Subject : 2301CS361 – Database Management Systems

Sr.	Practical
1	Database Name: **Branch_DIV_Rollno (Example: CSE_3C_101)** **Part A:** **Create following tables under above database.**

DEPOSIT

Column_Name	DataType
ACTNO	INT
CNAME	VARCHAR(50)
BNAME	VARCHAR(50)
AMOUNT	DECIMAL(8,2)
ADATE	DATETIME

BRANCH

Column_Name	DataType
BNAME	VARCHAR(50)
CITY	VARCHAR(50)

CUSTOMERS

Column_Name	DataType
CNAME	VARCHAR(50)
CITY	VARCHAR(50)

BORROW

Column_Name	DataType
LOANNO	INT
CNAME	VARCHAR(50)
BNAME	VARCHAR(50)
AMOUNT	DECIMAL(8,2)

Insert the data into above tables as shown below.

DEPOSIT

ACTNO	CNAME	BNAME	AMOUNT	ADATE
101	ANIL	VRCE	1000.00	1-3-95
102	SUNIL	AJNI	5000.00	4-1-96
103	MEHUL	KAROLBAGH	3500.00	17-11-95
104	MADHURI	CHANDI	1200.00	17-12-95
105	PRMOD	M.G. ROAD	3000.00	27-3-96
106	SANDIP	ANDHERI	2000.00	31-3-96
107	SHIVANI	VIRAR	1000.00	5-9-95
108	KRANTI	NEHRU PLACE	5000.00	2-7-95
109	MINU	POWAI	7000.00	10-8-95

BRANCH

BNAME	CITY
VRCE	NAGPUR
AJNI	NAGPUR
KAROLBAGH	DELHI
CHANDI	DELHI
DHARAMPETH	NAGPUR

M.G. ROAD	BANGLORE
ANDHERI	BOMBAY
VIRAR	BOMBAY
NEHRU PLACE	DELHI
POWAI	BOMBAY

CUSTOMERS

CNAME	CITY
ANIL	CALCUTTA
SUNIL	DELHI
MEHUL	BARODA
MANDAR	PATNA
MADHURI	NAGPUR
PRAMOD	NAGPUR
SANDIP	SURAT
SHIVANI	BOMBAY
KRANTI	BOMBAY
NAREN	BOMBAY

BORROW

LOANNO	CNAME	BNAME	AMOUNT
201	ANIL	VRCE	1000.00
206	MEHUL	AJNI	5000.00
311	SUNIL	DHARAMPETH	3000.00
321	MADHURI	ANDHERI	2000.00
375	PRMOD	VIRAR	8000.00
481	KRANTI	NEHRU PLACE	3000.00

From the above given tables perform the following SQL queries (SELECT Operation):

1. Retrieve all data from table DEPOSIT.
   SELECT * FROM DEPOSIT;
2. Retrieve all data from table BORROW.
   SELECT * FROM BORROW;
3. Retrieve all data from table CUSTOMERS.
   SELECT * FROM CUSTOMERS;
4. Insert a record (550,’JAY’,’AJNI’,NULL)in the BORROW table.
   INSERT INTO BORROW(LOANNO, CNAME, BNAME, AMOUNT) VALUES (550, 'JAY', 'AJNI', NULL);
5. Display Account No, Customer Name & Amount from DEPOSIT.
   SELECT ACTNO, CNAME, AMOUNT FROM DEPOSIT;
6. Display Loan No, Amount from BORROW.
   SELECT LOANNO, AMOUNT FROM BORROW;
7. Display loan details of all customers who belongs to ‘ANDHERI’ branch.
   SELECT * FROM BORROW WHERE BNAME = 'ANDHERI';
8. Give account no and amount of depositor, whose account no is equals to 106.
   SELECT ACTNO, AMOUNT FROM DEPOSIT WHERE ACTNO = 106;
9. Give name of borrowers having amount greater than 5000.
   SELECT CNAME FROM BORROW WHERE AMOUNT > 5000;

10. Give name of customers who opened account after date '1-12-96'.
    ```
    SELECT * FROM DEPOSIT WHERE ADATE> '1996-12-01';
    ```
11. Display name of customers whose account no is less than 105.
    ```
    SELECT CNAME FROM DEPOSIT WHERE ACTNO < 105;
    ```
12. Display name of customer who belongs to either ‘NAGPUR’ Or ‘DELHI’. (**OR & IN**)
    ```
    SELECT CNAME FROM CUSTOMERS WHERE CITY = 'NAGPUR' OR CITY = 'DELHI';
    SELECT CNAME FROM CUSTOMERS WHERE CITY IN ('NAGPUR', 'DELHI');
    ```
13. Display name of customers with branch whose amount is greater than 4000 and account no is less than 105.
    ```
    SELECT CNAME, BNAME FROM DEPOSIT WHERE AMOUNT> 4000 AND ACTNO < 105;
    ```
14. Find all borrowers whose amount is greater than equals to 3000 & less than equals to 8000. (**AND & BETWEEN**)
    ```
    SELECT * FROM BORROW WHERE AMOUNT >= 3000 AND AMOUNT <= 8000;
    SELECT * FROM BORROW WHERE AMOUNT BETWEEN 3000 AND 8000;
    ```
15. Find all depositors who do not belongs to ‘ANDHERI’ branch.
    ```
    SELECT * FROM DEPOSIT WHERE BNAME != 'ANDHERI';
    ```
16. Display Account No, Customer Name & Amount of such customers who belongs to ‘AJNI’, ‘KAROLBAGH’ Or ‘M.G.ROAD’ and Account No is less than 104.
    ```
    SELECT ACTNO, CNAME, AMOUNT FROM DEPOSIT WHERE BNAME IN ('AJNI', 'KAROLBAGH', 'M.G. ROAD') AND ACTNO < 104;
    ```
17. Display all the details of first five customers.
    ```
    SELECT TOP 5 * FROM CUSTOMERS
    ```
18. Display all the details of first three depositors whose amount is greater than 1000.
    ```
    SELECT TOP 3 * FROM DEPOSIT WHERE AMOUNT > 1000;
    ```
19. Display Loan No, Customer Name of first five borrowers whose branch name does not belongs to ‘ANDHERI’.
    ```
    SELECT TOP 5 LOANNO, CNAME FROM BORROW WHERE BNAME != 'ANDHERI';
    ```
20. Retrieve all unique cities using DISTINCT. (Use **Customers Table**)
    ```
    SELECT DISTINCT CITY FROM CUSTOMERS;
    ```
21. Retrieve all unique branches using DISTINCT. (Use **Branch Table**)
    ```
    SELECT DISTINCT BNAME FROM BRANCH;
    ```
22. Retrieve all the records of customer table as per their city name in ascending order.
    ```
    SELECT * FROM CUSTOMERS ORDER BY CITY ASC;
    ```
23. Retrieve all the records of deposit table as per their amount column in descending order.
    ```
    SELECT * FROM DEPOSIT ORDER BY AMOUNT DESC;
    ```
24. Update deposit amount of all customers from 3000 to 5000.
    ```
    UPDATE DEPOSIT SET AMOUNT= 5000 WHERE AMOUNT = 3000;
    ```
25. Change branch name of ANIL from VRCE to C.G. ROAD. (Use **Borrow Table**)
    ```
    UPDATE BORROW SET BNAME = 'C.G. ROAD' WHERE CNAME = 'ANIL' AND BNAME = 'VRCE';
    ```
26. Update Account No of SANDIP to 111 & Amount to 5000.
    ```
    UPDATE DEPOSIT SET ACTNO = 111, AMOUNT = 5000 WHERE CNAME = 'SANDIP';
    ```
27. Give 10% Increment in Loan Amount.
    ```
    UPDATE BORROW SET AMOUNT = AMOUNT * 1.10;
    ```
28. Update deposit amount of all depositors to 5000 whose account no between 103 & 107.

UPDATE DEPOSIT SET AMOUNT = 5000 WHERE ACTNO BETWEEN 103 AND 107;

29. Update amount of loan no 321 to *NULL*.
    UPDATE BORROW SET AMOUNT = NULL WHERE LOANNO = 321
30. Display the name of borrowers whose amount is *NULL*.
    SELECT CNAME FROM BORROW WHERE AMOUNT IS NULL;

Part B:

Create table as per following.

STUDENT

RollNo	Name	Birthdate	SPI	City	Backlog	Branch
101	Keyur	5-1-92	8.5	Rajkot	2	CE
102	Hardik	15-2-94	9.0	Ahmedabad	0	CE
103	Kajal	14-3-96	10.00	Baroda	0	IT
104	Bhoomi	23-6-95	8.90	Ahmedabad	1	ICT
105	Harmit	15-2-94	9.80	Rajkot	1	IT
106	Jay	15-2-94	7.9	Rajkot	2	CE

From the above given tables perform the following SQL queries (SELECT Operation):

1. Give RollNo and Name of students, whose RollNo is greater than 103 and backlog is greater than 0 and branch is either CE or IT.
   SELECT ROLLNO, NAME FROM STUDENT WHERE ROLLNO > 103 AND BACKLOG > 0 AND BRANCH IN ('CE', 'IT');
2. Give name of students whose SPI is between 8 and 9 and branch is either CE or IT. (**OR & IN**)
   SELECT NAME FROM STUDENT WHERE SPI BETWEEN 8 AND 9 AND BRANCH IN ('CE', 'IT');
3. Find all students who do not belongs to 'CE' branch.
   SELECT * FROM STUDENT WHERE BRANCH != 'CE';
4. Display RollNo and Name of first three students.
   SELECT TOP 3 ROLLNO, NAME FROM STUDENT;
5. Display all the details of first three students whose SPI is greater than 8.5.
   SELECT TOP 3 * FROM STUDENT WHERE SPI > 8.5;
6. Retrieve all unique cities using DISTINCT.
   SELECT DISTINCT CITY FROM STUDENT;
7. Retrieve all unique branches using DISTINCT.
   SELECT DISTINCT BRANCH FROM STUDENT;
8. Retrieve all the records of student table as per their Backlog in descending order and then SPI in ascending order.
   SELECT * FROM STUDENT ORDER BY BACKLOG DESC, SPI ASC;
9. Update the branch and city of Jay to MCA and Jamangar respectively.
   UPDATE STUDENT SET BRANCH = 'MCA', CITY = 'JAMNAGAR' WHERE NAME = 'JAY';
10. Update the backlog of Keyur and Bhoomi to *NULL*.
    UPDATE STUDENT SET BACKLOG = NULL WHERE NAME IN ('KEYUR', 'BHOOMI');
11. Display the name of students whose backlog is *NULL* and backlog is greater than 1 and branch is either CE or IT.
    SELECT NAME FROM STUDENT WHERE (BACKLOG IS NULL OR BACKLOG > 1) AND BRANCH IN ('CE', 'IT');

1

2

Part A:

Create table as per following.

EMPLOYEE

EmpNo	EmpName	JoiningDate	Salary	City
101	Keyur	5-1-02	12000.00	Rajkot
102	Hardik	15-2-04	14000.00	Ahmedabad
103	Kajal	14-3-06	15000.00	Baroda
104	Bhoomi	23-6-05	12500.00	Ahmedabad
102	Harmit	15-2-04	14000.00	Rajkot

From the above given tables perform the following SQL queries (DELETE Operation):

1. Display the name of employee whose salary is greater than 13000 and city is either Rajkot or Baroda.
   SELECT EMPNAME FROM EMPLOYEES WHERE SALARY > 13000 AND CITY IN ('RAJKOT', 'BARODA');
2. Display the name of employee in ascending order by their name.
   SELECT EMPNAME FROM EMPLOYEES ORDER BY EMPNAME ASC;
3. Retrieve all unique cities.
   SELECT DISTINCT CITY FROM EMPLOYEES;
4. Update the city of Keyur and Bhoomi to *NULL*.
   UPDATE EMPLOYEES SET CITY = NULL WHERE EMPNAME IN ('KEYUR', 'BHOOMI');
5. Display the name of employee whose city is *NULL*.
   SELECT EMPNAME FROM EMPLOYEES WHERE CITY IS NULL;
6. Delete all the records of Employee table having salary greater than and equals to 14000.
   DELETE FROM EMPLOYEES WHERE SALARY >= 14000;
7. Delete all the Employees who belongs to 'RAJKOT' city.
   DELETE FROM EMPLOYEES WHERE CITY = 'RAJKOT';
8. Delete all the Employees who joined after 1-1-2007.
   DELETE FROM EMPLOYEES WHERE JOININGDATE > '2007-01-01';
9. Delete all the records of Employee table. (Use **Truncate**)
   TRUNCATE TABLE EMPLOYEES;
10. Remove Employee table. (Use **Drop**)
    DROP TABLE EMPLOYEES;
11. Delete all the records of DEPOSIT table. (Use **Truncate**)
    TRUNCATE TABLE DEPOSIT;
12. Remove DEPOSIT table. (Use **Drop**)
    DROP TABLE DEPOSIT;
13. Remove BRANCH table. (Use **Drop**)
    DROP TABLE BRANCH;
14. Remove CUSTOMERS table. (Use **Drop**)
    DROP TABLE CUSTOMERS;
15. Remove BORROW table. (Use **Drop**)
    DROP TABLE BORROW;

Part B:

Create table as per following.

ACCOUNT

ACTNO	CNAME	BNAME	AMOUNT	ADATE
101	ANIL	VRCE	1000.00	1-3-95
102	SUNIL	AJNI	5000.00	4-1-96
103	MEHUL	KAROLBAGH	3500.00	17-11-95
104	MADHURI	CHANDI	1200.00	17-12-95
105	PRMOD	M.G. ROAD	3000.00	27-3-96
106	SANDIP	ANDHERI	2000.00	31-3-96
107	SHIVANI	VIRAR	1000.00	5-9-95
108	KRANTI	NEHRU PLACE	5000.00	2-7-95
109	MINU	POWAI	7000.00	10-8-95

From the above given tables perform the following SQL queries:

1. Retrieve all unique BNAME.
   ```
   SELECT DISTINCT BNAME FROM ACCOUNT;
   ```
2. Display the Cname in ascending order by their amount and if amount is same then in descending order by cname.
   ```
   SELECT CNAME FROM ACCOUNT ORDER BY AMOUNT ASC, CNAME DESC;
   ```
3. Update the BNAME of Anil and Shivani to *NULL.*
   ```
   UPDATE ACCOUNT SET BNAME = NULL WHERE CNAME IN ('ANIL', 'SHIVANI');
   ```
4. Display the Cname of customers whose Bname is *NULL*.
   ```
   SELECT CNAME FROM ACCOUNT WHERE BNAME IS NULL;
   ```
5. Delete all the records of Account table having amount greater than and equals to 4000.
   ```
   DELETE FROM ACCOUNT WHERE AMOUNT >= 4000;
   ```
6. Delete all the accounts Bname is CHANDI.
   ```
   DELETE FROM ACCOUNT WHERE BNAME = 'CHANDI';
   ```
7. Delete all the accounts having adate after 1-10-1995.
   ```
   DELETE FROM ACCOUNT WHERE ADATE > '1995-10-01';
   ```
8. Delete all the records of Account table. (Use **Truncate**)
   ```
   TRUNCATE TABLE ACCOUNT;
   ```
9. Remove Account table. (Use **Drop**)
   ```
   DROP TABLE ACCOUNT;
   ```

Part C:

Create table as per following.

ACCOUNT

ACTNO	CNAME	BNAME	AMOUNT	ADATE
101	ANIL	VRCE	1000.00	1-3-95
102	SUNIL	AJNI	5000.00	4-1-96
103	MEHUL	KAROLBAGH	3500.00	17-11-95
104	MADHURI	CHANDI	1200.00	17-12-95
105	PRMOD	M.G. ROAD	3000.00	27-3-96
106	SANDIP	ANDHERI	2000.00	31-3-96
107	SHIVANI	VIRAR	1000.00	5-9-95
108	KRANTI	NEHRU PLACE	5000.00	2-7-95

109	MINU	POWAI	7000.00	10-8-95

From the above given tables perform the following SQL queries:

1. Display the Cname whose Bname is either AJNI or CHANDI and amount is greater than 3000 and sort the result in ascending order by their amount and if amount is same then in descending order by cname.
   SELECT CNAME FROM ACCOUNT WHERE BNAME IN ('AJNI', 'CHANDI') AND AMOUNT > 3000 ORDER BY AMOUNT ASC, CNAME DESC;
2. Retrieve top 3 unique BNAME and sort them in ascending order on BNAME.
   SELECT TOP 3 BNAME FROM (SELECT DISTINCT BNAME FROM ACCOUNT) AS UNIQUEBNAMES ORDER BY BNAME ASC;
3. Display the Cname whose ACTNO is greater than 103 and sort the result in ascending order by their amount and if amount is same then in descending order by cname.
   SELECT CNAME FROM ACCOUNT WHERE ACTNO > 103 ORDER BY AMOUNT ASC, CNAME DESC;
4. Update the BNAME of Anil, Mehul and Shivani to *NULL*.
   UPDATE ACCOUNT SET BNAME = NULL WHERE CNAME IN ('ANIL', 'MEHUL', 'SHIVANI');
5. Display the Cname of customers whose Bname is *NULL*.
   SELECT CNAME FROM ACCOUNT WHERE BNAME IS NULL;
6. Update the amount of Anil to 5000.
   UPDATE ACCOUNT SET AMOUNT = 5000 WHERE CNAME = 'ANIL';
7. Update amount of actno 109 to *NULL*.
   UPDATE ACCOUNT SET AMOUNT = NULL WHERE ACTNO = 109;
8. Retrieve all the records of account table as per their bname in descending order.
   SELECT * FROM ACCOUNT ORDER BY BNAME DESC;
9. Delete all the records of Account table. (Use **Truncate**)
   TRUNCATE TABLE TRANSACTIONS;
10. Remove Account table. (Use **Drop**)
    DROP TABLE TRANSACTIONS;

3

Part A:

Create table as per following.

STUDENT

StuID	FirstName	LastName	Website	City	Division
1011	Keyur	Patel	techonthenet.com	Rajkot	II-BCX
1022	Hardik	Shah	digminecraft.com	Ahmedabad	I-BCY
1033	Kajal	Trivedi	bigactivities.com	Baroda	IV-DCX
1044	Bhoomi	Gajera	checkyourmath.com	Ahmedabad	III-DCW
1055	Harmit	Mitel	NULL	Rajkot	II-BCY
1066	Ashok	Jani	NULL	Baroda	II-BCZ

From the above given tables perform the following SQL queries (LIKE Operation):

1. Display the name of students whose name starts with 'k'.
   SELECT FirstName, LastName FROM STUDENT WHERE FirstName LIKE 'K%';
2. Display the name of students whose name consists of five characters.
   SELECT FirstName, LastName FROM STUDENT WHERE FIRSTNAME LIKE '_____';
3. Retrieve the first name & last name of students whose city name ends with a & contains six characters.
   SELECT FIRSTNAME, LASTNAME,CITY FROM STUDENT WHERE CITY LIKE '_____A'

4. Display all the students whose last name ends with ‘tel’.
   SELECT * FROM STUDENT WHERE LASTNAME LIKE '%TEL';
5. Display all the students whose first name starts with ‘ha’ & ends with‘t’.
   SELECT * FROM STUDENT WHERE FIRSTNAME LIKE 'HA%T'
6. Display all the students whose first name starts with ‘k’ and third character is ‘y’.
   SELECT * FROM STUDENT WHERE FIRSTNAME LIKE 'K_Y%'
7. Display the name of students having no website and name consists of five characters.
   SELECT * FROM STUDENT WHERE WEBSITE IS NULL AND FIRSTNAME LIKE '_____';
8. Display all the students whose last name consist of ‘jer’.
   SELECT * FROM STUDENT WHERE LASTNAME LIKE '%JER%'
9. Display all the students whose city name starts with either ‘r’ or ‘b’.
   SELECT FIRSTNAME, LASTNAME FROM STUDENT WHERE CITY LIKE '[R,B]%';
10. Display all the name students having websites.
    SELECT * FROM STUDENT WHERE WEBSITE IS NOT NULL;
11. Display all the students whose name starts from alphabet A to H.
    SELECT * FROM STUDENT WHERE FIRSTNAME LIKE '[A-H]%’;
12. Display all the students whose name’s second character is vowel.
    SELECT * FROM STUDENT WHERE FIRSTNAME LIKE '_[A,E,I,O,U]%'
13. Display student’s name whose city name consist of ‘rod’.
    SELECT * FROM STUDENT WHERE CITY LIKE '%ROD%'
14. Retrieve the First & Last Name of students whose website name starts with ‘bi’
    SELECT * FROM STUDENT WHERE WEBSITE LIKE 'BI%';
15. Display student’s city whose last name consists of six characters.
    SELECT * FROM STUDENT WHERE LASTNAME LIKE '______'
16. Display all the students whose city name consist of five characters & not starts with ‘ba’.
    SELECT * FROM STUDENT WHERE CITY LIKE '_____' AND CITY NOT LIKE 'BA%';
17. Show all the student’s whose division starts with ‘II’.
    SELECT * FROM STUDENT WHERE DIVISION LIKE 'II%'
18. Find out student’s first name whose division contains ‘bc’ anywhere in division name.
    SELECT * FROM STUDENT WHERE DIVISION LIKE '%BC%';
19. Show student id and city name in which division consist of six characters and having website name.
    SELECT STUID, CITY FROM STUDENT WHERE DIVISION LIKE '______' AND WEBSITE IS NOT NULL;
20. Display all the students whose name’s third character is consonant.
    SELECT * FROM STUDENT WHERE FIRSTNAME NOT LIKE '__[A,E,I,O,U]%'

Part B:

Create table as per following.

CUSTOMER

CID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitucion 2222	Mexico D.F.	05021	Mexico
3	Antonio Moreno Taqueria	Antonio Moreno	Mataderos 2312	Mexico D.F.	05023	Mexico

4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbkop	Christina Berglund	Berguvsvagen 8	Lulea	S-958 22	Sweden

From the above given tables perform the following SQL queries (LIKE Operation):

1. Return all customers from a city that starts with 'L' followed by one wildcard character, then 'nd' and then two wildcard characters.
   SELECT * FROM CUSTOMER WHERE CITY LIKE 'L_ND__%';
2. Return all customers from a city that contains the letter 'L'.
   SELECT * FROM CUSTOMER WHERE CITY LIKE '%L%';
3. Return all customers from a city that do not contains the letter 'L'.
   SELECT * FROM CUSTOMER WHERE CITY NOT LIKE '%L%';
4. Return all customers that starts with 'La'.
   SELECT * FROM CUSTOMER WHERE CUSTOMERNAME LIKE 'LA%';
5. Return all customers that do not starts with 'La'
   SELECT * FROM CUSTOMER WHERE CUSTOMERNAME NOT LIKE 'LA%';
6. Return all customers that starts with 'a' or starts with 'b'.
   SELECT * FROM CUSTOMER WHERE CUSTOMERNAME LIKE 'A%' OR CUSTOMERNAME LIKE 'B%';
7. Return all customers that starts with 'a' or starts with 'c' or starts with 't'.
   SELECT * FROM CUSTOMER WHERE CUSTOMERNAME LIKE 'A%' OR CUSTOMERNAME LIKE 'C%' OR CUSTOMERNAME LIKE 'T%';
8. Return all customers that starts with 'a' to 'd'.
   SELECT * FROM CUSTOMER WHERE CUSTOMERNAME LIKE '[A-D]%';
9. Return all customers that ends with 'a'.
   SELECT * FROM CUSTOMER WHERE CUSTOMERNAME LIKE '%A';
10. Return all customers that do not ends with 'a'.
    SELECT * FROM CUSTOMER WHERE CUSTOMERNAME NOT LIKE '%A';
11. Return all customers that starts with 'b' and ends with 's'.
    SELECT * FROM CUSTOMER WHERE CUSTOMERNAME LIKE 'B%S';
12. Return all customers that contains the phrase 'or'.
    SELECT * FROM CUSTOMER WHERE CUSTOMERNAME LIKE '%OR%';
13. Return all customers that starts with "a" and are at least 3 characters in length.
    SELECT * FROM CUSTOMER WHERE CUSTOMERNAME LIKE 'A__%';
14. Return all customers that have "r" in the second position.
    SELECT * FROM CUSTOMER WHERE CUSTOMERNAME LIKE '_R%';
15. Return all customers from Spain.
    SELECT * FROM CUSTOMER WHERE COUNTRY = 'SPAIN';

Part C:

Create table as per following.

