a document specifying criteria for removal. For example, this would remove the post we just created:

```
> db.blog.remove({title : "My Blog Post"})
```



```
[Sunita@localhost bin]$ ./mongo
MongoDB shell version: 2.6.1
connecting to: test
```

```
> show databases;
VBK      0.078GB
admin    (empty)
local    0.078GB
newsletter 0.078GB
```

```
> use COEM // Created a new Database named "COEM"
switched to db COEM
```

```
> db.createCollection("TE") // Created new Collection named as "TE"
{ "ok" : 1 }
```

```
> db.TE.insert({Roll:1,Name:"ABC",Address:"Pune",Per:76})
WriteResult({ "nInserted" : 1 })
```

```
> db.TE.insert({Roll:2,Name:"PQR",Address:"Pune",Per:75})
WriteResult({ "nInserted" : 1 })
```

```
> db.TE.insert({Roll:3,Name:"LMN",Address:"Hadapsar",Per:70})
WriteResult({ "nInserted" : 1 })
```

```
> db.TE.find({})
{ "_id" : ObjectId("541963be2741c7552caef0a9"), "Roll" : 1, "Name" : "ABC", "Address" :
"Pune", "Per" : 76 }
{ "_id" : ObjectId("541963cb2741c7552caef0aa"), "Roll" : 2, "Name" : "PQR", "Address" :
"Pune", "Per" : 75 }
{ "_id" : ObjectId("541963dc2741c7552caef0ab"), "Roll" : 3, "Name" : "LMN", "Address" :
"Hadapsar", "Per" : 70 }
```

```
> db.TE.find({})
{ "_id" : ObjectId("541963be2741c7552caef0a9"), "Roll" : 1, "Name" : "ABC", "Address" :
"Pune", "Per" : 76 }
{ "_id" : ObjectId("541963cb2741c7552caef0aa"), "Roll" : 2, "Name" : "PQR", "Address" :
"Pune", "Per" : 75 }
{ "_id" : ObjectId("541963dc2741c7552caef0ab"), "Roll" : 3, "Name" : "LMN", "Address" :
"Hadapsar", "Per" : 70 }
```

```
> db.TE.update({Roll:2},{ $set: {Name:"Wagholi"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

```
> db.TE.find({})
{ "_id" : ObjectId("541963be2741c7552caef0a9"), "Roll" : 1, "Name" : "ABC", "Address" :
"Pune", "Per" : 76 }
{ "_id" : ObjectId("541963cb2741c7552caef0aa"), "Roll" : 2, "Name" : "Wagholi", "Address" :
"Pune", "Per" : 75 }
```

```
{ "_id" : ObjectId("541963dc2741c7552caef0ab"), "Roll" : 3, "Name" : "LMN", "Address" :  
"Hadapsar", "Per" : 70 }
```

```
> db.TE.remove({Roll:3})  
WriteResult({ "nRemoved" : 1 })
```

```
> db.TE.find({})  
{ "_id" : ObjectId("541963be2741c7552caef0a9"), "Roll" : 1, "Name" : "ABC", "Address" :  
"Pune", "Per" : 76 }  
{ "_id" : ObjectId("541963cb2741c7552caef0aa"), "Roll" : 2, "Name" : "Wagholi", "Address" :  
"Pune", "Per" : 75 }
```

```
> db.TE.drop()  
true
```

```
> db.TE.find({})
```

```
> show collections  
system.indexes
```

Output:

```
> show databases;  
COEM      0.078GB  
VBK       0.078GB  
admin     (empty)  
local     0.078GB  
newsletter 0.078GB
```

```
> db.dropDatabase("COEM")  
2014-09-17T16:17:18.278+0530 dropDatabase doesn't take arguments at  
src/mongo/shell/db.js:141
```

```
> db.dropDatabase()  
{ "dropped" : "COEM", "ok" : 1 }
```

```
> show databases;  
VBK       0.078GB  
admin     (empty)  
local     0.078GB  
newsletter 0.078GB
```

Conclusion: In this experiment, we have studied NoSQL and implemented CRUD operations for MongoDB

Content Beyond Syllabus Experiment

11. Cassandra Query Execution

Aim: Cassandra case study

Theory: Apache Cassandra is a highly scalable, high-performance distributed database designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Apache Cassandra is an open-source NoSQL distributed database trusted by thousands of companies for scalability and high availability without compromising performance. It is a type of NoSQL database.

A NoSQL database (sometimes called as Not Only SQL) is a database that provides a mechanism to store and retrieve data other than the tabular relations used in relational databases. These databases are schema-free, support easy replication, have simple API, are eventually consistent, and can handle huge amounts of data.

Features of Cassandra

Cassandra has become so popular because of its outstanding technical features. Given below are some of the features of Cassandra:

- Elastic scalability – Cassandra is highly scalable; it allows to add of more hardware to accommodate more customers and more data as per requirement.
- Always on architecture – Cassandra has no single point of failure and it is continuously available for business-critical applications that cannot afford a failure.
- Fast linear-scale performance – Cassandra is linearly scalable, i.e., it increases your throughput as you increase the number of nodes in the cluster. Therefore it maintains a quick response time.
- Flexible data storage – Cassandra accommodates all possible data formats including structured, semi-structured, and unstructured. It can dynamically accommodate changes to your data structures according to your need.
- Easy data distribution – Cassandra provides the flexibility to distribute data where you need it by replicating data across multiple data centers.
- Transaction support – Cassandra supports properties like Atomicity, Consistency, Isolation, and Durability (ACID).
- Fast writes – Cassandra was designed to run on cheap commodity hardware. It performs blazingly fast writes and can store hundreds of terabytes of data, without sacrificing the read efficiency.

```
C:\apache-cassandra-3.11.6\bin>cqlsh Connected to Test Cluster at 127.0.0.1:9042.
```

```
[cqlsh 5.0.1 | Cassandra 3.11.6 | CQL spec 3.4.4 | Native protocol v4]
```

```
Use HELP for help.
```

```
(6 rows)
```

```
cqlsh> CREATE KEYSPACE test
```

```
... WITH REPLICATION = { 'class' : 'NetworkTopologyStrategy', 'datacenter1' : 3 }
```

```

... .. AND DURABLE_WRITES = false;
SyntaxException: line 2:0 mismatched input '.' expecting K_WITH (CREATE KEYSPACE
test[.]...)
cqlsh> SELECT * FROM system_schema.keyspaces;

```

```

keyspace_name | durable_writes | replication
-----+-----+-----
-----
system_auth | True | {'class': 'org.apache.cassandra.locator.SimpleStrategy',
'replication_factor': '1'}
system_schema | True | {'class':
'org.apache.cassandra.locator.LocalStrategy'}
tutorialspoint | True | {'class': 'org.apache.cassandra.locator.SimpleStrategy',
'replication_factor': '3'}
system_distributed | True | {'class': 'org.apache.cassandra.locator.SimpleStrategy',
'replication_factor': '3'}
system | True | {'class':
'org.apache.cassandra.locator.LocalStrategy'}
system_traces | True | {'class': 'org.apache.cassandra.locator.SimpleStrategy',
'replication_factor': '2'}
cqlsh> USE tutorialspoint;
cqlsh:tutorialspoint> INSERT INTO emp (emp_id, emp_name, emp_city,
... emp_phone, emp_sal) VALUES(1,'ram', 'Hyderabad', 9848022338, 50000);
InvalidRequest: Error from server: code=2200 [Invalid query] message="unconfigured table
emp"

```

```

cqlsh:tutorialspoint> CREATE TABLE emp(
... emp_id int PRIMARY KEY,
... emp_name text,
... emp_city text,
... emp_sal varint,

... emp_phone varint
... );
cqlsh:tutorialspoint> INSERT INTO emp (emp_id, emp_name, emp_city,
... emp_phone, emp_sal) VALUES(1,'ram', 'Hyderabad', 9848022338, 50000);
cqlsh:tutorialspoint> INSERT INTO emp (emp_id, emp_name, emp_city,
... emp_phone, emp_sal) VALUES(2,'robin', 'Hyderabad', 9848022339, 40000);
cqlsh:tutorialspoint> INSERT INTO emp (emp_id, emp_name, emp_city,
... emp_phone, emp_sal) VALUES(3,'rahman', 'Chennai', 9848022330, 45000);
cqlsh:tutorialspoint> SELECT * FROM emp;

```

```

emp_id | emp_city | emp_name | emp_phone | emp_sal
-----+-----+-----+-----+-----
1 | Hyderabad | ram | 9848022338 | 50000
2 | Hyderabad | robin | 9848022339 | 40000
3 | Chennai | rahman | 9848022330 | 45000

```

(3 rows)

```

cqlsh:tutorialspoint> UPDATE emp SET emp_city='Delhi',emp_sal=50000

```

```
... WHERE emp_id=2;
```

```
cqlsh:tutorialspoint> select * from emp;
```

emp_id	emp_city	emp_name	emp_phone	emp_sal
1	Hyderabad	ram	9848022338	50000
2	Delhi	robin	9848022339	50000
3	Chennai	rahman	9848022330	45000

(3 rows)

```
cqlsh:tutorialspoint> select * from emp;
```

emp_id	emp_city	emp_name	emp_phone	emp_sal
1	Hyderabad	ram	9848022338	50000
2	Delhi	robin	9848022339	50000
3	Chennai	rahman	9848022330	45000

(3 rows)

```
cqlsh:tutorialspoint> SELECT emp_name, emp_sal from emp;
```

emp_name	emp_sal
ram	50000
robin	50000
rahman	45000

(3 rows)

```
cqlsh> DELETE emp_sal FROM emp WHERE emp_id=3;
```

InvalidRequest: Error from server: code=2200 [Invalid query] message="No keyspace has been specified. USE a keyspace, or explicitly specify keyspace.tablename"

```
cqlsh> USE tutorialspoint;
```

```
cqlsh:tutorialspoint> DELETE FROM emp WHERE emp_id=3;
```

```
cqlsh:tutorialspoint> select * from emp;
```

emp_id	emp_city	emp_name	emp_phone	emp_sal
1	Hyderabad	ram	9848022338	50000
2	Delhi	robin	9848022339	50000

(2 rows)

Conclusion: In this experiment, we have studied Cassandra and implemented Cassandra queries creating keyspace, insert, select, and delete operations.