Practical Record

DBMS & PL/SQL

For Diploma in Computer Applications

INDEX:

9	STUDENT TABLE	
10	BOOK TABLE	
11	EMPLOYEE TABLE	
12	BANK TABLE	
13	STOCK TABLE	
14	Implementation of PL/SQL commands Write a PL/SQL program to add two numbers?	

Experiment No: 1

Title: Implementation of DDL commands of SQL with suitable examples

- Create table
- Alter table
- Drop Table

Objective:

- ✓ To understand the different issues involved in the design and implementation of a database system
- ✓ To understand and use data definition language to write query for a database

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.

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).

4. **DATE:** This data type is used to represent date and time.

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.

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. Four basic DDL commands are:

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;

***

Experiment No:2

Title : Implementation of DML commands of SQL with suitable examples

- Insert table
- Update table
- Delete Table

Objective :

- ✓ To understand the different issues involved in the design and implementation of a database system
- ✓ To understand and use data manipulation language to query, update, and manage a database

Theory :

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.

<u>Difference between Truncate & Delete:-</u>

- ✓ 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
```

Experiment No: 3

Title: Implementation of different types of functions with suitable examples.

- Number Function
- Aggregate Function
- Character Function
- Conversion Function
- Date Function

Objective:

- *To understand and implement various types of function in SQL.*

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.

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;

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
```

*****

Experiment No: 4

Title : Implementation of different types of operators in SQL.

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

Objective:

 To learn different types of operator.

Theory:

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:

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

<u>IS NULL:</u> The NULL operator is used to compare a value with a NULL attribute value.

<u>ALL:</u> The ALL operator is used to compare a value to all values in another value set

<u>ANY:</u> 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.

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

*****

Experiment No: 5

Title : Implementation of different types of Joins

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

Objective :

 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.

*****

Experiment No: 6

Title : Study & Implementation of

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

***Objective*:**

To learn the concept of group functions

***Theory*:**

<u>GROUP BY:</u> 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);

*****

Experiment No: 7

Title : Study & Implementation of

- Sub queries
- Views

***Objective*:**

 To perform nested Queries and joining Queries using DML command

 To understand the implementation of views.

***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.

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> ;

******

Experiment No: 8

Title : • Study & Implementation of different types of constraints

***Objective*:**

 To practice and implement constraints

***Theory*:**

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'));

<u>4.</u> **<u>PRIMARY KEY:</u>** 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**));

<u>5.</u> **<u>FOREIGN KEY:</u>** 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 No: 9

Title: **Create a table Student with the following fields and insert at least 5 records into the table except for the column Total.**

1. **Create a table Student with the following fields and insert at least 5 records into the table except for the column Total.**

ROLL_NUMBER	INTEGER	PRIMARY KEY
NAME	VARCHAR (25)	
BATCH	VARCHAR (15)	
MARK1	INTEGER	
MARK2	INTEGER	
MARK3	INTEGER	
TOTAL	INTEGER	

(1) Update the column Total with the sum of Mark1, Mark2 and Mark3.
(2) List the details of students in DCA batch.
(3) Display the name and total marks of students who are failed (Total < 90).
(4) Display the name and batch of those students who scored 90 or more in Mark1 and Mark2.
(5) Delete the student who scored below 30 in Mark3.

SQL statements :

1. SQL Query to create the table .

```
CREATE TABLE student
(
Roll_Number INT PRIMARY KEY,
Name VARCHAR (25),
Batch VARCHAR(15),
Mark1 INT,
Mark2 INT,
Mark3 INT,
Total INT
);
```

2. SQL Query to insert 5 records into the table
   I. INSERT INTO student (Roll_Number, Name, Batch, Mark1, Mark2, Mark3) VALUES (1, ’Akhil’, ’science’, 20, 30, 25);
   II. INSERT INTO student (Roll_Number, Name, Batch, Mark1, Mark2, Mark3) VALUES (2, ’Sreejith’, ’humanities’, 140, 110, 20);
   III. INSERT INTO student (Roll_Number, Name, Batch, Mark1, Mark2, Mark3) VALUES (3, ’Chithra’, ’commerce’, 45, 35, 28);

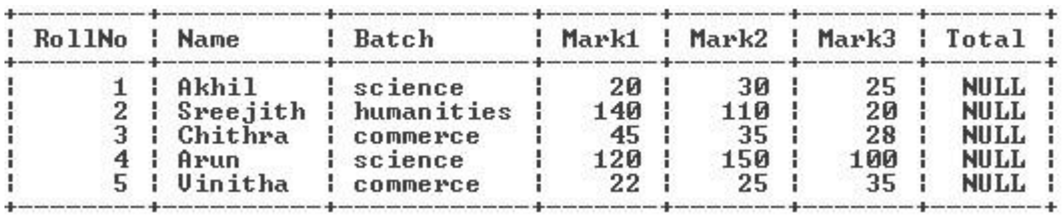
IV. INSERT INTO student (Roll_Number, Name, Batch, Mark1, Mark2, Mark3) VALUES (4, ’Arun’, ’science’, 120, 150, 100);

V. INSERT INTO student (Roll_Number, Name, Batch, Mark1, Mark2, Mark3) VALUES (5, ’Vinitha’, ’commerce’, 22, 25, 35);

3. SQL Query for each question

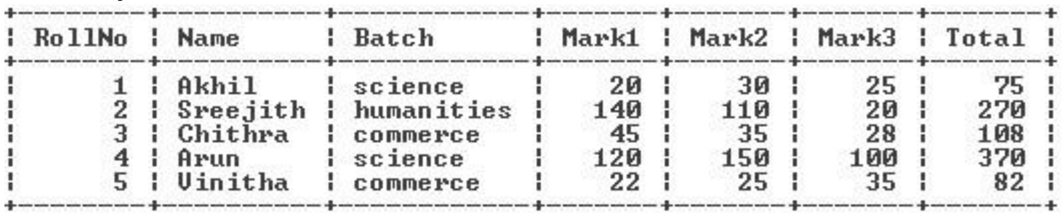
a. UPDATE student
   SET Total=Mark1+Mark2+Mark3;
b. SELECT *
   FROM student
   WHERE Batch=’commerce’;
c. SELECT Name,Total
   FROM student
   WHERE Total<90;
d. SELECT Name,Batch
   FROM student
   WHERE Mark1>90 AND Mark2>90;
e. DELETE
   FROM STUDENT
   WHERE Mark3<30;

