Q1. Create a database named 'Employee' and create a table 'Emp' with attributes 'ename', 'ecity', 'enumber', 'eaddress', 'deptname'.

A.

CREATE DATABASE EMPLOYEE;

USE DATABASE EMPLOYEE;

CREATE TABLE EMP(ENAME VARCHAR2(25) NOT NULL, ECITY VARCHAR2(25) NOT NULL, ENUMBER INTEGER(10) NOT NULL, EADDRESS VARCHAR2 (200) NOT NULL, DEPTNAME VARCHAR2(30) NOT NULL);

Q2. Create a table 'department' with attributes 'deptt_no', 'dcity', 'head', 'contact'.

A.

CREATE TABLE DPEARTMENT(DEPTT_ID INTEGER(6) NOT NULL, DCITY VARCHAR2(25) NOT NULL, HEAD VARCHAR2(25) NOT NULL, CONTACT INTEGER(10) NOT NULL);

Q3. Insert Data into tables using 'Insert' Command.

A.

INSERT INTO EMP(ENAME, ECITY, ENUMBER, EADDRESS, DEPTNAME) VALUES ("RAM SINGH", "AMRITSAR", 9876543210, "352, 3RD FLOOR MAYFAIR APPARTMENTS, G.T. ROAD, CHHEHARTA, AMRITSAR", "COMPUTER SCIENCE");

INSERT INTO DEPARTMENT(DEPTT_ID, DCITY, HEAD, CONTACT) VALUES(38, "AMRITSAR", "DR. HARDEEP SINGH", 9876543210);

Q4. To study command to modify the structure of table using Alter.

A.

ALTER TABLE DEPARTMENT ALTER COLUMN CONTACT INTEGER(12);

ALTER TABLE EMP ADD ESALARY INTEGER(6);

ALTER TABLE EMP ADD COLUMN EMPID INTEGER(6);

ALTER TABLE EMP DROP COLUMN EMPID;

Q5. To demonstrate various DML commands over above example.

A.

DML Commands: **SELECT, INSERT UPDATE DELETE.** Usage is as follows:

SELECT * FROM EMP;

SELECT HEAD, DCITY FROM DEPARTMENT;

INSERT INTO DEPARTMENT VALUES (34, "AMRITSAR", "DR. SANDEEP SHARMA", 9876543210);

UPDATE TABLE EMP SET ECITY="JALANDHAR" WHERE ENAME="RAM SINGH";

DELETE FROM DEPARTMENT WHERE DEPTT_ID=34;

Q6. Write and execute various queries to study the select command.

A.

SELECT * FROM DEPARTMENT;

SELECT ENAME, ECITY FROM EMP;

SELECT * FROM EMP WHERE ECITY="AMRITSAR";

Q7. Execute queries that execute commnads that involve compound conditions (and, or, in between, not between, like, not like etc.)

A.

SELECT ENAME, EADDRESS FROM EMP WHERE ECITY="AMRITSAR" AND ENAME="RAM SINGH";

SELECT ENAME, EADDRESS FROM EMP WHERE ECITY="AMRITSAR" OR ENAME="RAM SINGH";

SELECT ENAME, EADDRESS FROM EMP WHERE NOT ECITY="AMRITSAR" ;

SELECT * FROM DEPARTMENT WHERE DEPTT_ID BETWEEN 25 AND 66;

SELECT * FROM DEPARTMENT WHERE DEPTT_ID NOT BETWEEN 25 AND 66;

SELECT HEAD FROM DEPARTMENT WHERE HEAD LIKE "D%";

SELECT HEAD FROM DEPARTMENT WHERE HEAD NOT LIKE "%D";

Q8. To study character functions.

A.

The character functions are primarily used to manipulate strings. Following are some queries which describe and explain the major string or character functions.

1. **BIN(N)** – returns the binary string value of N, returns null if N is null.
    a. **SELECT BIN(12)** – returns '1100'.
2. **ASCII(C) –** returns the numeric value of leftmost character.
    a. **SELECT ASCII('Me')** – returns 01001101.
3. **CHAR_LENGTH(STR)** – returns the length of string.
    a. **SELECT CHAR_LENGTH('Hello World')** – returns 11.
4. **CONCAT(STR1,STR2.....)** – returns the single concatinated string.
    a. **SELECT CONCAT('Hello',' World')** – returns 'Hello World' as a single string.
5. **LOWER(STR)** – converts the string in lowercase letters.
    a. **SELECT LOWER('HELLO')** – returns 'hello'
6. **UPPER(STR) –** converts the string in uppercase letters.
    a. **SELECT UPPER('hello') –** returns 'HELLO'.
7. **LTRIM(STR) –** returns the string str with leading spaces removed.
    a. **SELECT LTRIM(' HELLO') –** returns 'HELLO'.
8. **RTRIM(STR) –** returns the string str with trailing spaces removed.
    a. **SELECT RTRIM('HELLO ') –** returns 'HELLO'.
9. **LPAD(STR,LEN,PADSTR) –** returns the string str, left-padded with the string padstr to a length of len characcters. If str is longer than len, the return value is shortened to len characters.
    a. **SELECT LPAD('HI',5,'X') –** returns 'XXXHI'.
10. **RPAD(STR,LEN,PADSTR) –** returns the string str, right-padded with the string padstr to a length of len characters. If str is longer than len, the return value is shortened to len characteres.
    a. **SELECT RPAD('HI',5,'X') –** returns 'HIXXX'.

Q9. To study the numeric functions.

A.

SQL numeric functions are primarily used for numeric manipulation or mathematical caluclations. Follwing are some queries which describe and explain the major numeric functions.

1. **ABS(X) –** returns the absolute value of number X.

    a. **SELECT ABS(-2) –** returns 2.
2. **CEIL(X) –** returns the smallest integer value that is not smaller than X.
    a. **SELECT CEIL(3.46) –** returns 4.
3. **FLOOR(X) –** returns the largest integer value that is not larger than X.
    a. **SELECT FLOOR(7.99) –** returns 7.
4. **LEAST(N1,N2,N3.....) –** returns the lowest number in the series or column.
    a. **SELECT LEAST(3,5,1,8,33) –** returns 1.
5. **GREATEST(N1,N2,N3....) –** returns the highest number in the series or column.
    a. **SELECT GREATEST(3,5,1,8,33) –** returns 33.

Besides this, we have all trignometric functions and logrithmic functions to perform mathematical caluclations which provide ease in Database programming.

Q10. Execute queries to study the aggregate functions (sum, count, max, min, average).

A.

SELECT SUM(ESAL) 'TOTAL SALARY GIVEN' FROM EMP;

SELECT COUNT(ENAME) AS 'TOTAL EMPLOYEES' FROM EMP;

SELECT MAX(ESAL) AS 'MAX SALARY' FROM EMP;

SELECT MIN(ESAL) AS 'MIN SALARY' FROM EMP;

SELECT * FROM EMP WHERE ESAL<AVERAGE(ESAL);

Q11. Execute queries to study the conversion functions.

A.

TO_CHAR TO_CHAR function is used to typecast a numeric or date input to character type with a format model (optional).

TO_NUMBER The TO_NUMBER function converts a character value to a numeric datatype. If the string being converted contains nonnumeric characters, the function returns an error.

Example Query:

SELECT TO_CHAR('03-02-1999' , 'MONTH DD, YYYY) AS DATE FROM DUAL;

SELECT TO_NUMBER('559') AS MYNUMBER FROM DUAL;

Q12. Execute queries to study the grouping commands(group by, order by , joins.).

A.

Consider that we have following tables with the following data for group by example.

ID	CITY
1	Amritsar
2	Amrtsar
3	Jalandhar

TableName: Customers

SELECT COUNT(ID) AS CUSTOMERS,CITY FROM IDTABLE GROUP BY(CITY);

Result:

CUSTOMERS	CITY
2	Amritsar
1	Jalandhar

Consider the following tables with the following data for order by example.

ID	CITY
1	Gurdaspur
2	Ambala
3	Jammu
4	Amritsar
5	Amritsar

TableName: Customers

SELECT * FROM CUSTOMERS ORDER BY (CITY)

Result:

ID	CITY
2	Ambala
4	Amritsar
5	Amritsar
1	Gurdaspur
3	Jammu

Consider the following tables with the following data for joins example.

ID	NAME	DEPT
1	SURAJ	1
2	GEETA	1
3	PAYAL	2
4	SUNIL	4

TableName: EMP

DEPT	DNAME
1	COMPUTER SCIENCE
2	ELECTRONICS
3	COMMERCE
4	PHYSICS

TableName: DEPTT

SELECT B1.NAME, B2.DNAME FROM EMP AS B1 JOINS DEPTT AS B2 ON B1.DEPT=B2.DEPT

Result:

NAME	DNAME
SURAJ	COMPUTER SCIENCE
GEETA	COMPUTER SCIENCE
PAYAL	ELECTRONICS
SUNIL	PHYSICS

Q13. Execute queries to demonstrate the use of indexes.

A.

Indexes are used to retrieve data from the database very fast. The users cannot see the indexes, they are just used to speed up searches/queries.

CREATE INDEX MYINDEX ON EMP (NAME) /* DUPLICATE VALUES ALLOWED*/

CREATE UNIQUE INDEX MYINDEX ON EMP (NAME) /* DUPLICATE VALUES NOT ALLOWED*/

DROP INDEX MYINDEX /*DROPS THE INDEX WITH INDEX NAME MYINDEX*/

Q14. Execute queries to demonstrate the use of sequences.

A.

A sequence is a set of integers 1, 2, 3, ... that are generated in order on demand. Sequences are frequently used in databases because many applications require each row in a table to contain a unique value and sequences provide an easy way to generate them.

CREATE SEQUENCE SEQ_X MINVALUE 1 START WITH 1 INCREMENT BY 1 CACHE 10;

The code above creates a sequence object called **SEQ_X**, that starts with 1 and will increment by 1. It will also cache up to 10 values for performance. The cache option specifies how many sequence values will be stored in memory for faster access.

To insert a new record into the "**CUSTOMERS**" table, we will have to use the nextval function (this function retrieves the next value from **SEQ_X** sequence):

INSERT INTO CUSTOMERS (SEQ_X.NEXTVAL, 'PARIS');

Q15. Create a PL/SQL block to demonstrate use of control structures.

A.

```
DECLARE
   i number(1);
   j number(1);
BEGIN
   << outer_loop >>
   FOR i IN 1..3 LOOP
      << inner_loop >>
      FOR j IN 1..3 LOOP
         dbms_output.put_line('i is: '|| i || ' and j is: ' || j);
      END loop inner_loop;
   END loop outer_loop;
END;
```

Q16. Create a PL/SQL block to demonstrate use of cursors.

A.

Here are some predefined cursors available in PL/SQL these cursors are also known as implicit cursors.

S.No	Attribute & Description
1	**%FOUND** Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
2	**%NOTFOUND** The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
3	**%ISOPEN** Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
4	**%ROWCOUNT** Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

PL/SQL Block for cursors.

```
DECLARE
   total_rows number(2);
BEGIN
   UPDATE customers
   SET salary = salary + 500;
   IF sql%notfound THEN
      dbms_output.put_line('no customers selected');
   ELSIF sql%found THEN
      total_rows := sql%rowcount;
      dbms_output.put_line( total_rows || ' customers selected ');
   END IF;
END;
```

Q17. Create a PL/SQL block to demonstrate Procedures.

A.

The main difference between PL/SQL procedures and functions is that procedures do not return any value but functions do return some values back.

```
CREATE OR REPLACE PROCEDURE greetings
AS
BEGIN
   dbms_output.put_line('Hello World!');
END;
```

Q18. Create a PL/SQL block to demonstrate triggers.

A.

Here is one example of triggers.

```
CREATE OR REPLACE TRIGGER display_salary_changes

BEFORE DELETE OR INSERT OR UPDATE ON customers

FOR EACH ROW

WHEN (NEW.ID > 0)

DECLARE

   sal_diff number;

BEGIN

   sal_diff := :NEW.salary  - :OLD.salary;

   dbms_output.put_line('Old salary: ' || :OLD.salary);

   dbms_output.put_line('New salary: ' || :NEW.salary);

   dbms_output.put_line('Salary difference: ' || sal_diff);

END;
```

Q19. Create a PL/SQL block to demonstrate the use of exception handling.

A.

A brief example of exception handling.

```
DECLARE
   c_id customers.id%type := 8;
   c_name customers.Name%type;
   c_addr customers.address%type;
BEGIN
   SELECT  name, address INTO  c_name, c_addr
   FROM customers
   WHERE id = c_id;
   DBMS_OUTPUT.PUT_LINE ('Name: '||  c_name);
   DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);

EXCEPTION
   WHEN no_data_found THEN
      dbms_output.put_line('No such customer!');
   WHEN others THEN
      dbms_output.put_line('Error!');
END;
```