CUSTOMER

CID	Name	Age	Address	Salary
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00

4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	Hyderabad	4500.00
7	Muffy	24	Indore	10000.00

From the above given tables perform the following SQL queries (LIKE Operation):

1. Display all the records from the CUSTOMERS table, where the SALARY starts with 200.
   SELECT * FROM CUSTOMER WHERE SALARY LIKE '200%';
2. Displays all the records from the CUSTOMERS table with the NAME that has 'al' in any position.
   SELECT * FROM CUSTOMER WHERE NAME LIKE '%AL%';
3. Display all the records from the CUSTOMERS table where the Name starts with K and is at least 4 characters in length.
   SELECT * FROM CUSTOMER WHERE NAME LIKE 'K___%';
4. Display all the records from the CUSTOMERS table, where the NAME has 'm' in the third position.
   SELECT * FROM CUSTOMER WHERE NAME LIKE '__M%';
5. Retrieves the records of the customers whose name starts with C and ends with i, or customers whose name ends with k.
   SELECT * FROM CUSTOMER WHERE NAME LIKE 'C%I' OR NAME LIKE '%K';
6. Retrieves all the customers whose name does not start with K.
   SELECT * FROM CUSTOMER WHERE NAME NOT LIKE 'K%';

4

Part A:

Create table as per following.

EMPLOYEE

EID	EName	Department	Salary	JoiningDate	City
101	Rahul	Admin	56000	1-Jan-90	Rajkot
102	Hardik	IT	18000	25-Sep-90	Ahmedabad
103	Bhavin	HR	25000	14-May-91	Baroda
104	Bhoomi	Admin	39000	8-Feb-91	Rajkot
105	Rohit	IT	17000	23-Jul-90	Jamnagar
106	Priya	IT	9000	18-Oct-90	Ahmedabad
107	Neha	HR	34000	25-Dec-91	Rajkot

From the above given tables perform the following SQL queries:

1. Display the Highest, Lowest, Total, and Average salary of all employees. Label the columns Maximum, Minimum, Total_Sal and Average_Sal, respectively.
   SELECT MAX(SALARY) AS MAXIMUM, MIN(SALARY) AS MINIMUM, SUM(SALARY) AS TOTAL_SAL, AVG(SALARY) AS AVERAGE_SAL FROM EMPLOYEE;
2. Find total number of employees of EMPLOYEE table.
   SELECT COUNT (*) AS TOTAL_EMPLOYEES FROM EMPLOYEE;
3. Give maximum salary from IT department.
   SELECT MAX(SALARY) AS MAX_IT_SALARY FROM EMPLOYEE WHERE DEPARTMENT = 'IT';
4. Count total number of cities of employee without duplication.
   SELECT COUNT (DISTINCT CITY) AS TOTAL_CITIES FROM EMPLOYEE;
5. Display city with the total number of employees belonging to each city.
   SELECT CITY, COUNT (*) AS NUMBER_OF_EMPLOYEES FROM EMPLOYEE

```
GROUP BY CITY;
```

6. Display city having more than one employee.
```
SELECT CITY FROM EMPLOYEE
GROUP BY CITY
HAVING COUNT (*) > 1;
```
7. Give total salary of each department of EMPLOYEE table.
```
SELECT DEPARTMENT, SUM(SALARY) AS TOTAL_SALARY FROM EMPLOYEE
GROUP BY DEPARTMENT;
```
8. Give average salary of each department of EMPLOYEE table without displaying the respective department name.
```
SELECT AVG(SALARY) AS AVERAGE_SALARY FROM EMPLOYEE
GROUP BY DEPARTMENT;
```
9. Display minimum salary of employee who belongs to Ahmedabad.
```
SELECT MIN(SALARY) AS MIN_SALARY_AHMEDABAD FROM EMPLOYEE WHERE CITY = 'AHMEDABAD';
```
10. List the departments having total salaries more than 50000 and located in city Rajkot.
```
SELECT DEPARTMENT FROM EMPLOYEE
WHERE CITY = 'RAJKOT'
GROUP BY DEPARTMENT
HAVING SUM(SALARY) > 50000;
```
11. Count the number of employees living in Rajkot.
```
SELECT COUNT (*) AS EMPLOYEES_RAJKOT FROM EMPLOYEE WHERE CITY = 'RAJKOT';
```
12. Display the difference between the highest and lowest salaries. Label the column DIFFERENCE.
```
SELECT MAX(SALARY) - MIN(SALARY) AS DIFFERENCE FROM EMPLOYEE;
```
13. Display the total number of employees hired before 1st January, 1991.
```
SELECT COUNT (*) AS EMPLOYEES_HIRED_BEFORE_1991 FROM EMPLOYEE
WHERE JOININGDATE  < '1991-01-01';
```
14. Display total salary of each department with total salary exceeding 35000 and sort the list by total salary.
```
SELECT DEPARTMENT, SUM(SALARY) AS TOTAL_SALARY FROM EMPLOYEE
GROUP BY DEPARTMENT
HAVING SUM(SALARY) > 35000
ORDER BY TOTAL_SALARY;
```
15. List out department names in which more than two employees.
```
SELECT DEPARTMENT FROM EMPLOYEE
GROUP BY DEPARTMENT
HAVING COUNT (*) > 2;
```

Part B:

Create table as per following.

COMPANY

Title	Company	Type	Production_year	System	Production_cost	Revenue	Rating
Blasting Boxes	Simone Games	action adventure	1998	PC	100000	200000	7

Run Run Run!	13 Mad Bits	shooter	2011	PS3	3500000	650000	3
Duck n'Go	13 Mad Bits	shooter	2012	Xbox	3000000	1500000	5
SQL Wars!	Vertabelo	wargames	2017	Xbox	5000000	25000000	10
Tap Tap Hex!	PixelGaming Inc.	rhythm	2006	PS2	2500000	3500000	7
NoRisk	Simone Games	action adventure	2004	PS2	1400000	3400000	8

From the above given tables perform the following SQL queries:

1. Display the name and total revenue for each company.
   ```
   SELECT COMPANY, SUM(REVENUE) AS TOTAL_REVENUE FROM COMPANY
   GROUP BY COMPANY;
   ```
2. Generate a report with the production year and the number of games released this year (named count), the average of production cost for all games produced in this year (named avg_cost) and the average revenue for that year (named avg_revenue).
   ```
   SELECT PRODUCTION_YEAR,  COUNT(*) AS COUNT, AVG(PRODUCTION_COST) AS AVG_COST,
   AVG(REVENUE)  AS AVG_REVENUE FROM COMPANY
   WHERE PRODUCTION_YEAR = YEAR(GETDATE())
   GROUP BY PRODUCTION_YEAR
   ```
3. Count how many games of a given type are profitable (i.e. the revenue was greater than the production cost). Show the game type and the number of profitable games (named number_of_games) for each type.
   ```
   SELECT TYPE, COUNT(*) AS NUMBER_OF_GAMES FROM COMPANY
   WHERE REVENUE  > PRODUCTION_COST
   GROUP BY TYPE;
   ```
4. Obtain the type of games and the total revenue generated for games with a production_year after 2010 and with a PS2 or PS3 system. Order the result so the types with the highest revenue come first.
   ```
   SELECT TYPE, SUM(REVENUE) AS TOTAL_REVENUE FROM COMPANY
   WHERE PRODUCTION_YEAR > 2010 AND SYSTEM IN ('PS2', 'PS3')
   GROUP BY TYPE
   ORDER BY TOTAL_REVENUE DESC;
   ```
5. For all companies present in the table, obtain their names and the sum of gross profit over all years. (Assume that gross profit = revenue - cost of production). Name this column gross_profit_sum. Order the results by gross profit, in descending order.
   ```
   SELECT COMPANY, SUM(REVENUE - PRODUCTION_COST) AS GROSS_PROFIT_SUM
   FROM COMPANY
   GROUP BY COMPANY
   ORDER BY GROSS_PROFIT_SUM DESC;
   ```
6. Obtain the yearly gross profit of each company. In other words, we want a report with the company name, the year, and the gross profit for that year. Order the report by company name and year.
   ```
   SELECT COMPANY, PRODUCTION_YEAR,  SUM(REVENUE - PRODUCTION_COST) AS YEARLY _
   GROSS_PROFIT FROM COMPANY
   ```

```
GROUP BY COMPANY, PRODUCTION_YEAR
ORDER BY COMPANY, PRODUCTION_YEAR;
```

7. For each company, select its name, the number of games it's produced (as the number_of_games column), and the average cost of production (as the avg_cost column). Show only companies producing more than one game.

```
SELECT COMPANY, COUNT(*) AS NUMBER_OF_GAMES,AVG(PRODUCTION_COST) AS AVG_COST
FROM COMPANY
GROUP BY COMPANY
HAVING COUNT(*) > 1;
```

5

Part A:

Create table as per following.

ORDERS

ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.5	05-10-2012	3005	5002
70009	270.65	10-09-2012	3001	5005
70002	65.26	05-10-2012	3002	5001
70004	110.5	17-08-2012	3009	5003
70007	948.5	10-09-2012	3005	5002
70005	2400.6	27-07-2012	3007	5001
70008	5760	10-09-2012	3002	5001
70010	1983.43	10-10-2012	3004	5006
70003	2480.4	10-10-2012	3009	5003
70012	250.45	27-06-2012	3008	5002
70011	75.29	17-08-2012	3003	5007
70013	3045.6	25-04-2012	3002	5001
70001	150.5	05-10-2012	3005	5002
70009	270.65	10-09-2012	3001	5005
70002	65.26	05-10-2012	3002	5001

CUSTOMER

customer_id	cust_name	city	Grade	salesman_id
3002	Nick Rimando	New York	100	5001
3007	Brad Davis	New York	200	5001
3005	Graham Zusi	California	200	5002
3008	Julian Green	London	300	5002
3004	Fabian Johnson	Paris	300	5006
3009	Geoff Cameron	Berlin	100	5003
3003	Jozy Altidor	Moscow	200	5007
3001	Brad Guzan	London		5005

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
5007	Paul Adam	Rome	0.13
5003	Lauson Hen	San Jose	0.12

From the above given tables perform the following SQL queries:

1. Write a SQL query to calculate total purchase amount of all orders. Return total purchase amount.
   SELECT SUM(PURCH_AMT) AS TOTAL_PURCHASE_AMOUNT FROM ORDERS;
2. Write a SQL query to calculate the average purchase amount of all orders. Return average purchase amount. SELECT AVG(PURCH_AMT) AS AVERAGE_PURCHASE_AMOUNT FROM ORDERS;
3. Write a SQL query that counts the number of unique salespeople. Return number of salespeople.
   SELECT COUNT(SALESMAN_ID) AS NUMBER_OF_SALESPEOPLE FROM SALESMEN;
4. Write a SQL query to count the number of customers. Return number of customers.
   SELECT COUNT(CUSTOMER_ID) AS NUMBER_OF_CUSTOMERS FROM CUSTOMERS;
5. Write a SQL query to determine the number of customers who received at least one grade for their activity.
   SELECT COUNT(CUSTOMER_ID) AS NUMBER_OF_CUSTOMERS_WITH_GRADE FROM CUSTOMERS
   WHERE GRADE IS NOT NULL;
6. Write a SQL query to find the maximum purchase amount.
   SELECT MAX(PURCH_AMT) AS MAXIMUM_PURCHASE_AMOUNT FROM ORDERS;
7. Write a SQL query to find the minimum purchase amount.
   SELECT MIN(PURCH_AMT) AS MINIMUM_PURCHASE_AMOUNT FROM ORDERS;
8. Write a SQL query to find the highest grade of the customers in each city. Return city, maximum grade.
   SELECT CITY, MAX(GRADE) AS MAXIMUM_GRADE FROM CUSTOMERS GROUP BY CITY;
9. Write a SQL query to find the highest purchase amount ordered by each customer. Return customer ID, maximum purchase amount.
   SELECT CUSTOMER_ID, MAX(PURCH_AMT) AS MAXIMUM_PURCHASE_AMOUNT FROM ORDERS
   GROUP BY CUSTOMER_ID;
10. Write a SQL query to find the highest purchase amount ordered by each customer on a particular date. Return, order date and highest purchase amount.
    SELECT ORD_DATE, MAX(PURCH_AMT) AS HIGHEST_PURCHASE_AMOUNT FROM ORDERS
    GROUP BY ORD_DATE;

Part B:

1. Write a SQL query to determine the highest purchase amount made by each salesperson on '2012-08-17'. Return salesperson ID, purchase amount.
   SELECT SALESMAN_ID, MAX(PURCH_AMT) AS PURCHASE_AMOUNT FROM ORDERS
   WHERE ORD_DATE = '2012-08-17'
   GROUP BY SALESMAN_ID;

2. Write a SQL query to find the highest order (purchase) amount by each customer on a particular order date. Filter the result by highest order (purchase) amount above 2000.00. Return customer id, order date and maximum purchase amount.

```
SELECT CUSTOMER_ID, ORD_DATE, MAX(PURCH_AMT) AS MAXIMUM_PURCHASE_AMOUNT
FROM ORDERS
GROUP BY CUSTOMER_ID, ORD_DATE
HAVING MAX(PURCH_AMT) > 2000.00;
```

3. Write a SQL query to find the maximum order (purchase) amount in the range 2000 - 6000 (Begin and end values are included.) by combination of each customer and order date. Return customer id, order date and maximum purchase amount.

```
SELECT CUSTOMER_ID, ORD_DATE, MAX(PURCH_AMT) AS MAXIMUM_PURCHASE_AMOUNT
FROM ORDERS
WHERE PURCH_AMT BETWEEN 2000 AND 6000
GROUP BY CUSTOMER_ID, ORD_DATE;
```

4. Filter the rows for maximum order (purchase) amount is either 2000, 3000, 5760, 6000. Return customer id, order date and maximum purchase amount.

```
SELECT CUSTOMER_ID, ORD_DATE, MAX(PURCH_AMT) AS MAXIMUM_PURCHASE_AMOUNT
FROM ORDERS
WHERE PURCH_AMT IN (2000, 3000, 5760, 6000)
GROUP BY CUSTOMER_ID, ORD_DATE;
```

5. Write a SQL query to determine the maximum order amount for each customer. The customer ID should be in the range 3002 and 3007(Begin and end values are included.). Return customer id and maximum purchase amount.

```
SELECT CUSTOMER_ID, MAX(PURCH_AMT) AS MAXIMUM_PURCHASE_AMOUNT FROM ORDERS
WHERE CUSTOMER_ID BETWEEN 3002 AND 3007
GROUP BY CUSTOMER_ID;
```

6. Write a SQL query to find the maximum order (purchase) amount for each customer. The customer ID should be in the range 3002 and 3007(Begin and end values are included.). Filter the rows for maximum order (purchase) amount is higher than 1000. Return customer id and maximum purchase amount.

```
SELECT CUSTOMER_ID, MAX(PURCH_AMT) AS MAXIMUM_PURCHASE_AMOUNT FROM ORDERS
WHERE CUSTOMER_ID BETWEEN 3002 AND 3007
GROUP BY CUSTOMER_ID
HAVING MAX(PURCH_AMT) > 1000;
```

7. Write a SQL query to determine the maximum order (purchase) amount generated by each salesperson. Filter the rows for the salesperson ID is in the range 5003 and 5008 (Begin and end values are included.). Return salesperson id and maximum purchase amount.

```
SELECT SALESMAN_ID, MAX(PURCH_AMT) AS MAXIMUM_PURCHASE_AMOUNT FROM ORDERS
WHERE SALESMAN_ID BETWEEN 5003 AND 5008
GROUP BY SALESMAN_ID;
```

8. Write a SQL query to count all the orders generated on '2012-08-17'. Return number of orders.

```
SELECT COUNT(*) AS NUMBER_OF_ORDERS FROM ORDERS WHERE ORD_DATE = '2012-08-17';
```

9. Write a SQL query to count the number of salespeople in a city. Return number of salespeople.

```
SELECT CITY,  COUNT(salesman_id) AS NUMBER_OF_SALESPEOPLE FROM SALESMEN
GROUP BY CITY;
```

10. Write a SQL query to count the number of orders based on the combination of each order date and salesperson. Return order date, salesperson id.

```
SELECT ORD_DATE, SALESMAN_ID, COUNT(*) AS NUMBER_OF_ORDERS FROM ORDERS
GROUP BY ORD_DATE, SALESMAN_ID;
```

Part C:

1. Write a SQL query to calculate the average product price. Return average product price.

```
SELECT AVG(PRICE) AS AVERAGE_PRODUCT_PRICE FROM PRODUCTS;
```

2. Write a SQL query to count the number of products whose price are higher than or equal to 350. Return number of products.

```
SELECT COUNT(*) AS NUMBER_OF_PRODUCTS FROM PRODUCTS WHERE PRICE >= 350;
```

3. Write a SQL query to compute the average price for unique companies. Return average price and company id.

```
SELECT COMPANY_ID, AVG(PRICE) AS AVERAGE_PRICE FROM PRODUCTS
GROUP BY COMPANY_ID;
```

4. Write a SQL query to compute the sum of the allotment amount of all departments. Return sum of the allotment amount.

```
SELECT SUM(ALLOTMENT_AMOUNT) AS TOTAL_ALLOTMENT_AMOUNT FROM DEPARTMENTS;
```

5. Write a SQL query to count the number of employees in each department. Return department code and number of employees.

```
SELECT DEPARTMENT_CODE, COUNT(*) AS NUMBER_OF_EMPLOYEES FROM EMPLOYEES
GROUP BY DEPARTMENT_CODE;
```

6

Part-A:

Create table as per following.

STUDENT

Rno	Name	Branch
101	Raju	CE
102	Amit	CE
103	Sanjay	ME
104	Neha	EC
105	Meera	EE
106	Mahesh	ME

RESULT

Rno	SPI
101	8.8
102	9.2
103	7.6
104	8.2
105	7.0
107	8.9

EMPLOYEE

EmployeeNo	Name	ManagerNo
E01	Tarun	*NULL*
E02	Rohan	E02
E03	Priya	E01
E04	Milan	E03
E05	Jay	E01
E06	Anjana	E04

From the above given tables perform the following queries (Join):

1. Combine information from student and result table using cross join or Cartesian product.

```
SELECT S.RNO, S.NAME, S.BRANCH, R.RNO, R.SPI
FROM STUDENT S
CROSS JOIN RESULT R;
```

2. Display Rno, Name, Branch and SPI of all students.

```
SELECT S.RNO, S.NAME, S.BRANCH, R.SPI
FROM STUDENT S
INNER JOIN RESULT R  ON S.RNO = R.RNO;
```

3. Display Rno, Name, Branch and SPI of CE branch's student only.

```
SELECT S.RNO, S.NAME, S.BRANCH, R.SPI
FROM STUDENT S
```

```
INNER JOIN RESULT R  ON S.RNO = R.RNO
WHERE S.BRANCH='CE';
```

4. Display Rno, Name, Branch and SPI of other than EC branch’s student only.

```
SELECT S.RNO, S.NAME, S.BRANCH, R.SPI
FROM STUDENT S
INNER JOIN RESULT R ON S.RNO = R.RNO
WHERE S.BRANCH!='EC';
```

5. Display average result of each branch.

```
SELECT S.BRANCH,AVG(R.SPI) AS AVERAGESPI
FROM STUDENT S
INNER JOIN RESULT R ON S.RNO = R.RNO
GROUP BY S.BRANCH
```

6. Display average result of each branch and sort them in ascending order by SPI.

```
SELECT S.BRANCH,AVG(R.SPI) AS AVERAGESPI
FROM STUDENT S
INNER JOIN RESULT R ON S.RNO = R.RNO
GROUP BY S.BRANCH
ORDER BY AVG(R.SPI)
```

7. Display average result of CE and ME branch.

```
SELECT S.BRANCH,AVG(R.SPI) AS AVERAGESPI
FROM STUDENT S
INNER JOIN RESULT R ON S.RNO = R.RNO
WHERE S.BRANCH IN('CE','ME')
GROUP BY S.BRANCH
```

8. Perform the left outer join on Student and Result tables.

```
SELECT  S.RNO, S.NAME, S.BRANCH, R.SPI
FROM STUDENT S
LEFT JOIN RESULT R ON S.RNO = R.RNO
```

9. Perform the right outer join on Student and Result tables.

```
SELECT  S.RNO, S.NAME, S.BRANCH, R.SPI
FROM STUDENT S
RIGHT JOIN RESULT R ON S.RNO = R.RNO
```

10. Perform the full outer join on Student and Result tables.

```
SELECT  S.RNO, S.NAME, S.BRANCH, R.SPI
FROM STUDENT S
FULL JOIN RESULT R ON S.RNO = R.RNO
```

11. Retrieve the names of employee along with their manager name from the Employee table.

```
SELECT E1.NAME AS EMPLOYEENAME, E2.NAME AS MANAGERNAME
FROM EMPLOYEE E1
INNERJOIN EMPLOYEE E2 ON E1.MANAGERNO = E2.EMPLOYEENO;
```

Part-B:

Create table as per following.

