||Jai Sri Gurudev ||
BGSKH Education Trust (R.) – A unit of Sri Adichunchanagiri Shikshana Trust(R
BGS College of Engineering and Technology (BGSCE
Mahalakshmipuram, West of Chord Road, Bengaluru-560086
(Approved by AICTE, New Delhi and Affiliated to VTU, Belagavi)

(2024-25 EVEN SEMESTER)

DEPARTMENT OF

ARTIFICIAL INTELLIGENCE & DATA SCIENCE

DATABASE MANAGEMENT SYSTEM

LABORATORY BCS403

FOR

IV Semester B.E

PREFACE

Structured Query Language is a programming language used for storing and managing data in RDBMS. SQL was the first commercial language introduced for E F Codd's Relational model. Today almost all RDBMS (MYSql, Oracle, infomix)uses SQL as the standard database language. SQL is used to perform all types of data operations in RDBMS.

This manual for practical "DBMS laboratory" is an introduction to all popular concepts of DBMS and their implementation. It is planned for beginners who would like to learn the subject through programs.

This edition emphasizes on abstract concepts of Database Management System and describes how these are useful in problem solving using available packages and utility from SQL package. Students will gain a good appreciation of the subject as this manual has a clear display of syntax and elegant programming examples. To simplify concepts of DBMS programs given are implemented in simple SQL language understandable even by beginners.

MongoDB is an open source NoSQL database management program. NoSQL (Not only SQL) is used as an alternative to traditional relational databases. NoSQL databases are quite useful for working with large sets of distributed data. MongoDB is a tool that can manage document-oriented information, store or retrieve information.

MongoDB is used for high-volume data storage, helping organizations store large amounts of data while still performing rapidly. Organizations also use MongoDB for its ad-hoc queries, indexing, load balancing, aggregation, server-side JavaScript execution and other features. MongoDB is used to perform the CRUD operations.

- Dr. Parvathi C

Department of Artificial Intelligence & Data Science

ABOUT

The department of Artificial Intelligence and Data Science (AI&DS) is established during the academic year 2022-23 with an intake of 60.

- Artificial Intelligence is a human-like intelligence provided to machines where machines act and think as humanly & solve problems faster than humans. Speech recognition, translation tools, etc., are the building areas of AI.
- Artificial Intelligence is the implementation of a predictive model to forecast future events and trends, Automation of the process & uses machine learning techniques
- Data Science is a subset of Artificial Intelligence. Data science is a collection of data to analyze and make a decision. It uses scientific methods, processes, algorithms, and insights from many structural and unstructured data.
- Data Science is a detailed process that mainly involves pre-processing analysis, visualization, and prediction with a high degree of scientific processing & extensive tools will be used to process the data uses the technique of data analysis and data analytics.

  AI and Data science is the current trend, ruling the business world and it is highly paid career now.

BGS COLLEGE OF ENGINEERING AND TECHNOLOGY

Department of Artificial Intelligence & Data Science

INSTITUTE VISION

Creating Competent IT Professionals With Core Values For The Real World

INSTITUTE MISSION

- Providing Students with a Sound Knowledge in IT Fundamentals
- Exposing Students to Emerging Frontiers in various domains of IT enabling Continuous Learning.
- Promoting Excellence in Teaching, Training, Research and Consultancy.
- Developing Entrepreneurial acumen to venture into Innovative areas of IT
- Imparting value based Professional Education with a sense of Social Responsibility.

DEPARTMENT VISION

To nurture innovative minds for academic excellence with emphasis on in-depth technical knowledge for creating a value-based sustainable society.

DEPARTMENT MISSION

1. Developing practically trained skilled professionals to meet the demands of the corporate world
2. Creating an ecosystem of academic excellence through the best teaching-learning methods.
3. Grooming professionals with high ethical values and ability to solve real-life problems

PROGRAM EDUCATIONAL OBJECTIVES:

PEO1: To produce graduates who have strong foundation of knowledge and skills in the field of computer science and engineering.

PEO2: To produce graduates who are employable in industries/public sector/research organizations or work as an entrepreneur.

PEO3: To produce graduates who can provide leadership and are effective in multidisciplinary environment**.**

PROGRAM SPECIFIC OUTCOMES:

PSO1: Students will be able to apply fundamental knowledge of theoretical computer science and critically analyse problems to develop computational systems for engineering and scientific applications

PSO2: Students will be able to design and develop computing systems by appropriate technology using the concepts of basic science, computer engineering and other related disciplines for engineering / social applications

PROGRAM OUTCOMES

PO1	**Engineering Knowledge**	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem Analysis**	Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/Development of Solutions**	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct Investigations of Complex Problems**	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions
PO5	**Modern Tool Usage**	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	**The Engineer And Society**	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and Sustainability**	Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics**	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and Team Work**	Function effectively as an individual, and as a member or leader in diverse teams, and in multi-disciplinary settings.
PO10	**Communication**	Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions
PO11	**Project Management and Finance**	Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments
PO12	**Life-Long Learning**	Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

BGS COLLEGE OF ENGINEERING AND TECHNOLOGY

Department of Artificial Intelligence & Data Science

LABORATORY MANUAL (EVEN 2024)

Database Management System Laboratory

(BCS403)

FOR

Bachelor of Engineering

IV SEMESTER

Prepared by

Dr. Parvathi C

Dept. of AI & DS, BGSCET

2024-25

TABLE OF CONTENTS

Sl. No.	PARTICULARS	Page No.
1	**PEO, PO, PSO**	1-3
2	**COURSE DETAILS** • **Course Objectives** • **Syllabus** • **Course Outcomes**	4-8
3	**Evaluation Process**	9
4	**Program 1:** Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)	16
5	**Program 2:** Create a table called Employee that contain attributes EMPNO, ENAME, JOB, MGR, SAL & execute the following.	19
6	**Program 3:** Queries using aggregate functions (COUNT, AVG, MIN, MAX, SUM), Groupby, Orderby. Employee (E_id, E_name, Age, Salary)	23
7	**Program 4:** Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old & new Salary.	29
8	Create cursor for Employee table & extract the values from the table. Declare the variables, Open the cursor & extract the values from the cursor. Close the cursor. Employee (E_id, E_name, Age, Salary	34
9	**Program 6:** Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table, then that data should be skipped.	37
10	**Program 7:** Install an Open Source NoSQL Data base MangoDB & perform basic CRUD(Create, Read, Update & Delete) operations. Execute MangoDB basic Queries using CRUD operations.	41
11	**Additional Programs**	49
12	**Viva**	61

COURSE DETAILS

Course Name: DATABASE MANAGEMENT SYSTEM LABORATORY

Subject Code: BCS403

COURSE OBJECTIVES

- To Provide a strong foundation in database concepts, technology, and practice.
- To Practice SQL programming through a variety of database problems.
- To Understand the relational database design principles.
- To Demonstrate the use of concurrency and transactions in database.
- To Design and build database applications for real world problems.
- To become familiar with database storage structures and access techniques.

COURSE OUTCOMES

At the end of the course, the student will be able to:

- Describe the basic elements of a relational database management system
- Design entity relationship for the given scenario.
- Apply various Structured Query Language (SQL) statements for database manipulation.
- Analyses various normalization forms for the given application.
- Develop database applications for the given real-world problem.
- Understand the concepts related to NoSQL databases.

PROGRAM OUTCOMES(POs)

COURSE OUTCOME	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
BCS403.1	3	2			3							
BCS403.2	3	3			3							
BCS403.3	3	3	2		3	1			2		3	3
BCS403.4	3	3	3	2	3				3		3	3
BCS403.5	3	3	3	2	3				3		3	3
BCS403.6	2	2	2	2	2				2		2	2

DBMS LAB BCS403	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
	3	3	3	2	3	1			3		3	3

COURSE OUTCOME	PSO1	PSO2
BCS403.1	3	3
BCS403.2	3	2
BCS403.3	2	3
BCS403.4	3	3
BCS403.5	3	3
BCS403.6	1	2

DBMS LAB	PSO1	PSO2
BCS403	3	3

Sl.NO	PROGRAMS
1	Create a table called Employee & execute the following. **Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)** 1. Create a user and grant all permissions to theuser. 2. Insert the any three records in the employee table contains attributes EMPNO,ENAME JOB, MANAGER_NO, SAL, COMMISSION and use rollback. Check the result. 3. Add primary key constraint and not null constraint to the employee table. 4. Insert null values to the employee table and verify the result.
2	Create a table called Employee that contain attributes EMPNO,ENAME,JOB, MGR,SAL & execute the following. 1. Add a column commission with domain to the Employee table. 2. Insert any five records into the table. 3. Update the column details of job 4. Rename the column of Employ table using alter command. 5. Delete the employee whose Empno is 105.
3	Queries using aggregate functions(COUNT,AVG,MIN,MAX,SUM),Group by,Orderby. **Employee(E_id, E_name, Age, Salary)** 1. Create Employee table containing all Records E_id, E_name, Age, Salary. 2. Count number of employee names from employee table 3. Find the Maximum age from employee table. 4. Find the Minimum age from employee table. 5. Find salaries of employee in Ascending Order. 6. Find grouped salaries of employees.
4	Create a row level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old & new Salary. **CUSTOMERS(ID,NAME,AGE,ADDRESS,SALARY)**
5	Create cursor for Employee table & extract the values from the table. Declare the variables ,Open the cursor & extract the values from the cursor. Close the cursor. **Employee(E_id, E_name, Age, Salary)**

6	Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped.
7	Install an Open Source NoSQL Data base MangoDB & perform basic CRUD(Create, Read, Update & Delete) operations. Execute MangoDB basic Queries using CRUD operations.

