1. Creation of A Database And Writing Sql Queries To Retrieve Information From The Database

AIM:

To create a database and write SQL queries to retrieve information from the database.

DESCRIPTION:

Data Definition Language:

A data definition language or data description language (DDL) is a syntax similar to a computer programming language for defining data structures, especially database schemas. Many data description languages use a declarative syntax to define fields and data types.

The DDL commands in the SQL are

- CREATE
- RENAME
- ALTER
- TRUNCATE
- DROP

CREATE

CREATE command is used to create tables in database.

Syntax

CREATE TABLE <table name> (column1 datatype1,…, column datatypen);

DESC

To view the table structure.

Syntax

Desc<table name>

RENAME

RENAME command is used to rename the tables.

Syntax

Rename <old table name> to <new table name>;

ALTER

The structure of a table can be changed using this command. Using this command we can do the following.

i. Add a new column.

ii. Change the width of a data type

iii. Change the data type of a column

While altering the columns the values of the particular column should be empty.

Syntax

ALTER TABLE <table name> MODIFY (column datatype,…);

ALTER TABLE <table name> ADD (column datatype,…);

TRUNCATE

It is used to remove all the records in the table including the space allocated for the table. But the structure of the table is retained.

Syntax

TRUNCATE TABLE <table name>;

PROGRAM :

DDL commands

1. CREATE

Example:

```
CREATE TABLE CUSTOMERS(
InsuranceID
INT, Name
VARCHAR(50)
,DOB DATE,
NIN INT,
Location VARCHAR(255)
);
```

2. ALTER

Example:

```
ALTER TABLE CUSTOMERS ADD email_id VARCHAR(50);
```

3. TRUNCATE

Example:

```
TRUNCATE table CUSTOMERS;
```

4. DROP

Example:

```
DROP TABLE CUSTOMERS;
```

5. RENAME

Example:

```
RENAME TABLE CUSTOMERS to CUSTOMERINFO;
```

6. COMMENT

Example:

--Line1;

Multi-Line comments:

Statements enclosed in /**/ are treated as Multi-line comments

/* Line1,
Line2 */

RESULT:

Thus the creation of a database and writing SQL queries to retrieve information from the database wasimplemented.

2. PERFORMING INSERTION, DELETION, MODIFYING, ALTERING, UPDATING AND VIEWING RECORDS BASED ON CONDITIONS

AIM:

To Perform Insertion, Deletion, Modifying, Altering, Updating and Viewing records based on conditions in DBMS.

DESCRIPTION:

Data Manipulation Language (DML)

Data Manipulation Language performs the following operations.

- Insert the Data
- Update the Data
- Delete the Data
- Retrieve the Data

The DML commands in the SQL are INSERT, UPDATE, DELETE, SELECT.

INSERT

It is used to insert the values into the table.

Syntax

INSERT INTO <table name> VALUES (value1, value2…);

SQL> INSERT INTO employee1 VALUES('&Employee_name', '&employee_no', '&dept_name', '&dept_no','&date_of_join');

Using this we can insert 'N' no. of rows.

UPDATE

The Update command is used to update (changing values in one or two columns of a row) rows in a table. Specific rows can also be updated based on some condition.

If the WHERE clause is omitted, then the changes take place in all rows of the table.

Syntax

UPDATE <table_name> SET column1=expression,column2=expression… WHERE <search_condition>;

DELETE

The delete command is used to delete rows from a table.

Syntax

DELETE FROM<table_name> [WHERE <search_condition>];Deletion of all rows

SQL>DELETE FROM <table_name>;

This causes the deletion of all rows in the table.

SELECT

The Select command is used to retrieve the stored data from a table.

Syntax

SELECT * FROM <table_name>;

PROGRAM :

DML Commands

INSERT

Example:

Both the below ways are correct.

INSERT INTO CUSTOMERS (InsuranceID, Name, DOB, NIN, Location,email_id) **VALUES** ('123', 'Mango','2000-01-01','56789','LO','Mango@xyz.com');

INSERT INTO CUSTOMERS **VALUES** ('123', 'Mango','2000-01-01','56789','LO','Mango@xyz.com');

2. SELECT

Example:

SELECT * **FROM** CUSTOMERS;

3. UPDATE

Example:

UPDATE CUSTOMERS **SET** email_id = 'mango.lo@xyz.com' **WHERE** InsuranceID='123';

4. DELETE

Example:

DELETE FROM CUSTOMERS **where** InsuranceID='123';

RESULT:

Thus the Insertion, Deletion, Modifying, Altering, Updating and Viewing records based on conditions in RDBMS were executed and verified.

3.DCL Commands

PERFORMING GRANT AND REVOKE BASED ON CONDITION.

AIM:

To Perform grant and revoke based on condition in dbms.

DESCRIPTION:

1. GRANT
GRANT statement is used to provide access privileges to users to access the database.

Syntax:
GRANT privileges **ON** object **TO user**;

Note: Privileges can be SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER, CREATE, ALL. You can also specify combination of these privileges in a statement.