DEPARTMENT

DepartmentID	DepartmentName	DepartmentCode	Location
1	Admin	Admin	A-Block
2	Computer	CE	C-Block
3	Civil	CI	G-Block
4	Electrical	EE	E-Block
5	Mechanical	ME	B-Block

PERSON

PersonID	PersonName	DepartmentID	Salary	JoiningDate	City
101	Rahul Tripathi	2	56000	01-01-2000	Rajkot
102	Hardik Pandya	3	18000	25-09-2001	Ahmedabad
103	Bhavin Kanani	4	25000	14-05-2000	Baroda
104	Bhoomi Vaishnav	1	39000	08-02-2005	Rajkot
105	Rohit Topiya	2	17000	23-07-2001	Jamnagar
106	Priya Menpara	NULL	9000	18-10-2000	Ahmedabad
107	Neha Sharma	2	34000	25-12-2002	Rajkot
108	Nayan Goswami	3	25000	01-07-2001	Rajkot
109	Mehul Bhundiya	4	13500	09-01-2005	Baroda
110	Mohit Maru	5	14000	25-05-2000	Jamnagar

From the above given table perform the following SQL queries (Join & Group By):

1. Find all persons with their department name & code.
   ```
   SELECT P.PERSONID, P.PERSONNAME, D.DEPARTMENTNAME, D.DEPARTMENTCODE
   FROM PERSON P
   INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID;
   ```
2. Give department wise maximum & minimum salary with department name.
   ```
   SELECT D.DEPARTMENTNAME, MAX(P.SALARY) AS MAXSALARY, MIN(P.SALARY) AS MINSALARY
   FROM PERSON P
   INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID
   GROUP BY D.DEPARTMENTNAME;
   ```
3. Find all departments whose total salary is exceeding 100000.
   ```
   SELECT D.DEPARTMENTNAME, SUM(P.SALARY) AS TOTALSALARY
   FROM PERSON P
   INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID
   GROUP BY D.DEPARTMENTNAME
   HAVING SUM(P.SALARY) > 100000;
   ```
4. Retrieve person name, salary & department name who belongs to Jamnagar city.
   ```
   SELECT P.PERSONNAME,P.SALARY, D.DEPARTMENTNAME
   FROM PERSON P
   INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID
   WHERE P.CITY='JAMNAGAR';
   ```
5. Find all persons who does not belongs to any department.
   ```
   SELECT * FROM PERSON
   ```

```
WHERE DEPARTMENTID IS NULL
```

6. Find department wise person counts.
   ```
   SELECT D.DEPARTMENTNAME,COUNT(P.PERSONID)
   FROM PERSON P
   INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID
   GROUP BY D.DEPARTMENTNAME
   ```
7. Find average salary of person who belongs to Ahmedabad city.
   ```
   SELECT AVG(SALARY)
   FROM PERSON
   WHERE CITY='AHMEDABAD'
   ```
8. Produce Output Like: <PersonName> earns <Salary> from department <DepartmentName> monthly. (In Single Column)
   ```
   SELECT P.PERSONNAME + ' EARNS ' + CAST(P.SALARY AS VARCHAR) + ' FROM DEPARTMENT ' +
   D.DEPARTMENTNAME + ' MONTHLY.' AS OUTPUT
   FROM PERSON P
   INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID;
   ```
9. List all departments who have no persons.
   ```
   SELECT D.DepartmentName
   FROM DEPARTMENT D
   LEFT JOIN PERSON P ON D.DepartmentID = P.DepartmentID
   WHERE P.PersonID IS NULL;
   ```
10. Find city & department wise total, average & maximum salaries.
    ```
    SELECT P.CITY, D.DEPARTMENTNAME, SUM(P.SALARY) AS TOTALSALARY, AVG(P.SALARY) AS
    AVERAGESALARY, MAX(P.SALARY) AS MAXSALARY FROM PERSON P
    INNER JOIN DEPARTMENT D
    ON P.DEPARTMENTID = D.DEPARTMENTID
    GROUP BY P.CITY, D.DEPARTMENTNAME;
    ```

Part – C:

1. Display Unique city names.
   ```
   SELECT DISTINCT CITY FROM PERSON;
   ```
2. List out department names in which more than two persons.
   ```
   SELECT D.DEPARTMENTNAME
   FROM PERSON P
   INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID
   GROUP BY D.DEPARTMENTNAME
   HAVING COUNT(P.PERSONID) > 2;
   ```
3. Combine person name's first three characters with city name's last three characters in single column.
   ```
   SELECT LEFT(P.PERSONNAME,3) + RIGHT(P.CITY,3) FROM PERSON P;
   ```
4. Give 10% increment in Computer department employee's salary.
   ```
   UPDATE P
   SET P.SALARY = P.SALARY * 1.10
   FROM PERSON P
   INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID
   ```

```
WHERE D.DEPARTMENTNAME = 'COMPUTER';
```

5. Display all the person name's who's joining dates difference with current date is more than 365 days.

```
SELECT PERSONNAME FROM PERSON
WHERE DATEDIFF(DAY, JOININGDATE, GETDATE()) > 365;
```

7

Part – A:

Create Database with Name: **Person_Info**
Create following table under Person_Info database**. (Using Design Mode)**

PERSON

Column_Name	DataType	Constraints
PersonID	Int	Primary Key
PersonName	Varchar (100)	Not Null
DepartmentID	Int	Foreign Key, Null
Salary	Decimal (8,2)	Not Null
JoiningDate	Datetime	Not Null
City	Varchar (100)	Not Null

DEPARTMENT

Column_Name	DataType	Constraints
DepartmentID	Int	Primary Key
DepartmentName	Varchar (100)	Not Null, Unique
DepartmentCode	Varchar (50)	Not Null, Unique
Location	Varchar (50)	Not Null

PersonID	PersonName	DepartmentID	Salary	JoiningDate	City
101	Rahul Tripathi	2	56000	01-01-2000	Rajkot
102	Hardik Pandya	3	18000	25-09-2001	Ahmedabad
103	Bhavin Kanani	4	25000	14-05-2000	Baroda
104	Bhoomi Vaishnav	1	39000	08-02-2005	Rajkot
105	Rohit Topiya	2	17000	23-07-2001	Jamnagar
106	Priya Menpara	NULL	9000	18-10-2000	Ahmedabad
107	Neha Sharma	2	34000	25-12-2002	Rajkot
108	Nayan Goswami	3	25000	01-07-2001	Rajkot
109	Mehul Bhundiya	4	13500	09-01-2005	Baroda
110	Mohit Maru	5	14000	25-05-2000	Jamnagar

DepartmentID	DepartmentName	DepartmentCode	Location
1	Admin	Adm	A-Block
2	Computer	CE	C-Block
3	Civil	CI	G-Block
4	Electrical	EE	E-Block
5	Mechanical	ME	B-Block

Part – B:

From the above given table perform the following SQL queries (Join):

1. Find all persons with their department name & code.

```
SELECT P.PERSONNAME, D.DEPARTMENTNAME, D.DEPARTMENTCODE
FROM PERSON P
```

```
INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID;
```

2. Find the person's name whose department is located in C-Block.
   ```
   SELECT P.PERSONNAME
   FROM PERSON P
   INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID
   WHERE D.LOCATION = 'C-BLOCK';
   ```
3. Retrieve person name, salary & department name who belongs to Jamnagar city.
   ```
   SELECT P.PERSONNAME, P.SALARY, D.DEPARTMENTNAME
   FROM PERSON P
   INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID
   WHERE P.CITY = 'JAMNAGAR';
   ```
4. Retrieve person name, salary & department name who does not belong to Rajkot city.
   ```
   SELECT P.PERSONNAME, P.SALARY, D.DEPARTMENTNAME
   FROM PERSON P
   INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID
   WHERE P.CITY <> 'RAJKOT';
   ```
5. Retrieve person’s name of the person who joined the Civil department after 1-Aug-2001.
   ```
   SELECT P.PERSONNAME
   FROM PERSON P
   INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID
   WHERE D.DEPARTMENTNAME = 'CIVIL' AND P.JOININGDATE > '2001-08-01';
   ```
6. Find details of all persons who belong to the Computer department.
   ```
   SELECT P.PERSONNAME, P.SALARY, P.CITY, P.JOININGDATE
   FROM PERSON P
   INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID
   WHERE D.DEPARTMENTNAME = 'COMPUTER';
   ```
7. Display all the person's name with the department whose joining date difference with the current date is more than 365 days.
   ```
   SELECT P.PERSONNAME, D.DEPARTMENTNAME
   FROM PERSON P
   INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID
   WHERE DATEDIFF(DAY, P.JOININGDATE, GETDATE()) > 365;
   ```
8. Find department wise person counts.
   ```
   SELECT D.DEPARTMENTNAME, COUNT(P.PERSONID) AS PERSONCOUNT
   FROM PERSON P
   INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID
   GROUP BY D.DEPARTMENTNAME;
   ```
9. Give department wise maximum & minimum salary with department name.
   ```
   SELECT D.DEPARTMENTNAME, MAX(P.SALARY) AS MAXSALARY, MIN(P.SALARY) AS MINSALARY
   FROM PERSON P
   INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID
   GROUP BY D.DEPARTMENTNAME;
   ```

10. Find city wise total, average, maximum and minimum salary.

```
SELECT P.CITY, SUM(P.SALARY) AS TOTALSALARY, AVG(P.SALARY) AS AVERAGESALARY,
MAX(P.SALARY) AS MAXSALARY, MIN(P.SALARY) AS MINSALARY FROM PERSON P
GROUP BY P.CITY;
```

11. Produce Output Like: <PersonName> lives in <City> and works in <DepartmentName> Department. (In single column)

```
SELECT P.PERSONNAME + ' LIVES IN ' + P.CITY + ' AND WORKS IN ' + D.DEPARTMENTNAME + '
DEPARTMENT.'
FROM PERSON P
INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID;
```

12. Produce Output Like: <PersonName> earns <Salary> from <DepartmentName> department monthly. (In single column)

```
SELECT P.PERSONNAME + ' EARNS ' + CAST(P.SALARY AS VARCHAR) + ' FROM ' +
D.DEPARTMENTNAME + ' DEPARTMENT MONTHLY.' AS OUTPUT
FROM PERSON P
INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID;
```

13. Find city & department wise total, average & maximum salaries.

```
SELECT P.CITY, D.DEPARTMENTNAME, SUM(P.SALARY) AS TOTALSALARY, AVG(P.SALARY) AS
AVERAGESALARY, MAX(P.SALARY) AS MAXSALARY FROM PERSON P
INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID
GROUP BY P.CITY, D.DEPARTMENTNAME;
```

14. Find all persons who do not belong to any department.

```
SELECT P.PERSONNAME
FROM PERSON P
WHERE P.DEPARTMENTID IS NULL;
```

Part – C:

1. Find all departments whose total salary is exceeding 100000.

```
SELECT D.DEPARTMENTNAME
FROM DEPARTMENT D
INNER JOIN PERSON P ON D.DEPARTMENTID = P.DEPARTMENTID
GROUP BY D.DEPARTMENTNAME
HAVING SUM(P.SALARY) > 100000;
```

2. List all departments who have no person.

```
SELECT D.DEPARTMENTNAME
FROM DEPARTMENT D
LEFT JOIN PERSON P ON D.DEPARTMENTID = P.DEPARTMENTID
WHERE P.PERSONID IS NULL;
```

3. List out department names in which more than two persons are working.

```
SELECT D.DEPARTMENTNAME
FROM PERSON P
INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID
GROUP BY D.DEPARTMENTNAME
HAVING COUNT(P.PERSONID) > 2;
```

4. Give a 10% increment in the Computer department employee’s salary. (Use Update)

```
 UPDATE P
 SET P.SALARY = P.SALARY * 1.10
 FROM PERSON P
 INNER JOIN DEPARTMENT D ON P.DEPARTMENTID = D.DEPARTMENTID
WHERE D.DEPARTMENTNAME = 'COMPUTER';
```

5. Calculate Employee Experience in Years, Months & Days with respect to their joining Date.

```
SELECT DATEDIFF(YEAR, JOININGDATE, GETDATE()) AS YEARS, DATEDIFF(MONTH, JOININGDATE, GETDATE()) % 12 AS MONTHS, DATEDIFF(DAY, DATEADD(MONTH, DATEDIFF(MONTH, JOININGDATE, GETDATE()),JOININGDATE), GETDATE()) AS DAYS
FROM PERSON;
```

8

Part A:

Create table as per following.

ORDERS

ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.5	05-10-2012	3005	5002
70009	270.65	10-09-2012	3001	5005
70002	65.26	05-10-2012	3002	5001
70004	110.5	17-08-2012	3009	5003
70007	948.5	10-09-2012	3005	5002
70005	2400.6	27-07-2012	3007	5001
70008	5760	10-09-2012	3002	5001
70010	1983.43	10-10-2012	3004	5006
70003	2480.4	10-10-2012	3009	5003
70012	250.45	27-06-2012	3008	5002
70011	75.29	17-08-2012	3003	5007
70013	3045.6	25-04-2012	3002	5001
70001	150.5	05-10-2012	3005	5002
70009	270.65	10-09-2012	3001	5005
70002	65.26	05-10-2012	3002	5001

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
5007	Paul Adam	Rome	0.13
5003	Lauson Hen	San Jose	0.12

CUSTOMER

customer_id	cust_name	City	Grade	salesman_id
3002	Nick Rimando	New York	100	5001
3007	Brad Davis	New York	200	5001
3005	Graham Zusi	California	200	5002
3008	Julian Green	London	300	5002
3004	Fabian Johnson	Paris	300	5006
3009	Geoff Cameron	Berlin	100	5003
3003	Jozy Altidor	Moscow	200	5007
3001	Brad Guzan	London		5005

From the above given tables perform the following SQL queries (Join):

1. Write a SQL query to find the salesperson and customer who reside in the same city. Return Salesman, cust_name and city.
   ```
   SELECT S.NAME AS SALESMAN, C.CUST_NAME AS CUSTOMER, S.CITY
   FROM SALESMAN S
   INNER JOIN CUSTOMER C ON S.CITY = C.CITY;
   ```
2. Write a SQL query to find those orders where the order amount exists between 500 and 2000. Return ord_no, purch_amt, cust_name, city.
   ```
   SELECT O.ORD_NO, O.PURCH_AMT, C.CUST_NAME, C.CITY
   FROM ORDERS O
   INNER JOIN CUSTOMER C ON O.CUSTOMER_ID = C.CUSTOMER_ID
   WHERE O.PURCH_AMT BETWEEN 500 AND 2000;
   ```
3. Write a SQL query to find the salesperson(s) and the customer(s) he represents. Return Customer Name, city, Salesman, commission.
   ```
   SELECT C.CUST_NAME AS CUSTOMER, C.CITY, S.NAME AS SALESMAN, S.COMMISSION
   FROM CUSTOMER C
   INNER JOIN SALESMAN S ON C.SALESMAN_ID = S.SALESMAN_ID;
   ```
4. Write a SQL query to find salespeople who received commissions of more than 12 percent from the company. Return Customer Name, customer city, Salesman, commission.
   ```
   SELECT C.CUST_NAME AS CUSTOMER, C.CITY AS CUSTOMERCITY, S.NAME AS SALESMAN,
   S.COMMISSION
   FROM CUSTOMER C
   INNER JOIN SALESMAN S ON C.SALESMAN_ID = S.SALESMAN_ID
   WHERE S.COMMISSION > 0.12;
   ```
5. Write a SQL query to locate those salespeople who do not live in the same city where their customers live and have received a commission of more than 12% from the company. Return Customer Name, customer city, Salesman, salesman city, commission.
   ```
   SELECT C.CUST_NAME AS CUSTOMER, C.CITY AS CUSTOMERCITY, S.NAME AS SALESMAN, S.CITY AS
   SALESMANCITY, S.COMMISSION FROM CUSTOMER C
   INNER JOIN SALESMAN S ON C.SALESMAN_ID = S.SALESMAN_ID
   WHERE C.CITY != S.CITY AND S.COMMISSION > 0.12;
   ```
6. Write a SQL query to find the details of an order. Return ord_no, ord_date, purch_amt, Customer Name, grade, Salesman, commission.

```
SELECT O.ORD_NO, O.ORD_DATE, O.PURCH_AMT, C.CUST_NAME AS CUSTOMER, C.GRADE, S.NAME AS SALESMAN, S.COMMISSION FROM ORDERS O
INNER JOIN CUSTOMER C ON O.CUSTOMER_ID = C.CUSTOMER_ID
INNER JOIN SALESMAN S ON O.SALESMAN_ID = S.SALESMAN_ID;
```

7. Write a SQL statement to join the tables salesman, customer and orders so that the same column of each table appears once and only the relational rows are returned.

```
SELECT O.ORD_NO, O.ORD_DATE, O.PURCH_AMT, C.CUST_NAME, C.CITY AS CUSTOMERCITY, C.GRADE, S.NAME AS SALESMAN, S.CITY AS SALESMANCITY, S.COMMISSION
FROM ORDERS O
INNER JOIN CUSTOMER C ON O.CUSTOMER_ID = C.CUSTOMER_ID
INNER JOIN SALESMAN S ON C.SALESMAN_ID = S.SALESMAN_ID;
```

8. Write a SQL query to display the customer name, customer city, grade, salesman, salesman city. The results should be sorted by ascending customer_id.

```
SELECT C.CUST_NAME, C.CITY AS CUSTOMERCITY, C.GRADE, S.NAME AS SALESMAN, S.CITY AS SALESMANCITY
FROM CUSTOMER C
INNER JOIN SALESMAN S ON C.SALESMAN_ID = S.SALESMAN_ID
ORDER BY C.CUSTOMER_ID ASC;
```

9. Write a SQL query to find those customers with a grade less than 300. Return cust_name, customer city, grade, Salesman, salesmancity. The result should be ordered by ascending customer_id.

```
SELECT C.CUST_NAME, C.CITY AS CUSTOMERCITY, C.GRADE, S.NAME AS SALESMAN, S.CITY AS SALESMANCITY
FROM CUSTOMER C
INNER JOIN SALESMAN S ON C.SALESMAN_ID = S.SALESMAN_ID
WHERE C.GRADE < 300
ORDER BY C.CUSTOMER_ID ASC;
```

10. Write a SQL statement to make a report with customer name, city, order number, order date, and order amount in ascending order according to the order date to determine whether any of the existing customers have placed an order or not.

```
SELECT C.CUST_NAME AS CUSTOMER, C.CITY AS CITY, O.ORD_NO AS ORDERNUMBER, O.ORD_DATE AS ORDERDATE, O.PURCH_AMT AS ORDERAMOUNT
FROM CUSTOMER C
LEFT JOIN ORDERS O ON C.CUSTOMER_ID = O.CUSTOMER_ID
ORDER BY O.ORD_DATE ASC;
```

Part-B:

1. Write a SQL statement to generate a report with customer name, city, order number, order date, order amount, salesperson name, and commission to determine if any of the existing customers have not placed orders or if they have placed orders through their salesman or by themselves.

```
SELECT C.CUST_NAME AS CUSTOMER, C.CITY AS CITY, O.ORD_NO AS ORDERNUMBER, O.ORD_DATE AS ORDERDATE, O.PURCH_AMT AS ORDERAMOUNT, S.NAME AS SALESPERSON, S.COMMISSION AS COMMISSION
FROM CUSTOMER C
LEFT JOIN ORDERS O ON C.CUSTOMER_ID = O.CUSTOMER_ID
LEFT JOIN SALESMAN S ON C.SALESMAN_ID = S.SALESMAN_ID
```

```
ORDER BY O.ORD_DATE;
```

2. Write a SQL statement to generate a list in ascending order of salespersons who work either for one or more customers or have not yet joined any of the customers.

```
SELECT S.NAME AS SALESPERSON, COUNT(C.CUSTOMER_ID) AS 'NO. OF CUSTOMERS'
FROM SALESMAN S
LEFT JOIN CUSTOMER C ON S.SALESMAN_ID = C.SALESMAN_ID
GROUP BY S.SALESMAN_ID,S.NAME
ORDER BY 'NO. OF CUSTOMERS',S.NAME;
```

3. Write a SQL query to list all salespersons along with customer name, city, grade, order number, date, and amount.

```
SELECT S.NAME AS SALESPERSON, C.CUST_NAME AS CUSTOMER, C.CITY AS CITY, C.GRADE,
O.ORD_NO AS ORDERNUMBER,  O.ORD_DATE AS ORDERDATE, O.PURCH_AMT AS ORDERAMOUNT
FROM SALESMAN S
LEFT JOIN CUSTOMER C ON S.SALESMAN_ID = C.SALESMAN_ID
LEFT JOIN ORDERS O ON C.CUSTOMER_ID = O.CUSTOMER_ID
```

4. Write a SQL statement to make a list for the salesmen who either work for one or more customers or yet to join any of the customer. The customer may have placed, either one or more orders on or above order amount 2000 and must have a grade, or he may not have placed any order to the associated supplier.

```
SELECT S.NAME AS SALESPERSON
FROM SALESMAN S
LEFT JOIN CUSTOMER C ON S.SALESMAN_ID = C.SALESMAN_ID
LEFT JOIN ORDERS O ON C.CUSTOMER_ID = O.CUSTOMER_ID
WHERE (O.PURCH_AMT >= 2000 OR C.GRADE IS NOT NULL)
```

5. For those customers from the existing list who put one or more orders, or which orders have been placed by the customer who is not on the list, create a report containing the customer name, city, order number, order date, and purchase amount.

```
SELECT C.CUST_NAME AS CUSTOMER, C.CITY AS CITY, O.ORD_NO AS ORDERNUMBER, O.ORD_DATE
AS ORDERDATE, O.PURCH_AMT AS ORDERAMOUNT
FROM ORDERS O
LEFT JOIN CUSTOMER C ON O.CUSTOMER_ID = C.CUSTOMER_ID
```

Part-C:

1. Write a SQL statement to generate a report with the customer name, city, order no. order date, purchase amount for only those customers on the list who must have a grade and placed one or more orders or which order(s) have been placed by the customer who neither is on the list nor has a grade.

```
SELECT C.CUST_NAME AS CUSTOMER, C.CITY AS CITY, O.ORD_NO AS ORDERNUMBER, O.ORD_DATE
AS ORDERDATE, O.PURCH_AMT AS ORDERAMOUNT
FROM CUSTOMER C
LEFT JOIN ORDERS O ON C.CUSTOMER_ID = O.CUSTOMER_ID
WHERE (C.GRADE IS NOT NULL AND O.ORD_NO IS NOT NULL)
OR (C.CUSTOMER_ID IS NULL AND C.GRADE IS NULL)
```

2. Write a SQL query to combine each row of the salesman table with each row of the customer table.

```
SELECT *
FROM SALESMAN
```

```
CROSS JOIN CUSTOMER
```

3. Write a SQL statement to create a Cartesian product between salesperson and customer, i.e. each salesperson will appear for all customers and vice versa for that salesperson who belongs to that city.

```
SELECT * FROM SALESMAN S
CROSS JOIN CUSTOMER C
WHERE S.CITY = C.CITY;
```

4. Write a SQL statement to create a Cartesian product between salesperson and customer, i.e. each salesperson will appear for every customer and vice versa for those salesmen who belong to a city and customers who require a grade.

```
SELECT * FROM SALESMAN S
CROSS JOIN CUSTOMER C
WHERE S.CITY = C.CITY AND C.GRADE IS NOT NULL;
```

5. Write a SQL statement to make a Cartesian product between salesman and customer i.e. each salesman will appear for all customers and vice versa for those salesmen who must belong to a city which is not the same as his customer and the customers should have their own grade.

```
SELECT * FROM SALESMAN S
CROSS JOIN CUSTOMER C
WHERE S.CITY != C.CITY AND C.GRADE IS NOT NULL;
```

9

Part – A: Create table as per following.