Course outcomes (Course Skill Set):

At the end of the course, the student will be able to:

- Describe the basic elements of a relational database management system
- Design entity relationship for the given scenario.
- Apply various Structured Query Language (SQL) statements for database manipulation.
- Analyze various normalization forms for the given application.
- Develop database applications for the given real world problem.
- Understand the concepts related to NoSQL databases.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum

passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

CIE for the theory component of the IPCC (maximum marks 50)

- IPCC means practical portion integrated with the theory of the course.
- CIE marks for the theory component are **25 marks** and that for the practical component is **25 marks**.
- 25 marks for the theory component are split into **15 marks** for two Internal Assessment Tests (Two Tests, each of 15 Marks with 01-hour duration, are to be conducted) and **10 marks** for other assessment methods mentioned in 22OB4.2. The first test at the end of 40-50% coverage of the syllabus and the second test after covering 85-90% of the syllabus.
- Scaled-down marks of the sum of two tests and other assessment methods will be CIE marks for the theory component of IPCC (that is for **25 marks)**.
- The student has to secure 40% of 25 marks to qualify in the CIE of the theory component of IPCC.

CIE for the practical component of the IPCC

- **15 marks** for the conduction of the experiment and preparation of laboratory record, and **10 marks** for the test to be conducted after the completion of all the laboratory sessions.
- On completion of every experiment/program in the laboratory, the students shall be evaluated including viva-voceand marks shall be awarded on the same day.
- The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluation of the laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all experiments' write-ups are added and scaled down to **15 marks**.
- The laboratory test **(duration 02/03 hours)** after completion of all the experiments shall be conducted for 50 marks and scaled down to **10 marks.**
- Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory
- component of IPCC for **25 marks**.
- The student has to secure 40% of 25 marks to qualify in the CIE of the practical component of the IPCC.

SEE for IPCC

Theory SEE will be conducted by University as per the scheduled timetable, with common question papers for the course(**duration 03 hours**)

1. The question paper will have ten questions. Each question is set for 20 marks.
2. There will be 2 questions from each module. Each of the two questions under a module (with a maximum of 3 sub-questions), **should have a mix of topics** under that module.
3. The students have to answer 5 full questions, selecting one full question from each module.
4. Marks scoredby the student shall be proportionally scaled down to 50 Marks

The theory portion of the IPCC shall be for both CIE and SEE, whereas the practical portion will have a CIEcomponent only. Questions mentioned in the SEE paper may include questions from the practical component.

Suggested Learning Resources:

Text Books:

1. Fundamentals of Database Systems, Ramez Elmasri and Shamkant B. Navathe, 7th Edition, 2017, Pearson.
2. Database management systems, Ramakrishnan, and Gehrke, 3rd Edition, 2014, McGraw Hill

Activity Based Learning (Suggested Activities in Class)/ Practical Based learningMini Project:

- Project Based Learning

LAB EVALUATION PROCESS

INTRERNAL ASSESMENT EVALUATION (TEST)		
1	Procedure	10
2	Execution	25
3	Viva	15
	TOTAL	**50**
4	Record	10 for each lab

INTRERNAL ASSESSMENT EVALUATION (End of Semester)		
SL.NO	**ACTIVITY**	**MARKS**
1	Record	15
2	Test	10
	TOTAL	**25**

Steps to log in to the SQL PLUS:

- Switch on the system.
- Enter user id: System, password: Manager
- Create the user.
- Connect to the database: connect username / password.
- Then start with the execution of program .

PROGRAM 1

Create a table called Employee & execute the following.

Employee (EMPNO, ENAME, JOB, MANAGER_NO, SAL, COMMISSION)

1. Create a user and grant all permissions to the user.
2. Insert the any three records in the employee table contains attributes EMPNO, ENAME JOB, MANAGER_NO, SAL, COMMISSION and use rollback. Check the result.
3. Add primary key constraint and not null constraint to the employee table.
4. Insert null values to the employee table and verify the result.

1. Create a user and grant all permissions to the user.

```
Sql> create user <username> identified by <password>;
Sql> grant resource, connect to <username>;
Sql> grant all privileges to username identified by password;
Sql> select privilege from dba_sys_privs
     where grantee='username'
     order by 1;

Sql> connect <username> / <password>;

Sql>create table employee1
    (
     empno number(5),
     ename varchar2(10),
     job varchar2(10),
      manager_no number(5),
      sal number(10, 2),
      commission number(10, 2));

Sql> desc employee1;
```

```
SQL> create table employee1
  2  (
  3       empno number(5),
  4       ename varchar2(10),
  5       job varchar2(10),
  6       manager_no number(5),
  7       sal number(10, 2),
  8       commission number(10, 2)
  9  );

Table created.

SQL> desc employee1;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                              NUMBER(5)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(10)
 MANAGER_NO                                         NUMBER(5)
 SAL                                                NUMBER(10,2)
 COMMISSION                                         NUMBER(10,2)
```

2. Insertion of Values to Tables

sql> insert into employee1 values(1, 'john', 'manager', null, 5000, 1000);

sql> insert into employee1 values(2, 'smith', 'developer', 1, 4000, null);

sql> insert into employee1 values(3, 'decock', 'ceo', null, 5000, 1000);

Sql> select * from employee1;

```
SQL> select * from employee1;

     EMPNO ENAME      JOB        MANAGER_NO        SAL COMMISSION
---------- ---------- ---------- ---------- ---------- ----------
         1 john       manager                    5000       1000
         2 smith      developer           1       4000
         3 decock     ceo                        5000       1000
```

Sql> delete from employee1;

Sql> select * from employee1;

Sql> rollback;

Sql> select * from employee1;

```
SQL> delete from employee1;

3 rows deleted.

SQL> select * from employee1;

no rows selected

SQL> rollback;

Rollback complete.

SQL> select * from employee1;

no rows selected
```

Again u have to insert records to employee1

sql> insert into employee1 values(1, 'john', 'manager', null, 5000, 1000);
sql> insert into employee1 values(2, 'smith', 'developer', 1, 4000, null);
sql> insert into employee1 values(1, 'decock', 'ceo', null, 5000, 1000);

sql> select * from employee1;

sql> commit;

Sql> delete from employee1;

Sql> select * from employee1;

Sql> rollback;

Sql> select * from employee1;

```
SQL> select * from employee1;

     EMPNO ENAME      JOB        MANAGER_NO        SAL COMMISSION
---------- ---------- ---------- ---------- ---------- ----------
         1 john       manager                    5000       1000
         2 smith      developer           1       4000
         3 decock     ceo                        5000       1000

SQL> commit;

Commit complete.

SQL> delete from employee1;

3 rows deleted.

SQL> select * from employee1;

no rows selected

SQL> rollback;

Rollback complete.

SQL> select * from employee1;

     EMPNO ENAME      JOB        MANAGER_NO        SAL COMMISSION
---------- ---------- ---------- ---------- ---------- ----------
         1 john       manager                    5000       1000
         2 smith      developer           1       4000
         3 decock     ceo                        5000       1000
```

3)Add primary key constraint and not null constraint to the employee table.

Sql>ALTER TABLE Employee1 ADD CONSTRAINT pk_employee1 PRIMARY KEY (empno);

```
SQL> ALTER TABLE Employee1
  2  ADD CONSTRAINT pk_employee1 PRIMARY KEY (empno);

Table altered.

SQL> desc employee1;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                     NOT NULL NUMBER(5)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(10)
 MANAGER_NO                                         NUMBER(5)
 SAL                                                NUMBER(10,2)
 COMMISSION                                         NUMBER(10,2)
```

4) Add NOT NULL constraint to ENAME, JOB, and SAL columns

Sql>alter table employee1

modify (ename varchar2(10) not null,

job varchar2(10) not null,

sal number(10, 2) not null);

Sql>Desc employee1;

```
SQL> ALTER TABLE Employee1
  2  MODIFY (ENAME VARCHAR2(10) NOT NULL,
  3          JOB VARCHAR2(10) NOT NULL,
  4          SAL NUMBER(10, 2) NOT NULL);

Table altered.

SQL> Desc employee1;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                     NOT NULL NUMBER(5)
 ENAME                                     NOT NULL VARCHAR2(10)
 JOB                                       NOT NULL VARCHAR2(10)
 MANAGER_NO                                         NUMBER(5)
 SAL                                       NOT NULL NUMBER(10,2)
 COMMISSION                                         NUMBER(10,2)
```

5) Insert null values to the employee table and verify the result.