2. REVOKE
REVOKE statement is used to withdraw the access priviliges given to a user by GRANT statement.

Syntax:
REVOKE privileges **ON** object **FROM user**;

PROGRAM :

DCL Commands

GRANT

GRANT Connect to Database
GRANT CONNECT ON DATABASE database_name **TO** username;

GRANT Usage on Schema
GRANT USAGE **ON** SCHEMA database_name **TO** username;

Grant access to all tables in the database:

GRANT ALL PRIVILEGES **ON ALL** TABLES **IN** SCHEMA schema_name TO username;

GRANT ALL PRIVILEGES **ON ALL** SEQUENCES **IN** SCHEMA schema_name **TO** username;

GRANT ALL PRIVILEGES **ON** DATABASE database_name **TO** username;

Grant permission to create database:

ALTER USER username CREATEDB;

Grant superuser access to a user
ALTER USER myuser **WITH** SUPERUSER;

REVOKE

Example:
REVOKE DELETE, **UPDATE ON** ORDERS **FROM** customer1;

RESULT:

Thus the GRANT and REVOKE Query were executed Successfully and verified.

4. Constraints in MySQL and Built-In.

PERFORMING Constraints in MySQL and Built-In BASED ON CONDITION.

AIM:

To perform the constraints and Built in functions in dbms.

DESCRIPTION:

1. Constraints in MySQL

Constraints are rules applied to table columns to ensure the accuracy and reliability of data in the database. Common types of constraints in MySQL include:

a) PRIMARY KEY

- Ensures that the column (or combination of columns) has unique values and cannot be NULL.

Syntax

```
CREATE TABLE table_name (
    column_name datatype PRIMARY KEY
);
```

b) FOREIGN KEY

- Establishes a relationship between two tables, ensuring that the value in a column matches values in a column of another table.

Syntax

```
CREATE TABLE child_table (
    column_name datatype,
    FOREIGN KEY (column_name) REFERENCES parent_table(parent_column)
);
```

c) UNIQUE

- Ensures that all values in a column are unique.

Syntax

```
CREATE TABLE table_name (
```

```
    column_name datatype UNIQUE
);
```

d) NOT NULL

- Ensures that a column cannot have a NULL value.

Syntax

```
CREATE TABLE table_name (
    column_name data_type NOT NULL
);
```

e) CHECK (Starting from MySQL 8.0)

- Ensures that all values in a column satisfy a specific condition.

Syntax

```
CREATE TABLE table_name (
    column_name data_type,
    CHECK (condition)
);
```

f) DEFAULT

- Assigns a default value to a column if no value is provided during the insertion of a row.

Syntax:

```
CREATE TABLE table_name (
    column_name data_type DEFAULT default_value
);
```

2. Built-in Functions in MySQL

MySQL provides a wide range of built-in functions to perform operations on data. Here are some common types of functions:

String Functions

CONCAT(): Concatenates two or more strings.

Syntax:
SELECT CONCAT(string1, string2, ...);

LOWER() / UPPER(): Converts a string to lowercase or uppercase.

Syntax

SELECT LOWER(string);

SELECT UPPER(string);

SUBSTRING(): Extracts a substring from a string.

Syntax
SELECT SUBSTRING(string, start_position, length);

LENGTH(): Returns the length of a string.

Syntax
SELECT LENGTH(string);

Numeric Functions

ABS(): Returns the absolute value of a number.

Syntax
SELECT ABS(number);

ROUND(): Rounds a number to a specified number of decimal places.

Syntax
SELECT ROUND(number, decimal_places);

CEIL() / FLOOR(): Returns the smallest integer greater than or equal to (ceiling) or the largest integer less than or equal to (floor) a number.

Syntax

SELECT CEILING(number);

SELECT FLOOR(number);

MOD(): Returns the remainder of a division.

Syntax
MOD(dividend, divisor)

Date and Time Functions

CURDATE(): Returns the current date.

Syntax
SELECT CURDATE();

NOW(): Returns the current date and time.

Syntax
SELECT NOW();

DATE_FORMAT(): Formats a date based on the provided format.

Syntax
DATE_FORMAT(date, format)

DATEDIFF(): Returns the number of days between two dates.

Syntax
DATEDIFF(date1, date2)

Aggregate Functions

COUNT(): Returns the number of rows.

Syntax
SELECT COUNT(column_name);

SUM(): Returns the sum of a numeric column.

Syntax
SELECT SUM(column_name);

AVG(): Returns the average value of a numeric column.

Syntax
SELECT AVG(column_name);

MAX() / MIN(): Returns the maximum or minimum value of a column.

Syntax

```
SELECT MIN(column_name);

SELECT MAX(column_name);
```

Conditional Functions

IF(): Returns a value based on a condition.

Syntax

```
SELECT IF(condition, value_if_true, value_if_false);
```

CASE: Evaluates a list of conditions and returns a value when the first condition is met.