CITY

CityID **(Primary Key)**	Name **(Unique Key)**	Pincode **(Not Null)**	Remakrs
1	Rajkot	360005	Good
2	Surat	335009	Very Good
3	Baroda	390001	Awesome
4	Jamnagar	361003	Smart
5	Junagadh	362229	Historic
6	Morvi	363641	Ceramic

VILLAGE

VID **(Primary Key)**	Name **(Not Null)**	CityID **(Foreign Key)**
101	Raiya	1
102	Madhapar	1
103	Dodka	3
104	Falla	4
105	Bhesan	5
106	Dhoraji	5

From the above given tables perform the following SQL queries:

1. Display all the villages of Rajkot city.

```
SELECT V.NAME
FROM VILLAGE V
INNER JOIN CITY C ON V.CITYID = C.CITYID
WHERE C.NAME = 'RAJKOT';
```

2. Display city along with their villages & pin code.

```
SELECT C.NAME AS CITYNAME, C.PINCODE, V.NAME AS VILLAGENAME
FROM CITY C
LEFT OUTER JOIN VILLAGE V ON C.CITYID = V.CITYID;
```

3. Display the city having more than one village.

```
SELECT C.NAME
FROM CITY C
INNER JOIN VILLAGE V ON C.CITYID = V.CITYID
GROUP BY C.NAME
```

```
HAVING COUNT(V.VID) > 1;
```

4. Display the city having no village.

```
SELECT C.NAME
FROM CITY C
LEFT OUTER JOIN VILLAGE V ON C.CITYID = V.CITYID
WHERE V.CITYID IS NULL;
```

5. Count the total number of villages in each city.

```
SELECT C.NAME, COUNT(V.VID) AS VILLAGECOUNT
FROM CITY C
LEFT OUTER JOIN VILLAGE V ON C.CITYID = V.CITYID
GROUP BY C.NAME;
```

6. Count the number of cities having more than one village.

```
SELECT COUNT(*) FROM
(SELECT C.NAME,COUNT(V.CITYID) AS VILLAGE FROM CITY C
INNER JOIN VILLAGE V ON C.CITYID=V.CITYID
GROUP BY C.NAME) AS T
WHERE VILLAGE >1
```

Create below table Student with following constraints

1. Do not allow SPI more than 10 and less than 0.
2. Do not allow Bklog less than 0.
3. Enter the default value as ‘General’ in branch to all new records IF no other value is specified.

```
CREATE TABLE STUDENT
(
        RNo INT IDENTITY(101,1) PRIMARY KEY,
        NAME VARCHAR(50) NULL,
        BRANCH VARCHAR(50) DEFAULT 'GENRAL',
        SPI DECIMAL(3,2) CHECK(SPI BETWEEN 0 AND 10),
        BKLOG INT CHECK(BKLOG>=0));
```

STUDENT

Rno(PK)	Name	Branch	SPI	Bklog
101	Raju	CE	8.80	0
102	Amit	CE	2.20	3
103	Sanjay	ME	1.50	6
104	Neha	EC	7.65	0
105	Meera	EE	5.52	2
106	Mahesh	General	4.50	3

► Try to update SPI of Raju from 8.80 to 12.

UPDATE Student SET SPI = 12 WHERE Name = 'Raju'; This will fail due to CHECK constraint

► Try to update Bklog of Neha from 0 to -1.

UPDATE Student SET Bklog = -1 WHERE Name = 'Neha'; This will fail due to CHECK constraint

Part - B: Create table as per following schema with proper validation and try to insert data which violate your validation.

1. Emp(Eid, Ename, Did, Cid, Salary, Experience)
   Dept(Did, Dname)
   City(Cid, Cname)

```
CREATE TABLE Dept (
Did INT PRIMARY KEY,
Dname VARCHAR(100) NOT NULL);
CREATE TABLE City (
Cid INT PRIMARY KEY,
Cname VARCHAR(100) NOT NULL );

CREATE TABLE Emp (
Eid INT PRIMARY KEY,
Ename VARCHAR(100) NOT NULL,
Did INT NOT NULL FOREIGN KEY REFERENCES Dept(Did),
Cid INT NOT NULL FOREIGN KEY REFERENCES City(Cid),
Salary DECIMAL(10, 2) NOT NULL CHECK (Salary > 0),
Experience INT NOT NULL CHECK (Salary > 0));
INSERT INTO Emp (Eid, Ename, Did, Cid, Salary, Experience) VALUES (1, 'John Doe', 1, 1, -5000, 3);
INSERT INTO Emp (Eid, Ename, Did, Cid, Salary, Experience) VALUES (2, 'Grace Lee', 3, 3, 900.00, -2);
```

Part - C: Create table as per following schema with proper validation and try to insert data which violate your validation.

1. Emp(Eid, Ename, Did, Cid, Salary, Experience)
   Dept(Did, Dname)
   City(Cid, Cname, Did))
   District(Did, Dname, Sid)
   State(Sid, Sname, Cid)
   Country(Cid, Cname)
2. Insert 5 records in each table.
3. Display employeename, departmentname, Salary, Experience, City, District, State and country of all employees.

```
SELECT E.ENAME AS EMPLOYEENAME,  D.DNAME AS DEPARTMENTNAME, E.SALARY, E.EXPERIENCE,
C1.CNAME AS CITY, D2.DNAME AS DISTRICT, S.SNAME AS STATE,  C2.CNAME AS COUNTRY
FROM EMP E
INNER JOIN DEPT D ON E.DID = D.DID
INNER JOIN CITY C1 ON E.CID = C1.CID
INNER JOIN DISTRICT D2 ON C1.DID = D2.DID
INNER JOIN STATE S ON D2.SID = S.SID
INNER JOIN COUNTRY C2 ON S.CID = C2.CID;
```

10

Part - A:

Create table as per following.

STUDENT

Rno	Name	City	DID
101	Raju	Rajkot	10
102	Amit	Ahmedabad	20
103	Sanjay	Baroda	40
104	Neha	Rajkot	20
105	Meera	Ahmedabad	30
106	Mahesh	Baroda	10

ACADEMIC

Rno	SPI	Bklog
101	8.8	0
102	9.2	2
103	7.6	1
104	8.2	4
105	7.0	2
106	8.9	3

DEPARTMENT

DID	DName
10	Computer
20	Electrical
30	Mechanical
40	Civil

From the above given tables perform the following SQL queries (Sub Queries):

1. Display details of students who are from computer department.

```
SELECT * FROM STUDENT
WHERE DID = (SELECT DID FROM DEPARTMENT
             WHERE DNAME = 'COMPUTER');
```

2. Displays name of students whose SPI is more than 8.

```
SELECT NAME
FROM STUDENT
WHERE RNO IN (SELECT RNO FROM ACADEMIC WHERE SPI > 8);
```

3. Display details of students of computer department who belongs to Rajkot city.

```
SELECT * FROM STUDENT
WHERE DID = (SELECT DID FROM DEPARTMENT WHERE DNAME = 'COMPUTER')
AND CITY = 'RAJKOT';
```

4. Find total number of students of electrical department.

```
SELECT COUNT(*) AS TOTAL_ELECTRICAL_STUDENTS
FROM STUDENT
WHERE DID = (SELECT DID FROM DEPARTMENT WHERE DNAME = 'ELECTRICAL');
```

5. Display name of student who is having maximum SPI.

```
SELECT NAME FROM STUDENT
WHERE RNO IN (SELECT RNO FROM ACADEMIC
WHERE SPI = (SELECT MAX(SPI)FROM ACADEMIC));
```

6. Display details of students having more than 1 backlog.

```
SELECT * FROM STUDENT
WHERE RNO IN (SELECT RNO FROM ACADEMIC WHERE BKLOG > 1);
```

7. Display name of student who is having second highest SPI.

```
SELECT NAME FROM STUDENT
WHERE RNO IN (SELECT RNO FROM ACADEMIC
               WHERE SPI = (SELECT MAX(SPI) FROM ACADEMIC
                            WHERE SPI < (SELECT MAX(SPI) FROM ACADEMIC)));
```

8. Display name of students who are either from computer department or from mechanical department.

```
SELECT NAME FROM STUDENT
WHERE DID IN (SELECT DID FROM DEPARTMENT
              WHERE DNAME IN ('COMPUTER', 'MECHANICAL'));
```

9. Display name of students who are in same department as 102 studying in.

```
SELECT NAME FROM STUDENT
WHERE DID = (SELECT DID FROM STUDENT
                WHERE RNO = 102);
```

10. Display name of students whose SPI is more than 9 and who is from electrical department.

```
SELECT NAME FROM STUDENT
WHERE DID IN (SELECT DID FROM DEPARTMENT WHERE DNAME = 'ELECTRICAL')
AND RNO IN (SELECT RNO FROM ACADEMIC WHERE SPI > 9);
```

Part - B: Create table as per following.

COMPANY_MASTER

COM_ID	COM_NAME
11	Samsung
12	iBall
13	Epsion
14	Zebronics
15	Asus
16	Frontech

ITEM_MASTER

PRO_ID	PRO_NAME	PRO_PRICE	PRO_COM
101	Mother Board	3200	15
102	Key Board	450	16
103	ZIP drive	250	14
104	Speaker	550	16
105	Monitor	5000	11
106	DVD drive	900	12
107	CD drive	800	12
108	Printer	2600	13
109	Refill cartridge	350	13
110	Mouse	250	12

EMP_DETAILS

EMP_IDNO	EMP_FNAME	EMP_LNAME	EMP_DEPT
127323	Michale	Robbin	57
526689	Carlos	Snares	63
843795	Enric	Dosio	57
328717	Jhon	Snares	63
444527	Joseph	Dosni	47
659831	Zanifer	Emily	47
847674	Kuleswar	Sitaraman	57
748681	Henrey	Gabriel	47
555935	Alex	Manuel	57
539569	George	Mardy	27
733843	Mario	Saule	63
631548	Alan	Snappy	27
839139	Maria	Foster	57

EMP_DEPARTMENT

DPT_CODE	DPT_NAME	DPT_ALLOTMENT
57	IT	65000
63	Finance	15000
47	HR	240000
27	RD	55000
89	QC	75000

From the above given tables perform the following SQL queries:

1. Write a SQL query to calculate the average price of each manufacturer's product along with their name. Return Average Price and Company.
```
SELECT CM.COM_NAME AS Company, AVG(IM.PRO_PRICE) AS Average_Price
FROM ITEM_MASTER IM
INNER JOIN COMPANY_MASTER CM ON IM.PRO_COM = CM.COM_ID
GROUP BY CM.COM_NAME;
```
2. Write a SQL query to calculate the average price of each manufacturer's product of 350 or more. Return Average Price and Company.
```
SELECT CM.COM_NAME AS Company, AVG(IM.PRO_PRICE) AS Average_Price
FROM ITEM_MASTER IM
INNER JOIN COMPANY_MASTER CM ON IM.PRO_COM = CM.COM_ID
WHERE IM.PRO_PRICE >= 350
GROUP BY CM.COM_NAME;
```
3. Write a SQL query to find the most expensive product of each company. Return Product Name, Price and Company.
```
SELECT IM.PRO_NAME, IM.PRO_PRICE, CM.COM_NAME
FROM ITEM_MASTER IM
INNER JOIN COMPANY_MASTER CM ON CM.COM_ID = IM.PRO_COM
WHERE IM.PRO_PRICE = (SELECT MAX(IM2.PRO_PRICE)
                      FROM ITEM_MASTER IM2
                      WHERE IM2.PRO_COM = IM.PRO_COM);
```
4. Write a SQL query to find employees whose last name is Gabriel or Dosio. Return emp_idno, emp_fname, emp_lname and emp_dept.
```
SELECT EMP_IDNO, EMP_FNAME, EMP_LNAME, EMP_DEPT
FROM EMP_DETAILS
WHERE EMP_LNAME IN ('GABRIEL', 'DOSIO');
```
5. Write a SQL query to find the employees who work in department 89 or 63. Return emp_idno, emp_fname, emp_lname and emp_dept.
```
SELECT EMP_IDNO, EMP_FNAME, EMP_LNAME, EMP_DEPT
FROM EMP_DETAILS
WHERE EMP_DEPT IN (89, 63);
```

Part - C:

1. Write a SQL query to find those employees who work for the department where the departmental allotment amount is more than Rs. 50000. Return emp_fname and emp_lname.
```
SELECT EMP_FNAME, EMP_LNAME FROM EMP_DETAILS
WHERE EMP_DEPT IN (SELECT DPT_CODE FROM EMP_DEPARTMENT
                   WHERE DPT_ALLOTMENT > 50000);
```
2. Write a SQL query to find the departments whose sanction amount is higher than the average sanction amount for all departments. Return dpt_code, dpt_name and dpt_allotment.
```
SELECT DPT_CODE, DPT_NAME, DPT_ALLOTMENT
```

```
FROM EMP_DEPARTMENT
WHERE DPT_ALLOTMENT > (SELECT AVG(DPT_ALLOTMENT) FROM EMP_DEPARTMENT);
```

3. Write a SQL query to find which departments have more than two employees. Return dpt_name.

```
SELECT DPT_NAME FROM EMP_DEPARTMENT
WHERE DPT_CODE IN (SELECT EMP_DEPT FROM EMP_DETAILS
GROUP BY EMP_DEPT
HAVING COUNT(*) > 2);
```

4. Write a SQL query to find the departments with the second lowest sanction amount. Return emp_fname and emp_lname

```
SELECT ED.EMP_FNAME, ED.EMP_LNAME
FROM EMP_DETAILS ED
INNER JOIN EMP_DEPARTMENT EDPT ON ED.EMP_DEPT = EDPT.DPT_CODE
WHERE EDPT.DPT_ALLOTMENT = (SELECT DPT_ALLOTMENT FROM EMP_DEPARTMENT
                            ORDER BY DPT_ALLOTMENT ASC
                            OFFSET 1 ROWS FETCH NEXT 1 ROWS ONLY);
```

11 **Part - A: Create table as per following.**

ORDERS

ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.5	05-10-2012	3005	5002
70009	270.65	10-09-2012	3001	5005
70002	65.26	05-10-2012	3002	5001
70004	110.5	17-08-2012	3009	5003
70007	948.5	10-09-2012	3005	5002
70005	2400.6	27-07-2012	3007	5001
70008	5760	10-09-2012	3002	5001
70010	1983.43	10-10-2012	3004	5006
70003	2480.4	10-10-2012	3009	5003
70012	250.45	27-06-2012	3008	5002
70011	75.29	17-08-2012	3003	5007
70013	3045.6	25-04-2012	3002	5001
70001	150.5	05-10-2012	3005	5002
70009	270.65	10-09-2012	3001	5005
70002	65.26	05-10-2012	3002	5001

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
5007	Paul Adam	Rome	0.13
5003	Lauson Hen	San Jose	0.12

CUSTOMER

customer_id	cust_name	city	Grade	salesman_id
3002	Nick Rimando	New York	100	5001
3007	Brad Davis	New York	200	5001
3005	Graham Zusi	California	200	5002
3008	Julian Green	London	300	5002
3004	Fabian Johnson	Paris	300	5006
3009	Geoff Cameron	Berlin	100	5003
3003	Jozy Altidor	Moscow	200	5007
3001	Brad Guzan	London		5005

From the above given tables perform the following queries:

1. Write a SQL query to find all the orders issued by the salesman 'Paul Adam'. Return ord_no, purch_amt, ord_date, customer_id and salesman_id.
```
SELECT O.ORD_NO, O.PURCH_AMT, O.ORD_DATE, O.CUSTOMER_ID, O.SALESMAN_ID
FROM ORDERS O
INNER JOIN SALESMAN S ON O.SALESMAN_ID = S.SALESMAN_ID
WHERE S.NAME = 'PAUL ADAM';
```
2. Write a SQL query to find all orders generated by London-based salespeople. Return ord_no, purch_amt, ord_date, customer_id, salesman_id.
```
SELECT ORD_NO, PURCH_AMT, ORD_DATE, CUSTOMER_ID, SALESMAN_ID
FROM ORDERS
WHERE SALESMAN_ID IN (SELECT SALESMAN_ID FROM SALESMAN
                      WHERE CITY = 'LONDON');
```
3. Write a SQL query to find all orders generated by the salespeople who may work for customers whose id is 3007. Return ord_no, purch_amt, ord_date, customer_id, salesman_id.
```
SELECT O.ORD_NO, O.PURCH_AMT, O.ORD_DATE, O.CUSTOMER_ID,C.CUSTOMER_ID,
O.SALESMAN_ID,C.SALESMAN_ID
FROM ORDERS O
INNER JOIN CUSTOMER C ON O.SALESMAN_ID= C.SALESMAN_ID
WHERE C.CUSTOMER_ID = 3007;
```
4. Write a SQL query to find the order values greater than the average order value of 10th October 2012. Return ord_no, purch_amt, ord_date, customer_id, salesman_id.
```
SELECT O.ORD_NO, O.PURCH_AMT, O.ORD_DATE, O.CUSTOMER_ID, O.SALESMAN_ID
FROM ORDERS O
WHERE O.PURCH_AMT > (SELECT AVG(PURCH_AMT) FROM ORDERS
                     WHERE ORD_DATE = '2012-10-10');
```
5. Write a SQL query to find all the orders generated in New York city. Return ord_no, purch_amt, ord_date, customer_id and salesman_id.
```
SELECT O.ORD_NO, O.PURCH_AMT, O.ORD_DATE, O.CUSTOMER_ID, O.SALESMAN_ID
FROM ORDERS O
INNER JOIN CUSTOMER C ON O.CUSTOMER_ID = C.CUSTOMER_ID
WHERE C.CITY = 'NEW YORK';
```
6. Write a SQL query to determine the commission of the salespeople in Paris. Return commission.
```
SELECT COMMISSION
FROM SALESMAN
WHERE CITY = 'PARIS';
```

7. Write a query to display all the customers whose ID is 2001 below the salesperson ID of Mc Lyon.

```
SELECT * FROM CUSTOMER
WHERE CUSTOMER_ID < 2001
AND SALESMAN_ID = (SELECT SALESMAN_ID  FROM SALESMAN
                          WHERE NAME = 'MC LYON');
```

8. write a SQL query to count the number of customers with grades above the average in New York City. Return grade and count.

```
SELECT GRADE, COUNT(*) AS COUNT
FROM CUSTOMER
WHERE GRADE > (SELECT AVG(GRADE) FROM CUSTOMER
                    WHERE CITY = 'NEW YORK')
GROUP BY GRADE;
```

9. Write a SQL query to find those salespeople who earned the maximum commission. Return ord_no, purch_amt, ord_date, and salesman_id.

```
SELECT O.ORD_NO, O.PURCH_AMT, O.ORD_DATE, O.SALESMAN_ID
FROM ORDERS O
INNER JOIN SALESMAN S ON O.SALESMAN_ID = S.SALESMAN_ID
WHERE S.COMMISSION = (SELECT MAX(COMMISSION) FROM SALESMAN);
```

10. Write SQL query to find the customers who placed orders on 17th August 2012. Return ord_no, purch_amt, ord_date, customer_id, salesman_id and cust_name.

```
SELECT O.ORD_NO, O.PURCH_AMT, O.ORD_DATE, O.CUSTOMER_ID, O.SALESMAN_ID, C.CUST_NAME
FROM ORDERS O
INNER JOIN CUSTOMER C ON O.CUSTOMER_ID = C.CUSTOMER_ID
WHERE O.ORD_DATE = '2012-08-17';
```

11. Write a SQL query to find salespeople who had more than one customer. Return salesman_id and name.

```
SELECT S.SALESMAN_ID, S.NAME
FROM SALESMAN S
INNER JOIN CUSTOMER C ON S.SALESMAN_ID = C.SALESMAN_ID
GROUP BY S.SALESMAN_ID, S.NAME
HAVING COUNT(C.CUSTOMER_ID) > 1;
```

12. Write a SQL query to find those orders, which are higher than the average amount of the orders. Return ord_no, purch_amt, ord_date, customer_id and salesman_id.

```
SELECT O.ORD_NO, O.PURCH_AMT, O.ORD_DATE, O.CUSTOMER_ID, O.SALESMAN_ID
FROM ORDERS O
WHERE O.PURCH_AMT > ( SELECT AVG(PURCH_AMT) FROM ORDERS);
```

13. Write a SQL query to find those orders that are equal or higher than the average amount of the orders. Return ord_no, purch_amt, ord_date, customer_id and salesman_id.

```
SELECT O.ORD_NO, O.PURCH_AMT, O.ORD_DATE, O.CUSTOMER_ID, O.SALESMAN_ID
FROM ORDERS O
WHERE O.PURCH_AMT >= (SELECT AVG(PURCH_AMT) FROM ORDERS);
```

14. Write a query to find the sums of the amounts from the orders table, grouped by date, and eliminate all dates where the sum was not at least 1000.00 above the maximum order amount for that date.

```
SELECT ORD_DATE, SUM(PURCH_AMT) AS TOTAL_AMOUNT
FROM ORDERS
GROUP BY ORD_DATE
HAVING SUM(PURCH_AMT) >= (SELECT MAX(PURCH_AMT) + 1000.00 FROM ORDERS
                                WHERE ORD_DATE = ORDERS.ORD_DATE);
```

15. Write a query to extract all data from the customer table if and only if one or more of the customers in the customer table are located in London. Sample table : Customer

```
SELECT * FROM CUSTOMER
WHERE EXISTS (SELECT 1 FROM CUSTOMER
                          WHERE CITY = 'LONDON');
```

Part - B:

1. Write a SQL query to find salespeople who deal with multiple customers. Return salesman_id, name, city and commission.

```
SELECT S.SALESMAN_ID, S.NAME, S.CITY, S.COMMISSION
FROM SALESMAN S
INNER JOIN CUSTOMER C ON S.SALESMAN_ID = C.SALESMAN_ID
GROUP BY S.SALESMAN_ID, S.NAME, S.CITY, S.COMMISSION
HAVING COUNT(C.CUSTOMER_ID) > 1;
```

2. Write a SQL query to find salespeople who deal with a single customer. Return salesman_id, name, city and commission.

```
SELECT S.SALESMAN_ID, S.NAME, S.CITY, S.COMMISSION
FROM SALESMAN S
INNER JOIN CUSTOMER C ON S.SALESMAN_ID = C.SALESMAN_ID
GROUP BY S.SALESMAN_ID, S.NAME, S.CITY, S.COMMISSION
HAVING COUNT(C.CUSTOMER_ID) = 1;
```

3. Write a SQL query to find the salespeople who deal the customers with more than one order. Return salesman_id, name, city and commission.

```
SELECT S.SALESMAN_ID, S.NAME, S.CITY, S.COMMISSION
FROM SALESMAN S
INNER JOIN ORDERS O ON S.SALESMAN_ID = O.SALESMAN_ID
GROUP BY  S.SALESMAN_ID, S.NAME, S.CITY, S.COMMISSION
HAVING COUNT(ORD_NO) > 1
```

4. Write a SQL query to find the salespeople who deal with those customers who live in the same city. RETURN SALESMAN_ID, NAME, CITY AND COMMISSION.

```
SELECT DISTINCT S.SALESMAN_ID, S.NAME,S.CITY, S.COMMISSION
FROM SALESMAN S
INNER JOIN CUSTOMER C ON S.SALESMAN_ID = C.SALESMAN_ID
WHERE S.CITY = C.CITY;
```

5. Write a SQL query to find salespeople whose place of residence matches any city where customers live. Return salesman_id, name, city and commission.

```
SELECT DISTINCT S.SALESMAN_ID, S.NAME,C.CUST_NAME, S.CITY, S.COMMISSION
FROM SALESMAN S
INNER JOIN CUSTOMER C ON S.CITY = C.CITY;
```

6. Write a SQL query to find all those salespeople whose names appear alphabetically lower than the customer's name. Return salesman_id, name, city, commission.

```
SELECT S.SALESMAN_ID, S.NAME, S.CITY, S.COMMISSION
FROM SALESMAN S
INNER JOIN CUSTOMER C ON S.SALESMAN_ID = C.SALESMAN_ID
WHERE S.NAME < C.CUST_NAME;
```