INSERT INTO Employee1 VALUES (4, NULL, 'Tester', 1, NULL, NULL);

```
SQL> INSERT INTO Employee1 VALUES (4, NULL, 'Tester', 1, NULL, NULL);
INSERT INTO Employee1 VALUES (4, NULL, 'Tester', 1, NULL, NULL)
                             *
ERROR at line 1:
ORA-01400: cannot insert NULL into ("SYSTEM"."EMPLOYEE1"."ENAME")
```

PROGRAM 2

Create a table called Employee that contain attributes EMPNO, ENAME, JOB, MGR, SAL & execute the following.

1. **Add a column commission with domain to the Employee table.**
2. **Insert any five records into the table.**
3. **Update the column details of job**
4. **Rename the column of Employ table using alter command.**
5. **Delete the employee whose Empno is 105.**

Create a table called Employee that contain attributes EMPNO, ENAME, JOB, MGR, SAL

```
Sql> create table employee2 (
      empno int primary key,
      ename varchar(10),
      job varchar(10),
     mgr int,
     sal decimal(10, 2)
     );
```

```
Sql> desc employee2;
```

```
SQL> desc employee2;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                     NOT NULL NUMBER(38)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(10)
 MGR                                                NUMBER(38)
 SAL                                                NUMBER(10,2)
```

1. Adding a column commission to the Employee table

sql>alter table employee2 add commission decimal(10, 2);

```
SQL> alter table employee2 add commission decimal(10, 2);

Table altered.
```

Sql> desc employee2;

```
SQL> desc employee2;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                     NOT NULL NUMBER(38)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(10)
 MGR                                                NUMBER(38)
 SAL                                                NUMBER(10,2)
 COMMISSION                                         NUMBER(10,2)
```

2. Insert any five records into the table.

Sql> insert into employee2 values (101, 'John Doe', 'Manager', 111, 50000.00, 1000.00);

Sql> insert into employee2 values (102, 'Jane Smith', 'Developer', 222, 40000.00, 1800.50);

Sql> insert into employee2 values (103, 'Mike John', 'Analyst', 333, 35000.50, 1700.00);

Sql> insert into employee2 values (104, 'Emily Bown', 'Designer',444, 38000.00, 1750.80);

Sql> insert into employee2 values (105, 'David Lee', 'Tester', 555, 32000.00, 1600.60);

Sql> select * from employee2;

```
SQL> select * from employee2;

     EMPNO ENAME      JOB              MGR        SAL COMMISSION
---------- ---------- ---------- ---------- ---------- ----------
       101 John Doe   Manager           111      50000       1000
       102 Jane Smith Developer         222      40000     1800.5
       103 Mike John  Analyst           333    35000.5       1700
       104 Emily Bown Designer          444      38000     1750.8
       105 David Lee  Tester            555      32000     1600.6
```

3. Updating the column details of job

-- For example, changing 'Manager' to 'Project Manager'

Sql> alter table employee2 modify job varchar(15);

```
SQL> alter table employee2 modify job varchar(10);

Table altered.

SQL> alter table employee2 modify job varchar(15);

Table altered.

SQL> desc employee2;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                     NOT NULL NUMBER(38)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(15)
 MGR                                                NUMBER(38)
 SAL                                                NUMBER(10,2)
 COMMISSION                                         NUMBER(10,2)
```

Sql> update employee2

set job = 'project manager'

where mgr = 111;

Sql> select * from employee2;

```
SQL> select * from employee2;

     EMPNO ENAME      JOB                 MGR        SAL COMMISSION
---------- ---------- ------------------- ---------- ---------- ----------
       101 John Doe   project manager     111      50000       1000
       102 Jane Smith Developer           222      40000     1800.5
       103 Mike John  Analyst             333    35000.5       1700
       104 Emily Bown Designer            444      38000     1750.8
       105 David Lee  Tester              555      32000     1600.6
```

3. Renaming the column of Employee table using ALTER command

Sql> alter table employee2

rename column mgr to manager_id;

sql> desc employee2;

```
SQL> desc employee2;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPNO                                     NOT NULL NUMBER(38)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(15)
 MANAGER_ID                                         NUMBER(38)
 SAL                                                NUMBER(10,2)
 COMMISSION                                         NUMBER(10,2)
```

Sql> select * from employee2;

```
SQL> alter table employee2
  2  rename column mgr to manager_id;

Table altered.

SQL> select * from employee2;

     EMPNO ENAME      JOB              MANAGER_ID        SAL COMMISSION
---------- ---------- --------------- ---------- ---------- ----------
       101 John Doe   project manager        111      50000       1000
       102 Jane Smith Developer              222      40000     1800.5
       103 Mike John  Analyst                333    35000.5       1700
       104 Emily Bown Designer               444      38000     1750.8
       105 David Lee  Tester                 555      32000     1600.6
```

4. Deleting the employee whose Empno is 105

Sql> delete from employee2

where empno = 105;

Sql> select * from employee2;

PROGRAM 3

Queries using aggregate functions (COUNT,AVG,MIN,MAX,SUM),Group by,Orderby.

Employee(E_id, E_name, Age, Salary)

1. Create Employee table containing all Records E_id, E_name, Age, Salary.
2. Count number of employee names from employeetable
3. Find the Maximum age from employee table.
4. Find the Minimum age from employeetable.
5. Find salaries of employee in Ascending Order.
6. Find grouped salaries of employees.

Q1. Create an Employee3 table containing all Records E_id, E_name, Age, Salary.

```
Sql> create table employee3 (
    e_id int primary key,
    e_name varchar(10),
    age int,
    salary decimal(10, 2)
    );
```

Sql> desc employee3;

```
SQL> desc employee3;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 E_ID                                      NOT NULL NUMBER(38)
 E_NAME                                             VARCHAR2(10)
 AGE                                                NUMBER(38)
 SALARY                                             NUMBER(10,2)
```

```
sql> insert into employee3 values(101,'ramkumar',32,45000);
sql> insert into employee3 values(102,'ajay',30,30000);
sql> insert into employee3 values(103,'srajan',35,70000);
sql> insert into employee3 values(104,'stany',340,60000);
```

sql> insert into employee3 values(105,’aramn’,32,40000);

sql> select * from employee3;

```
SQL> select * from employee3;

      E_ID E_NAME               AGE      SALARY
---------- ---------- ---------- ----------
       101 ramkumar               32       45000
       102 ajay                   30       30000
       103 srajan                 35       70000
       104 stany                  34       60000
       105 aramn                  32       40000
```

Q2. Count number of employee names from employee table

sql>select count(e_name) as totalemployees from employee3;

```
SQL> select count(e_name) as totalemployees from employee3;

TOTALEMPLOYEES
--------------
             5
```

Q3. Find the maximum age from employee table.

sql> select max(age) as maxage from employee3;

```
SQL> select max(age) as maxage from employee3;

    MAXAGE
----------
        35
```

Q4. Find the minimum age from employeetable.

sql>select min(age) as minage from employee3;

```
SQL> select min(age) as minage from employee3;

    MINAGE
----------
        30
```

Q5. Find salaries of employee in ascending order.

sql> select salary from employee3 order by salary asc;

```
SQL> select salary from employee3 order by salary asc;

    SALARY
----------
     30000
     40000
     45000
     60000
     70000
```

Q6. Find grouped salaries of employees

sql>select salary, count(*) as numemployees from employee3 group by salary;

```
SQL> select salary, count(*) as numemployees from employee3 group by salary;

    SALARY NUMEMPLOYEES
---------- ------------
     45000            1
     30000            1
     70000            1
     60000            1
     40000            1
```

```
SQL> update employee3 set salary=45000 where e_id=105;

1 row updated.

SQL> select salary, count(*) as numemployees from employee3 group by salary;

    SALARY NUMEMPLOYEES
---------- ------------
     45000            2
     30000            1
     70000            1
     60000            1
```

sql>select e_name, age, age + 5 as ageafter5years from employee3;

```
SQL> select e_name, age, age + 5 as ageafter5years from employee3;

E_NAME            AGE AGEAFTER5YEARS
---------- ---------- --------------
ramkumar           32             37
ajay               30             35
srajan             35             40
stany              34             39
aramn              32             37
```

sql> select e_name, salary, salary * 0.1 as salaryincrease from employee3;

```
SQL> select e_name, salary, salary * 0.1 as salaryincrease from employee3;

E_NAME         SALARY SALARYINCREASE
---------- ---------- --------------
ramkumar        45000           4500
ajay            30000           3000
srajan          70000           7000
stany           60000           6000
aramn           45000           4500
```

sql> select avg(salary) as averagesalary from employee3;

```
SQL> select avg(salary) as averagesalary from employee3;

AVERAGESALARY
-------------
        50000
```

PROGRAM 4

Create a row level trigger for the customers table that would fire for INSERT or UPDATE orDELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old & new Salary.