Table with sample records

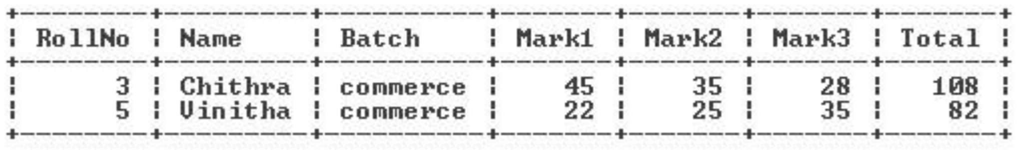

```
+--------+----------+------------+-------+-------+-------+-------+
| RollNo | Name     | Batch      | Mark1 | Mark2 | Mark3 | Total |
+--------+----------+------------+-------+-------+-------+-------+
|      1 | Akhil    | science    |    20 |    30 |    25 |  NULL |
|      2 | Sreejith | humanities |   140 |   110 |    20 |  NULL |
|      3 | Chithra  | commerce   |    45 |    35 |    28 |  NULL |
|      4 | Arun     | science    |   120 |   150 |   100 |  NULL |
|      5 | Vinitha  | commerce   |    22 |    25 |    35 |  NULL |
+--------+----------+------------+-------+-------+-------+-------+
```

Output of Queries

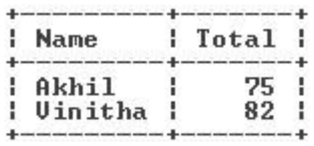
a. Query OK, 5 rows affected.


```
+--------+----------+------------+-------+-------+-------+-------+
| RollNo | Name     | Batch      | Mark1 | Mark2 | Mark3 | Total |
+--------+----------+------------+-------+-------+-------+-------+
|      1 | Akhil    | science    |    20 |    30 |    25 |    75 |
|      2 | Sreejith | humanities |   140 |   110 |    20 |   270 |
|      3 | Chithra  | commerce   |    45 |    35 |    28 |   108 |
|      4 | Arun     | science    |   120 |   150 |   100 |   370 |
|      5 | Vinitha  | commerce   |    22 |    25 |    35 |    82 |
+--------+----------+------------+-------+-------+-------+-------+
```

b. 2 rows in set.

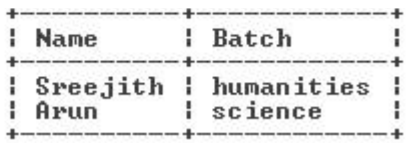

```
+--------+---------+----------+-------+-------+-------+-------+
| RollNo | Name    | Batch    | Mark1 | Mark2 | Mark3 | Total |
+--------+---------+----------+-------+-------+-------+-------+
|      3 | Chithra | commerce |    45 |    35 |    28 |   108 |
|      5 | Vinitha | commerce |    22 |    25 |    35 |    82 |
+--------+---------+----------+-------+-------+-------+-------+
```

c.2 rows in set.



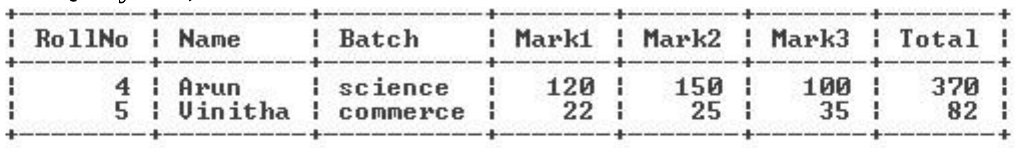
```
+----------+--------+
| Name     | Total  |
+----------+--------+
| Akhil    |     75 |
| Vinitha  |     82 |
+----------+--------+
```

d. 2 rows in set.



```
+-----------+-------------+
| Name      | Batch       |
+-----------+-------------+
| Sreejith  | humanities  |
| Arun      | science     |
+-----------+-------------+
```

e. Query OK, 3 rows affected.



```
+--------+----------+------------+--------+--------+--------+--------+
| RollNo | Name     | Batch      | Mark1  | Mark2  | Mark3  | Total  |
+--------+----------+------------+--------+--------+--------+--------+
|      4 | Arun     | science    |    120 |    150 |    100 |    370 |
|      5 | Vinitha  | commerce   |     22 |     25 |     35 |     82 |
+--------+----------+------------+--------+--------+--------+--------+
```

*****

Experiment No: 10

Title: **Create a table Book with the following fields and insert at least 5 records into the table.**

Book_ID Integer Primary key
Book_Name Varchar (20)
Author_Name Varchar (25)
Pub_Name Varchar (25)
Price Decimal (10,2)

a. Create a view containing the details of books published by SCERT.

b. Display the average price of books published by each publisher.

c. Display the details of book with the highest price.

d. Display the publisher and number of books of each publisher in the descending order of the count.

e. Display the title, current price and the price after a discount of 10% in the alphabetical order of book title.

SQL statements :

1. SQL Query to create the table .

```
CREATE TABLE Book
(
Book_ID  INT PRIMARY KEY,
Book_Name VARCHAR (20),
Author_Name VARCHAR(25),
Pub_Name VARCHAR(25),
Price DEC(10,2)
);
```

2. SQL Query to insert 5 records into the table
   I. INSERT INTO Book
   VALUES (1, ’Agnichirakukal’, ’A.P.J Abdul Kalam’, ’DC Books’,157);
   II. INSERT INTO Book
   VALUES (2, ’Computer Science’, ’Team of Teachers’, ’SCERT’,100);
   III. INSERT INTO Book
   VALUES (3, ’Aarachar’, ‘Meera.K.R’, ’DC Books’,370);
   IV. INSERT INTO Book
   VALUES (4, ’Randamoozham’, ’M.T Vasudevan Nair’, ‘Current Books’,249);

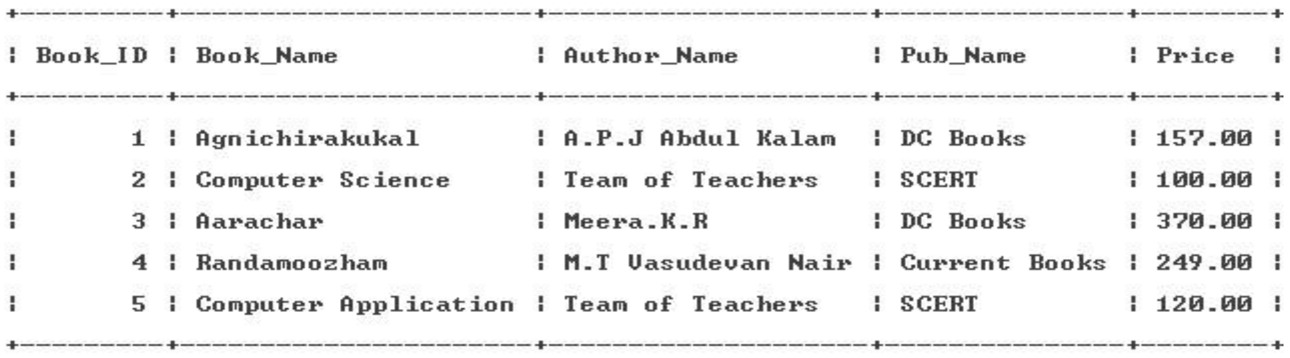
V. INSERT INTO Book
VALUES (5, ’Computer Application’, ’Team of Teachers’, ‘SCERT’,120);