7. Write a SQL query to find all those customers with a higher grade than all the customers alphabetically below the city of New York. Return customer_id, cust_name, city, grade, salesman_id.

```
SELECT customer_id, cust_name, city, grade, salesman_id
FROM CUSTOMER
WHERE grade >  (SELECT MAX(grade) FROM CUSTOMER
                WHERE city < 'New York');
```

8. Write a SQL query to find all those orders whose order amount exceeds at least one of the orders placed on September 10th 2012. Return ord_no, purch_amt, ord_date, customer_id and salesman_id.

```
SELECT ORD_NO, PURCH_AMT, ORD_DATE, CUSTOMER_ID, SALESMAN_ID
```

```
FROM ORDERS
WHERE PURCH_AMT > (SELECT MAX(PURCH_AMT) FROM ORDERS WHERE ORD_DATE = '2012-09-10');
```

9. Write a SQL query to find orders where the order amount is less than the order amount of a customer residing in London City. Return ord_no, purch_amt, ord_date, customer_id and salesman_id.

```
SELECT ORD_NO, PURCH_AMT, ORD_DATE, CUSTOMER_ID, SALESMAN_ID
FROM ORDERS
WHERE PURCH_AMT < (SELECT MAX(O2.PURCH_AMT)
                   FROM ORDERS O2
                   INNER JOIN CUSTOMER C ON O2.CUSTOMER_ID = C.CUSTOMER_ID
                   WHERE C.CITY = 'LONDON');
```

10. Write a SQL query to find those orders where every order amount is less than the maximum order amount of a customer who lives in London City. Return ord_no, purch_amt, ord_date, customer_id and salesman_id.

```
SELECT ORD_NO, PURCH_AMT, ORD_DATE, CUSTOMER_ID, SALESMAN_ID
FROM ORDERS
WHERE PURCH_AMT < (SELECT MAX(O2.PURCH_AMT) FROM ORDERS O2
                          INNER JOIN CUSTOMER C ON O2.CUSTOMER_ID = C.CUSTOMER_ID
                          WHERE C.CITY = 'LONDON');
```

Part - C:

1. Write a SQL query to find those customers whose grades are higher than those living in New York City. RETURN CUSTOMER_ID, CUST_NAME, CITY, GRADE AND SALESMAN_ID.

```
SELECT CUSTOMER_ID, CUST_NAME, CITY, GRADE, SALESMAN_ID
FROM CUSTOMER
WHERE grade >  ALL ( SELECT grade FROM CUSTOMER WHERE city = 'New York');
```

2. Write a SQL query to calculate the total order amount generated by a salesperson. Salespersons should be from the cities where the customers reside. Return salesperson name, city and total order amount.

```
SELECT S.NAME AS SALESPERSON_NAME, S.CITY AS SALESPERSON_CITY, SUM(O.PURCH_AMT) AS
TOTAL_ORDER_AMOUNT
FROM SALESMAN S
INNER JOIN ORDERS O ON S.SALESMAN_ID = O.SALESMAN_ID
INNER JOIN CUSTOMER C ON O.CUSTOMER_ID = C.CUSTOMER_ID
WHERE S.CITY = C.CITY
GROUP BY S.NAME, S.CITY;
```

3. Write a SQL query to find those customers whose grades are not the same as those who live in London City. Return customer_id, cust_name, city, grade and salesman_id.

```
SELECT C1.CUSTOMER_ID, C1.CUST_NAME, C1.CITY, C1.GRADE, C1.SALESMAN_ID
        FROM CUSTOMER C1
        WHERE GRADE NOT IN (
                SELECT GRADE
                FROM CUSTOMER C2
                WHERE C2.CITY = 'LONDON' AND C1.GRADE = C2.GRADE);
```

4. Write a SQL query to find those customers whose grades are different from those living in Paris. Return customer_id, cust_name, city, grade and salesman_id.

```
SELECT CUSTOMER_ID, CUST_NAME, CITY, GRADE, SALESMAN_ID
FROM CUSTOMER
WHERE GRADE != (SELECT GRADE FROM CUSTOMER WHERE CITY = 'PARIS');
```

5. Write a SQL query to find all those customers who have different grades than any customer who lives in Dallas City. Return customer_id, cust_name,city, grade and salesman_id.

```
SELECT CUSTOMER_ID, CUST_NAME, CITY, GRADE, SALESMAN_ID
FROM CUSTOMER
WHERE GRADE != ( SELECT GRADE FROM CUSTOMER WHERE CITY = 'PARIS');
```

12

Create table as per following.

PERSON

Column_Name	DataType	Constraints
WorkerID	Int	Primary Key, Auto Increment
FirstName	Varchar (100)	Not Null
LastName	Varchar (100)	Not Null
Salary	Decimal (8,2)	Not Null
JoiningDate	Datetime	Not Null
DepartmentID	Int	Foreign Key, Null
DesignationID	Int	Foreign Key, Null

Department

Column_Name	DataType	Constraints
DepartmentID	Int	Primary Key
DepartmentName	Varchar (100)	Not Null, Unique

Designation

Column_Name	DataType	Constraints
DesignationID	Int	Primary Key
DesignationName	Varchar (100)	Not Null, Unique

WorkerID	FirstName	LastName	Salary	JoiningDate	DepartmentID	DesignationID
101	Rahul	Anshu	56000	01-01-1990	1	12
102	Hardik	Hinsu	18000	25-09-1990	2	11
103	Bhavin	Kamani	25000	14-05-1991	NULL	11
104	Bhoomi	Patel	39000	20-02-2014	1	13
105	Rohit	Rajgor	17000	23-07-1990	2	15
106	Priya	Mehta	25000	18-10-1990	2	NULL
107	Neha	Trivedi	18000	20-02-2014	3	15

DepartmentID	DepartmentName
1	Admin
2	IT
3	HR
4	Account

DesignationID	DesignationName
11	Jobber
12	Welder
13	Clerk
14	Manager
15	CEO

From the above given tables create Stored Procedures:

Part – A:

1. Create a Procedure on Department, Designation & Person Table for INSERT, UPDATE & DELETE Procedures.

```
CREATE PROCEDURE INSERT_DEPARTMENT
@DEPARTMENTID INT,
@DEPNAME VARCHAR(100)
AS
BEGIN
	INSERT INTO DEPARTMENT
	VALUES( @DEPARTMENTID, @DEPNAME)
END
```

```
CREATE PROCEDURE INSERT_DESIGNATION
@DESIGID INT,
@DSIGNAME VARCHAR(100)
 AS
 BEGIN
        INSERT INTO DESIGNATION
        VALUES(@DESIGID, @DSIGNAME)
 END

 CREATE PROCEDURE INSERT_PERSON
 @FN VARCHAR(100),
 @LN VARCHAR(100),
 @SAL DECIMAL(8,2),
 @JD DATETIME,
 @DID INT,
 @DESID INT
  AS
  BEGIN
         INSERT INTO PERSON
         VALUES(@FN,@LN,@SAL,@JD,@DID,@DESID)
  END

  CREATE PROCEDURE DELETE_DEPARTMENT
  @DEPARTMENTID INT
  AS
  BEGIN
         DELETE FROM DEPARTMENT
         WHERE DEPARTMENTID = @DEPARTMENTID
  END

  CREATE PROCEDURE DELETE_DESIGNATION
  @DESIGID INT
  AS
  BEGIN
          DELETE FROM DESIGNATION
          WHERE DESIGNATIONID = @DESIGID
  END

  CREATE PROCEDURE DELETE_PERSON
  @WID INT
  AS
  BEGIN
         DELETE FROM PERSON
         WHERE WORKERID = @WID
  END

  CREATE PROCEDURE UPDATE_DEPARTMENT
  @DEPARTMENTID INT,
  @DEPNAME VARCHAR(100)
  AS
  BEGIN
```

```
        UPDATE DEPARTMENT
        SET DEPARTMENTNAME=@DEPNAME
        WHERE DEPARTMENTID=@DEPARTMENTID
 END

 CREATE PROCEDURE UPDATE_DESIGNATION
 @DESIGID INT,
 @DSIGNAME VARCHAR(100)
 AS
 BEGIN
        UPDATE DESIGNATION
        SET DESIGNATIONNAME=@DSIGNAME
        WHERE DESIGNATIONID=@DESIGID
END

CREATE PROCEDURE UPDATE_PERSON
@WID INT,
@FN VARCHAR(100),
@LN VARCHAR(100),
@SAL DECIMAL(8,2),
@JD DATETIME,
@DID INT,
@DESID INT
AS
BEGIN
        UPDATE PERSON
        SET FIRSTNAME = @FN, LASTNAME=@LN, SALARY = @SAL, JOININGDATE = @JD,
               DEPARTMENTID = @DID, DESIGNATIONID=@DESID
         WHERE WORKERID = @WID
END
```

2. Create a Procedure on Department, Designation & Person Table for SELECTBYPRIMARYKEY

```
CREATE PROCEDURE INSERT_DEPARTMENT
@DEPARTMENTID INT,
@DEPNAME VARCHAR(100)
AS
BEGIN
       INSERT INTO DEPARTMENT
       VALUES( @DEPARTMENTID, @DEPNAME)
END

CREATE PROCEDURE INSERT_DESIGNATION
@DESIGID INT,
@DSIGNAME VARCHAR(100)
AS
BEGIN
        INSERT INTO DESIGNATION
        VALUES(@DESIGID, @DSIGNAME)
 END

 CREATE PROCEDURE INSERT_PERSON
```

```
@FN VARCHAR(100),
@LN VARCHAR(100),
@SAL DECIMAL(8,2),
@JD DATETIME,
@DID INT,
@DESID INT
AS
BEGIN
    INSERT INTO PERSON
    VALUES(@FN,@LN,@SAL,@JD,@DID,@DESID)
END

CREATE PROCEDURE DELETE_DEPARTMENT
@DEPARTMENTID INT
AS
BEGIN
    DELETE FROM DEPARTMENT
    WHERE DepartmentID = @DEPARTMENTID
END

CREATE PROCEDURE DELETE_DESIGNATION
@DESIGID INT
AS
BEGIN
     DELETE FROM DESIGNATION
     WHERE DesignationID = @DESIGID
END

CREATE PROCEDURE DELETE_PERSON
@WID INT
AS
BEGIN
    DELETE FROM PERSON
    WHERE WorkerID = @WID
END

CREATE PROCEDURE UPDATE_DEPARTMENT
@DEPARTMENTID INT,
@DEPNAME VARCHAR(100)
AS
BEGIN
    UPDATE DEPARTMENT
    SET DepartmentName=@DEPNAME
    WHERE DepartmentID=@DEPARTMENTID
END

CREATE PROCEDURE UPDATE_DESIGNATION
@DESIGID INT,
@DSIGNAME VARCHAR(100)
AS
BEGIN
```

```
        UPDATE DESIGNATION
        SET DesignationName=@DSIGNAME
        WHERE DesignationID=@DESIGID
    END

    CREATE PROCEDURE UPDATE_PERSON
    @WID INT,
    @FN VARCHAR(100),
    @LN VARCHAR(100),
    @SAL DECIMAL(8,2),
    @JD DATETIME,
    @DID INT,
    @DESID INT
    AS
    BEGIN
        UPDATE PERSON
        SET FirstName = @FN, LastName=@LN, Salary = @SAL, JoiningDate = @JD,
            DepartmentID = @DID, DesignationID=@DESID
        WHERE WorkerID = @WID
    END
```

3. Create a Procedure on Department, Designation & Person Table (If foreign key is available then do write join and take columns on select list)

```
CREATE PROCEDURE JOIN_DEPARTMENT_DESIGNATION_PERSON
AS
BEGIN
    SELECT   P.WORKERID,P.FIRSTNAME,P.LASTNAME,P.JOININGDATE,P.SALARY,
    DEP.DEPARTMENTNAME, DESIG.DESIGNATIONNAME
    FROM PERSON P
    LEFT JOIN DEPARTMENT DEP
    ON DEP.DEPARTMENTID = P.DEPARTMENTID
    LEFT JOIN DESIGNATION DESIG
    ON DESIG.DESIGNATIONID = P.DESIGNATIONID
END
```

4. Create a Procedure that shows details of the first 3 persons.

```
CREATE PROCEDURE PR_TOP3PERSON_ALLDEATIL
AS
BEGIN
    SELECT TOP 3 FIRSTNAME,LASTNAME,SALARY,JOININGDATE,DEPARTMENTNAME,
    DESIGNATIONNAME
    FROM PERSON P   INNER JOIN DEPARTMENT D
    ON P.DEPARTMENTID=D.DEPARTMENTID
    INNER JOIN DESIGNATION DSG
    ON P.DESIGNATIONID=DSG.DESIGNATIONID
END
```

Part – B:

1. Create a Procedure that takes the department name as input and returns a table with all workers working in that department.

```
CREATE PROCEDURE DEPARTMENT_INPUT
@DEPNAME VARCHAR(100)
AS
BEGIN
```

```
          SELECT P.FIRSTNAME FROM PERSON P
          INNER JOIN DEPARTMENT D
          ON D.DEPARTMENTID = P.DEPARTMENTID
          WHERE D.DEPARTMENTNAME = @DEPNAME
      END
```

2. Create Procedure that takes department name & designation name as input and returns a table with worker's first name, salary, joining date & department name.

```
CREATE PROCEDURE DEPTDESIG_INPUT
@DEPTNAME VARCHAR(100),
@DESIGNAME VARCHAR(50)
AS
BEGIN
    SELECT P.FIRSTNAME,P.SALARY,P.JOININGDATE,D.DEPARTMENTNAME FROM PERSON P
    INNER JOIN DEPARTMENT D
    ON D.DEPARTMENTID = P.DEPARTMENTID
    INNER JOIN DESIGNATION DESIG
    ON P.DESIGNATIONID = DESIG.DESIGNATIONID
    WHERE D.DEPARTMENTNAME = @DEPTNAME AND DESIG.DESIGNATIONNAME = @DESIGNAME
END
```

3. Create a Procedure that takes the first name as an input parameter and display all the details of the worker with their department & designation name.

```
CREATE PROCEDURE FIRSTNAME_INPUT
@FN VARCHAR(100)
AS
BEGIN
    SELECT P.WORKERID, P.FIRSTNAME, P.LASTNAME, P.JOININGDATE, P.SALARY,
    D.DEPARTMENTNAME,DESIG.DESIGNATIONNAME
    FROM PERSON P
    INNER JOIN DEPARTMENT D
    ON P.DEPARTMENTID = D.DEPARTMENTID
    INNER JOIN DESIGNATION DESIG
    ON P.DESIGNATIONID = DESIG.DESIGNATIONID
    WHERE P.FIRSTNAME = @FN
END
```

4. Create Procedure which displays department wise maximum, minimum & total salaries.

```
CREATE PROCEDURE DEPTWISE_MAXMINTOTAL
AS
BEGIN
    SELECT D.DEPARTMENTNAME,MAX(SALARY),MIN(SALARY),SUM(SALARY) FROM PERSON P
    INNER JOIN DEPARTMENT D
    ON P.DEPARTMENTID = D.DEPARTMENTID
    GROUP BY DEPARTMENTNAME
END
```

5. Create Procedure which displays designation wise average & total salaries.

```
CREATE PROCEDURE DESIGNATION_AVGTOTALSAL
AS
BEGIN
    SELECT D.DESIGNATIONNAME,MAX(SALARY),MIN(SALARY),SUM(SALARY) FROM PERSON P
    INNER JOIN DESIGNATION D
    ON P.DESIGNATIONID= D.DESIGNATIONID
    GROUP BY D.DESIGNATIONNAME
```

```
END
```

Part – C:

1. Create Procedure that Accepts Department Name and Returns Person Count.

```
CREATE PROCEDURE DEPARTMENTNAME_INPUT
@DN VARCHAR(100)
AS
BEGIN
    SELECT COUNT(WORKERID) FROM PERSON P
    INNER JOIN DEPARTMENT D
    ON P.DEPARTMENTID = D.DEPARTMENTID
    WHERE D.DEPARTMENTNAME = @DN
END
```

2. Create Procedure that accepts Department Name & Designation as a parameter with given test cases and returns a table with FirstName, LastName, Salary, JoiningDate, DepartmentName & Designation.

Department Name	Designation
IT	NULL
NULL	Jobber
IT	Jobber
NULL	NULL

```
CREATE PROCEDURE DEPT_DESG_INPUT
@DEPNAME VARCHAR(100),
@DESIGNAME VARCHAR(100)
AS
BEGIN
    SELECT P.FIRSTNAME,P.LASTNAME,P.SALARY,P.JOININGDATE FROM PERSON P
    INNER JOIN DEPARTMENT D
    ON P.DEPARTMENTID = D.DEPARTMENTID
    INNER JOIN DESIGNATION DESIG
    ON P.DESIGNATIONID = DESIG.DESIGNATIONID
    WHERE (D.DEPARTMENTNAME = @DEPNAME OR @DEPNAME IS NULL)
    AND (DESIG.DESIGNATIONNAME = @DESIGNAME OR @DESIGNAME IS NULL)
END
```

3. Create Procedure that returns DepartmentID, DepartmentName & Count of all person belongs to that department. i.e. 1 | Admin | 2

```
CREATE PROCEDURE DEPTWISE_COUNT
AS
BEGIN
    SELECT D.DepartmentID,D.DepartmentName,COUNT(P.WorkerID) FROM PERSON P
    INNER JOIN DEPARTMENT D
    ON P.DepartmentID = D.DepartmentID
    GROUP BY D.DepartmentID,D.DepartmentName
    ORDER BY D.DepartmentID
END
```

13 **Create table as per following.**

Person

Column_Name	DataType	Constraints
PersonID	Int	Primary Key
PersonName	Varchar (100)	Not Null
Salary	Decimal (8,2)	Not Null
JoiningDate	Datetime	Not Null
City	Varchar (100)	Not Null
Age	Int	Null
BirthDate	Datetime	Not Null

PersonLog

Column_Name	DataType	Constraints
PLogID	Int	Primary Key, Auto increment
PersonID	Int	Not Null
PersonName	Varchar (250)	Not Null
Operation	Varchar (50)	Not Null
UpdateDate	Datetime	Not Null

From the above given tables create Triggers:

Part – A:

1. Create a trigger that fires on INSERT, UPDATE and DELETE operation on the Person table to display a message “Record is Affected.”

```
CREATE TRIGGER TRG_PERSON_AFFECTED
ON PERSON
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
   PRINT 'RECORD IS AFFECTED.'
END;
```

2. Create a trigger that fires on INSERT, UPDATE and DELETE operation on the Person table. For that, log all operations performed on the person table into PersonLog.

INSERT

```
CREATE TRIGGER TGR_INSERT_PERSONLOG
ON PERSON
FOR INSERT
AS
BEGIN
     INSERT INTO PERSONLOG(PERSONID,PERSONNAME,OPERATION,UPDATEDATE)
     SELECT PERSONID,PERSONNAME,'INSERT',GETDATE() FROM INSERTED
END
```

UPDATE

```
CREATE TRIGGER TGR_UPDATE_PERSONLOG
ON PERSON
FOR UPDATE
AS
BEGIN
     INSERT INTO PERSONLOG(PERSONID, PERSONNAME,OPERATION,UPDATEDATE)
     SELECT PERSONID, PERSONNAME,'UPDATED', GETDATE() FROM INSERTED
```

```
END
```

DELETE

```
CREATE TRIGGER TGR_DELETE_PERSONLOG
ON PERSON
FOR DELETE
AS
BEGIN
    INSERT INTO PERSONLOG(PERSONID, PERSONNAME,OPERATION,UPDATEDATE)
    SELECT PERSONID, PERSONNAME,'DELETED', GETDATE() FROM DELETED
END
```

Part – B:

1. Create an INSTEAD OF trigger that fires on INSERT, UPDATE and DELETE operation on the Person table. For that, log all operations performed on the person table into PersonLog.

INSERT

```
CREATE TRIGGER TGR_PERSON_INSTEADOF_INSERT
ON PERSON
INSTEAD OF INSERT
AS
BEGIN
    INSERT INTO PERSONLOG(PERSONID,PERSONNAME,OPERATION,UPDATEDATE)
    SELECT PERSONID,PERSONNAME,'INSERTED', GETDATE() FROM INSERTED
END
```

UPDATE

```
CREATE TRIGGER TGR_PERSON_INSTEADOF_UPDATE
ON PERSON
INSTEAD OF UPDATE
AS
BEGIN
    INSERT INTO PERSONLOG(PERSONID,PERSONNAME,OPERATION,UPDATEDATE)
    SELECT PERSONID,PERSONNAME,'UPDATED', GETDATE() FROM INSERTED
END
```

DELETE

```
CREATE TRIGGER TGR_PERSON_INSTEADOF_DELETE
ON PERSON
INSTEAD OF DELETE
AS
BEGIN
    INSERT INTO PERSONLOG(PERSONID,PERSONNAME,OPERATION,UPDATEDATE)
    SELECT PERSONID,PERSONNAME,'DELETED', GETDATE() FROM DELETED
END
```

2. Create a trigger that fires on INSERT operation on the Person table to convert person name into uppercase whenever the record is inserted.

```
CREATE TRIGGER TGR_PERSON_NAMEUPPER_INSET
ON PERSON
AFTER INSERT
AS
BEGIN
    UPDATE PERSON
    SET PERSONNAME = (SELECT UPPER(PERSONNAME) FROM INSERTED)
```

```
          WHERE PERSONID = (SELECT PERSONID FROM INSERTED)
     END
```

Part – C:

1. Create a trigger that fires on INSERT operation on person table, which calculates the age and update that age in Person table.

```
CREATE TRIGGER TGR_PERSON_UPDATE_AGE
ON PERSON
FOR INSERT
AS
BEGIN
     UPDATE PERSON
     SET AGE = (SELECT DATEDIFF(YEAR, BIRTHDATE, GETDATE()) FROM INSERTED)
     WHERE PERSONID = (SELECT PERSONID FROM INSERTED)
END
```

2. Create DELETE trigger on PersonLog table, when we delete any record of PersonLog table it prints 'Record deleted successfully from PersonLog'.

```
CREATE TRIGGER TGR_PERSONLOG_DELETE
ON PERSONLOG
FOR DELETE
AS
BEGIN
      PRINT 'RECORD DELETED SUCCESSFULLY FROM PERSONLOG'
END
```

14 **Create table as per following.**

Products		
Column_Name	**DataType**	**Constraints**
Product_id	Int	Primary Key
Product_Name	Varchar (250)	Not Null
Price	Decimal (10,2)	Not Null

Products		
Product_id	**Product_Name**	**Price**
1	Smatphone	35000
2	Laptop	65000
3	Headphones	5500
4	Television	85000
5	Gaming Console	32000

From the above given tables create Cursors:

Part – A:

1. Create a cursor Product_Cursor to fetch all the rows from a products table.

```
DECLARE
                  @PRODUCTID INT,
                  @PRODUCTNAME VARCHAR(250),
                  @PRICE DECIMAL(10,2);
```

```
DECLARE PRODUCT_CURSOR CURSOR
FOR SELECT
                PRODUCT_ID,
                PRODUCT_NAME,
                PRICE
FROM PRODUCTS;
OPEN PRODUCT_CURSOR;
FETCH NEXT FROM PRODUCT_CURSOR INTO
                @PRODUCTID,
                @PRODUCTNAME,
                @PRICE;
WHILE @@FETCH_STATUS=0
BEGIN
        SELECT @PRODUCTID, @PRODUCTNAME ,@PRICE;
FETCH NEXT FROM PRODUCT_CURSOR INTO
                @PRODUCTID,
                @PRODUCTNAME,
                @PRICE;
END;
CLOSE PRODUCT_CURSOR;
DEALLOCATE PRODUCT_CURSOR;
```