Syntax

```
SELECT
   CASE
      WHEN condition1 THEN result1
      WHEN condition2 THEN result2
      ELSE result
   END
FROM table_name;
```

PROGRAM:

1.Constraints in MySQL and Built-In.

a) PRIMARY KEY

Example:

```
sql
CREATE TABLE orders (
order_id INT PRIMARY KEY,
emp_id INT,
CONSTRAINT fk_employee FOREIGN KEY (emp_id) REFERENCES employees(emp_id)
);
```

b) FOREIGN KEY

Example:

```
sql
CREATE TABLE orders (
order_id INT PRIMARY KEY,
```

```
emp_id INT,
CONSTRAINT fk_employee FOREIGN KEY (emp_id) REFERENCES employees(emp_id)
);
```

c) UNIQUE
Example:

```
sql
CREATE TABLE customers (
customer_id INT,
email VARCHAR(255) UNIQUE
);
```

d) NOT NULL
Example:

```
sql
Copy code
CREATE TABLE products (
product_id INT,
product_name VARCHAR(100) NOT NULL
);
```

e) CHECK (Starting from MySQL 8.0)
Example:

```
sql
CREATE TABLE employees (
emp_id INT,
age INT,
CHECK (age >= 18)
);
```

f) DEFAULT
Example:

```
sql
CREATE TABLE orders (
order_id INT,
status VARCHAR(20) DEFAULT 'Pending'
);
```

2.Built-in Functions in

MySQLString Functions

CONCAT
example
sql
SELECT CONCAT('Hello', ' ', 'World!');

LOWER() / UPPER():
Example:
sql
SELECT LOWER('HELLO'), UPPER('hello');

SUBSTRING():
Example:
sql
SELECT SUBSTRING('Hello World', 1, 5); -- Output: 'Hello'

LENGTH():
Example:
sql
SELECT LENGTH('Hello');

Numeric Functions

ABS():
Example:
sql
SELECT ABS(-5); -- Output: 5

ROUND():
Example:
sql
SELECT ROUND(123.456, 2); -- Output: 123.46

CEIL() / FLOOR():
Example:
sql
SELECT CEIL(123.456), FLOOR(123.456); -- Output: 124, 123

MOD():
Example:
sql
SELECT MOD(10, 3); -- Output: 1

Date and Time Functions

CURDATE():
Example:
sql
SELECT CURDATE();

NOW():
Example:
sql
SELECT NOW();

DATE_FORMAT():
Example:
sql
SELECT DATE_FORMAT(NOW(), '%Y-%m-%d %H:%i:%s');

DATEDIFF():
Example:
sql
SELECT DATEDIFF('2024-12-31', '2024-01-01');

Aggregate Functions

COUNT():
Example:
sql
SELECT COUNT(*) FROM orders;

SUM():
Example:
sql
SELECT SUM(price) FROM products;

AVG():
Example:
sql
SELECT AVG(salary) FROM employees;

MAX() / MIN():
Example:
sql
SELECT MAX(salary), MIN(salary) FROM employees;

Conditional Functions

IF():

Example:

sql

```
SELECT IF(salary > 5000, 'High', 'Low') FROM employees;
```

CASE:

Example:

sql

```
SELECT
CASE
WHEN salary > 5000 THEN 'High'
WHEN salary BETWEEN 3000 AND 5000 THEN 'Medium'
ELSE 'Low'
END AS salary_level
FROM employees;
```

RESULT:

Thus the Queries in MYSQL were executed Successfully and output is verified.

5. Joins and Group-by functions

PERFORMING JOINS AND GROUP-BY FUNCTIONS BASED ON CONDITION.

AIM:

To perform the Joins and Group-by functions in dbms.

DESCRIPTION:

JOINs in MySQL

Joins are used to combine rows from two or more tables based on a related column between them.

a) INNER JOIN

- Returns records that have matching values in both tables.

Syntax

SELECT columns

FROM table1

INNER JOIN table2

ON table1.common_column = table2.common_column;

b) LEFT JOIN (or LEFT OUTER JOIN)

- Returns all records from the left table, and the matched records from the right table. If there is no match, NULL values will appear for columns from the right table.

Syntax

SELECT columns

FROM table1

LEFT JOIN table2

ON table1.common_column = table2.common_column;

c) RIGHT JOIN (or RIGHT OUTER JOIN)

- Returns all records from the right table, and the matched records from the left table. If there is no match, NULL values will appear for columns from the left table.

Syntax

SELECT columns

FROM table1

RIGHT JOIN table2

ON table1.common_column = table2.common_column;

GROUP BY in MySQL

The GROUP BY clause is used with aggregate functions (like COUNT(), SUM(), AVG(), etc.) to group the result set by one or more columns.

a) Using GROUP BY with Aggregate Functions

Syntax

SELECT column_name(s), aggregate_function(column_name)

FROM table_name

WHERE condition

GROUP BY column_name(s)

ORDER BY aggregate_function(column_name);

b) GROUP BY with HAVING

- The HAVING clause allows you to filter the results after grouping, similar to how WHERE works before grouping.