3. SQL Query for each question

a. CREATE VIEW ScertView AS
SELECT * FROM Book WHERE Pub_Name =’SCERT’;
b. SELECT Pub_Name , AVG(Price)
FROM Book
GROUP BY Pub_Name;
c. SELECT * FROM Book
WHERE Price = (SELECT MAX(Price) FROM Book);
d. SELECT Pub_Name , COUNT(*)
FROM Book
GROUP BY Pub_Name
ORDER BY COUNT(*) DESC;

e. SELECT Book_Name, Price, Price-(Price*10/100)
FROM Book
ORDER BY Book_Name ASC;

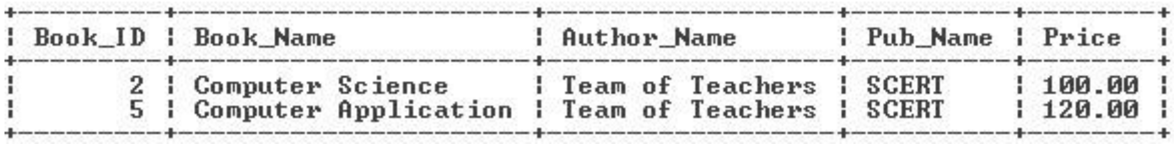
Table with sample records



```
+---------+----------------------+--------------------+---------------+--------+
| Book_ID | Book_Name            | Author_Name        | Pub_Name      | Price  |
+---------+----------------------+--------------------+---------------+--------+
|       1 | Agnichirakukal       | A.P.J Abdul Kalam  | DC Books      | 157.00 |
|       2 | Computer Science     | Team of Teachers   | SCERT         | 100.00 |
|       3 | Aarachar             | Meera.K.R          | DC Books      | 370.00 |
|       4 | Randamoozham         | M.T Vasudevan Nair | Current Books | 249.00 |
|       5 | Computer Application | Team of Teachers   | SCERT         | 120.00 |
+---------+----------------------+--------------------+---------------+--------+
```

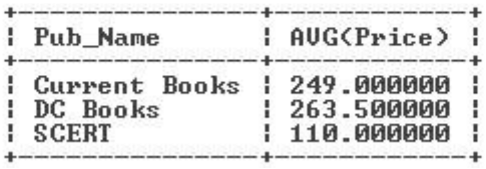
Output of Queries

a. Query OK, 0 rows affected



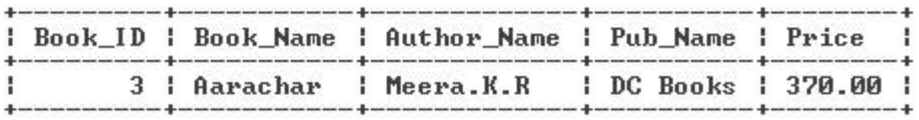
```
+---------+----------------------+------------------+----------+--------+
| Book_ID | Book_Name            | Author_Name      | Pub_Name | Price  |
+---------+----------------------+------------------+----------+--------+
|       2 | Computer Science     | Team of Teachers | SCERT    | 100.00 |
|       5 | Computer Application | Team of Teachers | SCERT    | 120.00 |
+---------+----------------------+------------------+----------+--------+
```

b. 3 rows in set.



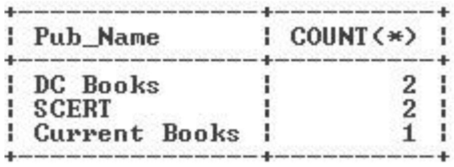
```
+---------------+------------+
| Pub_Name      | AVG(Price) |
+---------------+------------+
| Current Books | 249.000000 |
| DC Books      | 263.500000 |
| SCERT         | 110.000000 |
+---------------+------------+
```

c. 1 row in set.



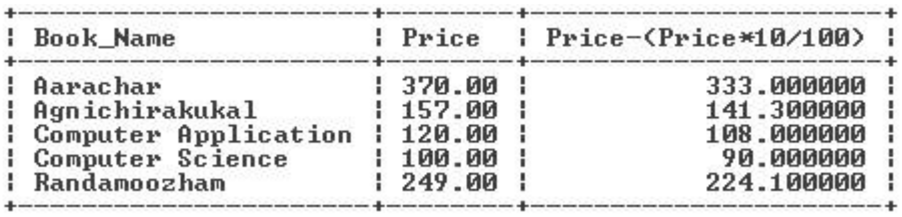
```
+---------+-----------+-------------+----------+--------+
| Book_ID | Book_Name | Author_Name | Pub_Name | Price  |
+---------+-----------+-------------+----------+--------+
|       3 | Aarachar  | Meera.K.R   | DC Books | 370.00 |
+---------+-----------+-------------+----------+--------+
```

d.3 rows in set.



```
+---------------+----------+
| Pub_Name      | COUNT(*) |
+---------------+----------+
| DC Books      |        2 |
| SCERT         |        2 |
| Current Books |        1 |
+---------------+----------+
```

e.5 rows in set.



```
+----------------------+--------+----------------------+
| Book_Name            | Price  | Price-(Price*10/100) |
+----------------------+--------+----------------------+
| Aarachar             | 370.00 |           333.000000 |
| Agnichirakukal       | 157.00 |           141.300000 |
| Computer Application | 120.00 |           108.000000 |
| Computer Science     | 100.00 |            90.000000 |
| Randamoozham         | 249.00 |           224.100000 |
+----------------------+--------+----------------------+
```

*****

Experiment No: 11

Title: **Create a table Employee with the following fields and insert at least 5 records into the table except the column Gross_pay and DA.**

Emp_code Integer Primary key
Emp_name Varchar (20)
Designation Varchar (25)
Department Varchar (25)
Basic Decimal (10,2)
DA Decimal (10,2)
Gross_pay Decimal (10,2)

a) Update DA with 75% of Basic.

b) Display the details of employees in Purchase, Sales and HR departments.

c) Update the Gross_pay with the sum of Basic and DA.

d) Display the details of employee with gross pay below 10000.

e) Delete all the clerks from the table.