2. Create a cursor Product_Cursor_Fetch to fetch the records in form of ProductID_ProductName. (Example: 1_Smartphone)

```
DECLARE
                @PRODUCTID INT,
                @PRODUCTNAME VARCHAR(250);
DECLARE PRODUCT_CURSOR_FETCH CURSOR
FOR SELECT
                PRODUCT_ID,
                PRODUCT_NAME
FROM PRODUCTS;
OPEN PRODUCT_CURSOR_FETCH;
FETCH NEXT FROM PRODUCT_CURSOR_FETCH INTO
                @PRODUCTID,
                @PRODUCTNAME;
WHILE @@FETCH_STATUS=0
BEGIN
        PRINT CAST(@PRODUCTID AS VARCHAR(5)) + '-' + @PRODUCTNAME;
FETCH NEXT FROM PRODUCT_CURSOR_FETCH INTO
        @PRODUCTID,
        @PRODUCTNAME;
END;
CLOSE PRODUCT_CURSOR_FETCH;
DEALLOCATE PRODUCT_CURSOR_FETCH;
```

3. Create a cursor Product_CursorDelete that deletes all the data from the Products table.

```
DECLARE @PRODUCTID INT;
DECLARE PRODUCT_CURSORDELETE CURSOR
FOR SELECT
                PRODUCT_ID FROM PRODUCTS;
```

```
OPEN PRODUCT_CURSORDELETE;
FETCH NEXT FROM PRODUCT_CURSORDELETE INTO
                @PRODUCTID;
WHILE @@FETCH_STATUS=0
BEGIN
        DELETE FROM PRODUCTS
        WHERE PRODUCT_ID=@PRODUCTID;
FETCH NEXT FROM PRODUCT_CURSORDELETE INTO
                @PRODUCTID;
END;
CLOSE PRODUCT_CURSORDELETE;
DEALLOCATE PRODUCT_CURSORDELETE;
```

Part – B:

1. Create a cursor Product_CursorUpdate that retrieves all the data from the products table and increases the price by 10%.

```
DECLARE
                @PRODUCTID INT,
                @PRODUCTNAME VARCHAR(250),
                @PRICE DECIMAL(10,2);
DECLARE PRODUCT_CURSORUPDATE CURSOR
FOR SELECT
                PRODUCT_ID,PRODUCT_NAME,PRICE
FROM PRODUCTS;
OPEN PRODUCT_CURSORUPDATE;
FETCH NEXT FROM PRODUCT_CURSORUPDATE INTO
                @PRODUCTID,
                @PRODUCTNAME,
                @PRICE;
WHILE @@FETCH_STATUS=0
BEGIN
        SET @PRICE=@PRICE*0.10+ @PRICE
        UPDATE PRODUCTS
        SET PRICE=@PRICE
        WHERE PRODUCT_ID=@PRODUCTID;
SELECT @PRODUCTID AS PRODUCT_ID,@PRODUCTNAME AS
PRODUCT_NAME,@PRICE AS UPDATED_PRICE;
FETCH NEXT FROM PRODUCT_CURSORUPDATE INTO
                @PRODUCTID,
                @PRODUCTNAME,
                @PRICE;
END;
CLOSE PRODUCT_CURSORUPDATE;
DEALLOCATE PRODUCT_CURSORUPDATE;
```

Part – C:

1. Create a cursor to insert details of Products into the NewProducts table if the product is “Laptop” (Note: Create NewProducts table first with same fields as Products table)

```
DECLARE
            @PRODUCTID INT,
            @PRODUCTNAME VARCHAR(250),
            @PRICE DECIMAL(10,2);
DECLARE PRODUCT_CURSOR_LAPTOPINSERT CURSOR
FOR SELECT
            PRODUCT_ID,
            PRODUCT_NAME,
            PRICE
FROM PRODUCTS;
OPEN PRODUCT_CURSOR_LAPTOPINSERT;
FETCH NEXT FROM PRODUCT_CURSOR_LAPTOPINSERT INTO
            @PRODUCTID,
            @PRODUCTNAME,
            @PRICE;
WHILE @@FETCH_STATUS=0
BEGIN
      IF @PRODUCTNAME='LAPTOP'
       INSERT INTO NEWPRODUCTS VALUES
       (@PRODUCTID,@PRODUCTNAME,@PRICE)
FETCH NEXT FROM PRODUCT_CURSOR_LAPTOPINSERT INTO
            @PRODUCTID,
            @PRODUCTNAME,
            @PRICE;
END;
CLOSE PRODUCT_CURSOR_LAPTOPINSERT;
DEALLOCATE PRODUCT_CURSOR_LAPTOPINSERT;
```

15 **User Defined Functions**

Part – A:

1. Write a function to print "hello world".

```
CREATE FUNCTION FN_HELLO()
RETURNS VARCHAR(50)
AS
BEGIN
   RETURN 'HELLO WORLD'
END
```

2. Write a function which returns addition of two numbers.

```
CREATE FUNCTION FN_ADDITION( @A INT,@B INT)
RETURNS INT
AS
BEGIN
  RETURN (@A+@B)
END
```

3. Write a function to print a cube of a given number.

```
CREATE FUNCTION FN_CUBE(@A INT)
RETURNS INT
```

```
AS
BEGIN
    RETURN(@A * @A * @A)
END
```

4. Write a function to check whether the given number is ODD or EVEN.

```
CREATE FUNCTION FN_ODDEVEN(@A INT)
RETURNS VARCHAR(50)
AS
BEGIN
            DECLARE @MSG VARCHAR(50)
             IF(@A%2=0)
             SET @MSG='THE NUMBER IS EVEN'
             ELSE
             SET @MSG='THE NUMBER IS ODD'
            RETURN @MSG
END
```

5. Write a function which returns a table with details of a person whose first name starts with B.

```
CREATE FUNCTION FNTB_FIRSTNAMEB()
RETURNS TABLE
AS
RETURN(           SELECT * FROM PERSON
                  WHERE FirstName LIKE 'B%')
```

6. Write a function which returns a table with unique first names from the person table.

```
CREATE FUNCTION FNTB_UNIQUEFNAME()
RETURNS TABLE
AS
RETURN(SELECT DISTINCT FirstName FROM PERSON)
```

Part – B:

1. Write a function to compare two integers and return the comparison result. (Using Case statement)

```
CREATE FUNCTION FN_CASECOMPARE(@A INT,@B INT)
RETURNS VARCHAR(50)
AS
BEGIN
     DECLARE @MSG VARCHAR(50)=
             CASE
               WHEN @A>@B THEN 'A IS GREATER THEN B'
                      WHEN @A<@B THEN 'B IS GREATER THEN A'
                      ELSE 'A IS EQUAL TO B'
             END
             RETURN @MSG

END
```

2. Write a function to print number from 1 to N. (Using while loop)

```
CREATE OR ALTER FUNCTION FN_PRINTNUMBER(@NUM INT)
RETURNS VARCHAR(500)
AS
BEGIN
   DECLARE @MSG VARCHAR(500),@I INT
            SET @MSG=''
            SET @I=1
```

```
                WHILE(@I<=@NUM)
                BEGIN
                SET @MSG=CONCAT(@MSG,' ',@I)
                SET @I=@I+1
               END
               RETURN @MSG
        END
```

3. Write a function to print the sum of even numbers between 1 to 20.

```
CREATE FUNCTION FN_SUMEVEN()
RETURNS INT
AS
BEGIN
  DECLARE @SUM INT,@I INT
        SET @SUM=0
        SET @I=1

         WHILE(@I<=20)
         BEGIN
           IF(@I%2=0)
                SET @SUM=@SUM+@I
           SET @I=@I+1
        END
        RETURN @SUM
END
```

Part – C:

1. Write a function to check whether a given number is prime or not.

```
CREATE FUNCTION FN_PRIMENUM(@NUM INT)
RETURNS VARCHAR(50)
AS
BEGIN
   DECLARE @MSG VARCHAR(50),@I INT,@FLAG INT
         SET @I=2
         SET @FLAG=1
         WHILE(@I<@NUM)
         BEGIN
           IF(@NUM%@I=0)
                SET @FLAG=0
                BREAK
         END
         IF(@FLAG=1)
         SET @MSG='PRIME'
         ELSE
         SET @MSG='NOT PRIME'
         RETURN @MSG
END
```

2. Write a function which accepts two parameters start date & end date, and returns a difference in days.

```
CREATE OR ALTER FUNCTION FN_DATEDIFF(@STARTDATE DATETIME,@ENDDATE DATETIME)
RETURNS INT
AS
BEGIN
DECLARE @DAYS INT
```

```
     SET @DAYS=(SELECT DATEDIFF(DAY,@STARTDATE,@ENDDATE))
          RETURN @DAYS
END
```

3. Write a function which accepts two parameters year & month in integer and returns total days in a given month & year.

```
CREATE FUNCTION DaysInMonth(@year INT, @month INT)
RETURNS INT
AS
BEGIN
  RETURN DAY(EOMONTH(CONCAT(@year, '-', @month, '-01')));
END
```

4. Write a function which accepts departmentID as a parameter & returns a detail of the persons.

```
CREATE FUNCTION FNTB_DEPTID_DETIALS
(@DEPTID INT)
RETURNS TABLE
AS
RETURN(  SELECT * FROM PERSON
          WHERE DepartmentID = @DEPTID)
```

16

Part – A: Create Database with Name: *BANK_INFO* and Create collection as per following.

DEPOSIT

ACTNO	CNAME	BNAME	AMOUNT	ADATE
101	ANIL	VRCE	1000.00	1-3-95
102	SUNIL	AJNI	5000.00	4-1-96
103	MEHUL	KAROLBAGH	3500.00	17-11-95
104	MADHURI	CHANDI	1200.00	17-12-95
105	PRMOD	M.G. ROAD	3000.00	27-3-96
106	SANDIP	ANDHERI	2000.00	31-3-96
107	SHIVANI	VIRAR	1000.00	5-9-95
108	KRANTI	NEHRU PLACE	5000.00	2-7-95

From the above given collection perform the following queries in MongoDB:

1. Retrieve/Display every document of Deposit collection.

```
db.DEPOSIT.find()
```

2. Retrieve/Display every document of Deposit collection. (Use: pretty())

```
db.DEPOSIT.find().pretty()
```

3. Display only one document of Deposit collection. (Use: findOne())

```
db.DEPOSIT.findOne()
```

4. Insert following document to Deposit collection. (Use: insertOne())

109	KIRTI	VIRAR	3000.00	3-5-97

```
db.DEPOSIT.insertOne({
  ACTNO: 109,
  CNAME: "KIRTI",
  BNAME: "VIRAR",
  AMOUNT: 3000.00,
  ADATE: new Date("1997-05-03")
})
```

5. Insert following documents to your collection. (Use: insertMany())

110	MITALI	ANDHERI	4500.00	4-9-95
111	RAJIV	NEHRU PLACE	7000.00	2-10-98

```
db.DEPOSIT.insertMany([
  {
    ACTNO: 110,
    CNAME: "MITALI",
    BNAME: "ANDHERI",
    AMOUNT: 4500.00,
    ADATE: new Date("1995-09-04")
  },
  {
    ACTNO: 111,
    CNAME: "RAJIV",
    BNAME: "NEHRU PLACE",
    AMOUNT: 7000.00,
    ADATE: new Date("1998-10-02")
  }
])
```

6. Display documents with CNAME, BNAME and AMOUNT fields.
   ```
   db.DEPOSIT.find({}, { CNAME: 1, BNAME: 1, AMOUNT: 1, _id: 0 }).pretty()
   ```
7. Display every document of Deposit collection on ascending order by CNAME.
   ```
   db.DEPOSIT.find().sort({ CNAME: 1 }).pretty()
   ```
8. Display every document of Deposit collection on descending order by BNAME.
   ```
   db.DEPOSIT.find().sort({ BNAME: -1 }).pretty()
   ```
9. Display every document of Deposit collection on ascending order by ACTNO and descending order by AMOUNT.
   ```
   db.DEPOSIT.find().sort({ ACTNO: 1, AMOUNT: -1 }).pretty()
   ```
10. Display only two documents of Deposit collection.
    ```
    db.DEPOSIT.find().limit(2).pretty()
    ```
11. Display 3rd document of Deposit collection.
    ```
    db.DEPOSIT.find().skip(2).limit(1).pretty()
    ```
12. Display 6th and 7th documents of Deposit collection.
    ```
    db.DEPOSIT.find().skip(5).limit(2).pretty()
    ```
13. Display the count of documents in Deposit collection.
    ```
    db.DEPOSIT.countDocuments()
    ```
14. Display only first documents of Deposit collection.
    ```
    db.DEPOSIT.findOne()
    ```
15. Display every document of Deposit collection on descending order by AMOUNT.
    ```
    db.DEPOSIT.find().sort({ AMOUNT: -1 }).pretty()
    ```

Part- B:

1. Insert following document to Deposit collection. (Use: insertOne())

112	MANISH	ANDHERI	8000.00	9-5-98

   ```
   db.DEPOSIT.insertOne({ACTNO: 112,CNAME: "MANISH",BNAME: "ANDHERI",AMOUNT: 8000.00,
     ADATE: new Date("1998-05-09")
   })
   ```
2. Display 9th document of Deposit collection.
   ```
   db.DEPOSIT.find().skip(8).limit(1).pretty()
   ```
3. Display 11th and 12th documents of Deposit collection.
   ```
   db.DEPOSIT.find().skip(10).limit(2).pretty()
   ```

Part – C:

1. Display every document of Deposit collection on ascending order by AMOUNT and descending order by BNAME.
   ```
   db.DEPOSIT.find().sort({ AMOUNT: 1, BNAME: -1 }).pretty()
   ```
2. Display only five documents of Deposit collection.
   ```
   db.DEPOSIT.find().limit(5).pretty()
   ```
3. Delete all the documents of collection Deposit.
   ```
   db.DEPOSIT.deleteMany({})
   ```
4. Drop BANK_INFO database.
   ```
   db.dropDatabase()
   ```

17

Part – A: Create collection as per following.

EMPLOYEE

EID	ENAME	GENDER	JOININGDATE	SALARY	CITY
1	Nick	Male	01-JAN-13	4000	London
2	Julian	Female	01-OCT-14	3000	New York
3	Roy	Male	01-JUN-16	3500	London
4	Tom	Male	NULL	4500	London
5	Jerry	Male	01-FEB-13	2800	Sydney
6	Philip	Male	01-JAN-15	7000	New York
7	Sara	Female	01-AUG-17	4800	Sydney
8	Emily	Female	01-JAN-15	5500	New York
9	Michael	Male	NULL	6500	London
10	John	Male	01-JAN-15	8800	London

From the above given collection perform the following queries in MongoDB:

1. Display employees whose gender is Male.
   ```
   db.Employee.find({GENDER:'Male'})
   ```
2. Display employees who belong to London city.
   ```
   db.Employee.find({CITY:'London'})
   ```
3. Display employees whose salary is greater than 3500.
   ```
   db.Employee.find({SALARY:{$gt:3500}})
   ```
4. Display employees whose joining date is before 2015-01-01.
   ```
   db.Employee.find({JOININGDATE:{$lt:new Date('2015-01-01')}})
   ```
5. Display employees whose EID is greater than or equal to 7.
   ```
   db.Employee.find({EID:{$gte:7}})
   ```
6. Display employees whose city is Landon or New York (use:IN)
   ```
   db.Employee.find({CITY:{$in:['London','New York']}})
   ```
7. Display employees who do not belongs to Landon or New York (use: NOT IN)
   ```
   db.Employee.find({CITY:{$nin:['London','New York']}})
   ```
8. Display the EID of those employee who lives in city London.
   ```
   db.Employee.find({CITY:{$eq:'London'}},{EID:1})
   ```
9. Display first 2 employee names who lives in New york.
   ```
   db.Employee.find({CITY:{$eq:'New york'}},{ENAME:1}).limit(2)
   ```
10. Display next 2 employee after skipping first 2 whose city is London.
    ```
    db.Employee.find({CITY:{$eq:'London'}}).skip(2).limit(2)
    ```
11. Display Male employees who lives Sydney.
    ```
    db.Employee.find({$and:[{GENDER:{'Male'},{CITY:{'Sydney'}]})
    ```
12. Display EID, ENAME, CITY and SALARY of those employees who belongs to London or Sydney.

```
db.Employee.find({CITY:{$in:['London','NewYork']}},{EID:1,ENAME:1,CITY:1, SALARY:1})
```

13. Display ENAME, SALARY, and CITY of those employee whose salary is more than 7000.
    ```
    db.Employee.find({SALARY:{$gt:7000}},{ENAME:1,SALARY:1,CITY:1})
    ```
14. Display documents whose name start with E.
    ```
    db.Employee.find({ENAME:/^E/})
    ```
15. Display documents whose name starts with S or M in your collection.
    ```
    db.Employee.find({ENAME:/^[S,M]/})
    ```
16. Display documents where city starts with A to M in your collection.
    ```
    db.Employee.find({CITY:/^[A-M]/})
    ```
17. Display documents where city name ends in ‘ney’.
    ```
    db.Employee.find({CITY:/ney$/})
    ```
18. Display employee info whose name contains n. (Both uppercase(N) and lowercase(n))
    ```
    db.Employee.find({ENAME:/n/i})
    ```
19. Display employee info whose name starts with E and having 5 characters.
    ```
    db.Employee.find({ENAME:/^E..../})
    ```
20. Display employee whose name start with S and ends in a.
    ```
    db.Employee.find({$and:[{ENAME:/^S/},{ENAME:/a$/}]})
    ```
21. Display EID, ENAME, CITY and SALARY whose name starts with ‘Phi’.
    ```
    db.Employee.find({ENAME:/^Phi/},{EID:1,ENAME:1,CITY:1,SALARY:1})
    ```
22. Display ENAME, JOININGDATE and CITY whose city contains ‘dne’ as three letters in city name.
    ```
    db.Employee.find({CITY:/.dne./i},{ENAME:1,JOININGDATE:1,CITY:1})
    ```
23. Display ENAME, JOININGDATE and CITY who does not belongs to city London or Sydney.
    ```
    db.Employee.find({CITY:{$nin:['London','Sydney']}},{ENAME:1,CITY:1, JOININGDATE:1})
    ```
24. Delete the documents whose city is New York.
    ```
    db.Employee.deleteMany({CITY:{$eq:'New York'}})
    ```
25. Update ENAME of Nick to ‘Naysa’ and GENDER to ‘Female’.
    ```
    db.Employee.updateMany({ENAME:'NICK'},{$set:{ENAME:'Naysa',GENDER: 'Female'}})
    ```

Part – B: Create collection as per following.

STUDENT

ROLLNO	SNAME	DEPARTMENT	FEES	SEM	GENDER	CITY
101	Vina	CE	15000	3	Female	Rajkot
102	Krisha	EC	8000	5	Female	Ahmedabad
103	Priti	Civil	12000	7	Female	Baroda
104	Mitul	CE	15000	3	Male	Rajkot
105	Keshav	CE	15000	3	Male	Jamnagar
106	Zarna	Civil	12000	5	Female	Ahmedabad
107	Nima	EE	9000	5	Female	Rajkot
108	Dhruv	Mechanical	10000	5	Male	Rajkot
109	Krish	Mechanical	10000	7	Male	Baroda
110	Zeel	EE	9000	3	Female	Jamnagar

From the above given collection perform the following queries in MongoDB:

1. Display Female students.
   ```
   db.STUDENT.find({ "GENDER": "Female" })
   ```
2. Display students who belong to Rajkot city.
   ```
   db.STUDENT.find({ "CITY": "Rajkot" })
   ```
3. Display students studying in 7th sem.
   ```
   db.STUDENT.find({ "SEM": 7 })
   ```
4. Display students not studying in 3rd sem.
   ```
   db.STUDENT.find({$ne :{ "SEM": 3 }})
   ```

5. Display students whose roll no is greater than 107.
   ```
   db.STUDENT.find({"ROLLNO" :{ $gt: 107 }})
   ```
6. Display students whose city is Jamnagar or Baroda (use:IN)
   ```
   db.STUDENT.find({"CITY" : {$in: ["Jamnagar","Baroda"]} })
   ```
7. Display students whose fees is less than 9000.
   ```
   db.STUDENT.find({"FEES" :{ $lt: 9000 }})
   ```
8. Display the roll no of those students who belongs to Mechanical department.
   ```
   db.STUDENT.find({ "DEPARTMENT": "Mechanical" }, { "ROLLNO": 1, "_id": 0 })
   ```
9. Display first 2 students names who lives in Baroda.
   ```
   db.STUDENT.find({ "CITY": "Baroda" }, { "SNAME": 1, "_id": 0 }).limit(2)
   ```
10. Display Male students who studying in 3rd sem.
    ```
    db.STUDENT.find({ "GENDER": "Male", "SEM": 3 })
    ```
11. Display sname and city and fees of those students whose roll no is less than 105.
    ```
    db.STUDENT.find({ "ROLLNO":{$lt: 105}},{"SNAME": 1, "CITY": 1, "FEES": 1, "_id": 0 })
    ```
12. Display documents where sname start with K.
    ```
    db.STUDENT.find({ "SNAME":  /^K/ }})
    ```
13. Display documents where sname starts with Z or D in your collection.
    ```
    db.STUDENT.find({ "SNAME": /^[ZD]/})
    ```
14. Display documents where city starts with A to R in your collection.
    ```
    db.STUDENT.find({ "CITY": /^[AR]/})
    ```
15. Display students' info whose name start with P and ends in i.
    ```
    db.STUDENT.find({ "SNAME":/^P.*i$/ })
    ```
16. Display students' info whose department name starts with 'C'.
    ```
    db.STUDENT.find({ "DEPARTMENT": /^C/ })
    ```
17. Display name, sem, fees, and department whose city contains 'med' as three letters somewhere in city name.
    ```
    db.STUDENT.find({ "CITY": /.med./ },{ "SNAME": 1, "SEM": 1, "FEES": 1, "DEPARTMENT": 1,
    "_id": 0 })
    ```
18. Display name, sem, fees, and department who does not belongs to city Rajkot or Baroda.
    ```
    db.STUDENT.find({"CITY": { $nin: ["Rajkot", "Baroda"]}},{"SNAME": 1, "SEM": 1, "FEES": 1,
    "DEPARTMENT": 1, "_id": 0 })
    ```
19. Delete the documents whose city is Jamnagar.
    ```
    db.STUDENT.deleteMany({ "CITY": "Jamnagar" })
    ```
20. Update sname of Krish to 'fenny' and gender to 'Female'.
    ```
    db.STUDENT.updateOne({ "SNAME": "Krish" },{ $set:{"SNAME": "Fenny", "GENDER": "Female" }})
    ```

Part – C:

1. Display next 2 students after skipping first 2 whose city is Ahmedabad.
   ```
   db.STUDENT.find({ "CITY": "Ahmedabad" }).skip(2).limit(2)
   ```
2. Display rollno, sname, fees, and department of those students who is from Baroda and belongs to CE department.
   ```
   db.STUDENT.find({ "CITY": "Baroda", "DEPARTMENT": "CE" },{ "ROLLNO": 1, "SNAME": 1, "FEES": 1,
   "DEPARTMENT": 1, "_id": 0 })
   ```
3. Display documents where city name ends in 'oda'.
   ```
   db.STUDENT.find({ "CITY": /oda$/ })
   ```
4. Display students' info whose name contains v. (Both uppercase(V) and lowercase(v))
   ```
   db.STUDENT.find({ "SNAME": /v/i  })
   ```
5. Display students' info whose name starts with V and having 4 characters.
   ```
   db.STUDENT.find({ "SNAME": /^V.../ })
   ```

2

18

Create collection as per following.

