

Experiment No:1

Title:- To Study SQL DDL, DML, and DCL Statements.

Aim: To implement the DDL, DML and DCL Command on customer table.

Theory:

Introduction to SQL:

- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a number of ways, using English-like statements.

Rules:

SQL follows the following rules:

- Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
- Using the SQL statements, you can perform most of the actions in a database.
- SQL depends on tuple relational calculus and relational algebra.

These SQL commands are mainly categorized into three categories:

1. DDL – Data Definition Language
2. DQL – Data Query Language
3. DML – Data Manipulation Language

1. DDL (Data Definition Language):-

DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database. DDL is a set of SQL commands used to create, modify, and delete database structures but not data.

List of DDL commands:

a)SQL CREATE TABLE Statement:

A **Table** is a combination of rows and columns. For creating a table we have to define the structure of a table by adding names to columns and providing data type and size of data to be stored in columns.

Syntax:

```
CREATE table table_name
(
  Column1 datatype (size),
  column2datatype (size),
  .
  .
  columnNdatatype(size)
);
```

SQL CREATE TABLE Example

Let us create a table to store data of Customers, so the table name is Customer, Columns are Name, Country, age, phone, and so on.

```
CREATE TABLE Customer(
CustomerID INT PRIMARY KEY,
CustomerName VARCHAR(50),
LastName VARCHAR(50),
Country VARCHAR(50),
    Age int(2),
    Phone int(10)
);
```

b) SQL ALTER TABLE – ADD, DROP, MODIFY

The **ALTER TABLE statement in SQL** is used to add, remove, or modify columns in an existing table. The ALTER TABLE statement is also used to add and remove various constraints on existing tables.

i) ALTER TABLE ADD Column Statement in SQL

ADD is used to add columns to the existing table. Sometimes we may require to add additional information, in that case, we do not require to create the whole database again, **ADD** comes to our rescue.

Syntax:

```
ALTER TABLE table_name ADD (Columnname_1 datatype,
Columnname_2 datatype, ...Columnname_n datatype);
```

The following SQL adds an “Email” column to the “Customer” table:

Example:

```
ALTER TABLE Customer
ADD Email varchar(255);
```

ii) ALTER TABLE DROP Column Statement

DROP COLUMN is used to drop columns in a table. Deleting the unwanted columns from the table.

Syntax:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Example:

```
ALTER TABLE Customer
DROP COLUMN Email;
```

iii) ALTER TABLE MODIFY Column Statement in SQL

It is used to modify the existing columns in a table. Multiple columns can also be modified at once. **Syntax may vary slightly in different databases.*

Syntax:

```
ALTER TABLE table_name
MODIFY column_namecolumn_type;
```

Example:

```
ALTER TABLE table_name
MODIFY COLUMN column_namedatatype;
```

```
ALTER TABLE Student ADD
(AGE number(3),COURSE varchar(40));
```

c) DROP, TRUNCATE

DROP

DROP is used to delete a whole database or just a table.

In this article, we will be learning about the DROP statement which destroys objects like an existing database, table, index, or view.

Syntax:

DROP TABLE table_name;

table_name: Name of the table to be deleted.

Examples 2:**To Drop a database**

DROP DATABASE database_name;

TRUNCATE

The major difference between TRUNCATE and DROP is that truncate is used to delete the data inside the table not the whole table.

The result of this operation quickly removes all data from a table,

Syntax:

TRUNCATE TABLE table_name;

table_name: Name of the table to be truncated.

DATABASE name – student_data

DML Commands in SQL

DML is an abbreviation of **Data Manipulation Language**.

The DML commands in Structured Query Language change the data present in the SQL database. We can easily access, store, modify, update and delete the existing records from the database using DML commands.

Following are the four main DML commands in SQL:

1. SELECT Command
2. INSERT Command
3. UPDATE Command
4. DELETE Command

SELECT DML Command

SELECT is the most important data manipulation command in Structured Query Language. The SELECT command shows the records of the specified table. It also shows the particular record of a particular column by using the WHERE clause.

Syntax of SELECT DML command

SELECT column_Name_1, column_Name_2,, column_Name_N **FROM** Name_of_table;

Here, **column_Name_1, column_Name_2,,column_Name_N** are the names of those columns whose data we want to retrieve from the table.

If we want to retrieve the data from all the columns of the table, we have to use the following

SELECT command:

SELECT * FROM table_name;

INSERT DML Command

INSERT is another most important data manipulation command in Structured Query Language, which allows users to insert data in database tables.

Syntax of INSERT Command

INSERT INTO TABLE_NAME (column_Name1 , column_Name2 , column_Name3 , column_Nam eN) **VALUES** (value_1, value_2, value_3, value_N) ;

Examples of INSERT Command

INSERT INTO Student (Stu_id, Stu_Name, Stu_Marks, Stu_Age) **VALUES** (104, Anmol, 89, 19);

UPDATE DML Command

UPDATE is another most important data manipulation command in Structured Query Language, which allows users to update or modify the existing data in database tables.

Syntax of UPDATE Command

UPDATE Table_name **SET** [column_name1= value_1,, column_nameN = value_N] **WHERE** CONDITION;

Example:

UPDATE Product **SET** Product_Price = 80 **WHERE** Product_Id = 'P102' ;

DELETE DML Command

DELETE is a DML command which allows SQL users to remove single or multiple existing records from the database tables.

This command of Data Manipulation Language does not delete the stored data permanently from the database. We use the WHERE clause with the DELETE command to select specific rows from the table.

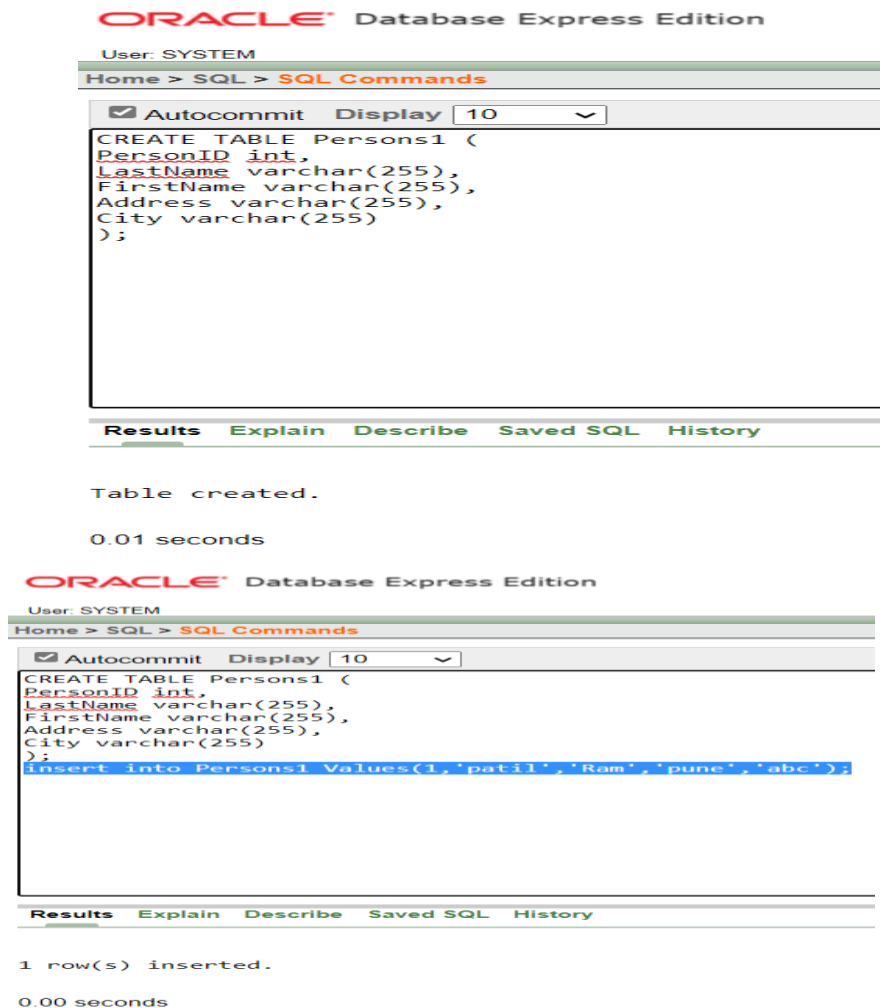
Syntax of DELETE Command

DELETE FROM Table_Name **WHERE** condition;

Examples of DELETE Command

DELETE FROM Product **WHERE** Product_Id = 'P202' ;

OUTPUT Screen:-



ORACLE Database Express Edition

User: SYSTEM

Home > SQL > SQL Commands

☒ Autocommit Display 10

```
CREATE TABLE Persons1 (
  PersonID int,
  LastName varchar(255),
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255)
);
insert into Persons1 Values(1,'patil','Ram','pune','abc');
insert into Persons1 Values(2,'Shinde','Rohan','kodoli','xyz');
insert into Persons1 Values(3,'Huddar','Akshay','warana','qwe');

select *from Persons1;
```

Results Explain Describe Saved SQL History

| PERSONID | LASTNAME | FIRSTNAME | ADDRESS | CITY |
|----------|----------|-----------|---------|------|
| 1 | patil | Ram | pune | abc |
| 2 | Shinde | Rohan | kodoli | xyz |
| 3 | Huddar | Akshay | warana | qwe |

3 rows returned in 0.00 seconds

[CSV Export](#)

ORACLE Database Express Edition

User: SYSTEM

Home > SQL > SQL Commands

☒ Autocommit Display 10

```
CREATE TABLE Persons1 (
  PersonID int,
  LastName varchar(255),
  FirstName varchar(255),
  Address varchar(255),
  City varchar(255)
);
insert into Persons1 Values(1,'patil','Ram','pune','abc');
insert into Persons1 Values(2,'Shinde','Rohan','kodoli','xyz');
insert into Persons1 Values(3,'Huddar','Akshay','warana','qwe');

select *from Persons1;

ALTER TABLE Persons1
ADD moblie int;
```

Results Explain Describe Saved SQL History

| PERSONID | LASTNAME | FIRSTNAME | ADDRESS | CITY | MOBLIE |
|----------|----------|-----------|---------|------|--------|
| 1 | patil | Ram | pune | abc | - |
| 2 | Shinde | Rohan | kodoli | xyz | - |
| 3 | Huddar | Akshay | warana | qwe | - |

ORACLE Database Express Edition

User: SYSTEM

Home > SQL > SQL Commands

☒ Autocommit Display 10

```
FirstName varchar(255),
Address varchar(255),
City varchar(255)
);
insert into Persons1 Values(1,'patil','Ram','pune','abc');
insert into Persons1 Values(2,'Shinde','Rohan','kodoli','xyz');
insert into Persons1 Values(3,'Huddar','Akshay','warana','qwe');

select *from Persons1;

ALTER TABLE Persons1
ADD moblie int;

UPDATE Persons1
SET LastName = 'sharma', City= 'poi'
WHERE PersonID = 1;
```

Results Explain Describe Saved SQL History

| PERSONID | LASTNAME | FIRSTNAME | ADDRESS | CITY | MOBLIE |
|----------|----------|-----------|---------|------|--------|
| 1 | sharma | Ram | pune | poi | - |
| 2 | Shinde | Rohan | kodoli | xyz | - |
| 3 | Huddar | Akshay | warana | qwe | - |

3 rows returned in 0.00 seconds

[CSV Export](#)

ORACLE® Database Express Edition

User: SYSTEM

Home > SQL > SQL Commands

☒ Autocommit Display 10

```
FirstName varchar(255),
Address varchar(255),
City varchar(255)
);
insert into Persons1 Values(1,'patil','Ram','pune','abc');
insert into Persons1 Values(2,'Shinde','Rohan','kodoli','xyz');
insert into Persons1 Values(3,'Huddar','Akshay','warana','qwe');

select *from Persons1;

ALTER TABLE Persons1
ADD moblie int;

UPDATE Persons1
SET LastName = 'sharma', City= 'poi'
WHERE PersonID = 1;
```

Results Explain Describe Saved SQL History

1 row(s) updated.

0.00 seconds

ORACLE® Database Express Edition

User: SYSTEM

Home > SQL > SQL Commands

☒ Autocommit Display 10

```
City varchar(255)
);
insert into Persons1 Values(1,'patil','Ram','pune','abc');
insert into Persons1 Values(2,'Shinde','Rohan','kodoli','xyz');
insert into Persons1 Values(3,'Huddar','Akshay','warana','qwe');

select *from Persons1;

ALTER TABLE Persons1
ADD moblie int;

UPDATE Persons1
SET LastName = 'sharma', City= 'poi'
WHERE PersonID = 1;

DROP table Persons1;
```

Results Explain Describe Saved SQL History

Table dropped.

0.81 seconds

Conclusion:-

Thus we study SQL DDL,DML and DCL Statements.

EXPERIMENT NO-2

Title:- To Study and implement the SQL queries on sample table using set operations and SQL join queries

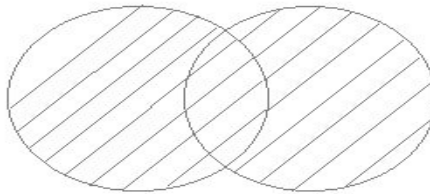
Aim:- Write and execute SQL queries using set operations and join on DB Schema like Student Schema, Employee Schema, College Schema etc.

Relational Set Operations:

DBMS supports relational set operators as well. The major relational set operators are union, intersection and set difference. All of these can be implemented in DBMS using different queries.

1) UNION Operation

UNION is used to combine the results of two or more **SELECT** statements. However it will eliminate duplicate rows from its result set. In case of union, number of columns and data type must be same in both the tables, on which **UNION** operation is being applied.

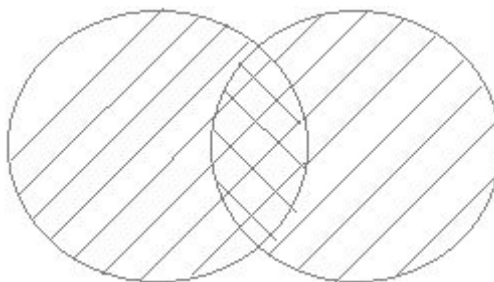


Syntax

```
SELECT column_name FROM table1  
UNION  
SELECT column_name FROM table2;
```

2) UNION ALL:

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

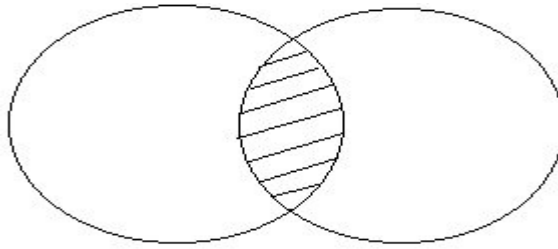


Syntax:

```
SELECT column_name FROM table1  
UNION ALL  
SELECT column_name FROM table2;
```

3) INTERSECT:

It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements. In the Intersect operation, the number of datatype and columns must be the same. It has no duplicates and it arranges the data in ascending order by default.

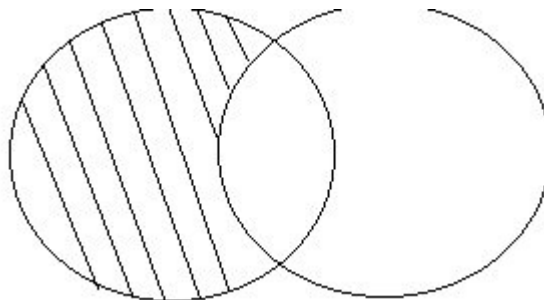


Syntax

```
SELECT column_name FROM table1  
INTERSECT  
SELECT column_name FROM table2;
```

4) MINUS

It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query. It has no duplicates and data arranged in ascending order by default.



Syntax:

```
SELECT column_name FROM table1  
MINUS  
SELECT column_name FROM table2;
```

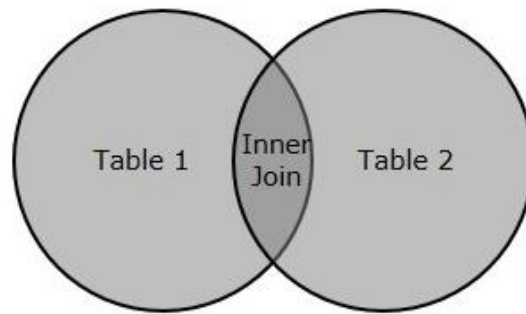
SQL Join queries:

An SQL **Join** clause is used to combine multiple related tables in a database, based on common fields/columns.

There are two major types of joins: **Inner Join** and **Outer Join**. Other joins like Left Join, Right Join, Full Join etc. are just subtypes of these two major joins

A) INNER JOIN:-

The **SQL Inner Join** is a type of join that combines multiple tables by retrieving records that have matching values in both tables (in the common column).



Syntax

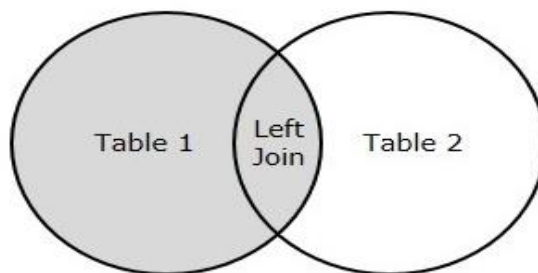
```
SELECT table1.column1, table1.column2, table2.column1,....  
FROM table1  
INNER JOIN table2  
ON table1.matching_column = table2.matching_column;
```

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT  
FROM EMPLOYEE  
INNER JOIN PROJECT  
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

B. LEFT JOIN:

The SQL left join returns all the values from left table and the matching values from the right table. If there is no matching join value, it will return NULL.



Syntax

```
SELECT table1.column1, table1.column2, table2.column1,....  
FROM table1  
LEFT JOIN table2  
ON table1.matching_column = table2.matching_column;
```

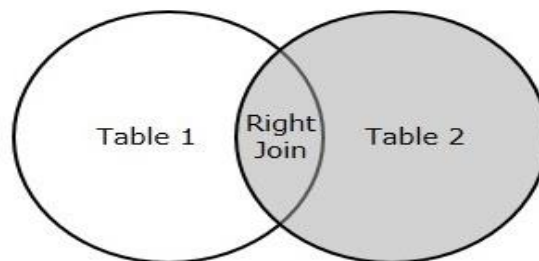
Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT  
FROM EMPLOYEE
```

LEFT JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;

C) RIGHT JOIN:-

In SQL, RIGHT JOIN returns all the values from the values from the rows of right table and the matched values from the left table. If there is no matching in both tables, it will return NULL.



Syntax

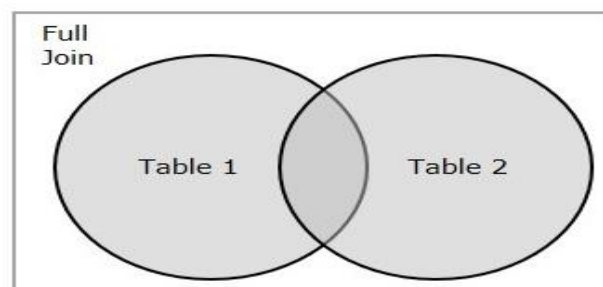
```
SELECT table1.column1, table1.column2, table2.column1,....  
FROM table1  
RIGHT JOIN table2  
ON table1.matching_column = table2.matching_column;
```

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT  
FROM EMPLOYEE  
RIGHT JOIN PROJECT  
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

D) FULL JOIN:-

In SQL, FULL JOIN is the result of a combination of both left and right outer join. Join tables have all the records from both tables. It puts NULL on the place of matches not found.



Syntax

```
SELECT table1.column1, table1.column2, table2.column1 ...  
FROM table1  
FULL JOIN table2
```

ON table1.matching_column = table2.matching_column;

Query

```
SELECT EMPLOYEE.EMP_NAME, PROJECT.DEPARTMENT
FROM EMPLOYEE
FULL JOIN PROJECT
ON PROJECT.EMP_ID = EMPLOYEE.EMP_ID;
```

E) CROSS JOIN:-

An **SQL Cross Join** is a basic type of inner join that is used to retrieve the Cartesian product (or cross product) of two individual tables. That means, this join will combine each row of the first table with each row of second table.

Syntax:

```
SELECT table1.column1 , table1.column2, table2.column1...
FROM table1
CROSS JOIN table2;
```

Query:-

```
SELECT Student.NAME, Student.AGE, StudentCourse.COURSE_ID
FROM Student
CROSS JOIN StudentCourse;
```

OUTPUT SCREEN:-

The screenshot shows the Oracle Database Express Edition interface. The 'SQL Commands' tab is active, displaying the following SQL code:

```
create table CustomerData(student_id INT,
first_name VARCHAR(50),
last_name VARCHAR(50),
address VARCHAR(50),
city VARCHAR(50),
state VARCHAR(50),
zip_code VARCHAR(50),
email VARCHAR(50),
PRIMARY KEY(student_id)
);
```

Below the code editor, the 'Results' tab shows the message 'Table created.' and the execution time '0.53 seconds'.

The screenshot shows the Oracle Database Express Edition interface. The 'SQL Commands' tab is active, displaying the following SQL code:

```
insert into CustomerData values(2,'Wallis','Breadwood','91869 Dexter Parkway','Pomana','CA','97896','wbreadwood1@co.uk')
insert into CustomerData values(3,'Brigid','McKevitt','73 Namekagon Park','Washington','DC','20525','wmckevitt0@co.uk')
select * from CustomerData
```

Below the code editor, the 'Results' tab shows the output of the SQL query. The output is a table with 8 columns: STUDENT_ID, FIRST_NAME, LAST_NAME, ADDRESS, CITY, STATE, ZIP_CODE, and EMAIL. The table contains 3 rows of data.

| STUDENT_ID | FIRST_NAME | LAST_NAME | ADDRESS | CITY | STATE | ZIP_CODE | EMAIL |
|------------|------------|-----------|----------------------|------------|-------|----------|-------------------|
| 1 | windham | McKevitt | 73 Namekagon Park | Washington | DC | 20525 | wmckevitt0@co.uk |
| 2 | Wallis | Breadwood | 91869 Dexter Parkway | Pomana | CA | 97896 | wbreadwood1@co.uk |
| 3 | Brigid | McKevitt | 73 Namekagon Park | Washington | DC | 20525 | wmckevitt0@co.uk |

3 rows returned in 0.00 seconds [CSV Export](#)

User: SYSTEM

Home > SQL > SQL Commands

☒ Autocommit Display 10

```
create table ORDERDATA(order_id INT,
order_date VARCHAR(50),
amount VARCHAR(50),
customer_id INT,
PRIMARY KEY(order_id));

insert into ORDERDATA values(35,'07-10-2023','$56',2)

select * from ORDERDATA
```

Results Explain Describe Saved SQL History

| ORDER_ID | ORDER_DATE | AMOUNT | CUSTOMER_ID |
|----------|------------|--------|-------------|
| 34 | 07-09-2023 | \$56 | 1 |
| 35 | 07-10-2023 | \$56 | 2 |

2 rows returned in 0.03 seconds

[CSV Export](#)

User: SYSTEM

Home > SQL > SQL Commands

☒ Autocommit Display 10

```
RENAME COLUMN STUDENT_ID TO customer_id ;

select first_name,last_name,order_date,amount from CustomerData c
inner join ORDERDATA o on c.CUSTOMER_ID=o.CUSTOMER_ID
ORDER BY order_date
```

Results Explain Describe Saved SQL History

| FIRST_NAME | LAST_NAME | ORDER_DATE | AMOUNT |
|------------|-----------|------------|--------|
| windham | McKevitt | 07-09-2023 | \$56 |
| Wallis | Breadwood | 07-10-2023 | \$56 |

2 rows returned in 0.13 seconds

[CSV Export](#)

ORACLE® Database Express Edition

User: SYSTEM

Home > SQL > SQL Commands

☒ Autocommit Display 10

```
select first_name,last_name,order_date,amount from CustomerData c
right join ORDERDATA o on c.CUSTOMER_ID=o.CUSTOMER_ID
ORDER BY order_date
```

Results Explain Describe Saved SQL History

| FIRST_NAME | LAST_NAME | ORDER_DATE | AMOUNT |
|------------|-----------|------------|--------|
| windham | McKevitt | 07-09-2023 | \$56 |
| Wallis | Breadwood | 07-10-2023 | \$56 |

2 rows returned in 0.02 seconds

[CSV Export](#)

Conclusion:

Thus, we Study and implement the SQL queries on sample tables using set operations and SQL Join queries.

EXPERIMENT NO: 03

Title: To Study and implement the SQL sub queries and co-related queries.

Aim:: Write and execute SQL sub queries and co-related queries

THEORY:

A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within clauses, most commonly in the WHERE clause. It is used to return data from a table, and this data 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.

There are a few rules that subqueries must follow –

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.
- Subqueries are nested inside statements like SELECT, INSERT, UPDATE, DELETE, or any other subquery.
- Subqueries are present in the WHERE clause, FROM clause, or HAVING clause of the PARENT SQL query.
- They are used with comparison operators and logical operators like >, <, >=, <=, <>, SOME, ANY, ALL, and IN.
- They execute before the outer query at the run time and pass the result to complete the statement.
- Subqueries are used to compare an expression to the output and check if any row gets selected.

A maximum of 255 subquery levels can be nested in a WHERE clause. The FROM clause has no limit in nesting subqueries. In the real world, we encounter not more than five subqueries. So, 255 is too large to be set as a limit.

Types of Sub queries:

1. Single Row Sub query
2. Multirow Sub query
3. Correlated Sub query
4. Nested Sub Queries
5. Scalar Sub queries

1. Single Row Subquery with real life example:

When Query within a query or subquery returns only one row then these type of queries are called as single row subqueries. Single row comparison operator is used to compare the two queries. The most widely used operator for single row subquery is Equal to operator(=). Here We need to make sure that the query is returning only one value. Here we are able to use Max, Min, AVG like functions which will return only one value.

Real life Scenario:

Write a query to find highest salaried Employee from Employee table.

Select Employee_No, Employee_Name from Employee

where Salary=(Select max(Salary) from Employee);

In above query the inner query is executed 1st then the value of inner-query is assigned to the outer query.

Step 1: Executed inner query :(Select max(Salary) from Employee);

consider Output is: 50000

Step 2:

Select Employee_No, Employee_Name from Employee

where Salary=50000;

2. Multi Row Subqueries with Real life Scenario:

If the output of Inner query count is more than 1 then these subqueries are called as multi row subqueries. We need to use ANY, IN, EXIST Operator in outer query of multi row subqueries because output of outer query is not a single value.

Real Life Scenario:

Fetch the list of Employees which is assigned to 'OBIEE' and 'Oracle' Department.

Select Employee_No, Employee_Name from Employee

where Department_Name in

(Select Department_Name from Employee where Department_name in ('OBIEE', 'Oracle'));

The Query is executed in following Steps:

Step 1:

Inner Query Execution:

Select Department_Name from Employee where Department_name in ('OBIEE', 'Oracle');

Consider the output is 'OBIEE' and 'ORACLE'

Step 2:

Outer Query Execution:

Select Employee_No, Employee_Name from Employee

where Department_Name in ('OBIEE', 'ORACLE');

Correlated Subquery :

Correlated Query is nothing but the subquery whose output is depending on the inner query used in that query. Correlated query is the query which is executed after the outer query is executed. The outer query is always dependent on inner query. The approach of the correlated subquery is bit different than normal subqueries. In normal subqueries the inner queries are executed first and then the outer query is executed but in Correlated Subquery outer query is always dependent on inner query so first outer query is executed then inner query is executed. Correlated Subqueries always uses operator like Exist, Not Exist, IN, Not IN.

“Correlated Queries are also called as Synchronized queries...”

Execution Steps of Correlated Subqueries:

- 1.Executes the outer Query
- 2.For Each row of outer query inner subquery is executed once
- 3.The result of correlated subquery determines whether the fetched row should be the part of our output results
- 4.The Process is Repeated for all Rows

“It is not recommended to use Correlated Subqueries as it slows down the performance”

Real Life Example:

Fetch the Employees who have not assigned a single department.

Select * from Employee E where Not exist

(Select Department_noFrom Department D where E.Employee_id=D.Employee_ID);

Execution of query:

Step 1:

Select * from Employee E ;

It will fetch the all employees

Step 2:

The First Record of the Employee second query is executed and output is given to first query.

(Select Department_noFrom Department D where E.Employee_id=D.Employee_ID);

Step 3:

Step 2 is repeated until and unless all output is been fetched.

Nested Subqueries:

The Subqueries are called as nested subqueries when another subquery is used in where or having condition of the Outer Query.The Execution of Nested subquery always follows bottom up approach.

Real Life Example:

Select * from Employee

whereEmployee_No Exist

(Select * from Employee

where Department_Name=

(Select Department_Name from Employee where Department_Name='OBIEE'));

Execution of Query:

Step 1:

Executed Bottom query:

Select Department_Name from Employee where Department_Name='OBIEE';

Step 2:

Executed The Second Query which is above bottom query:

Select * from Employee

where Department_Name='OBIEE';

Step 3:

Executed the Top Query

Select * from Employee

where Employee_No Exist

(Select * from Employee

where Department_Name='OBIEE');

4. Scalar Sub-queries :

Definition of Scalar Subquery:

A scalar sub-query expression is a sub-query that returns exactly one column value from one row. What if the oracle failed to return scalar sub-query? There are some specific conditions.

Usages of Scalar Query :

1. The scalar sub-queries are most used for removing the outer joins.
2. If user want to aggregate multiple tables then scalar sub-queries are useful.
3. Table insertion based on other table values.

Real Life Example:

If user want to find out the Department_name and Number_of_Departments using scalar query you can use following expression :

Select D.Department_name (Select Count(E.Department_name) From Employee E Where E.Department_no=D.Department_no;

Output:

ORACLE® Database Express Edition

User: SYSTEM

Home > SQL > SQL Commands

☒ Autocommit Display 10

```
create table student11(studno int, studname varchar(30),deptname varchar(20),marks int);
insert into student11 values(1,'ASD','MECH',80);
insert into student11 values(2,'LKJ','MECH',60);
insert into student11 values(3,'QWE','CSE',88);
select *from student11
select studno,studname from student11 where marks=80;
```

Results Explain Describe Saved SQL History

| STUDNO | STUDNAME |
|--------|----------|
| 1 | ASD |

1 rows returned in 0.00 seconds

[CSV Export](#)

ORACLE® Database Express Edition

User: SYSTEM

Home > SQL > SQL Commands

☒ Autocommit Display 10

```
create table student11(studno int, studname varchar(30),deptname varchar(20),marks int);
insert into student11 values(1,'ASD','MECH',80);
insert into student11 values(2,'LKJ','MECH',60);
insert into student11 values(3,'QWE','CSE',88);
select *from student11
select studno,studname from student11 where marks=80;
select studno,studname from student11 where deptname in('MECH','CSE');
```

Results Explain Describe Saved SQL History

| STUDNO | STUDNAME |
|--------|----------|
| 1 | ASD |
| 2 | LKJ |
| 3 | QWE |

3 rows returned in 0.00 seconds

[CSV Export](#)

ORACLE® Database Express Edition

User: SYSTEM

Home > SQL > SQL Commands

☒ Autocommit Display 10

```
create table student11(studno int, studname varchar(30),deptname varchar(20),marks int);
insert into student11 values(1,'ASD','MECH',80);
insert into student11 values(2,'LKJ','MECH',60);
insert into student11 values(3,'QWE','CSE',88);
insert into student11 values(4,'MNB','CIVIL',78);

select *from student11
select studno,studname from student11 where marks=80;
select studno,studname from student11 where deptname in('MECH','CSE');
select studno,studname from student11 where deptname not in('MECH','CSE');
```

Results Explain Describe Saved SQL History

| STUDNO | STUDNAME |
|--------|----------|
| 4 | MNB |

1 rows returned in 0.00 seconds

[CSV Export](#)

User: SYSTEM

Home > SQL > SQL Commands

☒ Autocommit Display 10

```
create table customer21(cust_ID int, firstname varchar(30),lastname varchar(20));
insert into customer21 values(1,'ASD','THU');
insert into customer21 values(2,'LKJ','RDG');
insert into customer21 values(3,'QWE','WSR');
insert into customer21 values(4,'MNB','PKY');
select *from customer21
create table order21(cust_ID int,order_ID int);
insert into order21 values(1,421);
insert into order21 values(2,422);
insert into order21 values(3,423);
insert into order21 values(4,424);
select *from order21
SELECT firstname, lastname FROM customer21 WHERE EXISTS (SELECT * FROM Order21 WHERE Customer21.cust_Id = Order21.order_ID);
```

Results Explain Describe Saved SQL History

no data found

User: SYSTEM

Home > SQL > SQL Commands

☒ Autocommit Display 10

```
create table customer21(cust_ID int, firstname varchar(30),lastname varchar(20));
insert into customer21 values(1,'ASD','THU');
insert into customer21 values(2,'LKJ','RDG');
insert into customer21 values(3,'QWE','WSR');
insert into customer21 values(4,'MNB','PKY');
select *from customer21
create table order21(cust_ID int,order_ID int);
insert into order21 values(1,421);
insert into order21 values(2,422);
insert into order21 values(3,423);
insert into order21 values(4,424);
select *from order21
SELECT firstname, lastname FROM customer21 WHERE NOT EXISTS (SELECT * FROM Order21 WHERE Customer21.cust_Id = Order21.order_ID);
```

Results Explain Describe Saved SQL History

| FIRSTNAME | LASTNAME |
|-----------|----------|
| MNB | PKY |
| QWE | WSR |
| ASD | THU |
| LKJ | RDG |

4 rows returned in 0.11 seconds

[CSV Export](#)

Conclusion:

Thus, we studied sub queries and co-related queries successfully.

EXPERIMENT NO-4

| |
|---|
| Title:- To study partitioning queries on parallel databases. |
| Aim:- To implement partitioning queries using MySQL. |

Partitioning

The **Partitioning** is a technique that can be used to divide a database table into smaller tables i.e. partitions.

These smaller tables are stored in different physical locations and are treated as separate tables. Thus, the data in these smaller tables can be accessed and managed individually.

But note that, even if the data smaller tables are managed separately, they are not independent tables; i.e., they are still a part of main table.

There are two forms of partitioning:

Horizontal Partitioning and **Vertical Partitioning**.

Horizontal Partitioning

The **Horizontal partitioning** is used to divide the table rows into multiple partitions. Since it divides the rows, all the columns will be present in each partition. All the partitions can be accessed individually or collectively.

There are several types of horizontal partitioning methods –

- Range Partitioning
- List Partitioning
- Hash Partitioning
- Key Partitioning
- Sub-partitioning

Range Partitioning

The MySQL RANGE partitioning is used to divide a table into partitions based on a specific range of column values. Each table partition contains rows with column values falling within that defined range.

Example

Let us create a table named CUSTOMERS and partition it by the AGE column into four partitions: P1, P2, P3, and P4 using the "PARTITION BY RANGE" clause –

```
CREATE TABLE CUSTOMERS(  
  ID int not null,  
  NAME varchar(40) not null,  
  AGE int not null,  
  ADDRESS char(25) not null,
```

```

SALARY decimal(18, 2)
)
PARTITION BY RANGE (AGE) (
PARTITION P1 VALUES LESS THAN (20),
PARTITION P2 VALUES LESS THAN (30),
PARTITION P3 VALUES LESS THAN (40),
PARTITION P4 VALUES LESS THAN (50)
);

```

Here, we are inserting rows into the above created table –

```

INSERT INTO CUSTOMERS VALUES
(1, 'Ramesh', 19, 'Ahmedabad', 2000.00 ),
(2, 'Khilan', 25, 'Delhi', 1500.00 ),
(3, 'kaushik', 23, 'Kota', 2000.00 ),
(4, 'Chaitali', 31, 'Mumbai', 6500.00 ),
(5, 'Hardik', 35, 'Bhopal', 8500.00 ),
(6, 'Komal', 47, 'MP', 4500.00 ),
(7, 'Muffy', 43, 'Indore', 10000.00 );

```

Following is the **CUSTOMERS** table obtained –

| ID | NAME | AGE | ADDRESS | SALARY |
|----|----------|-----|-----------|----------|
| 1 | Ramesh | 19 | Ahmedabad | 2000.00 |
| 2 | Khilan | 25 | Delhi | 1500.00 |
| 3 | Kaushik | 23 | Kota | 2000.00 |
| 4 | Chaitali | 31 | Mumbai | 6500.00 |
| 5 | Hardik | 35 | Bhopal | 8500.00 |
| 6 | Komal | 47 | MP | 4500.00 |
| 7 | Muffy | 43 | Indore | 10000.00 |

Now that we have some data in the CUSTOMERS table, we can display the partition status to see how the data is distributed among the partitions using the following query –

```

SELECT PARTITION_NAME, TABLE_ROWS
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_NAME='CUSTOMERS';

```

The above query will show us the number of rows in each partition. For example, P1 has 1 row, P2 has 2 rows, P3 has 2 rows, and P4 has 2 rows as shown below –

| PARTITION_NAME | TABLE_ROWS |
|----------------|------------|
| P1 | 1 |
| P2 | 2 |
| P3 | 2 |

| | |
|----|---|
| P4 | 2 |
|----|---|

Displaying Partitions –

We can also display data from specific partitions using the PARTITION clause. For instance, to retrieve data from partition P1, we use the following query –

```
SELECT * FROM CUSTOMERS PARTITION (p1);
```

It will display all the records in partition P1 –

| ID | NAME | AGE | ADDRESS | SALARY |
|----|--------|-----|-----------|---------|
| 1 | Ramesh | 19 | Ahmedabad | 2000.00 |

Similarly, we can display other partitions using the same syntax.

Handling Data Outside the Range –

If we attempt to insert a value into the AGE column that doesn't fall within any of the defined partitions, it will fail with an error, as shown below –

```
INSERT INTO CUSTOMERS VALUES  
(8, 'Brahmi', 70, 'Hyderabad', 19000.00 );
```

Following is the error obtained – ERROR 1526 (HY000): Table has no partition for value 70

Truncating Partitions –

We can also manage partitions by truncating them if needed. For example, to empty partition P2, we can use the following query –

```
ALTER TABLE CUSTOMERS TRUNCATE PARTITION p2;
```

The output obtained is as shown below – Query OK, 0 rows affected (0.03 sec)

This will remove all data from partition P2, making it empty as shown below –

```
SELECT * FROM CUSTOMERS PARTITION (p2);
```

Following is the output produced – Empty set (0.00 sec)

We can verify the CUSTOMERS table using the following SELECT query –

```
SELECT * FROM CUSTOMERS;
```

We can see in the table below that the rows belonging to p2 partition are deleted –

| ID | NAME | AGE | ADDRESS | SALARY |
|----|------|-----|---------|--------|
|----|------|-----|---------|--------|

| | | | | |
|---|---------|----|-----------|----------|
| 1 | Ramesh | 19 | Ahmedabad | 2000.00 |
| 2 | Khilan | 25 | Delhi | 1500.00 |
| 3 | Kaushik | 23 | Kota | 2000.00 |
| 6 | Komal | 47 | MP | 4500.00 |
| 7 | Muffy | 43 | Indore | 10000.00 |

List Partitioning

The MySQL List Partitioning is used to divide the table into partitions based on a discrete set of values for a specific column. Each partition contains rows that match a particular value within the defined set.

Example

In this example, we will create a table named STUDENTS and divide it into four partitions (P1, P2, P3, and P4) based on the "DEPARTMENT_ID" column using the "PARTITION BY LIST" clause –

```
CREATE TABLE STUDENTS( ID int, NAME varchar(50), DEPARTMENT varchar(50),
DEPARTMENT_ID int)

PARTITION BY LIST(DEPARTMENT_ID)(
  PARTITION P1 VALUES IN (3, 5, 6, 7, 9),
  PARTITION P2 VALUES IN (13, 15, 16, 17, 20),
  PARTITION P3 VALUES IN (23, 25, 26, 27, 30),
  PARTITION P4 VALUES IN (33, 35, 36, 37, 40)
);
```

Here, we are inserting rows into the above-created table –

```
INSERT INTO STUDENTS VALUES
(1, 'Ramesh', "cse", 5),
(2, 'Khilan', "mech", 20),
(3, 'kaushik', "ece", 17),
(4, 'Chaitali', "eee", 33),
(5, 'Hardik', "IT", 36),
(6, 'Komal', "Hotel management", 40),
(7, 'Muffy', "Fashion", 23);
```

Following is the **STUDENTS** table obtained –

| ID | NAME | DEPARTMENT | DEPARTMEN T_ID |
|----|--------|------------|-------------------|
| 1 | Ramesh | cse | 5 |
| 2 | Khilan | mech | 20 |

| | | | |
|---|----------|------------------|----|
| 3 | Kaushik | ece | 17 |
| 7 | Muffy | Fashion | 23 |
| 4 | Chaitali | eee | 33 |
| 5 | Hardik | IT | 36 |
| 6 | Komal | Hotel management | 40 |

We can display the partition status of the STUDENTS table to see how the data is distributed among partitions using the following query –

```
SELECT PARTITION_NAME, TABLE_ROWS
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_NAME='STUDENTS';
```

The output of this query will show the number of rows in each partition.
For instance, P1 has 1 row, P2 has 2 rows, P3 has 1 row, and P4 has 3 rows –

| PARTITION_NAME | TABLE_ROWS |
|----------------|------------|
| P1 | 1 |
| P2 | 2 |
| P3 | 1 |
| P4 | 3 |

Hash Partitioning

The MySQL HASH partitioning is used to divide the table data into partitions using a hash function based on a specific column(s). The data will be evenly distributed among the partitions.

Example

In the following query, we are creating a table with the name **EMPLOYEES** with four partitions based on the "id" column using the PARTITION BY HASH clause –

```
CREATE TABLE EMPLOYEES ( id INT NOT NULL, name VARCHAR(50) NOT NULL,
department VARCHAR(50) NOT NULL,
salary INT NOT NULL ) PARTITION BY HASH(id) PARTITIONS 4;
```

Here, we are inserting rows into the above-created table –

```
INSERT INTO EMPLOYEES VALUES
(1, 'Varun', 'Sales', 50000),
(2, 'Aarohi', 'Marketing', 60000),
(3, 'Paul', 'IT', 70000),
(4, 'Vaidhya', 'Finance', 80000),
(5, 'Nikhil', 'Sales', 55000),
(6, 'Sarah', 'Marketing', 65000),
(7, 'Tim', 'IT', 75000),
```

```
(8, 'Priya', 'Finance', 85000);
```

The **EMPLOYEES** table obtained is as follows –

| id | name | department | salary |
|----|---------|------------|--------|
| 4 | Vaidhya | Finance | 80000 |
| 8 | Priya | Finance | 85000 |
| 1 | Varun | Sales | 50000 |
| 5 | Nikhil | Sales | 55000 |
| 2 | Aarohi | Marketing | 60000 |
| 6 | Sarah | Marketing | 65000 |
| 3 | Paul | IT | 70000 |
| 7 | Tim | IT | 75000 |

The records are evenly distributed among four partitions based on the "id" column. You can verify the partition status using the following SELECT query –

```
SELECT PARTITION_NAME, TABLE_ROWS
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_NAME='EMPLOYEES';
```

The table obtained is as follows –

| PARTITION_NAME | TABLE_ROWS |
|----------------|------------|
| P0 | 2 |
| P1 | 2 |
| P2 | 2 |
| P3 | 2 |

Key Partitioning

The MySQL key partitioning is used to divide the table data into partitions based on the values of the primary key or a unique key.

Example

In the following query, we are creating a table with the name **PERSON** with Key partitioning on the "id" column. We have divided the table into four partitions, and the primary key is "id" –

```
CREATE TABLE PERSON (
  id INT NOT NULL,
  name VARCHAR(50) NOT NULL,
  email VARCHAR(50) NOT NULL,
```



```
address VARCHAR(100) NOT NULL,
PRIMARY KEY (id)
)
PARTITION BY KEY(id)
PARTITIONS 4;
```

Here, we are inserting rows into the above-created table –

```
INSERT INTO PERSON VALUES
(1, 'Krishna', 'Krishna@tutorialspoint.com', 'Ayodhya'),
(2, 'Kasyap', 'Kasyap@tutorialspoint.com', 'Ayodhya'),
(3, 'Radha', 'Radha@tutorialspoint.com', 'Ayodhya'),
(4, 'Sarah', 'Sarah@tutorialspoint.com', 'Sri Lanka'),
(5, 'Sita', 'Sita@tutorialspoint.com', 'Sri Lanka'),
(6, 'Arjun', 'Arjun@tutorialspoint.com', 'India'),
(7, 'Hanuman', 'Hanuman@tutorialspoint.com', 'Sri Lanka'),
(8, 'Lakshman', 'Lakshman@tutorialspoint.com', 'Sri Lanka');
```

Following is the **PERSON** table obtained –

| id | name | email | address |
|----|----------|-----------------------------|-----------|
| 1 | Krishna | Krishna@tutorialspoint.com | Ayodhya |
| 5 | Sita | Sita@tutorialspoint.com | Sri Lanka |
| 4 | Sarah | Sarah@tutorialspoint.com | Sri Lanka |
| 8 | Lakshman | Lakshman@tutorialspoint.com | Sri Lanka |
| 3 | Radha | Radha@tutorialspoint.com | Ayodhya |
| 7 | Hanuman | Hanuman@tutorialspoint.com | Sri Lanka |
| 2 | Kasyap | Kasyap@tutorialspoint.com | Ayodhya |
| 6 | Arjun | Arjun@tutorialspoint.com | India |

Again, the data is evenly distributed among partitions based on the "id" column, and you can verify the partition status using the query given below –

```
SELECT PARTITION_NAME, TABLE_ROWS
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_NAME='PERSON';
```

The output obtained is as shown below –

| PARTITION_NAME | TABLE_ROWS |
|----------------|------------|
| P0 | 2 |
| P1 | 2 |
| P2 | 2 |
| P3 | 2 |

Sub-partitioning

The sub partitioning is used to further divide partitions based on another column, often used in conjunction with other partitioning methods like RANGE or HASH.

Example

Let us create a CUSTOMER_ORDERS table with RANGE partitioning on the "order_date" column, and then we will subpartition by hashing on the month of "order_date"

```
CREATE TABLE CUSTOMER_ORDERS (
  order_id INT NOT NULL,
  customer_name VARCHAR(50) NOT NULL,
  order_date DATE NOT NULL,
  order_status VARCHAR(20) NOT NULL
)
PARTITION BY RANGE (YEAR(order_date))
SUBPARTITION BY HASH(MONTH(order_date))
SUBPARTITIONS 2(
  PARTITION p0 VALUES LESS THAN (2022),
  PARTITION p1 VALUES LESS THAN (2023),
  PARTITION p2 VALUES LESS THAN (2024)
);
```

Here, we are inserting rows into the above-created table –

```
INSERT INTO CUSTOMER_ORDERS VALUES
(1, 'John', '2021-03-15', 'Shipped'),
(2, 'Bob', '2019-01-10', 'Delivered'),
(3, 'Johnson', '2023-01-10', 'Delivered'),
(4, 'Jake', '2020-01-10', 'Delivered'),
(5, 'Smith', '2022-05-01', 'Pending'),
(6, 'Rob', '2023-01-10', 'Delivered');
```

Following is the CUSTOMERS_ORDERS table obtained –

| order_id | customer_name | order_date | order_status |
|----------|---------------|------------|--------------|
| 1 | John | 2021-03-15 | Shipped |
| 2 | Bob | 2019-01-10 | Delivered |
| 4 | Jake | 2020-01-10 | Delivered |
| 5 | Smith | 2022-05-01 | Pending |
| 3 | Johnson | 2023-01-10 | Delivered |
| 6 | Rob | 2023-01-10 | Delivered |

You can display the CUSTOMER_ORDERS table and verify the partition status using the following query –

```
SELECT PARTITION_NAME, TABLE_ROWS
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_NAME='CUSTOMER_ORDERS';
```

Following is the table obtained –

| PARTITION_NAME | TABLE_ROWS |
|----------------|------------|
| P0 | 0 |
| P0 | 3 |
| P1 | 0 |
| P1 | 1 |
| P2 | 0 |
| P2 | 2 |

Vertical Partitioning

The Vertical partitioning divides the table into multiple tables based on columns, rather than rows.

There are two main types of vertical partitioning, each serving specific purposes –

- RANGE Columns Partitioning
- LIST Columns Partitioning

Both Range Columns Partitioning and List Columns Partitioning support various data types, including integer types (TINYINT, SMALLINT, MEDIUMINT, INT, and BIGINT), string types (CHAR, VARCHAR, BINARY, and VARBINARY), as well as DATE and DATETIME data types.

Range Columns Partitioning

The Range Columns partitioning uses one or more columns as partition keys to divide the data into partitions based on a defined range of column values.

The values in these columns are compared to predefined ranges, and each row is assigned to the partition that encompasses the range containing its column values.

Example

In the following query, we are creating a table named **INVENTORY** and dividing it into three partitions based on "product_quantity" and "product_price" columns. Rows with specific values in these columns are stored in their corresponding partitions –

```
CREATE TABLE INVENTORY (
  id INT,
```

```
product_name VARCHAR(50),
product_quantity INT,
product_price int
)
PARTITION BY RANGE COLUMNS(product_quantity, product_price) (
PARTITION P_low_stock VALUES LESS THAN (10, 100),
PARTITION P_medium_stock VALUES LESS THAN (50, 500),
PARTITION P_high_stock VALUES LESS THAN (200, 1200)
);
```

Here, we are inserting rows into the above-created table –

```
INSERT INTO INVENTORY VALUES
(1, 'Headphones', 5, 50),
(2, 'Mouse', 15, 200),
(3, 'Monitor', 30, 300),
(4, 'Keyboard', 60, 600),
(5, 'CPU', 100, 1000);
```

Following is the **INVENTORY** table obtained –

| id | product_name | product_quantity | product_price |
|----|--------------|------------------|---------------|
| 1 | Headphones | 5 | 50 |
| 2 | Mouse | 15 | 200 |
| 3 | Monitor | 30 | 300 |
| 4 | Keyboard | 60 | 600 |
| 5 | CPU | 100 | 1000 |

Now that we have some data in the **INVENTORY** table, we can display the partition status to see how the data is distributed among the partitions using the following query –

```
SELECT PARTITION_NAME, TABLE_ROWS
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_NAME='inventory';
```

You will see in the output below that the respective columns are assigned to their respective partitions based on the defined range values –

| PARTITION_NAME | TABLE_ROWS |
|----------------|------------|
| P_high_stock | 2 |
| P_low_stock | 1 |
| P_medium_stock | 2 |

Displaying Partitions –

We can also display data from specific partitions using the **PARTITION** clause. For instance, to retrieve data from partition **P_high_stock**, we use the following query –

```
SELECT * FROM inventory PARTITION (P_high_stock);
```

It will display all the records in partition P_high_stock –

| ID | NAME | AGE | ADDRESS | SALARY |
|----|----------|-----|---------|--------|
| 4 | Keyboard | 60 | 600 | |
| 5 | CPU | 100 | 1000 | |

Similarly, we can display other partitions using the same syntax.

List Columns Partitioning

The List columns partitioning uses one or more columns as partition keys and assigns records to partitions based on specific values in those columns.

This method is handy when you want to group data into partitions based on discrete values or categories.

Example

Let us create a table named "EMPLOYEES" and partition it using LIST COLUMNS partitioning based on the "department" column –

```
CREATE TABLE EMPLOYEES (
  id INT,
  first_name VARCHAR(50),
  last_name VARCHAR(50),
  hiring_date DATE,
  department VARCHAR(50)
)
PARTITION BY LIST COLUMNS(department) (
  PARTITION p_sales VALUES IN ('Sales', 'Marketing'),
  PARTITION p_engineering VALUES IN ('Engineering', 'Research'),
  PARTITION p_operations VALUES IN ('Operations')
);
```

Here, we are inserting records into above-created table –

```
INSERT INTO EMPLOYEES VALUES
(1, 'John', 'Doe', '2020-01-01', 'Sales'),
(2, 'Jane', 'Doe', '2020-02-01', 'Marketing'),
(3, 'Bob', 'Smith', '2020-03-01', 'Engineering'),
(4, 'Alice', 'Johnson', '2020-04-01', 'Research'),
(5, 'Mike', 'Brown', '2020-05-01', 'Operations');
```

Following is the EMPLOYEES table obtained –

| id | first_name | last_name | hiring_date | department |
|----|------------|-----------|-------------|-------------|
| 1 | John | Doe | 2020-01-01 | Sales |
| 2 | Jane | Doe | 2020-02-01 | Marketing |
| 3 | Bob | Smith | 2020-03-01 | Engineering |
| 4 | Alice | Johnson | 2020-04-01 | Research |
| 5 | Mike | Brown | 2020-05-01 | Operations |

We can display the partition status of the EMPLOYEES table to see how the data is distributed among partitions using the following query –

```
SELECT PARTITION_NAME, TABLE_ROWS
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_NAME='EMPLOYEES';
```

It will display the partitions and the number of rows in each partition based on the department values –

| PARTITION_NAME | TABLE_ROWS |
|----------------|------------|
| p_engineering | 2 |
| p_operations | 1 |
| p_sales | 2 |

Conclusion:

Thus, we studied partitioning queries using MySQL

EXPERIMENT NO-5

| |
|---|
| Title:- To study PL/SQL Functions ,Sequences and Synonyms and PL/SQL Procedure |
|---|

| |
|---|
| Aim:- To implement PL/SQL function, sequences and Synonyms Using Oracle. |
|---|

PL/SQL Function:

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value.

Syntax to create a function:

```

CREATE [OR REPLACE] FUNCTION function_name [parameters]

[(parameter_name [IN | OUT | IN OUT] type [, ...])]

RETURN return_datatype

{IS | AS}

BEGIN

    < function_body >

END [function_name];
  
```

Here:

- **Function_name:** specifies the name of the function.
- **[OR REPLACE]** option allows modifying an existing function.
- The **optional parameter list** contains name, mode and types of the parameters.
- **IN** represents that value will be passed from outside and **OUT** represents that this parameter will be used to return a value outside of the procedure.

The function must contain a return statement.

- **RETURN** clause specifies that data type you are going to return from the function.
- **Function_body** contains the executable part.
- The **AS** keyword is used instead of the **IS** keyword for creating a standalone function.

```

DECLARE
  a number;
  b number;
  c number;
FUNCTION findMax(x IN number, y IN number)
RETURN number
IS
  
```

```

    z number;
BEGIN
    IF x > y THEN
        z:= x;
    ELSE
        Z:= y;
    END IF;
    RETURN z;
END;
BEGIN
    a:= 23;
    b:= 45;
    c := findMax(a, b);
    dbms_output.put_line(' Maximum of (23,45): ' || c);
END;
/

```

PLSQL: Synonyms

A **synonym** is an alternative name for objects such as tables, views, sequences, stored procedures, and other database objects.

Create Synonym (or Replace)

You may wish to create a synonym so that users do not have to prefix the table name with the schema name when using the table in a query.

Syntax

```

CREATE [OR REPLACE] [PUBLIC] SYNONYM [schema .] synonym_name

FOR [schema .] object_name [@ dblink];

```

OR REPLACE

Allows you to recreate the synonym (if it already exists) without having to issue a DROP synonym command.

PUBLIC

It means that the synonym is a public synonym and is accessible to all users. Remember though that the user must first have the appropriate privileges to the object to use the synonym.

schema

The appropriate schema. If this phrase is omitted, Oracle assumes that you are referring to your own schema.

object_name

The name of the object for which you are creating the synonym. It can be one of the following:

- table
- view

- sequence
- stored procedure
- function
- package
- materialized view
- java class schema object
- user-defined object
- synonym

PL/SQL Procedure

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

- **Header:** The header contains the name of the procedure and the parameters or variables passed to the procedure.
- **Body:** The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

How to pass parameters in procedure:

When you want to create a procedure or function, you have to define parameters. There are three ways to pass parameters in procedure:

1. **IN parameters:** The IN parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or the function.
2. **OUT parameters:** The OUT parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.
3. **INOUT parameters:** The INOUT parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

PL/SQL Create Procedure

Packages in PL/SQL:-

A package is a way of logically storing the subprograms like procedures, functions, exception or cursor into a single common unit.

A package can be defined as an oracle object that is compiled and stored in the database.

Once it is compiled and stored in the database it can be used by all the users of database who have executable permissions on Oracle database.

Components of Package

Package has two basic components:

- Specification: It is the declaration section of a Package
- Body: It is the definition section of a Package.

Benefits of using Package

Following are some of the benefits of packages in PL/SQL:

1. REUSABILITY

Whenever a package is created, it is compiled and stored in the database. So, you write the code once which can be reused by other applications.

2. OVERLOADING

Two or more >procedures or functions can be created in a package with the same name.

3. CREATING MODULES

A large application can be created by simply creating modules(or subprograms) clearly defined and easy to work.

4. IMPROVES PERFORMANCE

Package code gets loaded inside the SGA(system global area) of Oracle at first call itself due to which other subsequent calls will work very fast.

5. GLOBAL DECLARATION

If the objects(procedures, functions, variables, constants, exceptions, cursor etc) are declared globally in a package, they can be easily used when required.

How to create a PL/SQL Package?

Following are the steps to declare and use a package in PL/SQL code block:

STEP 1: Package specification or declaration

It mainly comprises of the following:

- Package Name.
- Variable/constant/cursor/procedure/function/exception declaration.
- This declaration is global to the package.

Here is the syntax:

```
CREATE OR REPLACE PACKAGE <package_name> IS/AS
    FUNCTION <function_name> (<list of arguments>)
    RETURN <datatype>;
    PROCEDURE <procedure_name> (<list of arguments>);
```

-- code statements

END <package_name>;

Copy

where,

CREATE OR REPLACE PACKAGE are keywords used to create a package

FUNCTION and PROCEDURE are keywords used to declare function and procedure while creating package.

<package_name>, <function_name>, <procedure_name> are user-defined names.

IS/AS are keywords used to declare package.

RETURN is a keyword specifying value returned by the function declared.

STEP 2: Package Body

It mainly comprises of the following:

- It contains the definition of procedure, function or cursor that is declared in the package specification.
- It contains the subprogram bodies containing executable statements for which package has been created

Here is the syntax:

CREATE OR REPLACE PACKAGE BODY <package_name> IS/AS

FUNCTION <function_name> (<list of arguments>) RETURN <datatype>IS/AS

-- local variable declaration;

BEGIN

-- executable statements;

EXCEPTION

-- error handling statements;

END <function_name>;

PROCEDURE <procedure_name> (<list of arguments>)IS/AS

-- local variable declaration;

BEGIN

-- executable statements;

EXCEPTION

-- error handling statements;

END <procedure_name>;

END <package_name>;

Copy

Where,

CREATE OR REPLACE PACKAGE BODY are keywords used to create the package with a body.

FUNCTION and PROCEDURE are keywords used to define function and procedure while creating package.

<package_name>, <function_name>, <procedure_name> are user-defined.

IS/AS are keywords used to define the body of package, function and procedure.

RETURN is a keyword specifying value returned by the function defined.

DECLARE, BEGIN, EXCEPTION, END are the different sections of PL/SQL code block containing variable declaration, executable statements, error handling statements and marking end of PL/SQL block respectively where DECLARE and EXCEPTION part are optional.

IMPLEMENTATION:

//Creating staff table and inserting data

```
create table staff (sidint,sname varchar2(20),dname varchar2(20));
```

```
begin
```

```
insert into staff values(1,'ATS','CS');
```

```
insert into staff values(2,'AVS','CS');
```

```
insert into staff values(3,'VJY','Civil');
```

```
insert into staff values(4,'PRP','Mech');
```

```
end;
```

//Creating Procedure

```
create or replace procedure employees (name in staff.dname%type,counter out int) as
```

```
begin
```

```
select count(*) into counter from staff where dname=name;
```

```
end;
```

//Calling Procedure

```
declare
```

```
anint;
```

```

begin
employees('Civil',an);
dbms_output.put_line('Total employees '||an);
end;

//Creating Package

create or replace package pack as

procedureadd_staff(id in staff.sid%type,name in staff.sname%type, dept in staff.dname%type);

end pack;

//Creating Package Body

create or replace package body pack as

procedureadd_staff(id in staff.sid%type,name in staff.sname%type, dept in staff.dname%type) is

begin

insert into staff values(id,name,dept);

endadd_staff;

end pack;

//Calling Procedure of Package

begin

pack.add_staff(6,'SVA','Chem');

end;
```

CREATE SEQUENCE

A sequence is a database object that is used to generate a unique integer, which is often used to populate a synthetic key. Sequences are created using the CREATE SEQUENCE command. The vast majority of the time you will just specify a sequence name and use the defaults values for all sequence attributes, or maybe increase the CACHE attribute above the default value of 20 to improve performance. If the schema isn't specified explicitly, it assumes you mean a sequence in the current schema.

```
CREATE SEQUENCE my_seq_1;
```

```
CREATE SEQUENCE my_seq_2 CACHE 50;
```

The CREATE SEQUENCE documentation lists all the available sequence attributes. The example below uses MINVALUE to set the starting point of the sequence, MAXVALUE to set the end point of the sequence, CYCLE to tell it to go back to the start once all available sequences are used and INCREMENT BY to make it increase in steps of 10. The queries show the impact of this.

```

CREATE SEQUENCE my_seq_3
  INCREMENT BY 10
  MINVALUE 10
  MAXVALUE 30
  CYCLE;
SQL> SELECT my_seq_3.NEXTVAL FROM dual;
      NEXTVAL
-----
         10
1 row selected.
SQL> SELECT my_seq_3.NEXTVAL FROM dual;
      NEXTVAL
-----
        20
SQL> SELECT my_seq_3.NEXTVAL FROM dual;
      NEXTVAL
-----
        30
SQL> SELECT my_seq_3.NEXTVAL FROM dual;
      NEXTVAL
-----
        10
SQL>

```

By default a sequence is global, so its value is maintained across all sessions, from any user that has privilege to select from it. From Oracle 12c onward sequences can be defined as session-specific, so their current value is only relevant to the current session, and effectively reset for each new session.

Session Sequences in Oracle Database 12c Release 1 (12.1)

Oracle 18c introduced the concept of scalable sequences.

Scalable Sequences in Oracle Database 18c

ALTER SEQUENCE

Many of the sequence attributes can be altered after creation using the ALTER SEQUENCE command. The full list of sequence attributes that can be altered are listed in the ALTER SEQUENCE documentation. If the schema isn't specified explicitly, it assumes you mean a sequence in the current schema. The following example alters some of the attributes of a sequence created in the previous section.

```

ALTER SEQUENCE my_seq_3
  INCREMENT BY 1
  MINVALUE 1
  MAXVALUE 1000000
  NOCYCLE;

```

DROP SEQUENCE

A sequence is dropped using the DROP SEQUENCE command. If the schema isn't specified explicitly, it assumes you mean a sequence in the current schema.

DROP SEQUENCE my_seq_1;

DROP SEQUENCE my_seq_2;

DROP SEQUENCE my_schema.my_seq_3;

Output:

```

User: SYSTEM
Home > SQL > SQL Commands

Autocommit Display 10

declare
mks t_111.marks%type;
begin
select marks into mks from t_111 where id=2;
dbms_output.put_line('marks are '||mks);
end;

Results Explain Describe Saved SQL History

marks are 80
Statement processed.

0.00 seconds

```

```

User: SYSTEM
Home > SQL > SQL Commands

Autocommit Display 10

insert into t_111 values (4,50);
end;
declare
a int;
begin
select max(marks) into a from t_111;
dbms_output.put_line('maximum marks are '||a);
a:=a+5;
dbms_output.put_line('maximum marks are with additional 5 are '||a);
end;

Results Explain Describe Saved SQL History

maximum marks are 90
maximum marks are with additional 5 are 95
Statement processed.

0.02 seconds

```

```

Autocommit Display 10

create table t_112(id int,marks int);
begin
insert into t_112 values (1,90);
insert into t_112 values (2,70);
insert into t_112 values (3,60);
insert into t_112 values(4,90);
end;
create synonym student_info for t_112
select * from student_info

Results Explain Describe Saved SQL Hist

ID MARKS
1 90
2 70
3 60
4 90
1 90
2 70
3 60
4 90
8 rows returned in 0.01 seconds CSV Export

```

```

Autocommit Display 10

create table t_112(id int,marks int);
begin
insert into t_112 values (1,90);
insert into t_112 values (2,70);
insert into t_112 values (3,60);
insert into t_112 values(4,90);
end;

create synonym student_info for t_112

Results Explain Describe Saved SQL Hist

Synonym created.

0.06 seconds

```

```

User: SYSTEM
Home > SQL > SQL Commands

Autocommit Display 10

create table t_111(id int,marks int);
begin
insert into t_111 values (1,90);
insert into t_111 values (2,80);
insert into t_111 values (3,70);
insert into t_111 values (4,50);
end;

Results Explain Describe Saved SQL Hist

Statement processed.

0.00 seconds

```

```

Home > SQL > SQL Commands

Autocommit Display 10

create or replace function find_marks(pn in t_111.id%type) return int
as
mks t_111.marks%type;
begin
select marks into mks from t_111 where id=pn;
return mks;
end;

Results Explain Describe Saved SQL History

Function created.

0.00 seconds

```

Conclusion:

Thus we studied PL/SQL functions ,Sequences and Synonyms using Oracle.

EXPERIMENT NO: 6

Title: Write and execute simple PL/SQL programs that demonstrate the use of all types of triggers and cursors.

Aim: To Study and implement PL/SQL Triggers and Cursors.

THEORY:-**PL/SQL CursorsTypes:****1. ImplicitCursors:**

Implicit cursors are automatically created and used by RDBMS every time you issue a select statement in PL/SQL. If you use an Implicit cursors, underlying RDBMS will perform the open, fetches, and close for you automatically. Implicit cursors are used in statements that return only one row.

If the SQL statement returns more than one row, an error will occur.

In the following PL/SQL code block, the select statement makes use of an implicit cursor:

Example:

Begin

Update emp Set empno=empno Where 1=2;

SELECT CONCAT(sql%rowcount ' ' rows are affected by the update statement'); End;

Mostly these type of cursors work fine in Oracle but not compatible with MySQL.

2. ExplicitCursors

Explicit cursors are created by the programmer, and with these you can do operations on a set of rows, which can be processed one by one. You use explicit cursors when you are sure that the SQL statement will return more than one row. You have to declare an explicit cursors in the declare section at the beginning of the PL/SQL block.

Once you declare your cursor, the explicit cursors will go through these steps:

- 1. Declare:** This clause initializes the cursor into memory.
- 2. Open:** the previously declared cursor is now open and memory is allotted.
- 3. Fetch:** The previously declared and opened cursor can now access data;
- 4. Close:** The previously declared, opened, and fetched cursor is closed, which also releases memory allocation.

Examples of this type of cursor have already been covered in last assignment.

Triggers

Triggers are simply stored procedures that are ran automatically by the database whenever some event (usually a table update) happens. If you know how to write stored procedures, you already know how to writetriggers.

PL/SQL Triggers

Triggers are basically PL/SQL procedures that are associated with tables, and are called whenever a certain modification (event) occurs. The modification statements may include INSERT, UPDATE, and DELETE.

SYNTAX:

```
CREATE
TRIGGER trigger_name BEFORE (or AFTER)
INSERT OR UPDATE [OF COLUMNS] OR DELETE
ON tablename
[FOR EACH ROW [WHEN (condition)]]
```

```
BEGIN
... END;
```

Example:

```
CREATE TRIGGER
PERSON_DELETE_AFTER AFTER

DELETE ON
PERSON FOR
EACH ROW
BEGIN

INSERT INTO PERSON_BACKUP VALUES (OLD.ID,OLD.NAME,OLD.SALARY);
END;
```

Program:

```
CREATE TABLE lecturer (id int,first_name CHAR(20),last_name CHAR(20),major
CHAR(30),current_creditsint);
```

```
INSERT INTO lecturer (id, first_name, last_name, major,current_credits)VALUES (10001,
'Scott', 'Lawson','Computer Science', 11);
```

```
INSERT INTO lecturer (id, first_name, last_name, major, current_credits)VALUES(10002,
'Mar', 'Wells','History', 4);
```

```
INSERT INTO
lecturer(id,first_name,last_name,major,current_credits)VALUES(10003,'Jone','Bliss','Computer
Science', 8);
```

```
INSERT INTO lecturer (id, first_name, last_name, major,current_credits)VALUES (10004,
'Man', 'Kyte','Economics', 8);
```

```
INSERT INTO lecturer (id, first_name, last_name, major,current_credits) VALUES (10005,
'Pat', 'Poll','History', 4);
```

```
INSERT INTO lecturer (id, first_name, last_name, major,current_credits)VALUES (10006,
'Tim', 'Viper','History', 4);
```

```
INSERT INTO lecturer (id, first_name, last_name, major,current_credits) VALUES (10007,
'Barbara', 'Blues','Economics', 7);
```

```
INSERT INTO lecturer (id, first_name, last_name, major,current_credits) VALUES (10008,
'David', 'Large','Music', 4);
```

```
INSERT INTO lecturer (id, first_name, last_name, major,current_credits) VALUES (10009,
'Chris', 'Elegant','Nutrition', 8);
```

```
INSERT INTO lecturer (id, first_name, last_name, major,current_credits)VALUES (10010,
'Rose', 'Bond','Music', 7);
```

```
INSERT INTO lecturer (id, first_name, last_name, major,current_credits)VALUES (10011, 'Rita',
'Johnson','Nutrition', 8);
```

```
INSERT INTO lecturer (id, first_name, last_name, major,current_credits)VALUES (10012,
'Sharon', 'Clear','Computer Science', 3);
```

```
CREATE TABLE major_stats( major CHAR(30),total_creditsNUMBER,total_lecturer
NUMBER);
```

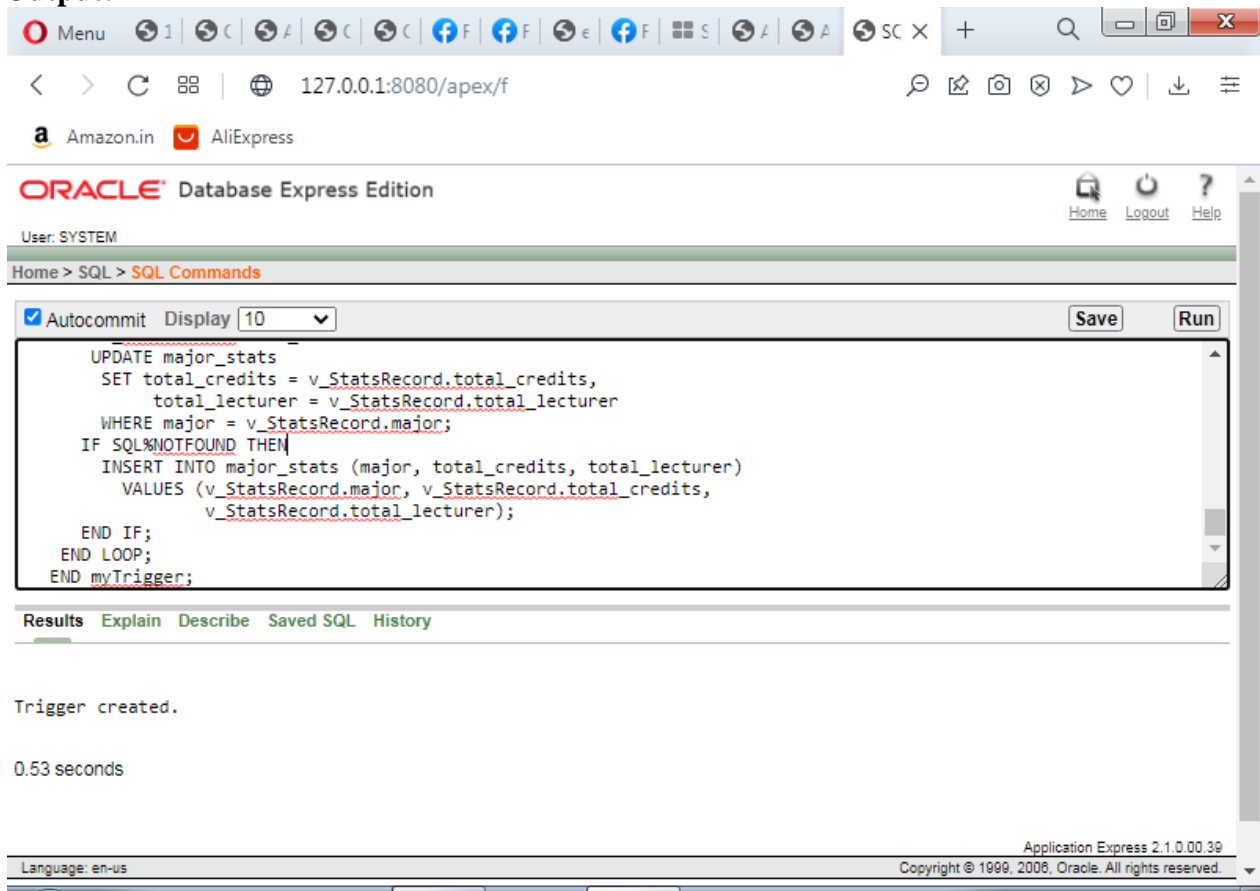
```
CREATE OR REPLACE TRIGGER myTrigger
  AFTER INSERT OR DELETE OR UPDATE ON lecturer
  DECLARE
    CURSOR c_Statistics IS
      SELECT major, COUNT(*) total_lecturer,
SUM(current_credits) total_credits
      FROM lecturer
      GROUP BY major;
  BEGIN
    FOR v_StatsRecord in c_Statistics LOOP
      UPDATE major_stats
      SET total_credits = v_StatsRecord.total_credits,
total_lecturer = v_StatsRecord.total_lecturer
      WHERE major = v_StatsRecord.major;
      IF SQL%NOTFOUND THEN
        INSERT INTO major_stats (major, total_credits, total_lecturer)
        VALUES (v_StatsRecord.major, v_StatsRecord.total_credits,
v_StatsRecord.total_lecturer);
```

```

END IF;
END LOOP;
END myTrigger;

```

Output:



Conclusion:

Thus we studied PL/SQL programme use of triggers& cursors.

EXPERIMENT NO: 07**Title:** To install MongoDB on windows.**Aim:** To install MongoDB on windows**Theory :-****Pre-Requisite Information On Windows**

The users of Windows must know that their windows desktop has got one of the two versions i.e. 32-bit & 64-bit.

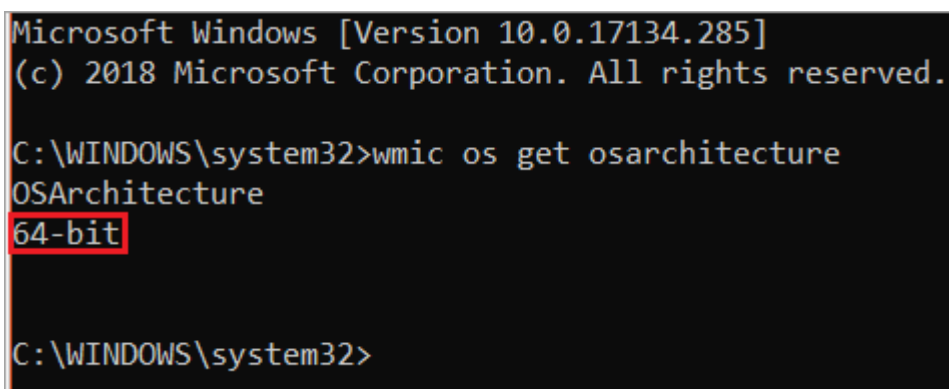
This information could be found out in the properties of one's "My Computer" or "This PC" on their device i.e. either their windows is 32-bit or 64-bit.

Further reading =>> Is My Windows 32 bit Or 64 bit

Meanwhile, in order to check the windows version, one can also use command prompt in the way as narrated in the snippet below:

The **command** is **C:\>wmic os get osarchitecture**

Figure 1: Command To Know Windows Version



```
Microsoft Windows [Version 10.0.17134.285]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>wmic os get osarchitecture
OSArchitecture
64-bit

C:\WINDOWS\system32>
```

After finding this out, the 32-bit windows users would follow the guide to install the version of MongoDB which supports 32-bit and vice versa. MongoDB is available in both the versions which support their respective 32-bit & 64-bit windows.

For instance, 32-bit windows users have got the advantage of having qualitative development and testing environments.

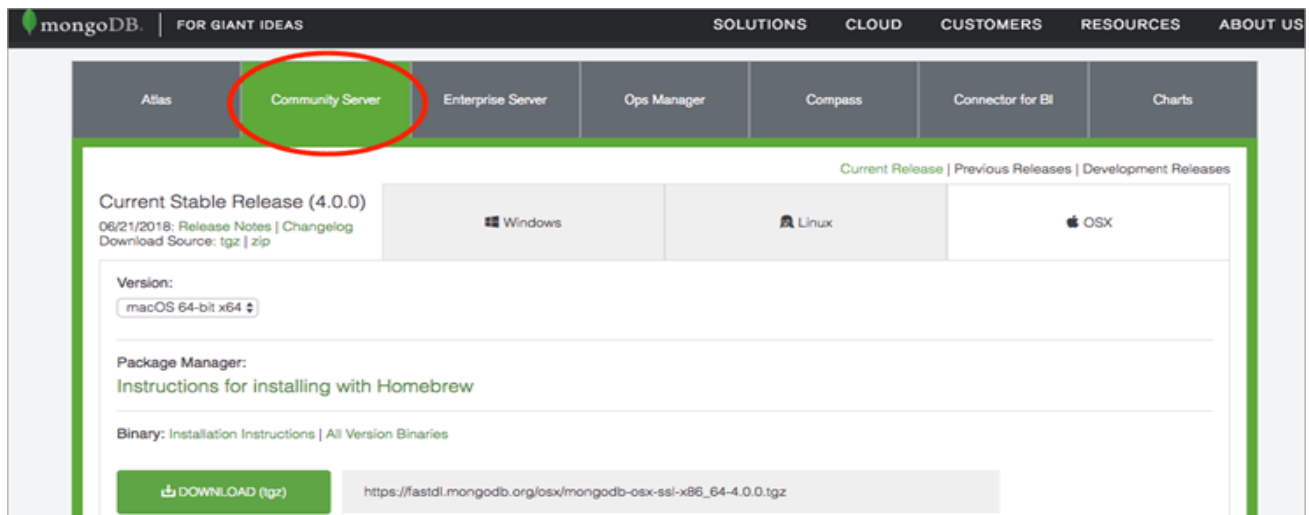
Meanwhile, if one must get into production environments, then they must have to adopt 64-bit windows, because in this case, using 32-bit would limit the data usage which would be stored in MongoDB. Therefore, the 32-bit windows version supports the MongoDB version which has the database size lesser than 2 GigaBytes.

MongoDB Download On Windows

Click on the following link to **Download MongoDB On Windows**

Figure 2: To Get the Download File of MongoDB from Website

OR



Click on any one of the following links which suit your Windows Version.

Windows [64-bit]

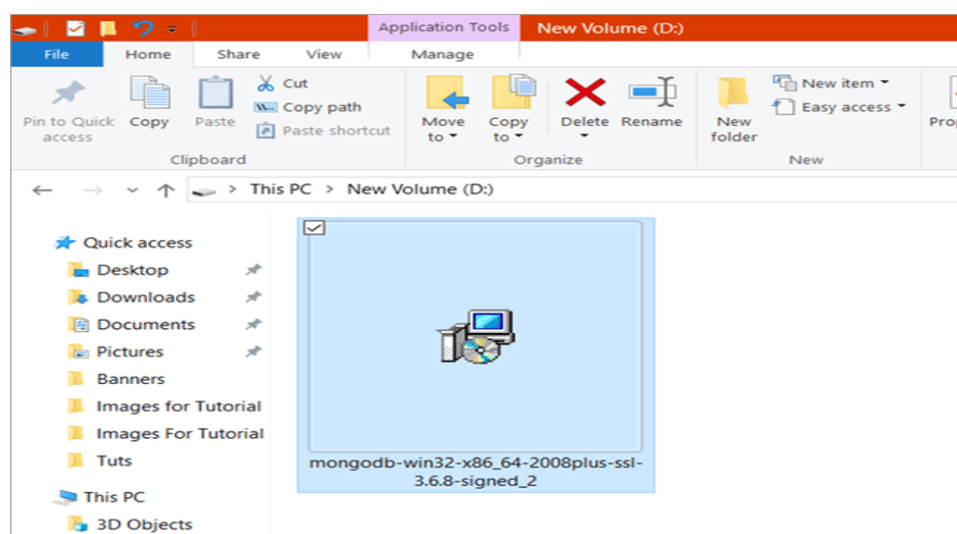
Windows [32-bit]

Installation Guide On Windows

Follow the below steps to install the proposed file:

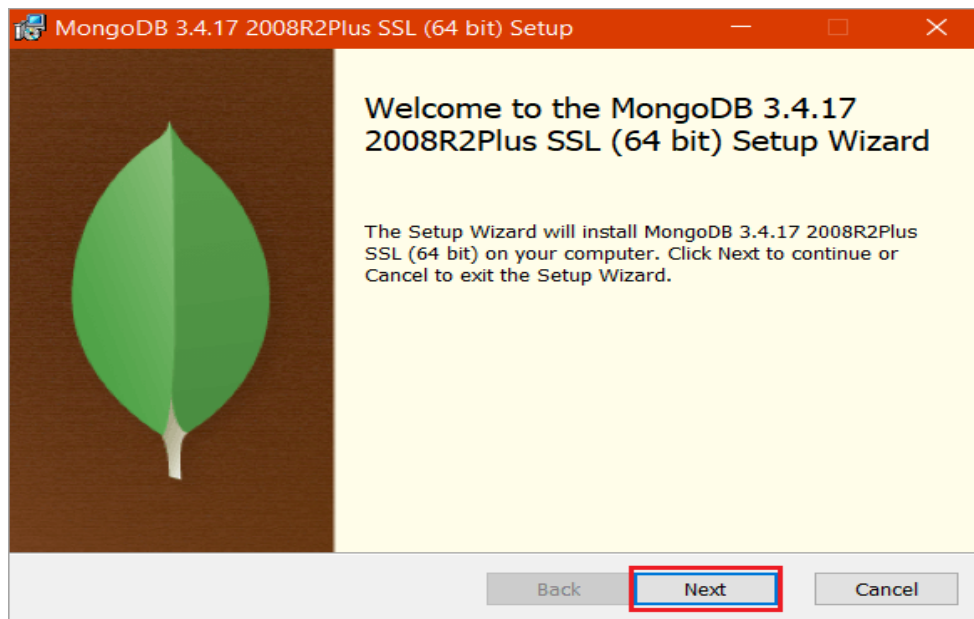
Step 1: Open the file. We have installed for 64-bit version with the name as “*MongoDB-win32-x86_64-2008plus-ssl-v3.4-latest-signed*”. It is saved in the *Local Disk C:/*, click on the file where you’ve saved it to start the wizard.

Figure 3:



Step 2: Click “Next”.

Figure 4

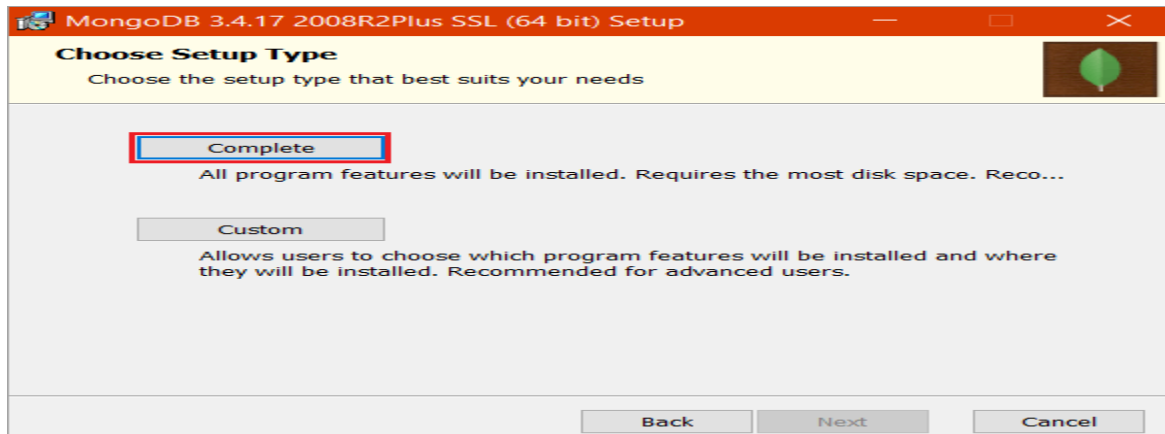


Step 3: Tick the check box next to ‘I accept the terms in the License Agreement’ and again click on “Next”.



Step 4: Click “Complete” to install all the features of MongoDB. As for “Custom”, this option would be used to install only the specific components of MongoDB and also if a user wants to change the location of where the installation must be done.

Figure 6



Step 5: Click “Install” to begin the installation drive.

Figure 7

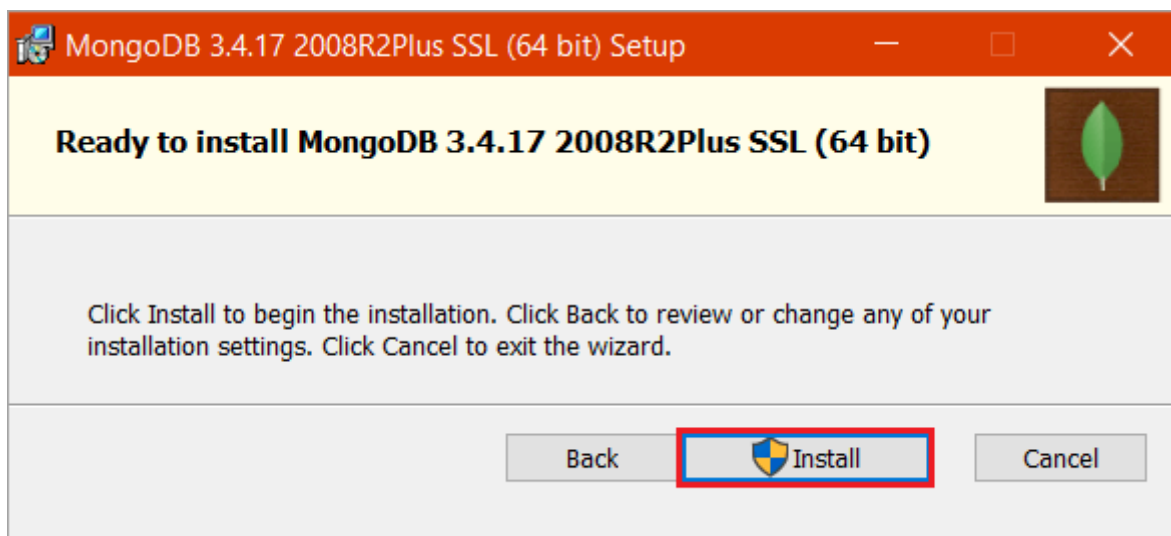
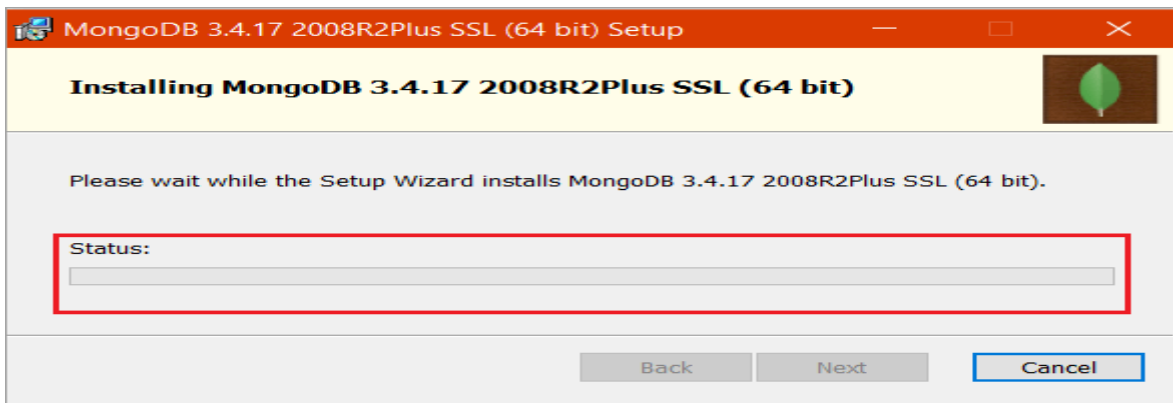
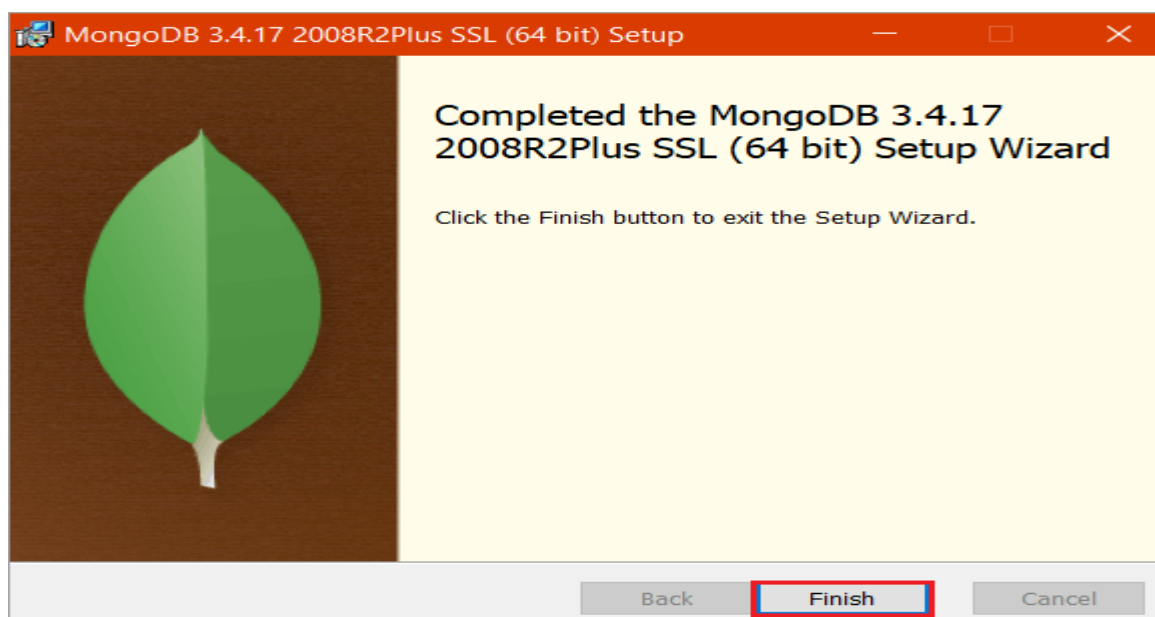


Figure 8



Step 6: After the installation has been finished. Simply, click “*Finish*”.

Figure 9



Configuration Through Mongo Shell

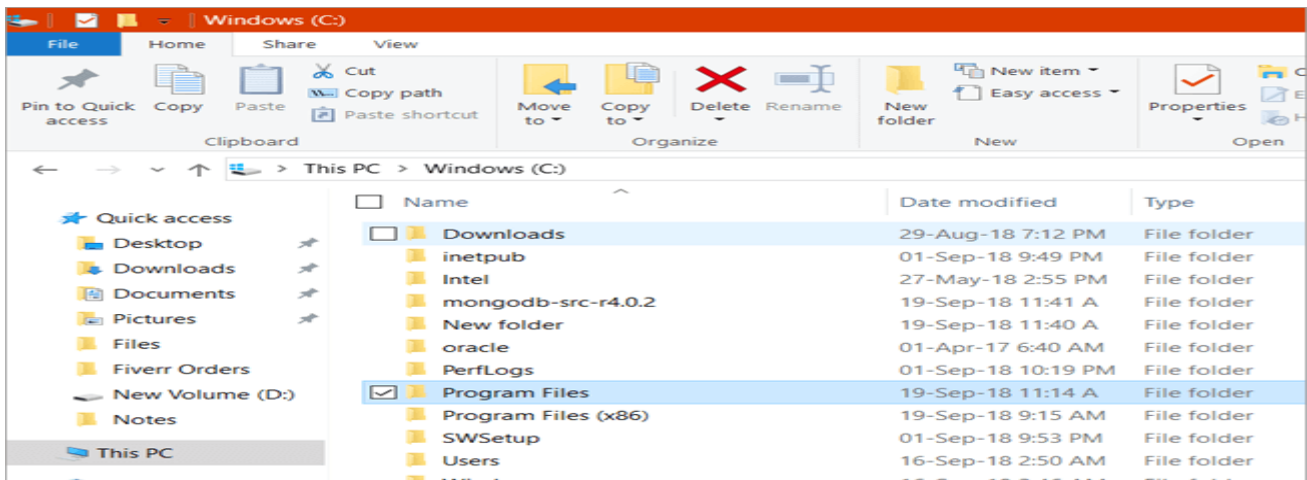
After the whole installation is done, the user must configure it.

Follow the below steps:

If you use “mongo” in the command prompt without configuring it, then it would give an error. Hence, configure it first.

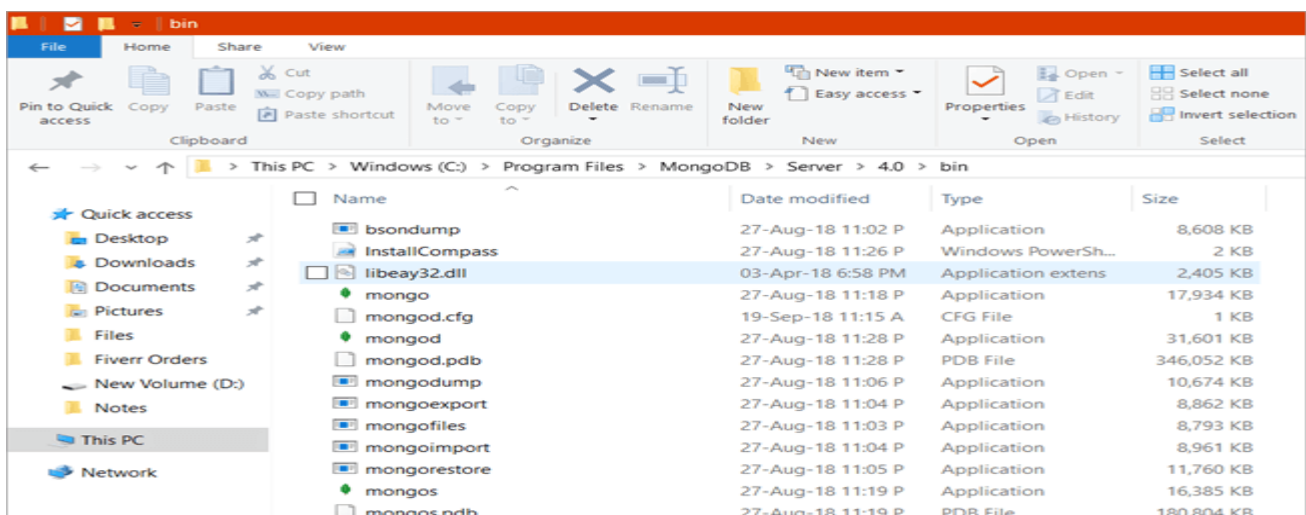
Step 1: Go to the local disk C and get into “*Program Files*”. There you’ll find a folder named “*MongoDB*”.

Figure 10



Step 2: Open it and you'll find a folder named "bin" i.e. binaries folder. You will have 15 to 17 files in it. Copy the path, as given in the snippet path i.e. `C:\Program Files\MongoDB\Server\4.0\bin`

Figure 11

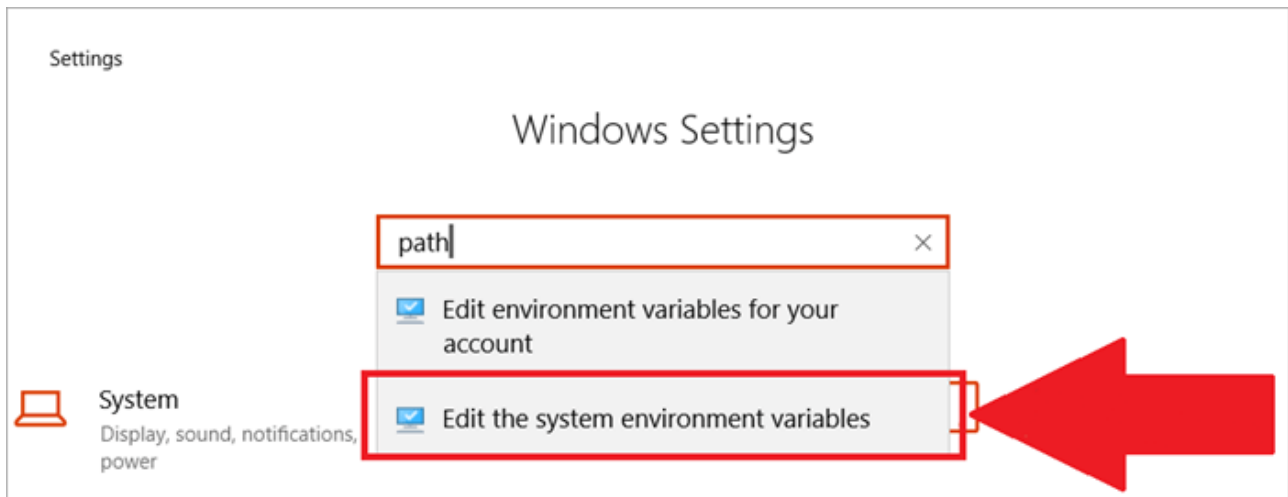


Step 3: Open Settings and search "Path".

The two options given below would pop up in front of you:

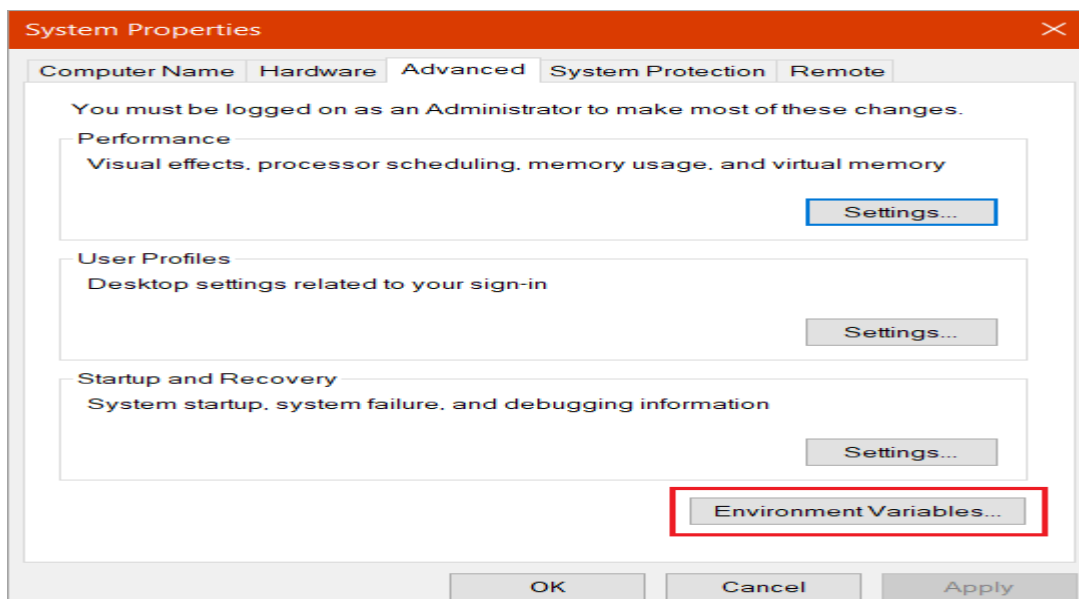
1. Edit environment variable of your account
2. Edit the system environment variable.

Figure 12



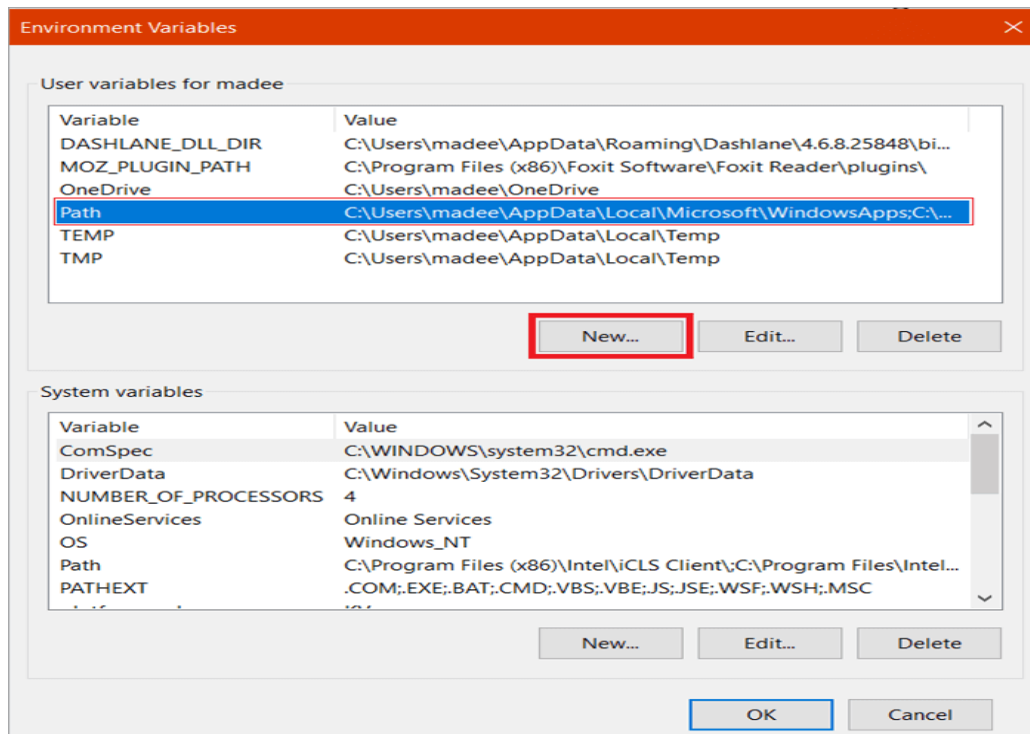
Step 4: Click on “*Edit the system environment variable*” and then click on “*Environment Variables*”.

Figure 13



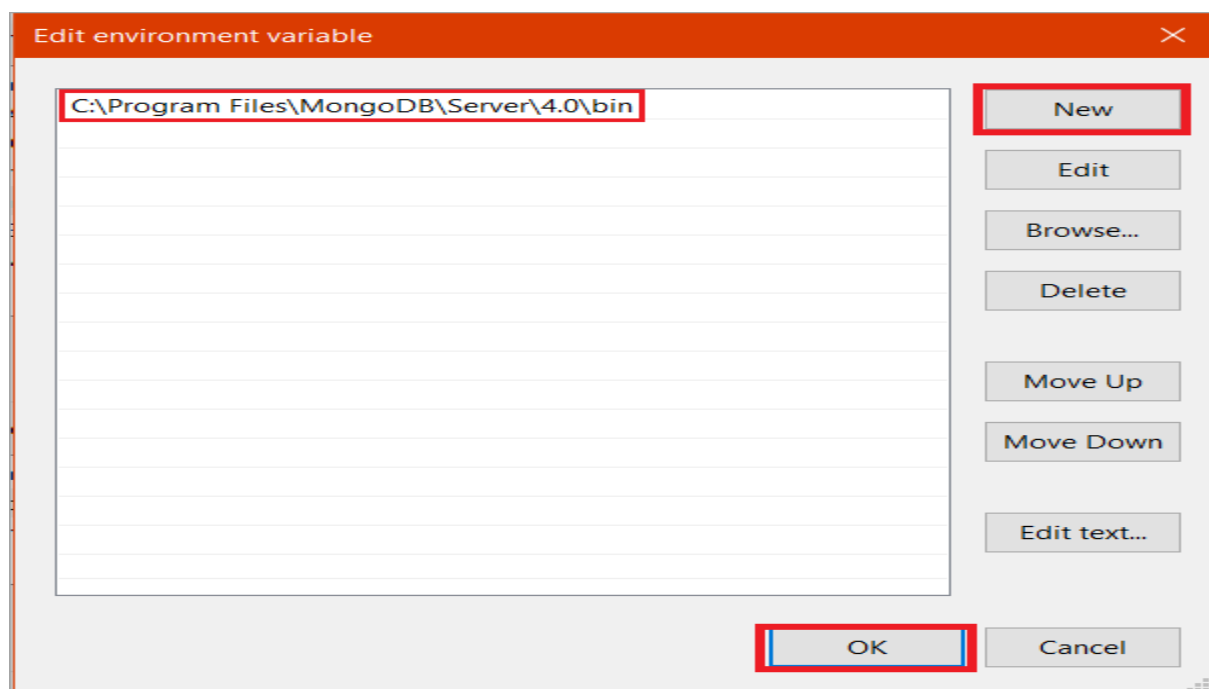
Step 5: In the Environment variable, you’ll see the path as given in the snippet. Click “*Path*” and then press “*Edit*”.

Figure 14



Step 6: In the Path given at your system, delete the previous ones and add new "The copied path" from binaries, and click "OK".

Figure 15



Step 7: Open Command prompts and type "*mongod*" to start the service.

Figure 16

```

Administrator: Command Prompt - mongod
Microsoft Windows [Version 10.0.17134.286]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd C:\Program Files\MongoDB\Server\3.6\bin

C:\Program Files\MongoDB\Server\3.6\bin>mongod
2018-09-20T10:08:59.240-0700 I CONTROL [initandlisten] MongoDB starting : pid=5532 port=27017 dbpath=C:\data\db\ 64-bit host=DESKTOP-GI9II6G
2018-09-20T10:08:59.240-0700 I CONTROL [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2018-09-20T10:08:59.242-0700 I CONTROL [initandlisten] db version v3.6.8
2018-09-20T10:08:59.242-0700 I CONTROL [initandlisten] git version: 6bc9ed599c3fa164703346a22bad17e33fa913e4
2018-09-20T10:08:59.242-0700 I CONTROL [initandlisten] OpenSSL version: OpenSSL 1.0.2o-fips 27 Mar 2018
2018-09-20T10:08:59.242-0700 I CONTROL [initandlisten] allocator: tcmalloc
2018-09-20T10:08:59.242-0700 I CONTROL [initandlisten] modules: none
2018-09-20T10:08:59.242-0700 I CONTROL [initandlisten] build environment:
2018-09-20T10:08:59.242-0700 I CONTROL [initandlisten]   distmod: 2008plus-ssl
2018-09-20T10:08:59.242-0700 I CONTROL [initandlisten]   distarch: x86_64
2018-09-20T10:08:59.242-0700 I CONTROL [initandlisten]   target_arch: x86_64
2018-09-20T10:08:59.242-0700 I CONTROL [initandlisten] options: {}
2018-09-20T10:08:59.279-0700 I - [initandlisten] Detected data files in C:\data\db\ created by the 'wiredTiger' storage engine, so setting the active stor
2018-09-20T10:08:59.299-0700 I STORAGE [initandlisten] wiredtiger_open config: create,cache_size=7616M,session_max=20000,eviction=(threads_min=4,threads_max=4),
false,compatibility=(release="3.0",require_max="3.0"),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000),stat
2018-09-20T10:08:59.607-0700 I STORAGE [initandlisten] WiredTiger message [1537463339:607254][5532:140715831147600], txn-recover: Main recovery loop: starting a
2018-09-20T10:08:59.731-0700 I STORAGE [initandlisten] WiredTiger message [1537463339:730921][5532:140715831147600], txn-recover: Recovering log 4 through 5
2018-09-20T10:08:59.842-0700 I STORAGE [initandlisten] WiredTiger message [1537463339:841836][5532:140715831147600], txn-recover: Recovering log 5 through 5
2018-09-20T10:08:59.909-0700 I STORAGE [initandlisten] WiredTiger message [1537463339:908554][5532:140715831147600], txn-recover: Set global recovery timestamp:
2018-09-20T10:09:00.491-0700 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-09-20T10:09:00.493-0700 I CONTROL [initandlisten] **      Read and write access to data and configuration is unrestricted.
2018-09-20T10:09:00.493-0700 I CONTROL [initandlisten] ** WARNING: This server is bound to localhost.
2018-09-20T10:09:00.496-0700 I CONTROL [initandlisten] **      Remote systems will be unable to connect to this server.
2018-09-20T10:09:00.497-0700 I CONTROL [initandlisten] **      Start the server with --bind_ip <address> to specify which IP
2018-09-20T10:09:00.497-0700 I CONTROL [initandlisten] **      addresses it should serve responses from, or with --bind_ip_all to
2018-09-20T10:09:00.498-0700 I CONTROL [initandlisten] **      bind to all interfaces. If this behavior is desired, start the
2018-09-20T10:09:00.498-0700 I CONTROL [initandlisten] **      server with --bind_ip 127.0.0.1 to disable this warning.
2018-09-20T10:09:00.498-0700 I CONTROL [initandlisten] Initializing full-time diagnostic data capture with directory 'C:\data\db\diagnostic.data'
2018-09-20T10:09:01.110-0700 I NETWORK [initandlisten] waiting for connections on port 27017

```

Step 8: Write command on the command prompt "*mongo*" to create the connection.

Figure 17

```

Command Prompt - mongo
C:\Program Files\MongoDB\Server\3.6\bin>mongo
MongoDB shell version v3.6.8
connecting to: mongod://127.0.0.1:27017
MongoDB server version: 3.6.8
Server has startup warnings:
2018-09-20T10:09:00.491-0700 I CONTROL [initandlisten]
2018-09-20T10:09:00.491-0700 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2018-09-20T10:09:00.493-0700 I CONTROL [initandlisten] **      Read and write access to data and configuration is u
nrestricted.
2018-09-20T10:09:00.493-0700 I CONTROL [initandlisten]
2018-09-20T10:09:00.496-0700 I CONTROL [initandlisten] ** WARNING: This server is bound to localhost.
2018-09-20T10:09:00.496-0700 I CONTROL [initandlisten] **      Remote systems will be unable to connect to this ser
ver.
2018-09-20T10:09:00.497-0700 I CONTROL [initandlisten] **      Start the server with --bind_ip <address> to specify
which IP
2018-09-20T10:09:00.497-0700 I CONTROL [initandlisten] **      addresses it should serve responses from, or with --
bind_ip_all to
2018-09-20T10:09:00.498-0700 I CONTROL [initandlisten] **      bind to all interfaces. If this behavior is desired,
start the
2018-09-20T10:09:00.498-0700 I CONTROL [initandlisten] **      server with --bind_ip 127.0.0.1 to disable this warn
ing.
2018-09-20T10:09:00.498-0700 I CONTROL [initandlisten]
>

```

Step 9: Open Command prompt with admin privileges and type: **md \data\db** to make a directory which would be \data\db at the same folder.

Figure 18

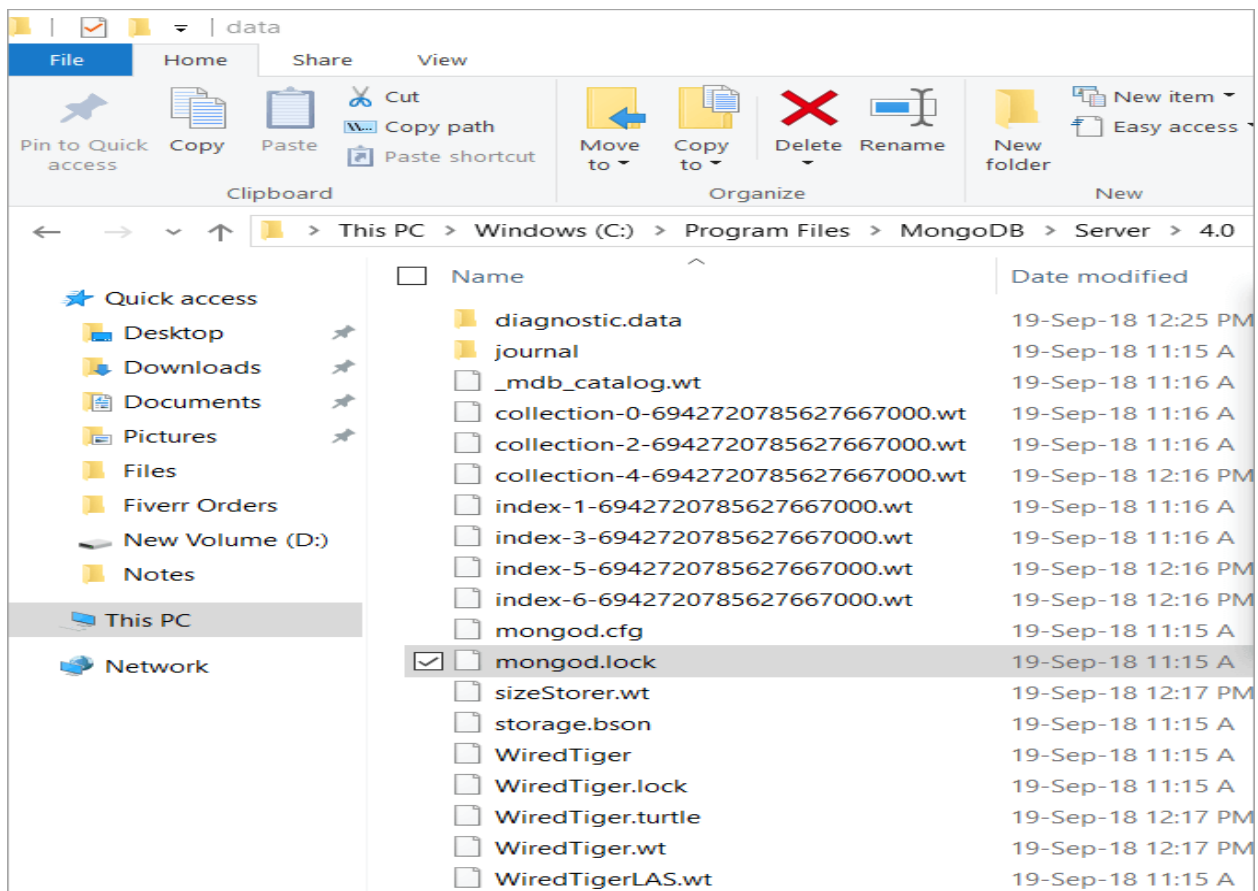
```

Administrator: Command Prompt
C:\Windows\System32>md \data\db

```

Then, Get into the folder “data”, which would be in the same folder i.e. MongoDB at C:/ and then find the file named “mongod.lock”.

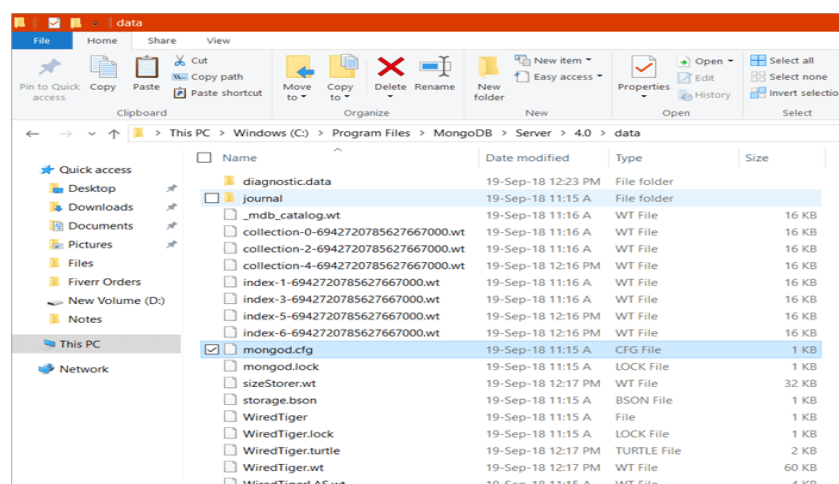
Figure 19



Step 10: Again, open Binaries i.e. “Bin” and find “mongod.cfg file”, if you find it, then copy this file and paste it in the folder named “data” which is also in the MongoDB folder in C:/ . If it is not here then finding in the “data” folder might help, usually, it happens to be there.

Copy the path which is “C:\Program Files\MongoDB\Server\4.0\data” in your system.

Figure 20



Step 11: Open Command prompt with admin privileges and write this command: ***mongod –config “C:\ data\mongod.cfg” –install*** Press enter.

Step 12: Again type the command: ***net start mongoDb***

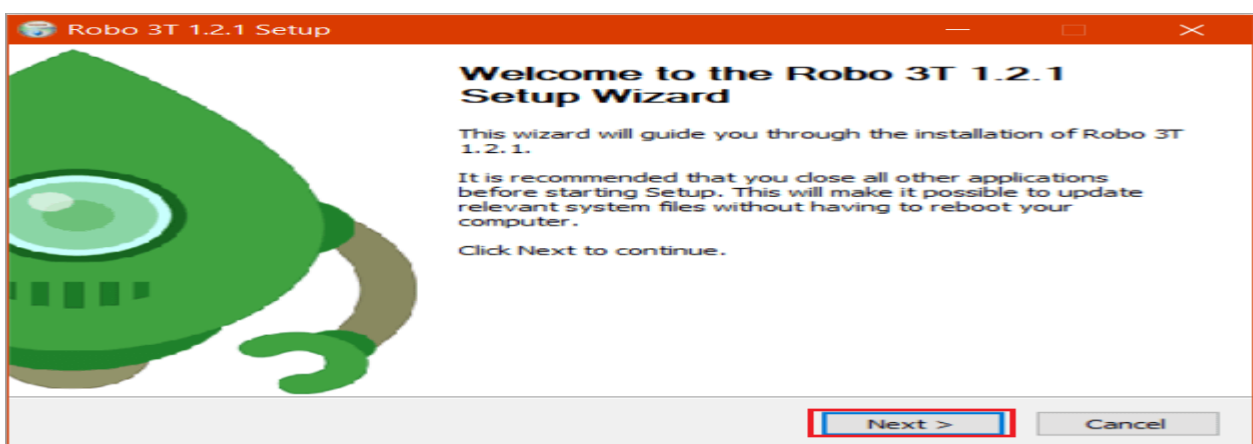
The MongoDB pops up with a message “*The MongoDB Service was started successfully*”. You are now good on the go. MongoDB has been successfully configured.

Configuration Through Robomongo i.e. Robo 3t

To configure MongoDB with non-commercial MongoDB management tool i.e. Robo 3t, one can easily use the full command on the mongo shell during configuration.

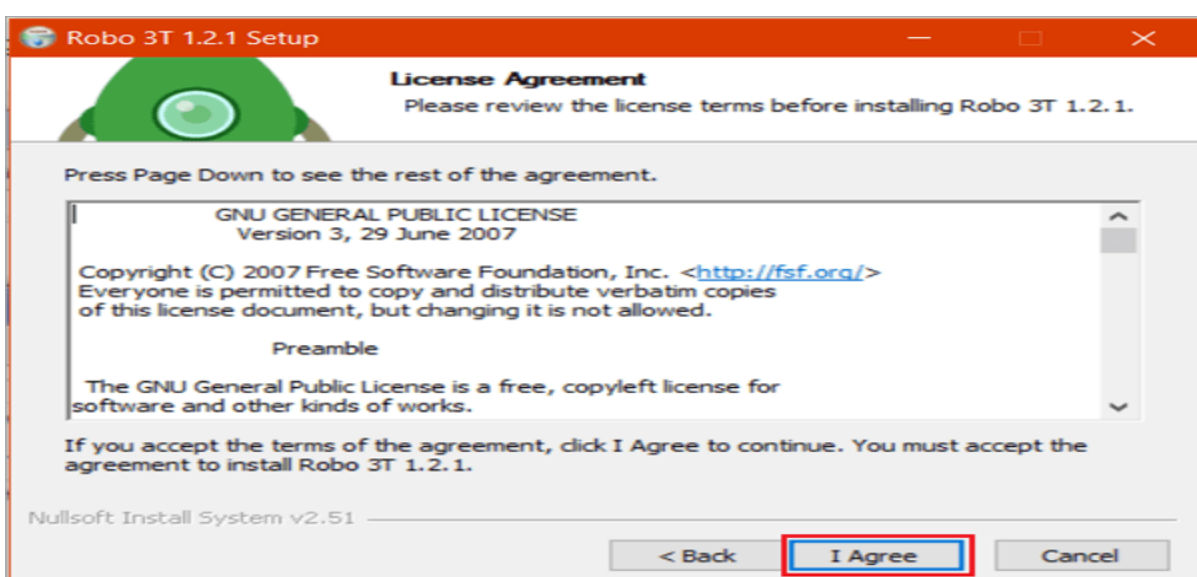
Step 1: Download Robo 3t and launch the installer.

Figure 21



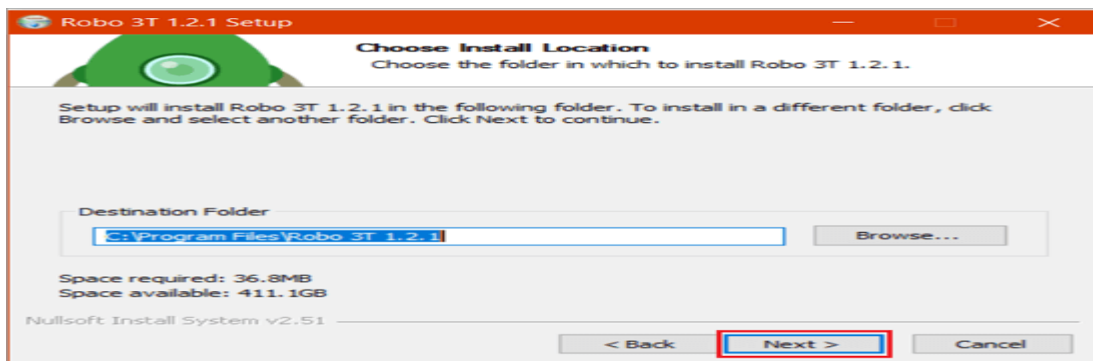
Step 2: Click “I agree” to agree to the End-user agreement.

Figure 22



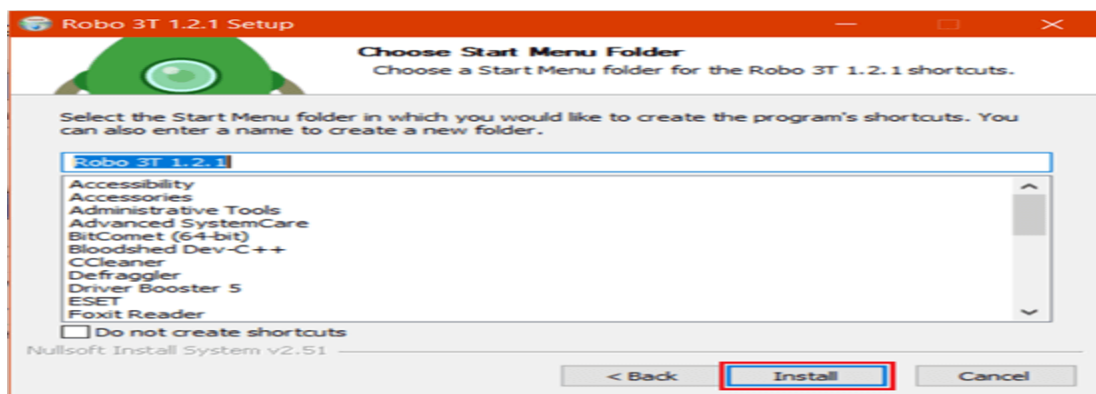
Step 3: Click “Next”.

Figure 23



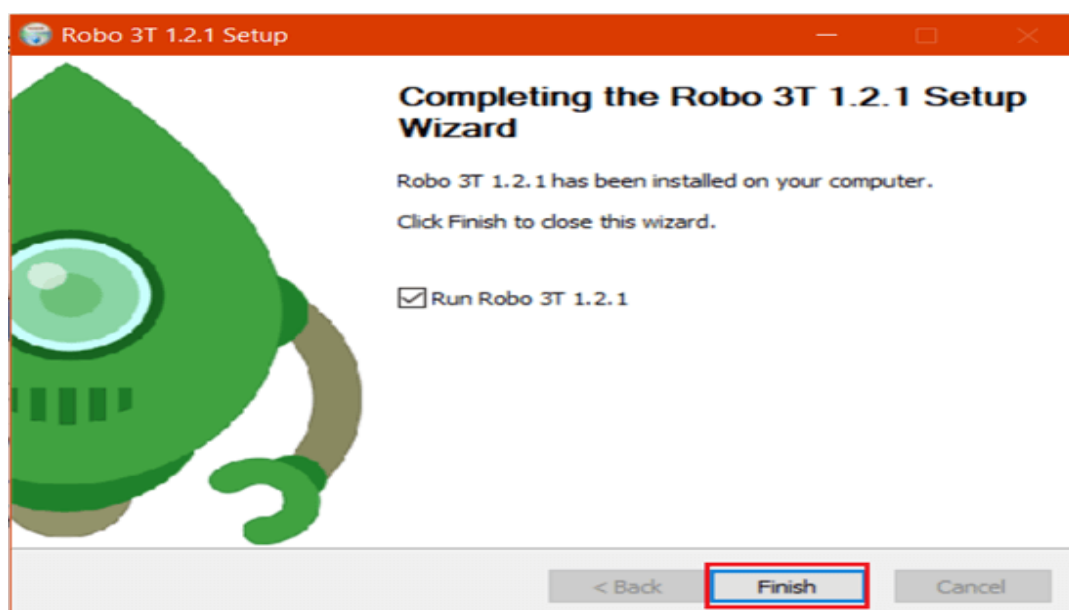
Step 4: Click “Install”.

Figure 24



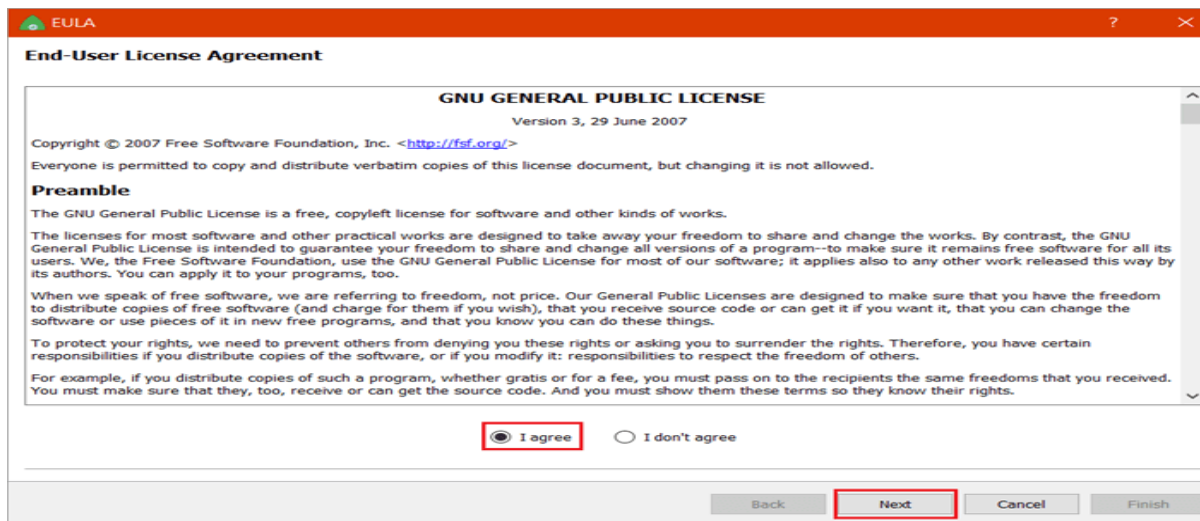
Step 5: Click “Finish”.

Figure 25



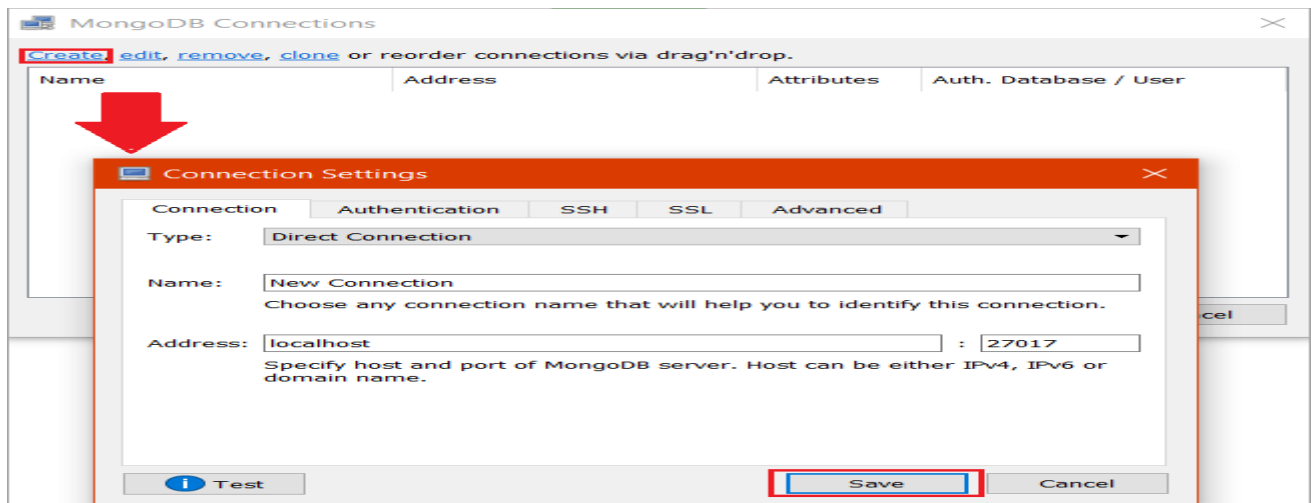
Step 6: Launch the Application & Click “I agree” to the agreement.

Figure 26



Step 7: Click on “Create” to create a new connection and save the address & port as it is given.

Figure 27



Step 8: Click On “Connect”. Once it is connected, the user could view all the databases, collections, and documents. Through UI, one can insert Documents, delete, update and find documents.

Figure 28

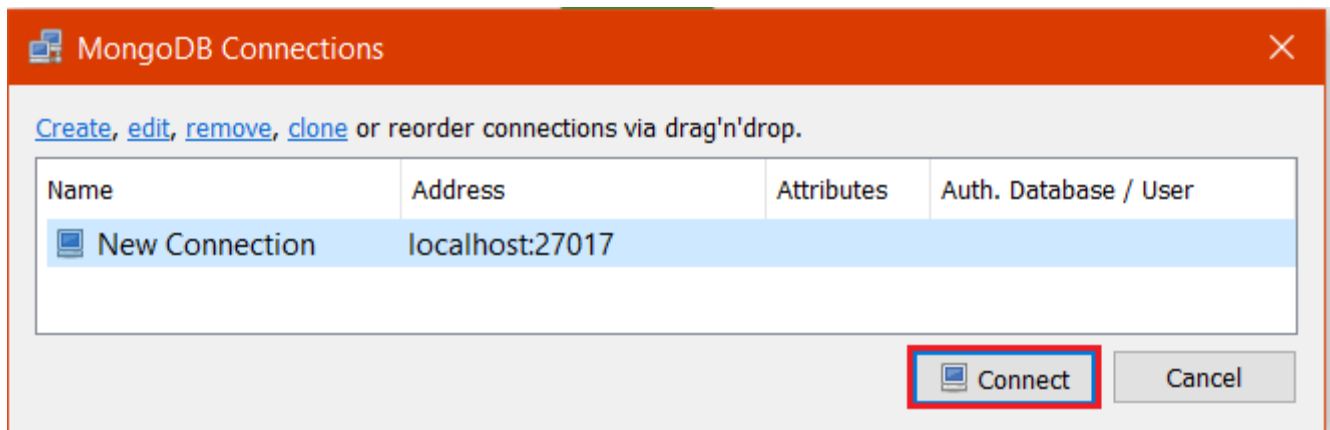
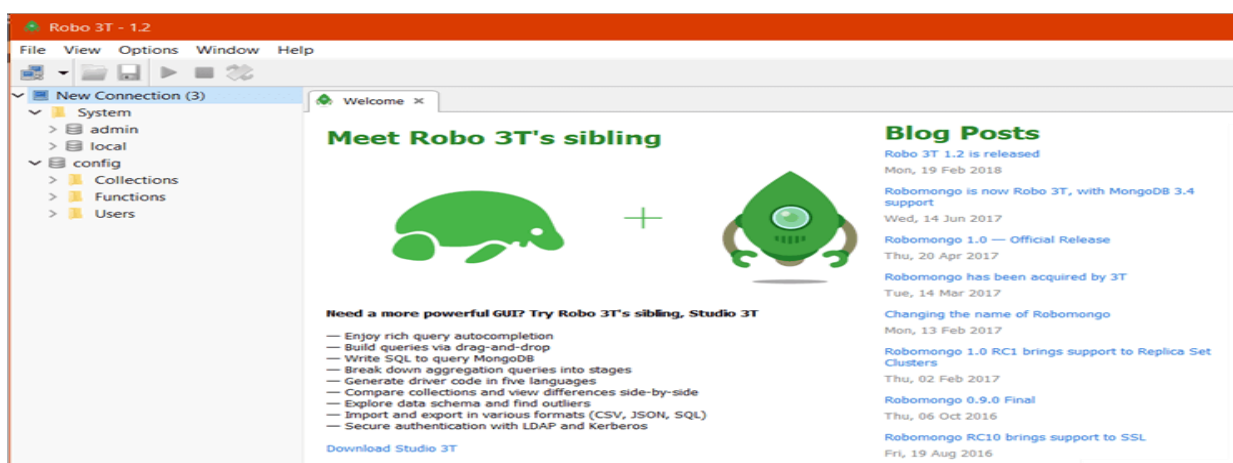
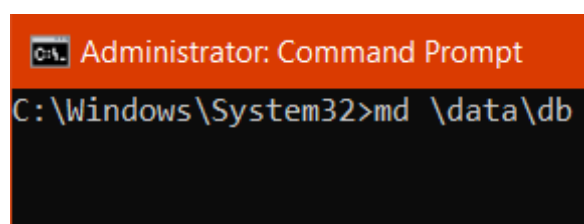


Figure 29



Step 9: Open **Command prompt** with admin privileges and type: **md \data\db** in order to make a directory which would be **\data\db** at the same folder.

Figure 30



MongoDB would create the database directly at the same folder or location as it would be the default location of MongoDB to store information. It is done to make sure that MongoDB would find the directory at the same location when it would start.

The data import through Robo 3t is done and could be seen by using the **mongoimport** command with some data to be stored.

Conclusion:

Thus we successfully installed MongoDB.

EXPERIMENT NO: 08

Title: To Accessing MongoDB& Perform Create, Retrieve, Update and Delete or CRUD operations in MongoDB.

Aim: Accessing MongoDB& Perform Create, Retrieve, Update and Delete or CRUD operations in MongoDB.

Theory:

MongoDB is an open-source, cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

Database in MongoDB

Database is a physical container for collections. Each database gets its own set of files on the file system.

A single MongoDB server typically has multiple databases.

Collection

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

Document

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.

Following table shows the relationship of traditional RDBMS and MongoDB.

| RDBMS | MongoDB |
|-----------------------------------|--|
| Database | Database |
| Table | Collection |
| Tuple/Row | Document |
| column | Field |
| Table Join | Embedded Documents |
| Primary Key | Primary Key (Default key _id provided by mongodb itself) |
| Database Server and Client | |
| Mysqld/Oracle | mongod |
| mysql/sqlplus | mongo |

Table 9.1: RDBMS and MongoDB relationship

Advantages of MongoDB over RDBMS

- **Schema less:** MongoDB is document database in which one collection holds different documents. Number of fields, content and size of the document can be differ from one document to another.
- Structure of a single object is clear.

- No complex joins
- **Deep query-ability:** MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL
- **Ease of scale-out:** MongoDB is easy to scale
- Conversion / mapping of application objects to database objects not needed
- Uses internal memory for storing the (windowed) working set, enabling faster access of data.

The use Command

MongoDB use DATABASE_NAME is used to create database. The command will create a new database, if it doesn't exist otherwise it will return the existing database.

Syntax:

Basic syntax of use DATABASE statement is as follows:

use DATABASE_NAME

Example:

If you want to create a database with name <mydb>, then use DATABASE statement would be as follows:

```
>use mydb
switched to dbmydb
```

To check your currently selected database use the command db

```
>dbmydb
```

If you want to check your databases list, then use the command show dbs.

```
>show dbs
```

```
local 0.78125GB test 0.23012GB
```

Your created database (mydb) is not present in list. To display database you need to insert at least one document into it.

```
>db.student.insert({"name":"sachin"})
```

```
>show dbs
```

```
local 0.78125GB mydb 0.23012GB test 0.23012GB
```

In MongoDB default database is test. If you didn't create any database then collections will be stored in test database.

The dropDatabase() Method

MongoDB **db.dropDatabase()** command is used to drop a existing database.

Syntax:

Basic syntax of **dropDatabase()** command is as follows: **db.dropDatabase()**

This will delete the selected database. If you have not selected any database, then it will delete default 'test' database

Collections in MongoDB

The createCollection() Method

MongoDBdb.createCollection(name, options) is used to create collection.

Syntax:

Basic syntax of createCollection() command is as follows db.createCollection(name, options)

In the command, name is name of collection to be created. Option is a document and used to specify configuration of collection(memory size etc. optional part).

Examples:

Basic syntax of **createCollection()** method without options is as follows

```
>use test
```

```
switched to db test
```

```
>db.createCollection("mycollection")
{ "ok" : 1 }
```

You can check the created collection by using the command **show collections**

```
>show collections mycollectionsystem.indexes
```

In MongoDB you don't need to create collection. MongoDB creates collection automatically, when you insert some document.

```
>db.student.insert({"name" : "sagar"})
```

Above command creates collection student automatically.

Multiple records in student collection can be inserted using following command:

```
>db.student.insert([
  { rno:101512,
    name: "James", city:"Pune"
  },
  { rno:101513,
    name: "Anderson", city: "London"
  }
])
```

Above command inserts two records in student collection. Here rno, name and city are treated as columns (known as key in MongoDB) and 101512, James and Pune are the actual values.

The drop() Method

MongoDB'sdb.collection.drop() is used to drop a collection from the database.

Syntax:

Basic syntax of drop() command is as follows

```
db.COLLECTION_NAME.drop()
```

Example:

```
db.student.drop ()
```

This command will drop collection student.

Retrieving Data in MongoDB

find() method is used to retrieve records in a collection.

Syntax:

db.COLLECTION_NAME.find()

Example:

db.student.find()

Above command will display all records (key-value pairs) present in collection “student”. Its working is similar as that of *select * from student* command in SQL.

Output:

🟢 C:\Program Files\MongoDB\Server\5.0\bin\mongo.exe

```
> show dbs
admin    0.000GB
config  0.000GB
local    0.000GB
> use Akshay
switched to db Akshay
> db.Akshay.insertOne({name:'akshay',age:24})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("61cde94171435bef0702856d")
}
> db.Akshay.find({name:'akshay'})
{ "_id" : ObjectId("61cde94171435bef0702856d"), "name" : "akshay", "age" : 24 }
> db.Akshay.updateOne({name: "akshay"}, {$set:{age:25}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.Akshay.find({name:'akshay'})
{ "_id" : ObjectId("61cde94171435bef0702856d"), "name" : "akshay", "age" : 25 }
> db.Akshay.deleteOne({name:'akshay'})
{ "acknowledged" : true, "deletedCount" : 1 }
> db.Akshay.find({name:'akshay'})
>
_
```

Conclusion:

Thus, we studied CRUD operations in MongoDB.

EXPERIMENT NO: 09

Title : To install CouchDB on windows.

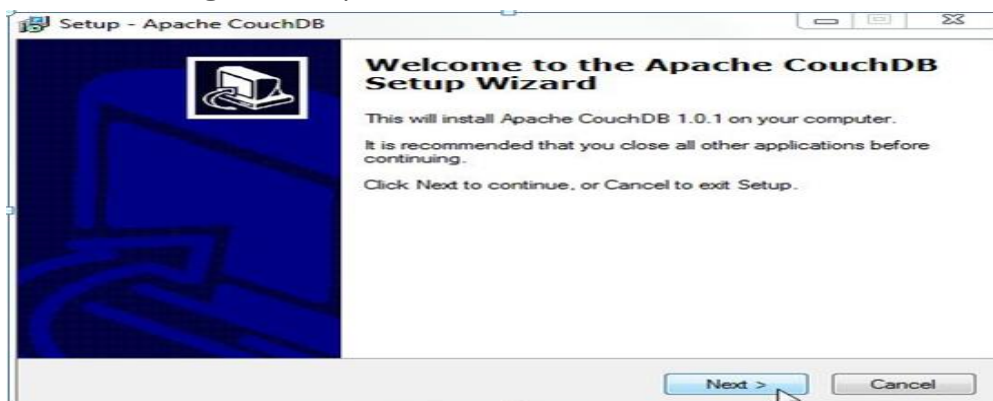
Aim: To install CouchDB on windows.

Theory:

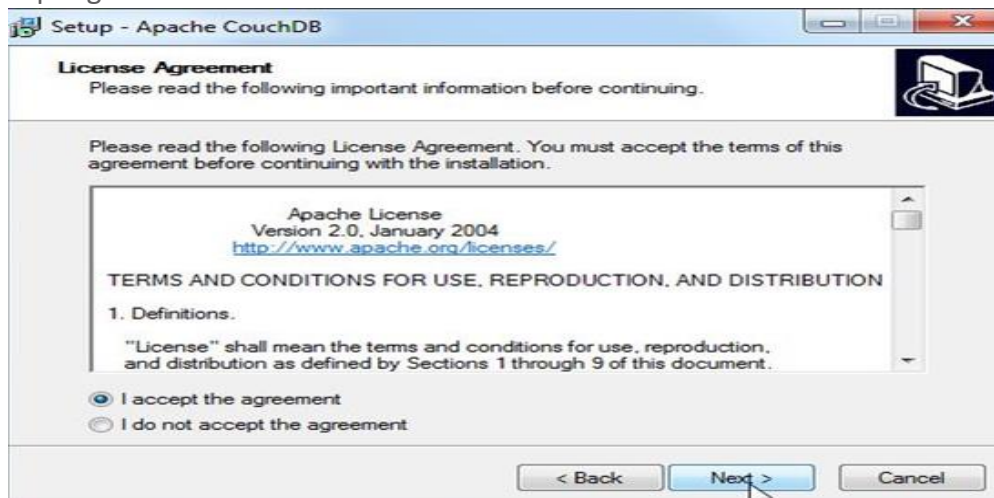
Couchdb is a document oriented database system, which stores data in *JSON* format. CouchDB is developed by the *Apache* organization, so if you want to install *Couchdb* in your Windows system, then follow the below steps.

First you need to download couch db from the given [link](#) for Windows operating systems.

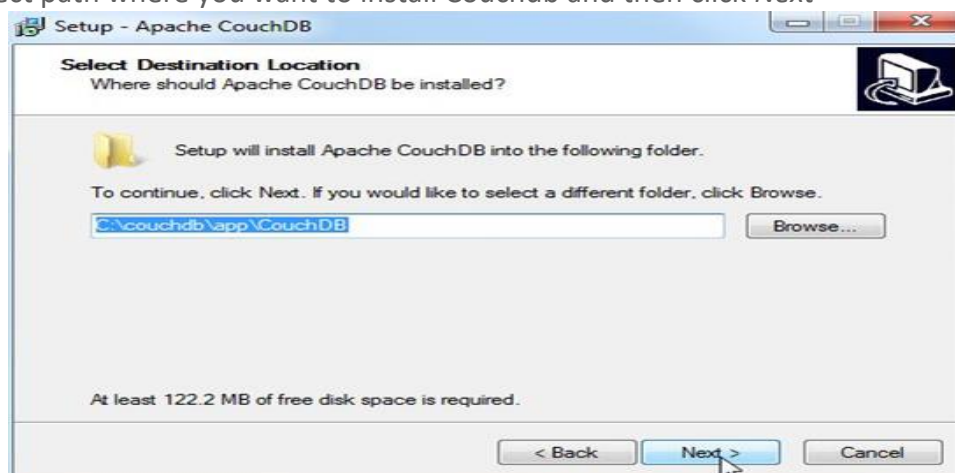
Step 1- After downloading, run setup file. Click on *Next* tab



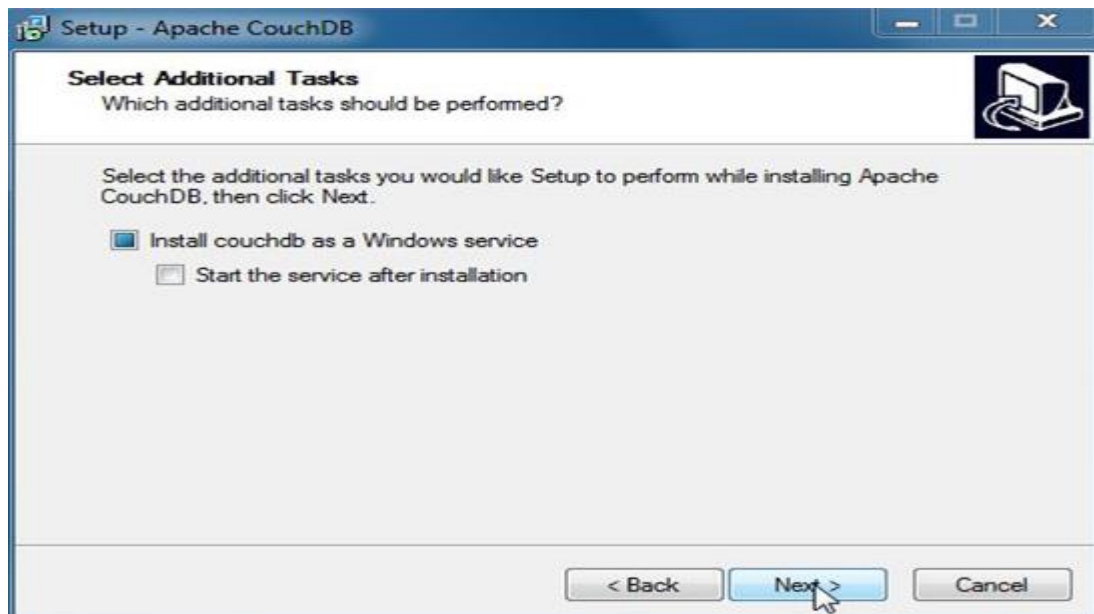
Step 2 - Accept agreement and then click *Next*



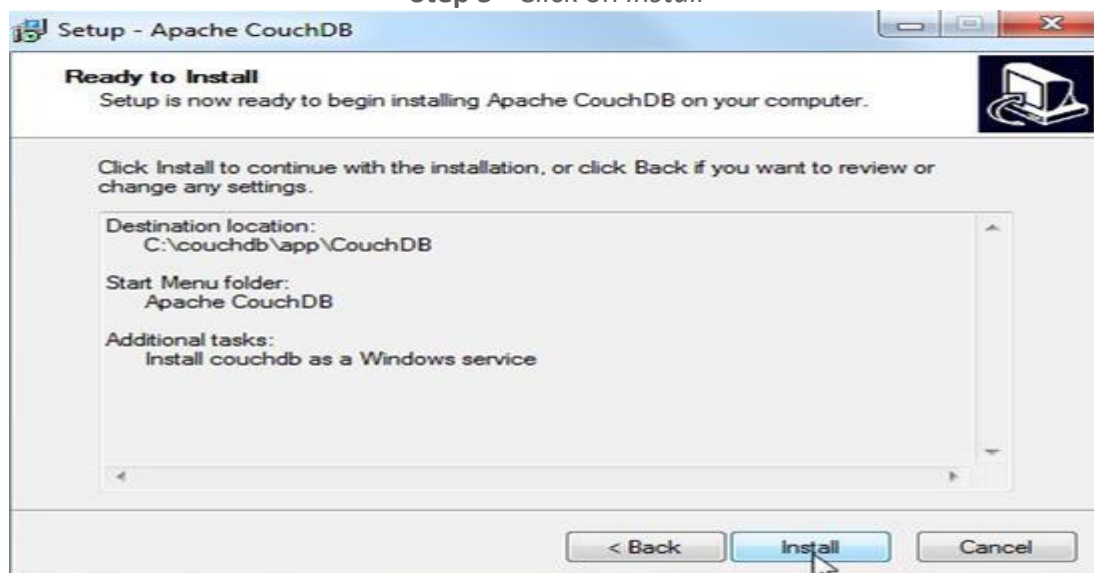
Step 3- Select path where you want to install Couchdb and then click *Next*



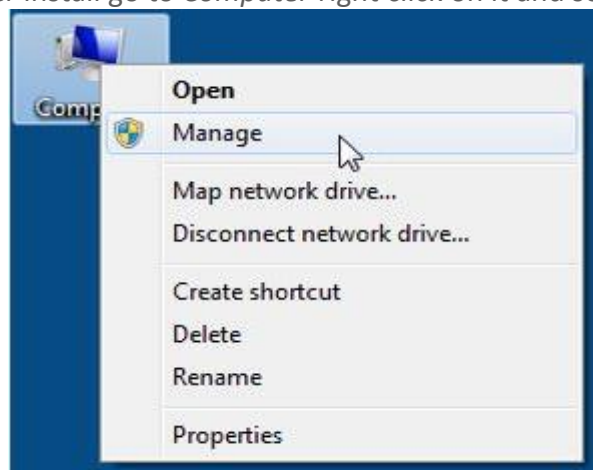
Step 4 - Select additional tasks and then click *Next*



Step 5 - Click on *Install*



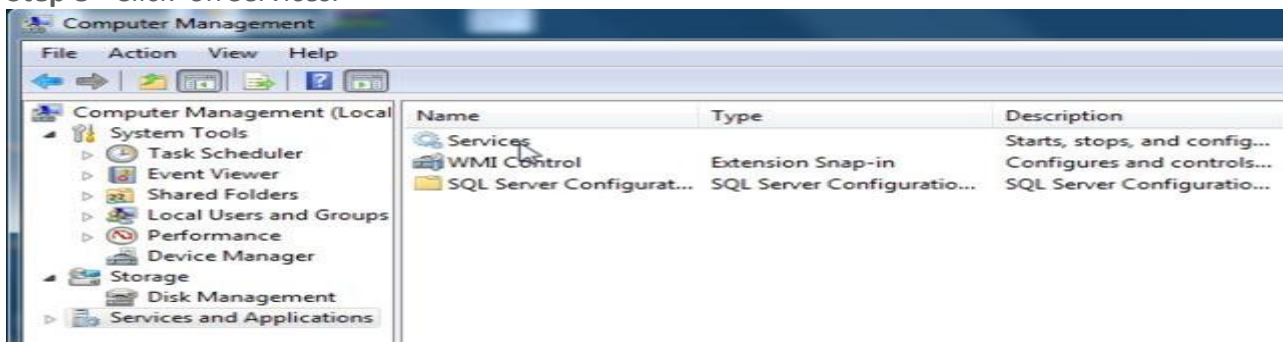
Step 6 - After install go to *Computer* right click on it and select *Manage*.



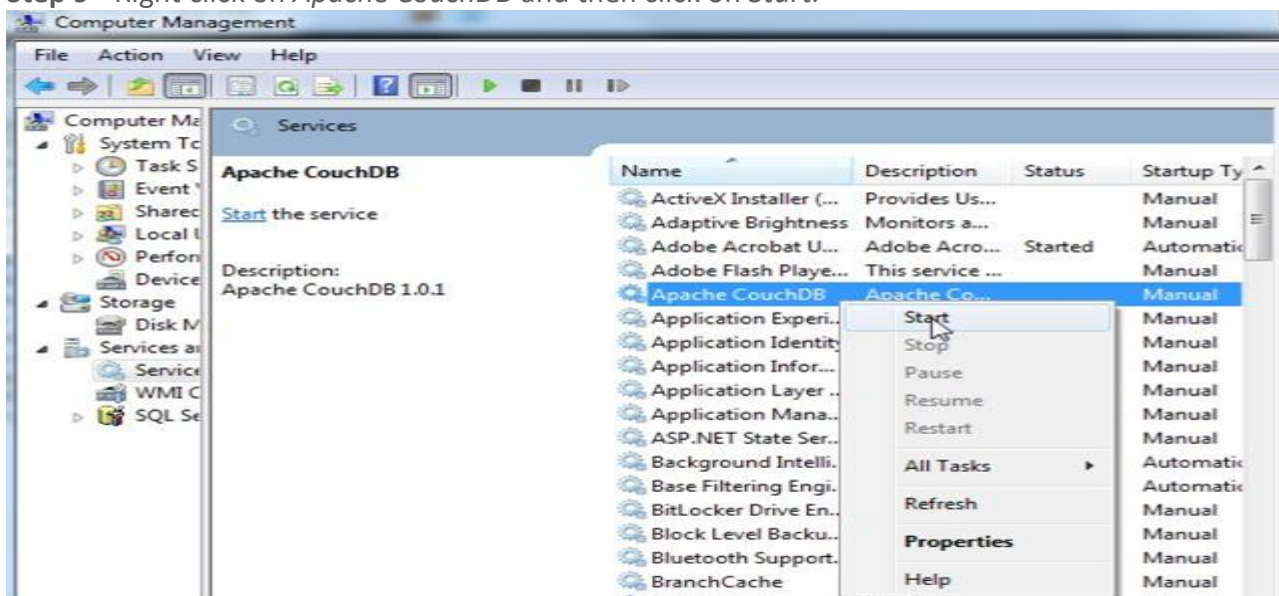
Step 7 - Click *Services and Applications*.



Step 8 - Click on *Services*.

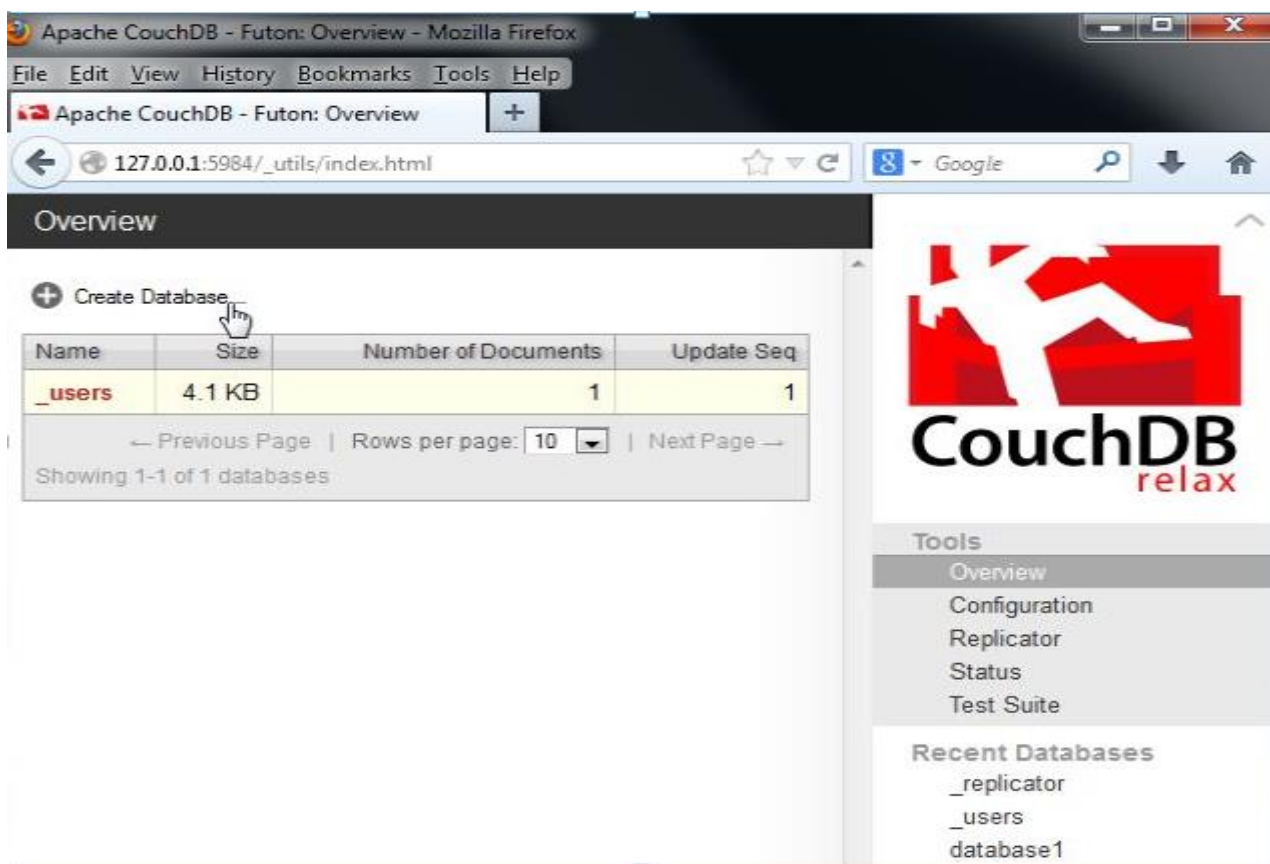
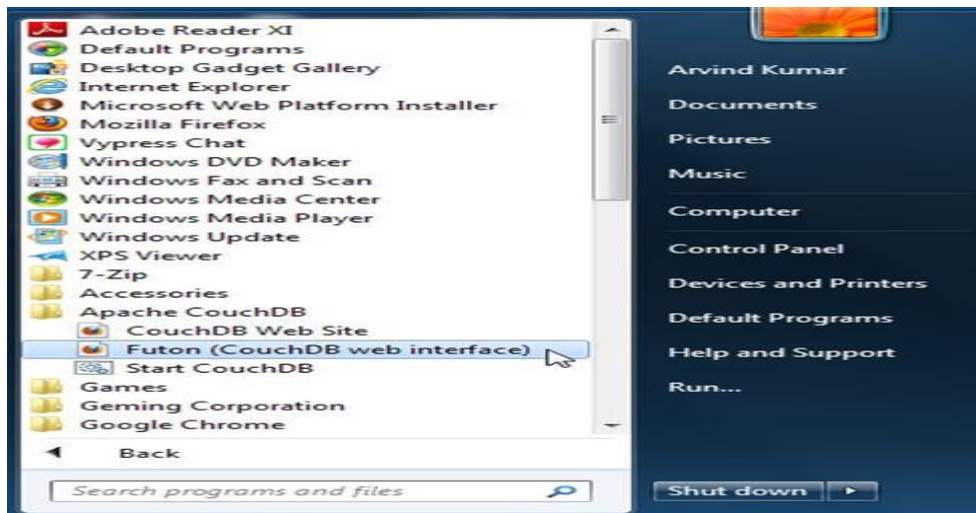


Step 9 - Right click on *Apache CouchDB* and then click on *Start*.



Step 10 - After selecting *start* go to start menu and select *all program*, click on *Apache CouchDB* and then click on *Futon* (Couch DB web interface). See given below figure.

Step 11 - Finally *CouchDB* start.



Conclusion: Thus, we have successfully installed Couch DB on windows.

EXPERIMENT NO: 10

Title: To create and delete CouchDB database. Run CouchDB query with Mongo.

Aim: To create and delete CouchDB database. Run CouchDB query with Mongo.

Create Database

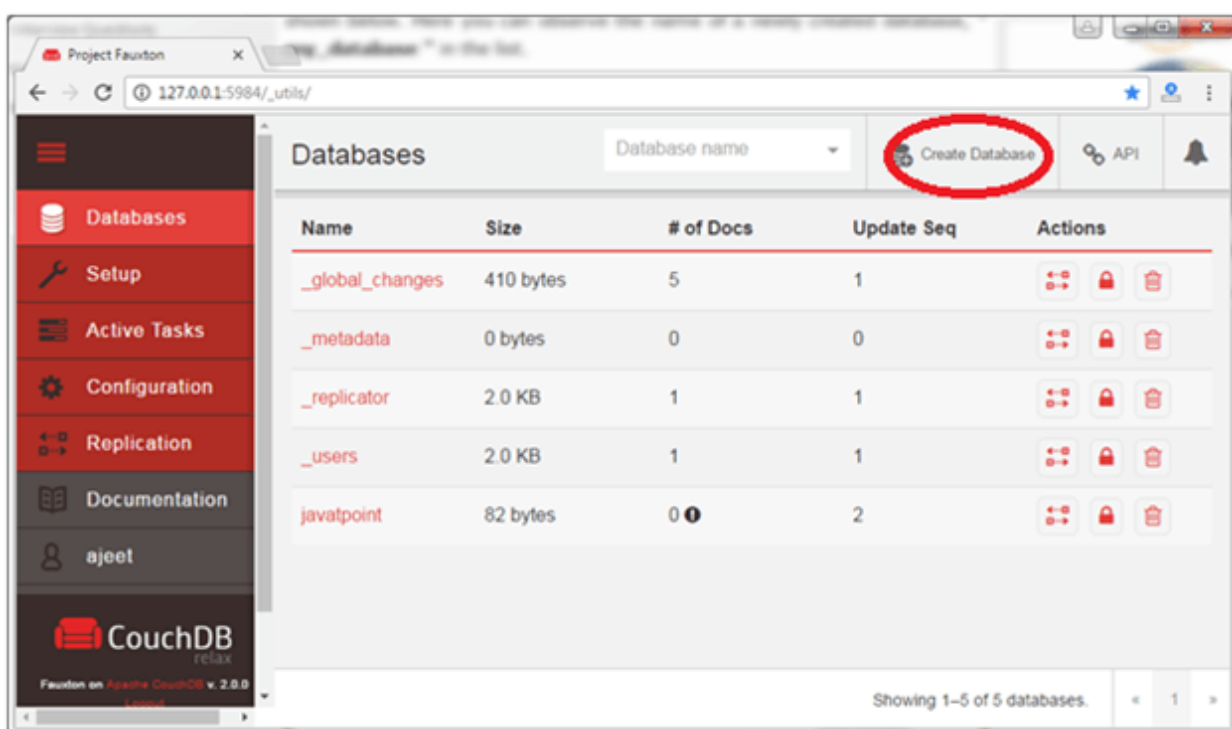
In CouchDB, database is the outermost structure where documents are stored. CouchDB provides cURL utility to create databases. You can also use Futon the web interface of CouchDB.

Creating a database using Fauxton

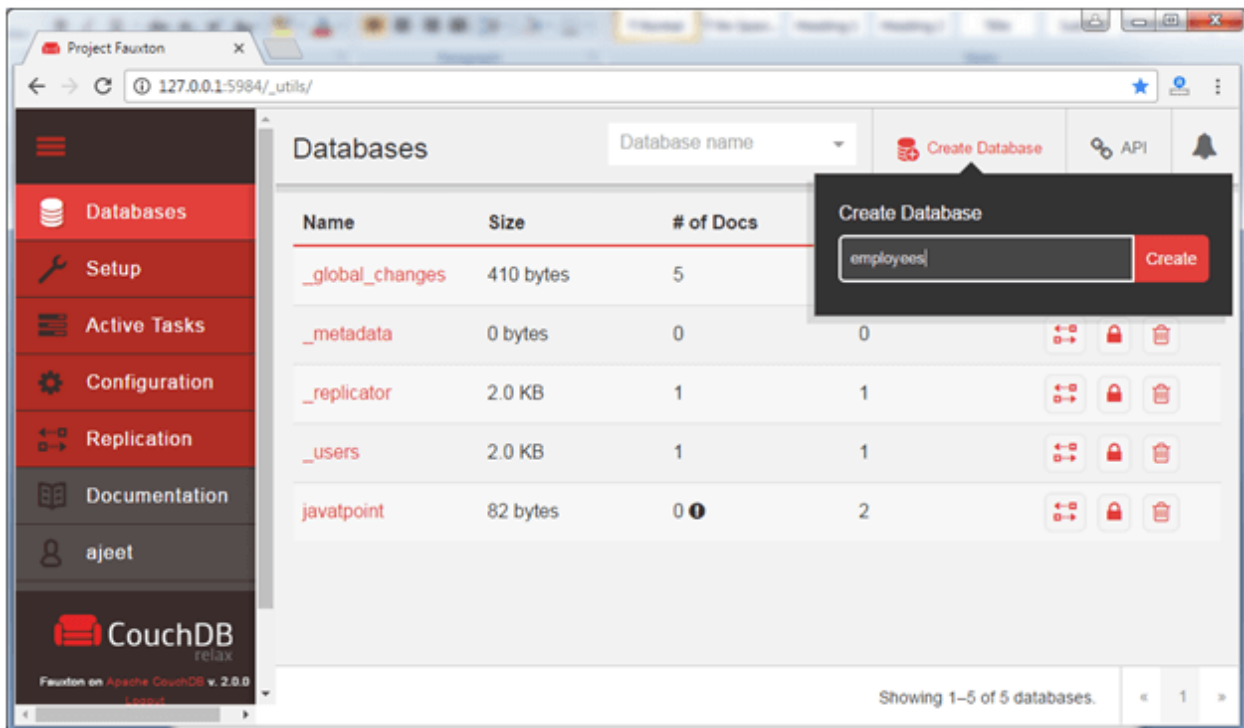
Open the following link in the web browser:

http://127.0.0.1:5984/_utils/.

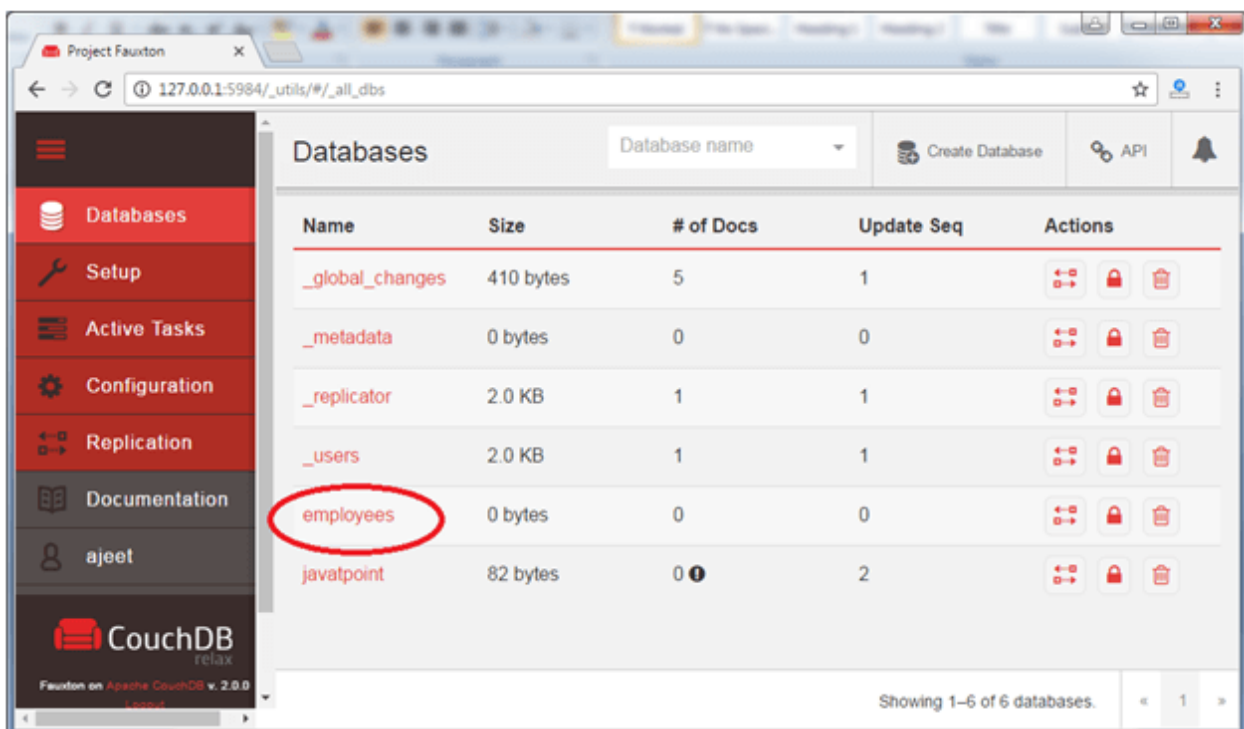
You will get a page like this:



Click on the "Create Database" tab in the red circle to create a database named "employees".



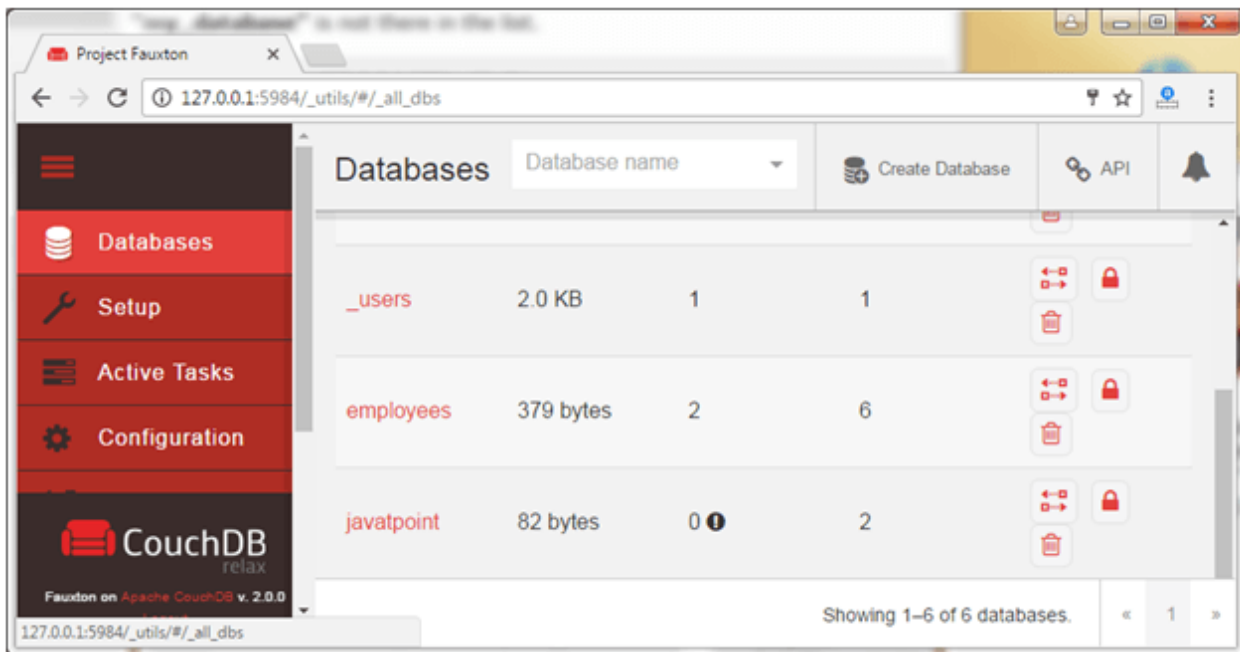
It will show a message that database is created successfully. You can check the created database in the database tab.



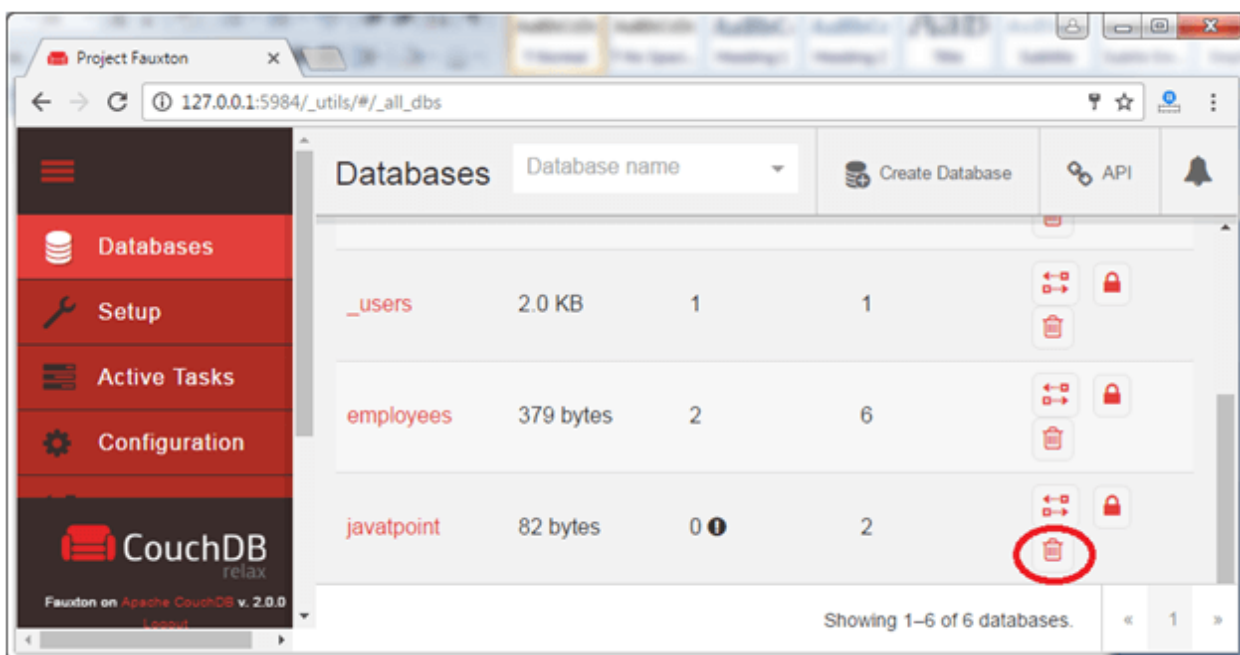
CouchDB Delete Database using Fauxton

Open the Fauxton url: http://127.0.0.1:5984/_utils/

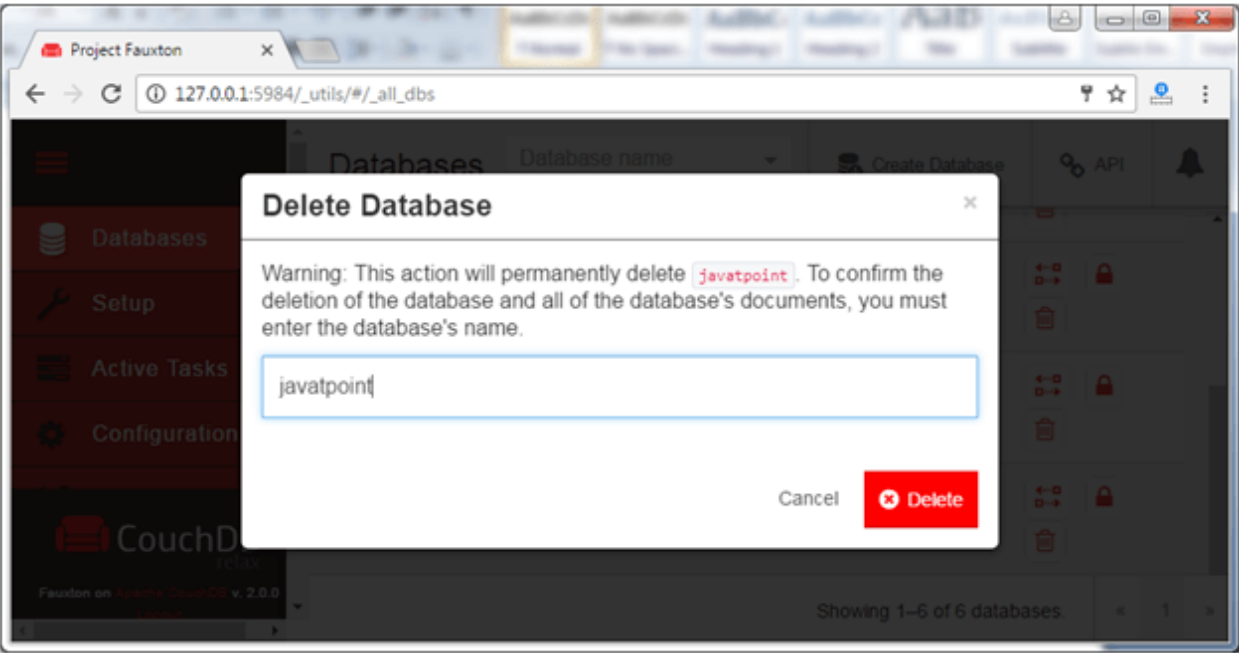
Click on the "Databases" tab and you will see all databases:



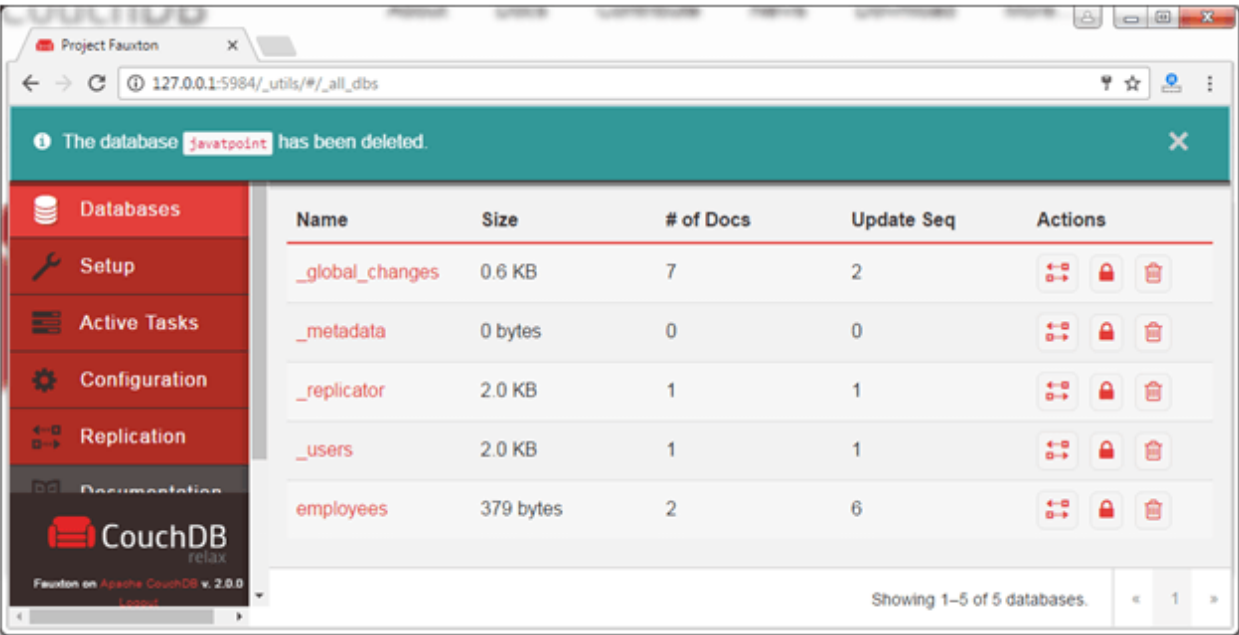
Here, we are going to delete "jvatpoint" database. Click on the delete icon encircled in red.



It will show a pop-up message asking to confirm the database name. Write down the name of the database.



Now the database is deleted.



Conclusion: Thus successfully created and deleted couchDB.

EXPERIMENT NO: 11

Title: To Demonstrate OLAP operations and cube operator in OLAP.

Aim: To Demonstrate OLAP operations and cube operator in OLAP..

Theory:-

- **Multidimensional Data Model**

Multidimensional data model stores data in the form of data cube. Mostly, data warehousing supports two or three-dimensional cubes. A data cube allows data to be viewed in multiple dimensions. Dimensions are entities with respect to which an organization wants to keep records. For example in store sales record, dimensions allow the store to keep track of things like monthly sales of items and the branches and locations. A multidimensional database helps to provide data-related answers to complex business queries quickly and accurately. Data warehouses and Online Analytical Processing (OLAP) tools are based on a multidimensional data model. OLAP in data warehousing enables users to view data from different angles and dimensions.

- **Online Analytical Processing:**

Online Analytical Processing Server (OLAP) is based on the multidimensional data model. It allows managers, and analysts to get an insight of the information through fast, consistent, and interactive access to information.

OLAP operations in multidimensional data.

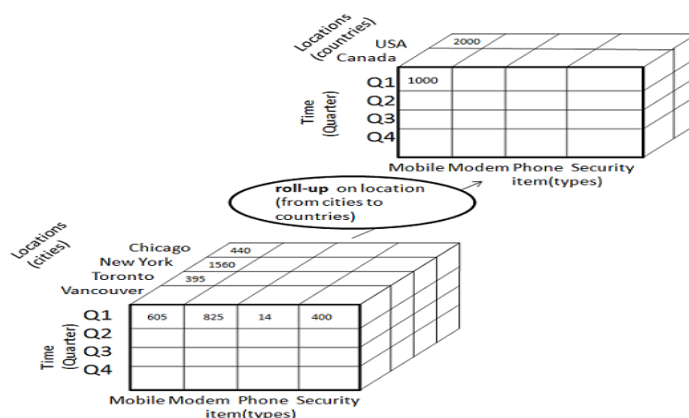
- Roll-up
- Drill-down
- Slice and dice
- Pivot (rotate)

Roll-up

Roll-up performs aggregation on a data cube in any of the following ways –

- By climbing up a concept hierarchy for a dimension
- By dimension reduction

The following diagram illustrates how roll-up works.



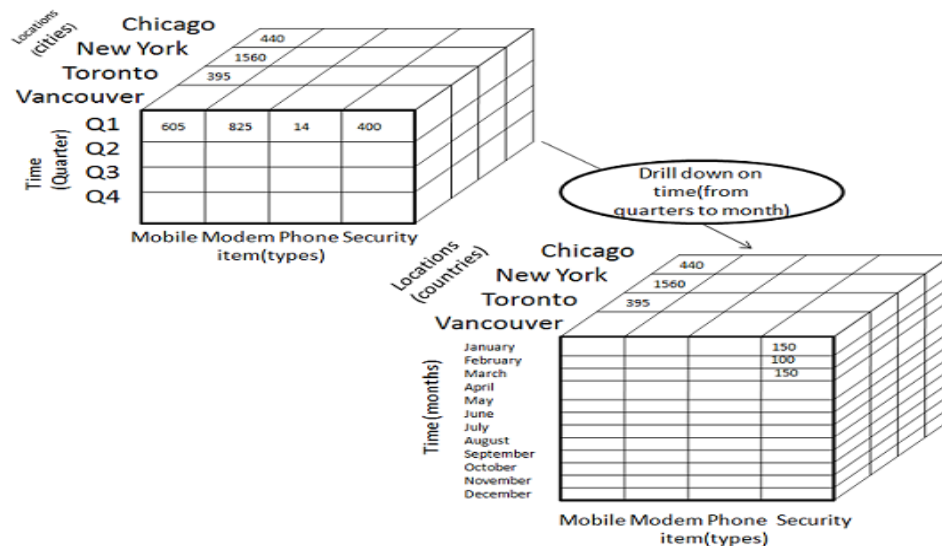
- Roll-up is performed by climbing up a concept hierarchy for the dimension location.
- Initially the concept hierarchy was "street < city < province < country".
- On rolling up, the data is aggregated by ascending the location hierarchy from the level of city to the level of country.
- The data is grouped into cities rather than countries.
- When roll-up is performed, one or more dimensions from the data cube are removed.

Drill-down

Drill-down is the reverse operation of roll-up. It is performed by either of the following ways –

- By stepping down a concept hierarchy for a dimension
- By introducing a new dimension.

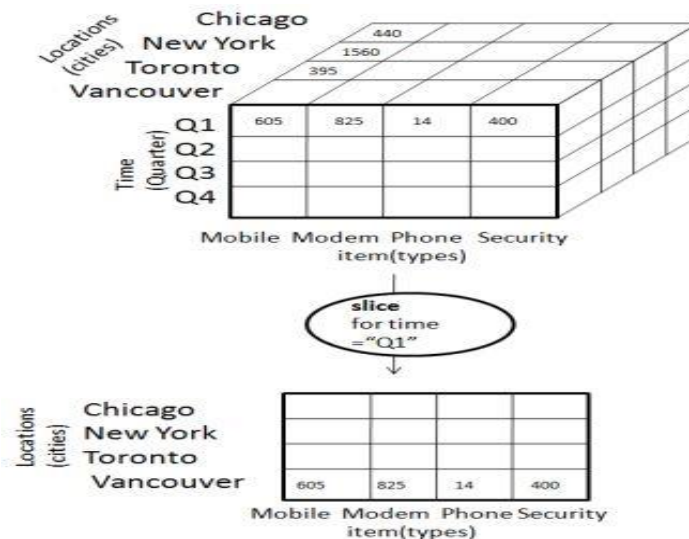
The following diagram illustrates how drill-down works –



- Drill-down is performed by stepping down a concept hierarchy for the dimension time.
- Initially the concept hierarchy was "day < month < quarter < year."
- On drilling down, the time dimension is descended from the level of quarter to the level of month.
- When drill-down is performed, one or more dimensions from the data cube are added.
- It navigates the data from less detailed data to highly detailed data.

Slice

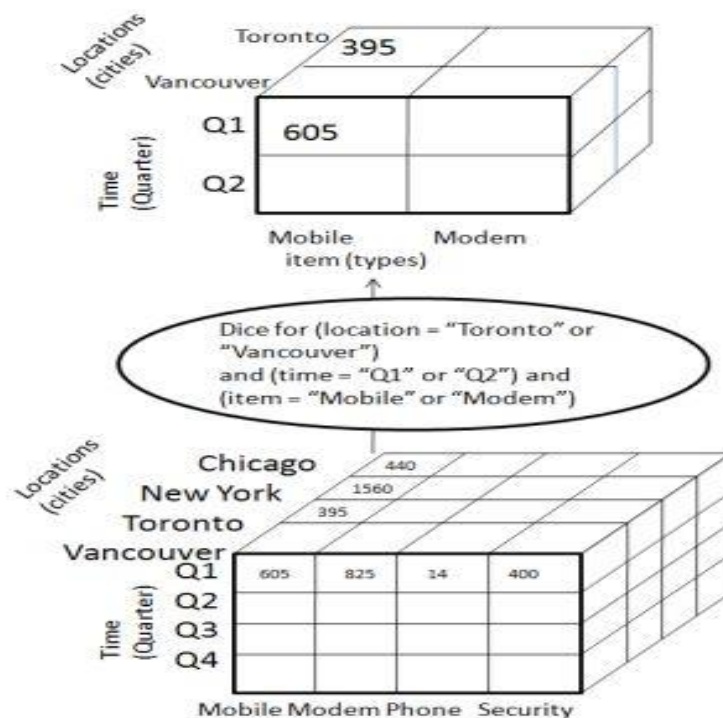
The slice operation selects one particular dimension from a given cube and provides a new sub-cube. Consider the following diagram that shows how slice works.



- Here Slice is performed for the dimension "time" using the criterion time = "Q1".
- It will form a new sub-cube by selecting one or more dimensions.

Dice

Dice selects two or more dimensions from a given cube and provides a new sub-cube. Consider the following diagram that shows the dice operation.



The dice operation on the cube based on the following selection criteria involves three dimensions.

- (location = "Toronto" or "Vancouver")
- (time = "Q1" or "Q2")
- (item = "Mobile" or "Modem")

IMPLEMENTATION:**//Creating Clothes table**

```
create table clothes (item varchar2(40),color varchar2(20),size1 varchar2(10),numint);
```

//Inserting Data

```
insert into clothes values('shirt','white','small',20);
```

```
insert into clothes values('shirt','blue','medium',10);
```

```
insert into clothes values('pant','black','large',25);
```

```
insert into clothes values('dress','white','medium',30);
```

```
insert into clothes values('pant','blue','large',20);
```

```
insert into clothes values('dress','pink','small',15);
```

//Rollup Query

```
select item,color,size1,sum(num) from clothes group by rollup(item,color,size1);
```

//Cube Query

```
select item,color,size1,sum(num) from clothes group by cube(item,color,size1);
```

Conclusion:

CUBE generates a result set that shows aggregates for all combinations of values in the selected columns. ROLLUP generates a result set that shows aggregates for a hierarchy of values in the selected columns.

Program:

```
create table clothes (item varchar2 (40),color varchar2 (20),size1 varchar2(10),numint);
```

```
insert into clothes values('shirt','white','small',20);
```

```
insert into clothes values('shirt','blue','medium',10);
```

```
insert into clothes values('pant','black','large',25);
```

```
insert into clothes values('dress','white','medium',30);
```

```
insert into clothes values('pant','white','large',20);
```

```
insert into clothes values('dress','pink','small',15);
```

```
select *from clothes;
```

```
select item,color,size1,sum(num)from clothes group by rollup(item,color,size1);
```

```
select item,color,size1,sum(num)from clothes group by cube(item,color,size1);
```

EXPERIMENT NO: 12

| |
|---|
| Title: Case study on CASE tools. |
| Aim: Case study on CASE tools. |

Theory:-

CASE stands for Computer Aided Software Engineering which is software that supports one or more software engineering activities within a software development process, and is gradually becoming popular for the development of software as they are improving in the capabilities and **functionality** and are proving to be beneficial for the development of quality software.

Casetools:

Whenever a new system is installed, the implementation integrates a number of related and different tasks. The process has to be efficiently organized and it is for this very reason that CASE tools are developed. With the help of CASE, the installation process can be automated and coordinated within the developed and adopted system life cycle.

CASE tools are the software engineering tools that permit collaborative software development and maintenance. Almost all the phases of the software development life cycle are supported by them such as analysis; design, etc., including umbrella activities such as project management, configuration management etc. In general, standard software development methods such as Jackson Structure programming or structured system analysis and design method are also supported by CASE tools. CASE tools may support the following development steps for developing data base application:

- Creation of data flow and entity models
- Establishing a relationship between requirements and models
- Development of top-level design
- Development of functional and process description
- Development of test cases.

Why CASE tools are developed:

CASE tools are designed to enhance and upgrade the computing system adopted and used. This is very important with regards to the dependence on a computer-based environment for business and/or personal pursuits. It is an important part of various business growth strategies. The CASE tools are developed for the following reasons:

1. Firstly Quick Installation.
2. Time Saving by reducing coding and testing time.
3. Enrich graphical techniques and data flow.
4. Optimum use of available information.
5. Enhanced analysis and design development.
6. Create and manipulate documentation.
7. Transfer the information between tools efficiently.
8. The speed during the system development increased.

How the Organization uses CASE tools:

Here are the ways where the CASE tools are used:

1. **To facilitate single design methodology:**

CASE tools help the organization to standardize the development process. It also facilitates coordinated development. Integration becomes easy as common methodology is adopted.

2. **Rapid Application Development:**

To improve the speed and quality of system development organizations use CASE tools.

3. **Testing:**

CASE tools help in improving the testing process through automated checking and simplified program maintenance.

4. **Documentation:**

In a traditional software development process, the quality of documentation at various stages depends on the individual. At various stages of SDLC CASE tools improve the quality and uniformity of documentation. It also ensures the completeness of the documentation.

5. **Project Management:**

It improves project management activity and to some extent automates various activities involved in project management.

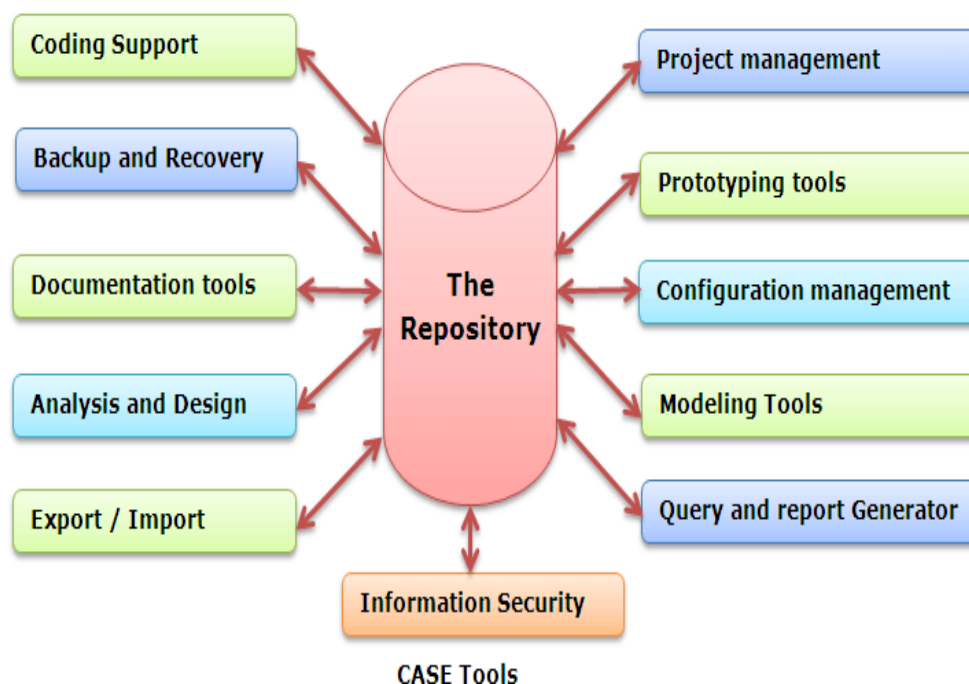
6. **Reduce the maintenance cost:**

Use of CASE tools makes the software easy to maintain and hence reduce the maintenance costs.

7. **Increase Productivity:**

Automation of various activities of system development and management processes increases productivity of the development team.

Environment having CASE:



Role of CASE tools:

CASE tools play a major role in the following activities:

- Project management
- Data dictionary
- Code generation
- User interface design
- Schema generation
- Creation of meta-data for data warehouse
- Reverse engineering
- Re-engineering
- Document generation
- Version control
- OO analysis and design
- Software testing
- Data modeling
- Project scheduling
- Cost estimation

Need of CASE tools:

The software development process is expensive and as the projects become more complex in nature, the project implementations become more demanding and expensive. That's why software developers always looking for such CASE tools that help them in many different ways during the different development stages of software, so that they can understand the software and prepare a good end product that efficiently fulfill the user requirements. CASE tools provide the integrated homogenous environment for the development of complex projects. These tools provide computerized setting to software developers to analyze a problem and then design its system model. The CASE tools also provide the environment for monitoring and controlling projects such that team leaders are able to manage the complex projects.

Basically, the CASE tools are used to

- Reduce the cost as they automate many repetitive manual tasks.
- Reduce development time of the project as they support standardization and avoid repetition and reuse.
- Develop better quality complex projects as they provide greater consistency and coordination.
- Create good quality documentation.
- Create systems that are maintainable because of proper control of configuration item that support traceability requirements.

Categories of CASE Tools:

Sometimes CASE tools are classified in to following categories due to their activities:

1. UPPER CASE Tools
2. LOWER CASE Tools
3. INTEGRATED CASE Tools

Upper CASE tools:

They support the analysis and the design phase. They include tools for analysis modeling, reports and forms generation.

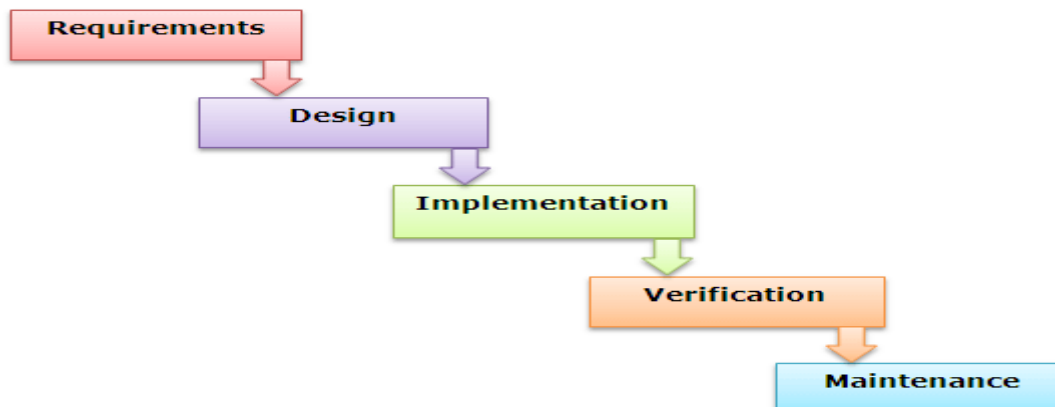
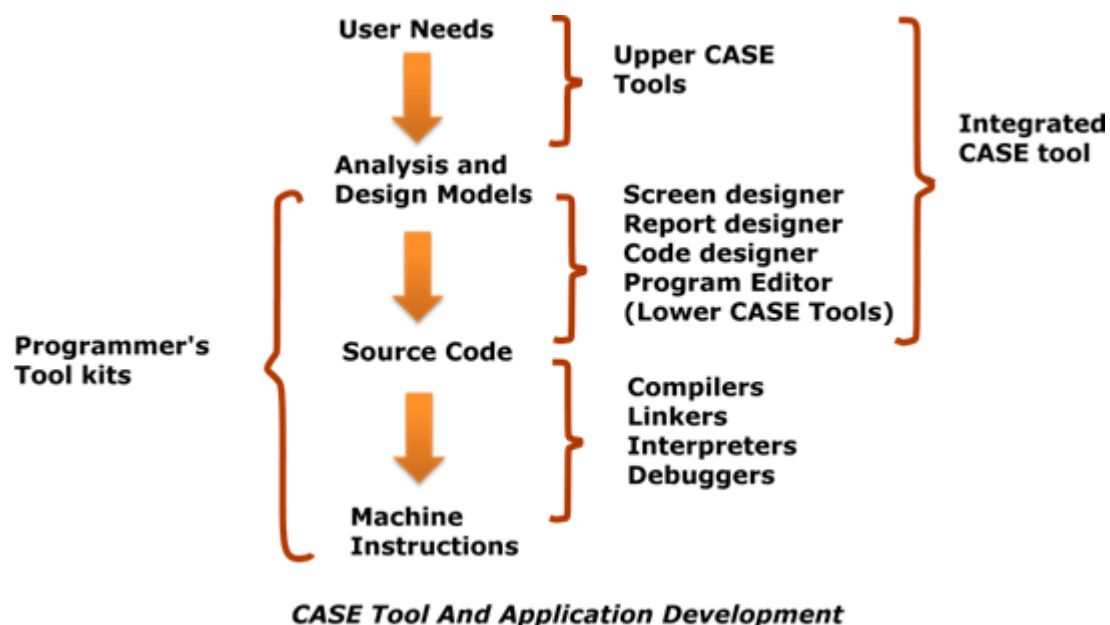
Lower CASE tools:

They support the coding phase, configuration management, etc.

Integrated CASE tools:

It is known as I-CASE and also supports analysis, design and coding phases.

The phases that are supported by the Lower and upper CASE tools are shown in the Waterfall model as under:

**Positioning of CASE tools in a Software Application development:**

Advantages and Disadvantages of CASE Tools:

| Advantages | Disadvantages |
|---|---|
| Produce system with a longer effective operational life. | Produce initial system that is more expensive to build and maintain. |
| Produces system that more closely meet user needs and requirements. | Require more extensive and accurate definitions of user needs and requirements. |
| Produces system with excellent documentation. | May be difficult to customize. |
| Produces system that needs less systems support. | Require training of maintenance staff. |
| Produce more flexible system. | May be difficult to use with existing system. |

Characteristics of a successful CASE Tool:

A CASE tool must have the following characteristics in order to be used efficiently:

- **A standard methodology:** A CASE tool must support a standard software development methodology and standard modeling techniques. In the present scenario most of the CASE tools are moving towards UML.
- **Flexibility:** Flexibility in use of editors and other tools. The CASE tool must offer flexibility and the choice for the user of editors' development environments.
- **Strong Integration:** The CASE tools should be integrated to support all the stages. This implies that if a change is made at any stage, for example, in the model, it should get reflected in the code documentation and all related design and other documents, thus providing a cohesive environment for software development.
- **Integration with testing software:** The CASE tools must provide interfaces for automatic testing tools that take care of regression and other kinds of testing software under the changing requirements.
- **Support for reverse engineering:** A CASE tools must be able to generate complex models from already generated code.
- **On-line help:** The CASE tools provide an online tutorial.

Conclusion: We studied CASE Tools.