SQL statements :

1. SQL Query to create the table .

```
CREATE TABLE Employee
(
Emp_code  INT PRIMARY KEY,
Emp_name VARCHAR (20),
Designation VARCHAR(25),
Department VARCHAR(25),
Basic DEC(10,2),
DA DEC(10,2),
Gross_pay DEC(10,2)
);
```

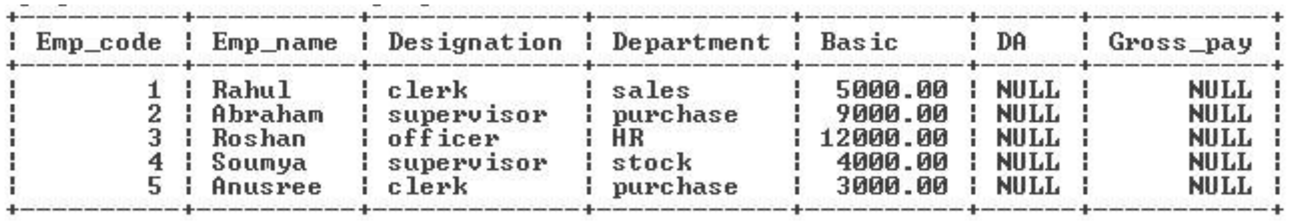
2. SQL Query to insert 5 records into the table
   - I. INSERT INTO Employee (Emp_code , Emp_name, Designation, Department, Basic) VALUES (1, ’Rahul’, ’clerk’, ‘sales’, 5000);
   - II. INSERT INTO Employee (Emp_code , Emp_name, Designation, Department, Basic) VALUES (2, ’Abraham’, ’supervisor’, ‘purchase’, 9000);
   - III. INSERT INTO Employee (Emp_code , Emp_name, Designation, Department, Basic) VALUES (3, ’Roshan’, ’officer’, ‘HR’ , 12000);
   - IV. INSERT INTO Employee (Emp_code , Emp_name, Designation, Department, Basic) VALUES (4, ’Soumya’, ’supervisor’, ‘stock’, 4000);

V. INSERT INTO Employee (Emp_code , Emp_name, Designation, Department, Basic) VALUES (5, ’Anusree’, ’clerk’, ‘purchase’, 3000);

3. SQL Query for each question

a. UPDATE Employee
   SET DA = Basic * 75 /100 ;
b. SELECT * FROM Employee
   WHERE Department IN (‘sales’, ’purchase’, ’HR’);
c. UPDATE Employee
   SET Gross_pay = Basic + DA;
d. SELECT * FROM Employee
   WHERE Gross_pay < 10000;
e. DELETE
   FROM Employee
   WHERE Designation = ‘clerk’ ;

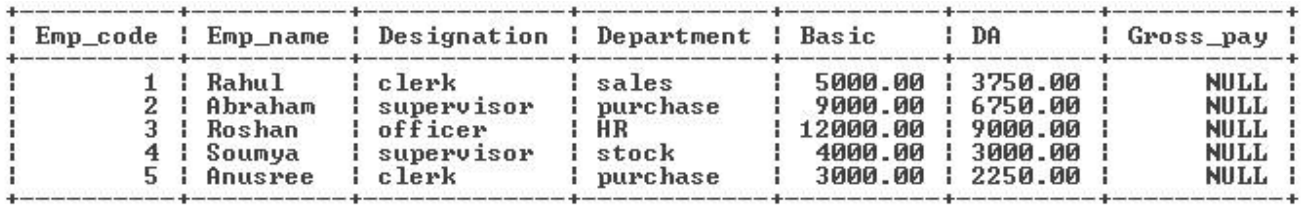
Table with sample records



```
+----------+----------+------------+------------+----------+------+-----------+
| Emp_code | Emp_name | Designation | Department | Basic    | DA   | Gross_pay |
+----------+----------+------------+------------+----------+------+-----------+
|        1 | Rahul    | clerk      | sales      |  5000.00 | NULL |      NULL |
|        2 | Abraham  | supervisor | purchase   |  9000.00 | NULL |      NULL |
|        3 | Roshan   | officer    | HR         | 12000.00 | NULL |      NULL |
|        4 | Soumya   | supervisor | stock      |  4000.00 | NULL |      NULL |
|        5 | Anusree  | clerk      | purchase   |  3000.00 | NULL |      NULL |
+----------+----------+------------+------------+----------+------+-----------+
```

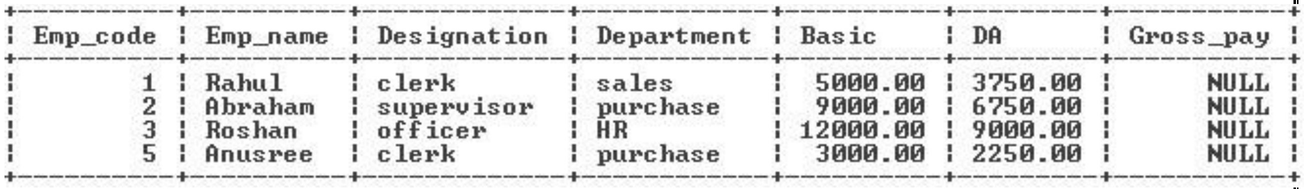
Output of Queries

a. Query OK, 5 rows affected.



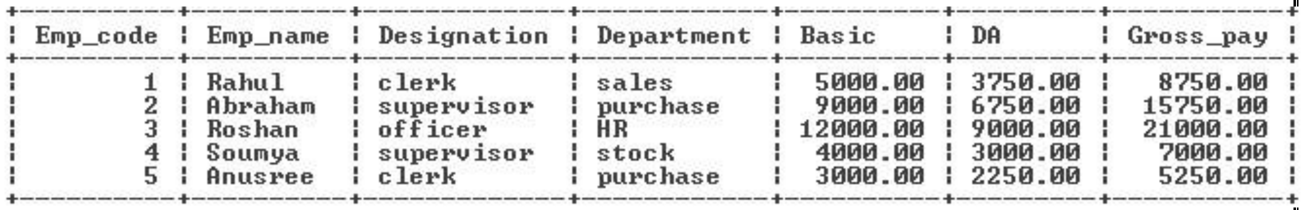
```
+----------+----------+-------------+------------+----------+---------+-----------+
| Emp_code | Emp_name | Designation | Department | Basic    | DA      | Gross_pay |
+----------+----------+-------------+------------+----------+---------+-----------+
|        1 | Rahul    | clerk       | sales      |  5000.00 | 3750.00 |      NULL |
|        2 | Abraham  | supervisor  | purchase   |  9000.00 | 6750.00 |      NULL |
|        3 | Roshan   | officer     | HR         | 12000.00 | 9000.00 |      NULL |
|        4 | Soumya   | supervisor  | stock      |  4000.00 | 3000.00 |      NULL |
|        5 | Anusree  | clerk       | purchase   |  3000.00 | 2250.00 |      NULL |
+----------+----------+-------------+------------+----------+---------+-----------+
```

b. 4 rows in set.