DEPOSIT

ACTNO	CNAME	BNAME	AMOUNT	ADATE
101	ANIL	VRCE	1000.00	1-3-95
102	SUNIL	AJNI	5000.00	4-1-96
103	MEHUL	KAROLBAGH	3500.00	17-11-95
104	MADHURI	CHANDI	1200.00	17-12-95
105	PRMOD	M.G. ROAD	3000.00	27-3-96
106	SANDIP	ANDHERI	2000.00	31-3-96
107	SHIVANI	VIRAR	1000.00	5-9-95
108	KRANTI	NEHRU PLACE	5000.00	2-7-95
109	MINU	POWAI	7000.00	10-8-95

BRANCH

BNAME	CITY
VRCE	NAGPUR
AJNI	NAGPUR
KAROLBAGH	DELHI
CHANDI	DELHI
DHARAMPETH	NAGPUR
M.G. ROAD	BANGLORE
ANDHERI	BOMBAY
VIRAR	BOMBAY
NEHRU PLACE	DELHI
POWAI	BOMBAY

CUSTOMERS

CNAME	CITY
ANIL	CALCUTTA
SUNIL	DELHI
MEHUL	BARODA
MANDAR	PATNA
MADHURI	NAGPUR
PRAMOD	NAGPUR
SANDIP	SURAT
SHIVANI	BOMBAY
KRANTI	BOMBAY
NAREN	BOMBAY

BORROW

LOANNO	CNAME	BNAME	AMOUNT
201	ANIL	VRCE	1000.00
206	MEHUL	AJNI	5000.00
311	SUNIL	DHARAMPETH	3000.00
321	MADHURI	ANDHERI	2000.00
375	PRMOD	VIRAR	8000.00
481	KRANTI	NEHRU PLACE	3000.00

From the above given tables perform the following queries in MongoDB:
Part A:

1. Retrieve all data from table DEPOSIT.
   db.DEPOSIT.find({});
2. Retrieve all data from table BORROW.
   db.BORROW.find({});
3. Retrieve all data from table CUSTOMERS.
   db.CUSTOMERS.find({});
4. Insert a record (550,’JAY’,’AJNI’,NULL)in the BORROW table.
   db.BORROW.insertOne({ LOANNO: 550, CNAME: "JAY", BNAME: "AJNI", AMOUNT: null });
5. Display Account No, Customer Name & Amount from DEPOSIT.
   db.DEPOSIT.find({}, { ACTNO: 1, CNAME: 1, AMOUNT: 1 });
6. Display Loan No, Amount from BORROW.
   db.BORROW.find({}, { LOANNO: 1, AMOUNT: 1 });
7. Display loan details of all customers who belongs to ‘ANDHERI’ branch.
   db.BORROW.find({ BNAME: "ANDHERI" });
8. Give account no and amount of depositor, whose account no is equals to 106.
   db.DEPOSIT.find({ ACTNO: 106 }, { ACTNO: 1, AMOUNT: 1 });
9. Give name of borrowers having amount greater than 5000.
   db.BORROW.find({ AMOUNT: { $gt: 5000 } }, { CNAME: 1 });
10. Give name of customers who opened account after date '1-12-96'.
    db.DEPOSIT.find({ ADATE: { $gt: new Date('1996-12-01') } }, { CNAME: 1 });

Part B:

11. Display name of customers whose account no is less than 105.
    db.DEPOSIT.find({ ACTNO: { $lt: 105 } }, { CNAME: 1, _id: 0 });
12. Display name of customer who belongs to either ‘NAGPUR’ Or ‘DELHI’.
    db.CUSTOMERS.find({ CITY: { $in: ["NAGPUR", "DELHI"] } }, { CNAME: 1, _id: 0 });
13. Display name of customers with branch whose amount is greater than 4000 and account no is less than 105.
    db.DEPOSIT.find({ AMOUNT: { $gt: 4000 }, ACTNO: { $lt: 105 } }, { CNAME: 1, BNAME: 1, _id: 0 });
14. Find all borrowers whose amount is greater than equals to 3000 & less than equals to 8000.
    db.BORROW.find({ AMOUNT: { $gte: 3000, $lte: 8000 } });
15. Find all depositors who do not belongs to ‘ANDHERI’ branch.
    db.DEPOSIT.find({ BNAME: { $ne: "ANDHERI" } });
16. Display Account No, Customer Name & Amount of such customers who belongs to ‘AJNI’, ‘KAROLBAGH’ Or ‘M.G.ROAD’ and Account No is less than 104.
    db.DEPOSIT.find({ BNAME: { $in: ["AJNI", "KAROLBAGH", "M.G. ROAD"] }, ACTNO: { $lt: 104 } },
    { ACTNO: 1, CNAME: 1, AMOUNT: 1, _id: 0 });
17. Display all the details of first five customers.
    db.CUSTOMERS.find().limit(5);
18. Display all the details of first three depositors whose amount is greater than 1000.
    db.DEPOSIT.find({ AMOUNT: { $gt: 1000 } }).limit(3);

19. Display Loan No, Customer Name of first five borrowers whose branch name does not belongs to 'ANDHERI'.

```
db.BORROW.find({ BNAME: { $ne: "ANDHERI" } }, { LOANNO: 1, CNAME: 1, _id: 0 }).limit(5);
```

20. Retrieve all unique cities using DISTINCT. (Use **Customers collection)**

```
db.CUSTOMERS.distinct("CITY");
```

Part C:

21. Retrieve all unique branches using DISTINCT. (Use **Branch collection)**

```
db.BRANCH.distinct("BNAME");
```

22. Retrieve all the records of customer table as per their city name in ascending order.

```
db.CUSTOMERS.find().sort({ CITY: 1 });
```

23. Retrieve all the records of deposit table as per their amount column in descending order.

```
db.DEPOSIT.find().sort({ AMOUNT: -1 });
```

24. Update deposit amount of all customers from 3000 to 5000.

```
db.DEPOSIT.updateMany({ AMOUNT: 3000 }, { $set: { AMOUNT: 5000 } });
```

25. Change branch name of ANIL from VRCE to C.G. ROAD. (Use **Borrow collection**)

```
db.BORROW.updateMany({ CNAME: "ANIL", BNAME: "VRCE" }, { $set: { BNAME: "C.G. ROAD" } });
```

26. Update Account No of SANDIP to 111 & Amount to 5000.

```
db.DEPOSIT.updateMany({ CNAME: "SANDIP" }, { $set: { ACTNO: 111, AMOUNT: 5000 } });
```

27. Give 10% Increment in Loan Amount.

```
db.BORROW.updateMany({}, { $mul: { AMOUNT: 1.1 } });
```

28. Update deposit amount of all depositors to 5000 whose account no between 103 & 107.

```
db.DEPOSIT.updateMany({ ACTNO: { $gte: 103, $lte: 107 } }, { $set: { AMOUNT: 5000 } });
```

29. Update amount of loan no 321 to *NULL*.

```
db.BORROW.updateMany({ LOANNO: 321 }, { $set: { AMOUNT: null } });
```

30. Display the name of borrowers whose amount is *NULL*.

```
db.BORROW.find({ AMOUNT: null }, { CNAME: 1, _id: 0 });
```

19

Create collection as per following.

STUDENT

RollNo	Name	Birthdate	SPI	City	Backlog	Branch
101	Keyur	5-1-92	8.5	Rajkot	2	CE
102	Hardik	15-2-94	9.0	Ahmedabad	0	CE
103	Kajal	14-3-96	10.00	Baroda	0	IT
104	Bhoomi	23-6-95	8.90	Ahmedabad	1	ICT
105	Harmit	15-2-94	9.80	Rajkot	1	IT
106	Jay	15-2-94	7.9	Rajkot	2	CE

From the above given tables perform the following queries in MongoDB:

Part A:

1. Give RollNo and Name of students, whose RollNo is greater than 103 and backlog is greater than 0 and branch is either CE or IT.

```
db.STUDENT.find( { RollNo: { $gt: 103 }, Backlog: { $gt: 0 }, Branch: { $in: ["CE", "IT"] }},
 {RollNo: 1,Name: 1, _id: 0 })
```

2. Give name of students whose SPI is between 8 and 9 and branch is either CE or IT.

```
db.STUDENT.find( {SPI: { $gte: 8, $lte: 9 }, Branch: { $in: ["CE", "IT"] }}, {Name: 1, _id: 0  })
```

3. Find all students who do not belongs to 'CE' branch.
   ```
   db.STUDENT.find({ Branch: { $ne: "CE" }});
   ```
4. Display RollNo and Name of first three students.
   ```
   db.STUDENT.find({},{ RollNo: 1,  Name: 1,_id: 0}).limit(3);
   ```
5. Display all the details of first three students whose SPI is greater than 8.5.
   ```
   db.STUDENT.find({SPI: { $gt: 8.5 } }).limit(3);
   ```
6. Retrieve all unique cities using DISTINCT.
   ```
   db.STUDENT.distinct("City");
   ```
7. Retrieve all unique branches using DISTINCT.
   ```
   db.STUDENT.distinct("Branch");
   ```
8. Retrieve all the records of student table as per their Backlog in descending order and then SPI in ascending order.
   ```
   db.STUDENT.find().sort({ Backlog: -1, SPI: 1 })
   ```
9. Update the branch and city of Jay to MCA and Jamangar respectively.
   ```
   db.STUDENT.updateOne({ Name: "Jay" }, { $set: {Branch: "MCA",City: "Jamnagar"}});
   ```
10. Update the backlog of Keyur and Bhoomi to *NULL*.
    ```
    db.STUDENT.updateMany({Name: { $in: ["Keyur", "Bhoomi"] }},{$set: {Backlog: null}});
    ```

Part B:

1. Display the name of students whose backlog is *NULL* and backlog is greater than 1 and branch is either CE or IT.
   ```
   db.STUDENT.find({ $or: [{ Backlog: null },{ Backlog: { $gt: 1 } }  ], Branch: { $in: ["CE", "IT"] }},{Name: 1, _id: 0})
   ```
2. Remove field Backlog for the students having 0 backlog.
   ```
   db.STUDENT.updateMany({ Backlog: 0 },{ $unset: { Backlog: "" } });
   ```
3. Add new field mobile number in the Keyur's record with the value as '9825052365'
   ```
   db.STUDENT.updateOne( { Name: "Keyur" }, { $set: { mobile_number: "9825052365" } });
   ```
4. Remove birthdate field from all the documents.
   ```
   db.STUDENT.updateMany({},{ $unset: { Birthdate: "" } });
   ```
5. Delete all the Employees who joined after 1-1-2007.
   ```
   db.STUDENT.deleteMany({ Birthdate: { $gt: new Date("2007-01-01") } });
   ```
6. Delete all the records of Employee collection.
   ```
   db.Employee.deleteMany({});
   ```

20 **Create collection as per following.**

EMPLOYEE

EmpNo	EmpName	JoiningDate	Salary	City
101	Keyur	5-1-02	12000.00	Rajkot
102	Hardik	15-2-04	14000.00	Ahmedabad
103	Kajal	14-3-06	15000.00	Baroda
104	Bhoomi	23-6-05	12500.00	Ahmedabad
102	Harmit	15-2-04	14000.00	Rajkot

From the above given tables perform the following queries in MongoDB:

Part A:

1. Display the name of employee whose salary is greater than 13000 and city is either Rajkot or Baroda.

```
db.EMPLOYEE.find( {Salary: { $gt: 13000 }, City: { $in: ["Rajkot", "Baroda"] }, }, { EmpName: 1, _id: 0 });
```

2. Display the name of employee in ascending order by their name.
   ```
   db.EMPLOYEE.find({}, { EmpName: 1, _id: 0 }).sort({ EmpName: 1 });
   ```
3. Retrieve all unique cities.
   ```
   db.EMPLOYEE.distinct("City");
   ```
4. Update the city of Keyur and Bhoomi to *NULL.*
   ```
   db.EMPLOYEE.updateMany({ EmpName: { $in: ["Keyur", "Bhoomi"] } },{ $set: { City: null } });
   ```
5. Display the name of employee whose city is *NULL*.
   ```
   db.EMPLOYEE.find({ City: null }, { EmpName: 1, _id: 0 });
   ```
6. Delete all the records of Employee table having salary greater than and equals to 14000.
   ```
   db.EMPLOYEE.deleteMany({ Salary: { $gte: 14000 } });
   ```
7. Delete all the Employees who belongs to 'RAJKOT' city.
   ```
   db.EMPLOYEE.deleteMany({ City: "Rajkot" });
   ```
8. Delete all the Employees who joined after 1-1-2007.
   ```
   db.EMPLOYEE.deleteMany({ JoiningDate: { $gt: new Date("2007-01-01") } });
   ```
9. Delete all the records of Employee collection.
   ```
   db.EMPLOYEE.deleteMany({ JoiningDate: { $gt: new Date("2007-01-01") } });
   ```
10. Remove Employee collection.
    ```
    db.EMPLOYEE.drop();
    ```

Part B:

ACCOUNT

ACTNO	CNAME	BNAME	AMOUNT	ADATE
101	ANIL	VRCE	1000.00	1-3-95
102	SUNIL	AJNI	5000.00	4-1-96
103	MEHUL	KAROLBAGH	3500.00	17-11-95
104	MADHURI	CHANDI	1200.00	17-12-95
105	PRMOD	M.G. ROAD	3000.00	27-3-96
106	SANDIP	ANDHERI	2000.00	31-3-96
107	SHIVANI	VIRAR	1000.00	5-9-95
108	KRANTI	NEHRU PLACE	5000.00	2-7-95
109	MINU	POWAI	7000.00	10-8-95

From the above given tables perform the following queries in MongoDB:

1. Retrieve all unique BNAME.
   ```
   db.ACCOUNT.distinct("BNAME");
   ```
2. Display the Cname in ascending order by their amount and if amount is same then in descending order by cname.
   ```
   cname.db.ACCOUNT.find({}, { CNAME: 1, _id: 0 }).sort({ AMOUNT: 1, CNAME: -1 });
   ```
3. Update the BNAME of Anil and Shivani to *NULL*.
   ```
   db.ACCOUNT.updateMany({ CNAME: { $in: ["ANIL", "SHIVANI"] } }, { $set: { BNAME: null } });
   ```
4. Display the Cname of customers whose Bname is *NULL*.
   ```
   db.ACCOUNT.find({ BNAME: null }, { CNAME: 1, _id: 0 });
   ```
5. Delete all the records of Account table having amount greater than and equals to 4000.
   ```
   db.ACCOUNT.deleteMany({ AMOUNT: { $gte: 4000 } });
   ```

6. Delete all the accounts Bname is CHANDI.
   db.ACCOUNT.deleteMany({ BNAME: "CHANDI" });
7. Delete all the accounts having adate after 1-10-1995.
   db.ACCOUNT.deleteMany({ ADATE: { $gt: new Date("1995-10-01") } });
8. Delete all the records of Account collection.
   db.ACCOUNT.deleteMany({});
9. Remove Account collection.
   db.ACCOUNT.drop();

Part C:

ACCOUNT

ACTNO	CNAME	BNAME	AMOUNT	ADATE
101	ANIL	VRCE	1000.00	1-3-95
102	SUNIL	AJNI	5000.00	4-1-96
103	MEHUL	KAROLBAGH	3500.00	17-11-95
104	MADHURI	CHANDI	1200.00	17-12-95
105	PRMOD	M.G. ROAD	3000.00	27-3-96
106	SANDIP	ANDHERI	2000.00	31-3-96
107	SHIVANI	VIRAR	1000.00	5-9-95
108	KRANTI	NEHRU PLACE	5000.00	2-7-95
109	MINU	POWAI	7000.00	10-8-95

From the above given tables perform the following queries in MongoDB:

1. Display the Cname whose Bname is either AJNI or CHANDI and amount is greater than 3000 and sort the result in ascending order by their amount and if amount is same then in descending order by cname.
   db.ACCOUNT.find({ BNAME: { $in: ["AJNI", "CHANDI"] },AMOUNT: { $gt: 3000 }, },{ CNAME: 1, AMOUNT: 1, _id: 0 }).sort({ AMOUNT: 1, CNAME: -1 });
2. Retrieve top 3 unique BNAME and sort them in ascending order on BNAME.
   db.ACCOUNT.distinct("BNAME").sort().slice(0, 3);
3. Display the Cname whose ACTNO is greater than 103 and sort the result in ascending order by their amount and if amount is same then in descending order by cname.
   db.ACCOUNT.find({ ACTNO: { $gt: 103 } }, { CNAME: 1, AMOUNT: 1, _id: 0 }).sort({ AMOUNT: 1, CNAME: -1});
4. Update the BNAME of Anil, Mehul and Shivani to *NULL*.
   db.ACCOUNT.updateMany({ CNAME: { $in: ["ANIL", "MEHUL", "SHIVANI"] } },{ $set: { BNAME: null } });
5. Display the Cname of customers whose Bname is *NULL*.
   db.ACCOUNT.find({ BNAME: null }, { CNAME: 1, _id: 0 });
6. Update the amount of Anil to 5000.
   db.ACCOUNT.updateOne({ CNAME: "ANIL" }, { $set: { AMOUNT: 5000 } });
7. Update amount of actno 109 to *NULL*.
   db.ACCOUNT.updateOne({ ACTNO: 109 }, { $set: { AMOUNT: null } });
8. Retrieve all the records of account table as per their bname in descending order.
   db.ACCOUNT.find().sort({ BNAME: -1 });
9. Delete all the records of Account collection.
   db.ACCOUNT.deleteMany({});

	10. Remove Account collection. db.ACCOUNT.drop();
21	**From the above given collection perform the following queries in MongoDB:** **Part – A: (Use EMPLOYEE collection of Lab No 17)** 1. Display distinct city. db.EMPLOYEE.distinct('CITY') 2. Display city wise number of persons. db.Employee.aggregate([{$group:{_id:"$CITY",person_count:{$sum:1}}}]) 3. Display sum of salary in your collection. db.Employee.aggregate([{$group:{_id:null,total:{$sum:"$SALARY"}}}]) 4. Display average of salary in your document. db.Employee.aggregate([{$group:{_id:null,avg:{$avg:"$SALARY"}}}]) 5. Display maximum and minimum salary of your document. db.Employee.aggregate([{$group:{_id:null,max:{$max:"$SALARY"},min:{$min:" $SALARY"}}}]) 6. Display city wise total salary in your collection. db.Employee.aggregate([{$group:{_id:"$CITY",total_salary_citywise:{$sum: "$SALARY"}}}]) 7. Display gender wise maximum salary in your collection. db.Employee.aggregate([{$group:{_id:"$GENDER",Max_salary:{$max: "$SALARY"}}}]) 8. Display city wise maximum and minimum salary. db.Employee.aggregate([{$group:{_id:"$CITY",max:{$max:"$SALARY"},min: {$min:"$SALARY"}}}]) 9. Display count of persons lives in Sydney city in your collection. db.Employee.aggregate([{$match:{CITY:"Sydney"}},{$group:{_id:"$CITY",count: {$sum:1}}}]) 10. Display average salary of New York city. db.Employee.aggregate([{$match:{CITY:"New York"}},{$group:{_id:"$CITY",avg:{$avg:"$SALARY"}}}]) **Part – B: (Use STUDENT collection of Lab No 17)** 1. Display distinct department. db.STUDENT.distinct('DEPARTMENT') 2. Display city wise number of students. db.STUDENT.aggregate([{$group:{_id:"$CITY",numofstudent:{$sum : 1}}}]) 3. Display sum of fees in your collection. db.STUDENT.aggregate([{$group:{_id :null ,SUM_of_FEE:{$sum : "$FEES"}}}]) 4. Display average of fees in your document. db.STUDENT.aggregate([{$group:{_id :null ,Average_of_FEE:{$avg : "$FEES"}}}]) 5. Display maximum and minimum fees of your document. db.STUDENT.aggregate([{$group:{_id :null ,Maximum_FEE:{$max : "$FEES"}, Minimum_FEE:{$min:"$FEES"}}}]) **Part – C: (Use STUDENT collection of Lab No 17)** 1. Display department wise total fees in your collection. db.STUDENT.aggregate([{$group:{_id :"$DEPARTMENT" ,Total_FEE:{$sum:"$FEES"}}}]) 2. Display gender wise maximum fees in your collection. db.STUDENT.aggregate([{$group:{_id :"$GENDER" ,MAXIMUM_FEE:{$max:"$FEES"}}}]) 3. Display department maximum and minimum fees. db.STUDENT.aggregate([{$group:{_id :"$DEPARTMENT" ,Maximum_FEE:{$max :"$FEES"}, Minimum_FEE:{$min:"$FEES"}}}]) 4. Display count of persons lives in Rajkot city in your collection. db.STUDENT.aggregate([{$match:{CITY : 'Rajkot'}},{$group:{_id :null ,Total_Person:{$sum : 1}}}]) 5. Display department wise number of students.

```
db.STUDENT.aggregate([{$group:{_id:"$DEPARTMENT",numofstudent:{$sum : 1}}}])
```

22

Part A: Create collection as per following.