Syntax

SELECT column_name(s), aggregate_function(column_name)

FROM table_name

WHERE condition

GROUP BY column_name(s)

HAVING aggregate_condition

ORDER BY column_name(s);

c) Using GROUP BY with Multiple Columns

You can group by more than one column to get more granular results.

Syntax

SELECT column_name1, column_name2, aggregate_function(column_name)

FROM table_name

WHERE condition

GROUP BY column_name1, column_name2

ORDER BY aggregate_function(column_name);

Combining JOIN and GROUP BY

You can also combine JOIN and GROUP BY in the same query to summarize and aggregate data across multiple tables.

Syntax

SELECT column_name1, column_name2, aggregate_function(column_name)

FROM table1

JOIN table2 ON table1.common_column = table2.common_column

WHERE condition

GROUP BY column_name1, column_name2

ORDER BY aggregate_function(column_name);

PROGRAM :

Joins and Group-by functions

a) INNER JOIN

Example:

sql

SELECT orders.order_id, customers.customer_name

FROM orders

INNER JOIN customers ON orders.customer_id = customers.customer_id;

This query retrieves orders with their corresponding customer names, assuming that both the orders and customers tables have a customer_id column to join on.

b) LEFT JOIN (or LEFT OUTER JOIN)

Example:

sql

SELECT orders.order_id, customers.customer_name

FROM orders

LEFT JOIN customers ON orders.customer_id = customers.customer_id;

This query returns all orders, even if they do not have a corresponding customer. If a customer is missing, NULL will be shown in the customer_name column.

c) RIGHT JOIN (or RIGHT OUTER JOIN)

Example:

sql

SELECT orders.order_id, customers.customer_name

FROM orders

RIGHT JOIN customers ON orders.customer_id = customers.customer_id;

This query retrieves all customers, even if they don't have any orders.

d) FULL JOIN (or FULL OUTER JOIN)

Example:

sql

SELECT orders.order_id, customers.customer_name

FROM orders

LEFT JOIN customers ON orders.customer_id = customers.customer_id

UNION

SELECT orders.order_id, customers.customer_name

FROM orders

RIGHT JOIN customers ON orders.customer_id = customers.customer_id;

This query returns all orders and customers, whether or not they have a match in the other table.

GROUP BY in MySQL

a) Using GROUP BY with Aggregate Functions

Example:

sql

SELECT customer_id, COUNT(order_id) AS total_orders

FROM orders

GROUP BY customer_id;

This query returns the total number of orders for each customer.

b) GROUP BY with HAVING

Example:

sql

SELECT customer_id, COUNT(order_id) AS total_orders

FROM orders

GROUP BY customer_id

HAVING total_orders > 5;

This query shows customers who have placed more than 5 orders.

c) Using GROUP BY with Multiple Columns

Example:

sql

SELECT customer_id, product_id, COUNT(order_id) AS total_orders

FROM orders

GROUP BY customer_id, product_id;

This query returns the number of orders for each customer by each product.

Combining JOIN and GROUP BY

Example:

sql

SELECT customers.customer_name, COUNT(orders.order_id) AS total_orders

FROM customers

INNER JOIN orders ON customers.customer_id = orders.customer_id

GROUP BY customers.customer_name;

This query retrieves the total number of orders placed by each customer by joining the customers and orders tables and grouping the results by the customer name.

RESULT:

Thus the Queries in MYSQL were executed Successfully and output is verified.

6. SQL OPERATORS- ARITHMETIC, COMPARISON, AND LOGICAL OPERATIONS TO MANIPULATE AND RETRIEVE DATA FROM DATABASES.

AIM:

To perform arithmetic, comparison, and logical operations to manipulate and retrieve data from databases.

DESCRIPTION:

Operators in SQL

- Operators in SQL are symbols that help us to perform specific mathematical and logical computations on operands. An operator can either be unary or binary.
- The unary operator operates on one operand, and the binary operator operates on two operands

Types of Operators in SQL

Different types of operators in SQL are:

- Arithmetic operator
- Comparison operator
- Logical operator
- Bitwise Operators
- Compound Operators

PROGRAM :

CREATION OF TABLE

SQL>create table stud (sname varchar2(30), sid varchar2(10), sage number(10), sarea varchar2(20),
sdept varchar2(20));
Table created.

INSERTION OF VALUES INTO THE TABLE

SQL> insert into stud values ('ashwin',101,19,'anna nagar','aeronautical');
1 row created.
SQL> insert into stud values ('bhavesh',102,18,'nungambakkam','marine');
1 row created.
SQL> insert into stud values ('pruthvik',103,20,'annanagar','aerospace');
1 row created.
SQL> insert into stud values ('charith',104,20,'kilpauk','mechanical');
1 row created.
SQL> select * from stud;

SNAME	SID	SAGE	SAREA	SDEPT
ashwin	101	19	annanagar	aeronautical
bhavesh	102	18	nungambakkam	marine
pruthvik	103	20	annanagar	aerospace
charith	104	20	kilpauk	mechanical

RENAMING THE TABLE 'STUD'

SQL> rename stud to studs;

Table renamed.