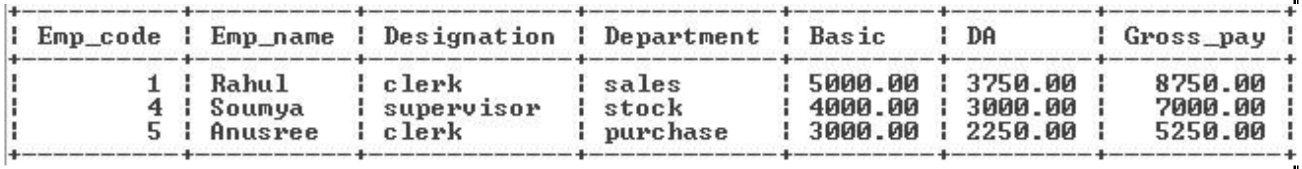
```
+----------+----------+-------------+------------+----------+---------+-----------+
| Emp_code | Emp_name | Designation | Department | Basic    | DA      | Gross_pay |
+----------+----------+-------------+------------+----------+---------+-----------+
|        1 | Rahul    | clerk       | sales      |  5000.00 | 3750.00 |      NULL |
|        2 | Abraham  | supervisor  | purchase   |  9000.00 | 6750.00 |      NULL |
|        3 | Roshan   | officer     | HR         | 12000.00 | 9000.00 |      NULL |
|        5 | Anusree  | clerk       | purchase   |  3000.00 | 2250.00 |      NULL |
+----------+----------+-------------+------------+----------+---------+-----------+
```

c. Query OK, 5 rows affected.



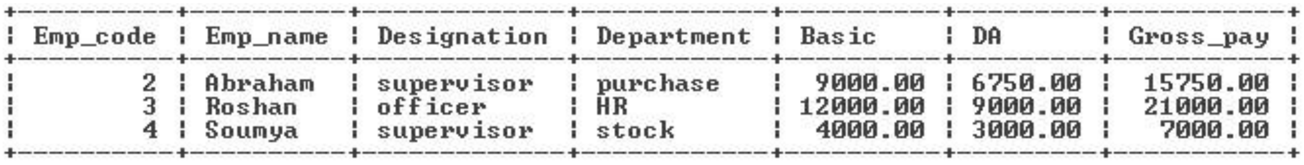
```
+----------+----------+-------------+------------+----------+---------+-----------+
| Emp_code | Emp_name | Designation | Department | Basic    | DA      | Gross_pay |
+----------+----------+-------------+------------+----------+---------+-----------+
|        1 | Rahul    | clerk       | sales      |  5000.00 | 3750.00 |   8750.00 |
|        2 | Abraham  | supervisor  | purchase   |  9000.00 | 6750.00 |  15750.00 |
|        3 | Roshan   | officer     | HR         | 12000.00 | 9000.00 |  21000.00 |
|        4 | Soumya   | supervisor  | stock      |  4000.00 | 3000.00 |   7000.00 |
|        5 | Anusree  | clerk       | purchase   |  3000.00 | 2250.00 |   5250.00 |
+----------+----------+-------------+------------+----------+---------+-----------+
```

d. 3 rows in set.



```
+----------+----------+-------------+------------+---------+---------+-----------+
| Emp_code | Emp_name | Designation | Department | Basic   | DA      | Gross_pay |
+----------+----------+-------------+------------+---------+---------+-----------+
|        1 | Rahul    | clerk       | sales      | 5000.00 | 3750.00 |   8750.00 |
|        4 | Soumya   | supervisor  | stock      | 4000.00 | 3000.00 |   7000.00 |
|        5 | Anusree  | clerk       | purchase   | 3000.00 | 2250.00 |   5250.00 |
+----------+----------+-------------+------------+---------+---------+-----------+
```

e. Query OK, 2 rows affected.



```
+----------+----------+-------------+------------+----------+---------+-----------+
| Emp_code | Emp_name | Designation | Department | Basic    | DA      | Gross_pay |
+----------+----------+-------------+------------+----------+---------+-----------+
|        2 | Abraham  | supervisor  | purchase   |  9000.00 | 6750.00 |  15750.00 |
|        3 | Roshan   | officer     | HR         | 12000.00 | 9000.00 |  21000.00 |
|        4 | Soumya   | supervisor  | stock      |  4000.00 | 3000.00 |   7000.00 |
+----------+----------+-------------+------------+----------+---------+-----------+
```

*****

Experiment No: 12

Title: **Create a table *Bank* with the following fields and insert at least 5 records into the table except the column Gross_pay and DA.**

Acc_No Integer Primary key
Acc_Name Varchar (20)
Branch_Name Varchar (25)
Acc_ Type Varchar (10)
Amount Decimal (10,2)

a. Display the branch-wise details of account holders in the ascending order of the amount.

b. Insert a new column named Minimum_Amount into the table with default value 1000.

c. Update the Minimum_Amount column with the value 500 for the customers in branches other than Alappuzha and Malappuram.

d. Find the number of customers who do not have the minimum amount 1000.

e. Remove the details of SB accounts from Thiruvananthapuram branch who have zero (0) balance in their account.

SQL statements :

1. SQL Query to create the table .

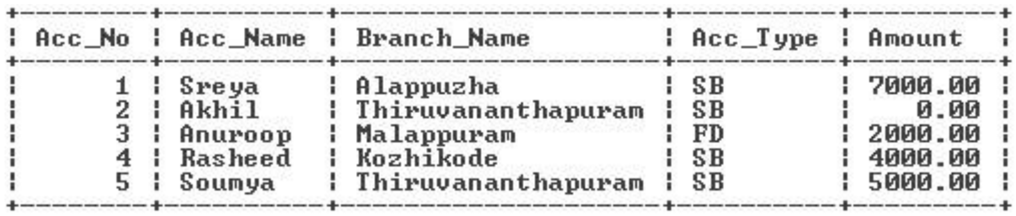
```
CREATE TABLE Bank
(
Acc_No  INT PRIMARY KEY,
Acc_Name VARCHAR (20),
Branch_Name VARCHAR(25),
Acc_Type VARCHAR(10),
Amount DEC(10,2)
);
```

2. SQL Query to insert 5 records into the table

I.
```
INSERT INTO Bank
VALUES (1, ’Sreya’, ’Alappuzha’,  ’SB’,7000);
```
II.
```
INSERT INTO Bank
VALUES (2, ’Akhil’, ’Thiruvananthapuram’, ’SB’,0);
```
III.
```
INSERT INTO Bank
VALUES (3, ’Anuroop’, ‘Malappuram’, ’FD’,2000);
```
IV.
```
INSERT INTO Bank
VALUES (4, ’Rasheed’, ’Kozhikode’, ‘SB’,4000);
```
V.
```
INSERT INTO Bank
VALUES (5, ’Soumya’, ’Thiruvananthapuram’,  ‘SB’,5000);
```

