PRACTICAL N0 – 1

Aim: DDL operations on Relational Schema

Design the following schema and execute the following queries on it:

salesman

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5003	Lauson Hen		0.12
5007	Paul Adam	Rome	0.13

customer

customer_id	customer_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3005	Graham Zusi	California	200	5002
3001	Brad Guzan	London		
3004	Fabian Johns	Paris	300	5006
3007	Brad Davis	New York	200	5001
3009	Geoff Camero	Berlin	100	
3008	Julian Green	London	300	5002
3003	Jozy Altidor	Moncow	200	5007

order

order_no	purch_amt	order_date	customer_id	salesman_id
70001	150.5	2016-10-05	3005	5002
70009	270.65	2016-09-10	3001	
70002	65.26	2016-10-05	3002	5001
70004	110.5	2016-08-17	3009	
70007	948.5	2016-09-10	3005	5002
70005	2400.6	2016-07-27	3007	5001
70008	5760	2016-09-10	3002	5001
70010	1983.43	2016-10-10	3004	5006
70003	2480.4	2016-10-10	3009	
70012	250.45	2016-06-27	3008	5002
70011	75.29	2016-08-17	3003	5007

Code:

```
create database salesman;

use salesman

CREATE TABLE salesman(salesman_id INT NOT NULL
AUTO_INCREMENT PRIMARY key,

name VARCHAR(100)NOT NULL,

city VARCHAR(100)NOT NULL,

commission DECIMAL(10,2)

 );
```

desc salesman;

Output:

```
mysql> create database salesman
    -> ;
Query OK, 1 row affected (0.00 sec)

mysql> use salesman
Database changed
mysql> CREATE TABLE salesman(salesman_id INT NOT NULL AUTO_INCREMENT PRIMARY key,
    -> name VARCHAR(100)NOT NULL,
    -> city VARCHAR(100)NOT NULL,
    -> commission DECIMAL(10,2)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> desc salesman
    -> ;
+-------------+---------------+------+-----+---------+----------------+
| Field       | Type          | Null | Key | Default | Extra          |
+-------------+---------------+------+-----+---------+----------------+
| salesman_id | int(11)       | NO   | PRI | NULL    | auto_increment |
| name        | varchar(100)  | NO   |     | NULL    |                |
| city        | varchar(100)  | NO   |     | NULL    |                |
| commission  | decimal(10,2) | YES  |     | NULL    |                |
+-------------+---------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
```

Code:

```
insert into salesman values(5001, 'James Hoog', 'New York', 0.15);

insert into salesman values(5002, 'Nail Knite', 'Paris', 0.13);

insert into salesman values(5005, 'Pit Alex', 'London', 0.11);

insert into salesman values(5006, 'Mc Lyon', 'Paris', 0.14);

insert into salesman values(5003, 'Lauson Hen', '', 0.12);

insert into salesman values(5007, 'Paul Adam', 'Rome', 0.13);

select * from salesman;
```

Output:

```
mysql> insert into salesman values(5001, 'James Hoog', 'New York', 0.15);
Query OK, 1 row affected (0.04 sec)

mysql> insert into salesman values(5002, 'Nail Knite', 'Paris', 0.13);
Query OK, 1 row affected (0.00 sec)

mysql> insert into salesman values(5005, 'Pit Alex', 'London', 0.11);
Query OK, 1 row affected (0.00 sec)

mysql> insert into salesman values(5006, 'Mc Lyon', 'Paris', 0.14);
Query OK, 1 row affected (0.00 sec)

mysql> insert into salesman values(5003, 'Lauson Hen', '', 0.12);
Query OK, 1 row affected (0.00 sec)

mysql> insert into salesman values(5007, 'Paul Adam', 'Rome', 0.13);
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> select * from salesman;
+-------------+------------+----------+------------+
| salesman_id | name       | city     | commission |
+-------------+------------+----------+------------+
|        5001 | James Hoog | New York |       0.15 |
|        5002 | Nail Knite | Paris    |       0.13 |
|        5003 | Lauson Hen |          |       0.12 |
|        5005 | Pit Alex   | London   |       0.11 |
|        5006 | Mc Lyon    | Paris    |       0.14 |
|        5007 | Paul Adam  | Rome     |       0.13 |
+-------------+------------+----------+------------+
6 rows in set (0.00 sec)
```

Code:

```
CREATE TABLE customer(customer_id INT NOT NULL
AUTO_INCREMENT PRIMARY key,

customer_name VARCHAR(100)NOT NULL,

city VARCHAR(100)NOT NULL,

grade INT,

salesman_id INT,

FOREIGN KEY (salesman_id) REFERENCES salesman(salesman_id)

);
```

desc customer;

Output:

```
mysql> CREATE TABLE customer(customer_id INT NOT NULL AUTO_INCREMENT PRIMARY key,
    -> customer_name VARCHAR(100)NOT NULL,
    -> city VARCHAR(100)NOT NULL,
    -> grade INT,
    -> salesman_id INT,
    -> FOREIGN KEY (salesman_id) REFERENCES salesman(salesman_id)
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql> desc customer;
+---------------+--------------+------+-----+---------+----------------+
| Field         | Type         | Null | Key | Default | Extra          |
+---------------+--------------+------+-----+---------+----------------+
| customer_id   | int(11)      | NO   | PRI | NULL    | auto_increment |
| customer_name | varchar(100) | NO   |     | NULL    |                |
| city          | varchar(100) | NO   |     | NULL    |                |
| grade         | int(11)      | YES  |     | NULL    |                |
| salesman_id   | int(11)      | YES  | MUL | NULL    |                |
+---------------+--------------+------+-----+---------+----------------+
5 rows in set (0.04 sec)
```

Code:

insert into customer values(3002, 'Nick Rimando', 'New York', 100, 5001);

insert into customer values(3005, 'Graham Zusi', 'California', 200, 5002);

insert into customer values(3001, 'Brad Guzan', 'Londan', 100, 5005);

insert into customer values(3004, 'Fabian Johns', 'Paris', 300, 5006);

insert into customer values(3007, 'Brad Davis', 'New York', 200, 5001);

insert into customer values(3009, 'Geoff Camero', 'Berlin', 100, 5003);

insert into customer values(3008, 'Julian Green', 'London', 300, 5002);

insert into customer values(3003, 'Jory Altidor', 'Moncow', 200, 5007);

select * from customer;

Output:

```
mysql> insert into customer values(3002, 'Nick Rimando', 'New York', 100, 5001);
Query OK, 1 row affected (0.01 sec)

mysql> insert into customer values(3005, 'Graham Zusi', 'California', 200, 5002);
Query OK, 1 row affected (0.03 sec)

mysql> insert into customer values(3001, 'Brad Guzan', 'Londan', 100, 5005);
Query OK, 1 row affected (0.00 sec)

mysql> insert into customer values(3004, 'Fabian Johns', 'Paris', 300, 5006);
Query OK, 1 row affected (0.00 sec)

mysql> insert into customer values(3007, 'Brad Davis', 'New York', 200, 5001);
Query OK, 1 row affected (0.01 sec)

mysql> insert into customer values(3009, 'Geoff Camero', 'Berlin', 100, 5003);
Query OK, 1 row affected (0.01 sec)

mysql> insert into customer values(3008, 'Julian Green', 'London', 300, 5002);
Query OK, 1 row affected (0.03 sec)

mysql> insert into customer values(3003, 'Jory Altidor', 'Moncow', 200, 5007);
Query OK, 1 row affected (0.01 sec)

mysql> select * from customer
    -> ;
+-------------+---------------+------------+-------+-------------+
| customer_id | customer_name | city       | grade | salesman_id |
+-------------+---------------+------------+-------+-------------+
|        3001 | Brad Guzan    | Londan     |   100 |        5005 |
|        3002 | Nick Rimando  | New York   |   100 |        5001 |
|        3003 | Jory Altidor  | Moncow     |   200 |        5007 |
|        3004 | Fabian Johns  | Paris      |   300 |        5006 |
|        3005 | Graham Zusi   | California |   200 |        5002 |
|        3007 | Brad Davis    | New York   |   200 |        5001 |
|        3008 | Julian Green  | London     |   300 |        5002 |
|        3009 | Geoff Camero  | Berlin     |   100 |        5003 |
+-------------+---------------+------------+-------+-------------+
8 rows in set (0.00 sec)
```

Code:

CREATE TABLE orders(order_no INT NOT NULL AUTO_INCREMENT PRIMARY key,

purch_amt DECIMAL(10,2) NOT NULL,

order_date DATE NOT NULL,

customer_id INT,

salesman_id INT,

FOREIGN KEY (customer_id) REFERENCES customer(customer_id),

FOREIGN KEY (salesman_id) REFERENCES salesman(salesman_id)

);

desc orders;

Output:

```
mysql> CREATE TABLE orders(order_no INT NOT NULL AUTO_INCREMENT PRIMARY key,
    -> purch_amt DECIMAL(10,2) NOT NULL,
    -> order_date DATE NOT NULL,
    -> customer_id INT,
    -> salesman_id INT,
    -> FOREIGN KEY (customer_id) REFERENCES customer(customer_id),
    -> FOREIGN KEY (salesman_id) REFERENCES salesman(salesman_id)
    ->  );
Query OK, 0 rows affected (0.02 sec)

mysql> desc orders;
+-------------+---------------+------+-----+---------+----------------+
| Field       | Type          | Null | Key | Default | Extra          |
+-------------+---------------+------+-----+---------+----------------+
| order_no    | int(11)       | NO   | PRI | NULL    | auto_increment |
| purch_amt   | decimal(10,2) | NO   |     | NULL    |                |
| order_date  | date          | NO   |     | NULL    |                |
| customer_id | int(11)       | YES  | MUL | NULL    |                |
| salesman_id | int(11)       | YES  | MUL | NULL    |                |
+-------------+---------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)
```

Code:

insert into orders values(70001, 150.5, '2016-10-05', 3005, 5002);

insert into orders values(70009, 270.65, '2016-09-10', 3001, NULL);

insert into orders values(70002, 65.26, '2016-10-05', 3002, 5001);

insert into orders values(70004, 110.5, '2016-08-17', 3009, NULL);

insert into orders values(70007, 948.5, '2016-09-10', 3005, 5002);

insert into orders values(70005, 2400.6, '2016-07-27', 3007, 5001);

insert into orders values(70008, 5760, '2016-09-10', 3002, 5001);

insert into orders values(70010, 1983.43, '2016-10-10', 3004, NULL);

insert into orders values(70003, 2480.4, '2016-10-10', 3009, 5006);

insert into orders values(70012, 250.45, '2016-06-27', 3008, 5002);

insert into orders values(70011, 75.29, '2016-08-17', 3003, 5007);

select * from orders;