ARITHMETIC OPERATION

SQL> select sname, sid+100 "stid" from studs;

SNAME	stid
ashwin	201
bhavesh	202
pruthvik	203
charith	204

DISPLAY ONLY DISTINCT VALUES

SQL> select distinct sarea from studs;

SAREA

annanagar

kilpauk

nungambakkam

USING THE WHERE CLAUSE

SQL> select sname,sage from studs where sage<=19;

SNAME	SAGE
ashwin	19
bhavesh	18

BETWEEN OPERATOR

SQL> select sname,sarea, sid from studs where sid between 102 and 104;

SNAME	SAREA	SID
bhavesh	nungambakkam	102
pruthvik	annanagar	103
charith	kilpauk	104

PATTERN MATCHING

SQL> select sname, sarea from studs where sarea like '%g%';

SNAME	SAREA
ashwin	annanagar
bhavesh	nungambakkam
pruthvik	Annanagar

LOGICAL AND OPERATOR

SQL> select sname ,sid from studs where sid>102 and sarea='annanagar';

SNAME	SID
pruthvik	103

LOGICAL OR OPERATOR

SQL> select sname ,sid from studs where sid>102 or sarea='annanagar';

SNAME	SID
ashwin	101
pruthvik	103
charith	104

Exercise 7:	PROGRAMS FOR IMPLEMENTATION OF FUNCTIONS USING PL/SQL
Date:	

Aim

To create a PL/SQL function that calculates and returns the total number of customers from a database table named customers.

Description

This function, named totalCustomers, queries the customers table to count the total number of records (customers) and returns this count. The function is then used in an anonymous PL/SQL block to displaythe total number of customers using the DBMS_OUTPUT.PUT_LINE procedure.

Syntax

The syntax to create a PL/SQL function is as follows:

```
sql
CREATE [OR REPLACE] FUNCTION function_name
  [ (parameter_name [IN | OUT | IN OUT] datatype [, ...]) ]
RETURN return_datatype IS | AS
  [declaration_section]
BEGIN
  executable_section
[EXCEPTION
  exception_section]
END [function_name];
```

SQL> Select * from customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

PROGRAM

```
CREATE OR REPLACE FUNCTION totalCustomers
RETURN number IS
  total number(2) := 0;
BEGIN
  SELECT count(*) into total FROM customers;
       RETURN total;
END;
/
DECLARE
  c number(2);
BEGIN
  c := totalCustomers();
  dbms_output.put_line('Total no. of Customers: ' || c);
END;
/
```

Output: 6

Exercise 8:	PROGRAMS FOR IMPLEMENTATION OF CURSORS USING PL/SQL
Date:	

Aim

To display details of employees from the employee table where the employee's name, department, or designation matches the provided input value.

Description

This PL/SQL block uses a cursor to iterate through the employee table and checks if the empname, depart, or disg matches the input value n. If a match is found, it prints the employee's details.

Syntax

The syntax for declaring a cursor and using it in a PL/SQL block is:

sql

```
DECLARE
  -- Variable declarations
  CURSOR cursor_name IS
    SELECT_statement;
  record_variable cursor_name%ROWTYPE;
BEGIN
  OPEN cursor_name;
  LOOP
    FETCH cursor_name INTO record_variable;
    EXIT WHEN cursor_name%NOTFOUND;
    -- Process each row
  END LOOP;
  CLOSE cursor_name;
END;
```

SQL> select * from employee;

EMPNO	EMPNAME	DEPART	DISG	SALARY	AGE
1000	Krishna	cs	hod	100000.5	29
1001	nagaraj	commerce	professor	55000	30
1002	robin	chemistry	professor	55000	33
1003	vanaga	cs	professor	44000	25
1004	ram	cs	staff	13000.5	34
1005	raja	agri	hod	20000	33
1006	ram	physics	hod	30000	45

PROGRAM:

```
declare
n varchar(20);
empno number;
empname varchar(20);
depart varchar(20);
disg varchar(20);
salary number;
age  number;
cursor c is select * from employee;
r sri % rowtype;
begin
```

```
n:='&n';
open c;
dbms_output.put_line('.....');
dbms_output.put_line(' EMP NUMBER EMP NAME DEPARTMENT
DIGN SALARY AGE' );

dbms_output.put_line('.....-');
loop
fetch c into r;
exit when c % notfound;
if r.empname=n or r.depart=n or r.disg=n then
dbms_output.put_line(r.empno||' '||r.empname||' '||r.depart||' '||r.disg||' '||r.salary||'
'||r.age);

end if;
end loop;
dbms_output.put_line('.....-');
close c;
end;
```

Output:

Enter value for n: Krishna

old 12: n:='&n';

new 12: n:='Krishna';

EMP NO	EMP NAME	DEPARTMENT	DIGN	SALARY	AGE
1000	Krishna	cs	hod	100000.5	29

PL/SQL procedure successfully completed.

