GOVT. POLYTECHNIC, KANDHAMAL

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

LABORATORY MANUAL FOR DATABASE MANAGEMENT SYSTEM LAB

BRANCH – CSE

NAME OF FACULTY – SMT. RAJSHREE SENAPTI

Semester – 4th

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 ASNI 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. DATA DEFINITION LANGUAGE (DDL):

The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project.

Let's take a look at the structure and usage of four basic DDL commands:

1. CREATE 2. ALTER 3. DROP 4. RENAME

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), .. .);

Example:

SQL> CREATE TABLE Student (sno NUMBER (3), sname CHAR (10), class CHAR (5));

2. 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;

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. These are three type 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;

5. 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

NUMBER FUNCTION:
Abs(n) :Select abs(-15) from dual;
Exp(n): Select exp(4) from dual;
Power(m,n): Select power(4,2) from dual;
Mod(m,n): Select mod(10,3) from dual;
Round(m,n): Select round(100.256,2) from dual;
Trunc(m,n): ;Select trunc(100.256,2) from dual;
Sqrt(m,n);Select sqrt(16) from dual;

Develop aggregate plan strategies to assist with summarization of several data entries.

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;

CONVERSION FUNCTIONS:

To_char: TO_CHAR (number) converts n to a value of VARCHAR2 data type, using the optional number format fmt. The value n can be of type NUMBER, BINARY_FLOAT, or BINARY_DOUBLE.

SQL>select to_char(65,'RN')from dual;
LXV

To_number : TO_NUMBER converts expr to a value of NUMBER data type.
SQL>Select to_number ('1234.64') from Dual;
1234.64

To_date:TO_DATE converts char of CHAR, VARCHAR2, NCHAR, or NVARCHAR2 data type to a value of DATE data type.

SQL>SELECT TO_DATE('January 15, 1989, 11:00 A.M.')FROM DUAL;
TO_DATE
15-JAN-89

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

DATE FUNCTIONS:

Sysdate:

SQL>SELECT SYSDATE FROM DUAL;

29-DEC-08

next_day:

SQL>SELECT NEXT_DAY(SYSDATE,'WED')FROM DUAL;

05-JAN-09

add_months:

SQL>SELECT ADD_MONTHS(SYSDATE,2)FROM DUAL;

28-FEB-09

last_day:

SQL>SELECT LAST_DAY(SYSDATE)FROM DUAL;

31-DEC-08

months_between:

SQL>SELECT MONTHS_BETWEEN(SYSDATE,HIREDATE)FROM EMP;

4

Least:

SQL>SELECT LEAST('10-JAN-07','12-OCT-07')FROM DUAL;

10-JAN-07

Greatest:

SQL>SELECT GREATEST('10-JAN-07','12-OCT-07')FROM DUAL;

10-JAN-07

Trunc:

SQL>SELECT TRUNC(SYSDATE,'DAY')FROM DUAL;

28-DEC-08

Round:

SQL>SELECT ROUND(SYSDATE,'DAY')FROM DUAL;

28-DEC-08

to_char:

SQL> select to_char(sysdate, "dd\mm\yy") from dual;

24-mar-05.

to_date:

SQL> select to date (sysdate, "dd\mm\yy") from dual;

24-mar-o5.

IMPLEMENTATION OF DIFFERENT TYPES OF OPERATORS IN SQL.

- Arithmetic Operator
- Logical Operator
- Comparision Operator
- Special Operator
- Set Operator

ARIHMETIC OPERATORS:

(+) : Addition - Adds values on either side of the operator .
(-):Subtraction - Subtracts right hand operand from left hand operand .
(*):Multiplication - Multiplies values on either side of the operator .
(/):Division - Divides left hand operand by right hand operand .
(^):Power- raise to power of .
(%):Modulus - Divides left hand operand by right hand operand and returns remainder.

LOGICAL OPERATORS:

AND : The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

OR: The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

NOT: The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**

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.

SPECIAL OPERATOR:

BETWEEN: The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.

IS NULL: The NULL operator is used to compare a value with a NULL attribute value.

ALL: The ALL operator is used to compare a value to all values in another value set

ANY: The ANY operator is used to compare a value to any applicable value in the list according to the condition.

LIKE: The LIKE operator is used to compare a value to similar values using wildcard operators. It allows to use percent sign(%) and underscore (_) to match a given string pattern.
IN: The IN operator is used to compare a value to a list of literal values that have been specified.

EXIST: The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.

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

IMPLEMENTATION OF DIFFERENT TYPES OF JOINS

- Inner Join
- Outer Join
- Natural Join..etc

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

STUDY & IMPLEMENTATION OF

- Group by & Having Clause
- Order by Clause
- Indexing

 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);

STUDY & IMPLEMENTATION OF DIFFERENT TYPES OF CONSTRAINTS

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');

Experiment 1:

i) Show the Structure of DEPT.