Output:

```
mysql> insert into orders values(70001, 150.5, '2016-10-05', 3005, 5002);
Query OK, 1 row affected (0.01 sec)

mysql> insert into orders values(70009, 270.65, '2016-09-10', 3001, NULL);
Query OK, 1 row affected (0.02 sec)

mysql> insert into orders values(70002, 65.26, '2016-10-05', 3002, 5001);
Query OK, 1 row affected (0.01 sec)

mysql> insert into orders values(70004, 110.5, '2016-08-17', 3009, NULL);
Query OK, 1 row affected (0.01 sec)

mysql> insert into orders values(70007, 948.5, '2016-09-10', 3005, 5002);
Query OK, 1 row affected (0.02 sec)

mysql> insert into orders values(70005, 2400.6, '2016-07-27', 3007, 5001);
Query OK, 1 row affected (0.02 sec)

mysql> insert into orders values(70008, 5760, '2016-09-10', 3002, 5001);
Query OK, 1 row affected (0.02 sec)

mysql> insert into orders values(70010, 1983.43, '2016-10-10', 3004, NULL);
Query OK, 1 row affected (0.01 sec)

mysql> insert into orders values(70003, 2480.4, '2016-10-10', 3009, 5006);
Query OK, 1 row affected (0.01 sec)

mysql> insert into orders values(70012, 250.45, '2016-06-27', 3008, 5002);
Query OK, 1 row affected (0.01 sec)
```

```
mysql> insert into orders values(70011, 75.29, '2016-08-17', 3003, 5007);
Query OK, 1 row affected (0.01 sec)

mysql> select * from orders;
+----------+-----------+------------+-------------+-------------+
| order_no | purch_amt | order_date | customer_id | salesman_id |
+----------+-----------+------------+-------------+-------------+
|    70001 |    150.50 | 2016-10-05 |        3005 |        5002 |
|    70002 |     65.26 | 2016-10-05 |        3002 |        5001 |
|    70003 |   2480.40 | 2016-10-10 |        3009 |        5006 |
|    70004 |    110.50 | 2016-08-17 |        3009 |        NULL |
|    70005 |   2400.60 | 2016-07-27 |        3007 |        5001 |
|    70007 |    948.50 | 2016-09-10 |        3005 |        5002 |
|    70008 |   5760.00 | 2016-09-10 |        3002 |        5001 |
|    70009 |    270.65 | 2016-09-10 |        3001 |        NULL |
|    70010 |   1983.43 | 2016-10-10 |        3004 |        NULL |
|    70011 |     75.29 | 2016-08-17 |        3003 |        5007 |
|    70012 |    250.45 | 2016-06-27 |        3008 |        5002 |
+----------+-----------+------------+-------------+-------------+
11 rows in set (0.00 sec)
```

1. Display name and commission for all the salesmen.

Code:

```
select name, commission FROM salesman;
```

Output:

```
mysql> select name, commission FROM salesman;
+------------+------------+
| name       | commission |
+------------+------------+
| James Hoog |       0.15 |
| Nail Knite |       0.13 |
| Lauson Hen |       0.12 |
| Pit Alex   |       0.11 |
| Mc Lyon    |       0.14 |
| Paul Adam  |       0.13 |
+------------+------------+
6 rows in set (0.00 sec)
```

2. Retrieve salesman id of all salesmen from orders table without any repeats.

Code:

```
select DISTINCT salesman_id FROM orders;
```

Output:

```
mysql> select DISTINCT salesman_id FROM orders;
+-------------+
| salesman_id |
+-------------+
|        NULL |
|        5001 |
|        5002 |
|        5006 |
|        5007 |
+-------------+
5 rows in set (0.00 sec)
```

3. Display names and city of salesman, who belongs to the city of Paris.

Code:

```
select name, city FROM salesman WHERE city = 'Paris';
```

Output:

```
mysql> select name, city FROM salesman WHERE city = 'Paris';
+------------+-------+
| name       | city  |
+------------+-------+
| Nail Knite | Paris |
| Mc Lyon    | Paris |
+------------+-------+
2 rows in set (0.00 sec)
```

4. Display all the information for those customers with a grade of 200.

Code:

```
select * FROM customer WHERE grade = 200;
```

Output:

```
mysql> select * FROM customer WHERE grade = 200;
+-------------+---------------+------------+-------+-------------+
| customer_id | customer_name | city       | grade | salesman_id |
+-------------+---------------+------------+-------+-------------+
|        3003 | Jory Altidor  | Moncow     |   200 |        5007 |
|        3005 | Graham Zusi   | California |   200 |        5002 |
|        3007 | Brad Davis    | New York   |   200 |        5001 |
+-------------+---------------+------------+-------+-------------+
3 rows in set (0.00 sec)
```

5. Display the order number, order date and the purchase amount for order(s) which will be delivered by the salesman with ID 5001

Code:

```
select order_no,order_date,purch_amt FROM orders WHERE
salesman_id = 5001;
```

Output:

```
mysql> select order_no,order_date,purch_amt FROM orders WHERE salesman_id = 5001;
+----------+------------+-----------+
| order_no | order_date | purch_amt |
+----------+------------+-----------+
|    70002 | 2016-10-05 |     65.26 |
|    70005 | 2016-07-27 |   2400.60 |
|    70008 | 2016-09-10 |   5760.00 |
+----------+------------+-----------+
3 rows in set (0.00 sec)
```

12. Display all the customers, who are either belongs to the city New York or not had a grade above 100.

Code:

```
select  * FROM customer WHERE city ='New York' OR grade<=100;
```

Output:

```
mysql> select  * FROM customer WHERE city ='New York' OR grade<=100;
+-------------+---------------+----------+-------+-------------+
| customer_id | customer_name | city     | grade | salesman_id |
+-------------+---------------+----------+-------+-------------+
|        3001 | Brad Guzan    | Londan   |   100 |        5005 |
|        3002 | Nick Rimando  | New York |   100 |        5001 |
|        3007 | Brad Davis    | New York |   200 |        5001 |
|        3009 | Geoff Camero  | Berlin   |   100 |        5003 |
+-------------+---------------+----------+-------+-------------+
4 rows in set (0.00 sec)
```

13. Find those salesmen with all information who gets the commission within a range of 0.12 and 0.14.

Code:

```
select * FROM salesman WHERE commission BETWEEN 0.12 AND 0.14;
```

Output:

```
mysql> select * FROM salesman WHERE commission BETWEEN 0.12 AND 0.14;
+-------------+------------+-------+------------+
| salesman_id | name       | city  | commission |
+-------------+------------+-------+------------+
|        5002 | Nail Knite | Paris |       0.13 |
|        5003 | Lauson Hen |       |       0.12 |
|        5006 | Mc Lyon    | Paris |       0.14 |
|        5007 | Paul Adam  | Rome  |       0.13 |
+-------------+------------+-------+------------+
4 rows in set (0.00 sec)
```

14. Find all those customers with all information whose names are ending with the letter 'n'.

Code:

```
select * FROM customer WHERE customer_name LIKE '%n';
```

Output:

```
mysql> select * FROM customer WHERE customer_name LIKE '%n';
+-------------+---------------+--------+-------+-------------+
| customer_id | customer_name | city   | grade | salesman_id |
+-------------+---------------+--------+-------+-------------+
|        3001 | Brad Guzan    | Londan |   100 |        5005 |
|        3008 | Julian Green  | London |   300 |        5002 |
+-------------+---------------+--------+-------+-------------+
2 rows in set (0.00 sec)
```

15. Find those salesmen with all information whose name containing the 1st character is 'N' and the 4th character is 'l' and rests may be any character.

Code:

```
select * FROM salesman WHERE name LIKE 'N_I%';
```

Output:

```
mysql> select * FROM salesman WHERE name LIKE 'N_I%';
+-------------+------------+-------+------------+
| salesman_id | name       | city  | commission |
+-------------+------------+-------+------------+
|        5002 | Nail Knite | Paris |       0.13 |
+-------------+------------+-------+------------+
1 row in set (0.00 sec)
```

16. Find that customer with all information who does not get any grade except NULL.

Code:

```
select * FROM customer WHERE grade is NULL;
```

Output:

```
mysql> select * FROM customer WHERE grade is NULL;
Empty set (0.00 sec)
```

17. Find the total purchase amount of all orders.

Code:

```
select SUM(purch_amt) AS total_purchase FROM orders;
```

Output:

```
mysql> select SUM(purch_amt) AS total_purchase FROM orders;
+----------------+
| total_purchase |
+----------------+
|       14495.58 |
+----------------+
1 row in set (0.02 sec)
```

18. Find the number of salesman currently listing for all of their customers.

Code:

```
select salesman_id, COUNT(customer_id) AS total_customers FROM customer GROUP BY salesman_id;
```

Output:

```
mysql> select salesman_id, COUNT(customer_id) AS total_customers FROM customer GROUP BY salesman_id;
+------------+-----------------+
| salesman_id | total_customers |
+------------+-----------------+
|       5001 |               2 |
|       5002 |               2 |
|       5003 |               1 |
|       5005 |               1 |
|       5006 |               1 |
|       5007 |               1 |
+------------+-----------------+
6 rows in set (0.00 sec)
```

19. Find the highest grade for each of the cities of the customers.

Code:

```
select city, Max(grade) As highest_grade FROM customer GROUP BY city;
```

Output:

```
mysql> select city, Max(grade) As highest_grade FROM customer GROUP BY city;
+------------+---------------+
| city       | highest_grade |
+------------+---------------+
| Berlin     |           100 |
| California |           200 |
| Londan     |           100 |
| London     |           300 |
| Moncow     |           200 |
| New York   |           200 |
| Paris      |           300 |
+------------+---------------+
7 rows in set (0.00 sec)
```

20. Find the highest purchase amount ordered by each customer with their ID and highest purchase amount.

Code:

```
select customer_id, Max(purch_amt) AS highest_purchase FROM orders GROUP BY customer_id;
```

Output:

```
mysql> select customer_id, Max(purch_amt) AS highest_purchase FROM orders GROUP BY customer_id;
+-------------+------------------+
| customer_id | highest_purchase |
+-------------+------------------+
|        3001 |           270.65 |
|        3002 |          5760.00 |
|        3003 |            75.29 |
|        3004 |          1983.43 |
|        3005 |           948.50 |
|        3007 |          2400.60 |
|        3008 |           250.45 |
|        3009 |          2480.40 |
+-------------+------------------+
8 rows in set (0.00 sec)
```

21. Find the highest purchase amount ordered by each customer on a particular date with their ID, order date and highest purchase amount.

Code:

```
select customer_id, order_date, Max(purch_amt) AS highest_purchase FROM orders GROUP BY customer_id, order_date;
```

Output:

```
mysql> select customer_id, order_date, Max(purch_amt) AS highest_purchase FROM orders GROUP BY customer_id, order_date;
+-------------+------------+------------------+
| customer_id | order_date | highest_purchase |
+-------------+------------+------------------+
|        3001 | 2016-09-10 |           270.65 |
|        3002 | 2016-09-10 |          5760.00 |
|        3002 | 2016-10-05 |            65.26 |
|        3003 | 2016-08-17 |            75.29 |
|        3004 | 2016-10-10 |          1983.43 |
|        3005 | 2016-09-10 |           948.50 |
|        3005 | 2016-10-05 |           150.50 |
|        3007 | 2016-07-27 |          2400.60 |
|        3008 | 2016-06-27 |           250.45 |
|        3009 | 2016-08-17 |           110.50 |
|        3009 | 2016-10-10 |          2480.40 |
+-------------+------------+------------------+
11 rows in set (0.00 sec)
```

22. Find the highest purchase amount on a date '2012-08-17' for each salesman with their ID.

Code:

```
select salesman_id, MAX(purch_amt) AS highest_purchase FROM orders WHERE order_date = 2012-08-17 GROUP BY salesman_id;
```

Output:

```
mysql> select salesman_id, MAX(purch_amt) AS highest_purchase FROM orders WHERE order_date = 2012-08-17 GROUP BY salesman_id;
Empty set, 1 warning (0.00 sec)
```

23. Find the highest purchase amount with their customer ID and order date, for only those customers who have the highest purchase amount in a day is more than 2000.