SQL> desc employeenew;

Name	Null?	Type
ENO		NUMBER(5)
ENAME		VARCHAR2(20)
BP		NUMBER(10)
TA		NUMBER(10)
PF		NUMBER(10)
HRA		NUMBER(10)
TAX		NUMBER(10)
SALARY		NUMBER(10)

II.

Aim

To calculate the total salary for an employee by including allowances and deductions, insert the employee's details into the employeenew table, and display all records from the table.

Description

This PL/SQL block calculates Travel Allowance (TA), Provident Fund (PF), House Rent Allowance (HRA), and tax based on the basic pay (BP). It then calculates the total salary, inserts the data into the employeenew table, and displays all records from the table using a cursor.

Syntax

```
DECLARE
  variable_declarations;
  CURSOR cursor_name IS SELECT_statement;
  record_variable cursor_name%ROWTYPE;
BEGIN
  calculations;
  SQL_operations;
  cursor_operations;
END;
/
```

PROGRAM:

```
SQL> declare
       n number;
       m number;
       eno number:=&eno;
       ename varchar(20):='&ename';
       bp number:=&bp;
       ta  number;
       pf  number;
       hra number;
       tax number;
       salary number:=0;
       cursor c is select * from employeenew;
       r employeenew % rowtype;
       begin
       ta:=bp*0.03;
       pf:=bp*0.05;
```

```
 hra:=bp*0.02;
tax:=bp*0.06;
salary:=bp+ta-pf+hra-tax;
                select count(salary) into m from employeenew;
                insert into employeenew values(eno,ename,bp,ta,pf,hra,tax,salary);
                 open c;
                 dbms_output.put_line('EMPNO    ENAME    BP    TA    PF    HRA    TAX
                SALARY');
                dbms_output.put_line('.....');
                for n in 1..m+1 loop
                fetch c into r;
                 dbms_output.put_line(r.eno||'    '||r.ename||'    '||r.bp||'    '||r.ta||'    '||r.pf|
                |'    '||r.hra||'    '||r.tax||'    '||r.salary);
                end loop;
                dbms_output.put_line('.....');
                end;
   /
```

Output:

Enter value for eno: 1

old 4: eno number:=&eno;
new 4: eno number:=1;
Enter value for ename: sri

old 5: ename varchar(20):='&ename';
new 5: ename varchar(20):='krishna';
Enter value for bp: 40000
old 6: bp number:=&bp;
new 6: bp number:=40000;

EMPNO	ENAME	BP	TA	PF	HRA	TAX	SALARY
1	'Krishna'	40000	1200	2000	800	2400	37600

PL/SQL procedure successfully completed.

Exercise 9:	PROGRAMS FOR IMPLEMENTATION OF TRIGGERS USING PL/SQL
Date:	

Aim

To create a trigger that automatically assigns a grade to a student based on their average marks when a new record is inserted into the mark table.

Description

This PL/SQL block creates a trigger that evaluates the avg column of the mark table upon insertion of a new record. Depending on the value of avg, it assigns a grade (S, A, B, C, D, or F) to the grade column.

Syntax

The syntax for creating a trigger in PL/SQL is:

```
sql
CREATE [OR REPLACE] TRIGGER trigger_name
BEFORE | AFTER INSERT | UPDATE | DELETE
  ON table_name
  [FOR EACH ROW]
DECLARE
  -- variable declarations
BEGIN
  -- trigger logic
END;
/
```

PROGRAM

TRIGGER:

```
create or replace trigger grade before insert or update of avg on mark for each row
begin
    if :new.avg>=90 and :new.avg<=100 then :new.grade:='S';
    elseif :new.avg>=80 and :new.avg<90 then :new.grade:='A';
    elseif :new.avg>=70 and :new.avg<80 then :new.grade:='B';
    elseif :new.avg>=60 and :new.avg<70 then :new.grade:='C';
    elseif :new.avg>=50 and :new.avg<60 then :new.grade:='D';
    else :new.grade:='F';
end if;
end;
/
Trigger created
SQL> insert into mark values (&no,'&name',&avg,0);Enter value for no: 1
Enter value for name: sri
Enter value for avg: 90
old  1: insert into mark values (&no,'&name',&avg,0)
new  1: insert into mark values (1,'sri',90,0)
1 row created.
```

OUTPUT:

SQL> select * from mark;

NO	NAME	AVG	GRADE
2	robin	89	A
3	sakthi	56	D
1	sri	90	S

Exercise 10:	PROGRAMS FOR IMPLEMENTATION OF PACKAGES USING PL/SQL
Date:	

Aim

To create this package is to manage customer records in the customers table by providing procedures to add, delete, and list customers.

