



SHRI VILEPARLE KELAVANI MANDAL'S
SHRI BHAGUBHAI MAFATLAL POLYTECHNIC



Computer Engineering Department

A

Laboratory Manual

DATABASE MANAGEMENT SYSTEM
(DBS198917)

Semester: IV

Vision /Mission Statements

Institute Vision

SBM Polytechnic aspires to be the lead institute in providing need based technical education.

Institute Mission

- **To provide state-of-the-art infrastructure and latest equipments for providing a stimulating learning environment.**
- **To prepare students to meet the dynamic needs of the industry by periodic reviewing and upgradation of curriculum through an interactive process with industry.**
- **To inculcate a spirit of excellence in terms of academic performance, research and innovation in faculty by providing appropriate support and incentive systems.**
- **To promote and support Co-Curricular, extra-curricular activities and industry interaction to make students socially sensitive and employable.**

Department Vision

Create a sustainable academic environment to produce highly competent computer professionals of the future

Department Mission

- **To expose students to latest tools and technologies in computing**
- **To foster the professional development of students by providing excellence in education.**
- **To adapt rapid advancements in computing by engaging students in the lifelong learning.**
- **To inculcate sound ethical, moral and social values amongst students for benefit of the society.**

Program Educational objectives

PEO1-Identify frame and solve computing problems by applying knowledge in Computer engineering

PEO2- Promote lifelong learning by integrating academic knowledge and practical applications

PEO3- Depict effective team work and practical skills for holistic development

Program Outcomes

1. **Basic and Discipline specific knowledge: Apply knowledge of basic mathematics, science and engineering fundamentals and engineering specialization to solve the engineering problems**
2. **Problem analysis: Identify and analyse well-defined engineering problems using codified standard methods**

3. Design/ development of solutions: Design solutions for well-defined technical problems and assist with the design of systems components or processes to meet specified needs
4. Engineering Tools, Experimentation and Testing: Apply modern engineering tools and appropriate technique to conduct standard tests and measurements
5. Engineering practices for society, sustainability and environment: Apply appropriate technology in context of society, sustainability, environment and ethical practices.
6. Project Management: Use engineering management principles individually, as a team member or a leader to manage projects and effectively communicate about well-defined engineering activities
7. Life-long learning: Ability to analyse individual needs and engage in updating in the context of technological changes

Program Specific Outcomes

PSO1: Demonstrate the fundamental knowledge in the areas of Operating system, Web Technology, Microprocessor based system and IOT by applying programming skills and developing applications

PSO2: Administer and manage Open source, Networking, Security and Database domains to enhance student growth

Course Outcomes

1. Extrapolate the fundamental elements of relational database management systems.
2. Construct a real-world database using SQL.
3. Analyze, database storage and normalize database to meet business requirements.
4. Conceptualize query processing, transactions, concurrency control, backup and recovery.

CO-PO Mapping

Course and Code	Course Outcomes	Programme Outcomes							Programme Specific Outcomes	
		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PSO1	PSO2
Database Management System DBS198917	C209.1	3							3	
	C209.2	2	2	3	2				3	
	C209.3	2	3	1	2				3	
	C209.4	3							2	
	Avg.	2.5	2.5	2	2				2.75	

1=slightly, 2=moderately, 3=substantially

INDEX

Sr. No.	Name of Experiment	POs Covered	Page No.
1	Draw an ER Model for college database.	PO1,PSO1	6-10
2	Creating of table with constraint and insertion of data.	PO1,PO2, PO3,PO4,PSO1	11-16
3	Running simple SQL queries. (Select, Distinct, Desc, where)	PO1,PO2, PO3,PO4,PSO1	17-20
4	Execution of Alter Update, Delete and Drop.	PO1,PO2, PO3,PO4,PSO1	21-23
5	Implementation of aggregate and character function.	PO1,PO2, PO3,PO4,PSO1	24-26
6	Implementation of various clauses in SQL	PO1,PO2, PO3,PO4,PSO1	27-29
7	Execution of string, comparison and set operations.	PO1,PO2, PO3,PO4,PSO1	30-32
8	Implementation of various types of joins.	PO1,PO2, PO3,PO4,PSO1	33-35
9	Implementation of views and triggers.	PO1,PO2, PO3,PO4,PSO1	36-37
10	Implement subqueries in SQL	PO1,PO2, PO3,PO4,PSO1	38
11	Implement Normalization on the table in a database	PO1,PSO1	39-40

Experiment 1: ER Diagram of a Database

Aim: To draw ER Diagram of a college database

Theory:

Database Management System

This model is like a hierarchical tree structure, used to construct a hierarchy of records in the form of nodes and branches. The data elements present in the structure have Parent-Child relationship. Closely related information in the parent-child structure is stored together as a logical unit. A parent unit may have many child units, but a child is restricted to have only one parent.

The drawbacks of this model are:

The hierarchical structure is not flexible to represent all the relationship proportions, which occur in the real world.

It cannot demonstrate the overall data model for the enterprise because of the non-availability of actual data at the time of designing the data model. It cannot represent the Many-to-Many relationship.

Network Model

It supports the One-To-One and One-To-Many types only. The basic objects in this model are Data Items, Data Aggregates, Records and Sets.

It is an improvement on the Hierarchical Model. Here multiple parent-child relationships are used. Rapid and easy access to data is possible in this model due to multiple access paths to the data elements.