STUDENT

StuID	FirstName	LastName	Website	City	Division
1011	Keyur	Patel	techonthenet.com	Rajkot	II-BCX
1022	Hardik	Shah	digminecraft.com	Ahmedabad	I-BCY
1033	Kajal	Trivedi	bigactivities.com	Baroda	IV-DCX
1044	Bhoomi	Gajera	checkyourmath.com	Ahmedabad	III-DCW
1055	Harmit	Mitel	NULL	Rajkot	II-BCY
1066	Ashok	Jani	NULL	Baroda	II-BCZ

From the above given tables perform the following queries in MongoDB:

1. Display the name of students whose name starts with ‘k’.
   ```
   db.STUDENT.find({ "FirstName": /^k/i }, { "FirstName": 1, "_id": 0 })
   ```
2. Display the name of students whose name consists of five characters.
   ```
   db.STUDENT.find({ "FirstName": /^.{5}$/ }, { "FirstName": 1, "_id": 0 })
   ```
3. Retrieve the first name & last name of students whose city name ends with a & contains six characters.
   ```
   db.STUDENT.find({ "City": /^.{5}a$/ }, { "FirstName": 1,"LastName": 1, "_id": 0 });
   ```
4. Display all the students whose last name ends with ‘tel’.
   ```
   db.STUDENT.find({ "LastName": /tel$/i }, { "FirstName": 1, "LastName": 1, "_id": 0 })
   ```
5. Display all the students whose first name starts with ‘ha’ & ends with‘t’.
   ```
   db.STUDENT.find({ "FirstName": /^ha/,"FirstName": /t$/i}, { "FirstName": 1, "_id": 0 })
   ```
6. Display all the students whose first name starts with ‘k’ and third character is ‘y’.
   ```
   db.STUDENT.find({ "FirstName": /^K.y/ }, { "FirstName": 1, "_id": 0 })
   ```
7. Display the name of students having no website and name consists of five characters.
   ```
   db.STUDENT.find({ $and: [{ "FirstName":  /^.{5}$/ }, { "website": null }] },{ "FirstName": 1, "_id": 0 })
   ```
8. Display all the students whose last name consist of ‘jer’.
   ```
   db.STUDENT.find({ "LastName": /jer/ }, { "FirstName": 1,"LastName": 1, "_id": 0 })
   ```
9. Display all the students whose city name starts with either ‘r’ or ‘b’.
   ```
   db.STUDENT.find({ "City": /^[R,B]/ }, { "City": 1, "FirstName": 1, "_id": 0 })
   ```
10. Display all the name students having websites.
    ```
    db.STUDENT.find({ "Website": { $ne: null } }, { "FirstName": 1, "Website": 1, "_id": 0 });
    ```

Part B:

1. Display all the students whose name starts from alphabet A to H.
   ```
   db.STUDENT.find({ "FirstName":  /^[A-H]/ }, { "FirstName": 1, "_id": 0 })
   ```
2. Display all the students whose name’s second character is vowel.
   ```
   db.STUDENT.find({ "FirstName": /^.[a,e,i,o,u]/i }, { "FirstName": 1, "_id": 0 })
   ```
3. Display student’s name whose city name consist of ‘rod’.
   ```
   db.STUDENT.find({ "City": /rod/ }, { "FirstName": 1, "City": 1, "_id": 0 })
   ```
4. Retrieve the First & Last Name of students whose website name starts with ‘bi’.
   ```
   db.STUDENT.find({ "Website": /^bi/ }, { "FirstName": 1, "Website": 1, "_id": 0 })
   ```
5. Display student’s city whose last name consists of six characters.
   ```
   db.STUDENT.find({ "LastName": /^.{6}$/ },{ "City": 1, "_id": 0 })
   ```

Part C:

1. Display all the students whose city name consist of five characters & not starts with ‘ba’.
   db.STUDENT.find({ $and: [{ "City": /^.{5}$/ },{ "City": { $not: /^ba/i } }]},{ "FirstName": 1, "City": 1, "_id": 0 });
2. Show all the student’s whose division starts with ‘II’.
   db.STUDENT.find({ "Division": /^II/ }, { "FirstName": 1, "Division": 1, "_id": 0 })
3. Find out student’s first name whose division contains ‘bc’ anywhere in division name.
   db.STUDENT.find({ "Division": /BC/ }, { "FirstName": 1, "Division": 1, "_id": 0 })
4. Show student id and city name in which division consist of six characters and having website name.
   db.STUDENT.find({ $and: [{ "Division": /^.{6}$/ }, { "Website": {$ne:null}}] },{ "StuID": 1, "City": 1, "_id": 0 })
5. Display all the students whose name’s third character is consonant.
   db.STUDENT.find({ "FirstName": /^..[^a,e,i,o,u]/ }, { "FirstName": 1, "_id": 0 })

23

Part A: Create collection as per following.

CUSTOMER

CID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitucion 2222	Mexico D.F.	05021	Mexico
3	Antonio Moreno Taqueria	Antonio Moreno	Mataderos 2312	Mexico D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbkop	Christina Berglund	Berguvsvagen 8	Lulea	S-958 22	Sweden

From the above given tables perform the following queries in MongoDB:

1. Return all customers from a city that starts with 'L' followed by one wildcard character, then 'nd' and then two wildcard characters.
   db.CUSTOMER.find({City: /^L.nd../ });
2. Return all customers from a city that contains the letter 'L'.
   db.CUSTOMER.find({City: /L/i });
3. Return all customers from a city that do not contains the letter 'L'.
   db.CUSTOMER.find({City:{$not: /L/i }});
4. Return all customers that name starts with 'La'.
   db.CUSTOMER.find({CustomerName: /^La/i });
5. Return all customers that name does not starts with 'La'.
   db.CUSTOMER.find({CustomerName:{$not: /La/i }});
6. Return all customers that name starts with Vowels.
   db.CUSTOMER.find({CustomerName:/^[a,e,i,o,u]/i });
7. Return all customers that name does not starts with Vowels.
   db.CUSTOMER.find({CustomerName:{$not: /^[a,e,i,o,u]/i }});
8. Return all customers that name starts with 'a' to ’d’ or 'x' to ’z’.
   db.customers.find({CustomerName: /^[a-dx-z]/i });
9. Return all customers that name does not starts with 'a' to ’d’ or 'x' to ’z’.

```
db.CUSTOMER.find({CustomerName:{$not: /^[a-dx-z]/i }});
```

10. Return all customers that name starts with Vowels as well as ends with Vowels.
    ```
    db.CUSTOMER.find({CustomerName: /^[A,E,I,O,U].*[A,E,I,O,U]$/i});
    ```

Part B:

1. Return all customers that name starts with 'a' or starts with 'b'.
   ```
   db.CUSTOMER.find({ CustomerName: /^[a,b]/i });
   ```
2. Return all customers that name starts with 'a' or starts with 'c' or starts with ’t’.
   ```
   db.CUSTOMER.find({ CustomerName: /^[a,c,t]/i });
   ```
3. Return all customers that name starts with 'a' to ’d’.
   ```
   db.CUSTOMER.find({CustomerName: /^[a-d]/i });
   ```
4. Return all customers that name with 'a'.
   ```
   db.CUSTOMER.find({CustomerName: /a$/ });
   ```
5. Return all customers that name do not ends with 'a'.
   ```
   db.CUSTOMER.find({CustomerName:{$not: /a$/}});
   ```

Part C:

1. Return all customers that starts with 'b' and ends with ’s’.
   ```
   db.CUSTOMER.find({CustomerName: /^b.*s$/i});
   ```
2. Return all customers that contains the phrase 'or'.
   ```
   db.CUSTOMER.find({CustomerName: /or/i});
   ```
3. Return all customers that starts with "a" and are at least 3 characters in length.
   ```
   db.CUSTOMER.find({CustomerName: /^a.{2,}/i});
   ```
4. Return all customers that have "r" in the second position.
   ```
   db.CUSTOMER.find({CustomerName: /^.r/i});
   ```
5. Return all customers from Spain.
   ```
   db.CUSTOMER.find({Country: "Spain"});
   ```

24 **Part A: Create collection as per following.**

EMPLOYEE

EID	EName	Department	Salary	JoiningDate	City
101	Rahul	Admin	56000	1-Jan-90	Rajkot
102	Hardik	IT	18000	25-Sep-90	Ahmedabad
103	Bhavin	HR	25000	14-May-91	Baroda
104	Bhoomi	Admin	39000	8-Feb-91	Rajkot
105	Rohit	IT	17000	23-Jul-90	Jamnagar
106	Priya	IT	9000	18-Oct-90	Ahmedabad
107	Neha	HR	34000	25-Dec-91	Rajkot

From the above given tables perform the following queries in MongoDB:

1. Display the Highest, Lowest, Total, and Average salary of all employees. Label the columns Maximum, Minimum, Total_Sal and Average_Sal, respectively.
   ```
   db.EMPLOYEE.aggregate([{$group: { _id: null, Maximum: { $max: "$Salary" },Minimum: { $min: "$Salary" },Total_Sal: { $sum: "$Salary" },Average_Sal: { $avg: "$Salary" }}}]);
   ```
2. Find total number of employees of EMPLOYEE table.
   ```
   db.EMPLOYEE.countDocuments();
   ```
3. Give maximum salary from IT department.
   ```
   db.EMPLOYEE.aggregate([{ $match: { Department: "IT" } },{ $group: { _id: null, Maximum: { $max: "$Salary" } } }]);
   ```

4. Count total number of cities of employee without duplication.
   ```
   db.EMPLOYEE.distinct("City").length
   ```
5. Display city with the total number of employees belonging to each city.
   ```
   db.EMPLOYEE.aggregate([{ $group: {_id: "$City",total_employees: { $sum: 1 }} }]);
   ```
6. Display city having more than one employee.
   ```
   db.EMPLOYEE.aggregate([{ $group: {_id: "$City",total_employees: { $sum: 1 } }},{ $match: { total_employees: { $gt: 1 } } }]);
   ```
7. Give total salary of each department of EMPLOYEE table.
   ```
   db.EMPLOYEE.aggregate([{  $group: { _id: "$Department",Total_Sal: { $sum: "$Salary" }} }]);
   ```
8. Give average salary of each department of EMPLOYEE table without displaying the respective department name.
   ```
   db.EMPLOYEE.aggregate([{  $group: { _id: "$Department",avgSalary: { $avg: "$Salary" }} },{  $project: { _id: 0, avgSalary: 1}}]);
   ```
9. Give minimum salary of employee who belongs to Ahmedabad.
   ```
   db.EMPLOYEE.aggregate([{ $match: { City: "Ahmedabad" }}, {$group: {_id: null, Minimum: { $min: "$Salary" } }}]);
   ```
10. List the departments having total salaries more than 50000 and located in city Rajkot.
    ```
    db.EMPLOYEE.aggregate([{ $match: { City: "Rajkot" } },{$group: {_id: "$department",Total_Sal: { $sum: "$Salary" }}},{ $match: { Total_Sal: { $gt: 50000 } } }]);
    ```

Part B:

1. Count the number of employees living in Rajkot.
   ```
   db.EMPLOYEE.countDocuments({ city: "Rajkot" });
   ```
2. Display the difference between the highest and lowest salaries. Label the column DIFFERENCE.
   ```
   db.EMPLOYEE.aggregate([{ $group: { _id: null,  maxSalary: { $max: "$Salary" },minSalary: { $min: "$Salary" }}},
    {$addFields: {DIFFERENCE: { $subtract: ["$maxSalary", "$minSalary"] }}},
    {$project: {_id: 0, DIFFERENCE: 1 }}]);
   ```
3. Display the total number of employees hired before 1st January, 1991.
   ```
   db.EMPLOYEE.aggregate([ { $match: { JoiningDate: { $lt: new Date("1991-01-01") } } },{ $count: "employees_hired_before_1991" }]);
   ```
4. Display total salary of each department with total salary exceeding 35000 and sort the list by total salary.
   ```
   db.EMPLOYEE.aggregate([{$group: { _id: "$Department",Total_Sal: { $sum: "$Salary" }}},{ $match: { Total_Sal: { $gt: 35000 } } },{ $sort: { Total_Sal: -1 } }]);
   ```
5. List out department names in which more than two employees.
   ```
   db.EMPLOYEE.aggregate([{$group: {_id: "$Department",employee_count: { $sum: 1 }}},{ $match: { employee_count: { $gt: 2 } } }]);
   ```
6. Return all employee whose name consist of 5 character and starts with 'a' or starts with 'b'.
   ```
   db.EMPLOYEE.find({EName: /^[a,b].{4}$/i });
   ```
7. Return all employee whose name consist of minimum 3 character and starts with 'b' or 'r' or ’p’.
   ```
   db.EMPLOYEE.find({EName: /^[b,r,p].{2,}$/i });
   ```
8. Return all employee whose name ends with 'a' to ’d’.
   ```
   db.EMPLOYEE.find({EName: /[a-d]$/i });
   ```
9. Return all employee whose name ends with Vowels.

```
db.EMPLOYEE.find({ EName: /[a,e,i,o,u]$/i });
```

10. Return all employee whose name ends with Vowels or ‘f’ or ‘g’.
    ```
    db.EMPLOYEE.find({EName: /[a,e,i,o,u,f,g]$/i });
    ```

Part C: Create collection as per following.

COMPANY

Title	Company	Type	Production_year	System	Production_cost	Revenue	Rating
Blasting Boxes	Simone Games	action adventure	1998	PC	100000	200000	7
Run Run Run!	13 Mad Bits	shooter	2011	PS3	3500000	650000	3
Duck n’Go	13 Mad Bits	shooter	2012	Xbox	3000000	1500000	5
SQL Wars!	Vertabelo	wargames	2017	Xbox	5000000	25000000	10
Tap Tap Hex!	PixelGaming Inc.	rhythm	2006	PS2	2500000	3500000	7
NoRisk	Simone Games	action adventure	2004	PS2	1400000	3400000	8

From the above given tables perform the following queries in MongoDB:

1. Display the name and total revenue for each company.
   ```
   db.COMPANY.aggregate([{  $group: {_id: "$Company",total_revenue: { $sum: "$Revenue" }}}]);
   ```
2. Generate a report with the production year and the number of games released this year (named count), the average of production cost for all games produced in this year (named avg_cost) and the average revenue for that year (named avg_revenue).
   ```
   db.COMPANY.aggregate([{$group: {_id: "$Production_year",count: { $sum: 1 },avg_cost: { $avg: "$Production_cost" },avg_revenue: { $avg: "$Revenue" }}}]);
   ```
3. Count how many games of a given type are profitable (i.e. the revenue was greater than the production cost). Show the game type and the number of profitable games (named number_of_games) for each type.
   ```
   db.COMPANY.aggregate([{$match: { $expr: { $gt: ["$Revenue", "$Production_cost"] }}},{$group: {_id: "$Type", number_of_games: { $sum: 1 }}}]);
   ```
4. Obtain the type of games and the total revenue generated for games with a production_year after 2010 and with a PS2 or PS3 system. Order the result so the types with the highest revenue come first.
   ```
   db.COMPANY.aggregate([{ $match: {Production_year: { $gt: 2010 },System: { $in: ["PS2", "PS3"] }}},{ $group: {_id: "$Type",total_revenue: { $sum: "$Revenue" }}}, { $sort: { total_revenue: -1 }}]);
   ```
5. For all companies present in the table, obtain their names and the sum of gross profit over all years. (Assume that gross profit = revenue - cost of production). Name this column gross_profit_sum. Order the results by gross profit, in descending order.
   ```
   db.COMPANY.aggregate([{  $group:  {_id:  "$Company",gross_profit_sum:  {  $sum:  {  $subtract: ["$Revenue", "$Production_cost"] } }}},{ $sort: { gross_profit_sum: -1 }}]);
   ```
6. Obtain the yearly gross profit of each company. In other words, we want a report with the company name, the year, and the gross profit for that year. Order the report by company name and year.
   ```
   db.COMPANY.aggregate([  {$group:  {  _id:  {  company:  "$Company",  year:  "$Production_year" },gross_profit: { $sum: { $subtract: ["$Revenue", "$Production_cost"] }}}},
   {$sort: { "_id.company": 1, "_id.year": 1 }}]);
   ```

7. For each company, select its name, the number of games it's produced (as the number_of_games column), and the average cost of production (as the avg_cost column). Show only companies producing more than one game.

```
db.COMPANY.aggregate([{$group: { _id: "$Company",number_of_games: { $sum: 1 },avg_cost: { $avg: "$Production_cost" }}},
{$match: { number_of_games: { $gt: 1 } }}]);
```

25 **Create collection as per following.**

STUDENT

Rno	Name	Branch
101	Raju	CE
102	Amit	CE
103	Sanjay	ME
104	Neha	EC
105	Meera	EE
106	Mahesh	ME

RESULT

Rno	SPI
101	8.8
102	9.2
103	7.6
104	8.2
105	7.0
107	8.9

EMPLOYEE

EmployeeNo	Name	ManagerNo
E01	Tarun	*NULL*
E02	Rohan	E02
E03	Priya	E01
E04	Milan	E03
E05	Jay	E01
E06	Anjana	E04

From the above given tables perform the following queries in MongoDB:

Part A:

1. Display Rno, Name, Branch and SPI of all students.
```
db.Students.find({}, { "Rno": 1, "Name": 1, "Branch": 1, "SPI": 1 })
```
2. Display Rno, Name, Branch and SPI of CE branch's student only.
```
db.Students.find({ "Branch": "CE" }, { "Rno": 1, "Name": 1, "Branch": 1, "SPI": 1 })
```
3. Display Rno, Name, Branch and SPI of other than EC branch's student only.
```
db.Students.find({ "Branch": { $ne: "EC" } }, { "Rno": 1, "Name": 1, "Branch": 1, "SPI": 1 })
```
4. Display average result of each branch.
```
db.Students.aggregate([{ $group: { _id: "$Branch", avgSPI: { $avg: "$SPI" }}}])
```
5. Display average result of each branch and sort them in ascending order by SPI.
```
db.Students.aggregate([{ $group: { _id: "$Branch", avgSPI: { $avg: "$SPI" }}},{ $sort: { avgSPI: 1 } }])
```
6. Display average result of CE and ME branch.
```
db.Students.aggregate([{ $match: { "Branch": { $in: ["CE", "ME"]}}},{ $group: { _id: "$Branch", avgSPI: { $avg: "$SPI" }}}])
```
7. Retrieve the names of employee along with their manager name from the Employee table.
```
db.Employee.aggregate([{$lookup:{from:"Employee",localField:"ManagerNo",foreignField:"EmployeeNo",as: "manager"}},
{$project: {EmployeeName: "$Name",ManagerName: { $arrayElemAt: ["$manager.Name", 0] }} }])
```

Part-B:

Create collection as per following.

PERSON

PersonID	PersonName	DepartmentID	Salary	JoiningDate	City
101	Rahul Tripathi	2	56000	01-01-2000	Rajkot
102	Hardik Pandya	3	18000	25-09-2001	Ahmedabad
103	Bhavin Kanani	4	25000	14-05-2000	Baroda
104	Bhoomi Vaishnav	1	39000	08-02-2005	Rajkot
105	Rohit Topiya	2	17000	23-07-2001	Jamnagar
106	Priya Menpara	NULL	9000	18-10-2000	Ahmedabad

107	Neha Sharma	2	34000	25-12-2002	Rajkot
108	Nayan Goswami	3	25000	01-07-2001	Rajkot
109	Mehul Bhundiya	4	13500	09-01-2005	Baroda
110	Mohit Maru	5	14000	25-05-2000	Jamnagar

DEPARTMENT

DepartmentID	DepartmentName	DepartmentCode	Location
1	Admin	Adm	A-Block
2	Computer	CE	C-Block
3	Civil	CI	G-Block
4	Electrical	EE	E-Block
5	Mechanical	ME	B-Block

From the above given table perform the following queries in MongoDB:

1. Find all persons with their department name & code.
   ```
   db.PERSON.find({}, { PersonName: 1, DepartmentName: 1, DepartmentCode: 1 });
   ```
2. Give department wise maximum & minimum salary with department name.
   ```
   db.PERSON.aggregate([{$group: {_id: { DepartmentName: "$DepartmentName", DepartmentCode:
   "$DepartmentCode" },maxSalary: { $max: "$Salary" },minSalary: { $min: "$Salary" }}}]);
   ```
3. Find all departments whose total salary is exceeding 100000.
   ```
   db.PERSON.aggregate([{$group: {_id: { DepartmentName: "$DepartmentName", DepartmentCode:
   "$DepartmentCode" },totalSalary: { $sum: "$Salary" }}},
   { $match: { totalSalary: { $gt: 100000 } } }]);
   ```
4. Retrieve person name, salary & department name who belongs to Jamnagar city.
   ```
   db.PERSON.find({ City: "Jamnagar" }, { PersonName: 1, Salary: 1, DepartmentName: 1 });
   ```
5. Find all persons who does not belongs to any department.
   ```
   db.PERSON.find({ DepartmentName: null }, { PersonID: 1, PersonName: 1 });
   ```
6. Find department wise person counts.
   ```
   db.PERSON.aggregate([{$group: {_id: {DepartmentName: "$DepartmentName", DepartmentCode:
   "$DepartmentCode" },personCount: { $sum: 1 }}}]);
   ```
7. Find average salary of person who belongs to Ahmedabad city.
   ```
   db.PERSON.aggregate([{ $match: { City: "Ahmedabad" } },{ $group: { _id: null, averageSalary: { $avg:
   "$Salary" } } }]);
   ```
8. Produce Output Like: <PersonName> earns <Salary> from department <DepartmentName> monthly. (In Single Column)
   ```
   db.PERSON.find({}, {output: {$concat: [ "$PersonName"," earns ",{ $toString: "$Salary" }," from
   department ","$DepartmentName"," monthly." ]}});
   ```
9. List all departments who have no persons.
   ```
   db.PERSON.find({PersonName:null},{DepartmentName:1})
   ```
10. Find city & department wise total, average & maximum salaries.
    ```
    db.PERSON.aggregate([{ $group: {_id: { City: "$City", DepartmentName: "$DepartmentName"
    },totalSalary: { $sum: "$Salary" },averageSalary: { $avg: "$Salary" },maxSalary: { $max: "$Salary" }}}]);
    ```

Part - C

1. Display Unique city names.
   ```
   db.PERSON.distinct("City");
   ```

2. List out department names in which more than two persons.
   db.PERSON.aggregate([{$group: {_id: "$DepartmentName",count: { $sum: 1 } }},{ $match: {count: { $gt: 2 }}}]);
3. Combine person name's first three characters with city name's last three characters in single column.
   db.PERSON.aggregate([{$project: {combined: {$concat: [{ $substr: ["$PersonName", 0, 3] },{ $substr: ["$City", -3, 3] }]}}}]);
4. Give 10% increment in Computer department employee's salary.
   db.PERSON.updateMany({ DepartmentName: "Computer" },{ $mul: { Salary: 1.10 } });
5. Display all the person name's who's joining dates difference with current date is more than 365 days.
   db.PERSON.find({JoiningDate: { $lt: new Date(new Date().setFullYear(new Date().getFullYear() - 1)) }}, { PersonName: 1 });

26

Part A: Create database in MongoDB for the following RDBMS schema and enter data given in below tables.

STUDENT

Rno	Name	City	DID
101	Raju	Rajkot	10
102	Amit	Ahmedabad	20
103	Sanjay	Baroda	40
104	Neha	Rajkot	20
105	Meera	Ahmedabad	30
106	Mahesh	Baroda	10

ACADEMIC

Rno	SPI	Bklog
101	8.8	0
102	9.2	2
103	7.6	1
104	8.2	4
105	7.0	2
106	8.9	3

DEPARTMENT

DID	DName
10	Computer
20	Electrical
30	Mechanical
40	Civil

Part B: Update the below records in above MongoDB database.

Rno	Mobile
101	12345678
101	23456789
103	23456781
103	52345678
103	82345678
103	42345678

1. db.Students.updateOne({ "Rno": 101 }, { $set: { "Mobile": ["12345678", "23456789"] } })
2. db.Students.updateOne({ "Rno": 103 },{ $set: { "Mobile": ["23456781", "52345678", "82345678", "42345678"] } })

Part C:

1. Delate any one mobile no of student whose Rno is 103.
   db.Students.updateOne({ "Rno": 103 },{ $pull: { "Mobile": "52345678" } })
2. Update any one mobile no of student whose Rno is 103.
   db.Students.updateOne({ "Rno": 103, "Mobile.1": "52345678" },{ $set: { "Mobile.$": "98765432" }})
3. Delete mobile no field of student whose Rno is 101.
   db.Students.updateOne({ "Rno": 101 },{ $unset: { "Mobile": ""}})
4. Update address of student whose Rno is 105 as (Building Name: 'Darshan Building', Road Name: 'Raiya Road', Area: 'KKV area')
   db.Students.updateOne({ "Rno": 105 },{ $set: { "Address": { "BuildingName": "Darshan Building", "RoadName": "Raiya Road", "Area": "KKV area" } } })

	5. Delete all the documents of Computer Department. db.Students.deleteMany({ "DName": "Computer" })

3