CUSTOMERS(ID,NAME,AGE,ADDRESS,SALARY)

Q1.Create the CUSTOMERS table

```
Sql> Create table customers (
        id int primary key,
        name varchar(10),
        Age int,
        Address varchar(15),
        Salary decimal(10,2)
        );
```

Sql> desc customers;

```
SQL> DESC CUSTOMERS;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                        NOT NULL NUMBER(38)
 NAME                                               VARCHAR2(10)
 AGE                                                NUMBER(38)
 ADDRESS                                            VARCHAR2(10)
 SALARY                                             NUMBER(10,2)

SQL>
```

```
Sql> insert into customers values(101,'smith',32,'capgemini',55000);
Sql> insert into customers values(102,'smitharani',30,'capgemini',45000);
Sql> insert into customers values(103,'smitharani',30,'capgemini',45000);
Sql> insert into customers values(104,'kamelesh',32,'capgemini',55000);
Sql> insert into customers values(105,'amith',40,'capgemini',65000);
```

Sql> select * from customers;

```
SQL> SELECT * FROM CUSTOMERS;

        ID NAME                AGE ADDRESS         SALARY
---------- ---------- ---------- ---------- ----------
       101 smith                32 capgemini        55000
       102 smitharani           30 capgemini        45000
       103 jeeven               42 capgemini        85000
       104 kamelesh             32 capgemini        55000
       105 amith                40 capgemini        65000
```

Create the trigger

```
sql >create or replace trigger salary_difference_trigger
before insert or update or delete on customers
for each row
declare
   old_salary number;
   new_salary number;
begin
   if inserting or updating then
      old_salary := nvl(:old.salary, 0);
      new_salary := nvl(:new.salary, 0);
      dbms_output.put_line('salary difference: ' || (new_salary - old_salary));
   elsif deleting then
      old_salary := nvl(:old.salary, 0);
      dbms_output.put_line('salary before deletion: ' || old_salary);
   end if;
end;
/
```

```
SQL> CREATE OR REPLACE TRIGGER salary_difference_trigger
  2  BEFORE INSERT OR UPDATE OR DELETE ON CUSTOMERS
  3  FOR EACH ROW
  4  DECLARE
  5      old_salary NUMBER;
  6      new_salary NUMBER;
  7  BEGIN
  8      IF INSERTING OR UPDATING THEN
  9          old_salary := NVL(:OLD.SALARY, 0);
 10          new_salary := NVL(:NEW.SALARY, 0);
 11          DBMS_OUTPUT.PUT_LINE('Salary difference: ' || (new_salary - old_salary));
 12      ELSIF DELETING THEN
 13          old_salary := NVL(:OLD.SALARY, 0);
 14          DBMS_OUTPUT.PUT_LINE('Salary before deletion: ' || old_salary);
 15      END IF;
 16  END;
 17  /

Trigger created.
```

b) create or replace trigger salary_difference_trigger
after insert or update or delete on customers
for each row
declare
old_salary customers.salary%type;
new_salary customers.salary%type;
  difference number;
begin
  if inserting then
    dbms_output.put_line('new record inserted.');
    dbms_output.put_line('id: ' || :new.id || ', name: ' || :new.name || ', age: ' || :new.age || ', address: ' ||
:new.address || ', salary: ' || :new.salary);
  elsif updating then
old_salary := :old.salary;
new_salary := :new.salary;
difference := new_salary - old_salary;
    dbms_output.put_line('salary updated for id: ' || :new.id || '. old salary: ' || old_salary || ', new salary: ' ||
new_salary || ', salary difference: ' || difference);
  elsif deleting then
    dbms_output.put_line('record deleted for id: ' || :old.id || ', name: ' || :old.name || ', age: ' || :old.age || ',
address: ' || :old.address || ', salary: ' || :old.salary);
  end if;
end;
/

```
SQL> CREATE OR REPLACE TRIGGER salary_difference_trigger
  2  AFTER INSERT OR UPDATE OR DELETE ON customers
  3  FOR EACH ROW
  4  DECLARE
  5  old_salary customers.salary%TYPE;
  6  new_salary customers.salary%TYPE;
  7      difference NUMBER;
  8  BEGIN
  9      IF INSERTING THEN
 10          DBMS_OUTPUT.PUT_LINE('New record inserted.');
 11          DBMS_OUTPUT.PUT_LINE('ID: ' || :NEW.ID || ', Name: ' || :NEW.NAME || ', Age: ' || :NEW.AGE || ', Address: '
|| :NEW.ADDRESS || ', Salary: ' || :NEW.SALARY);
 12      ELSIF UPDATING THEN
 13  old_salary := :OLD.SALARY;
 14  new_salary := :NEW.SALARY;
 15  difference := new_salary - old_salary;
 16          DBMS_OUTPUT.PUT_LINE('Salary updated for ID: ' || :NEW.ID || '. Old Salary: ' || old_salary || ', New Salar
y: ' || new_salary || ', Salary Difference: ' || difference);
 17      ELSIF DELETING THEN
 18          DBMS_OUTPUT.PUT_LINE('Record deleted for ID: ' || :OLD.ID || ', Name: ' || :OLD.NAME || ', Age: ' || :OLD.A
GE || ', Address: ' || :OLD.ADDRESS || ', Salary: ' || :OLD.SALARY);
 19      END IF;
 20  END;
 21  /

Trigger created.
```

Sql>set serveroutput on;

//Enable serveroutput for executing at dbms_output.put_line//

Sql> insert into customers values(106,’arun’,40,’capgemini’,85000);

```
SQL> insert into customers values(106,'arun',40,'capgemini',85000);
Salary difference: 85000
New record inserted.
ID: 106, Name: arun, Age: 40, Address: capgemini, Salary: 85000

1 row created.
```

sql>update customers set salary=85000 where id=101;

```
SQL> set serveroutput on;
SQL> UPDATE CUSTOMERS SET SALARY=85000 WHERE ID=101;
Salary difference: 10000
Salary updated for ID: 101. Old Salary: 75000, New Salary: 85000, Salary
Difference: 10000

1 row updated.
```

sql> delete from customers where id=104;

```
SQL> DELETE FROM CUSTOMERS WHERE ID=104;
Salary before deletion: 55000
Record deleted for ID: 104, Name: kamelesh, Age: 32, Address: capgemini, Salary:
55000

1 row deleted.
```

PROGRAM 5

Create cursor for Employee table & extract the values from the table. Declare the variables

Open the cursor & extract the values from the cursor. Close the cursor.

Employee (E_id, E_name, Age, Salary)

Q1. Create an Employee3 table containing all Records E_id, E_name, Age, Salary.

```
Sql> create table employee5 (
        e_id int primary key,
        e_name varchar(10),
        age int,
        salary decimal(10, 2)
        );
```

sql> desc employee5;

```
SQL> desc employee5;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 E_ID                                      NOT NULL NUMBER(38)
 E_NAME                                             VARCHAR2(10)
 AGE                                                NUMBER(38)
 SALARY                                             NUMBER(10,2)
```

```
sql> insert into employee5 values(101,'ramkumar',32,45000);
sql> insert into employee5 values(102,'ajay',30,30000);
sql> insert into employee5 values(103,'srajan',35,70000);
sql> insert into employee5 values(104,'stany',34,60000);
sql> insert into employee5 values(105,'stany',34,60000);
```

```
sql>commit;
```

```
sql> select * from employee5;
```

```
SQL> select * from employee5;

      E_ID E_NAME              AGE     SALARY
---------- ---------- ---------- ----------
       101 ramkumar           32      45000
       102 ajay               30      30000
       103 srajan             35      70000
       104 stany              34      60000
       105 stany              34      60000

SQL> commit;

Commit complete.
```

Q2. Declare the variables Open the cursor & extract the values from the cursor. Close the cursor.

```
sql> declare
   e_id employee5.e_id%type;
   e_name employee5.e_name%type;
   age employee5.age%type;
   salary employee5.salary%type;

-- declare cursor
cursor employee5_cursor is
   select e_id, e_name, age, salary
   from employee5;

-- open the cursor
begin
   open employee5_cursor;

   -- fetch data from cursor
   loop
      fetch employee5_cursor into e_id, e_name, age, salary;
      exit when employee5_cursor%notfound;
      -- output or use the fetched values
      dbms_output.put_line('employee id: ' || e_id || ', name: ' || e_name || ', age: ' || age || ', salary: ' || salary);
   end loop;

   -- close the cursor
   close employee5_cursor;
end;
/
```

```
SQL> DECLARE
  2      E_id Employee5.E_id%TYPE;
  3      E_name Employee5.E_name%TYPE;
  4      Age Employee5.Age%TYPE;
  5      Salary Employee5.Salary%TYPE;
  6
  7  -- Declare cursor
  8  CURSOR employee5_cursor IS
  9      SELECT E_id, E_name, Age, Salary
 10      FROM Employee5;
 11
 12  -- Open the cursor
 13  BEGIN
 14      OPEN employee5_cursor;
 15
 16      -- Fetch data from cursor
 17      LOOP
 18          FETCH employee5_cursor INTO E_id, E_name, Age, Salary;
 19          EXIT WHEN employee5_cursor%NOTFOUND;
 20          -- Output or use the fetched values
 21          DBMS_OUTPUT.PUT_LINE('Employee ID: ' || E_id || ', Name: ' || E_name || ', Age: ' || Age || ', Salary: ' ||
Salary);
 22      END LOOP;
 23
 24      -- Close the cursor
 25      CLOSE employee5_cursor;
 26  END;
 27  /
Employee ID: 101, Name: ramkumar, Age: 32, Salary: 45000
Employee ID: 102, Name: ajay, Age: 30, Salary: 30000
Employee ID: 103, Name: srajan, Age: 35, Salary: 70000
Employee ID: 104, Name: stany, Age: 34, Salary: 60000
Employee ID: 105, Name: stany, Age: 34, Salary: 60000

PL/SQL procedure successfully completed.
```

PROGRAM 6

Write a PL/SQL block of code using parameterized Cursor, that will merge the data availablein the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped.