Relational Model

Does not maintain physical connection between relations Data is organized in terms of rows and columns in a table The position of a row and/or column in a table is of no importance The intersection of a row and column must give a single value .

Features of an RDBMS

The ability to create multiple relations and enter data into them An attractive query language
Retrieval of information stored in more than one table

An RDBMS product has to satisfy at least seven of the 12 rules of Codd to be accepted as a full- fledged RDBMS.

Relational Database Management System

RDBMS is acronym for Relation Database Management System. Dr. E. F. Codd first introduced the Relational Database Model in 1970. The Relational model allows data to be represented in a simple row- column. Each data field is considered as a column and each record is considered as a row. Relational Database is more or less similar to Database Management System. In relational model there is relation between their data elements. Data is stored in tables. Tables have columns, rows and names. Tables can be related to each other if each has a column with a common type of information.

The most famous RDBMS packages are Oracle, Sybase and Informix.

Simple example of Relational model is as follows :

Student Details Table

<u>Roll no</u>	<u>Sname</u>	<u>S Address</u>
1	Rahul	Satelite
2	Sachin	Ambawadi
3	Saurav	Naranpura

Student Marksheet Table

<u>Rollno</u>	<u>Sub1</u>	<u>Sub2</u>	<u>Sub3</u>
1	78	89	94
2	54	65	77
3	23	78	46

Here, both tables are based on students details. Common field in both tables is Rollno. So we can say both tables are related with each other through Rollno column.

Degree of Relationship

One to One (1:1)

One to Many or Many to One (1:M / M: 1)

Many to Many (M: M)

The Degree of Relationship indicates the link between two entities for a specified occurrence of each.

One to One Relationship: (1:1)

Student Has Roll No.

One student has only one Rollno. For one occurrence of the first entity, there can be, at the most one related occurrence of the second entity, and vice-versa.

One to Many or Many to One Relationship: (1:M/M: 1)

1 M

Course Contains Students

As per the Institutions Norm, One student can enroll in one course at a time however, in one course, there can be more than one student.

For one occurrence of the first entity there can exist many related occurrences of the second entity and for every occurrence of the second entity there exists only one associated occurrence of the first.

Many to Many Relationship: (M:M)


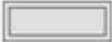










M M

Students Appears Tests

The major disadvantage of the relational model is that a clear-cut interface cannot be determined. Reusability of a structure is not possible. The Relational Database now accepted model on which major database system are built.

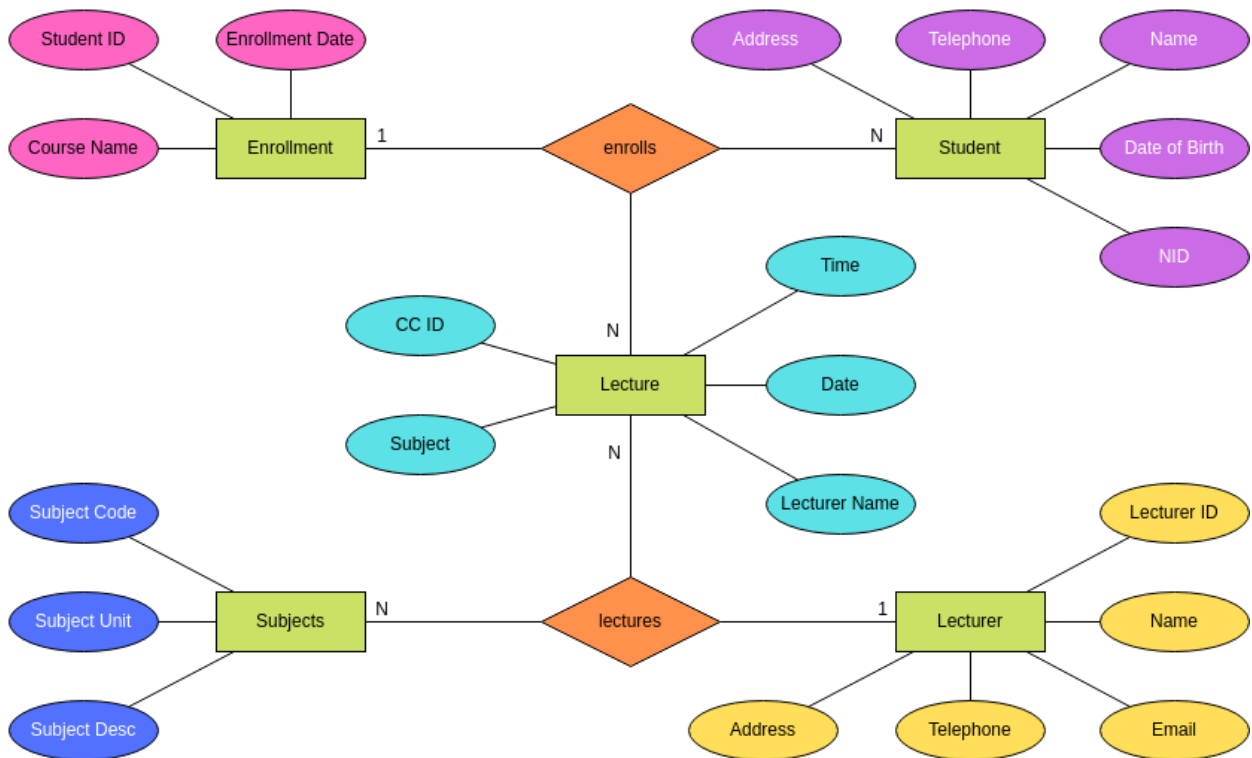
Oracle has introduced added functionality to this by incorporated object-oriented capabilities. Now it is known as Object Relational Database Management System (ORDBMS). Object-oriented concept is added in Oracle8.