Description

This PL/SQL package, c_package, contains three procedures:

- **addCustomer**: Adds a new customer to the customers table.

  Syntax:

```
sql
PROCEDURE addCustomer(
 c_id customers.id%type,
 c_name customers.name%type,
 c_age customers.age%type,
 c_addr customers.address%type,
 c_sal customers.salary%type
);
```

- **delCustomer**: Deletes an existing customer from the customers table based on the customer ID.

  Syntax:

```
sql
PROCEDURE delCustomer(c_id customers.id%TYPE);
```

- **listCustomer**:
  - Lists all customer names from the customers table.
  - This procedure fetches all customer names from the customers table and outputs them using DBMS_OUTPUT.PUT_LINE.

Syntax:

sql

PROCEDURE listCustomer;

SQL> Select * from customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	3000.00
2	Khilan	25	Delhi	3000.00
3	kaushik	23	Kota	3000.00
4	Chaitali	25	Mumbai	7500.00
5	Hardik	27	Bhopal	9500.00
6	Komal	22	MP	5500.00

PROGRAM

Package Specification

```
CREATE OR REPLACE PACKAGE c_package AS
  //Adds a customer
  PROCEDURE addCustomer(c_id  customers.id%type,
  c_name  customerS.No.ame%type,
  c_age customers.age%type,
  c_addr customers.address%type,
  c_sal  customers.salary%type);
```

```
    //Removes a customer
    PROCEDURE delCustomer(c_id  customers.id%TYPE);
//Lists all customers
PROCEDURE listCustomer;
  END c_package;
  /
```

Package Body

```
  CREATE OR REPLACE PACKAGE BODY c_package AS

    PROCEDURE addCustomer(c_id  customers.id%type,
      c_name customerS.No.ame%type,
      c_age customers.age%type,
      c_addr  customers.address%type,
      c_sal  customers.salary%type)
    IS
    BEGIN
      INSERT INTO customers (id,name,age,address,salary)
        VALUES(c_id, c_name, c_age, c_addr, c_sal);
    END addCustomer;

    PROCEDURE delCustomer(c_id  customers.id%type) IS
    BEGIN
      DELETE FROM customers
      WHERE id = c_id;
    END delCustomer;

    PROCEDURE listCustomer IS
    CURSOR c_customers is
      SELECT  name FROM customers;
    TYPE c_list is TABLE OF customerS.No.ame%type;
    name_list c_list := c_list();
```

```
  counter integer :=0;
  BEGIN
    FOR n IN c_customers LOOP
    counter := counter +1;
    name_list.extend;
    name_list(counter) := n.name;
    dbms_output.put_line('Customer(' ||counter|| ')'||name_list(counter));
    END LOOP;
  END listCustomer;


END c_package;
  /
```

Exercise 11:	PROGRAMS FOR IMPLEMENTATION OF PROCEDURES USING PL/SQL
Date:	

AIM

To create this PL/SQL block is to demonstrate the use of a simple stored procedure to find and return the minimum of two numbers.

Description

This PL/SQL block declares three variables (a, b, and c). It also defines a procedure findMin that takes two input parameters (x and y) and returns the minimum of the two in the output parameter (z). The main block assigns values to a and b, calls findMin to find the minimum of a and b, and prints the result.

SYNTAX

```
DECLARE
   variable_name1 DATA_TYPE;
   variable_name2 DATA_TYPE;
   ...
PROCEDURE procedure_name(parameter1 IN DATA_TYPE, parameter2 IN DATA_TYPE, parameter3
OUT DATA_TYPE) IS
BEGIN
   -- Procedure logic here
END procedure_name;
BEGIN
   -- Variable initialization
   variable_name := value;

   -- Procedure call
   procedure_name(parameter1, parameter2, parameter3);
```

```
    -- Output result
    DBMS_OUTPUT.PUT_LINE('Your message: ' || variable_name);
END;
/
```

PROGRAM

```
  DECLARE
    a number;
    b number;
    c number;
  PROCEDURE findMin(x IN number, y IN number, z OUT number) IS
  BEGIN
    IF x < y THEN
      z:= x;
    ELSE
      z:= y;
    END IF;
  END;
  BEGIN
    a:= 23;
    b:= 45;
    findMin(a, b, c);
    dbms_output.put_line(' Minimum of (23, 45) : ' || c);
  END;
  /
```

Output:

```
    Minimum of (23, 45) : 23
```

Exercise 12:	PROGRAMS FOR CREATION OF FORMS AND REPORTS USING DEVELOPER TOOLS
Date:	

Aim

To creation of forms and reports usingdeveloper tools.

Description

The code consists of multiple event-driven subroutines that handle various tasks, including adding new customer records, retrieving and displaying existing records, and calculating billing amounts based on different tariffs. It uses ADO (ActiveX Data Objects) for database connectivity and interaction.