Create table N_RollCall & O_ RollCall (id,name,roll)

```
Sql> create table N_RollCall(
        id int ,
        name varchar(10),
        roll int
        );
```

Sql> desc N_RollCall;

```
SQL> desc N_RollCall;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                                 NUMBER(38)
 NAME                                               VARCHAR2(10)
 ROLL                                               NUMBER(38)
```

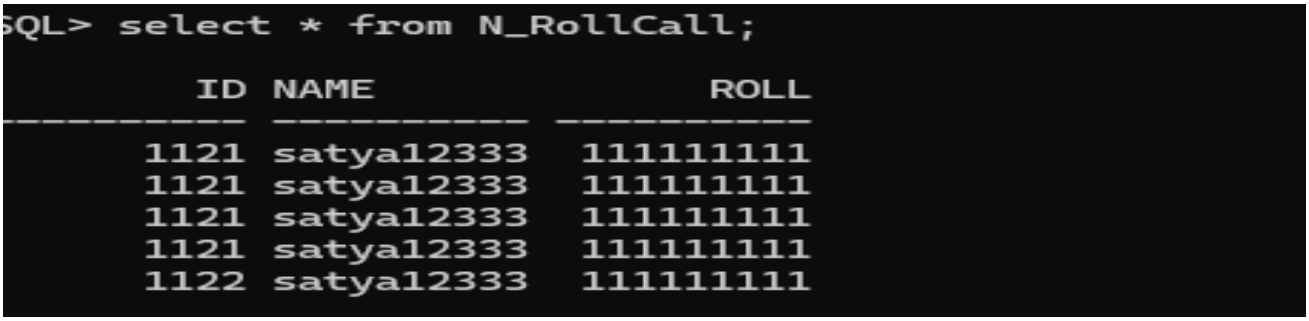
```
SQL> create table O_RollCall(
         id int ,
        name varchar(10),
        roll int
        );
```

Sql> desc O_RollCall;

```
SQL> desc O_RollCall;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                                 NUMBER(38)
 NAME                                               VARCHAR2(10)
 ROLL                                               NUMBER(38)
```

```
Sql> insert into N_RollCall  values (1121, 'satya12333',  111111111);
Sql> insert into N_RollCall  values (1121, 'satya12333',  111111111);
Sql> insert into N_RollCall  values (1121, 'satya12333',  111111111);
Sql> insert into N_RollCall  values (1121, 'satya12333',  111111111);
Sql> insert into N_RollCall  values (1122, 'satya12333',  111111111);

Sql> select * from N_RollCall;
```



```
SQL> select * from N_RollCall;

        ID NAME                 ROLL
---------- -------------------- ----------
      1121 satya12333           111111111
      1121 satya12333           111111111
      1121 satya12333           111111111
      1121 satya12333           111111111
      1122 satya12333           111111111
```

```
declare
   v_count number;
   cursor c_new_rollcall is
      select id, name, roll
      from n_rollcall;
begin
   for new_rec in c_new_rollcall loop
      -- check if the record already exists in o_rollcall
      select count(*)
      into  v_count
      from  o_rollcall
      where  id = new_rec.id;

      -- if record doesn't exist, insert it
      if v_count = 0 then
         insert into o_rollcall (id, name, roll)
         values (new_rec.id, new_rec.name, new_rec.roll);
         dbms_output.put_line('record inserted: ' || new_rec.id);
      else
```

```
        dbms_output.put_line('record skipped: ' || new_rec.id);
    end if;
  end loop;
  commit;
end;
/
```

```
SQL> DECLARE
  2      v_count NUMBER;
  3      CURSOR c_new_rollcall IS
  4          SELECT id, name, roll
  5          FROM N_RollCall;
  6  BEGIN
  7      FOR new_rec IN c_new_rollcall LOOP
  8          -- Check if the record already exists in O_RollCall
  9          SELECT COUNT(*)
 10          INTO   v_count
 11          FROM   O_RollCall
 12          WHERE  id = new_rec.id;
 13
 14          -- If record doesn't exist, insert it
 15          IF v_count = 0 THEN
 16              INSERT INTO O_RollCall (id, name, roll)
 17              VALUES (new_rec.id, new_rec.name, new_rec.roll);
 18              DBMS_OUTPUT.PUT_LINE('Record inserted: ' || new_rec.id);
 19          ELSE
 20              DBMS_OUTPUT.PUT_LINE('Record skipped: ' || new_rec.id);
 21          END IF;
 22      END LOOP;
 23      COMMIT;
 24  END;
 25  /
Record inserted: 1121
Record skipped: 1121
Record skipped: 1121
Record skipped: 1121
Record inserted: 1122

PL/SQL procedure successfully completed.
```

SQL> select * from N_RollCall;

```
SQL> select * from N_RollCall;

        ID NAME            ROLL
---------- ---------- ----------
      1121 satya12333  111111111
      1121 satya12333  111111111
      1121 satya12333  111111111
      1121 satya12333  111111111
      1122 satya12333  111111111
```

SQL> select * from O_RollCall;

```
SQL> select * from O_RollCall;

        ID NAME            ROLL
---------- ---------- ----------
      1121 satya12333  111111111
      1122 satya12333  111111111
```

Sql> commit;

```
SQL> commit;

Commit complete.
```

PROGRAM 7
MONGO DB

Question 7

Install an Open Source NoSQL Data base MongoDB & perform basic CRUD(Create, Read, Update & Delete) operations. Execute MongoDB basic Queries using CRUD operations.

Solution

1. Installing Open Source NoSQL Data base MongoDB

Please refer to the blog below which contains detailed procedure of installing Open Source

NoSQL Data base MongoDB.

Installation steps for mangodb in ubuntu:

Press alt+ctrl+t (To open Terminal)

Line by line

```
sudo apt-get update
wget -q0 -https://www.mongodb.org/static/pgp/server-4.0.asc | sudo apt-key add -
sudo apt-get install  curl
sudo apt-get install gnupg curl

curl -fsSL https://www.mongodb.org/static/pgp/server-4.4.asc |     sudo gpg -o
/usr/share/keyrings/mongodb-server-4.4.gpg      --dearmor
echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server-4.4.gpg ]
https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.4 multiverse" | sudo tee
/etc/apt/sources.list.d/mongodb-org-4.4.list

clear
sudo apt-get update

sudo apt-get install -y mongodb-org=4.4.29 mongodb-org-server=4.4.29 mongodb-org-shell=4.4.29
mongodb-org-mongos=4.4.29 mongodb-org-tools=4.4.29
ps --no-headers -o comm 1
Clear
```

2. Perform basic CRUD(Create, Read, Update & Delete) operations.

1. Start MongoDB.

Launch the MongoDB daemon using the following command:

sudo service mongodb start

2. Start the MongoDB Shell

Launch the MongoDB shell to perform basic CRUD operations.

Mongo:

To create user

3. Switch to a Database (Optional):

If you want to use a specific database, switch to that database using the **use** command. If the database doesn't exist, MongoDB will create it implicitly when you insert data into it:

```
test> use cse
switched to db cse
```

Create a table

4.Create the **ProgrammingBooks** Collection:

To create the **ProgrammingBooks** collection, use the **createCollection()** method. This step is optional because MongoDB will automatically create the collection when you insert data into it, but you can explicitly create it if needed:

```
cse> db.createCollection("ProgrammingBooks")
{ ok: 1 }
cse> show collections
ProgrammingBooks
cse>
```

This command will create an empty collection in the current database cse

5. INSERT operations

a. Insert 5 Documents into the **ProgrammingBooks** Collection:

Now, insert 5 documents representing programming books into the **ProgrammingBooks** collection using the **insertMany()** method:

```
cse> db.ProgrammingBooks.insertMany([
...   {
...     title: "Clean Code: A Handbook of Agile Software Craftsmanship",
...     author: "Robert C. Martin",
...     category: "Software Development",
...     year: 2008
...   },
...   {
...     title: "JavaScript: The Good Parts",
...     author: "Douglas Crockford",
...     category: "JavaScript",
...     year: 2008
...   },
...   {
...     title: "Design Patterns: Elements of Reusable Object-Oriented Software",
...     author: "Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides",
...     category: "Software Design",
...     year: 1994
...   },
...   {
...     title: "Introduction to Algorithms",
...     author: "Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein",
...     category: "Algorithms",
...     year: 1990
...   },
...   {
...     title: "Python Crash Course: A Hands-On, Project-Based Introduction to Programming",
...     author: "Eric Matthes",
...     category: "Python",
...     year: 2015
...   }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('666c18655541dd4ed5cdce00'),
    '1': ObjectId('666c18655541dd4ed5cdce01'),
    '2': ObjectId('666c18655541dd4ed5cdce02'),
    '3': ObjectId('666c18655541dd4ed5cdce03'),
    '4': ObjectId('666c18655541dd4ed5cdce04')
  }
}
cse>
```