Some basic rules have to be followed for a DBMS to be relational. They are known as Codd's rules, designed in such a way that when the database is ready for use it encapsulates the relational theory to its full potential. The shapes of ER Diagram are as follows:

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of E_2 in R
	Cardinality Ratio 1: N for $E_1:E_2$ in R
	Structural Constraint (min, max) on Participation of E in R

Procedure:

Draw an ER Diagram of a college



Conclusion: To be written by student.

Experiment 2: Creating of table with constraint and insertion of data.

Aim:

Creating of table with constraint and insertion of data.

Theory:

SQL (Structured Query Language):

Structured Query Language is a database computer language designed for managing data in relational database management systems (RDBMS), and originally based upon Relational Algebra. Its scope includes data query and update, schema creation and modification, and data access control.

SQL was one of the first languages for Edgar F. Codd's relational model and became the most widely used language for relational databases.

- IBM developed SQL in mid of 1970's.
- Oracle incorporated in the year 1979.
- SQL used by IBM/DB2 and DS Database Systems. □ SQL adopted as standard language for RDBS by ANSI in 1989.

DATA TYPES:

1. CHAR (Size): This data type is used to store character strings values of fixed length. The size in brackets determines the number of characters the cell can hold. The maximum number of character is 255 characters.

2. VARCHAR (Size) / VARCHAR2 (Size): This data type is used to store variable length alphanumeric data. The maximum character can hold is 2000 character.

3. NUMBER (P, S): The NUMBER data type is used to store number (fixed or floating point). Number of virtually any magnitude may be stored up to 38 digits of precision.

Number as large as 9.99×10^{124} . The precision (p) determines the number of places to the right of the decimal. If scale is omitted then the default is zero. If precision is omitted, values are stored with their original precision up to the maximum of 38 digits.

4. DATE: This data type is used to represent date and time. The standard format is DD-MM-YY as in 17-SEP-2009. To enter dates other than the standard format, use the appropriate functions. Date time stores date in the 24-Hours format. By default the time in a date field is 12:00:00 am, if no time portion is specified. The default date for a date field is the first day the current month.

5. LONG: This data type is used to store variable length character strings containing up to 2GB. Long data can be used to store arrays of binary data in ASCII format. LONG values cannot be indexed, and the normal character functions such as SUBSTR cannot be applied.

6. RAW: The RAW data type is used to store binary data, such as digitized picture or image. Data loaded into columns of these data types are stored without any further conversion. RAW data type can have a maximum length of 255 bytes. LONG RAW data type can contain up to 2GB.

SQL language is sub-divided into several language elements, including:

- *Clauses*, which are in some cases optional, constituent components of statements and queries.
- *Expressions*, which can produce either scalar values or tables consisting of columns and rows of data.
- *Predicates* which specify conditions that can be evaluated to SQL three-valued logic (3VL) Boolean truth values and which are used to limit the effects of statements and queries, or to change program flow.
- *Queries* which retrieve data based on specific criteria.
- *Statements* which may have a persistent effect on schemas and data, or which may control transactions, program flow, connections, sessions, or diagnostics.
- SQL statements also include the semicolon (";") statement terminator. Though not required on every platform, it is defined as a standard part of the SQL grammar.

- Insignificant white space is generally ignored in SQL statements and queries, making it easier to format SQL code for readability.

There are five types of SQL statements. They are:

1. DATA DEFINITION LANGUAGE (DDL)
2. DATA MANIPULATION LANGUAGE (DML)
3. DATA RETRIEVAL LANGUAGE (DRL)
4. TRANSATIONAL CONTROL LANGUAGE (TCL)
5. DATA CONTROL LANGUAGE (DCL)

1. CREATE:

(a)CREATE TABLE: This is used to create a new relation (table)

Syntax: CREATE TABLE <relation_name/table_name >

(field_1 data_type(size),field_2 data_type(size), ..);

CONSTRAINTS:

Constraints are used to specify rules for the data in a table. If there is any violation between the constraint and the data action, the action is aborted by the constraint. It can be specified when the table is created (using CREATE TABLE statement) or after the table is created (using ALTER TABLE statement).

1. NOT NULL: When a column is defined as NOTNULL, then that column becomes a mandatory column. It implies that a value must be entered into the column if the record is to be accepted for storage in the table.

Syntax:

CREATE TABLE Table_Name (column_name data_type (size) **NOT NULL**,);

Example:

CREATE TABLE student (sno **NUMBER(3)NOT NULL**, name **CHAR(10)**);

2. UNIQUE: The purpose of a unique key is to ensure that information in the column(s) is unique i.e. a value entered in column(s) defined in the unique constraint must not be repeated across the column(s). A table may have many unique keys.

Syntax:

CREATE TABLE Table_Name(column_name data_type(size) **UNIQUE**,);

Example:

CREATE TABLE student (sno **NUMBER(3) UNIQUE**, name **CHAR(10)**);

3. CHECK: Specifies a condition that each row in the table must satisfy. To satisfy the constraint, each row in the table must make the condition either TRUE or unknown (due to a null).