3. SQL Query for each question
   a. SELECT * FROM Bank
      ORDER BY Branch_Name, Amount ASC;
   b. ALTER TABLE Bank
      ADD COLUMN Minimum_Amount DEC(10,2) DEFAULT
      1000;
   c. UPDATE Bank
      SET Minimum_Amount = 500
      WHERE Branch_Name NOT IN (‘Alappuzha’ , ’Malappuram’);
   d. SELECT COUNT(*) FROM Bank
      WHERE Minimum_Amount<1000;
   e. DELETE FROM Bank
      WHERE Acc_Type =’SB’ AND Branch_Name=’Thiruvananthapuram’ AND
      Amount<=0;

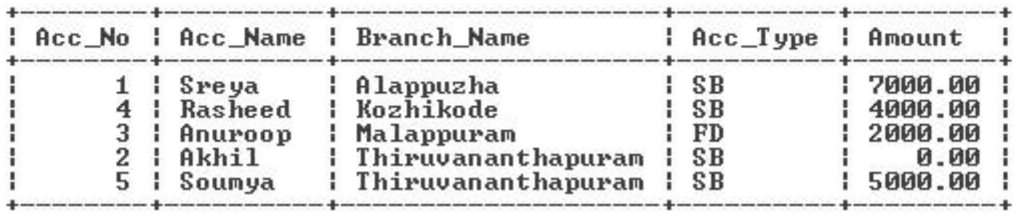
Table with sample records



```
+--------+----------+--------------------+----------+---------+
| Acc_No | Acc_Name | Branch_Name        | Acc_Type | Amount  |
+--------+----------+--------------------+----------+---------+
|      1 | Sreya    | Alappuzha          | SB       | 7000.00 |
|      2 | Akhil    | Thiruvananthapuram | SB       |    0.00 |
|      3 | Anuroop  | Malappuram         | FD       | 2000.00 |
|      4 | Rasheed  | Kozhikode          | SB       | 4000.00 |
|      5 | Soumya   | Thiruvananthapuram | SB       | 5000.00 |
+--------+----------+--------------------+----------+---------+
```

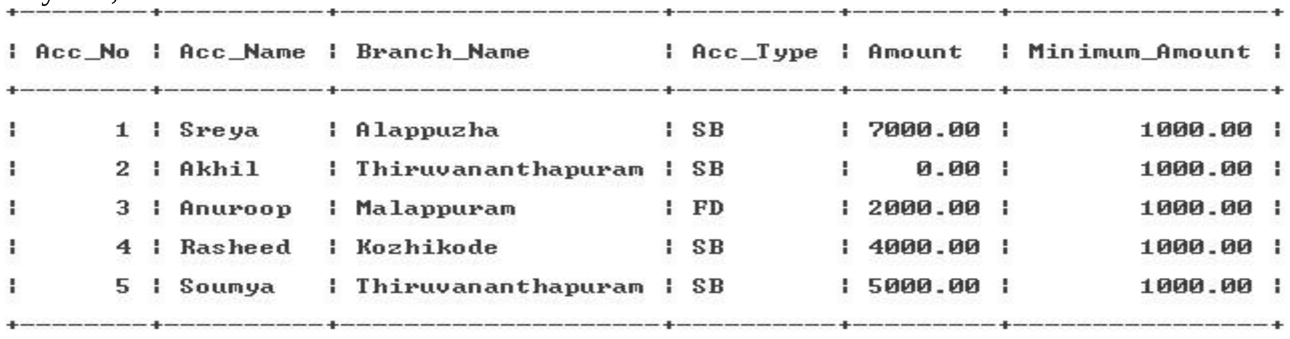
Output of Queries

a.5 rows in set



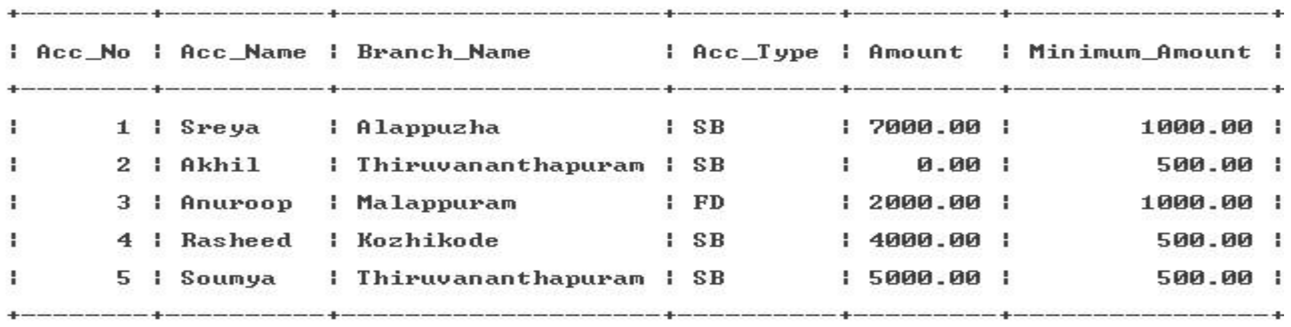
```
+--------+----------+--------------------+----------+---------+
| Acc_No | Acc_Name | Branch_Name        | Acc_Type | Amount  |
+--------+----------+--------------------+----------+---------+
|      1 | Sreya    | Alappuzha          | SB       | 7000.00 |
|      4 | Rasheed  | Kozhikode          | SB       | 4000.00 |
|      3 | Anuroop  | Malappuram         | FD       | 2000.00 |
|      2 | Akhil    | Thiruvananthapuram | SB       |    0.00 |
|      5 | Soumya   | Thiruvananthapuram | SB       | 5000.00 |
+--------+----------+--------------------+----------+---------+
```

b.Query OK, 0 rows affected.



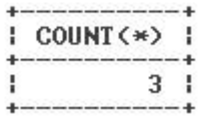
```
+--------+----------+--------------------+----------+---------+----------------+
| Acc_No | Acc_Name | Branch_Name        | Acc_Type | Amount  | Minimum_Amount |
+--------+----------+--------------------+----------+---------+----------------+
|      1 | Sreya    | Alappuzha          | SB       | 7000.00 |        1000.00 |
|      2 | Akhil    | Thiruvananthapuram | SB       |    0.00 |        1000.00 |
|      3 | Anuroop  | Malappuram         | FD       | 2000.00 |        1000.00 |
|      4 | Rasheed  | Kozhikode          | SB       | 4000.00 |        1000.00 |
|      5 | Soumya   | Thiruvananthapuram | SB       | 5000.00 |        1000.00 |
+--------+----------+--------------------+----------+---------+----------------+
```

c.Query OK, 3 rows affected.



