Government Girls' Polytechnic, Bilaspur

Class: IT 6th sem
Name of the Lab: Networking Lab

Title of the Practical : **Client server lab**

Teachers Assessment: 20 End Semester Examination: 50

EXPERIMENT NO:-1

1. **OBJECTIVE** :- To create a table named customer using CREATE command having First_Name
   Last_Name,Address , City ,Country ,Birth_Date in SQL**.**

2. **HARDWARE & SYSTEM SOFTWARE REQUIRED :-Windows Xp and later version.**

3. **SOFTWARE REQUIRED :- Oracle 8i.**

4. **THEORY :-** Tables are the basic structure where data is stored in the database. Given that in most cases, there is no way for the database vendor to know ahead of time what your data storage needs are, chances are that you will need to create tables in the database yourself. Many database tools allow you to create tables without writing SQL, but given that tables are the container of all the data, it is important to include the **CREATE TABLE** syntax.

   The SQL syntax for **CREATE TABLE** is

   CREATE TABLE "table_name"
   ("column 1" "data_type_for_column_1",
   "column 2" "data_type_for_column_2",
   ...)

5. **CODING (IF REQUIRED) :-**

CREATE TABLE customer(First_Name char(50),Last_Name char(50),Address char(50),City char(50),Country char(25),Birth_Date date)

6. **PROGRAM INPUTS & OUTPUT :-Input to this code is written in coding statement . Output will be a table look like this given below.**

First_Name	Last_Name	Address	City	Country	Birth_Date

7. **OBSERVATIONS :-** This command to create table will only make the structure of the table.

EXPERIMENT NO:- 2

1. **OBJECTIVE** :- To insert values into the table named Store_Information using INSERT INTO command.

2. **HARDWARE & SYSTEM SOFTWARE REQUIRED :-Windows Xp and later version.**

3. **SOFTWARE REQUIRED :- Oracle 8i.**

4. **THEORY :-** SQL, there are essentially basically two ways to **INSERT** data into a table: One is to insert it one row at a time, the other is to insert multiple rows at a time. Let's take a look at each of them individually:

The syntax for inserting data into a table one row at a time is as follows:

```
INSERT INTO "table_name" ("column1", "column2", ...)
VALUES ("value1", "value2", ...)
```

Assuming that we have a table that has the following structure,

Table ***Store_Information***

Column Name	Data Type
store_name	char(50)
Sales	float
Date	datetime

and now we wish to insert one additional row into the table representing the sales data for Los Angeles on January 10, 1999. On that day, this store had $900 in sales. We will hence use the following SQL script:

```
INSERT INTO Store_Information (store_name, Sales, Date)
VALUES ('Los Angeles', 900, 'Jan-10-1999')
```

5. **CODING (IF REQUIRED) :-**

```
INSERT INTO Store_Information (store_name, Sales, Date)
VALUES ('Los Angeles', 900, 'Jan-10-1999')
```

6. **PROGRAM INPUTS & OUTPUT :-Input to this code is written in coding statement . Output will be a table look like this given below.**

Store_Name	Sales	Date
Los Angeles	900	'Jan-10-1999'

7. **OBSERVATIONS :-** This command to create table will only make the structure of the table.

EXPERIMENT NO:- 3

1. **OBJECTIVE** :- To Use ALTER Table command in different ways such as to add , modify columns, rows etc.

2. **HARDWARE & SYSTEM SOFTWARE REQUIRED :-Windows Xp and later version.**

3. **SOFTWARE REQUIRED :- Oracle 8i.**

4. **THEORY** :- Once a table is created in the database, there are many occasions where one may wish to change the structure of the table. In general, the SQL syntax for **ALTER TABLE** is

ALTER TABLE "table_name"
[alter specification]

[alter specification] is dependent on the type of alteration we wish to perform. We list a number of common changes below:

- ALTER TABLE Add Column

The SQL syntax for **ALTER TABLE Add Column** is

ALTER TABLE "table_name"
ADD "column 1" "Data Type"

Let's look at the example. Assuming our starting point is the "customer" table created in the **CREATE TABLE** section:

Table ***customer***

Column Name	Data Type
First_Name	char(50)
Last_Name	char(50)
Address	char(50)

City	char(50)
Country	char(25)
Birth_Date	date

5. CODING (IF REQUIRED) :- Our goal is to add a column called "Gender". To do this, we key in:

MySQL:
ALTER TABLE customer ADD Gender char(1);

Oracle:
ALTER TABLE customer ADD Gender char(1);

SQL Server:
ALTER TABLE customer ADD Gender char(1);

6. PROGRAM INPUTS & OUTPUT :-Input to this code is written in coding statement .
Resulting table structure:

Table ***customer***

Column Name	Data Type
First_Name	char(50)
Last_Name	char(50)
Address	char(50)
City	char(50)
Country	char(25)
Birth_Date	date
Gender	char(1)

Note that the new column Gender becomes the last column in the customer table

7.OBSERVATIONS :- This command to alter table is used to add another column in table structure.

EXPERIMENT NO:- 4

1. **OBJECTIVE** :- To Update a table named Store_information using UPDATE command .

2. **HARDWARE & SYSTEM SOFTWARE REQUIRED :-Windows Xp and later version.**

3. **SOFTWARE REQUIRED :- Oracle 8i.**

4. **THEORY :-** Once there's data in the table, we might find that there is a need to modify the data. To do so, we can use the **UPDATE** command. The syntax for this is

```
UPDATE "table_name"
SET "column_1" = [new value]
WHERE {condition}
```

For example, say we currently have a table as below:

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

and we notice that the sales for Los Angeles on 01/08/1999 is actually $500 instead of $300, and that particular entry needs to be updated. To do so, we use the following SQL query:

5. **CODING (IF REQUIRED) :-**

```
UPDATE Store_Information SET Sales = 500 WHERE store_name = "Los Angeles" AND Date = "Jan-08-1999"
```

6. **PROGRAM INPUTS & OUTPUT** :-Input to this code is written in coding statement . Output will be a table look like this given below. The resulting table would look like

Table ***Store_Information***

store_name	Sales	Date

Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$500	Jan-08-1999
Boston	$700	Jan-08-1999

7. **OBSERVATIONS :-** This command to update table will only make updation in particular row or multiple rows.

EXPERIMENT NO:- 5

1. **OBJECTIVE** :- To delete a record from table Store_information using DELETE FROM command.

2. **HARDWARE & SYSTEM SOFTWARE REQUIRED :-Windows Xp and later version.**

3. **SOFTWARE REQUIRED :- Oracle 8i.**

4. **THEORY :-** Sometimes we may wish to use a query to remove records from a table. To do so, we can use the **DELETE FROM** command. The syntax for this is

DELETE FROM "table_name"
WHERE {condition}

It is easiest to use an example. Say we currently have a table as below:

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

5. **CODING (IF REQUIRED) :-**

And we decide not to keep any information on Los Angeles in this table. To accomplish this, we type the following SQL:

DELETE FROM Store_Information
WHERE store_name = "Los Angeles"

6. PROGRAM INPUTS & OUTPUT :-Input to this code is written in coding statement . Now the content of table would look like,

Table ***Store_Information***

store_name	Sales	Date
San Diego	$250	Jan-07-1999

Boston	$700	Jan-08-1999

7. OBSERVATIONS :- This command is used whwnever we want to delete any data fram a table giving certaib condition in WHERE Clause..

EXPERIMENT NO:- 6

1. **OBJECTIVE** :- To create a view of the table named customer using the CREATE VIEW command.

2. **HARDWARE & SYSTEM SOFTWARE REQUIRED :-Windows Xp and later version.**

3. **SOFTWARE REQUIRED :- Oracle 8i.**

4. **THEORY :-** Views can be considered as virtual tables. Generally speaking, a table has a set of definition, and it physically stores the data. A view also has a set of definitions, which is build on top of table(s) or other view(s), and it does not physically store the data.

The syntax for creating a view is as follows:

CREATE VIEW "VIEW_NAME" AS "SQL Statement"

Let's use a simple example to illustrate. Say we have the following table:

TABLE *Customer*(First_Name char(50),Last_Name char(50), Address char(50), City char(50),
Country char(25),Birth_Date date)

and we want to create a view called ***V_Customer*** that contains only the First_Name, Last_Name, and Country columns from this table, we would type in

5. CODING (IF REQUIRED) :-

```
CREATE VIEW V_Customer AS SELECT First_Name, Last_Name,
CountryFROM Customer
```

Now we have a view called ***V_Customer*** with the following structure:

View *V_Customer* (First_Name char(50), Last_Name char(50),Country char(25))

6. PROGRAM INPUTS & OUTPUT :-Input to this code is written in coding statement . We can also use a view to apply joins to two tables. In this case, users only see one view rather than two tables, and the SQL statement users need to issue becomes much simpler. Let's say we have the following two tables:

Table ***Store_Information***

store_name	Sales	Date

Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

Table ***Geography***

region_name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

and we want to build a view that has sales by region information. We would issue the following SQL statement:

```
CREATE VIEW V_REGION_SALES
AS SELECT A1.region_name REGION, SUM(A2.Sales) SALES
FROM Geography A1, Store_Information A2
WHERE A1.store_name = A2.store_name
GROUP BY A1.region_name
```

This gives us a view, ***V_REGION_SALES***, that has been defined to store sales by region records. If we want to find out the content of this view, we type in,

```
SELECT * FROM V_REGION_SALES
```

Result:

REGION	SALES
East	$700
West	$2050

7.OBSERVATIONS :- This command is used to create a view of the table without taking the entire structure of the Parent table.

EXPERIMENT NO:- 7

1. **OBJECTIVE** :- To Study different Aggregate functions of SQL.

2. **HARDWARE & SYSTEM SOFTWARE REQUIRED :-Windows Xp and later version.**

3. **SOFTWARE REQUIRED :- Oracle 8i.**

4. **THEORY :-** Since we have started dealing with numbers, the next natural question to ask is if it is possible to do math on those numbers, such as summing them up or taking their average. The answer is yes! SQL has several arithematic functions, and they are:

• **AVG**: Average of the column.

• **COUNT**: Number of records.

• **MAX**: Maximum of the column.

• **MIN**: Minimum of the column.

• **SUM**: Sum of the column.

The syntax for using functions is,

SELECT "function type" ("column_name") FROM "table_name"

5. CODING (IF REQUIRED) :- 1. AVERAGE FUNCTION - SQL uses the AVG() function to calculate the average of a column. The syntax for using this function is,

SELECT AVG("column_name") FROM "table_name"

For example, if we want to get the average of all sales from the following table,

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

we would type in
SELECT AVG(Sales) FROM Store_Information
Result:

AVG(Sales)
$687.5

$687.5 represents the average of all Sales entries: ($1500 + $250 + $300 + $700) / 4.

2. COUNT FUNCTION -Another arithmetic function is **COUNT**. This allows us to **COUNT** up the number of row in a certain table. The syntax is,

SELECT COUNT("column_name")
FROM "table_name"

For example, if we want to find the number of store entries in our table,

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

we'd key in

SELECT COUNT(store_name) FROM Store_Information

Result:

Count(store_name)
4

COUNT and **DISTINCT** can be used together in a statement to fetch the number of distinct entries in a table. For example, if we want to find out the number of distinct stores, we'd type,

SELECT COUNT(DISTINCT store_name) FROM Store_Information

Result:

Count(DISTINCT store_name)
3

3. MAX FUNCTION - SQL uses the MAX function to find the maximum value in a column. The syntax for using the MAX function is,

SELECT MAX("column_name") FROM "table_name"

For example, if we want to get the highest sales from the following table,

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

we would type in
SELECT MAX(Sales) FROM Store_Information

Result:

MAX(Sales)
$1500

$1500 represents the maximum value of all Sales entries: $1500, $250, $300, and $700.

4. MIN FUNCTION -SQL uses the MIN function to find the maximum value in a column. The syntax for using the MIN function is,

SELECT MIN("column_name") FROM "table_name"

For example, if we want to get the lowest sales from the following table,

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999

Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

we would type in
SELECT MIN(Sales) FROM Store_Information

Result:

MIN(Sales)
$250

$250 represents the minimum value of all Sales entries: $1500, $250, $300, and $700.

5. SUM FUNCTION -The SUM function is used to calculate the total for a column. The syntax is,

SELECT SUM("column_name")FROM "table_name"

For example, if we want to get the sum of all sales from the following table,

Table ***Store_Information***

store_name	Sales	Date
Los Angeles	$1500	Jan-05-1999
San Diego	$250	Jan-07-1999
Los Angeles	$300	Jan-08-1999
Boston	$700	Jan-08-1999

we would type in

SELECT SUM(Sales) FROM Store_Information

Result:

SUM(Sales)
$2750

$2750 represents the sum of all Sales entries: $1500 + $250 + $300 + $700.

6. PROGRAM INPUTS & OUTPUT :-Input to this code is written in coding statement . Output is shown in the coding section for each functions**.**

8. **OBSERVATIONS :-** This aggregate functions are used when calculations are to be peerformed.

EXPERIMENT NO:- 8

1. **OBJECTIVE** :- To study different constraints in SQL.

2. **HARDWARE & SYSTEM SOFTWARE REQUIRED :-Windows Xp and later version.**

3. **SOFTWARE REQUIRED :- Oracle 8i.**

4. **THEORY :-** You can place constraints to limit the type of data that can go into a table. Such constraints can be specified when the table when the table is first created via the **CREATE TABLE** statement, or after the table is already created via the **ALTER TABLE** statement.

Common types of constraints include the following:

- **NOT NULL Constraint**: Ensures that a column cannot have NULL value.
- **DEFAULT Constraint**: Provides a default value for a column when none is specified.
- **UNIQUE Constraint**: Ensures that all values in a column are different.
- **CHECK Constraint**: Makes sure that all values in a column satisfy certain criteria.
- **Primary Key Constraint**: Used to uniquely identify a row in the table.
- **Foreign Key Constraint**: Used to ensure referential integrity of the data.

5. **CODING (IF REQUIRED) :- 1. NOT NULL Constraint-**

By default, a column can hold NULL. If you not want to allow NULL value in a column, you will want to place a constraint on this column specifying that NULL is now not an allowable value.

For example, in the following statement,

CREATE TABLE Customer (SID integer NOT NULL, Last_Name varchar (30) NOT NULL, First_Name varchar(30));

Columns "SID" and "Last_Name" cannot include NULL, while "First_Name" can include NULL.

An attempt to execute the following SQL statement,

INSERT INTO Customer (Last_Name, First_Name) values ('Wong','Ken');

will result in an error because this will lead to column "SID" being NULL, which violates the NOT NULL constraint on that column.

2. DEFAULT Constraint -The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value. For example, if we create a table as below:

CREATE TABLE Student (Student_ID integer Unique, Last_Name varchar (30), First_Name varchar (30), Score DEFAULT 80);

and execute the following SQL statement,

INSERT INTO Student (Student_ID, Last_Name, First_Name) values ('10','Johnson','Rick');

The table will look like the following:

Student_ID	Last_Name	First_Name	Score
10	Johnson	Rick	80

Even though we didn't specify a value for the "Score" column in the INSERT INTO statement, it does get assigned the default value of 80 since we had already set 80 as the default value for this column.

3. UNIQUE Constraint-The UNIQUE constraint ensures that all values in a column are distinct.

For example, in the following CREATE TABLE statement,

CREATE TABLE Customer (SID integer Unique, Last_Name varchar (30), First_Name varchar(30));

column "SID" has a unique constraint, and hence cannot include duplicate values. Such constraint does not hold for columns "Last_Name" and "First_Name". So, if the table already contains the following rows:

SID	Last_Name	First_Name
1	Johnson	Stella
2	James	Gina
3	Aaron	Ralph

Executing the following SQL statement,

INSERT INTO Customer values ('3','Lee','Grace');

will result in an error because '3' already exists in the SID column, thus trying to insert another row with that value violates the UNIQUE constraint.

Please note that a column that is specified as a primary key must also be unique. At the same time, a column that's unique may or may not be a primary key. In addition, multiple UNIQUE constraints can be defined on a table.

4. CHECK Constraint-The CHECK constraint ensures that all values in a column satisfy certain conditions. Once defined, the database will only insert a new row or update an existing row if the new value satisfies the CHECK constraint. The CHECK constraint is used to ensure data quality.

For example, in the following CREATE TABLE statement,

CREATE TABLE Customer (SID integer CHECK (SID > 0), Last_Name varchar (30), First_Name varchar(30));

Column "SID" has a constraint -- its value must only include integers greater than 0. So, attempting to execute the following statement,

INSERT INTO Customer values ('-3','Gonzales','Lynn');

will result in an error because the values for SID must be greater than 0.

Please note that the **CHECK** constraint does not get enforced by MySQL at this time.

A primary key is used to uniquely identify each row in a table. It can either be part of the actual record itself , or it can be an artificial field (one that has nothing to do with the actual record). A primary key can consist of one or more fields on a table. When multiple fields are used as a primary key, they are called a composite key.

5. PRIMARY KEY Constraint -Primary keys can be specified either when the table is created (using **CREATE TABLE**) or by changing the existing table structure (using **ALTER TABLE**).

Below are examples for specifying a primary key when creating a table:

MySQL:
CREATE TABLE Customer (SID integer, Last_Name varchar(30), First_Name varchar(30),
PRIMARY KEY (SID));

Oracle:
CREATE TABLE Customer (SID integer PRIMARY KEY, Last_Name varchar(30), First_Name varchar(30));

SQL Server:
CREATE TABLE Customer (SID integer PRIMARY KEY, Last_Name varchar(30), First_Name varchar(30));

Below are examples for specifying a primary key by altering a table:

MySQL:
ALTER TABLE Customer ADD PRIMARY KEY (SID);

Oracle:
ALTER TABLE Customer ADD PRIMARY KEY (SID);

SQL Server:
ALTER TABLE Customer ADD PRIMARY KEY (SID);

Note: Before using the ALTER TABLE command to add a primary key, you'll need to make sure that the field is defined as 'NOT NULL' -- in other words, NULL cannot be an accepted value for that field.

6. FOREIGN KEY Constraint :-A foreign key is a field (or fields) that points to the primary key of another table. The purpose of the foreign key is to ensure referential integrity of the data. In other words, only values that are supposed to appear in the database are permitted.

For example, say we have two tables, a CUSTOMER table that includes all customer data, and an ORDERS table that includes all customer orders. The constraint here is that all orders must be associated with a customer that is already in the CUSTOMER table. In this case, we will place a foreign key on the ORDERS table and have it relate to the primary key of the CUSTOMER table. This way, we can ensure that all orders in the ORDERS table are related to a customer in the CUSTOMER table. In other words, the ORDERS table cannot contain information on a customer that is not in the CUSTOMER table.

The structure of these two tables will be as follows:

Table ***CUSTOMER***

column name	characteristic
SID	Primary Key
Last_Name	
First_Name	

Table ***ORDERS***

column name	characteristic
Order_ID	Primary Key
Order_Date	
Customer_SID	Foreign Key
Amount	

In the above example, the Customer_SID column in the ORDERS table is a foreign key pointing to the SID column in the CUSTOMER table.

Below we show examples of how to specify the foreign key when creating the ORDERS table:

MySQL:
CREATE TABLE ORDERS (Order_ID integer, Order_Date date, Customer_SID integer, Amount double, Primary Key (Order_ID), Foreign Key (Customer_SID) references CUSTOMER(SID));

Oracle:
CREATE TABLE ORDERS (Order_ID integer primary key, Order_Date date, Customer_SID integer references CUSTOMER(SID), Amount double);

SQL Server:
CREATE TABLE ORDERS (Order_ID integer primary key, Order_Date datetime, Customer_SID integer references CUSTOMER(SID), Amount double);

Below are examples for specifying a foreign key by altering a table. This assumes that the ORDERS table has been created, and the foreign key has not yet been put in:

MySQL:
ALTER TABLE ORDERS ADD FOREIGN KEY (customer_sid) REFERENCES CUSTOMER(SID);

Oracle:

ALTER TABLE ORDERS ADD (CONSTRAINT fk_orders1) FOREIGN KEY (customer_sid) REFERENCES CUSTOMER(SID);

SQL Server:
ALTER TABLE ORDERS ADD FOREIGN KEY (customer_sid) REFERENCES CUSTOMER(SID);

6. **PROGRAM INPUTS & OUTPUT :-Input to this code is written in coding statement . Output is given in the coding section.**

7. **OBSERVATIONS :-** This command are used to limit any table so that the wrong data canot be inserted.

EXPERIMENT NO:-9

1. **OBJECTIVE** :- To study DROP and TRUNCATE Command in SQL.

2. **HARDWARE & SYSTEM SOFTWARE REQUIRED :-Windows Xp and later version.**

3. **SOFTWARE REQUIRED :- Oracle 8i.**

4. **THEORY :- 1. DROP TABLE STATEMENT:-** Sometimes we may decide that we need to get rid of a table in the database for some reason. In fact, it would be problematic if we cannot do so because this could create a maintenance nightmare for the DBA's. Fortunately, SQL allows us to do it, as we can use the **DROP TABLE** command. The syntax for **DROP TABLE** is

DROP TABLE "table_name"

2.TRUNCATE TABLE STATEMENT -Sometimes we wish to get rid of all the data in a table. One way of doing this is with **DROP TABLE**, which we saw in the **last section**. But what if we wish to simply get rid of the data but not the table itself? For this, we can use the **TRUNCATE TABLE** command. The syntax for **TRUNCATE TABLE** is

TRUNCATE TABLE "table_name"

5. **CODING (IF REQUIRED) :-**

So, if we wanted to drop the table called customer that we created in the **CREATE TABLE** section, we simply type

DROP TABLE customer.

So, if we wanted to truncate the table called customer that we created in **SQL CREATE TABLE**, we simply type,

TRUNCATE TABLE customer

6.PROGRAM INPUTS & OUTPUT :-Input to this code is written in coding statement . Output is the deletion of the entire table from database for DROP and only the Table data for TRUNCATE.

7.OBSERVATIONS :- This commands are very useful when we want to delete entire table and if we only want to delete the data of table not the structure of the rable we will use TRUNCATE command.

EXPERIMENT NO:-10

1. **OBJECTIVE** :- To study Substring function in SQL.

2. **HARDWARE & SYSTEM SOFTWARE REQUIRED :-Windows Xp and later version.**

3. **SOFTWARE REQUIRED :- Oracle 8i.**

4. **THEORY :- 1. DROP TABLE STATEMENT:-** The Substring function in SQL is used to grab a portion of the stored data. This function is called differently for the different databases:

   - MySQL: SUBSTR(), SUBSTRING()
   - Oracle: SUBSTR()
   - SQL Server: SUBSTRING()

The most frequent uses are as follows (we will use SUBSTR() here):

SUBSTR(str,pos): Select all characters from <str> starting with position <pos>. Note that this syntax is not supported in SQL Server.

SUBSTR(str,pos,len): Starting with the <pos>th character in string <str> and select the next <len> characters.

Assume we have the following table:

Table ***Geography***

region_name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

5. **CODING (IF REQUIRED) :-**

1. SELECT SUBSTR(store_name, 3) FROM Geography WHERE store_name = 'Los Angeles';

2. SELECT SUBSTR(store_name,2,4) FROM Geography WHERE store_name = 'San Diego';

6.PROGRAM INPUTS & OUTPUT :-Input to this code is written in coding statement . Output –

1. *Result*: **'s Angeles'**

2. Result: **'an D'**

7.OBSERVATIONS :- This commands are very useful when we want to get a portion of the statements.