SQL>DESC DEPT;

ii) Select all data from DEPT table.

SQL> SELECT * FROM DEPT;

OUTPUT:

DeptNo	DeptName	Location
101	HR	New York
102	Finance	Chicago
103	Marketing	Los Angeles

iii) Create a query to display unique jobs from the EMP table.

SQL> SELECT DISTINCT JOB FROM EMP;

OUTPUT:

JOB
Manager
Clerk
Analyst

Experiment 2:

Write a query to Name the column headings EMP#, Employee, Job and Hire date, respectively. Run the query.

SQL> SELECT EMPNO AS "EMP#", ENAME AS "Employee", JOB, HIREDATE FROM EMP;

OUTPUT:

EMP#	Employee	Job	Hire Date
1	John	Manager	2022-01-01
2	Jane	Clerk	2022-02-15
3	Bob	Analyst	2022-03-10
4	Alice	Manager	2022-04-05
5	Carol	Clerk	2022-05-20
6	David	Analyst	2022-06-15

Experiment 3:

Create a query to display the Name and salary of employees earning more than Rs.2850.Save the query and run it.

SQL> SELECT Name, Salary
FROM EMP
WHERE Salary > 2850;

OUTPUT:

Name	Salary
John	2500
Mary	3000
Bob	2800
Alice	2900
Charlie	3200

Experiment 4:

Create a query to display the employee name and department no. for employee no. 7566.

SQL> SELECT Name, DepartmentNo
FROM EmployeeTable
WHERE EmployeeNo = 7566;

OUTPUT:

Name	DepartmentNo
Smith	20

Experiment 5:

Display the employee name, job and start date of employees hire date between Feb.20.1981 and May 1, 1981. Order the query in ascending order of start date.

SQL>SELECT Name, Job, StartDate
FROM EmployeeTable
WHERE HireDate BETWEEN '1981-02-20' AND '1981-05-01'
ORDER BY StartDate ASC;

OUTPUT:

Name	Job	StartDate
John	Engineer	1981-02-25
Mary	Analyst	1981-03-15
Bob	Manager	1981-04-01
Alice	Clerk	1981-04-20
Charlie	Technician	1981-05-01

Experiment 6:

Display the name and title of all employees who don't have a Manager.

SQL> SELECT Name, Title
FROM EmployeeTable
WHERE ManagerID IS NULL;

OUTPUT:

Name	Title
John	Manager

Experiment 7:

Display the name, salary and comm. For all employee who earn comm. Sort data in descending order of salary and comm.

SQL> SELECT Name, Salary, Commission
FROM EmployeeTable
WHERE Commission IS NOT NULL
ORDER BY Salary DESC, Commission DESC;

OUTPUT:

Name	Salary	Commission
Charlie	7000	1500
Bob	6000	1200
John	5000	1000
Alice	5500	1000
Mary	4500	800

Experiment 8:

Display the name job, salary for all employees whose job is Clerk or Analyst their salary is not equal to Rs.1000, Rs.3000, Rs.5000.

SQL> SELECT Name, Job, Salary
FROM EmployeeTable
WHERE (Job = 'Clerk' OR Job = 'Analyst') AND Salary NOT IN (1000, 3000, 5000);

OUTPUT>

Name	Job	Salary
Mary	Analyst	2500
Charlie	Analyst	800

Experiment 9:

Write a query to display the date. Label the column DATE.

SQL> SELECT CURRENT_DATE AS DATE;

OUTPUT:

| DATE |
| 2024-01-11 |

Experiment 10:

Create a unique listing of all jobs that are in department 30.

SQL> SELECT DISTINCT Job
FROM EmployeeTable
WHERE DepartmentNo = 30;

OUTPUT:

| Job |
| Clerk |
| Engineer |

Experiment 11:

Write a query to display the name, department number and department name for all employees.

SQL> SELECT e.Name, e.DepartmentNo, d.DepartmentName
FROM EmployeeTable e
JOIN DepartmentTable d ON e.DepartmentNo = d.DepartmentNo;

OUTPUT:

Name	DepartmentNo	DepartmentName
John	10	HR
Mary	20	Finance
Bob	30	IT
Alice	20	Finance
Charlie	30	IT

Experiment 12:

Write a query to display the employee name, department name, and location of all employee who earn a commission.