6. Read (Query) Operations

a. Find All Documents

To retrieve all documents from the **ProgrammingBooks** collection:

```
cse> db.ProgrammingBooks.find().pretty()
[
  {
    _id: ObjectId('666c18655541dd4ed5cdce00'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: ObjectId('666c18655541dd4ed5cdce01'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  },
  {
    _id: ObjectId('666c18655541dd4ed5cdce02'),
    title: 'Design Patterns: Elements of Reusable Object-Oriented Software',
    author: 'Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides',
    category: 'Software Design',
    year: 1994
  },
  {
    _id: ObjectId('666c18655541dd4ed5cdce03'),
    title: 'Introduction to Algorithms',
    author: 'Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein',
    category: 'Algorithms',
    year: 1990
  },
  {
    _id: ObjectId('666c18655541dd4ed5cdce04'),
    title: 'Python Crash Course: A Hands-On, Project-Based Introduction to Programming',
    author: 'Eric Matthes',
    category: 'Python',
    year: 2015
  }
]
cse>
```

Find Documents Matching a Condition

To find books published after the year 2000:

```
cse> db.ProgrammingBooks.find({ year: { $gt: 2000 } }).pretty()
[
  {
    _id: ObjectId('666c18655541dd4ed5cdce00'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin',
    category: 'Software Development',
    year: 2008
  },
  {
    _id: ObjectId('666c18655541dd4ed5cdce01'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'JavaScript',
    year: 2008
  },
  {
    _id: ObjectId('666c18655541dd4ed5cdce04'),
    title: 'Python Crash Course: A Hands-On, Project-Based Introduction to Programming',
    author: 'Eric Matthes',
    category: 'Python',
    year: 2015
  }
]
cse>
```

7. Update Operations

a. Update a Single Document

To update a specific book (e.g., change the author of a book):

```
cse> db.ProgrammingBooks.updateOne(
...   { title: "Clean Code: A Handbook of Agile Software Craftsmanship" },
...   { $set: { author: "Robert C. Martin (Uncle Bob)" } }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
cse>
```

b. Update Multiple Documents

To update multiple books (e.g., update the category of books published before 2010):

```
cse> db.ProgrammingBooks.updateMany(
...    { year: { $lt: 2010 } },
...    { $set: { category: "Classic Programming Books" } }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 4,
  modifiedCount: 4,
  upsertedCount: 0
}
cse>
```

```
cse> db.ProgrammingBooks.find({ year: { $lt: 2010 } }).pretty()
[
  {
    _id: ObjectId('666c18655541dd4ed5cdce00'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin (Uncle Bob)',
    category: 'Classic Programming Books',
    year: 2008
  },
  {
    _id: ObjectId('666c18655541dd4ed5cdce01'),
    title: 'JavaScript: The Good Parts',
    author: 'Douglas Crockford',
    category: 'Classic Programming Books',
    year: 2008
  },
  {
    _id: ObjectId('666c18655541dd4ed5cdce02'),
    title: 'Design Patterns: Elements of Reusable Object-Oriented Software',
    author: 'Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides',
    category: 'Classic Programming Books',
    year: 1994
  },
  {
    _id: ObjectId('666c18655541dd4ed5cdce03'),
    title: 'Introduction to Algorithms',
    author: 'Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein',
    category: 'Classic Programming Books',
    year: 1990
  }
]
cse>
```

8. Delete Operations

a. Delete a Single Document

To delete a specific book from the collection (e.g., delete a book by title):

b. Delete Multiple Documents

To delete multiple books based on a condition (e.g., delete all books published before 1995):

```
cse> db.ProgrammingBooks.deleteOne({ title: "JavaScript: The Good Parts" })
{ acknowledged: true, deletedCount: 0 }
cse> db.ProgrammingBooks.deleteMany({ year: { $lt: 1995 } })
{ acknowledged: true, deletedCount: 2 }
cse> db.ProgrammingBooks.find().pretty()
[
  {
    _id: ObjectId('666c18655541dd4ed5cdce00'),
    title: 'Clean Code: A Handbook of Agile Software Craftsmanship',
    author: 'Robert C. Martin (Uncle Bob)',
    category: 'Classic Programming Books',
    year: 2008
  },
  {
    _id: ObjectId('666c18655541dd4ed5cdce04'),
    title: 'Python Crash Course: A Hands-On, Project-Based Introduction to Programming',
    author: 'Eric Matthes',
    category: 'Python',
    year: 2015
  }
]
cse>
```

c. Delete All Documents in the Collection:

To delete all documents in a collection (e.g., **ProgrammingBooks**), use the **deleteMany()** method with an empty filter **{}**:

```
cse> db.ProgrammingBooks.deleteMany({})
{ acknowledged: true, deletedCount: 2 }
cse> db.ProgrammingBooks.find().pretty()

cse>
```

```
cse> show dbs
admin    40.00 KiB
config   96.00 KiB
cse      48.00 KiB
local    40.00 KiB
cse>
```

```
cse> db.dropDatabase()
{ ok: 1, dropped: 'cse' }
cse> show dbs
admin   40.00 KiB
config  96.00 KiB
local   40.00 KiB
cse>
```

ADDITIONAL PROGRAMS

1) Consider the following relations:

STUDENT (snum: integer, sname: string, major: string, level: string, age: integer)

CLASS (name: string, meets at: string, room: string, d: integer)

ENROLLED (snum: integer, cname: string)

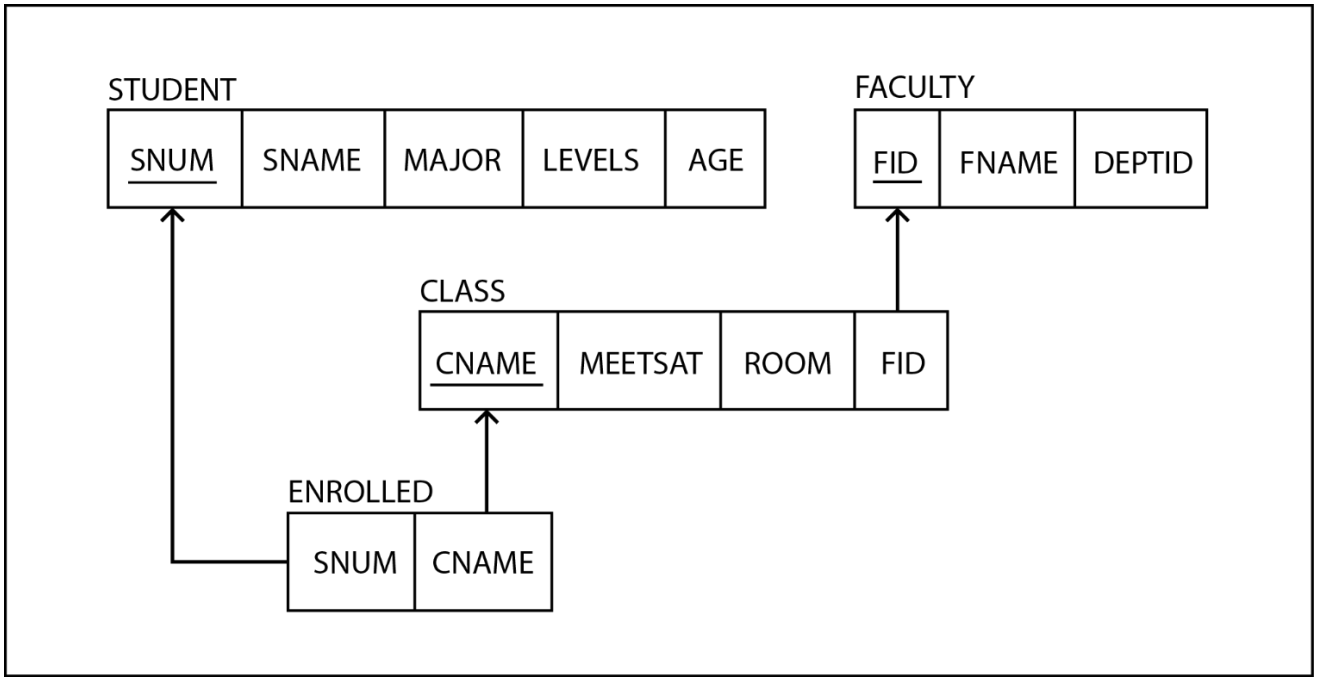
FACULTY (fid: integer, fname: string, deptid: integer)

The meaning of these relations is straightforward; for example, Enrolled has one record per student-class pair such that the student is enrolled in the class. Level is a two character code with 4 different values (example: Junior: JR etc.)