Syntax:

CREATE TABLE Table_Name(column_name data_type(size) **CHECK(logical expression)**,); **Example:**

CREATE TABLE student (sno **NUMBER (3)**, name **CHAR(10)**,class **CHAR(5)**,**CHECK**(class **IN**('CSE','CAD','VLSI'));

4. PRIMARY KEY: A field which is used to identify a record uniquely. A column or combination of columns can be created as primary key, which can be used as a reference from other tables. A table contains primary key is known as Master Table.

- ✓ It must uniquely identify each record in a table.
- ✓ It must contain unique values.
- ✓ It cannot be a null field.
- ✓ It cannot be multi port field.
- ✓ It should contain a minimum no. of fields necessary to be called unique.

Syntax:

CREATE TABLE Table_Name(column_name data_type(size) **PRIMARY KEY**,);

Example:

CREATE TABLE faculty (fcode **NUMBER(3) PRIMARY KEY**, fname **CHAR(10)**);

5. FOREIGN KEY: It is a table level constraint. We cannot add this at column level. To reference any primary key column from other table this constraint can be used. The table in which the foreign key is defined is called a **detail table**. The table that defines the primary key and is referenced by the foreign key is called the **master table**.

Syntax: **CREATE TABLE** Table_Name(column_name data_type(size)

FOREIGN KEY(column_name) REFERENCES table_name);

Example:

CREATE TABLE subject (scode **NUMBER (3) PRIMARY KEY**, subname
CHAR(10),fcode **NUMBER(3)**, **FOREIGN KEY(fcode) REFERENCE** faculty);

Defining integrity constraints in the alter table command:

Syntax: **ALTER TABLE** Table_Name **ADD PRIMARY KEY** (column_name);

Example: **ALTER TABLE** student **ADD PRIMARY KEY** (sno);

(Or)

Syntax: **ALTER TABLE** table_name **ADD CONSTRAINT** constraint_name
PRIMARY KEY(colname)

Example: **ALTER TABLE** student **ADD CONSTRAINT SN PRIMARY KEY(SNO)**

Dropping integrity constraints in the alter table command:

Syntax: **ALTER TABLE** Table_Name **DROP** constraint_name;

Example: **ALTER TABLE** student **DROP PRIMARY KEY**;

(or)

Syntax: **ALTER TABLE** student **DROP CONSTRAINT** constraint_name;

Example: **ALTER TABLE** student **DROP CONSTRAINT SN**;

6. DEFAULT : The DEFAULT constraint is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified. **Syntax:**

CREATE TABLE Table_Name(col_name1,col_name2,col_name3
DEFAULT '<value>');

Example:

CREATE TABLE student (sno **NUMBER(3) UNIQUE**, name
CHAR(10),address
VARCHAR(20) DEFAULT 'Aurangabad');

Conclusion: To be written by student.

LAB PRACTICE ASSIGNMENT:

1. Create a table EMPLOYEE with following schema:
(*Emp_no, E_name, E_address, E_ph_no, Dept_no, Dept_name, Job_id , Salary*)
2. Add a new column; HIREDATE to the existing relation.
3. Change the datatype of JOB_ID from char to varchar2.
4. Change the name of column/field Emp_no to E_no. Modify the column width of the job field of emp table



Experiment-3: Running of Simple SQL Queries

Aim: To run Simple SQL Queries

DATA MANIPULATION LANGUAGE (DML): The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database. Let's take a brief look at the basic DML commands:

1. INSERT

2. UPDATE

3. DELETE

1. INSERT INTO: This is used to add records into a relation. There are three types of INSERT INTO queries which are as **a) Inserting a single record**

Syntax: INSERT INTO < relation/table name> (field_1,field_2.....field_n)VALUES
(data_1,data_2, data_n);

Example: SQL>INSERT INTO student(sno,sname,class,address)VALUES
(1,'Ravi','M.Tech','Palakol');

b) Inserting a single record

Syntax: INSERT INTO < relation/table name>VALUES (data_1,data_2, data_n);

Example: SQL>INSERT INTO student VALUES (1,'Ravi','M.Tech','Palakol');

c) Inserting all records from another relation

Syntax: INSERT INTO relation_name_1 SELECT Field_1,field_2,field_n
FROM relation_name_2 WHERE field_x=data;

Example: SQL>INSERT INTO std SELECT sno,sname FROM student
WHERE name = 'Ramu';

d) Inserting multiple records

Syntax: INSERT INTO relation_name field_1,field_2, field_n) VALUES
(&data_1,&data_2, &data_n);

Example: SQL>INSERT INTO student (sno, sname, class,address)
VALUES (&sno,'&sname','&class','&address');

Enter value for sno: 101

Enter value for name: Ravi

Enter value for class: M.Tech

Enter value for name: Palakol

2. UPDATE-SET-WHERE: This is used to update the content of a record in a relation.

Syntax: SQL>UPDATE relation name SET Field_name1=data,field_name2=data,
WHERE field_name=data;

Example: SQL>UPDATE student SET sname = 'kumar' WHERE sno=1;

3. DELETE-FROM: This is used to delete all the records of a relation but it will retain the structure of that relation.

a) DELETE-FROM: This is used to delete all the records of relation.

Syntax: SQL>DELETE FROM relation_name;

Example: SQL>DELETE FROM std;

b) DELETE -FROM-WHERE: This is used to delete a selected record from a relation. **Syntax:** SQL>DELETE FROM relation_name WHERE condition;

Example: SQL>DELETE FROM student WHERE sno = 2;

2. TRUNCATE: This command will remove the data permanently. But structure will not be removed.

Difference between Truncate & Delete:-

- ✓ By using truncate command data will be removed permanently & will not get back where as by using delete command data will be removed temporally & get back by using roll back command.
- ✓ By using delete command data will be removed based on the condition where as by using truncate command there is no condition.
- ✓ Truncate is a DDL command & delete is a DML command.

Syntax: TRUNCATE TABLE <Table name>

Example TRUNCATE TABLE student;

□ **To Retrieve data from one or more tables.**

1. SELECT FROM: To display all fields for all records.

Syntax : SELECT * FROM relation_name;

Example : SQL> select * from dept;

DEPTNO	DNAME	LOC
-----	-----	-----
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

2. SELECT FROM: To display a set of fields for all records of relation.

Syntax: SELECT a set of fields FROM relation_name;

Example: SQL> select deptno, dname from dept;

DEPTNO	DNAME
-----	-----
10	ACCOUNTING
20	RESEARCH
30	SALES

3. SELECT - FROM -WHERE: This query is used to display a selected set of fields for a selected set of records of a relation.

Syntax: SELECT a set of fields FROM relation_name WHERE condition;

Example: SQL> select * FROM dept WHERE deptno<=20;

DEPTNO	DNAME	LOC
-----	-----	-----
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS

LAB PRACTICE ASSIGNMENT:

Create a table EMPLOYEE with following schema:

(Emp_no, E_name, E_address, E_ph_no, Dept_no, Dept_name, Job_id , Salary)

Write SQL queries for following question:

1. Insert atleast 5 rows in the table.
2. Display all the information of EMP table.
3. Display the record of each employee who works in department D10.
4. Update the city of Emp_no-12 with current city as Nagpur.
5. Display the details of Employee who works in department MECH.
6. Delete the email_id of employee James.

7. Display the complete record of employees working in SALES Department.

Conclusion: To be written by the student



Experiment 4: Implementation of Alter, Update, Delete and Drop

Aim: To implement Alter, Update, Delete and Drop commands

Theory:

1. ALTER:

(a) **ALTER TABLE ...ADD...:** This is used to add some extra fields into existing relation.

Syntax: ALTER TABLE relation_name ADD (new field_1 data_type(size), new field_2 data_type(size),...);

Example: SQL>ALTER TABLE std ADD (Address CHAR(10));

(b) **ALTER TABLE...MODIFY...:** This is used to change the width as well as data type of fields of existing relations.

Syntax: ALTER TABLE relation_name MODIFY (field_1 newdata_type(Size), field_2 newdata_type(Size), ... field_newdata_type(Size));

Example:SQL>ALTER TABLE student MODIFY(sname VARCHAR(10),class VARCHAR(5));

c) **ALTER TABLE..DROP** This is used to remove any field of existing relations.

Syntax: ALTER TABLE relation_name DROP COLUMN (field_name);

Example:SQL>ALTER TABLE student DROP column (sname);

d)**ALTER TABLE..RENAME...:** This is used to change the name of fields in existing relations.

Syntax: ALTER TABLE relation_name RENAME COLUMN (OLD field_name) to (NEW field_name);

Example: SQL>ALTER TABLE student RENAME COLUMN sname to stu_name;

3. **DROP TABLE:** This is used to delete the structure of a relation. It permanently deletes the records in the table.

Syntax: DROP TABLE relation_name; **Example:**

SQL>DROP TABLE std;

4. **RENAME:** It is used to modify the name of the existing database object.

Syntax: RENAME TABLE old_relation_name TO new_relation_name;

Example: SQL>RENAME TABLE std TO std1;

5. **UPDATE-SET-WHERE:** This is used to update the content of a record in a relation.

Syntax: SQL>UPDATE relation name SET Field_name1=data,field_name2=data, WHERE field_name=data;

Example: SQL>UPDATE student SET sname = 'kumar' WHERE sno=1;

4. **DELETE-FROM:** This is used to delete all the records of a relation but it will retain the structure of that relation.

c) **DELETE-FROM:** This is used to delete all the records of relation.

Syntax: SQL>DELETE FROM relation_name;

Example: SQL>DELETE FROM std;

d) **DELETE -FROM-WHERE:** This is used to delete a selected record from a relation.

Syntax: SQL>DELETE FROM relation_name WHERE condition;

Example: SQL>DELETE FROM student WHERE sno = 2;

Conclusion: To be written by students

LAB PRACTICE ASSIGNMENT:

Create a table EMPLOYEE with following schema:

(Emp_no, E_name, E_address, E_ph_no, Dept_no, Dept_name, Job_id , Salary)

Write SQL queries for following question:

1. Insert atleast 5 rows in the table.
2. Display all the information of EMP table.
3. Display the record of each employee who works in department D10.
4. Update the city of Emp_no-12 with current city as Nagpur.
5. Display the details of Employee who works in department MECH.
6. Delete the email_id of employee James.
7. Display the complete record of employees working in SALES Department.



Experiment 5: Implementation of Aggregate and Character Function

Aim: Implementation of Aggregate and Character Function

Theory:

Aggregative operators: In addition to simply retrieving data, we often want to perform some computation or summarization. SQL allows the use of arithmetic expressions. We now consider a powerful class of constructs for computing aggregate values such as MIN and SUM.

1. **Count:** COUNT following by a column name returns the count of tuple in that column. If DISTINCT keyword is used then it will return only the count of unique tuple in the column. Otherwise, it will return count of all the tuples (including duplicates) count (*) indicates all the tuples of the column.

Syntax: COUNT (Column name)

Example: SELECT COUNT (Sal) FROM emp;

2. **SUM:** SUM followed by a column name returns the sum of all the values in that column.

Syntax: SUM (Column name)

Example: SELECT SUM (Sal) From emp;

3. **AVG:** AVG followed by a column name returns the average value of that column values.

Syntax: AVG (n1, n2...)

Example: Select AVG (10, 15, 30) FROM DUAL;

4. **MAX:** MAX followed by a column name returns the maximum value of that column.

Syntax: MAX (Column name)

Example: SELECT MAX (Sal) FROM emp;

SQL> select deptno, max(sal) from emp group by deptno;

DEPTNO	MAX (SAL)
10	5000
20	3000

30 2850

SQL> select deptno, max (sal) from emp group by deptno having max(sal)<3000;

DEPTNO MAX(SAL)

— —
30 2850

5. MIN: MIN followed by column name returns the minimum value of that column.

Syntax: MIN (Column name)

Example: SELECT MIN (Sal) FROM emp;

SQL>select deptno,min(sal) from emp group by deptno having min(sal)>1000;

DEPTNO MIN (SAL)

— —
10 1300

CHARACTER FUNCTION:

initcap(char) : select initcap("hello") from dual; lower

(char): select lower ('HELLO') from dual; upper

(char) :select upper ('hello') from dual; ltrim

(char,[set]): select ltrim ('cseit', 'cse') from dual; rtrim

(char,[set]): select rtrim ('cseit', 'it') from dual;

replace (char,search): select replace('jack and jue','j','bl') from dual;

Conclusion: To be written by students

LAB PRACTICE ASSIGNMENT:

Create a table EMPLOYEE with following schema:

(Emp_no, E_name, E_address, E_ph_no, Dept_no, Dept_name, Job_id, Designation , Salary)

Write SQL statements for the following query.

Computer Engineering Department, S. B. M. Polytechnic, Mumbai

1. List the E_no, E_name, Salary of all employees working for MANAGER.
2. Display all the details of the employee whose salary is more than the Sal of any IT PROFF..
3. List the employees in the ascending order of Designations of those joined after 1981.
4. List the employees along with their Experience and Daily Salary.
5. List the employees who are either 'CLERK' or 'ANALYST' .
6. List the employees who joined on 1-MAY-81, 3-DEC-81, 17-DEC-81,19-JAN-80 .
7. List the employees who are working for the Deptno 10 or20.
8. List the Enames those are starting with 'S' .
9. Display the name as well as the first five characters of name(s) starting with 'H'
10. List all the emps except 'PRESIDENT' & 'MGR' in asc order of Salaries.



Experiment 6: Implementation of various clauses in SQL

Aim: Implementation of various clauses in SQL

Objective:

To learn the concept of group functions

Theory:

- **GROUP BY:** This query is used to group to all the records in a relation together for each and every value of a specific key(s) and then display them for a selected set of fields the relation. □

Syntax: SELECT <set of fields> FROM <relation_name>
GROUP BY <field_name>;

Example: SQL> SELECT EMPNO, SUM (SALARY) FROM EMP GROUP BY EMPNO;

GROUP BY-HAVING : The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions. The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used.

Syntax: SELECT column_name, aggregate_function(column_name) FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value;

Example: SELECT Employees.LastName, COUNT(Orders.OrderID) AS
NumberOfOrders
FROM (Orders
INNER JOIN Employees
ON Orders.EmployeeID=Employees.EmployeeID) GROUP BY LastName
HAVING COUNT (Orders.OrderID) > 10;

JOIN using GROUP BY: This query is used to display a set of fields from two relations by matching a common field in them and also group the corresponding records for each and every value of a specified key(s) while displaying.

Syntax: SELECT <set of fields (from both relations)> FROM
relation_1,relation_2 WHERE relation_1.field_x=relation_2.field_y
GROUP BY field_z; **Example:**

SQL> SELECT empno,SUM(SALARY) FROM emp,dept
WHERE emp.deptno =20 GROUP BY empno;

- **ORDER BY:** This query is used to display a selected set of fields from a relation in an ordered manner base on some field. □

Syntax: SELECT <set of fields> FROM <relation_name>
ORDER BY <field_name>;

Example: SQL> SELECT empno, ename, job FROM emp ORDER BY job;

JOIN using ORDER BY: This query is used to display a set of fields from two relations by matching a common field in them in an ordered manner based on some fields. **Syntax:** SELECT <set of fields (from both relations)> FROM relation_1,
relation_2

WHERE relation_1.field_x = relation_2.field_y ORDER BY field_z;

Example: SQL> SELECT empno,ename,job,dname FROM emp,dept
WHERE emp.deptno = 20 ORDER BY job;

- **INDEXING:** An *index* is an ordered set of pointers to the data in a table. It is based on the data values in one or more columns of the table. SQL Base stores indexes separately □ from tables.

An index provides two benefits:

- It improves performance because it makes data access faster.
- It ensures uniqueness. A table with a unique index cannot have two rows with the same values in the column or columns that form the index key.

Syntax:
CREATE INDEX <index_name> on <table_name> (attrib1,attrib 2....attrib
n);

Example:
CREATE INDEX id1 on emp(empno,dept_no);

LAB PRACTICE ASSIGNMENT:

Create a relation and implement the following queries.

1. Display total salary spent for each job category.
2. Display lowest paid employee details under each manager.
3. Display number of employees working in each department and their department name.
4. Display the details of employees sorting the salary in increasing order.
5. Show the record of employee earning salary greater than 16000 in each department.
6. Write queries to implement and practice the above clause.



Experiment 7: Execution of string, comparison and set operations.

Aim:

Execution of string, comparison and set operations.

Theory:

STRING FUNCTIONS:

Concat: CONCAT returns char1 concatenated with char2. Both char1 and char2 can be any of the datatypes

```
SQL>SELECT CONCAT('ORACLE','CORPORATION')FROM DUAL;  
ORACLECORPORATION
```

Lpad: LPAD returns expr1, left-padded to length n characters with the sequence of characters in expr2.

```
SQL>SELECT LPAD('ORACLE',15,'*')FROM DUAL;  
*****ORACLE
```

Rpad: RPAD returns expr1, right-padded to length n characters with expr2, replicated as many times as necessary.

```
SQL>SELECT RPAD ('ORACLE',15,'*')FROM DUAL;  
ORACLE*****
```

Ltrim: Returns a character expression after removing leading blanks.

```
SQL>SELECT LTRIM('SSMITHSS','S')FROM DUAL;  
MITHSS
```

Rtrim: Returns a character string after truncating all trailing blanks

```
SQL>SELECT RTRIM('SSMITHSS','S')FROM DUAL;  
SSMITH
```

Lower: Returns a character expression after converting uppercase character data to lowercase.

```
SQL>SELECT LOWER('DBMS')FROM DUAL;  
dbms
```

Upper: Returns a character expression with lowercase character data converted to uppercase

```
SQL>SELECT UPPER('dbms')FROM DUAL;  
DBMS
```

Length: Returns the number of characters, rather than the number of bytes, of the given string expression, excluding trailing blanks.

```
SQL>SELECT LENGTH('DATABASE')FROM DUAL;  
8
```

Substr: Returns part of a character, binary, text, or image expression.

```
SQL>SELECT SUBSTR('ABCDEFGHIJ'3,4)FROM DUAL;  
CDEF
```

Instr: The INSTR functions search string for substring. The function returns an integer indicating the position of the character in string that is the first character of this occurrence.

```
SQL>SELECT INSTR('CORPORATE FLOOR','OR',3,2)FROM DUAL;  
14
```

COMPARISION OPERATORS:

(=): Checks if the values of two operands are equal or not, if yes then condition becomes true.

(!=): Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.

(< >): Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.

(>): Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true

(<): Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.

(>=): Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.

(<=): Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

SET OPERATORS:

The Set operator combines the result of 2 queries into a single result. The following are the operators:

- Union
- Union all
- Intersect
- Minus

Union: Returns all distinct rows selected by both the queries

Union all: Returns all rows selected by either query including the duplicates.

Intersect: Returns rows selected that are common to both queries. **Minus:** Returns all distinct rows selected by the first query and are not by the second

Conclusion: To be written by the students.



Experiment 8: Inner Join, Outer Join & Natural Join

Aim:

To implement different types of joins

Theory :

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each. The join is actually performed by the 'where' clause which combines specified rows of tables.

Syntax:

```
SELECT column 1, column 2, column 3...  
FROM table_name1, table_name2  
WHERE table_name1.column name = table_name2.columnname;
```

Types of Joins :

1. Simple Join
2. Self Join
3. Outer Join

Simple Join:

It is the most common type of join. It retrieves the rows from 2 tables having a common column and is further classified into

Equi-join :

A join, which is based on equalities, is called equi-join.

Example:

```
Select * from item, cust where item.id=cust.id;
```

In the above statement, item-id = cust-id performs the join statement. It retrieves rows from both the tables provided they both have the same id as specified by the where clause. Since the where clause uses the comparison operator (=) to perform a join, it is said to be equijoin. It combines the matched rows of tables. It can be used as follows:

- ✓ To insert records in the target table.
- ✓ To create tables and insert records in this table.
- ✓ To update records in the target table.
- ✓ To create views.

Non Equi-join:

It specifies the relationship between columns belonging to different tables by making use of relational operators other than '='.

Example:

Select * from item, cust where item.id<cust.id;

Table Aliases

Table aliases are used to make multiple table queries shorted and more readable. We give an alias name to the table in the 'from' clause and use it instead of the name throughout the query.

Self join:

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

Example:

```
select * from emp x ,emp y where x.salary >= (select avg(salary) from x.emp
where x. deptno =y.deptno);
```

Outer Join:

It extends the result of a simple join. An outer join returns all the rows returned by simple join as well as those rows from one table that do not match any row from the table. The symbol(+) represents outer join.

–Left outer join

–Right outer join

–Full outer join

LAB PRACTICE ASSIGNMENT:

Consider the following schema:

Sailors (sid, sname, rating, age)

Boats (bid, bname, color)

Reserves (sid, bid,

day(date))

1. Find all information of sailors who have reserved boat number 101.
2. Find the name of boat reserved by Bob.
3. Find the names of sailors who have reserved a red boat, and list in the order of age.
4. Find the names of sailors who have reserved at least one boat.
5. Find the ids and names of sailors who have reserved two different boats on the same day.
6. Find the ids of sailors who have reserved a red boat or a green boat.
7. Find the name and the age of the youngest sailor.
8. Count the number of different sailor names.
9. Find the average age of sailors for each rating level.

Conclusion: To be written by the students.



Experiment 9: Implementation of views

AIM: Implementation of views

Theory:

VIEW: In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

A view is a virtual table, which consists of a set of columns from one or more tables. It is similar to a table but it does not store in the database. View is a query stored as an object.

Syntax: CREATE VIEW <view_name> AS SELECT <set
of fields> FROM relation_name WHERE (Condition)

Example:

```
SQL> CREATE VIEW employee AS SELECT empno,ename,job FROM EMP
      WHERE job =
      'clerk'; SQL> View created.
```

Example:

```
CREATE VIEW [Current Product List] AS
SELECT ProductID, ProductName
FROM Products
WHERE Discontinued=No;
```

UPDATING A VIEW : A view can updated by using the following syntax :

Syntax : CREATE OR REPLACE VIEW view_name AS
 SELECT column_name(s)
 FROM table_name
 WHERE condition

DROPPING A VIEW: A view can deleted with the DROP VIEW command.

Syntax: DROP VIEW <view_name> ;

LAB PRACTICE ASSIGNMENT:

Consider the following schema:

Sailors (sid, sname, rating, age)

Boats (bid, bname, color)

Reserves (sid, bid, day(date))

Write subquery statement for the following queries.

1. Find all information of sailors who have reserved boat number 101.
2. Find the name of boat reserved by Bob.
3. Find the names of sailors who have reserved a red boat, and list in the order of age.
4. Find the names of sailors who have reserved at least one boat.
5. Find the ids and names of sailors who have reserved two different boats on the same day.
6. Find the ids of sailors who have reserved a red boat or a green boat.
7. Find the name and the age of the youngest sailor.
8. Count the number of different sailor names.
9. Find the average age of sailors for each rating level.
10. Find the average age of sailors for each rating level that has at least two sailors.

Conclusion: To be written by the students.



Experiment 10: Subqueries in SQL

Aim: Implement subqueries in SQL

Theory:

SUBQUERIES: The query within another is known as a sub query. A statement containing sub query is called parent statement. The rows returned by sub query are used by the parent statement or in other words A subquery is a SELECT statement that is embedded in a clause of another SELECT statement You can place the subquery in a number of SQL clauses:

- WHERE clause
- HAVING clause
- FROM clause
- OPERATORS(IN,ANY,ALL,<,>,>=,<= etc..)

Types

1. Sub queries that return several values

Sub queries can also return more than one value. Such results should be made use along with the operators in and any.

2. Multiple queries

Here more than one sub query is used. These multiple sub queries are combined by means of 'and' & 'or' keywords.

3. Correlated sub query

A sub query is evaluated once for the entire parent statement whereas a correlated Sub query is evaluated once per row processed by the parent statement.

Conclusion: To be written by the students.



Experiment 11: Implement Normalization on the table in a database

Aim: Implement Normalization on the table in a database.

Theory:

Database normalization is a technique for designing relational database tables to minimize duplication of information and, in so doing, to safeguard the database against certain types of logical or structural problems, namely data anomalies.

For example, when multiple instances of a given piece of information occur in a table, the possibility exists that these instances will not be kept consistent when the data within the table is updated, leading to a loss of data integrity.

A table that is sufficiently normalized is less vulnerable to problems of this kind, because its structure reflects the basic assumptions for when multiple instances of the same information should be

Normalization is a process of converting a relation to be standard form by decomposition a larger relation into smaller efficient relation that depicts a good database design.

- 1NF: A Relation scheme is said to be in 1NF if the attribute values in the relation are atomic.i.e., Multi-valued attributes are not permitted.
- 2NF: A Relation scheme is said to be in 2NF,iff and every Non-key attribute is fully functionally dependent on primary Key.
- 3NF: A Relation scheme is said to be in 3NF,iff and does not have transitivity dependencies. A Relation is said to be 3NF if every determinant is a key for each & every functional dependency.
- BCNF: A Relation scheme is said to be BCNF if the following statements are true for eg FD $P \rightarrow Q$ in set F of FDs that holds for each FD. $P \rightarrow Q$ in set F of FD's that holds over R. Here P is the subset of attributes of R & Q is a single attribute of R represented by a single instance only.

Normalized tables are:- `mysql> create table Bus(BusNo varchar(20) primary key,Source varchar(20),Destination varchar(20));`

`mysql> Create table passenger(PPN varchar(15) Primary key,Name varchar(20),Age integer,Sex char,Address varchar(20));`

`mysql> Create table PassengerTicket(PPN varchar(15) Primary key,TicketNo integer);`

Conclusion: To be written by the students.

1. Explain the need of normalization?
2. What is functional dependency?
3. Explain difference between third normal form and boyce codd normal form?
4. What is PJNF?
5. What is transitivity dependency?