SQL> SELECT e.Name AS EmployeeName, d.DepartmentName, l.Location
FROM EmployeeTable e
JOIN DepartmentTable d ON e.DepartmentNo = d.DepartmentNo
JOIN LocationTable l ON d.LocationID = l.LocationID
WHERE e.Commission IS NOT NULL;

OUTPUT:

EmployeeName	DepartmentName	Location
John	HR	New York
Mary	Finance	San Francisco
Bob	IT	New York
Charlie	IT	New York

Experiment 13:

Write a query to display the name, job, department number and department name for all employees who works in DALLAS.

SQL> SELECT e.Name, e.Job, e.DepartmentNo, d.DepartmentName
FROM EmployeeTable e
JOIN DepartmentTable d ON e.DepartmentNo = d.DepartmentNo
JOIN LocationTable l ON d.LocationID = l.LocationID
WHERE l.Location = 'DALLAS';

OUTPUT:

Name	Job	DepartmentNo	DepartmentName
Bob	Clerk	30	IT
Charlie	Technician	30	IT

Experiment 14:

Write a query to display the number of people with the same job. Save the query @ run it.

SQL>SELECT Job, COUNT(*) AS NumberOfPeople
FROM EmployeeTable
GROUP BY Job;

OUTPUT:

Job	NumberOfPeople
Manager	1
Analyst	2
Clerk	2
Engineer	1
Technician	1

Experiment 15:

Create a query to display the employee name and hire date for all employees in same department.

SQL> SELECT employee_name, hire_date
FROM employees
WHERE department_id = (SELECT department_id FROM employees WHERE employee_id = :employee_id);

OUTPUT:

employee_name	hire_date
John Doe	2020-05-15
Jane Smith	2019-10-22
Mark Johnson	2021-03-08
Sarah Brown	2022-01-05

Experiment 16:

Display the employee name and salary of all employees who report to KING.

SQL> SELECT employee_name, salary
FROM employees
WHERE manager_id = (SELECT employee_id FROM employees WHERE employee_name = 'KING');

OUTPUT:

employee_name	salary
SMITH	800
JONES	2975
MARTIN	1250

Experiment 17:

. Display the mane, department name and salary of any employee whose salary and commission matches both the salary and commission of any employee located in DALLAS.

SQL> SELECT e.employee_name, d.department_name, e.salary
FROM employees e
JOIN departments d ON e.department_id = d.department_id
WHERE e.salary IN (SELECT salary FROM employees WHERE commission IS NOT NULL AND location = 'DALLAS')
  AND e.commission IN (SELECT commission FROM employees WHERE commission IS NOT NULL AND location = 'DALLAS');

OUTPUT:

employee_name	DepartmentName	salary
John Doe	HR	3000
Mark Johnson	HR	3000

Experiment 18:

Create a student database table using create command using Regd. No as Primary Key , insert data of your class.

SQL> CREATE TABLE students (
   regd_no INT PRIMARY KEY,
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   date_of_birth DATE,
   department VARCHAR(50),
   grade FLOAT);

SQL>INSERT INTO students (regd_no, first_name, last_name, date_of_birth, department, grade)
VALUES (1, 'John', 'Doe', '2000-05-15', 'Computer Science', 85.5),
   (2, 'Jane', 'Smith', '2001-02-20', 'Mathematics', 78.3),
   (3, 'Mark', 'Johnson', '1999-09-10', 'Physics', 92.0),
   (4, 'Sarah', 'Brown', '2000-12-05', 'Chemistry', 88.7);

OUTPUT:

regd_no	first_name	last_name	date_of_birth	department	grade
1	John	Doe	2000-05-15	Computer Science	85.5
2	Jane	Smith	2001-02-20	Mathematics	78.3
3	Mark	Johnson	1999-09-10	Physics	92.0
4	Sarah	Brown	2000-12-05	Chemistry	88.7

Experiment 19:

Delete the information of student having roll No -15 and City- Bhubaneswar. Rename the Student database table to STUDENT INFORMATION.

SQL> DELETE FROM students WHERE regd_no = 15 AND city = 'Bhubaneswar';

SQL>EXEC sp_rename 'students', 'STUDENT_INFORMATION';

OUTPUT:

Before deletion

regd_no	first_name	last_name	date_of_birth	department	grade
1	John	Doe	2000-05-15	Computer Science	85.5
2	Jane	Smith	2001-02-20	Mathematics	78.3
15	Alice	Johnson	2000-09-10	Physics	92.0
4	Sarah	Brown	2000-12-05	Chemistry	88.7

After Deletion

regd_no	first_name	last_name	date_of_birth	department	grade
1	John	Doe	2000-05-15	Computer Science	85.5
2	Jane	Smith	2001-02-20	Mathematics	78.3
4	Sarah	Brown	2000-12-05	Chemistry	88.7