SQL> create table eb27(scno number(5) primary key, name varchar(15),address varchar(15),tariff varchar(10),lmu number, cmu number, units number, amt number(10,2), rdate date);

Table created.

SQL> create table ebbill27 (scno number(5), name varchar(15), tariff varchar(10), units number, amt number(10,2), rdate date);

Table created.

PROGRAM

```
Dim connect As ADODB.Connection
Dim rs As ADODB.Recordset
Private Sub Command1_Click()
Text1.SetFocus
Set rs = New ADODB.Recordset
rs.Open "select * from eb27", connect, adOpenKeyset, adLockPessimistic
rs.AddNew
rs!scno = Val(Text1)
rs!name = Text2
rs!address = Text3
```

```
rs!tariff = Combo1.Text
rs!lmu = Val(Text4)
rs!cmu = Val(Text5)
rs!units = Val(Text6)
rs!amt = Val(Text7)
rs!rdate = Format(Now, "dd-mmm-yy")
rs.Update
connect.Execute "commit"
MsgBox "New Record Added Successfully", vbInformation
Call clear
End Sub
Private Sub Command2_Click()
Dim n, units, amt As Integer
Dim name, tariff As String
Dim d As Date
n = InputBox("Enter Sc Number")
Set rs = New ADODB.Recordset
rs.Open "select * from eb27 where scno='" & Val(n) & "'", connect, adOpenKeyset, adLockPessimistic
If (rs.RecordCount) Then
name = rs!name
tariff = rs!tariff
units = rs!units
amt = rs!amt
d = rs!rdate
rs.Close
Set rs = New ADODB.Recordset
rs.Open "select * from ebbill27", connect, adOpenKeyset, adLockPessimistic
rs.AddNew
rs!scno = n
```

```
rs!name = name
rs!tariff = tariff
rs!units = units
rs!amt = amt
rs!rdate = Format(d, "dd-mmm-yy")
rs.Update
DataReport1.Refresh
DataReport1.Show
rs.Delete
DataReport1.Refresh
connect.Execute "commit"
rs.Close
Else
MsgBox "Invalid input", vbCritical
End If
End Sub

Private Sub Command3_Click()
n = InputBox("Enter SC Number")
Set rs = New ADODB.Recordset
rs.Open " select * from eb27 where scno='" & Val(n) & "' order by(rdate)", connect, adOpenKeyset,
adLockPessimistic

If (rs.RecordCount) Then
Text1 = rs!scno
Text2 = rs!name
Text3 = rs!address
```

```
Combo1.Text = rs!tariff
Text4 = rs!lmu
Text5 = rs!cmu
Text6 = rs!units
Text7 = rs!amt
MsgBox "Record viewed Successfully", vbInformation
Call clear
Else

MsgBox "Invalid Input", vbCritical
End If
End Sub



Private Sub Command4_Click()
End
End Sub



Private Sub Form_Load()
Label10.Caption = Format(Now, "dd-mmm-yy")
Combo1.AddItem ("Domestic")
Combo1.AddItem ("commercial")
Combo1.AddItem ("Others")
Set connect = New ADODB.Connection
connect.Open " Provider=MSDAORA.1;Password=tiger;User ID=scott;Data Source=dcslab;Persist
Security Info=True"

connect.CursorLocation = adUseClient

MsgBox "Connection Established Successfully", vbInformation
```

```
Call clear
Call Timer1_Timer
End Sub
Private Sub Text5_change()
Dim n As Integer
n = Val(Text5) - Val(Text4)
Text6 = n
If (Combo1.Text = "Domestic") Then
If (n > 0 And n <= 50) Then
Text7 = n * 1.1
ElseIf (n > 50 And n <= 100) Then
Text7 = n * 1.3
ElseIf (n > 100 And n <= 200) Then
Text7 = n * 2.6
ElseIf (n > 200 And n <= 600) Then
Text7 = n * 3.5
ElseIf (n > 600) Then
Text7  = n * 5.75
Else
Text7 = 40
End If
ElseIf (Combo1.Text = "commercial") Then
If (n > 0 And n <= 100) Then
Text7 = n * 4.3
ElseIf (n > 100 And n <= 200) Then
```

```
Text7 = n * 5.3
ElseIf (n > 200) Then
Text7 = n * 6.5
Else
Text7 = 80
End  If
Else
If (n > 0 And n <= 500) Then
Text7 = n * 1.4
ElseIf (n > 500 And n <= 1500) Then
Text7 = n * 2.25
ElseIf (n > 1500) Then
Text7 = n * 2.5
Else
Text7 = 160
End If
End If
End Sub
Public Sub clear()
Text1 = ""
Text2 = ""
Text3 = ""
Text4 = ""
Text5 = ""
Text6 = ""
Text7 = ""
```

```
Combo1.Text = ""
End Sub

Private Sub Timer1_Timer()
Label12.Caption = Time
End Sub
```

FORM DESIGN: