GALGOTIAS UNIVERSITY

Data Base and management System

BCSE2073

DETAILS

Name: Piyush Anand

Sec – 5

Batch – P2

Admission No.: 20SCSE1010375

Submitted to: Dr. Basetty Mallikarjuna

INDEX

S.No	Experiment Name
1.	Design ER diagrams for various scenarios or based on given projects.
2.	Implement DDL Statements and DML statements.
3.	Execute the SELECT command with different clauses.
4.	Execute various types of Integrity Constraints on database.
5.	Implement SINGLE ROW functions (Character, Numeric, Date functions) and GROUP functions (avg, count, max, min, sum).
6.	Implement the concept of grouping of Data and Sub-queries.
7.	Analysis and design of the normalized tables.
8.	Execute the concept of Data Control Language (DCL).
9.	Implement Transaction Control Language (TCL).
10.	Implement Simple and Complex View.
11.	Write a PL/SQL block to satisfy some conditions by accepting input from the user.
12.	Write a PL/SQL block for greatest of three numbers using IF AND ELSEIF.
13.	Write a PL/SQL block for summation of odd numbers using for LOOP.
14.	Write a PL/SQL Procedure for GCD Numbers
15.	Write a PL/SQL Procedure for cursor implementation
16.	Write a PL/SQL block to implementation of factorial using function.

Experiment – 1

Aim - Design ER diagrams for various scenarios or based on given projects.

Theory - ER Model stands for Entity Relationship Model is a high-level conceptual data model diagram. ER model helps to systematically analyze data requirements to produce a well-designed database. The ER Model represents real-world entities and the relationships between them. Creating an ER Model in DBMS is considered as a best practice before implementing your database.

Components of the ER Diagram

This model is based on three basic concepts:

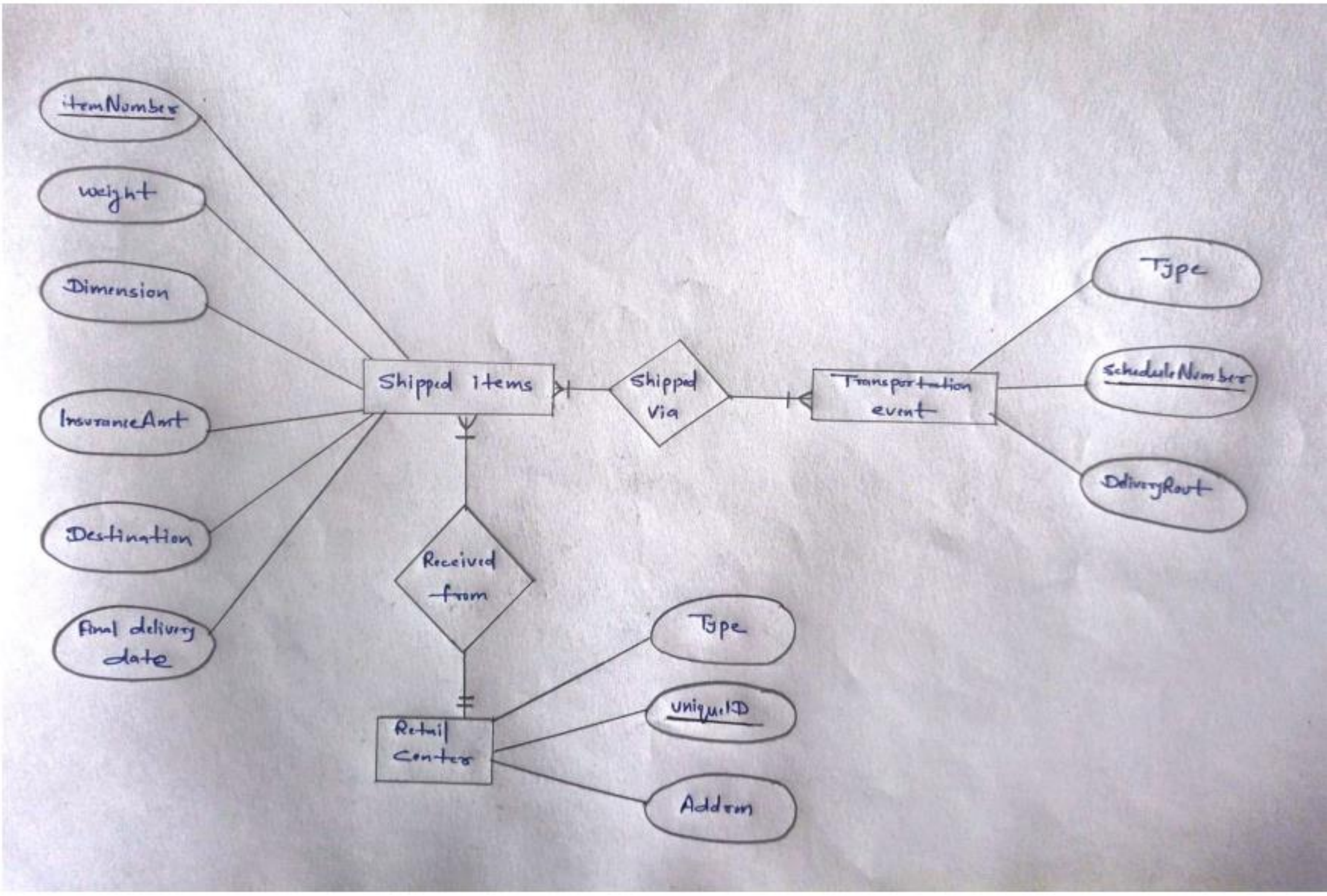
- Entities
- Attributes
- Relationships

EXAM QUESTIONS AS SCENARIOS

UPS prides itself on having up-to-date information on the processing and current location of each shipped item. To do this, UPS relies on a company-wide information system. Shipped items are the heart of the UPS product tracking information system. Shipped items can be characterized by item number (unique), weight, dimensions, insurance amount, destination, and final delivery date. Shipped items are received into the UPS system at a single retail center. Retail centers are characterized by their type, uniqueID, and address. Shipped items make their way to their destination via one or more standard UPS transportation events (i.e., flights, truck deliveries). These transportation events are characterized by a unique scheduleNumber, a type (e.g, flight, truck), and a deliveryRoute.

Please create an Entity Relationship diagram that captures this information about the UPS system. Be certain to indicate identifiers and cardinality constraints.

E-R diagram –


itemNumber
weight
Dimension
InsuranceAmt
Destination
Final delivery date
Shipped Items
Shipped Via
Transportation event
Type
ScheduleNumber
DeliveryRout
Received from
Retail Center
Type
uniqueID
Address

Experiment – 2

Aim - Implement DDL Statements and DML statements.

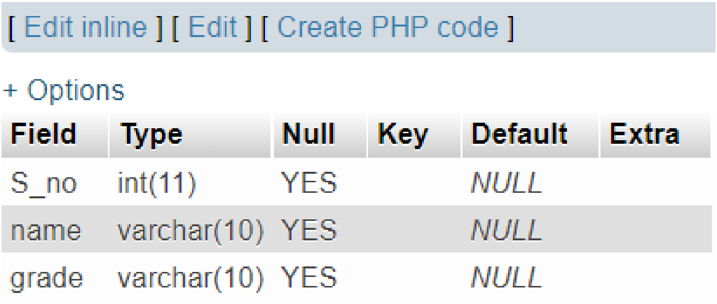
Theory – The SQL DDL allows specification of not only a set of relations but also information about each relation, including-

- Schema for each relation
- The domain of values associated with each attribute.
- The integrity constraints.
- The set of indices to be maintained for each relation.
- The security and authorization information for each relation.

1. CREATE TABLE

Create table student(S_no int, name varchar(10), grade varchar(10));

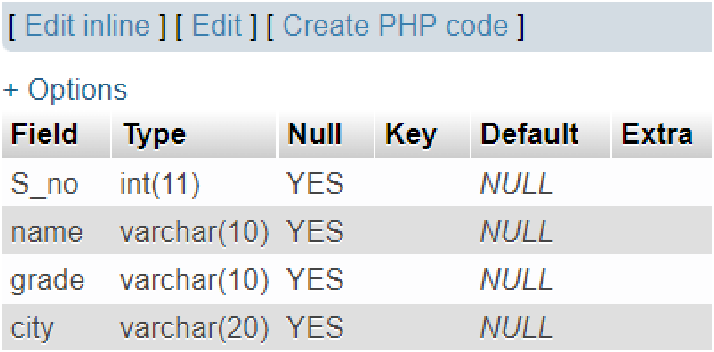
Desc student; (SHOW TABLE)



Field	Type	Null	Key	Default	Extra
S_no	int(11)	YES		*NULL*	
name	varchar(10)	YES		*NULL*	
grade	varchar(10)	YES		*NULL*	

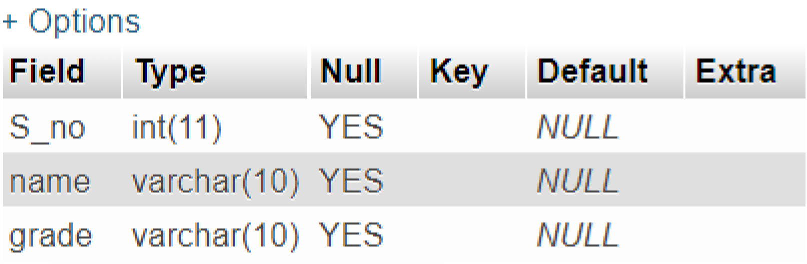
2. ALTER TABLE

a. alter table student add city varchar(20);



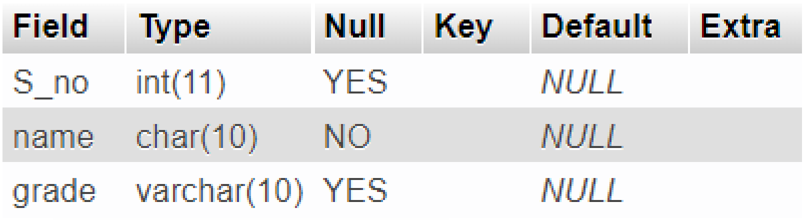
Field	Type	Null	Key	Default	Extra
S_no	int(11)	YES		*NULL*	
name	varchar(10)	YES		*NULL*	
grade	varchar(10)	YES		*NULL*	
city	varchar(20)	YES		*NULL*	

b. alter TABLE student drop city;



Field	Type	Null	Key	Default	Extra
S_no	int(11)	YES		*NULL*	
name	varchar(10)	YES		*NULL*	
grade	varchar(10)	YES		*NULL*	

c. alter TABLE student MODIFY name char(10) not null;

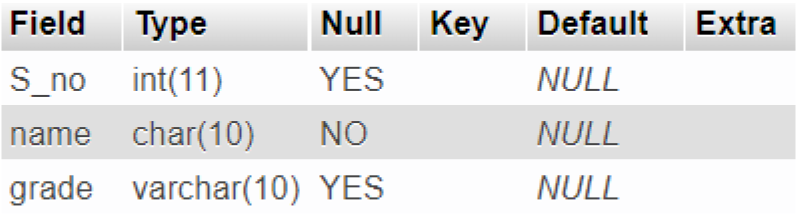


Field	Type	Null	Key	Default	Extra
S_no	int(11)	YES		*NULL*	
name	char(10)	NO		*NULL*	
grade	varchar(10)	YES		*NULL*	

3. RENAME COMMAND

Alter table student RENAME to student_gu;

Desc student_gu;

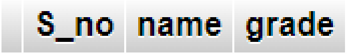


Field	Type	Null	Key	Default	Extra
S_no	int(11)	YES		*NULL*	
name	char(10)	NO		*NULL*	
grade	varchar(10)	YES		*NULL*	

4. TRUNCATE COMMAND

TRUNCATE student_gu;

SELECT * from student_gu;

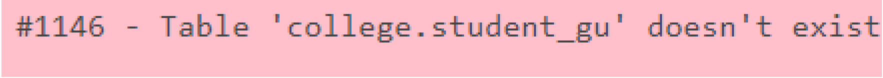


	S_no	name	grade

5. DROP COMMAND

DROP TABLE student_gu;

Select * from student_gu;



```
#1146 - Table 'college.student_gu' doesn't exist
```

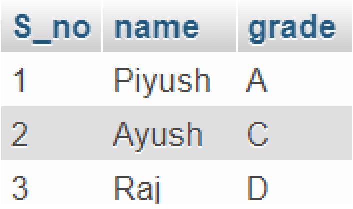
DML COMMANDS

1. INSERT COMMAND

INSERT into student VALUE(1, "Piyush", "A");

INSERT into student VALUE(2, "Ayush", "C");

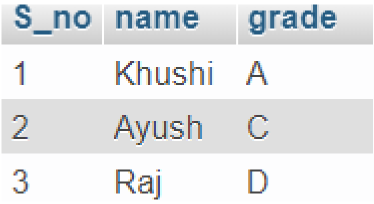
insert INTO student VALUES(3, "Raj", "D");



S_no	name	grade
1	Piyush	A
2	Ayush	C
3	Raj	D

2. UPDATE COMMAND

```
update student set name = "Khushi" where S_no = 1;
```



S_no	name	grade
1	Khushi	A
2	Ayush	C
3	Raj	D

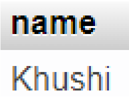
Experiment – 3

Aim - Execute the SELECT command with different clauses.

Theory - The most commonly used SQL command is SELECT statement. SQL SELECT statement is used to query or retrieve data from a table in the database. A query may retrieve information from specified columns or from all of the columns in the table. To create a simple SQL SELECT Statement, you must specify the column(s) name and the table name. The whole query is called SQL SELECT Statement.

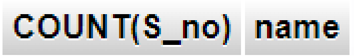
Syntax –

1.SELECT name from student WHERE grade = "A";



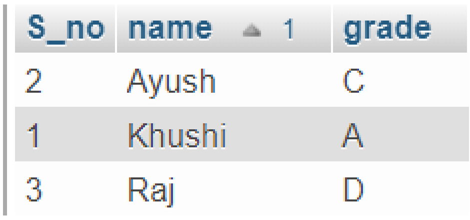
name
Khushi

2. SELECT COUNT(S_no), name FROM student GROUP BY name HAVING COUNT(S_no) > 1;



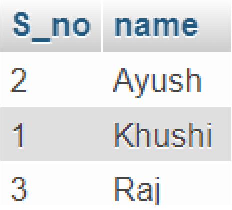
COUNT(S_no)	name

3. SELECT * from student ORDER BY name;



S_no	name ▲ 1	grade
2	Ayush	C
1	Khushi	A
3	Raj	D

4. SELECT S_no, name FROM student GROUP BY name;



S_no	name
2	Ayush
1	Khushi
3	Raj

Experiment – 4

Aim - Execute various types of Integrity Constraints on database.

Theory –

SQL 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. Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).

In SQL, we have the following constraints:

- NOT NULL - Indicates that a column cannot store NULL value
- UNIQUE - Ensures that each row for a column must have a unique value
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have a unique identity which helps to find a particular record in a table more easily and quickly
- FOREIGN KEY - Ensure the referential integrity of the data in one table to match values in another table
- CHECK - Ensures that the value in a column meets a specific condition
- DEFAULT - Specifies a default value for a column

The PRIMARY KEY constraint uniquely identifies each record in a database table. Primary keys must contain UNIQUE values. A primary key column cannot contain NULL values. Most tables should have a primary key, and each table can have only ONE primary key.

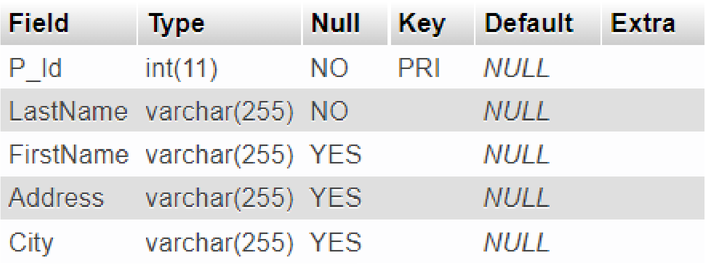
A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

Syntax –

CREATE TABLE table_name (

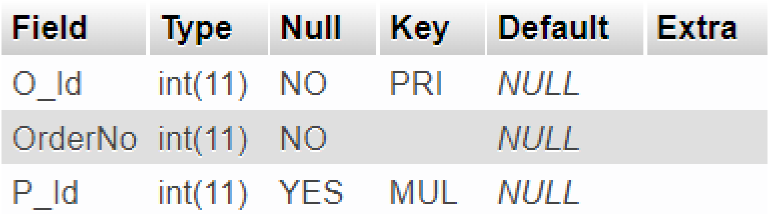
column_name1 data_type(size) constraint_name);

```
CREATE TABLE Persons

(

P_Id int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Address varchar(255),

City varchar(255),

PRIMARY KEY (P_Id)

);
```



Field	Type	Null	Key	Default	Extra
P_Id	int(11)	NO	PRI	*NULL*	
LastName	varchar(255)	NO		*NULL*	
FirstName	varchar(255)	YES		*NULL*	
Address	varchar(255)	YES		*NULL*	
City	varchar(255)	YES		*NULL*	

```
CREATE TABLE Orders

(

O_Id int NOT NULL,

OrderNo int NOT NULL,

P_Id int,

PRIMARY KEY (O_Id),

FOREIGN KEY (P_Id) REFERENCES Persons(P_Id)

);
```



Field	Type	Null	Key	Default	Extra
O_Id	int(11)	NO	PRI	*NULL*	
OrderNo	int(11)	NO		*NULL*	
P_Id	int(11)	YES	MUL	*NULL*	

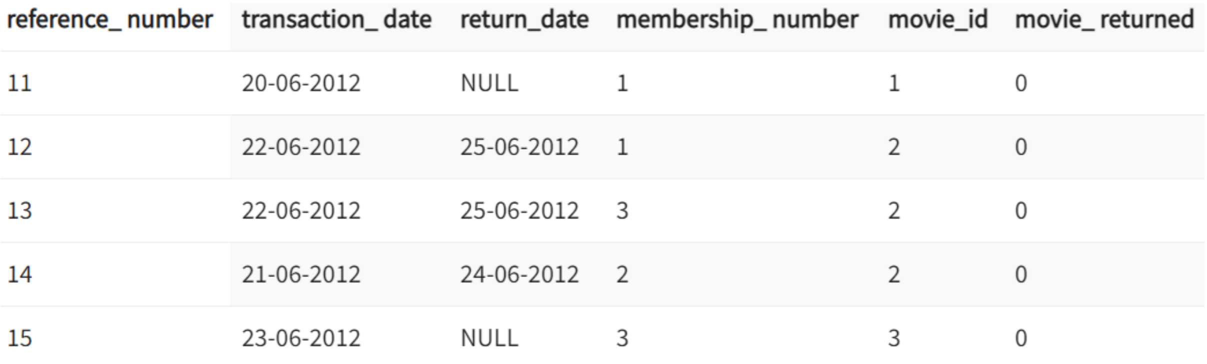
Experiment – 5

Aim - Implement SINGLE ROW functions (Character, Numeric, Date functions) and GROUP functions (avg, count, max, min, sum).

Theory –

Aggregate functions perform a variety of actions such as counting all the rows in a table, averaging a column's data, and summing numeric data. Aggregates can also search a table to find the highest "MAX" or lowest "MIN" values in a column. As with other types of queries, you can restrict, or filter out the rows these functions act on with the WHERE clause. For example, if a manager needs to know how many employees work in an organization, the aggregate function named COUNT(*) can be used to produce this information. The COUNT(*) function shown in the below SELECT statement counts all rows in a table.

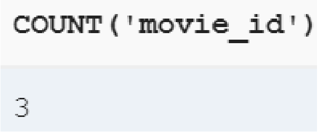
Table – movierental



reference_ number	transaction_ date	return_date	membership_ number	movie_id	movie_ returned
11	20-06-2012	NULL	1	1	0
12	22-06-2012	25-06-2012	1	2	0
13	22-06-2012	25-06-2012	3	2	0
14	21-06-2012	24-06-2012	2	2	0
15	23-06-2012	NULL	3	3	0

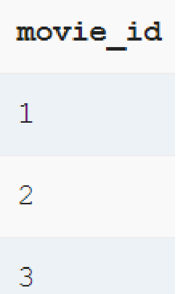
Syntax –

1. SELECT COUNT(`movie_id`) FROM `movierentals` WHERE `movie_id` = 2;



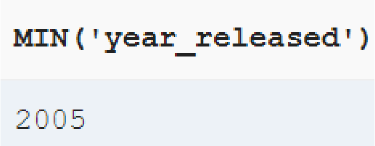
COUNT('movie_id')
3

2. SELECT DISTINCT `movie_id` FROM `movierentals`;



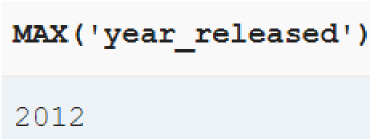
movie_id
1
2
3

3. SELECT MIN(`year_released`) FROM `movies`;



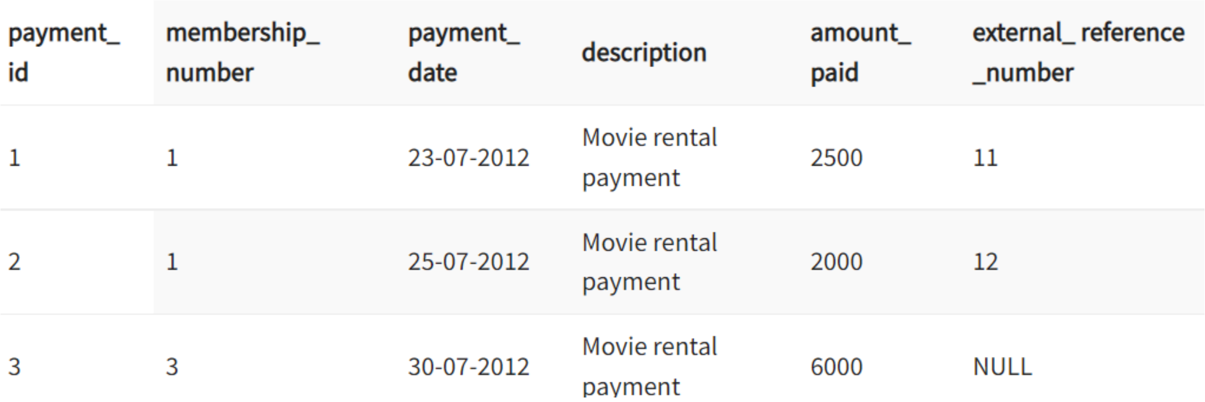
MIN('year_released')
2005

4. SELECT MAX(`year_released`) FROM `movies`;



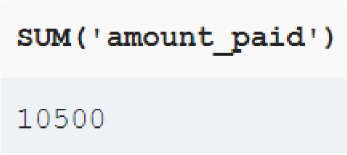
```
MAX('year_released')
2012
```

Table – payment



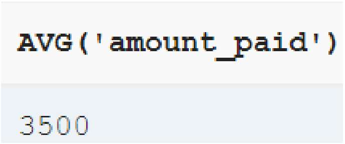
payment_ id	membership_ number	payment_ date	description	amount_ paid	external_ reference _number
1	1	23-07-2012	Movie rental payment	2500	11
2	1	25-07-2012	Movie rental payment	2000	12
3	3	30-07-2012	Movie rental payment	6000	NULL

5. SELECT SUM(`amount_paid`) FROM `payments`;



```
SUM('amount_paid')
10500
```

6. SELECT AVG(`amount_paid`) FROM `payments`;



```
AVG('amount_paid')
3500
```

Experiment – 6

Aim - Implement the concept of grouping of Data and Sub-queries.

Theory –

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause. A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

Subquery syntax –

```
 SELECT column_name [, column_name ]

FROM   table1 [, table2 ]

WHERE  column_name OPERATOR

  (SELECT column_name [, column_name ]

  FROM table1 [, table2 ]

  [WHERE])
```

The GROUP BY Statement in SQL is used to arrange identical data into groups with the help of some functions. i.e if a particular column has same values in different rows then it will arrange these rows in a group.

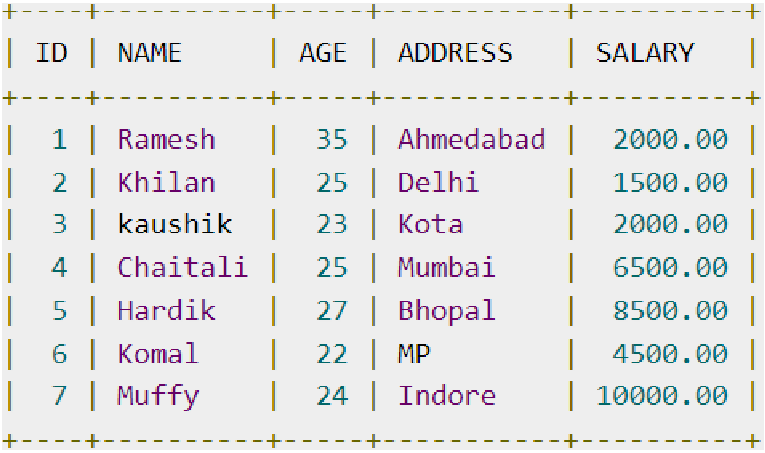
Important points –

- GROUP BY clause is used with the SELECT statement.
- In the query, GROUP BY clause is placed after the WHERE clause.
- In the query, GROUP BY clause is placed before ORDER BY clause if used any.

Group By syntax –

```
SELECT column1, function_name(column2)

FROM table_name

WHERE condition

GROUP BY column1, column2

ORDER BY column1, column2;
```

Table – CUSTOMERS



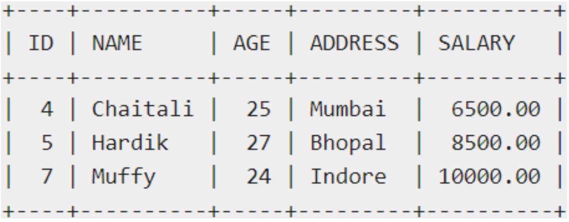
```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

SQL > SELECT * FROM CUSTOMERS

WHERE ID IN

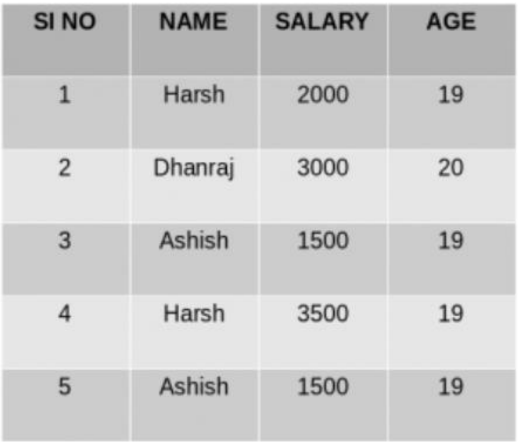
(SELECT ID FROM CUSTOMERS

WHERE SALARY > 4500) ;



```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  5 | Hardik   |  27 | Bhopal  |  8500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

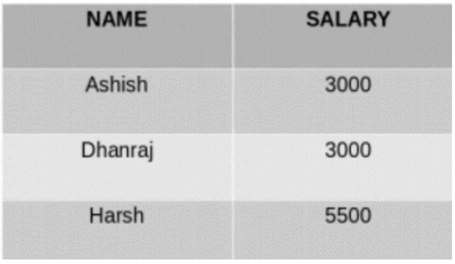
Table – Employee



SI NO	NAME	SALARY	AGE
1	Harsh	2000	19
2	Dhanraj	3000	20
3	Ashish	1500	19
4	Harsh	3500	19
5	Ashish	1500	19

SQL > SELECT NAME, SUM(SALARY) FROM Employee

GROUP BY NAME;



NAME	SALARY
Ashish	3000
Dhanraj	3000
Harsh	5500

Experiment – 7

Aim - Analysis and design of the normalized tables.

Theory –

Normalization defines as following –

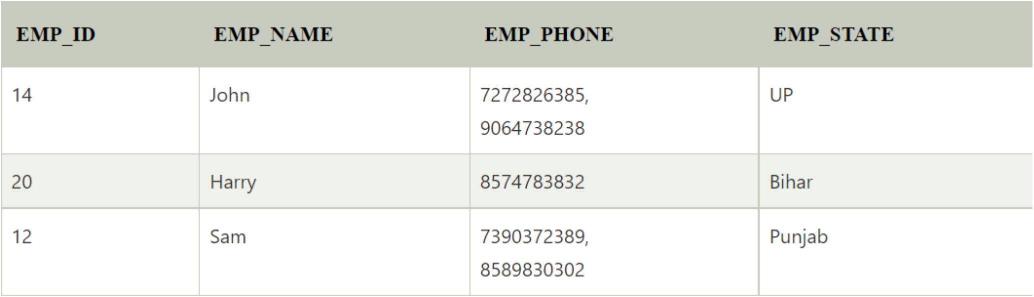
- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.

Types of Normal Forms –

First Normal Form (1 NF) – A relation is in 1NF if it contains an atomic value.

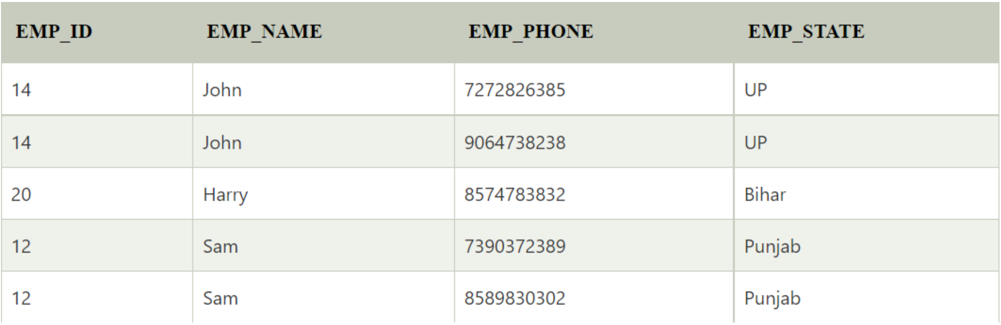
- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

EMPLOYEE table:



EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF

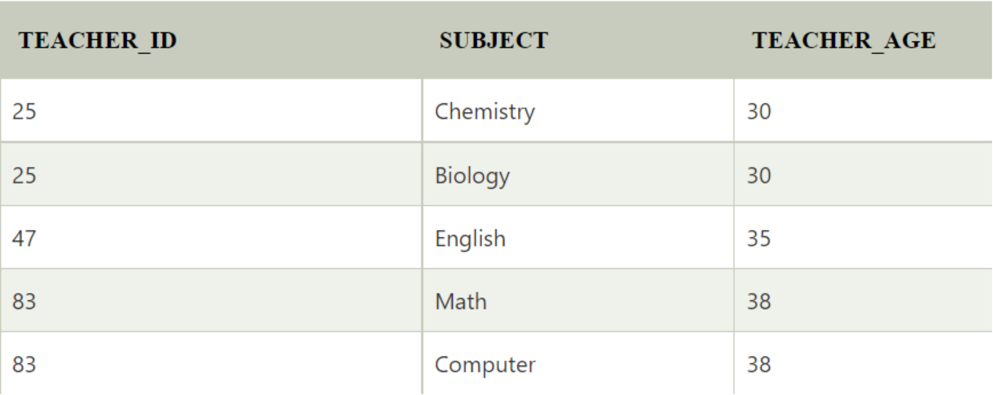


EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF) - A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.

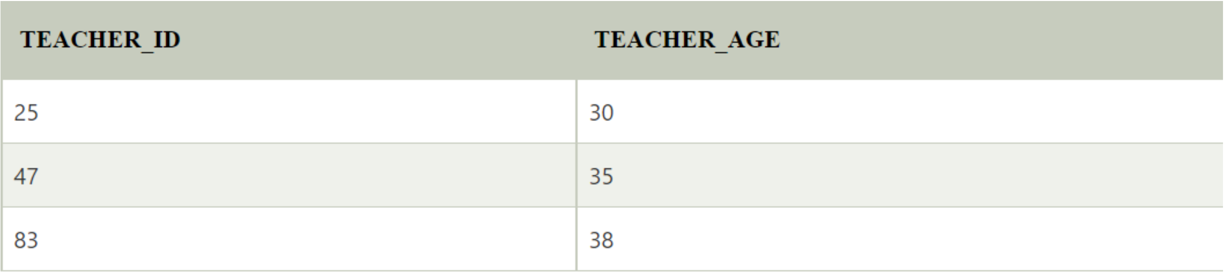
- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Table – Teacher

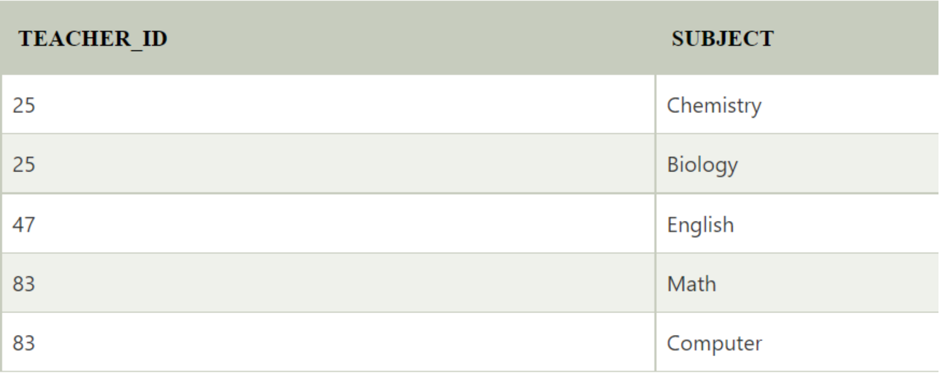


TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

To convert the given table into 2NF, we have to decompose it into two tables



TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

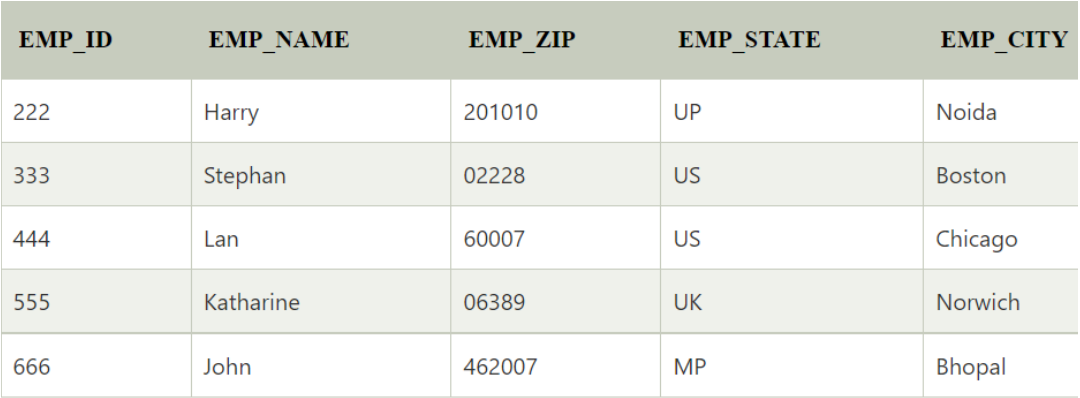


TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Third Normal Form (3NF) – A relation will be in 3NF if it is in 2NF and no transition dependency exists.

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

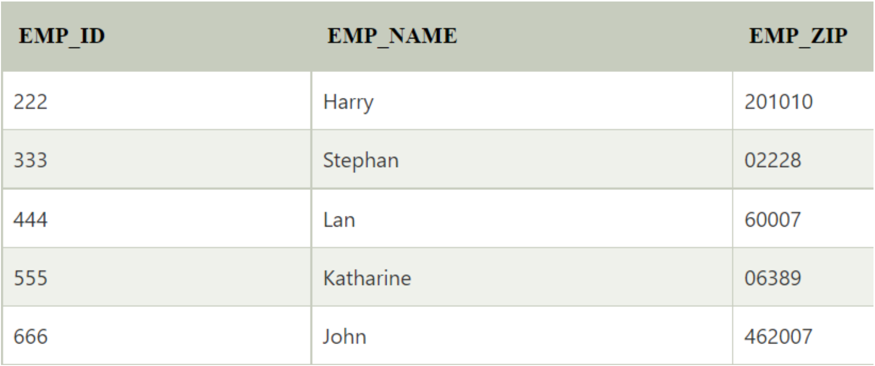
Table – Employee_detail



EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

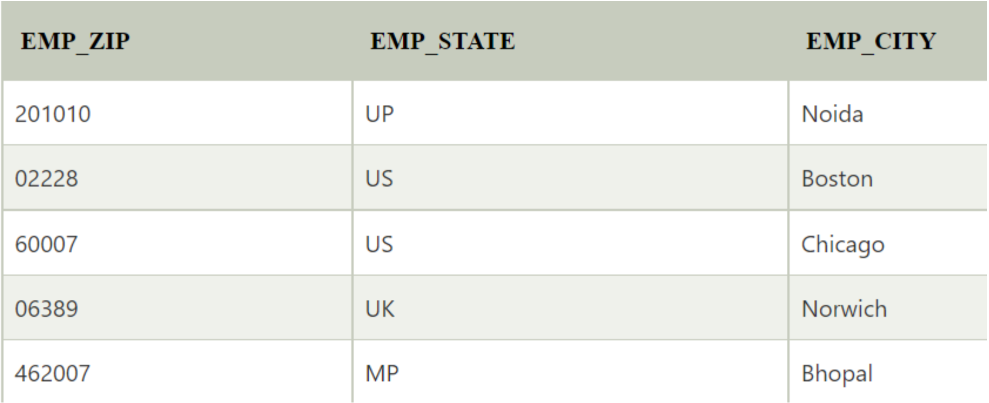
To convert it into 3NF we have to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

Employee table –



EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

Employee_ZIP –



EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Experiment – 8

Aim - Execute the concept of Data Control Language (DCL).

Theory –

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

List of DCL commands:

- GRANT: This command gives users access privileges to the database.

  Syntax –

  ```
  GRANT privilege_name
  ON object_name
  TO

  {
      user_name|PUBLIC|role_name
  }

  [WITH GRANT OPTION];
  ```

- REVOKE: This command withdraws the user's access privileges given by using the GRANT command.

  Syntax –

  ```
  REVOKE privilege_name
  ON object_name

  FROM
  {
      user_name|PUBLIC|role_name
  }
  ```

1. GRANT ALL ON employee TO ABC;

2. REVOKE UPDATE ON employee FROM ABC;

Experiment – 9

Aim - Implement Transaction Control Language (TCL).

Theory –

Transaction Control Language commands are used to manage transactions in the database.

Transactional control commands are used only DML Commands such as INSERT, UPDATE and DELETE.

COMMIT, ROLLBACK and SAVEPOINT are the TCL commands used in SQL.

Commit - This command is used to save all the transactions to the database.

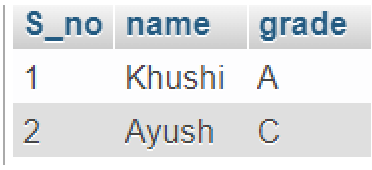
Syntax –

```
DELETE FROM Students
WHERE RollNo =25;
COMMIT;
```

Rollback - Rollback command allows you to undo transactions that have not already been saved to the database.

Syntax –

```
DELETE from student WHERE S_no =3;
```



S_no	name	grade
1	Khushi	A
2	Ayush	C

Savepoint – This command helps you to sets a savepoint within a transaction.

Syntax –

```
SAVEPOINT RollNo;
```

Experiment – 10

Aim – Implement Simple and Complex View.

Theory –

Views are a special version of tables in SQL. They provide a virtual table environment for various complex operations. You can select data from multiple tables, or you can select specific data based on certain criteria in views. It does not hold the actual data; it holds only the definition of the view in the data dictionary.

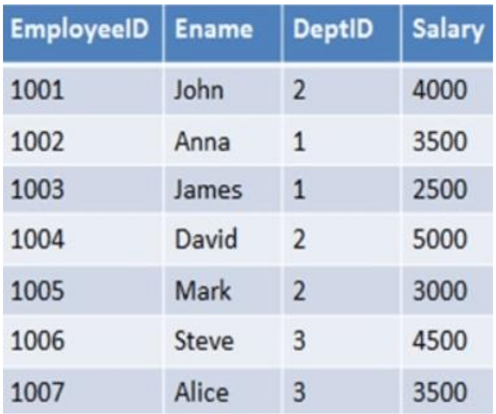
1. Simple View –

A view based on the only a single table, which doesn't contain GROUP BY clause and any functions.

2. Complex view –

A view based on multiple tables, which contain GROUP BY clause and functions.

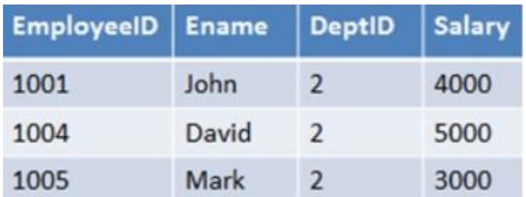
Table – Employee



EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

SQL >

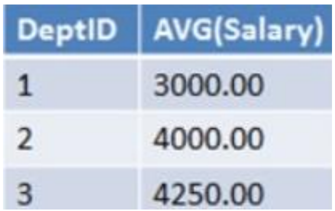
```
CREATE view emp_view AS

SELECT EmployeeID, Ename

FROM Emplyee

WHERE DeptID = 2;
```



EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1004	David	2	5000
1005	Mark	2	3000

SQL >

```
CREATE view emp_view AS

SELECT DeptID, Avg(Salary)

FROM Emplyee

GROUP BY DeptID;
```



DeptID	AVG(Salary)
1	3000.00
2	4000.00
3	4250.00

Experiment – 11

Aim - Write a PL/SQL block to satisfy some conditions by accepting input from the user.

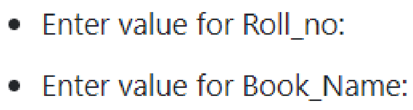
Theory –

PL/SQL is a procedural language designed specifically to embrace SQL statements within its syntax. PL/SQL program units are compiled by the Oracle Database server and stored inside the database. And at run-time, both PL/SQL and SQL run within the same server process, bringing optimal efficiency. PL/SQL automatically inherits the robustness, security, and portability of the Oracle Database.

Code –

```
Declare
  Roll_no  number;
  B_name varchar(20);
Begin
  Roll_no := &Roll_no;
  B_name := '&Book_Name';
  input(Roll_no,B_name);
end;
/
```

Output –



- Enter value for Roll_no:
- Enter value for Book_Name:

Experiment – 12

Aim - Write a PL/SQL block for greatest of three numbers using IF AND ELSEIF.

Theory –

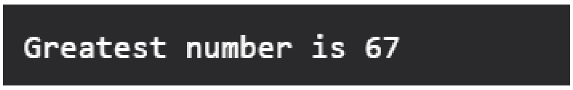
In PL/SQL code groups of commands are arranged within a block. A block group related declarations or statements. In declare part, we declare variables and between begin and end part, we perform the operations.

Given three numbers and the task is to find greatest among them.

Code –

```
--To find the greatest number
DECLARE
        a NUMBER := 46;
        b NUMBER := 67;
        c NUMBER := 21;
BEGIN
        IF a > b AND a > c THEN
        dbms_output.Put_line('Greatest number is ' ||a);
        ELSIF b > a AND b > c THEN
        dbms_output.Put_line('Greatest number is ' ||b);
        ELSE
        dbms_output.Put_line('Greatest number is ' ||c);
        END IF;
END;
```

Output –



```
Greatest number is 67
```

Experiment – 13

Aim - Write a PL/SQL block for summation of odd numbers using for LOOP.

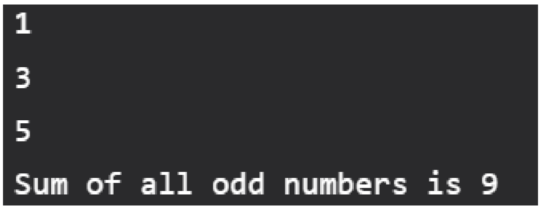
Theory –

In PL/SQL code groups of commands are arranged within a block. A block group related declarations or statements. In declare part, we declare variables and between begin and end part, we perform the operations.

Code –

```
-- display all odd number from 1 to n
DECLARE
        -- declare variable num
        num NUMBER(3) := 1;
        sum1 NUMBER(4) := 0;
BEGIN
        WHILE num <= 5 LOOP
                dbms_output.Put_line(num);
                sum1 := sum1 + num;
                num := num + 2;
        END LOOP;
           dbms_output.Put_line('Sum of all odd numbers is '|| sum1);
END;
```

Output –



```
1
3
5
Sum of all odd numbers is 9
```

Experiment – 14

Aim – Write a PL/SQL Procedure for GCD Numbers

Theory –

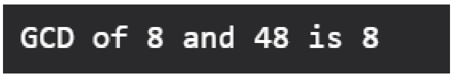
In PL/SQL code groups of commands are arranged within a block. A block group related declarations or statements. In declare part, we declare variables and between begin and end part, we perform the operations.

Given two numbers and task is to find the GCD (Greatest Common Divisor) or HCF (Highest Common Factor) value of the numbers.

Code –

```
DECLARE
        -- declare variable num1, num2 and t
        num1 INTEGER;
        num2 INTEGER;
        t INTEGER;
BEGIN
        num1 := 8;
        num2 := 48;
        WHILE MOD(num2, num1) != 0 LOOP
                t := MOD(num2, num1);
                num2 := num1;
                num1 := t;
        END LOOP;
        dbms_output.Put_line('GCD of ' ||num1 ||' and ' ||num2 ||' is ' ||num1);
END;
```

Output –



```
GCD of 8 and 48 is 8
```

Experiment - 15

Aim – Write a PL/SQL Procedure for cursor implementation.

Theory –

When an SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. It contains all information needed for processing the statement. In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it.

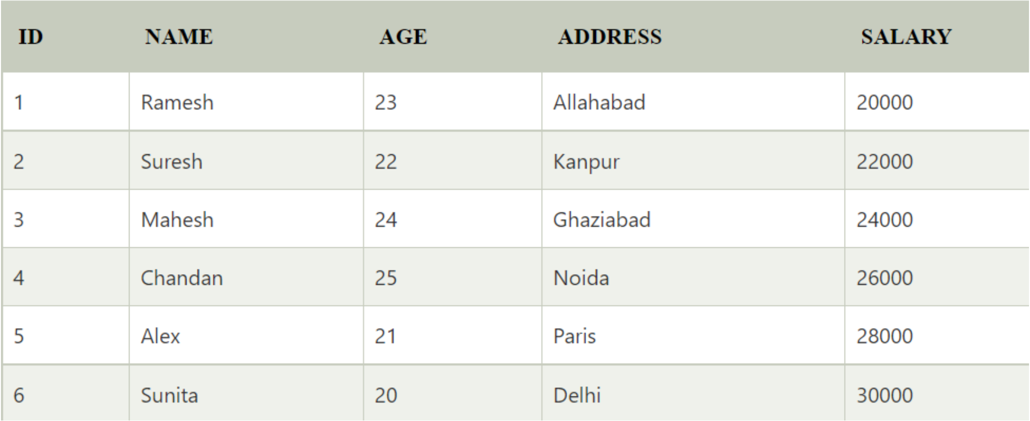
A cursor is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors:

- Implicit Cursors
- Explicit Cursors

Cursor actions-

- Declare Cursor: A cursor is declared by defining the SQL statement that returns a result set.
- Open: A Cursor is opened and populated by executing the SQL statement defined by the cursor.
- Fetch: When the cursor is opened, rows can be fetched from the cursor one by one or in a block to perform data manipulation.
- Close: After data manipulation, close the cursor explicitly.
- Deallocate: Finally, delete the cursor definition and release all the system resources associated with the cursor.

Table – customers

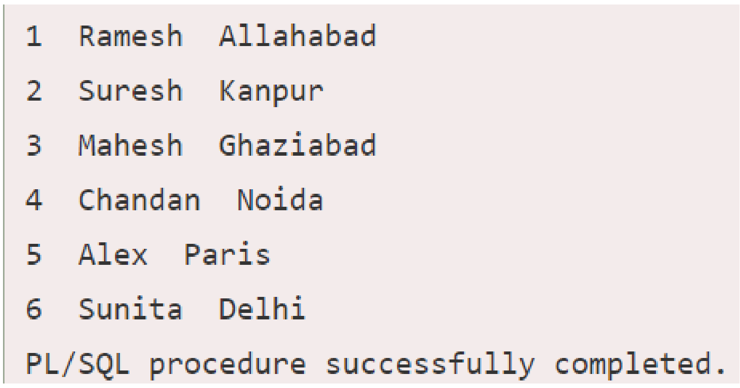


ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Code –

```
DECLARE
  c_id customers.id%type;
  c_name customers.name%type;
  c_addr customers.address%type;
  CURSOR c_customers is
    SELECT id, name, address FROM customers;
BEGIN
  OPEN c_customers;
  LOOP
    FETCH c_customers into c_id, c_name, c_addr;
    EXIT WHEN c_customers%notfound;
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
  END LOOP;
  CLOSE c_customers;
END;
/
```

Output –



```
1  Ramesh  Allahabad
2  Suresh  Kanpur
3  Mahesh  Ghaziabad
4  Chandan  Noida
5  Alex  Paris
6  Sunita  Delhi
PL/SQL procedure successfully completed.
```

Experiment – 16

Aim – Write a PL/SQL block to implementation of factorial using function.

Theory –

Factorial number: The factorial of a non-negative integer 'n' is denoted by n!. It is the product of all positive integers less then or equal to 'n'.

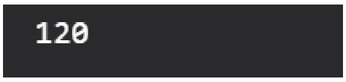
For example:

5! = 5*4*3*2*1 = 120

Code –

```
Declare
     fac number :=1;
     n number := &1;
begin
    while n > 0 loop
    fac:=n*fac;
    n:=n-1;
    end loop;
    dbms_output.put_line(fac);
end;
```

Output –



```
120
```