Code:

```
select customer_id, order_date, MAX(purch_amt) AS highest_purchase FROM orders GROUP BY customer_id, order_date HAVING MAX(purch_amt)>2000;
```

Output:

```
mysql> select customer_id, order_date, MAX(purch_amt) AS highest_purchase FROM orders GROUP BY customer_id, order_date HAVING MAX(purch_amt)>2000;
+-------------+------------+------------------+
| customer_id | order_date | highest_purchase |
+-------------+------------+------------------+
|        3002 | 2016-09-10 |          5760.00 |
|        3007 | 2016-07-27 |          2400.60 |
|        3009 | 2016-10-10 |          2480.40 |
+-------------+------------+------------------+
3 rows in set (0.00 sec)
```

24. Write a SQL statement that counts all orders for a date August 17th, 2012.

Code:

```
select COUNT(*) AS total_orders FROM orders WHERE order_date = 2012-08-17;
```

Output:

```
mysql> select COUNT(*) AS total_orders FROM orders WHERE order_date = 2012-08-17;
+--------------+
| total_orders |
+--------------+
|            0 |
+--------------+
1 row in set, 1 warning (0.00 sec)
```

PRACTICAL N0 – 2

Aim: Subquery-join operations on Relational Schema

USING (practical 1)

1. **Count the customers with grades above Bangalore's average.**

Code:

```
SELECT COUNT(*)
FROM Customer
WHERE grade > (
SELECT AVG(grade)
 FROM Customer
 WHERE city = 'Bangalores'
);
```

Output:

```
mysql> SELECT COUNT(*)
    -> FROM Customer
    -> WHERE grade > (
    -> SELECT AVG(grade)
    ->  FROM Customer
    ->  WHERE city = 'Bangalores'
    -> );
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
1 row in set (0.00 sec)
```

2. **Find the name and numbers of all salesmen who had more than one customer.**

Code:

```
SELECT S.name, S.salesman_id

FROM Salesman S

JOIN Customer C ON S.salesman_id = C.salesman_id

GROUP BY S.salesman_id, S.name

HAVING COUNT(C.customer_id) > 1;
```

Output:

```
mysql> SELECT S.name, S.salesman_id
    -> FROM Salesman S
    -> JOIN Customer C ON S.salesman_id = C.salesman_id
    -> GROUP BY S.salesman_id, S.name
    -> HAVING COUNT(C.customer_id) > 1;
+------------+-------------+
| name       | salesman_id |
+------------+-------------+
| James Hoog |        5001 |
| Nail Knite |        5002 |
+------------+-------------+
2 rows in set (0.02 sec)
```

3. **List all salesmen and indicate those who have and don't have customers in their cities**

   (Use UNION operation.)

Code:

```
SELECT S.salesman_id, S.name, 'Has Customers' AS customer_status

FROM Salesman S

JOIN Customer C ON S.salesman_id = C.salesman_id

WHERE S.city = C.city

UNION

SELECT S.salesman_id, S.name, 'No Customers' AS customer_status

FROM Salesman S

LEFT JOIN Customer C ON S.salesman_id = C.salesman_id AND S.city = C.city

WHERE C.customer_id IS NULL;
```

Output:

```
mysql> SELECT S.salesman_id, S.name, 'Has Customers' AS customer_status
    -> FROM Salesman S
    -> JOIN Customer C ON S.salesman_id = C.salesman_id
    -> WHERE S.city = C.city
    -> UNION
    -> SELECT S.salesman_id, S.name, 'No Customers' AS customer_status
    -> FROM Salesman S
    -> LEFT JOIN Customer C ON S.salesman_id = C.salesman_id AND S.city = C.city
    -> WHERE C.customer_id IS NULL;
+-------------+------------+-----------------+
| salesman_id | name       | customer_status |
+-------------+------------+-----------------+
|        5001 | James Hoog | Has Customers   |
|        5006 | Mc Lyon    | Has Customers   |
|        5002 | Nail Knite | No Customers    |
|        5003 | Lauson Hen | No Customers    |
|        5005 | Pit Alex   | No Customers    |
|        5007 | Paul Adam  | No Customers    |
+-------------+------------+-----------------+
6 rows in set (0.00 sec)
```

4. **Create a view that finds the salesman who has the customer with the highest order of a day.**

Code:

```
CREATE VIEW SalesmanWithHighestOrder AS

SELECT S.salesman_id, S.name, O.order_date, MAX(O.purch_amt) AS max_order_amount

FROM Salesman S

JOIN Customer C ON S.salesman_id = C.salesman_id

JOIN `orders` O ON C.customer_id = O.customer_id

GROUP BY S.salesman_id, S.name, O.order_date;

select * from SalesmanWithHighestOrder;
```

Output:

```
mysql> CREATE VIEW SalesmanWithHighestOrder AS
    -> SELECT S.salesman_id, S.name, O.order_date, MAX(O.purch_amt) AS max_order_amount
    -> FROM Salesman S
    -> JOIN Customer C ON S.salesman_id = C.salesman_id
    -> JOIN `orders` O ON C.customer_id = O.customer_id
    -> GROUP BY S.salesman_id, S.name, O.order_date;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from SalesmanWithHighestOrder;
+-------------+------------+------------+------------------+
| salesman_id | name       | order_date | max_order_amount |
+-------------+------------+------------+------------------+
|        5001 | James Hoog | 2016-07-27 |          2400.60 |
|        5001 | James Hoog | 2016-09-10 |          5760.00 |
|        5001 | James Hoog | 2016-10-05 |            65.26 |
|        5002 | Nail Knite | 2016-06-27 |           250.45 |
|        5002 | Nail Knite | 2016-09-10 |           948.50 |
|        5002 | Nail Knite | 2016-10-05 |           150.50 |
|        5003 | Lauson Hen | 2016-08-17 |           110.50 |
|        5003 | Lauson Hen | 2016-10-10 |          2480.40 |
|        5005 | Pit Alex   | 2016-09-10 |           270.65 |
|        5006 | Mc Lyon    | 2016-10-10 |          1983.43 |
|        5007 | Paul Adam  | 2016-08-17 |            75.29 |
+-------------+------------+------------+------------------+
11 rows in set (0.03 sec)
```

5. **Demonstrate the DELETE operation by removing salesman with id 1000. All his orders must also be deleted**

Code:

```
DELETE FROM salesman WHERE salesman_id = 1000;

SELECT * FROM Salesman;

SELECT * FROM Orders;
```

Output:

```
mysql> DELETE FROM salesman WHERE salesman_id = 1000;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT * FROM Salesman;
+-------------+------------+----------+------------+
| salesman_id | name       | city     | commission |
+-------------+------------+----------+------------+
|        5001 | James Hoog | New York |       0.15 |
|        5002 | Nail Knite | Paris    |       0.13 |
|        5003 | Lauson Hen |          |       0.12 |
|        5005 | Pit Alex   | London   |       0.11 |
|        5006 | Mc Lyon    | Paris    |       0.14 |
|        5007 | Paul Adam  | Rome     |       0.13 |
+-------------+------------+----------+------------+
6 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Orders;
+----------+-----------+------------+-------------+-------------+
| order_no | purch_amt | order_date | customer_id | salesman_id |
+----------+-----------+------------+-------------+-------------+
|    70001 |    150.50 | 2016-10-05 |        3005 |        5002 |
|    70002 |     65.26 | 2016-10-05 |        3002 |        5001 |
|    70003 |   2480.40 | 2016-10-10 |        3009 |        5006 |
|    70004 |    110.50 | 2016-08-17 |        3009 |        NULL |
|    70005 |   2400.60 | 2016-07-27 |        3007 |        5001 |
|    70007 |    948.50 | 2016-09-10 |        3005 |        5002 |
|    70008 |   5760.00 | 2016-09-10 |        3002 |        5001 |
|    70009 |    270.65 | 2016-09-10 |        3001 |        NULL |
|    70010 |   1983.43 | 2016-10-10 |        3004 |        NULL |
|    70011 |     75.29 | 2016-08-17 |        3003 |        5007 |
|    70012 |    250.45 | 2016-06-27 |        3008 |        5002 |
+----------+-----------+------------+-------------+-------------+
11 rows in set (0.00 sec)
```

2. Design ERD for the following schema and execute the following Queries on it:

Consider the schema for Movie Database:

ACTOR (Act_id, Act_Name, Act_Gender)

DIRECTOR (Dir_id, Dir_Name, Dir_Phone)

MOVIES (Mov_id, Mov_Title, Mov_Year, Mov_Lang, Dir_id)

MOVIE_CAST (Act_id, Mov_id, Role)

RATING (Mov_id, Rev_Stars)

Code:

```
CREATE TABLE ACTOR (
ACT_ID INT (3),
ACT_NAME VARCHAR (20),
ACT_GENDER CHAR (1),
PRIMARY KEY (ACT_ID));
```

Output:

```
mysql> CREATE TABLE ACTOR (
    ->     ACT_ID INT(3),
    ->     ACT_NAME VARCHAR(20),
    ->     ACT_GENDER CHAR(1),
    ->     PRIMARY KEY (ACT_ID)
    -> );
Query OK, 0 rows affected (0.02 sec)
```

Code:

```
CREATE TABLE DIRECTOR (

DIR_ID INT (3),

DIR_NAME VARCHAR (20),

DIR_PHONE INT (10),

PRIMARY KEY (DIR_ID));
```

Output:

```
mysql> CREATE TABLE DIRECTOR (
    -> DIR_ID INT (3),
    -> DIR_NAME VARCHAR (20),
    -> DIR_PHONE INT (10),
    -> PRIMARY KEY (DIR_ID));
Query OK, 0 rows affected (0.01 sec)
```

Code:

```
CREATE TABLE MOVIES (

MOV_ID INT (4),

MOV_TITLE VARCHAR (25),

MOV_YEAR INT (4),

MOV_LANG VARCHAR (12),

DIR_ID INT (3),

PRIMARY KEY (MOV_ID),

FOREIGN KEY (DIR_ID) REFERENCES DIRECTOR (DIR_ID));
```

Output:

```
mysql> CREATE TABLE MOVIES (
    -> MOV_ID INT (4),
    -> MOV_TITLE VARCHAR (25),
    -> MOV_YEAR INT (4),
    -> MOV_LANG VARCHAR (12),
    -> DIR_ID INT (3),
    -> PRIMARY KEY (MOV_ID),
    -> FOREIGN KEY (DIR_ID) REFERENCES DIRECTOR (DIR_ID));
Query OK, 0 rows affected (0.05 sec)
```

Code:

```
CREATE TABLE MOVIE_CAST (

ACT_ID INT (3),

MOV_ID INT (4),

OLE VARCHAR (10),

PRIMARY KEY (ACT_ID, MOV_ID),

FOREIGN KEY (ACT_ID) REFERENCES ACTOR (ACT_ID),

FOREIGN KEY (MOV_ID) REFERENCES MOVIES (MOV_ID));
```

Output:

```
mysql> CREATE TABLE MOVIE_CAST (
    -> ACT_ID INT (3),
    -> MOV_ID INT (4),
    ->
    -> OLE VARCHAR (10),
    -> PRIMARY KEY (ACT_ID, MOV_ID),
    -> FOREIGN KEY (ACT_ID) REFERENCES ACTOR (ACT_ID),
    -> FOREIGN KEY (MOV_ID) REFERENCES MOVIES (MOV_ID));
Query OK, 0 rows affected (0.01 sec)
```

Code:

```
CREATE TABLE RATING (

MOV_ID INT (4),

REV_STARS VARCHAR (25),

PRIMARY KEY (MOV_ID),

FOREIGN KEY (MOV_ID) REFERENCES MOVIES (MOV_ID));
```

Output:

```
mysql> CREATE TABLE RATING (
    -> MOV_ID INT (4),
    -> REV_STARS VARCHAR (25),
    -> PRIMARY KEY (MOV_ID),
    -> FOREIGN KEY (MOV_ID) REFERENCES MOVIES (MOV_ID));
Query OK, 0 rows affected (0.01 sec)
```

Code:

```
INSERT INTO ACTOR VALUES (301,'ANUSHKA','F');

INSERT INTO ACTOR VALUES (302,'PRABHAS','M');

INSERT INTO ACTOR VALUES (303,'PUNITH','M');

INSERT INTO ACTOR VALUES (304,'JERMY','M');
```

Output:

```
mysql> INSERT INTO ACTOR VALUES (301,'ANUSHKA','F');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO ACTOR VALUES (302,'PRABHAS','M');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO ACTOR VALUES (303,'PUNITH','M');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO ACTOR VALUES (304,'JERMY','M');
Query OK, 1 row affected (0.00 sec)
```

Code:

```
INSERT INTO DIRECTOR VALUES (60,'RAJAMOULI', 875161100);

INSERT INTO DIRECTOR VALUES (61,'HITCHCOCK', 776613891);

INSERT INTO DIRECTOR VALUES (62,'FARAN', 998677653);

INSERT INTO DIRECTOR VALUES (63,'STEVEN SPIELBERG',
898977653);
```

Output:

```
mysql> INSERT INTO DIRECTOR VALUES (60,'RAJAMOULI', 875161100);
Query OK, 1 row affected (0.03 sec)

mysql> INSERT INTO DIRECTOR VALUES (61,'HITCHCOCK', 776613891);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO DIRECTOR VALUES (62,'FARAN', 998677653);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO DIRECTOR VALUES (63,'STEVEN SPIELBERG', 898977653);
Query OK, 1 row affected (0.00 sec)
```

Code:

```
INSERT INTO MOVIES VALUES (1001,'BAHUBALI-2', 2017, 'TELAGU', 60);

INSERT INTO MOVIES VALUES (1002,'BAHUBALI-1', 2015, 'TELAGU', 60);

INSERT INTO MOVIES VALUES (1003,'AKASH', 2008, 'KANNADA', 61);

INSERT INTO MOVIES VALUES (1004,'WAR HORSE', 2011, 'ENGLISH', 63);
```

Output:

```
mysql> INSERT INTO MOVIES VALUES (1001,'BAHUBALI-2', 2017, 'TELAGU', 60);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO MOVIES VALUES (1002,'BAHUBALI-1', 2015, 'TELAGU', 60);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO MOVIES VALUES (1003,'AKASH', 2008, 'KANNADA', 61);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO MOVIES VALUES (1004,'WAR HORSE', 2011, 'ENGLISH', 63);
Query OK, 1 row affected (0.01 sec)
```

Code:

```
INSERT INTO MOVIE_CAST VALUES (301, 1002, 'HEROINE');

INSERT INTO MOVIE_CAST VALUES (301, 1001, 'HEROINE');

INSERT INTO MOVIE_CAST VALUES (303, 1003, 'HERO');

INSERT INTO MOVIE_CAST VALUES (303, 1002, 'GUEST');

INSERT INTO MOVIE_CAST VALUES (304, 1004, 'HERO');
```

Output:

```
mysql> INSERT INTO MOVIE_CAST VALUES (301, 1002, 'HEROINE');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO MOVIE_CAST VALUES (301, 1001, 'HEROINE');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO MOVIE_CAST VALUES (303, 1003, 'HERO');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO MOVIE_CAST VALUES (303, 1002, 'GUEST');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO MOVIE_CAST VALUES (304, 1004, 'HERO');
Query OK, 1 row affected (0.02 sec)
```

Code:

```
INSERT INTO RATING VALUES (1001, 4);

INSERT INTO RATING VALUES (1002, 2);

INSERT INTO RATING VALUES (1003, 5);

INSERT INTO RATING VALUES (1004, 4);
```

Output:

```
mysql> INSERT INTO RATING VALUES (1001, 4);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO RATING VALUES (1002, 2);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO RATING VALUES (1003, 5);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO RATING VALUES (1004, 4);
Query OK, 1 row affected (0.01 sec)
```

Write SQL queries to

1. **List the titles of all movies directed by ‘Hitchcock’.**

Code:

```
SELECT MOV_TITLE

FROM MOVIES m

JOIN DIRECTOR d ON m.DIR_ID = d.DIR_ID

WHERE d.DIR_NAME = 'HITCHCOCK';
```

Output:

```
mysql> SELECT MOV_TITLE
    -> FROM MOVIES m
    -> JOIN DIRECTOR d ON m.DIR_ID = d.DIR_ID
    -> WHERE d.DIR_NAME = 'HITCHCOCK';
+-----------+
| MOV_TITLE |
+-----------+
| AKASH     |
+-----------+
1 row in set (0.00 sec)
```

2. **Find the movie names where one or more actors acted in two or more movies.**

Code:

```
SELECT DISTINCT m.MOV_TITLE
FROM MOVIES m
JOIN MOVIE_CAST mc ON m.MOV_ID = mc.MOV_ID
WHERE mc.ACT_ID IN (
   SELECT ACT_ID
   FROM MOVIE_CAST
   GROUP BY ACT_ID
   HAVING COUNT(DISTINCT MOV_ID) >= 2
);
```

Output:

```
mysql> SELECT DISTINCT m.MOV_TITLE
    -> FROM MOVIES m
    -> JOIN MOVIE_CAST mc ON m.MOV_ID = mc.MOV_ID
    -> WHERE mc.ACT_ID IN (
    ->     SELECT ACT_ID
    ->     FROM MOVIE_CAST
    ->     GROUP BY ACT_ID
    ->     HAVING COUNT(DISTINCT MOV_ID) >= 2
    -> );
+------------+
| MOV_TITLE  |
+------------+
| BAHUBALI-2 |
| BAHUBALI-1 |
| AKASH      |
+------------+
3 rows in set (0.03 sec)
```

3. **List all actors who acted in a movie before 2000 and also in a movie after 2015 (use JOIN operation).**

Code:

```
SELECT DISTINCT a.ACT_NAME

FROM ACTOR a

JOIN MOVIE_CAST mc1 ON a.ACT_ID = mc1.ACT_ID

JOIN MOVIES m1 ON mc1.MOV_ID = m1.MOV_ID

JOIN MOVIE_CAST mc2 ON a.ACT_ID = mc2.ACT_ID

JOIN MOVIES m2 ON mc2.MOV_ID = m2.MOV_ID

WHERE m1.MOV_YEAR < 2000 AND m2.MOV_YEAR > 2015;
```

Output:

```
mysql> SELECT DISTINCT a.ACT_NAME
    -> FROM ACTOR a
    -> JOIN MOVIE_CAST mc1 ON a.ACT_ID = mc1.ACT_ID
    -> JOIN MOVIES m1 ON mc1.MOV_ID = m1.MOV_ID
    -> JOIN MOVIE_CAST mc2 ON a.ACT_ID = mc2.ACT_ID
    -> JOIN MOVIES m2 ON mc2.MOV_ID = m2.MOV_ID
    -> WHERE m1.MOV_YEAR < 2000 AND m2.MOV_YEAR > 2015;
Empty set (0.00 sec)
```

4. **Find the title of movies and number of stars for each movie that has at least one rating and find the highest number of stars that movie received. Sort the result by movie title.**

Code:

```
SELECT m.MOV_TITLE, r.REV_STARS, (
   SELECT MAX(r1.REV_STARS)
   FROM RATING r1
   WHERE r1.MOV_ID = m.MOV_ID
) AS MAX_STARS
FROM MOVIES m
JOIN RATING r ON m.MOV_ID = r.MOV_ID
ORDER BY m.MOV_TITLE;
```

Output:

```
mysql> SELECT m.MOV_TITLE, r.REV_STARS, (
    ->     SELECT MAX(r1.REV_STARS)
    ->     FROM RATING r1
    ->     WHERE r1.MOV_ID = m.MOV_ID
    -> ) AS MAX_STARS
    -> FROM MOVIES m
    -> JOIN RATING r ON m.MOV_ID = r.MOV_ID
    -> ORDER BY m.MOV_TITLE;
+------------+-----------+-----------+
| MOV_TITLE  | REV_STARS | MAX_STARS |
+------------+-----------+-----------+
| AKASH      | 5         | 5         |
| BAHUBALI-1 | 2         | 2         |
| BAHUBALI-2 | 4         | 4         |
| WAR HORSE  | 5         | 5         |
+------------+-----------+-----------+
4 rows in set (0.00 sec)
```

5. Update rating of all movies directed by ‘Steven Spielberg’ to 5.

Code:

```
UPDATE RATING
SET REV_STARS = '5'
WHERE MOV_ID IN (
   SELECT m.MOV_ID
   FROM MOVIES m
   JOIN DIRECTOR d ON m.DIR_ID = d.DIR_ID
   WHERE d.DIR_NAME = 'STEVEN SPIELBERG'
);
```

Output:

```
mysql> UPDATE RATING
    -> SET REV_STARS = '5'
    -> WHERE MOV_ID IN (
    ->     SELECT m.MOV_ID
    ->     FROM MOVIES m
    ->     JOIN DIRECTOR d ON m.DIR_ID = d.DIR_ID
    ->     WHERE d.DIR_NAME = 'STEVEN SPIELBERG'
    -> );
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

3. Design ERD for the following schema and execute the following Queries on it:

Code:

```
CREATE TABLE students (
   stno INT PRIMARY KEY,
   name VARCHAR(50),
   addr VARCHAR(255),
   city VARCHAR(50),
   state VARCHAR(2),
   zip VARCHAR(10)
);

CREATE TABLE INSTRUCTORS (
   empno INT PRIMARY KEY,
   name VARCHAR(50),
   rank VARCHAR(20),
   roomno VARCHAR(10),
   telno VARCHAR(15)
);

CREATE TABLE COURSES (
   cno INT PRIMARY KEY,
```

```
    cname VARCHAR(50),
    cr INT,
    cap INT
);
CREATE TABLE GRADES (
    stno INT,
    empno INT,
    cno INT,
    sem VARCHAR(10),
    year INT,
    grade INT,
    PRIMARY KEY (stno),
    FOREIGN KEY (stno) REFERENCES students(stno),
    FOREIGN KEY (empno) REFERENCES INSTRUCTORS(empno),
    FOREIGN KEY (cno) REFERENCES COURSES(cno)
);

CREATE TABLE ADVISING (
    stno INT,
    empno INT,
    PRIMARY KEY (stno, empno),
```

FOREIGN KEY (stno) REFERENCES students(stno),

FOREIGN KEY (empno) REFERENCES INSTRUCTORS(empno)

);

Output:

```
mysql> CREATE TABLE students (
    ->     stno INT PRIMARY KEY,
    ->     name VARCHAR(50),
    ->     addr VARCHAR(255),
    ->     city VARCHAR(50),
    ->     state VARCHAR(2),
    ->     zip VARCHAR(10)
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql>
mysql> CREATE TABLE INSTRUCTORS (
    ->     empno INT PRIMARY KEY,
    ->     name VARCHAR(50),
    ->     rank VARCHAR(20),
    ->     roomno VARCHAR(10),
    ->     telno VARCHAR(15)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> CREATE TABLE COURSES (
    ->     cno INT PRIMARY KEY,
    ->     cname VARCHAR(50),
    ->     cr INT,
    ->     cap INT
    -> );
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> CREATE TABLE GRADES (
    ->     stno INT,
    ->     empno INT,
    ->     cno INT,
    ->     sem VARCHAR(10),
    ->     year INT,
    ->     grade INT,
    ->     PRIMARY KEY (stno),
    ->     FOREIGN KEY (stno) REFERENCES students(stno),
    ->     FOREIGN KEY (empno) REFERENCES INSTRUCTORS(empno),
    ->     FOREIGN KEY (cno) REFERENCES COURSES(cno)
    -> );
Query OK, 0 rows affected (0.04 sec)

mysql>
mysql> CREATE TABLE ADVISING (
    ->     stno INT,
    ->     empno INT,
    ->     PRIMARY KEY (stno, empno),
    ->     FOREIGN KEY (stno) REFERENCES students(stno),
    ->     FOREIGN KEY (empno) REFERENCES INSTRUCTORS(empno)
    -> );
Query OK, 0 rows affected (0.04 sec)
```

Code:

```
INSERT INTO COURSES (cno, cname, cr, cap)
VALUES
   (1, 'Math101', 3, 30),
   (2, 'CS210', 4, 25),
   (3, 'Physics101', 3, 20);


INSERT INTO students (stno, name)
 VALUES
       (1, 'John Doe'),
       (2, 'Jane Smith'),
       (3, 'Alice Johnson');


INSERT INTO instructors (empno, name)
 VALUES
      (101, 'Instructor A'),
      (102, 'Instructor B'),
      (103, 'Instructor C');
```

```
INSERT INTO GRADES (stno, empno, cno, sem, year, grade)

VALUES

   (1, 101, 1, 'Fall', 2021, 85),

   (2, 102, 2, 'Fall', 2021, 92),

   (3, 103, 3, 'Fall', 2021, 78);


INSERT INTO ADVISING (stno, empno)

VALUES

   (1, 101),

   (2, 102),

   (3, 103);
```

Output:

```
mysql> INSERT INTO COURSES (cno, cname, cr, cap)
    -> VALUES
    ->     (1, 'Math101', 3, 30),
    ->     (2, 'CS210', 4, 25),
    ->     (3, 'Physics101', 3, 20);
Query OK, 3 rows affected (0.04 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
mysql> INSERT INTO students (stno, name)
    -> VALUES
    ->     (1, 'John Doe'),
    ->     (2, 'Jane Smith'),
    ->     (3, 'Alice Johnson');
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

```
mysql> INSERT INTO instructors (empno, name)
    -> VALUES
    ->     (101, 'Instructor A'),
    ->     (102, 'Instructor B'),
    ->     (103, 'Instructor C');
Query OK, 3 rows affected (0.03 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> INSERT INTO GRADES (stno, empno, cno, sem, year, grade)
    -> VALUES
    ->     (1, 101, 1, 'Fall', 2021, 85),
    ->     (2, 102, 2, 'Fall', 2021, 92),
    ->     (3, 103, 3, 'Fall', 2021, 78);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> INSERT INTO ADVISING (stno, empno)
    -> VALUES
    ->     (1, 101),
    ->     (2, 102),
    ->     (3, 103);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

For odd rollnumbers(any 10)

1. **Find the names of students who took some four-credit courses.**

Code:

```
SELECT DISTINCT s.name

FROM students s

JOIN grades g ON s.stno = g.stno

JOIN courses c ON g.cno = c.cno

WHERE c.cr = 4;
```

Output:

```
mysql> SELECT DISTINCT s.name
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> JOIN courses c ON g.cno = c.cno
    -> WHERE c.cr = 4;
+------------+
| name       |
+------------+
| Jane Smith |
+------------+
1 row in set (0.00 sec)
```

2. Find the names of students who took every four-credit course.

Code:

```
SELECT s.name
FROM students s
WHERE NOT EXISTS (
   SELECT 1
   FROM courses c
   WHERE c.cr = 4 AND NOT EXISTS (
      SELECT 1
      FROM grades g
      WHERE g.stno = s.stno AND g.cno = c.cno
   )
);
```

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> WHERE NOT EXISTS (
    ->     SELECT 1
    ->     FROM courses c
    ->     WHERE c.cr = 4 AND NOT EXISTS (
    ->         SELECT 1
    ->         FROM grades g
    ->         WHERE g.stno = s.stno AND g.cno = c.cno
    ->     )
    -> );
+------------+
| name       |
+------------+
| Jane Smith |
+------------+
1 row in set (0.00 sec)
```

3. **Find the names of students who took a course with an instructor who is also their advisor.**

Code:

```
SELECT DISTINCT s.name

FROM students s

JOIN grades g ON s.stno = g.stno

JOIN instructors i ON g.empno = i.empno

JOIN advising a ON s.stno = a.stno

WHERE g.empno = a.empno;
```

Output:

```
mysql> SELECT DISTINCT s.name
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> JOIN instructors i ON g.empno = i.empno
    -> JOIN advising a ON s.stno = a.stno
    -> WHERE g.empno = a.empno;
+---------------+
| name          |
+---------------+
| John Doe      |
| Jane Smith    |
| Alice Johnson |
+---------------+
3 rows in set (0.00 sec)
```

4. Find the names of students who took cs210 and cs310.

Code:

```
SELECT s.name
FROM students s
WHERE EXISTS (
   SELECT 1
   FROM grades g
   JOIN courses c ON g.cno = c.cno
   WHERE s.stno = g.stno AND c.cname = 'cs210'
)
AND EXISTS (
   SELECT 1
   FROM grades g
   JOIN courses c ON g.cno = c.cno
   WHERE s.stno = g.stno AND c.cname = 'cs310'
);
```

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> WHERE EXISTS (
    ->     SELECT 1
    ->     FROM grades g
    ->     JOIN courses c ON g.cno = c.cno
    ->     WHERE s.stno = g.stno AND c.cname = 'cs210'
    -> )
    -> AND EXISTS (
    ->     SELECT 1
    ->     FROM grades g
    ->     JOIN courses c ON g.cno = c.cno
    ->     WHERE s.stno = g.stno AND c.cname = 'cs310'
    -> );
Empty set (0.00 sec)
```

5. **Find the names of all students whose advisor is not a full professor.**

Code:

```
SELECT DISTINCT s.name

FROM students s

JOIN advising a ON s.stno = a.stno

JOIN instructors i ON a.empno = i.empno

WHERE i.rank <> 'Full Professor';
```

Output:

```
mysql> SELECT DISTINCT s.name
    -> FROM students s
    -> JOIN advising a ON s.stno = a.stno
    -> JOIN instructors i ON a.empno = i.empno
    -> WHERE i.rank <> 'Full Professor';
Empty set (0.00 sec)
```

6. **Find instructors who taught students who are advised by another instructor who shares the same room.**

Code:

```
SELECT DISTINCT i1.name

FROM instructors i1

JOIN grades g ON i1.empno = g.empno

JOIN advising a ON g.stno = a.stno

JOIN instructors i2 ON a.empno = i2.empno

WHERE i1.roomno = i2.roomno AND i1.empno <> i2.empno;
```

Output:

```
mysql> SELECT DISTINCT i1.name
    -> FROM instructors i1
    -> JOIN grades g ON i1.empno = g.empno
    -> JOIN advising a ON g.stno = a.stno
    -> JOIN instructors i2 ON a.empno = i2.empno
    -> WHERE i1.roomno = i2.roomno AND i1.empno <> i2.empno;
Empty set (0.00 sec)
```

7. **Find course numbers for courses that enroll exactly two students**

Code:

```
SELECT g.cno

FROM grades g

GROUP BY g.cno

HAVING COUNT(DISTINCT g.stno) = 2;
```

Output:

```
mysql> SELECT g.cno
    -> FROM grades g
    -> GROUP BY g.cno
    -> HAVING COUNT(DISTINCT g.stno) = 2;
Empty set (0.00 sec)
```

8. **Find the names of all students for whom no other student lives in the same city.**

Code:

```
SELECT s1.name

FROM students s1

WHERE NOT EXISTS (

  SELECT 1

  FROM students s2
```

```
   WHERE s1.city = s2.city AND s1.stno <> s2.stno
);
```

Output:

```
mysql> SELECT s1.name
    -> FROM students s1
    -> WHERE NOT EXISTS (
    ->     SELECT 1
    ->     FROM students s2
    ->     WHERE s1.city = s2.city AND s1.stno <> s2.stno
    -> );
+---------------+
| name          |
+---------------+
| John Doe      |
| Jane Smith    |
| Alice Johnson |
+---------------+
3 rows in set (0.00 sec)
```

9. **Find course numbers of courses taken by students who live in Boston and which are taught by an associate professor.**

Code:

```
SELECT DISTINCT g.cno
FROM grades g
JOIN students s ON g.stno = s.stno
JOIN instructors i ON g.empno = i.empno
WHERE s.city = 'Boston' AND i.rank = 'Associate Professor';
```

Output:

```
mysql> SELECT DISTINCT g.cno
    -> FROM grades g
    -> JOIN students s ON g.stno = s.stno
    -> JOIN instructors i ON g.empno = i.empno
    -> WHERE s.city = 'Boston' AND i.rank = 'Associate Professor';
Empty set (0.00 sec)
```

10. Find the telephone numbers of instructors who teach a course taken by any student who lives in Boston.

Code:

```
SELECT DISTINCT i.telno

FROM instructors i

JOIN grades g ON i.empno = g.empno

JOIN students s ON g.stno = s.stno

WHERE s.city = 'Boston';
```

Output:

```
mysql> SELECT DISTINCT i.telno
    -> FROM instructors i
    -> JOIN grades g ON i.empno = g.empno
    -> JOIN students s ON g.stno = s.stno
    -> WHERE s.city = 'Boston';
Empty set (0.00 sec)
```

11. Find names of students who took every course taken by Richard Pierce.

Code:

```
SELECT s.name

FROM students s

WHERE NOT EXISTS (

   SELECT 1

   FROM grades g1

   JOIN students rp ON rp.name = 'Richard Pierce'

   JOIN grades g2 ON rp.stno = g2.stno

   WHERE g1.cno = g2.cno AND g1.stno <> s.stno
```

);

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> WHERE NOT EXISTS (
    ->     SELECT 1
    ->     FROM grades g1
    ->     JOIN students rp ON rp.name = 'Richard Pierce'
    ->     JOIN grades g2 ON rp.stno = g2.stno
    ->     WHERE g1.cno = g2.cno AND g1.stno <> s.stno
    -> );
+---------------+
| name          |
+---------------+
| John Doe      |
| Jane Smith    |
| Alice Johnson |
+---------------+
3 rows in set (0.00 sec)
```

12. Find the names of students who took only one course.

Code:

```
SELECT s.name

FROM students s

JOIN grades g ON s.stno = g.stno

GROUP BY s.stno, s.name

HAVING COUNT(DISTINCT g.cno) = 1;
```

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> GROUP BY s.stno, s.name
    -> HAVING COUNT(DISTINCT g.cno) = 1;
+---------------+
| name          |
+---------------+
| John Doe      |
| Jane Smith    |
| Alice Johnson |
+---------------+
3 rows in set (0.01 sec)
```

13. Find the names of instructors who teach no course.

Code:

```
SELECT i.name

FROM instructors i

LEFT JOIN grades g ON i.empno = g.empno

WHERE g.cno IS NULL;
```

Output:

```
mysql> SELECT i.name
    -> FROM instructors i
    -> LEFT JOIN grades g ON i.empno = g.empno
    -> WHERE g.cno IS NULL;
Empty set (0.00 sec)
```

14. Find the names of the instructors who taught only one course during the spring semester of 2001.

Code:

```
SELECT i.name

FROM instructors i

JOIN grades g ON i.empno = g.empno

WHERE g.sem = 'Spring' AND g.year = 2001

GROUP BY i.empno, i.name

HAVING COUNT(DISTINCT g.cno) = 1;
```

Output:

```
mysql> SELECT i.name
    -> FROM instructors i
    -> JOIN grades g ON i.empno = g.empno
    -> WHERE g.sem = 'Spring' AND g.year = 2001
    -> GROUP BY i.empno, i.name
    -> HAVING COUNT(DISTINCT g.cno) = 1;
Empty set (0.00 sec)
```

For even rollnumbers(any 10)

1. **Find the names of students who took only four-credit courses.**

Code:

```
SELECT s.name
FROM students s
JOIN grades g ON s.stno = g.stno
JOIN courses c ON g.cno = c.cno
GROUP BY s.stno, s.name
HAVING COUNT(DISTINCT CASE WHEN c.cr = 4 THEN g.cno END) =
COUNT(DISTINCT g.cno)
    AND COUNT(DISTINCT CASE WHEN c.cr <> 4 THEN g.cno END) =
0;
```

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> JOIN courses c ON g.cno = c.cno
    -> GROUP BY s.stno, s.name
    -> HAVING COUNT(DISTINCT CASE WHEN c.cr = 4 THEN g.cno END) = COUNT(DISTINCT g.cno)
    ->        AND COUNT(DISTINCT CASE WHEN c.cr <> 4 THEN g.cno END) = 0;
+------------+
| name       |
+------------+
| Jane Smith |
+------------+
1 row in set (0.00 sec)
```

2. **Find the names of students who took no four-credit courses.**

Code:

```
SELECT s.name

FROM students s

WHERE NOT EXISTS (

  SELECT 1

  FROM grades g

  JOIN courses c ON g.cno = c.cno
```

```
   WHERE g.stno = s.stno AND c.cr = 4
);
```

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> WHERE NOT EXISTS (
    ->     SELECT 1
    ->     FROM grades g
    ->     JOIN courses c ON g.cno = c.cno
    ->     WHERE g.stno = s.stno AND c.cr = 4
    -> );
+---------------+
| name          |
+---------------+
| John Doe      |
| Alice Johnson |
+---------------+
2 rows in set (0.00 sec)
```

3. **Find the names of students who took cs210 or cs310.**

Code:

```
SELECT DISTINCT s.name
FROM students s
JOIN grades g ON s.stno = g.stno
JOIN courses c ON g.cno = c.cno
WHERE c.cname IN ('cs210', 'cs310');
```

Output:

```
mysql> SELECT DISTINCT s.name
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> JOIN courses c ON g.cno = c.cno
    -> WHERE c.cname IN ('cs210', 'cs310');
+------------+
| name       |
+------------+
| Jane Smith |
+------------+
1 row in set (0.00 sec)
```

4. **Find names of all students who have a cs210 grade higher than the highest grade given in cs310 and did not take any course with Prof. Evans.**

Code:

```
SELECT DISTINCT s.name
FROM students s
JOIN grades g1 ON s.stno = g1.stno
JOIN courses c1 ON g1.cno = c1.cno
WHERE c1.cname = 'cs210' AND g1.grade > (
   SELECT MAX(g2.grade)
   FROM grades g2
   JOIN courses c2 ON g2.cno = c2.cno
   WHERE c2.cname = 'cs310'
)
AND NOT EXISTS (
   SELECT 1
   FROM grades g3
   JOIN instructors i ON g3.empno = i.empno
   WHERE g3.stno = s.stno AND i.name = 'Prof. Evans'
);
```

Output:

```
mysql> SELECT DISTINCT s.name
    -> FROM students s
    -> JOIN grades g1 ON s.stno = g1.stno
    -> JOIN courses c1 ON g1.cno = c1.cno
    -> WHERE c1.cname = 'cs210' AND g1.grade > (
    ->     SELECT MAX(g2.grade)
    ->     FROM grades g2
    ->     JOIN courses c2 ON g2.cno = c2.cno
    ->     WHERE c2.cname = 'cs310'
    -> )
    -> AND NOT EXISTS (
    ->     SELECT 1
    ->     FROM grades g3
    ->     JOIN instructors i ON g3.empno = i.empno
    ->     WHERE g3.stno = s.stno AND i.name = 'Prof. Evans'
    -> );
Empty set (0.00 sec)
```

5. **Find course numbers for courses that enrol at least two students; solve the same query for courses that enroll at least three students.**

Code:

-- For courses with at least two students

SELECT g.cno

FROM grades g

GROUP BY g.cno

HAVING COUNT(DISTINCT g.stno) >= 2;

-- For courses with at least three students

SELECT g.cno

FROM grades g

GROUP BY g.cno

```
HAVING COUNT(DISTINCT g.stno) >= 3;
```

Output:

```
mysql> -- For courses with at least two students
mysql> SELECT g.cno
    -> FROM grades g
    -> GROUP BY g.cno
    -> HAVING COUNT(DISTINCT g.stno) >= 2;
Empty set (0.00 sec)

mysql>
mysql> -- For courses with at least three students
mysql> SELECT g.cno
    -> FROM grades g
    -> GROUP BY g.cno
    -> HAVING COUNT(DISTINCT g.stno) >= 3;
Empty set (0.00 sec)
```

6. **Find the names of students who obtained the highest grade in cs210.**

Code:

```
SELECT s.name

FROM students s

JOIN grades g ON s.stno = g.stno

JOIN courses c ON g.cno = c.cno

WHERE c.cname = 'cs210' AND g.grade = (

   SELECT MAX(grade)

   FROM grades g1

   JOIN courses c1 ON g1.cno = c1.cno

   WHERE c1.cname = 'cs210'

);
```

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> JOIN courses c ON g.cno = c.cno
    -> WHERE c.cname = 'cs210' AND g.grade = (
    ->     SELECT MAX(grade)
    ->     FROM grades g1
    ->     JOIN courses c1 ON g1.cno = c1.cno
    ->     WHERE c1.cname = 'cs210'
    -> );
+------------+
| name       |
+------------+
| Jane Smith |
+------------+
1 row in set (0.00 sec)
```

7. **Find the names of instructors who teach courses attended by students who took a course with an instructor who is an assistant professor.**

Code:

```
SELECT DISTINCT i1.name

FROM instructors i1

JOIN grades g ON i1.empno = g.empno

JOIN students s ON g.stno = s.stno

JOIN grades g2 ON s.stno = g2.stno

JOIN instructors i2 ON g2.empno = i2.empno

WHERE i2.rank = 'Assistant Professor';
```

Output:

```
mysql> SELECT DISTINCT i1.name
    -> FROM instructors i1
    -> JOIN grades g ON i1.empno = g.empno
    -> JOIN students s ON g.stno = s.stno
    -> JOIN grades g2 ON s.stno = g2.stno
    -> JOIN instructors i2 ON g2.empno = i2.empno
    -> WHERE i2.rank = 'Assistant Professor';
Empty set (0.00 sec)
```

8. **Find the lowest grade of a student who took a course during the spring of 2003.**

Code:

```
SELECT MIN(g.grade)

FROM grades g

WHERE g.sem = 'Spring' AND g.year = 2003;
```

Output:

```
mysql> SELECT MIN(g.grade)
    -> FROM grades g
    -> WHERE g.sem = 'Spring' AND g.year = 2003;
+--------------+
| MIN(g.grade) |
+--------------+
|         NULL |
+--------------+
1 row in set (0.00 sec)
```

9. **Find the names for students such that if prof. Evans teaches a course, then the student takes that course (although not necessarily with prof. Evans).**

Code:

```
SELECT s.name

FROM students s

WHERE NOT EXISTS (

  SELECT 1

  FROM courses c

  WHERE EXISTS (

    SELECT 1

    FROM grades g
```

```
    WHERE g.stno = s.stno AND g.cno = c.cno
  ) AND EXISTS (
    SELECT 1
    FROM grades g
    JOIN instructors i ON g.empno = i.empno
    WHERE g.cno = c.cno AND i.name = 'Prof. Evans'
  )
);
```

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> WHERE NOT EXISTS (
    ->     SELECT 1
    ->     FROM courses c
    ->     WHERE EXISTS (
    ->         SELECT 1
    ->         FROM grades g
    ->         WHERE g.stno = s.stno AND g.cno = c.cno
    ->     ) AND EXISTS (
    ->         SELECT 1
    ->         FROM grades g
    ->         JOIN instructors i ON g.empno = i.empno
    ->         WHERE g.cno = c.cno AND i.name = 'Prof. Evans'
    ->     )
    -> );
+---------------+
| name          |
+---------------+
| John Doe      |
| Jane Smith    |
| Alice Johnson |
+---------------+
3 rows in set (0.00 sec)
```

10. Find the names of students whose advisor did not teach them any course.

Code:

```
SELECT s.name

FROM students s

JOIN advising a ON s.stno = a.stno

LEFT JOIN grades g ON s.stno = g.stno AND g.empno = a.empno

WHERE g.empno IS NULL;
```

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> JOIN advising a ON s.stno = a.stno
    -> LEFT JOIN grades g ON s.stno = g.stno AND g.empno = a.empno
    -> WHERE g.empno IS NULL;
Empty set (0.00 sec)
```

11. Find the names of students who have failed all their courses (failing is defined as a grade less than 60).

Code:

```
SELECT s.name

FROM students s

JOIN grades g ON s.stno = g.stno

GROUP BY s.stno, s.name

HAVING MIN(g.grade) < 60 AND MAX(g.grade) < 60;
```

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> GROUP BY s.stno, s.name
    -> HAVING MIN(g.grade) < 60 AND MAX(g.grade) < 60;
Empty set (0.00 sec)
```

12. Find the highest grade of a student who never took cs110.

Code:

```
SELECT MAX(g.grade)
FROM grades g
WHERE g.stno NOT IN (
   SELECT g2.stno
   FROM grades g2
   JOIN courses c ON g2.cno = c.cno
   WHERE c.cname = 'cs110'
)
GROUP BY g.stno;
```

Output:

```
mysql> SELECT MAX(g.grade)
    -> FROM grades g
    -> WHERE g.stno NOT IN (
    ->     SELECT g2.stno
    ->     FROM grades g2
    ->     JOIN courses c ON g2.cno = c.cno
    ->     WHERE c.cname = 'cs110'
    -> )
    -> GROUP BY g.stno;
+--------------+
| MAX(g.grade) |
+--------------+
|           85 |
|           92 |
|           78 |
+--------------+
3 rows in set (0.00 sec)
```

13. Find the names of students who do not have an advisor.

Code:

```
SELECT s.name

FROM students s

LEFT JOIN advising a ON s.stno = a.stno

WHERE a.empno IS NULL;
```

Output:

```
mysql> SELECT s.name
    -> FROM students s
    -> LEFT JOIN advising a ON s.stno = a.stno
    -> WHERE a.empno IS NULL;
Empty set (0.00 sec)
```

14. Find names of courses taken by students who do not live in Massachusetts (MA).

Code:

```
SELECT DISTINCT c.cname

FROM students s

JOIN grades g ON s.stno = g.stno

JOIN courses c ON g.cno = c.cno

WHERE s.state <> 'MA';
```

Output:

```
mysql> SELECT DISTINCT c.cname
    -> FROM students s
    -> JOIN grades g ON s.stno = g.stno
    -> JOIN courses c ON g.cno = c.cno
    -> WHERE s.state <> 'MA';
Empty set (0.00 sec)
```

PRACTICAL N0 – 3

Aim: CRUD using Mongodb

#Creating a New database:

Code:

mongod –version

Output:

```
C:\Users\Admin>mongod --version
db version v6.0.13
Build Info: {
    "version": "6.0.13",
    "gitVersion": "3b13907f9bdf6bd3264d67140d6c215d51bbd20c",
    "modules": [],
    "allocator": "tcmalloc",
    "environment": {
        "distmod": "windows",
        "distarch": "x86_64",
        "target_arch": "x86_64"
    }
}
```

mongosh

```
C:\Users\Admin>mongosh
Current Mongosh Log ID: 6780ea0e36336b93975d893f
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2
.1.4
Using MongoDB:          6.0.13
Using Mongosh:          2.1.4
mongosh 2.2.0 is available for download: https://www.mongodb.com/try/download/shell

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2024-12-19T10:52:33.605+05:30: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
------
```

#Create a database name userdb

Code:

use userdb;

Output:

```
test> use userdb;
```

```
test> use userdb;
switched to db userdb
```

#Creating a New Collection:

Code:

db.createCollection("users")

Output:

```
userdb> db.createCollection("users")
{ ok: 1 }
```

#Create Operation

1.insertOne():

Code:

db.users.insertOne({

name:"Angela",

age:27,

});

```
userdb> db.users.insertOne({
... name:"Angela",
... age:27,
... });
```

Output:

```
{
  acknowledged: true,
  insertedId: ObjectId('6780ece736336b93975d8940')
}
```

2.insertMany()

Code:

db.users.insertMany([{ name:"Angela", age:27, }, { name:"Dwight", age:30, }, {name:"Jim", age:29,}]);

```
userdb> db.users.insertMany([ { name:"Angela", age:27, }, { name:"Dwight", age:30, }, {name:"Jim", age:29,}]);
```

Output:

```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6780ee8936336b93975d8941'),
    '1': ObjectId('6780ee8936336b93975d8942'),
    '2': ObjectId('6780ee8936336b93975d8943')
  }
}
```

#Read Operations

1.find()

Code:

db.users.find()

```
userdb> db.users.find()
```

Output:

```
[
  {
    _id: ObjectId('6780ece736336b93975d8940'),
    name: 'Angela',
    age: 27
  },
  {
    _id: ObjectId('6780ee8936336b93975d8941'),
    name: 'Angela',
    age: 27
  },
  {
    _id: ObjectId('6780ee8936336b93975d8942'),
    name: 'Dwight',
    age: 30
  },
  { _id: ObjectId('6780ee8936336b93975d8943'), name: 'Jim', age: 29 }
]
```

Code:

db.users.find({age: {$gt:29}}, {name:1, age:1 })

```
userdb> db.users.find({age: {$gt:29}}, {name:1, age:1 })
```

Output:

```
[
  {
    _id: ObjectId('6780ee8936336b93975d8942'),
    name: 'Dwight',
    age: 30
  }
]
```

2.findOne()

Code:

db.users.findOne({name:"Jim"})

```
userdb> db.users.findOne({name:"Jim"})
```

Output:

```
{ _id: ObjectId('6780ee8936336b93975d8943'), name: 'Jim', age: 29 }
```

#Update Operations

1.updateOne()

Code:

db.users.updateOne({name:"Angela"},{$set:{email:"angela@gmail.com"} })

```
userdb> db.users.updateOne({name:"Angela"},{$set:{email:"angela@gmail.com"} })
```

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Output:

db.users.findOne({name:"Angela"})

```
userdb> db.users.findOne({name:"Angela"})
{
  _id: ObjectId('6780ece736336b93975d8940'),
  name: 'Angela',
  age: 27,
  email: 'angela@gmail.com'
}
```

2.updateMany

Code:

db.users.updateMany({ age:{$lt: 30}}, {$set: {status:"active"} })

```
userdb> db.users.updateMany({ age:{$lt: 30}}, {$set: {status:"active"} })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
}
```

Output:

db.users.find()

```
userdb> db.users.find()
[
  {
    _id: ObjectId('6780ece736336b93975d8940'),
    name: 'Angela',
    age: 27,
    email: 'angela@gmail.com',
    status: 'active'
  },
  {
    _id: ObjectId('6780ee8936336b93975d8941'),
    name: 'Angela',
    age: 27,
    status: 'active'
  },
  {
    _id: ObjectId('6780ee8936336b93975d8942'),
    name: 'Dwight',
    age: 30
  },
  {
    _id: ObjectId('6780ee8936336b93975d8943'),
    name: 'Jim',
    age: 29,
    status: 'active'
  }
]
```

#Delete Operations

1.deleteOne()

Code:

db.users.deleteOne({name:"Angela"})

```
userdb> db.users.deleteOne({name:"Angela"})
{ acknowledged: true, deletedCount: 1 }
```

Output:

db.users.find()

```
userdb> db.users.find()
[
  {
    _id: ObjectId('6780ee8936336b93975d8941'),
    name: 'Angela',
    age: 27,
    status: 'active'
  },
  {
    _id: ObjectId('6780ee8936336b93975d8942'),
    name: 'Dwight',
    age: 30
  },
  {
    _id: ObjectId('6780ee8936336b93975d8943'),
    name: 'Jim',
    age: 29,
    status: 'active'
  }
]
```

2.deleteMany()

Code:

db.users.deleteMany({age:{$lt:30} })

```
userdb> db.users.deleteMany({age:{$lt:30} })
{ acknowledged: true, deletedCount: 2 }
```

Output:

```
userdb> db.users.deleteMany({age:{$lt:30} })
{ acknowledged: true, deletedCount: 2 }
userdb> db.users.find()
[
  {
    _id: ObjectId('6780ee8936336b93975d8942'),
    name: 'Dwight',
    age: 30
  }
]
```

#drop()

db.users.drop()

```
userdb> db.users.drop()
true
```

PRACTICAL N0 – 4

Aim: Indexing using Mongodb

1. Mongo DB indexing
a. Create index in Mongo DB
b. Finding the indexes in a collection
c. Drop indexes in a collection
d. Drop all the indexes

```
use students
db.createCollection("studentgrades")
db.studentgrades.insertMany(
 [
 {name: "Barry", subject: "Maths", score: 92},
 {name: "Kent", subject: "Physics", score: 87},
 {name: "Harry", subject: "Maths", score: 99, notes: "Exceptional Performance"},
 {name: "Alex", subject: "Literature", score: 78},
 {name: "Tom", subject: "History", score: 65, notes: "Adequate"}
 ]
)db
db.studentgrades.find({},{_id:0})
db.studentgrades.find().pretty()

db.studentgrades.createIndex( {name: 1}, {name: "student name index"} )
```

Code:

use students;

db.createCollection("studentsgrades")

db.studentgrades.insertMany(

[

{name:"Barray",subject:"Maths",score:92},

{name:"Kent",subject:"Physics",score:87},

{name:"Harry",subject:"Maths",score:99,notes:"Exceptional Performance"},

{name:"Alex",subject:"Literature",score:78},

{name:"Tom",subject:"History",score:65,notes:"Adequate"}]);

Output:

```
test> use students;
switched to db students
students> db.createCollection("studentgrades")
MongoServerError[NamespaceExists]: Collection students.studentgrades already exists.
students> db.createCollection("studentsgrades")
{ ok: 1 }
students> db.studentgrades.insertMany(
... [
... {name:"Barray",subject:"Maths",score:92},
... {name:"Kent",subject:"Physics",score:87},
... {name:"Harry",subject:"Maths",score:99,notes:"Exceptional Performance"},
... {name:"Alex",subject:"Literature",score:78},
... {name:"Tom",subject:"History",score:65,notes:"Adequate"}]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('678a29fd0b1aea3e66bb51cb'),
    '1': ObjectId('678a29fd0b1aea3e66bb51cc'),
    '2': ObjectId('678a29fd0b1aea3e66bb51cd'),
    '3': ObjectId('678a29fd0b1aea3e66bb51ce'),
    '4': ObjectId('678a29fd0b1aea3e66bb51cf')
  }
}
```

Code:

db.studentgrades.find({},{_id:0});

Output:

```
students> db.studentgrades.find({},{_id:0});
[
  { name: 'Barry', subject: 'Maths', score: 92 },
  { name: 'kent', subject: 'physics', score: 98 },
  {
    name: 'Harry',
    subject: 'Maths',
    score: 99,
    notes: 'Exceptional Performance'
  },
  { name: 'Alex', subject: 'Literature', score: 78 },
  { name: 'Tom', subject: 'History', score: 78 },
  { name: 'Tom', subject: 'History', score: 65, notes: 'Adequate' },
  { name: 'Barray', subject: 'Maths', score: 92 },
  { name: 'Kent', subject: 'Physics', score: 87 },
  {
    name: 'Harry',
    subject: 'Maths',
    score: 99,
    notes: 'Exceptional Performance'
  },
  { name: 'Alex', subject: 'Literature', score: 78 },
```

Code:

db.studentgrades.find().pretty();

Output:

```
students> db.studentgrades.find().pretty();
[
  {
    _id: ObjectId('65f52d56270308fddd06a1ab'),
    name: 'Barry',
    subject: 'Maths',
    score: 92
  },
  {
    _id: ObjectId('65f52d56270308fddd06a1ac'),
    name: 'kent',
    subject: 'physics',
    score: 98
  },
  {
    _id: ObjectId('65f52d56270308fddd06a1ad'),
    name: 'Harry',
    subject: 'Maths',
    score: 99,
    notes: 'Exceptional Performance'
  },
  {
    _id: ObjectId('65f52d56270308fddd06a1ae'),
    name: 'Alex',
    subject: 'Literature',
    score: 78
  },
  {
    _id: ObjectId('65f52d56270308fddd06a1af'),
    name: 'Tom',
    subject: 'History',
    score: 78
  },
  {
    _id: ObjectId('65f52d56270308fddd06a1b0'),
    name: 'Tom',
    subject: 'History',
    score: 65,
    notes: 'Adequate'
  },
```

Code:

db.studentgrades.createIndex({name: 1},{name:"student name index"});

Output:

```
students> db.studentgrades.createIndex({name: 1},{name:"student name index"});
student name index
```

Finding indexes You can find all the available indexes in a MongoDB collection by using the getIndexes method. This will return all the indexes in a specific collection. db..getIndexes() Let's view all the indexes in the studentgrades collection using the following command:
db.studentgrades.getIndexes()

Code:

```
db.studentgrades.getIndexes();
```

Output:

```
students>
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { name: 1 }, name: 'student name index' }
]
```

Dropping indexes To delete an index from a collection, use the dropIndex method while specifying the index name to be dropped. db..dropIndex() Let's remove the user-created index with the index name student name index, as shown below.
db.studentgrades.dropIndex("student name index")

Code:

```
db.studentgrades.dropIndex("student name index");
```

Output:

```
students> db.studentgrades.dropIndex("student name index");
{ nIndexesWas: 2, ok: 1 }
```

You can also use the index field value for removing an index without a defined name:
db.studentgrades.dropIndex({name:1})

Code:

```
db.studentgrades.dropIndex({name:1});
```

Output:

```
students> db.studentgrades.dropIndex({name:1});
```

The dropIndexes command can also drop all the indexes excluding the default _id index. db.studentgrades.dropIndexes()

Code:

```
db.studentgrades.dropIndexes();
```

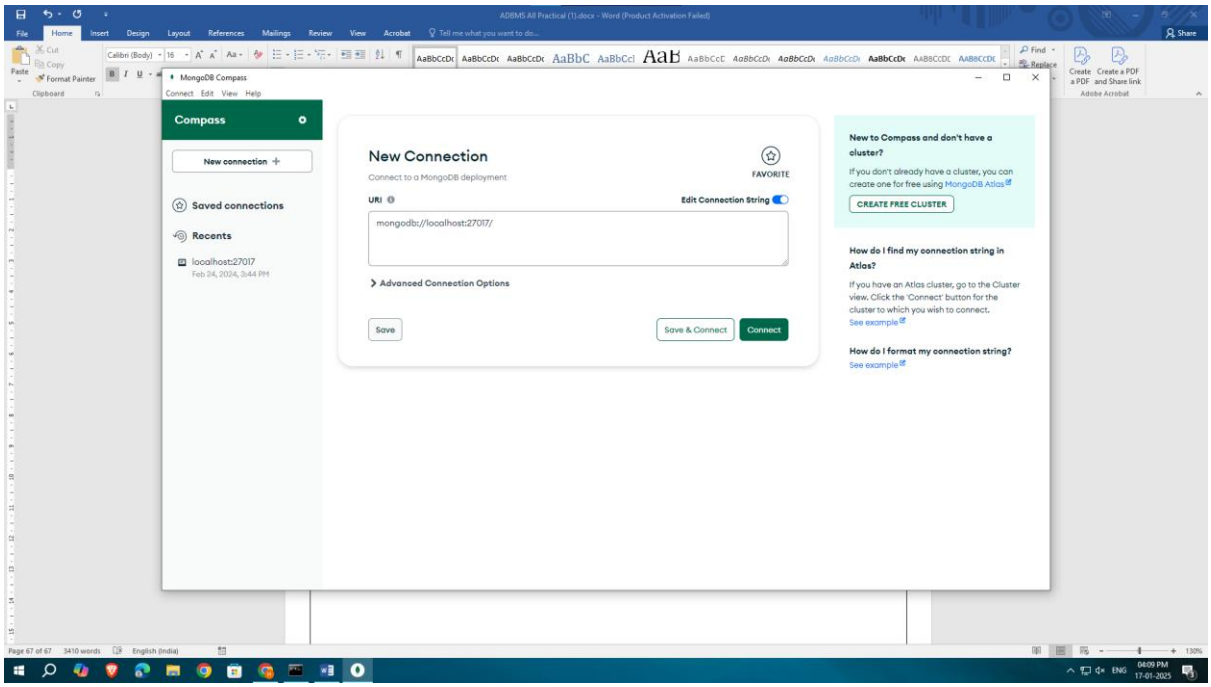
Output:

```
students> db.studentgrades.dropIndexes();
{
  nIndexesWas: 1,
  msg: 'non-_id indexes dropped for collection',
  ok: 1
}
```

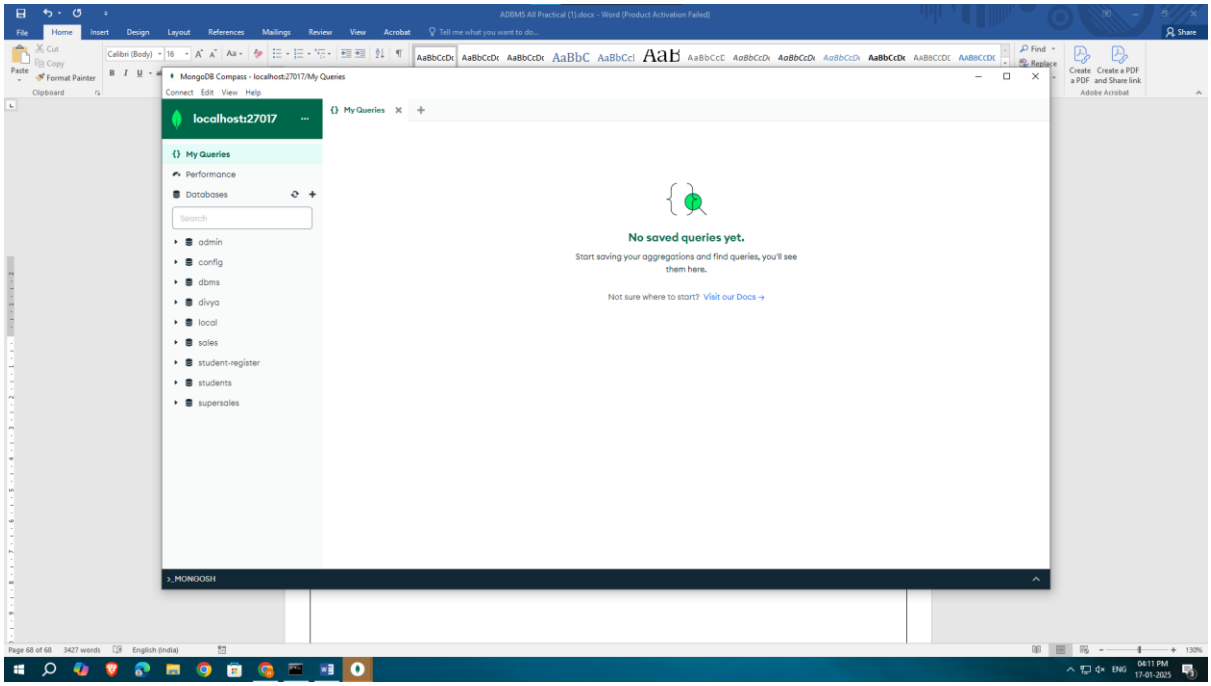
2. Create all the types of indexes (discussed in class) which will help in finding certain words in a document by using AIRPORT (dataset).

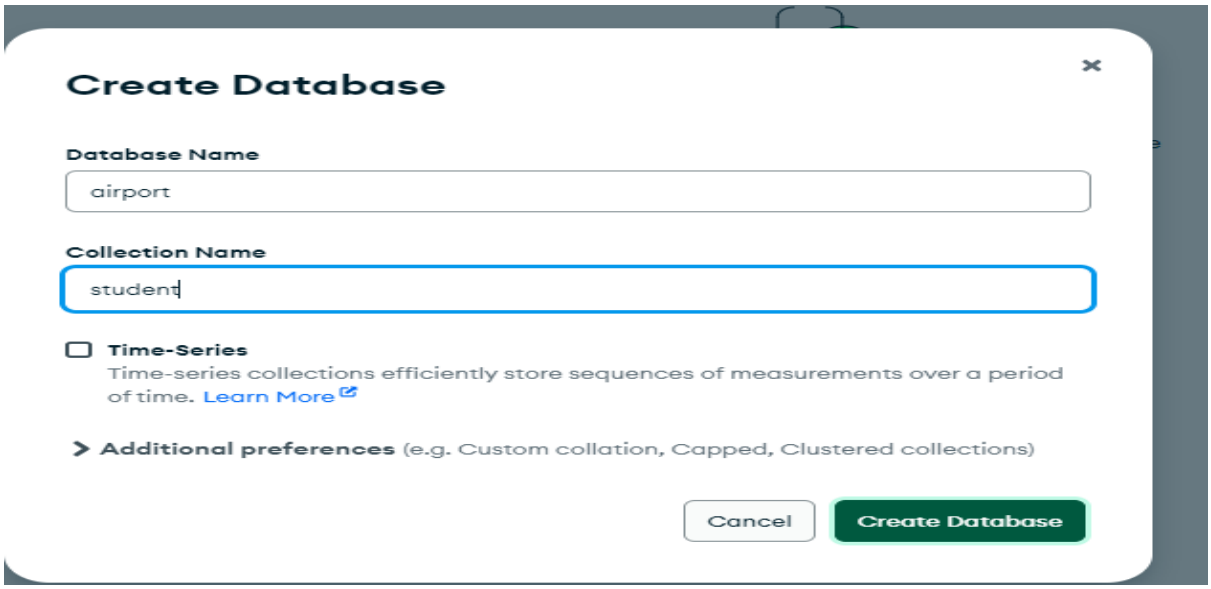
Step 1: Go to mongodb compass

Step 2: Connect to the localhost

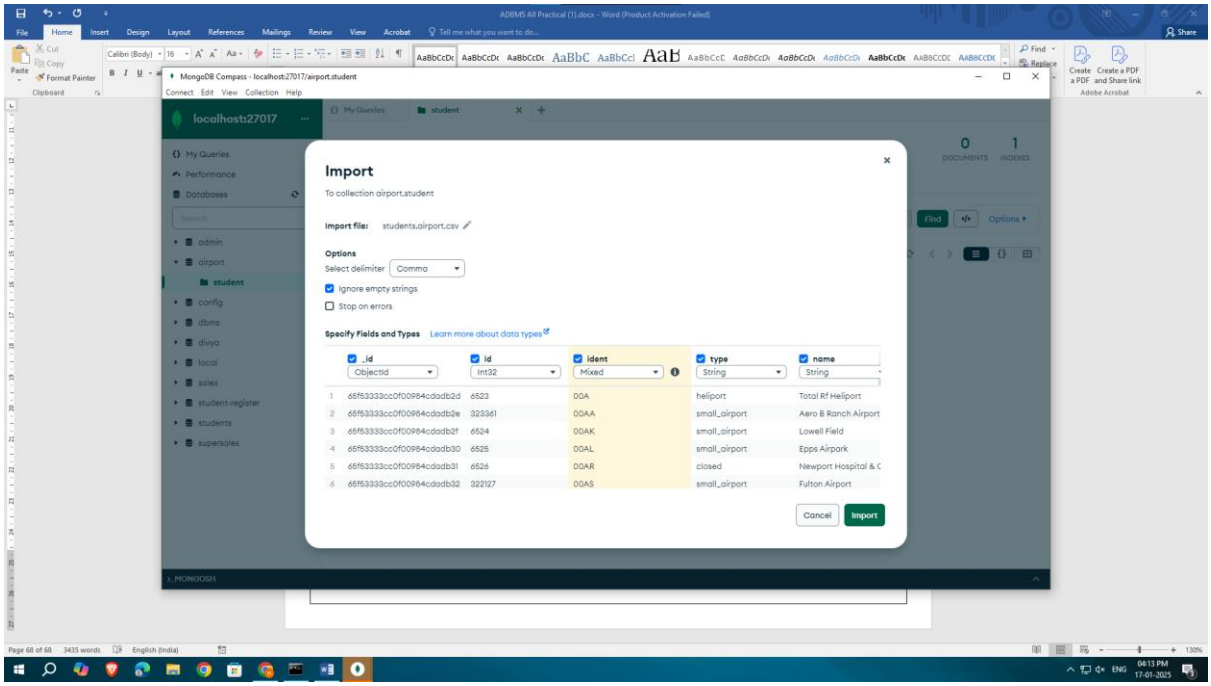


Step 3: Create a databases and upload airport file

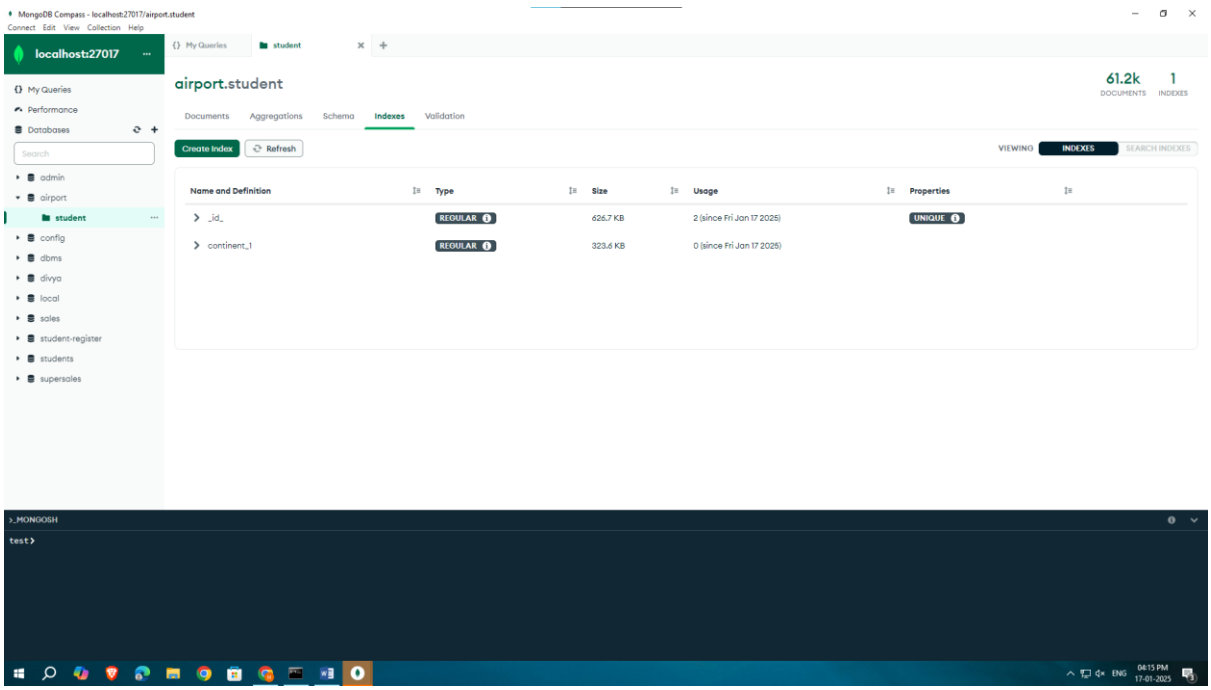




Step 4: Import data and click on import



Step 5: Create Indexes



Step 6: Using different indexes