Write the following queries in SQL. No duplicates should be printed in any of the answers.

- **i.** Find the names of all Juniors (level = JR) who are enrolled in a class taught by Prof. Harshith
- **ii.** Find the names of all classes that either meet in room R128 or have five or more Students enrolled.
- **iii.** Find the names of all students who are enrolled in two classes that meet at the same time.
- **iv.** Find the names of faculty members who teach in every room in which some class is taught.
- **v.** Find the names of faculty members for whom the combined enrollment of the courses that they teach is less than five.

SCHEMA DIAGRAM



Create the above tables by properly specifying the primary keys and foreign keys.

Enter at least five tuples for each relation.

```
SQL> create table student(
        snum integer,
        sname varchar( 10),
        major varchar( 10),
        levels varchar( 2),
        age number( 2),
        primary key(snum));
```

Table created.

```
sql>insert into student values('&snum','&sname','&major','&levels','&age');
sql> select * from student;
```

SNUM	SNAME	MAJOR	LEVELS	AGE
101	LIKI	MATHS	JR	21
102	LAVI	MATHS	SR	22
103	GOPI	SE	SR	27
104	CHANDU	OR	JR	20
105	LAHI	SE	SR	27

5 rows selected.

```
sql>create table faculty (
    fid number( 5),
    fname varchar( 10),
    deptid number( 5),
    primary key( fid));
```

table created.

```
sql> insert into faculty values(&fid,'&fname','&depid');
sql> select * from faculty;
```

FID	FNAME	DEPID
501	HARSHITH	401
502	SATHYA	402
503	RAMYA	401
504	SRINATH	405
505	LAHARI	403
506	ASHA	404
507	USHA	402

7 rows selected.

sql> create table class

(

cname varchar(20),

meetsat varchar(10),

room varchar(5),

fid number(5),

primary key(cname),

foreign key(fid)references faculty(fid)

);

table created.

sql>insert into class values('&cname','&meetsat','&room','&fid');

sql>select * from class;

CNAME	MEETSAT	ROOM	FID
3A	10.00	R128	501
3B	10.00	R02	502
4A	11.00	R03	503
4B	11.00	R128	502
5A	12.00	R04	504
5B	11.00	R128	501
6A	10.00	R02	501
6B	12.00	R03	501
7A	10.00	R04	501

9 rows selected.

sql> create table enrolled

(

snum integer,

cname varchar(20),

foreign key(cname) references class(cname),

foreign key(snum) references student(snum)

);

table created.

sql> insert into enrolled values(&snum,'&cname');

sql> select * from enrolled;

SNUM	CNAME
101	3A
101	3B
101	4A
101	4B
101	5A
101	5B
101	6A
101	6B
102	3A
102	3B
103	3A
104	4A
105	5A
101	5B
103	4B
105	3A
104	3A

17 rows selected.

sql>select distinct sname

from student s,enrolled e,class c,faculty f

where s.snum=e.snum and e.cname=c.cname and

c.fid=f.fid and

fname='harshith' and levels='jr';

SNAME
CHANDU
LIKI

sql>select distinct c.cname

from class c,enrolled e

where c.cname=e.cname and room='r128' or c.cname in

(select cname from enrolled group by cname having count(snum)>=5);

CNAME
3A
4B
5B

i.

sql> select distinct s.sname

from student s where s.snum in

(select e1.snum from enrolled e1,enrolled e2,class c1,class c2

where e1.cname=c1.cname and e2.cname=c2.cname and

e1.snum=e2.snum and

e1.cname<>c2.cname and c1.meetsat=c2.meetsat);

SNAME
LAVI
LIKI

ii.

sql> select f.fname

from faculty f

where not exists((select c.room from class c)

minus (select c1.room from class c1 where c1.fid=f.fid));

FNAME
HARSHITH

v.

sql> select distinct f.fname

from faculty f

where 5>(select count(e.snum)

from class c,enrolled e

where c.cname=e.cname and f.fid=c.fid);

FNAME
AHSA
LAHARI
RAMYA
SATHYA
SRINATH
USHA

2) The following relations keep track of **AIRLINE FLIGHT INFORMATION**:

FLIGHTS (no: integer, from: string, to: string, distance: integer, Departs: time, arrives: time, price: real)

AIRCRAFT (aid: integer, aname: string, cruisingrange: integer)

CERTIFIED (eid: integer, aid: integer)

EMPLOYEES (eid: integer, ename: string, salary: integer)

Note that the Employees relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft, and only pilots are certified to fly.

Write each of the following queries in SQL.

i. Find the names of aircraft such that all pilots certified to operate them have salaries more than Rs.80,000.

ii. For each pilot who is certified for more than three aircrafts, find the eidand the maximum cruisingrange of the aircraft for which she or he is certified.

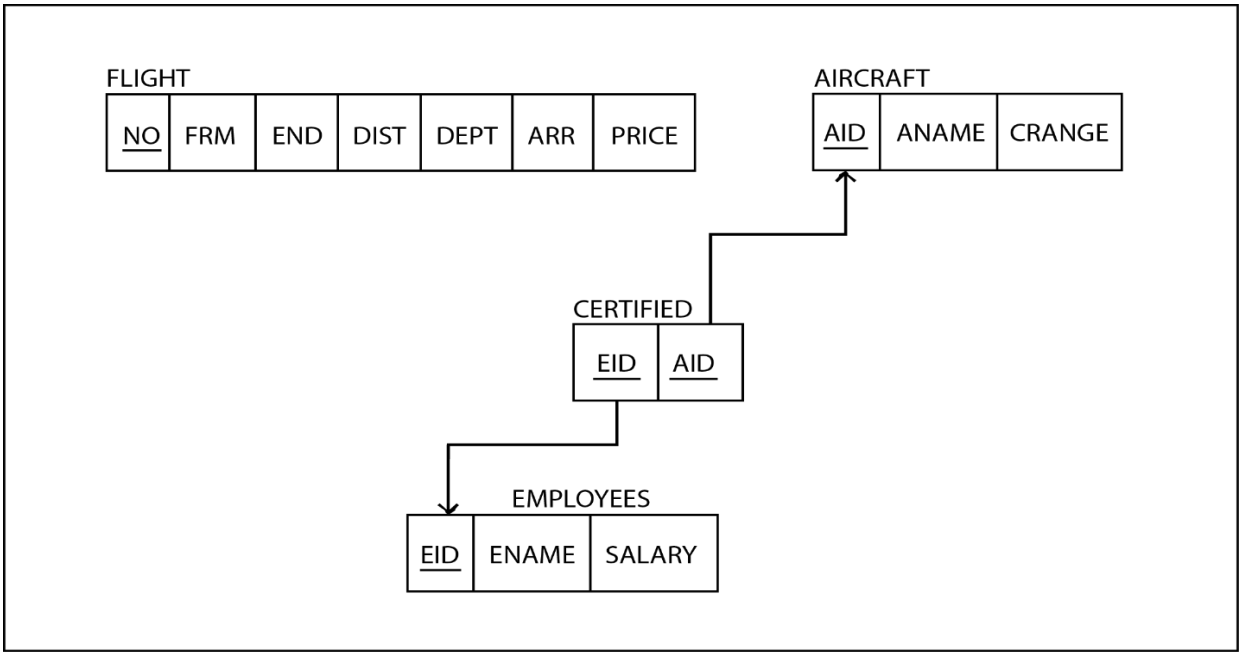
iii. Find the names of pilots whose salary is less than the price of the cheapest route from Bengaluru to Frankfurt.

iv. For all aircraft with cruisingrange over 1000 Kms. Find the name of the aircraft and the average salary of all pilots certified for this aircraft.

v. Find the names of pilots certified for some Boeing aircraft.

vi. Find the aids of all aircraft that can be used on routes from Bengaluru to New Delhi.

SCHEMA DIAGRAM



Create the above tables by properly specifying the primary keys and foreign keys.Enter at least five tuples for each relation.

```
sql>create table flight(
      no integer,
      frm varchar( 20),
      end varchar( 20),
      dist integer,
      dept date,
      arr date,
      price real,
     primary key(no));
```

table created.

sql> insert into flight values('&no','&frm','&end','&dist','&dept','&arr','&price');

sql> select * from flight;

NO	FRM	END	DIST	DEPT	ARR	PRICE
255	BANGALORE	FRANKFURT	200	01-AUG-11	01-AUG-11	5000
256	BANGALORE	FRANKFURT	200	01-AUG-11	01-AUG-11	8000
257	BANGALORE	DELHI	200	01-AUG-11	01-AUG-11	5000
258	BANGALORE	DELHI	200	01-AUG-11	01-AUG-11	6000
259	BANGALORE	MANGALORE	200	01-AUG-11	01-AUG-11	8000

sql> create table aircraft (
      aid int,
      aname varchar(15),
      crange integer,
      primary key(aid));
      table created

sql>insert into aircraft values('&aid','&aname','&crange');

sql>select * from aircraft;

AID	ANAME	CRANGE
685	BOEING15	1000
686	BOEING10	2000
687	SKYTRAIN	1000
688	AVENGER	100

4 rows selected.

sql> create table employees

(eid int,

ename varchar(15),

salary real,

primary key(eid));

table created.

sql> insert into employees values('&eid','&ename','&salary');

sql> select * from employees;

EID	ENAME	SALARY
101	ASHA	90000
102	ARUN	85000
103	ANAND	3000
104	RAMYA	40000

4 rows selected.

sql> create table certified(

eid int,

aid int,

primary key(eid,aid));

table created.

sql> insert into certified values('&eid','&aid');

sql> select * from certified;

EID	AID
101	685
101	686
101	687
101	688
102	685
103	686
103	687

7 rows selected.

i. Find the names of aircraft such that all pilots certified to operate them have salaries more than Rs.80, 000.

```
SQL> select distinct a.aname
     from aircraft a,certified c,employees e
     where c.eid=e.eid and
     a.aid=c.aid and
     salary in(select
     salary from employees
     where salary> 80000);
```

ANAME
AVENGER
BOEING10
BOEING15
SKYTRAIN

ii) For each pilot who is certified for more than three aircrafts, find the eid and the maximum cruisingrange of the aircraft for which she or he is certified.

```
sql> select c.eid,max(a.crange)
     from certified c,aircraft a
     where c.aid=a.aid group by c.eid having count(c.aid)>3;
```

EID	MAX(A.ORANGE)
101	2000

iii) Find the names of pilots whose *salary* is less than the price of the cheapest route from Bengaluru to Frankfurt.

```
SQL> select e.ename
     from employees e
     where e.salary<(select min(price)
     from flight where frm='bangalore' and end='frankfurt');
```

ENAME
ANAND
RAMYA

iv) For all aircraft with *cruisingrange* over 1000 Kms. Find the name of the aircraft and the average salary of all pilots certified for this aircraft.

```
SQL> select name,avgsal
     from(select a.aid,a.aname as name,avg(e.salary) as avgsal
     from aircraft a,certified c,employees e
     where a.aid=c.aid and c.eid=e.eid and crange>1000
     group by a.aid,a.aname);
```

NAME	AVGSAL
BOEING10	46500

v) Find the names of pilots certified for some Boeing aircraft.

```
SQL> select distinct e.ename
     from employees e,certified c,aircraft a
     where a.aid=c.aid and c.eid=e.eid and a.aname like 'boeing%';
```

ENAME
ANAND
ARUN
ASHA

vi) Find the *aid*s of all aircraft that can be used on routes from Bengaluru to New Delhi.

```
SQL> select a.aid
     from aircraft a
     where a.crange>(select min(dist)
     from flight
     where frm='bangalore' and end='delhi');
```

AID
685
686
687

Viva Questions

1. **What is SQL?**

   Structured Query Language

2. **What is database?**

   A database is a logically coherent collection of data with some inherent meaning, representing some aspect of real world and which is designed, built and populated with data for a specific purpose.

3. **What is DBMS?**

   It is a collection of programs that enables user to create and maintain a database. In other words it is general-purpose software that provides the users with the processes of defining, constructing and manipulating the database for various applications.

4. **What is a Database system?**

   The database and DBMS software together is called as Database system.

5. **Advantages of DBMS?**

   - Redundancy is controlled.
   - Unauthorized access is restricted.
   - Providing multiple user interfaces.
   - Enforcing integrity constraints.
   - Providing backup and recovery.

6. **Disadvantage in File Processing System?**

   - Data redundancy & inconsistency.
   - Difficult in accessing data.
   - Data isolation.
   - Data integrity.
   - Concurrent access is not possible.
   - Security Problems.

7. **Describe the three levels of data abstraction?**

There are three levels of abstraction:

- Physical level: The lowest level of abstraction describes how data are stored.
- Logical level*:* The next higher level of abstraction, describes what data are stored in database and what relationship among those data.
- View level: The highest level of abstraction describes only part of entire database.

8. **Define the "integrity rules"**

There are two Integrity rules.

- Entity Integrity: States that “Primary key cannot have NULL value”
- Referential Integrity: States that “Foreign Key can be either a NULL value or should be Primary Key value of other relation.

9. **What is extension and intension?**

Extension - It is the number of tuples present in a table at any instance. This is time dependent.

Intension -It is a constant value that gives the name, structure of table and the constraints laid on it.

10. **What is Data Independence?**

Data independence means that “the application is independent of the storage structure and access strategy of data”. In other words, The ability to modify the schema definition in one level should not affect the schema definition in the next higher level.

Two types of Data Independence:

- Physical Data Independence: Modification in physical level should not affect the logical level.
- Logical Data Independence: Modification in logical level should affect the view level.

NOTE: Logical Data Independence is more difficult to achieve

11. **What is a view? How it is related to data independence?**

A view may be thought of as a virtual table, that is, a table that does not really exist in its own right but is instead derived from one or more underlying base table. In other words, there is no stored file that direct represents the view instead a definition of view is stored in data dictionary.

Growth and restructuring of base tables is not reflected in views. Thus the view can insulate users from the effects of restructuring and growth in the database. Hence accounts for logical data independence.

12. **What is Data Model?**

A collection of conceptual tools for describing data, data relationships data semantics and constraints

13. **What is E-R model?**

This data model is based on real world that consists of basic objects called entities and of relationship among these objects. Entities are described in a database by a set of attributes.

14. **What is Object Oriented model?**

This model is based on collection of objects. An object contains values stored in instance variables within the object. An object also contains bodies of code that operate on the object. These bodies of code are called methods. Objects that contain same types of values and the same methods are grouped together into classes.

15. **What is an Entity?**

It is an 'object' in the real world with an independent existence.

16. **What is an Entity type?**

It is a collection (set) of entities that have same attributes.

17. **What is an Entity set?**

It is a collection of all entities of particular entity type in the database.

18. **What is an Extension of entity type?**

The collections of entities of a particular entity type are grouped together into an entity set.

19. **What is an attribute?**

It is a particular property, which describes the entity.

20. **What is a Relation Schema and a Relation?**

R and the list of attributes A_i that it contains. A relation is defined as a set of tuples. Let r be the relation which contains set tuples (t1, t2, t3, ...,tn). Each tuple is an ordered list of n- values t=(v1,v2, ..., vn).

21. **What is degree of a Relation?**

It is the number of attribute of its relation schema.

22. **What is Relationship?**

The collection (or set) of similar relationships.

24. **What is Relationship type?**

Relationship type defines a set of associations or a relationship set among a given set of entity types.

25. **What is degree of Relationship type?**

It is the number of entity type participating.

26. **What is DDL (Data Definition Language)?**

A data base schema is specified by a set of definitions expressed by a special language called DDL.

27. **What is VDL (View Definition Language)?**

It specifies user views and their mappings to the conceptual schema.

28. **What is SDL (Storage Definition Language)?**

This language is to specify the internal schema. This language may specify the mapping between two schemas.

29. **What is Data Storage - Definition Language?**

The storage structures and access methods used by database system are specified by a set of definition in a special type of DDL called data storage- definition language.

30. What is DML (Data Manipulation Language)?

This language that enable user to access or manipulate data as organized by appropriate data model.

- Procedural DML or Low level: DML requires a user to specify what data are needed and how to get those data.
- Non-Procedural DML or High level: DML requires a user to specify what data are needed without specifying how to get those data.

31. What is Relational Calculus?

It is an applied predicate calculus specifically tailored for relational databases proposed by E.F. Codd. E.g. of languages based on it are DSL, ALPHA, QUEL.

32. What is normalization?

It is a process of analyzing the given relation schemas based on their Dependencies (FDs) and primary key to achieve the properties

- Minimizing redundancy
- Minimizing insertion, deletion and update anomalies.

33. What is 1 NF (Normal Form)?

The domain of attribute must include only atomic (simple, indivisible) values.

34. What is Fully Functional dependency? →

It is based on concept of full functional dependency. A functional dependency X Y is fully functional dependency if removal of any attribute A from X means that the dependency does not hold any more.

35. What is 2NF?

A relation schema R is in 2NF if it is in 1NF and every non-prime attribute A in R is fully functionally dependent on primary key.

36. What is 3NF?

A relation schema R is in 3NF if it is in 2NF and for every FD X → A either of the following is true

- X is a Super-key of R.
- A is a prime attribute of R.

In other words, if every non prime attribute is non-transitively dependent on primary key.

37. What is BCNF (Boyce-Codd Normal Form)?

A relation schema R is in BCNF if it is in 3NF and satisfies additional constraints that for every FD X A, X must be a candidate key.

38. What is 4NF?

A relation schema R is said to be in 4NF if for every Multivalued dependency X → Y that holds over R, one of following is true

- X is subset or equal to (or) XY = R.
- X is a super key.

39. What is 5NF?

A Relation schema R is said to be 5NF if for every join dependency {R1, R2, ...,Rn} that holds R, one the following is true

- Ri = R for some i.
- The join dependency is implied by the set of FD, over R in which the left side is key of R.