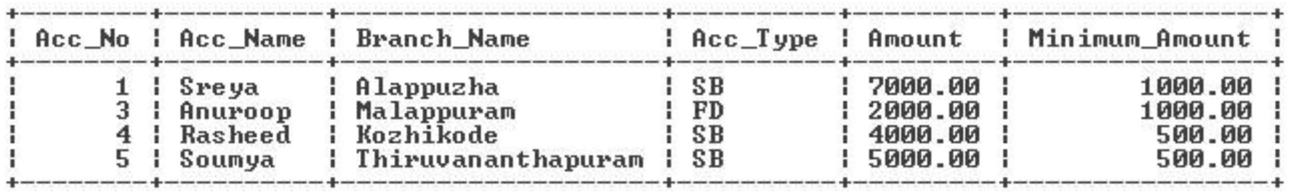
```
+--------+-----------+--------------------+----------+---------+----------------+
| Acc_No | Acc_Name  | Branch_Name        | Acc_Type | Amount  | Minimum_Amount |
+--------+-----------+--------------------+----------+---------+----------------+
|      1 | Sreya     | Alappuzha          | SB       | 7000.00 |        1000.00 |
|      2 | Akhil     | Thiruvananthapuram | SB       |    0.00 |         500.00 |
|      3 | Anuroop   | Malappuram         | FD       | 2000.00 |        1000.00 |
|      4 | Rasheed   | Kozhikode          | SB       | 4000.00 |         500.00 |
|      5 | Soumya    | Thiruvananthapuram | SB       | 5000.00 |         500.00 |
+--------+-----------+--------------------+----------+---------+----------------+
```

d.1 row in set.



```
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
```

e.Query OK, 1 row affected.



```
+--------+----------+--------------------+----------+---------+----------------+
| Acc_No | Acc_Name | Branch_Name        | Acc_Type | Amount  | Minimum_Amount |
+--------+----------+--------------------+----------+---------+----------------+
|      1 | Sreya    | Alappuzha          | SB       | 7000.00 |        1000.00 |
|      3 | Anuroop  | Malappuram         | FD       | 2000.00 |        1000.00 |
|      4 | Rasheed  | Kozhikode          | SB       | 4000.00 |         500.00 |
|      5 | Soumya   | Thiruvananthapuram | SB       | 5000.00 |         500.00 |
+--------+----------+--------------------+----------+---------+----------------+
```

*****

Experiment No: 13

Title: **Create a table *Stock*, which stores daily sales of items in a shop, with the following fields and insert at least 5 records into the table.**

Item_code Integer Primary key
Item_name Varchar (20)
Manufacturer_Code Varchar (5)
Qty Integer
Unit_Price Decimal (10,2)
Exp_Date Date

a. Display the details of items which expire after 31/3/2016 in the order of expiry date.

b. Find the number of items manufactured by the company "SATA".

c. Remove the items which expire between 31/12/2015 and 01/06/2016.

d. Add a new column named Reorder in the table to store the reorder level of items.

e. Update the column Reorder with value obtained by deducting 10% of the current stock.

SQL statements :

1. SQL Query to create the table .

```
CREATE TABLE Stock
(
Item_code  INT PRIMARY KEY,
Item_name VARCHAR (20),
Manufacturer_Code VARCHAR(5),
Qty INT,
Unit_Price DEC(10,2),
Exp_Date Date
);
```

2. SQL Query to insert 5 records into the table

I.
```
INSERT INTO Stock
VALUES (1, ’Fridge’, ’VOLTA’, 30 , 9000, ’2018-12-25’);
```
II.
```
INSERT INTO Stock
VALUES (2, ’Washing Machine’, ’SATA’,55,  8000, ’2016-05-
01’);
```
III.
```
INSERT INTO Stock
```

VALUES (3, ’Pressure Cooker’, ‘PREST’, 38, 4000, ’2016-02-04’);

IV. INSERT INTO Stock
VALUES (4, ’Grinder’, ’SATA’, 17, 6000, ’2016-03-22’);

V. INSERT INTO Stock
VALUES (5, ’Mixie’, ’SATA’, 60, 3000, ‘2016-07-28’);

3. SQL Query for each question

a. SELECT * FROM Stock
WHERE Exp_Date>’2016-03-31’
ORDER BY Exp_Date ASC ;

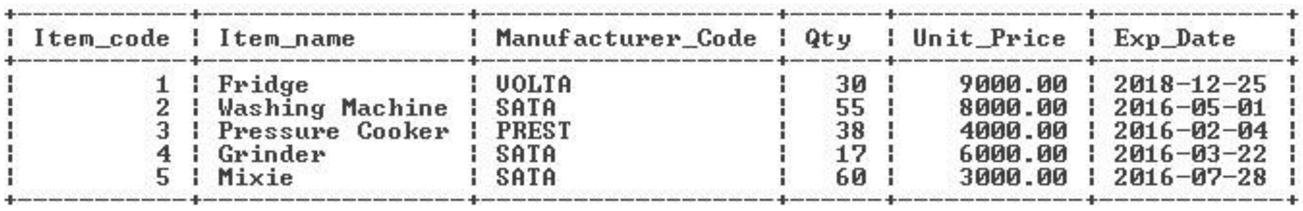
b. SELECT Manufacturer_Code , COUNT(*)
FROM Stock
WHERE Manufacturer_Code=’SATA’;

c. DELETE
FROM Stock
WHERE Exp_date BETWEEN '2015-12-31' and '2016-06-01';

d. ALTER TABLE Stock
ADD COLUMN Reorder INT ;

e. UPDATE Stock
SET Reorder=Qty-(Qty*10/100);

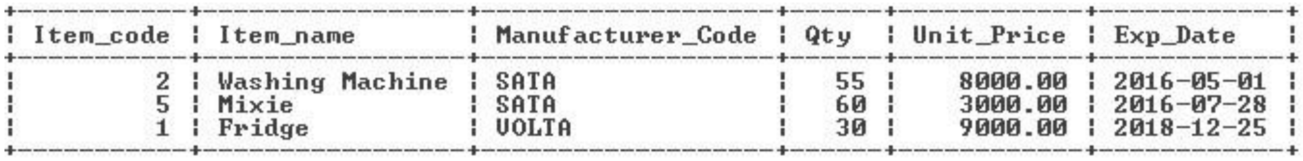
Table with sample records



```
+-----------+-----------------+-------------------+------+------------+------------+
| Item_code | Item_name       | Manufacturer_Code | Qty  | Unit_Price | Exp_Date   |
+-----------+-----------------+-------------------+------+------------+------------+
|         1 | Fridge          | VOLTA             |   30 |    9000.00 | 2018-12-25 |
|         2 | Washing Machine | SATA              |   55 |    8000.00 | 2016-05-01 |
|         3 | Pressure Cooker | PREST             |   38 |    4000.00 | 2016-02-04 |
|         4 | Grinder         | SATA              |   17 |    6000.00 | 2016-03-22 |
|         5 | Mixie           | SATA              |   60 |    3000.00 | 2016-07-28 |
+-----------+-----------------+-------------------+------+------------+------------+
```

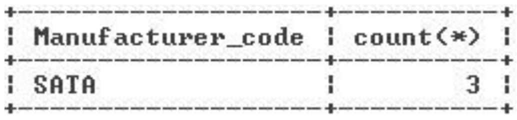
Output of Queries

a. 3 rows in set



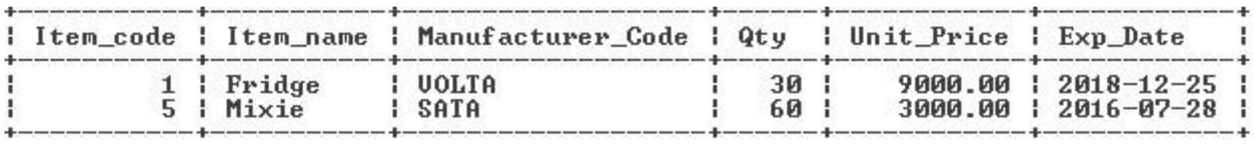
```
+-----------+-----------------+-------------------+------+------------+------------+
| Item_code | Item_name       | Manufacturer_Code | Qty  | Unit_Price | Exp_Date   |
+-----------+-----------------+-------------------+------+------------+------------+
|         2 | Washing Machine | SATA              |   55 |    8000.00 | 2016-05-01 |
|         5 | Mixie           | SATA              |   60 |    3000.00 | 2016-07-28 |
|         1 | Fridge          | VOLTA             |   30 |    9000.00 | 2018-12-25 |
+-----------+-----------------+-------------------+------+------------+------------+
```

b. 1 row in set.



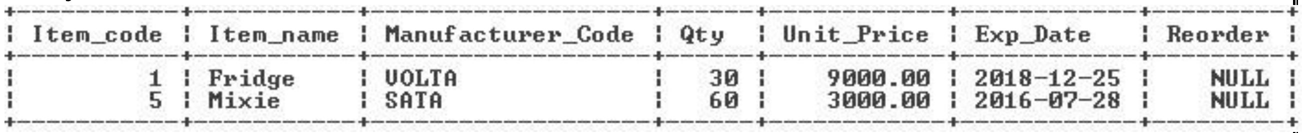
```
+-------------------+----------+
| Manufacturer_code | count(*) |
+-------------------+----------+
| SATA              |        3 |
+-------------------+----------+
```

c. Query OK, 3 rows affected.



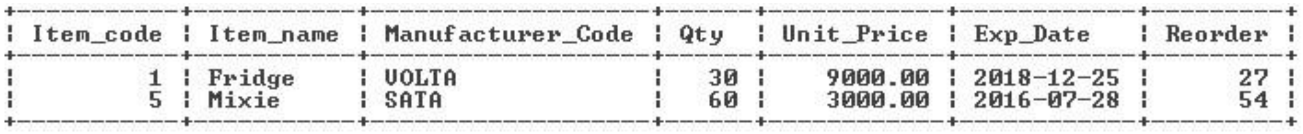
```
+-----------+-----------+-------------------+-----+------------+------------+
| Item_code | Item_name | Manufacturer_Code | Qty | Unit_Price | Exp_Date   |
+-----------+-----------+-------------------+-----+------------+------------+
|         1 | Fridge    | VOLTA             |  30 |    9000.00 | 2018-12-25 |
|         5 | Mixie     | SATA              |  60 |    3000.00 | 2016-07-28 |
+-----------+-----------+-------------------+-----+------------+------------+
```

d. Query OK, 0 rows affected



```
+-----------+-----------+-------------------+-----+------------+------------+---------+
| Item_code | Item_name | Manufacturer_Code | Qty | Unit_Price | Exp_Date   | Reorder |
+-----------+-----------+-------------------+-----+------------+------------+---------+
|         1 | Fridge    | VOLTA             |  30 |    9000.00 | 2018-12-25 |    NULL |
|         5 | Mixie     | SATA              |  60 |    3000.00 | 2016-07-28 |    NULL |
+-----------+-----------+-------------------+-----+------------+------------+---------+
```

e. Query OK, 2 rows affected.



```
+-----------+-----------+-------------------+-----+------------+------------+---------+
| Item_code | Item_name | Manufacturer_Code | Qty | Unit_Price | Exp_Date   | Reorder |
+-----------+-----------+-------------------+-----+------------+------------+---------+
|         1 | Fridge    | VOLTA             |  30 |    9000.00 | 2018-12-25 |      27 |
|         5 | Mixie     | SATA              |  60 |    3000.00 | 2016-07-28 |      54 |
+-----------+-----------+-------------------+-----+------------+------------+---------+
```

*****

Experiment No: 14

Title: Implementation of PL/SQL commands

Objective:

- ✓ To understand the implementation of PL/SQL.
- ✓ To write query for a database

Theory:

PL/SQL (Procedural Language/SQL) is a procedural extension of Oracle-SQL. PL/SQL allows users and designers to develop complex database applications that require the usage of control structures and procedural elements such as procedures, functions, and modules. The basic construct in PL/SQL is a block. In a block, constants and variables can be declared, and variables can be used to store query results. Statements in a PL/SQL block include SQL statements, control structures (loops), condition statements (if-then-else), exception handling, and calls of other PL/SQL blocks.

Structure of PL/SQL-Blocks

A PL/SQL block has an optional declare section, a part containing PL/SQL statements, and an optional exception-handling part. Thus the structure of a PL/SQL looks as follows (brackets [ ] enclose optional parts):

```
[<Block header>]
[declare
<Constants>
<Variables>
<Cursors>
<User defined exceptions>]
begin
<PL/SQL statements>
[exception
<Exception handling>]
end;
```

Declarations

Constants, variables, cursors, and exceptions used in a PL/SQL block must be declared in the declare section of that block. Variables and constants can be declared as follows:

<variable name> [constant] <data type> [not null] [:= <expression>];

Exception Handling

A PL/SQL block may contain statements that specify exception handling routines. Each error or warning during the execution of a PL/SQL block raises an exception.

Write a PL/SQL program to add two numbers?

```
SQL>   declare
        a number(10):=&a;
        b number(10):=&b;
        c number(10);
        begin
        c:=a+b;
        dbms_output.put_line('Sum is '||c);
        end;
  /
```