Experiment No.2

Title: Implementation of Distributed Database.

Batch:SY-IT(B3) **Roll No.:16010423076** **Experiment No.: 2**

Aim: To Implement Distributed Database.

Resources needed: PostgreSQL 9.3

Theory

A distributed database system allows applications to access and manipulate data from local and remote databases. It partitions the data and stores at different physical locations. Partitioning refers to splitting what is logically one large table into smaller physical pieces. Partitioning can provide several benefits:

Query performance can be improved dramatically for certain kinds of queries.

Update performance can be improved too, since each piece of the table has indexes smaller than an index on the entire data set would be. When an index no longer fits easily in memory, both read and write operations on the index take progressively more disk accesses.

Bulk deletes may be accomplished by simply removing one of the partitions, if that requirement is planned into the partitioning design. DROP TABLE is far faster than a bulk DELETE, to say nothing of the ensuing VACUUM overhead.
Seldom-used data can be migrated to cheaper and slower storage media.

Partitioning enhances the performance, manageability, and availability of a wide variety of applications and helps reduce the total cost of ownership for storing large amounts of data. Partitioning allows tables, indexes, and index-organized tables to be subdivided into smaller pieces, enabling these database objects to be managed and accessed at a finer level of granularity.

The benefits will normally be worthwhile only when a table would otherwise be very large. The exact point at which a table will benefit from partitioning depends on the application, although a rule of thumb is that the size of the table should exceed the physical memory of

The following forms of partitioning can be implemented in PostgreSQL:

Range Partitioning

The table is partitioned into "ranges" defined by a key column or set of columns, with no overlap between the ranges of values assigned to different partitions. For example one might partition by date ranges, or by ranges of identifiers for particular business objects.

List Partitioning

The table is partitioned by explicitly listing which key values appear in each partition.

After creating the partition, **database link (DBLINK)** is used to create a connection of the host database server with the client database.

Database Links:

The central concept in distributed database systems is a **database link**. A database link is a connection between two physical database servers that allows a client to access them as one logical database. Database link is a pointer that defines a one-way communication path from one Database server to another database server. The link pointer is actually defined as an entry in a data dictionary table. To access the link, you must be connected to the local database that contains the data dictionary entry.

A database link connection is one-way in the sense that a client connected to local database A can use a link stored in database A to access information in remote database B, but users connected to database B cannot use the same link to access data in database A. If local users on database B want to access data on database A, then they must define a link that is stored in the data dictionary of database B.

A database link connection allows local users to access data on a remote database. For this connection to occur, each database in the distributed system must have a unique global database name in the network domain. The global database name uniquely identifies a database server in a distributed system.

dblink executes a query (usually a SELECT, but it can be any SQL statement that returns rows) in a remote database.

When two text arguments are given, the first one is first looked up as a persistent connection's name; if found, the command is executed on that connection. If not found, the first argument is treated as a connection info string as for dblink_connect, and the indicated connection is made just for the duration of this command.

Arguments

conname

Name of the connection to use; omit this parameter to use the unnamed connection.

connstr

A connection info string, as previously described for dblink_connect.

sql

The SQL query that you wish to execute in the remote database, for example select * from foo.

fail_on_error

If true (the default when omitted) then an error thrown on the remote side of the connection causes an error to also be thrown locally. If false, the remote error is locally reported as a NOTICE, and the function returns no rows.

Return Value

The function returns the row(s) produced by the query. Since dblink can be used with any query, it is declared to return record, rather than specifying any particular set of columns. This means that you must specify the expected set of columns in the calling query — otherwise PostgreSQL would not know what to expect. Here is an example:

```
SELECT *
  FROM dblink('dbname=mydb', 'select proname, prosrc from pg_proc') AS
  t1(proname name, prosrc text) WHERE proname LIKE 'bytea%';
```

Procedure:

Implementing distributed database:

1. **Create the parent table.**

Create the parent table.

```
CREATE TABLE sales(org int, name varchar(10));
```

2. **Create the child (partitioned) tables**

```
CREATE TABLE sales_part1
(CHECK (org < 6))
INHERITS (sales);

CREATE TABLE sales_part2
(CHECK (org >=6 and org <=10))
INHERITS (sales);
```

3. **Create the rules**

```
CREATE OR REPLACE RULE insert_sales_p1
AS ON INSERT TO sales
WHERE (org <6)
DO INSTEAD
INSERT INTO sales_part1 VALUES(NEW.org, NEW.name);

CREATE OR REPLACE RULE insert_sales_p2
AS ON INSERT TO sales
WHERE (org >=6 and org <=10 )
DO INSTEAD
INSERT INTO sales_part2 VALUES(New.org,New.name);
```

4. **Add sample data to the new table.**

```
INSERT INTO sales VALUES(1,'Craig');
INSERT INTO sales VALUES(2,'Mike');
INSERT INTO sales VALUES(3,'Michelle');
INSERT INTO sales VALUES(4,'Joe');
INSERT INTO sales VALUES(5,'Scott');
INSERT INTO sales VALUES(6,'Roger');
INSERT INTO sales VALUES(7,'Fred');
INSERT INTO sales VALUES(8,'Sam');
INSERT INTO sales VALUES(9,'Sonny');
INSERT INTO sales VALUES(10,'Chris');
```

5. **Confirm that the data was added to the parent table and the partition tables**

```
SELECT * FROM sales;

SELECT * FROM sales_part1;
SELECT * FROM sales_part2;
```

6. **Create a dblink_connect to create a connection string to use.**

   Access the file : pg_hba.conf file under C:\Program Files\PostgreSQL\9.3\data and make the following entry , stating that the host machine accessible to other machines.

```
host    all       all       all       trust
```

```
Create Extension dblink;
SELECT dblink_connect('myconn' ,'hostaddr=172.17.17.103  dbname=postgres
user=postgres password=postgres')
```
172.17.17.103 *is the host address that has the database and the partitions* .

7. **Use dblink command on the remote machine to access the partitions present in the host machine.**

   Access the file : pg_hba.conf file under C:\Program Files\PostgreSQL\9.3\data and make the following entry , stating that the remote machine needs to access the host machine:

   ```
   host    postgres    postgres       172.17.17.103/32      md5
   ```
   And the client can execute the following command, in the SQL Query window:
   sample:

   ```
   Create Extension dblink;

   SELECT dblink_connect('myconn' ,'hostaddr=172.17.17.103  dbname=sachin
   user=postgres password=postgres')


   select * from dblink('myconn','select * from sales_part2')AS T1(Column1 int,
   column2 varchar(10)) order by column2 desc;
   ```

   Inserting data into the table remotely

   ```
   select dblink_exec('myconn','insert into sales values(12,''John'')')
   select * from dblink('myconn','select * from sales')AS T1(Column1 int, column2
   varchar(10)) order by column1 asc;
   ```

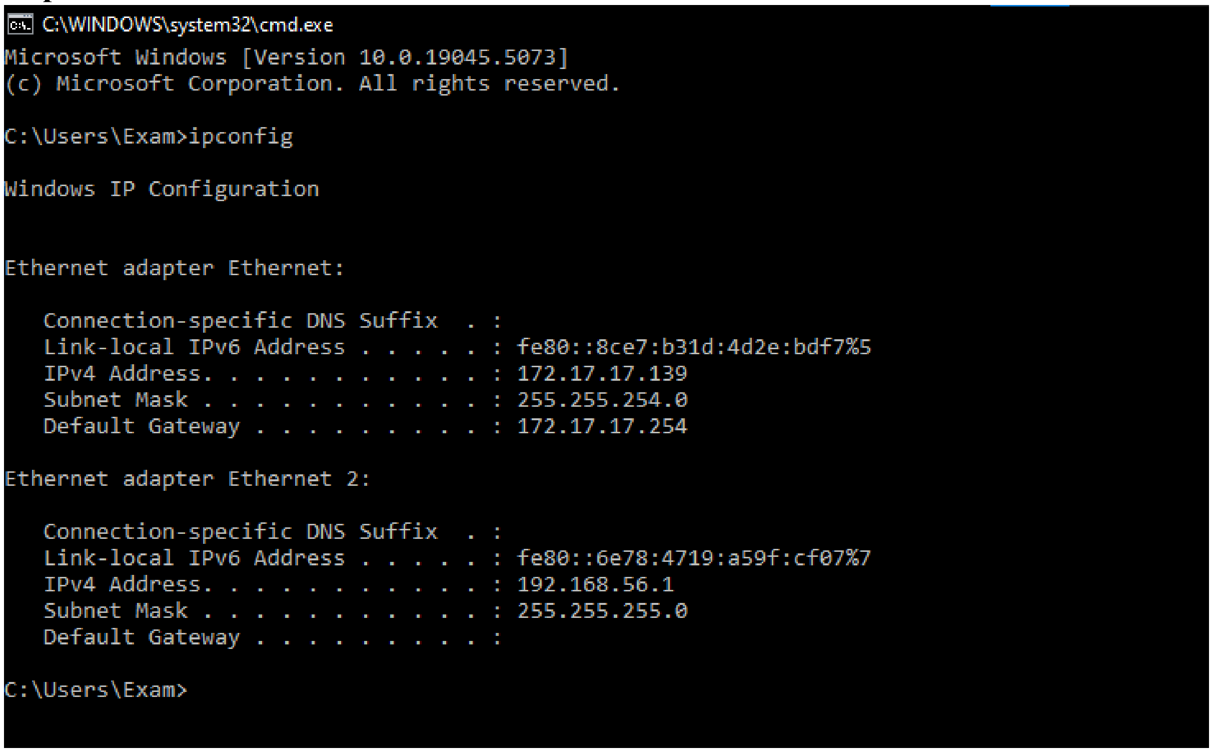
   Delete data from table remotely

   ```
   select dblink_exec('myconn','delete from sales org=3')
   select * from dblink('myconn','select * from sales')AS T1(Column1 int, column2
   varchar(10)) order by column1 asc;
   ```

\

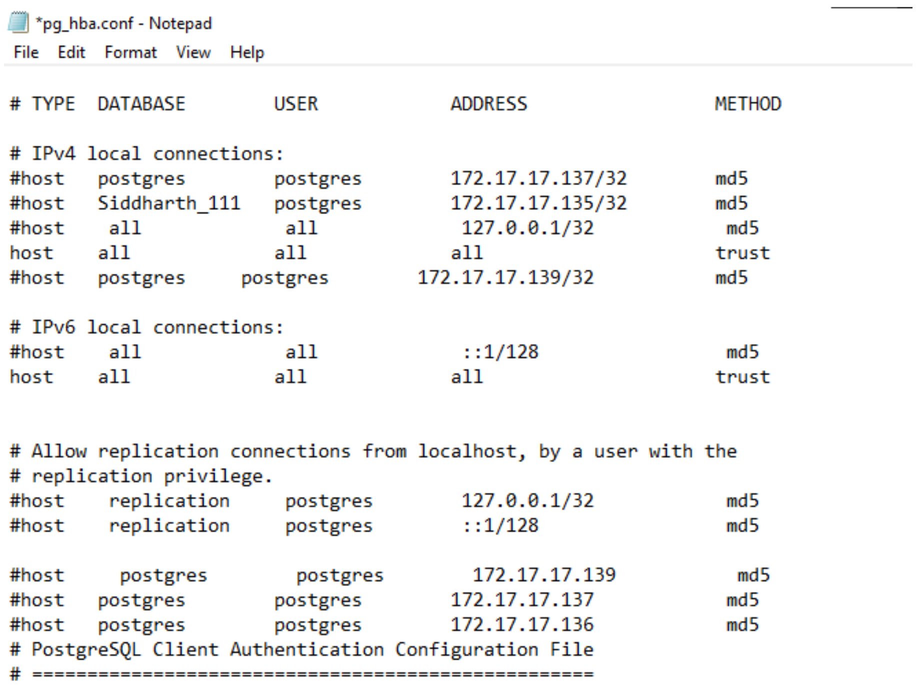
Results: (Program printout with output)

Step 1 : Get IP of client machine



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19045.5073]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Exam>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::8ce7:b31d:4d2e:bdf7%5
   IPv4 Address. . . . . . . . . . . : 172.17.17.139
   Subnet Mask . . . . . . . . . . . : 255.255.254.0
   Default Gateway . . . . . . . . . : 172.17.17.254

Ethernet adapter Ethernet 2:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::6e78:4719:a59f:cf07%7
   IPv4 Address. . . . . . . . . . . : 192.168.56.1
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :

C:\Users\Exam>
```

Step 2 : Add Ip of client machine to host config file



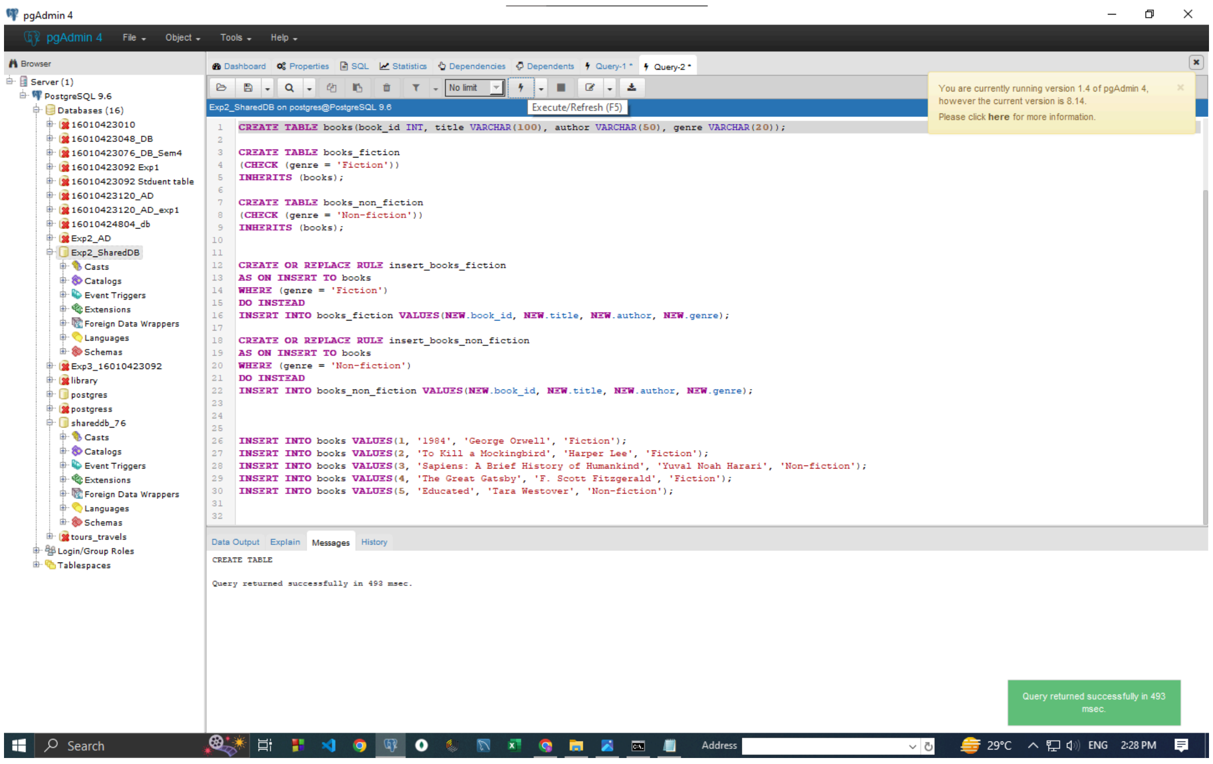
```
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# IPv4 local connections:
#host    postgres        postgres        172.17.17.137/32        md5
#host    Siddharth_111   postgres        172.17.17.135/32        md5
#host     all             all             127.0.0.1/32            md5
host     all             all             all                     trust
#host    postgres     postgres       172.17.17.139/32            md5

# IPv6 local connections:
#host     all             all             ::1/128                 md5
host     all             all             all                     trust


# Allow replication connections from localhost, by a user with the
# replication privilege.
#host     replication     postgres        127.0.0.1/32            md5
#host     replication     postgres        ::1/128                 md5

#host      postgres        postgres        172.17.17.139           md5
#host    postgres        postgres        172.17.17.137           md5
#host    postgres        postgres        172.17.17.136           md5
# PostgreSQL Client Authentication Configuration File
# ===================================================
```

Step 3 : Execute all Postgres Queries on Host machine to create tables inside Database

-- On host machine
CREATE TABLE books(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20));

CREATE TABLE books_fiction
(CHECK (genre = 'Fiction'))
INHERITS (books);

```
CREATE TABLE books_non_fiction
(CHECK (genre = 'Non-fiction'))
INHERITS (books);


CREATE OR REPLACE RULE insert_books_fiction
AS ON INSERT TO books
WHERE (genre = 'Fiction')
DO INSTEAD
INSERT INTO books_fiction VALUES(NEW.book_id, NEW.title, NEW.author,
NEW.genre);

CREATE OR REPLACE RULE insert_books_non_fiction
AS ON INSERT TO books
WHERE (genre = 'Non-fiction')
DO INSTEAD
INSERT INTO books_non_fiction VALUES(NEW.book_id, NEW.title, NEW.author,
NEW.genre);



INSERT INTO books VALUES(1, '1984', 'George Orwell', 'Fiction');
INSERT INTO books VALUES(2, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction');
INSERT INTO books VALUES(3, 'Sapiens: A Brief History of Humankind', 'Yuval Noah
Harari', 'Non-fiction');
INSERT INTO books VALUES(4, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction');
INSERT INTO books VALUES(5, 'Educated', 'Tara Westover', 'Non-fiction');



SELECT * FROM books;

SELECT * FROM books_fiction;
SELECT * FROM books_non_fiction;
```



Server (1)
PostgreSQL 9.6
Databases (16)
16010423010
16010423048_DB
16010423076_DB_Sem4
16010423092 Exp1
16010423092 Stduent table
16010423120_AD
16010423120_AD_exp1
16010424804_db
Exp2_AD
Exp2_SharedDB
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Schemas
Exp3_16010423092
library
postgres
postgress
shareddb_76
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Schemas
tours_travels
Login/Group Roles
Tablespaces

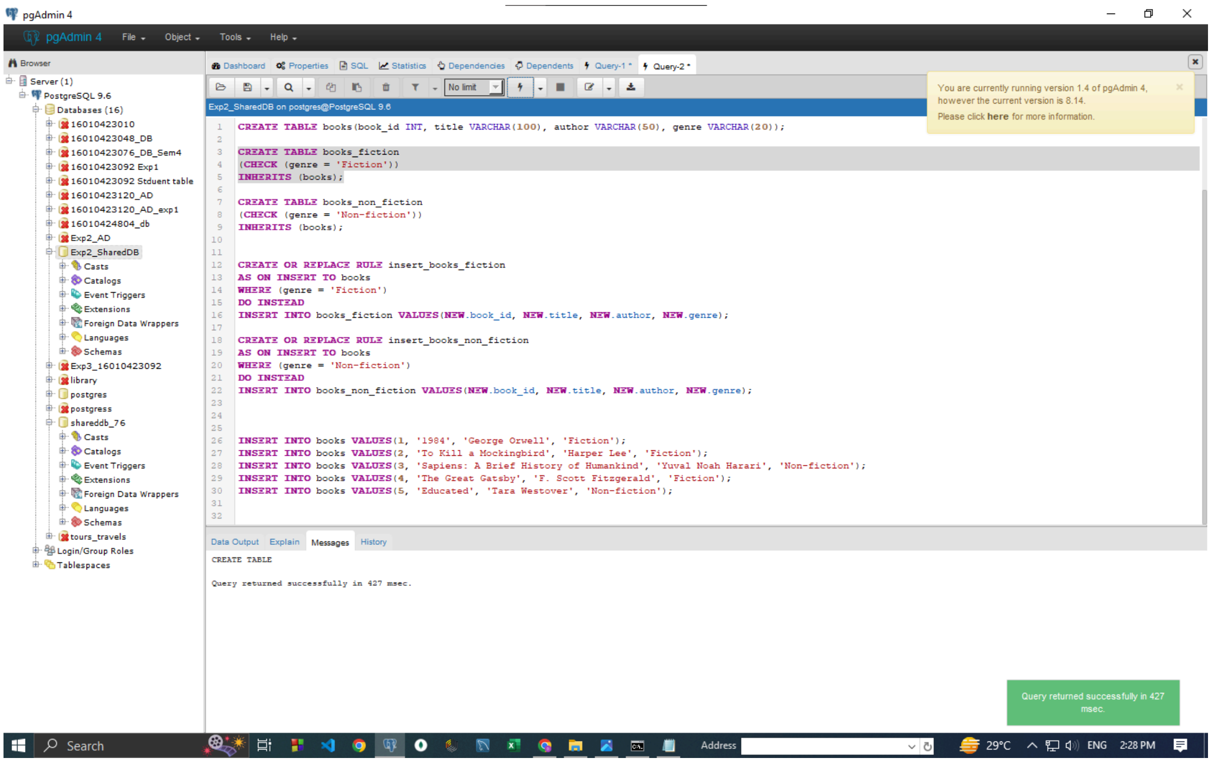
You are currently running version 1.4 of pgAdmin 4, however the current version is 8.14.
Please click **here** for more information.

Exp2_SharedDB on postgres@PostgreSQL 9.6

```
CREATE TABLE books(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20));

CREATE TABLE books_fiction
(CHECK (genre = 'Fiction'))
INHERITS (books);

CREATE TABLE books_non_fiction
(CHECK (genre = 'Non-fiction'))
INHERITS (books);


CREATE OR REPLACE RULE insert_books_fiction
AS ON INSERT TO books
WHERE (genre = 'Fiction')
DO INSTEAD
INSERT INTO books_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);

CREATE OR REPLACE RULE insert_books_non_fiction
AS ON INSERT TO books
WHERE (genre = 'Non-fiction')
DO INSTEAD
INSERT INTO books_non_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);



INSERT INTO books VALUES(1, '1984', 'George Orwell', 'Fiction');
INSERT INTO books VALUES(2, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction');
INSERT INTO books VALUES(3, 'Sapiens: A Brief History of Humankind', 'Yuval Noah Harari', 'Non-fiction');
INSERT INTO books VALUES(4, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction');
INSERT INTO books VALUES(5, 'Educated', 'Tara Westover', 'Non-fiction');
```

```
CREATE TABLE

Query returned successfully in 493 msec.
```

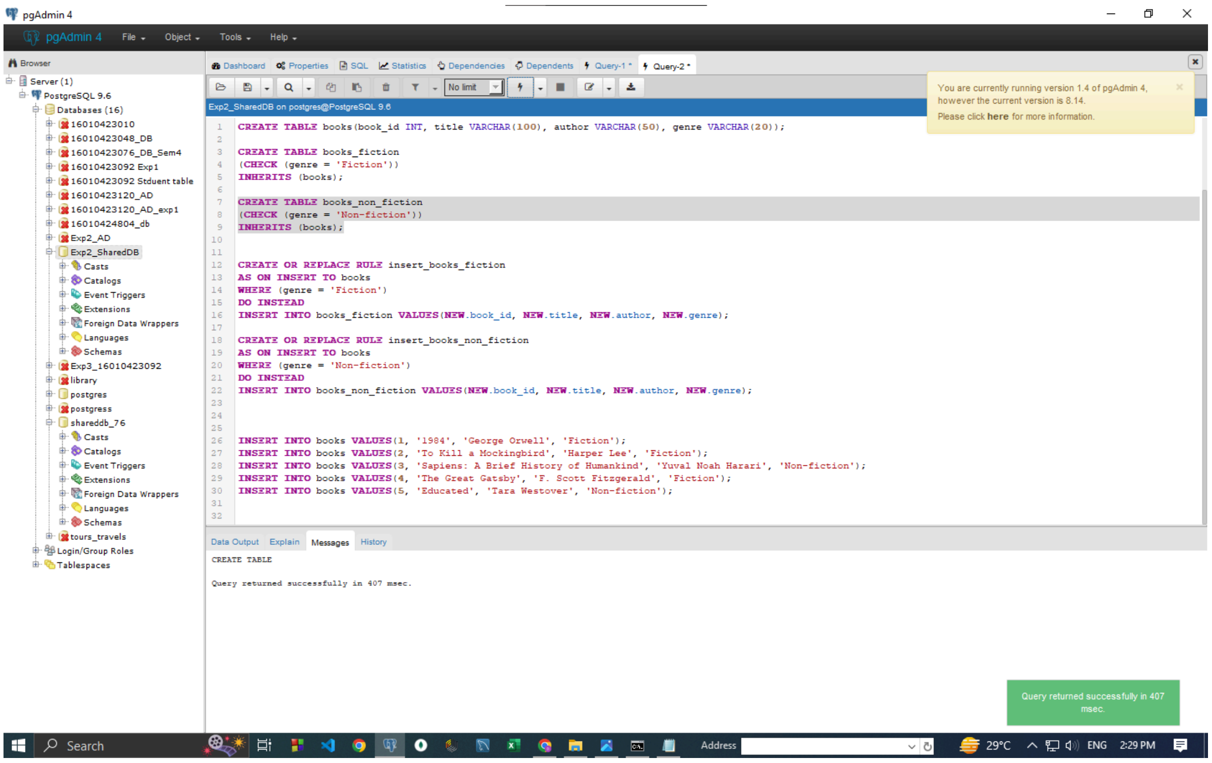
Query returned successfully in 493 msec.



```
CREATE TABLE books(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20));

CREATE TABLE books_fiction
(CHECK (genre = 'Fiction'))
INHERITS (books);

CREATE TABLE books_non_fiction
(CHECK (genre = 'Non-fiction'))
INHERITS (books);


CREATE OR REPLACE RULE insert_books_fiction
AS ON INSERT TO books
WHERE (genre = 'Fiction')
DO INSTEAD
INSERT INTO books_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);

CREATE OR REPLACE RULE insert_books_non_fiction
AS ON INSERT TO books
WHERE (genre = 'Non-fiction')
DO INSTEAD
INSERT INTO books_non_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);



INSERT INTO books VALUES(1, '1984', 'George Orwell', 'Fiction');
INSERT INTO books VALUES(2, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction');
INSERT INTO books VALUES(3, 'Sapiens: A Brief History of Humankind', 'Yuval Noah Harari', 'Non-fiction');
INSERT INTO books VALUES(4, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction');
INSERT INTO books VALUES(5, 'Educated', 'Tara Westover', 'Non-fiction');
```

```
CREATE TABLE

Query returned successfully in 427 msec.
```

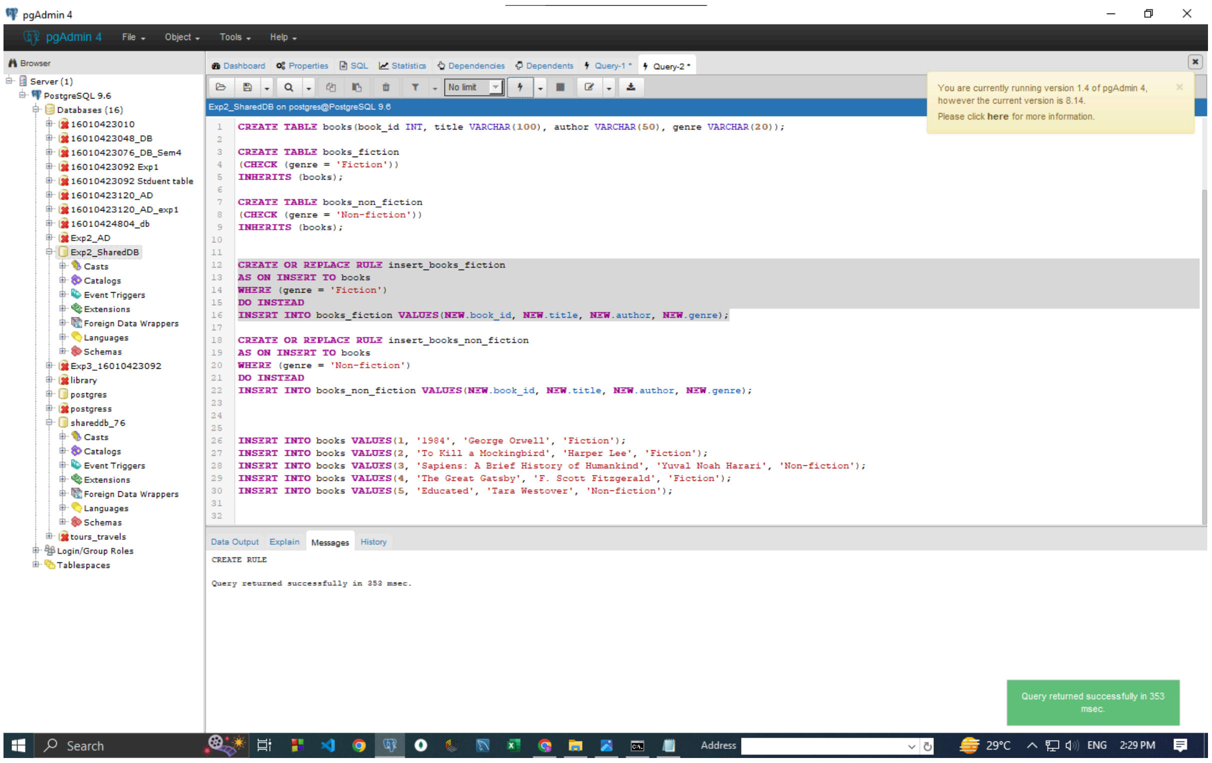
Query returned successfully in 427 msec.



```
CREATE TABLE books(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20));

CREATE TABLE books_fiction
(CHECK (genre = 'Fiction'))
INHERITS (books);

CREATE TABLE books_non_fiction
(CHECK (genre = 'Non-fiction'))
INHERITS (books);


CREATE OR REPLACE RULE insert_books_fiction
AS ON INSERT TO books
WHERE (genre = 'Fiction')
DO INSTEAD
INSERT INTO books_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);

CREATE OR REPLACE RULE insert_books_non_fiction
AS ON INSERT TO books
WHERE (genre = 'Non-fiction')
DO INSTEAD
INSERT INTO books_non_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);



INSERT INTO books VALUES(1, '1984', 'George Orwell', 'Fiction');
INSERT INTO books VALUES(2, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction');
INSERT INTO books VALUES(3, 'Sapiens: A Brief History of Humankind', 'Yuval Noah Harari', 'Non-fiction');
INSERT INTO books VALUES(4, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction');
INSERT INTO books VALUES(5, 'Educated', 'Tara Westover', 'Non-fiction');
```

```
CREATE TABLE

Query returned successfully in 407 msec.
```

Query returned successfully in 407 msec.

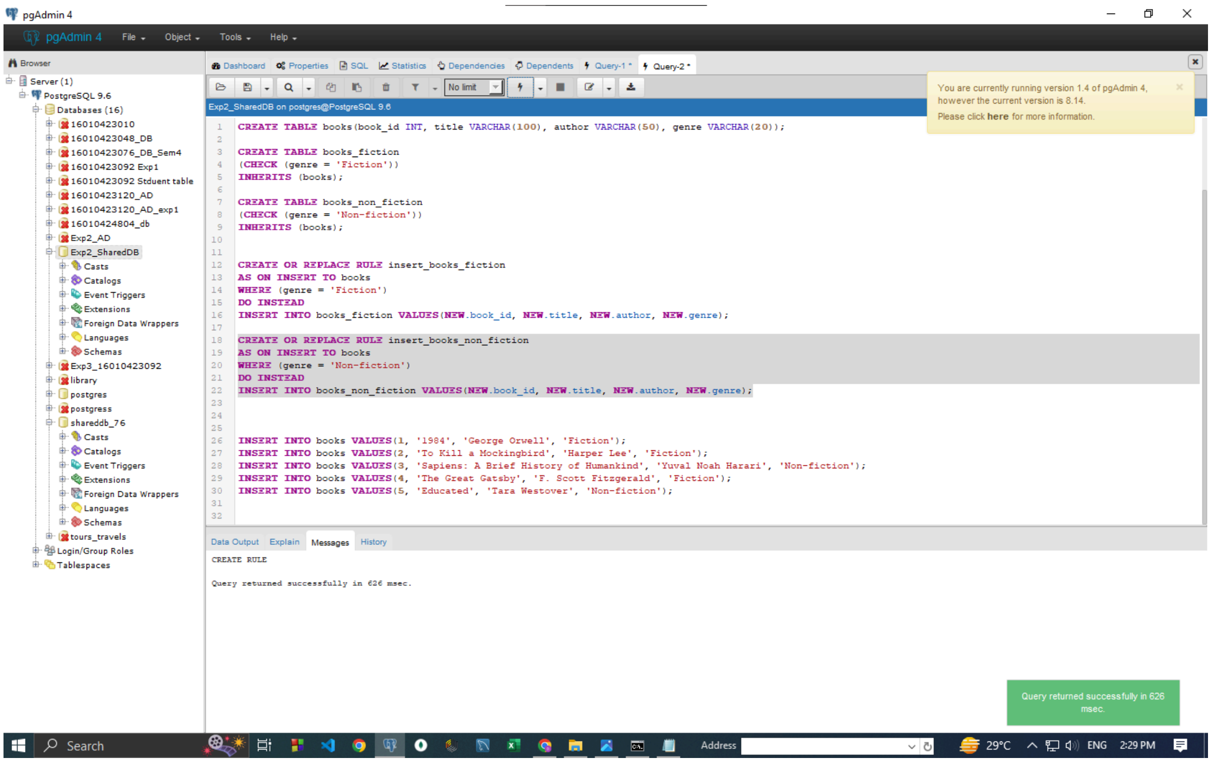


You are currently running version 1.4 of pgAdmin 4, however the current version is 8.14.
Please click **here** for more information.

```sql
CREATE TABLE books(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20));

CREATE TABLE books_fiction
(CHECK (genre = 'Fiction'))
INHERITS (books);

CREATE TABLE books_non_fiction
(CHECK (genre = 'Non-fiction'))
INHERITS (books);


CREATE OR REPLACE RULE insert_books_fiction
AS ON INSERT TO books
WHERE (genre = 'Fiction')
DO INSTEAD
INSERT INTO books_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);

CREATE OR REPLACE RULE insert_books_non_fiction
AS ON INSERT TO books
WHERE (genre = 'Non-fiction')
DO INSTEAD
INSERT INTO books_non_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);



INSERT INTO books VALUES(1, '1984', 'George Orwell', 'Fiction');
INSERT INTO books VALUES(2, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction');
INSERT INTO books VALUES(3, 'Sapiens: A Brief History of Humankind', 'Yuval Noah Harari', 'Non-fiction');
INSERT INTO books VALUES(4, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction');
INSERT INTO books VALUES(5, 'Educated', 'Tara Westover', 'Non-fiction');
```

```
CREATE RULE

Query returned successfully in 353 msec.
```

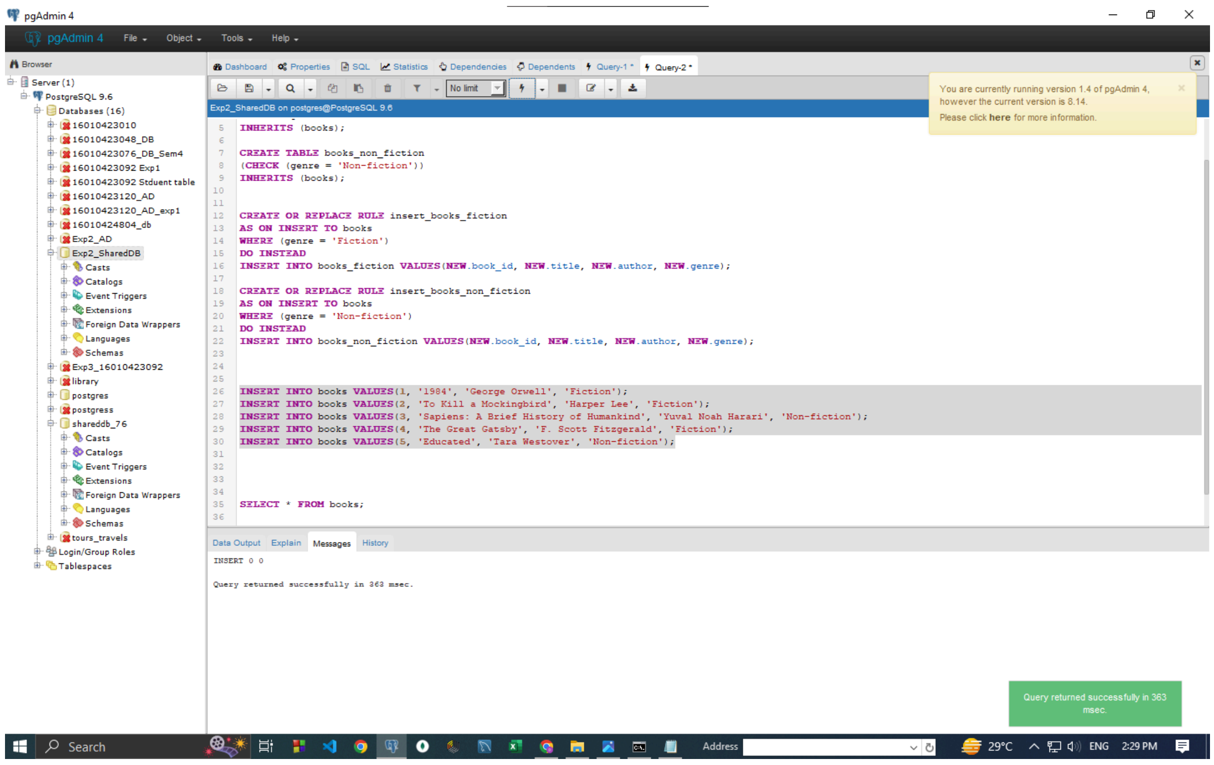
Query returned successfully in 353 msec.


You are currently running version 1.4 of pgAdmin 4, however the current version is 8.14.
Please click **here** for more information.

```sql
CREATE TABLE books(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20));

CREATE TABLE books_fiction
(CHECK (genre = 'Fiction'))
INHERITS (books);

CREATE TABLE books_non_fiction
(CHECK (genre = 'Non-fiction'))
INHERITS (books);


CREATE OR REPLACE RULE insert_books_fiction
AS ON INSERT TO books
WHERE (genre = 'Fiction')
DO INSTEAD
INSERT INTO books_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);

CREATE OR REPLACE RULE insert_books_non_fiction
AS ON INSERT TO books
WHERE (genre = 'Non-fiction')
DO INSTEAD
INSERT INTO books_non_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);



INSERT INTO books VALUES(1, '1984', 'George Orwell', 'Fiction');
INSERT INTO books VALUES(2, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction');
INSERT INTO books VALUES(3, 'Sapiens: A Brief History of Humankind', 'Yuval Noah Harari', 'Non-fiction');
INSERT INTO books VALUES(4, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction');
INSERT INTO books VALUES(5, 'Educated', 'Tara Westover', 'Non-fiction');
```

```
CREATE RULE

Query returned successfully in 626 msec.
```

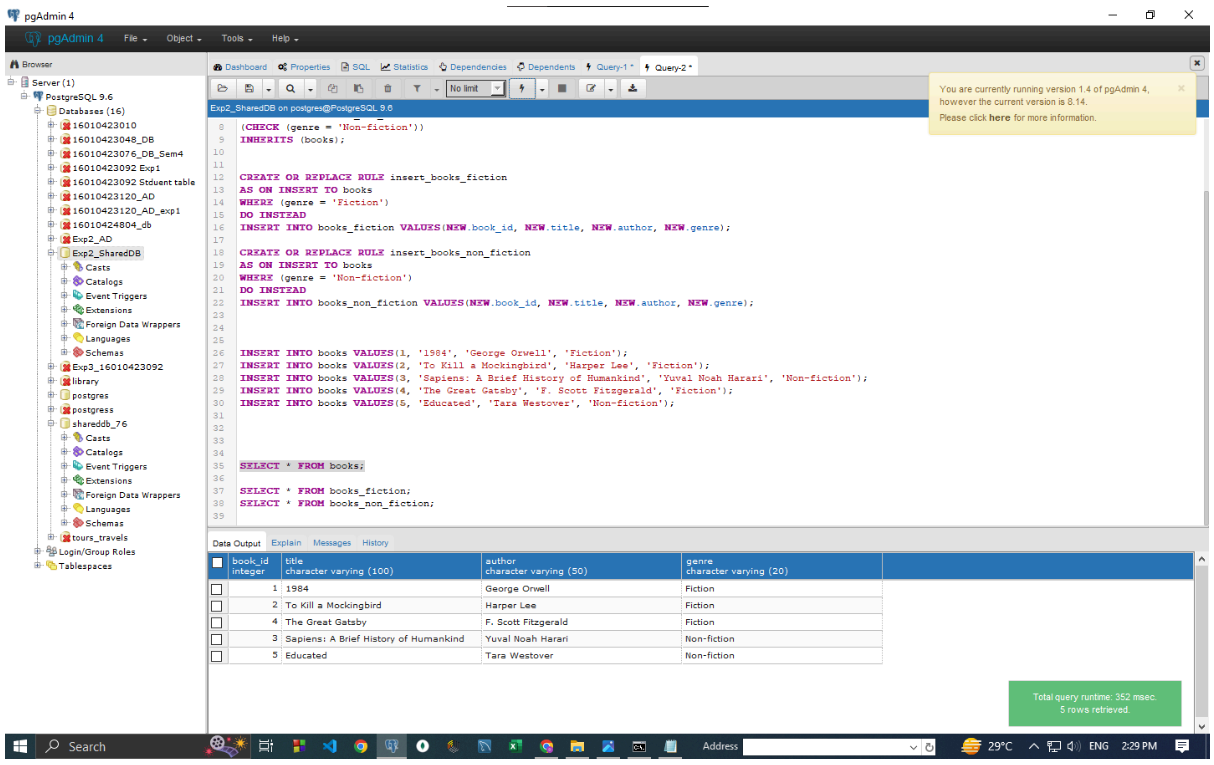
Query returned successfully in 626 msec.



Exp2_SharedDB on postgres@PostgreSQL 9.6

```sql
INHERITS (books);

CREATE TABLE books_non_fiction
(CHECK (genre = 'Non-fiction'))
INHERITS (books);


CREATE OR REPLACE RULE insert_books_fiction
AS ON INSERT TO books
WHERE (genre = 'Fiction')
DO INSTEAD
INSERT INTO books_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);

CREATE OR REPLACE RULE insert_books_non_fiction
AS ON INSERT TO books
WHERE (genre = 'Non-fiction')
DO INSTEAD
INSERT INTO books_non_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);



INSERT INTO books VALUES(1, '1984', 'George Orwell', 'Fiction');
INSERT INTO books VALUES(2, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction');
INSERT INTO books VALUES(3, 'Sapiens: A Brief History of Humankind', 'Yuval Noah Harari', 'Non-fiction');
INSERT INTO books VALUES(4, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction');
INSERT INTO books VALUES(5, 'Educated', 'Tara Westover', 'Non-fiction');




SELECT * FROM books;
```

```
INSERT 0 0

Query returned successfully in 363 msec.
```

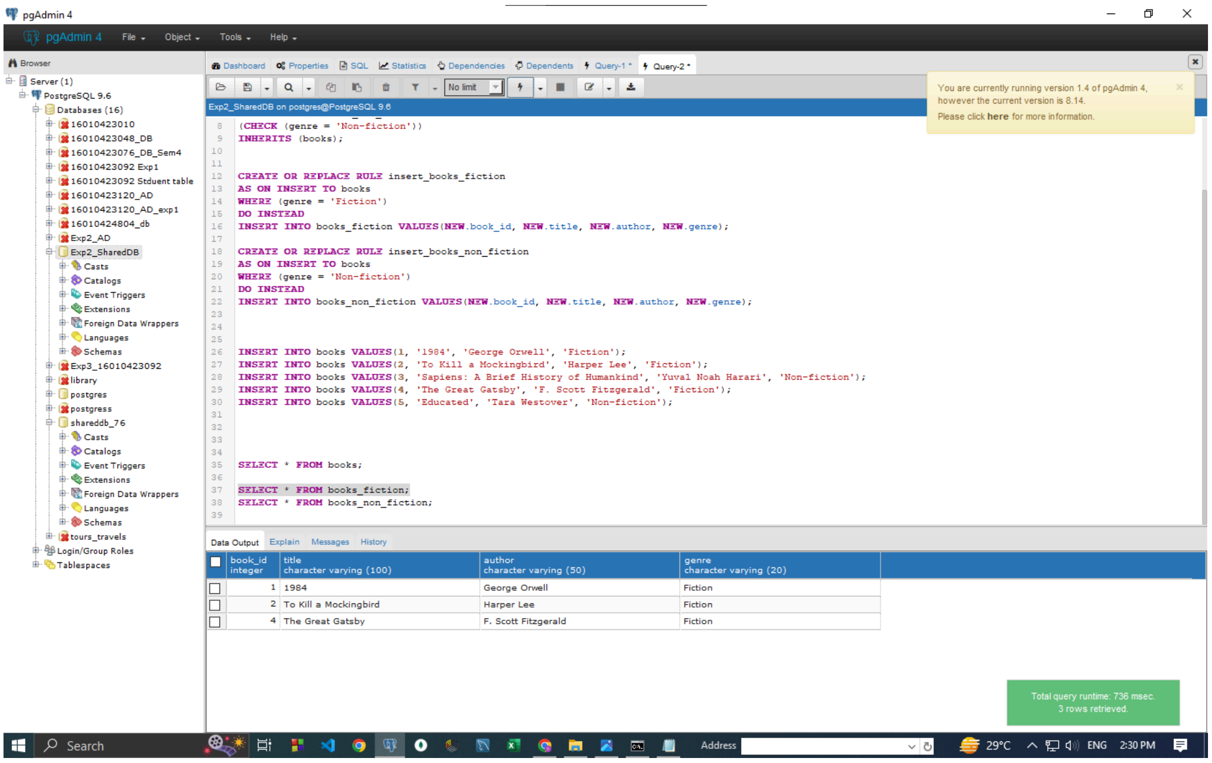


Exp2_SharedDB on postgres@PostgreSQL 9.6

```sql
(CHECK (genre = 'Non-fiction'))
INHERITS (books);


CREATE OR REPLACE RULE insert_books_fiction
AS ON INSERT TO books
WHERE (genre = 'Fiction')
DO INSTEAD
INSERT INTO books_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);

CREATE OR REPLACE RULE insert_books_non_fiction
AS ON INSERT TO books
WHERE (genre = 'Non-fiction')
DO INSTEAD
INSERT INTO books_non_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);



INSERT INTO books VALUES(1, '1984', 'George Orwell', 'Fiction');
INSERT INTO books VALUES(2, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction');
INSERT INTO books VALUES(3, 'Sapiens: A Brief History of Humankind', 'Yuval Noah Harari', 'Non-fiction');
INSERT INTO books VALUES(4, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction');
INSERT INTO books VALUES(5, 'Educated', 'Tara Westover', 'Non-fiction');




SELECT * FROM books;

SELECT * FROM books_fiction;
SELECT * FROM books_non_fiction;
```

book_id integer	title character varying (100)	author character varying (50)	genre character varying (20)
1	1984	George Orwell	Fiction
2	To Kill a Mockingbird	Harper Lee	Fiction
4	The Great Gatsby	F. Scott Fitzgerald	Fiction
3	Sapiens: A Brief History of Humankind	Yuval Noah Harari	Non-fiction
5	Educated	Tara Westover	Non-fiction

Total query runtime: 352 msec.
5 rows retrieved.

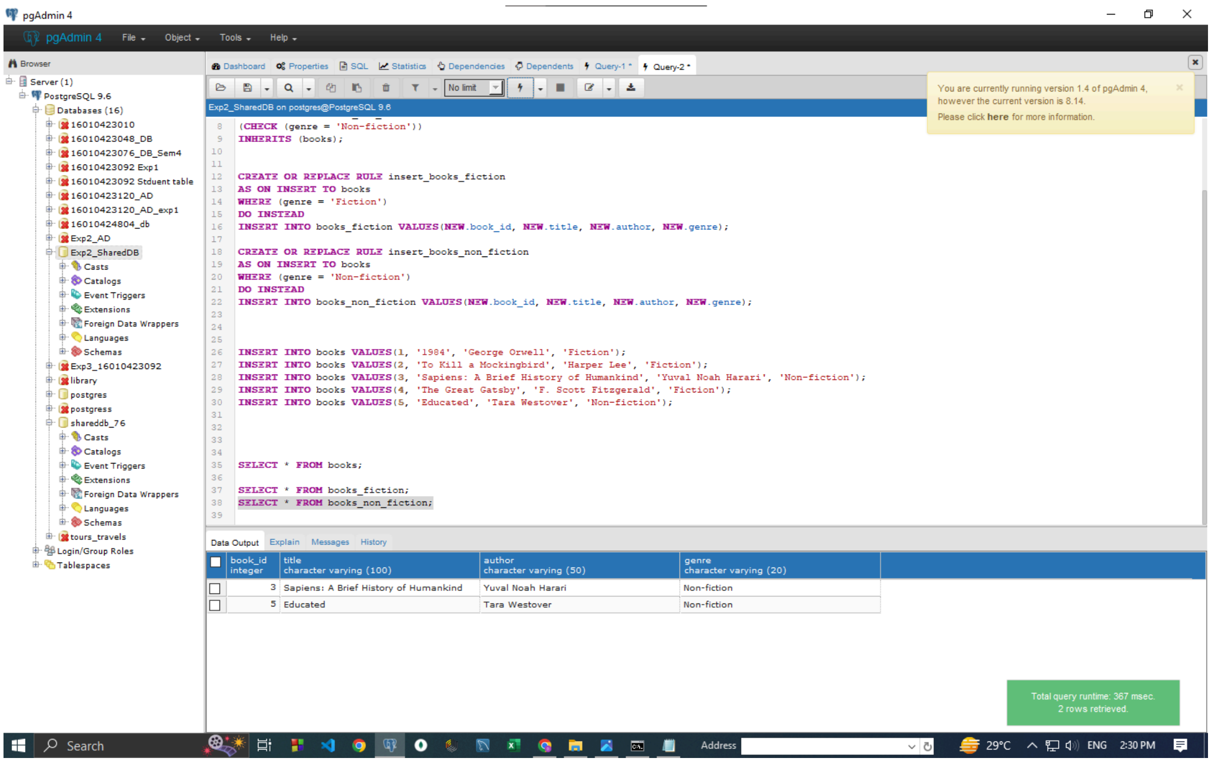


Exp2_SharedDB on postgres@PostgreSQL 9.6

```sql
(CHECK (genre = 'Non-fiction'))
INHERITS (books);


CREATE OR REPLACE RULE insert_books_fiction
AS ON INSERT TO books
WHERE (genre = 'Fiction')
DO INSTEAD
INSERT INTO books_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);

CREATE OR REPLACE RULE insert_books_non_fiction
AS ON INSERT TO books
WHERE (genre = 'Non-fiction')
DO INSTEAD
INSERT INTO books_non_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);



INSERT INTO books VALUES(1, '1984', 'George Orwell', 'Fiction');
INSERT INTO books VALUES(2, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction');
INSERT INTO books VALUES(3, 'Sapiens: A Brief History of Humankind', 'Yuval Noah Harari', 'Non-fiction');
INSERT INTO books VALUES(4, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction');
INSERT INTO books VALUES(5, 'Educated', 'Tara Westover', 'Non-fiction');




SELECT * FROM books;

SELECT * FROM books_fiction;
SELECT * FROM books_non_fiction;
```

book_id integer	title character varying (100)	author character varying (50)	genre character varying (20)
1	1984	George Orwell	Fiction
2	To Kill a Mockingbird	Harper Lee	Fiction
4	The Great Gatsby	F. Scott Fitzgerald	Fiction

Total query runtime: 736 msec.
3 rows retrieved.



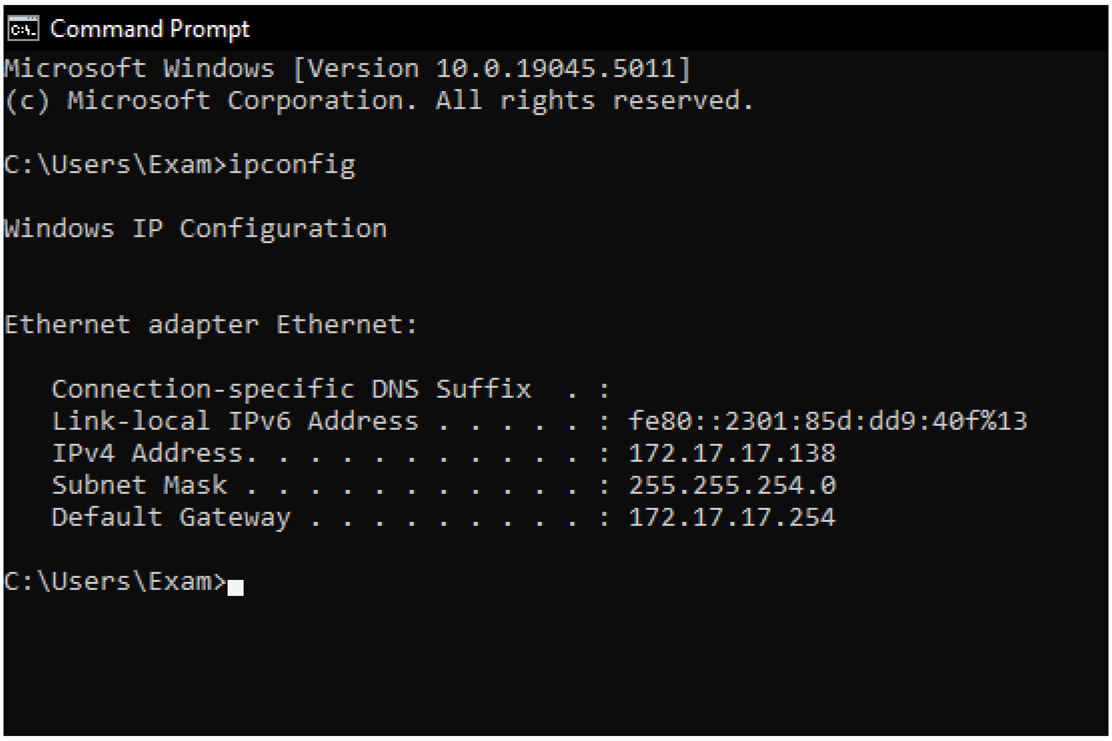
Exp2_SharedDB on postgres@PostgreSQL 9.6

```sql
(CHECK (genre = 'Non-fiction'))
INHERITS (books);


CREATE OR REPLACE RULE insert_books_fiction
AS ON INSERT TO books
WHERE (genre = 'Fiction')
DO INSTEAD
INSERT INTO books_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);

CREATE OR REPLACE RULE insert_books_non_fiction
AS ON INSERT TO books
WHERE (genre = 'Non-fiction')
DO INSTEAD
INSERT INTO books_non_fiction VALUES(NEW.book_id, NEW.title, NEW.author, NEW.genre);



INSERT INTO books VALUES(1, '1984', 'George Orwell', 'Fiction');
INSERT INTO books VALUES(2, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction');
INSERT INTO books VALUES(3, 'Sapiens: A Brief History of Humankind', 'Yuval Noah Harari', 'Non-fiction');
INSERT INTO books VALUES(4, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction');
INSERT INTO books VALUES(5, 'Educated', 'Tara Westover', 'Non-fiction');




SELECT * FROM books;

SELECT * FROM books_fiction;
SELECT * FROM books_non_fiction;
```

book_id integer	title character varying (100)	author character varying (50)	genre character varying (20)
3	Sapiens: A Brief History of Humankind	Yuval Noah Harari	Non-fiction
5	Educated	Tara Westover	Non-fiction

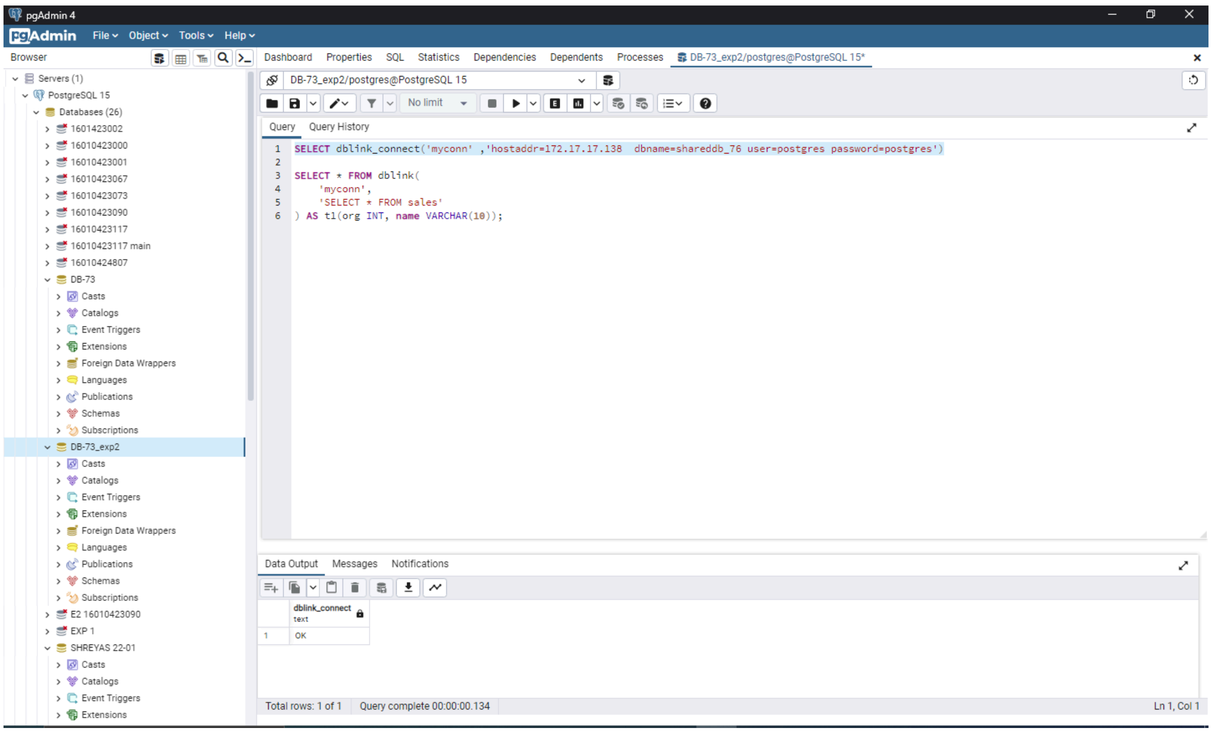
Total query runtime: 367 msec.
2 rows retrieved.

Step 4 : Get IP address of host machine



```
Command Prompt
Microsoft Windows [Version 10.0.19045.5011]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Exam>ipconfig

Windows IP Configuration


Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::2301:85d:dd9:40f%13
   IPv4 Address. . . . . . . . . . . : 172.17.17.138
   Subnet Mask . . . . . . . . . . . : 255.255.254.0
   Default Gateway . . . . . . . . . : 172.17.17.254

C:\Users\Exam>
```

Step 5 : On Client machine execute the below dblink command to check if connect has been established



```
DB-73_exp2/postgres@PostgreSQL 15

SELECT dblink_connect('myconn' ,'hostaddr=172.17.17.138  dbname=shareddb_76 user=postgres password=postgres')

SELECT * FROM dblink(
    'myconn',
    'SELECT * FROM sales'
) AS t1(org INT, name VARCHAR(10));
```

```
   dblink_connect
   text
1  OK

Total rows: 1 of 1    Query complete 00:00:00.134    Ln 1, Col 1
```

Step 6 : Run postgres commands to make changes on the shared database on the host machine

--On client machine
CREATE EXTENSION dblink;

SELECT dblink_connect('myconnection', 'hostaddr=172.17.17.138 dbname=Exp2_SharedDB user=postgres password=postgres');

--Query remote partitions

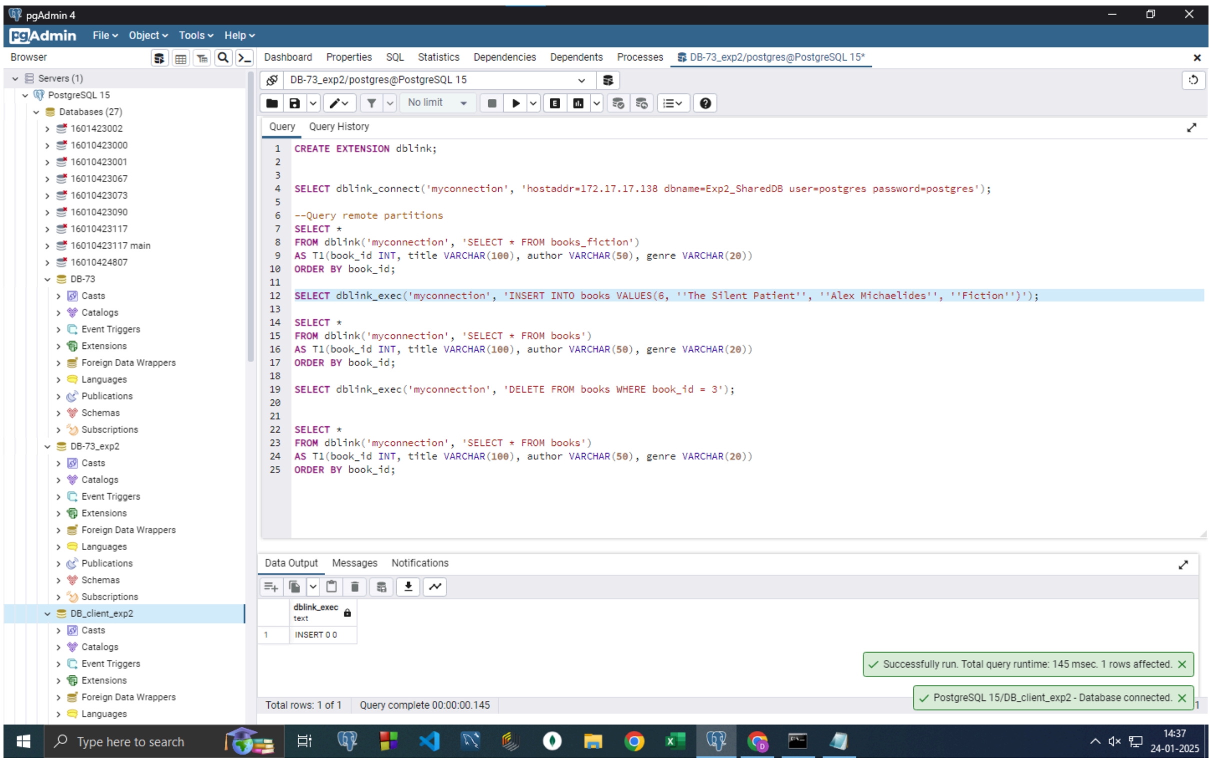
SELECT *
FROM dblink('myconnection', 'SELECT * FROM books_fiction')
AS T1(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20))
ORDER BY book_id;

–Insert
SELECT dblink_exec('myconnection', 'INSERT INTO books VALUES(6, "The Silent Patient", "Alex Michaelides", "Fiction")');

–Verify insertion
SELECT *
FROM dblink('myconnection', 'SELECT * FROM books')
AS T1(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20))
ORDER BY book_id;

–Delete
SELECT dblink_exec('myconnection', 'DELETE FROM books WHERE book_id = 3');

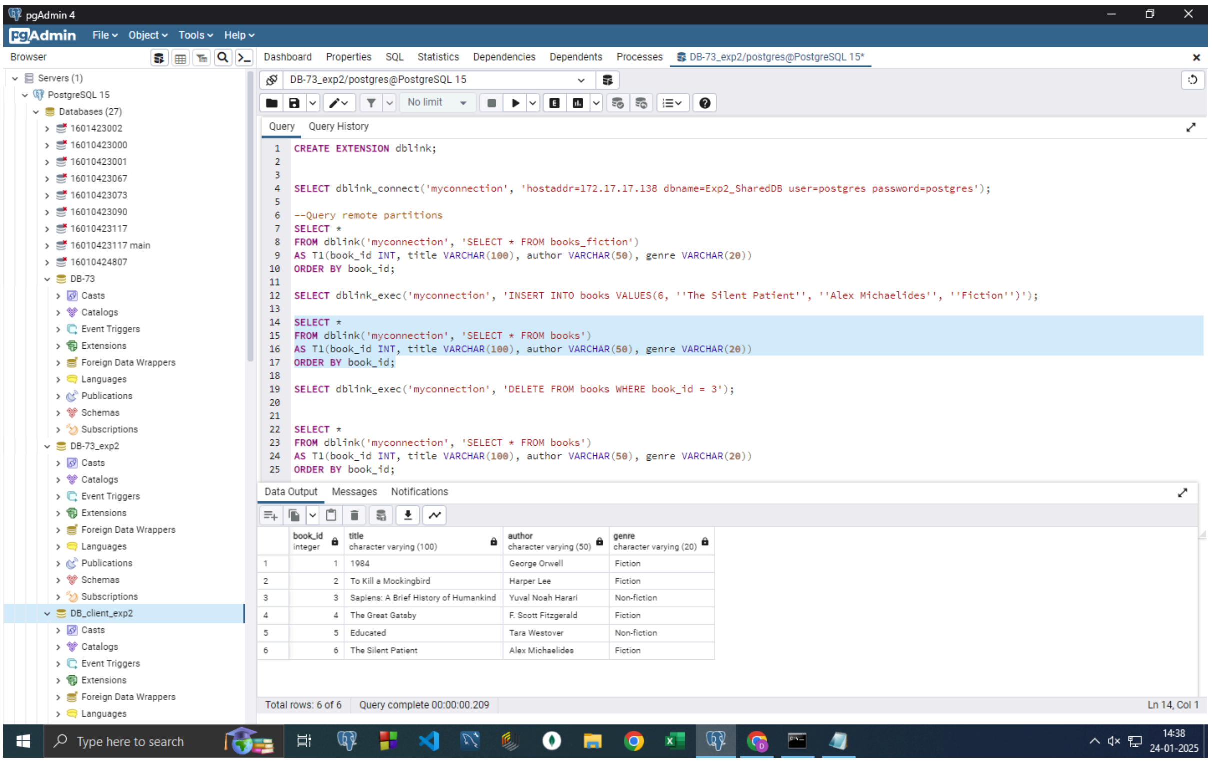
–Verify Deletion
SELECT *
FROM dblink('myconnection', 'SELECT * FROM books')
AS T1(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20))
ORDER BY book_id;



```
CREATE EXTENSION dblink;


SELECT dblink_connect('myconnection', 'hostaddr=172.17.17.138 dbname=Exp2_SharedDB user=postgres password=postgres');

--Query remote partitions
SELECT *
FROM dblink('myconnection', 'SELECT * FROM books_fiction')
AS T1(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20))
ORDER BY book_id;

SELECT dblink_exec('myconnection', 'INSERT INTO books VALUES(6, ''The Silent Patient'', ''Alex Michaelides'', ''Fiction'')');

SELECT *
FROM dblink('myconnection', 'SELECT * FROM books')
AS T1(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20))
ORDER BY book_id;

SELECT dblink_exec('myconnection', 'DELETE FROM books WHERE book_id = 3');


SELECT *
FROM dblink('myconnection', 'SELECT * FROM books')
AS T1(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20))
ORDER BY book_id;
```

	dblink_exec text
1	INSERT 0 0

```
Successfully run. Total query runtime: 145 msec. 1 rows affected.
PostgreSQL 15/DB_client_exp2 - Database connected.
Total rows: 1 of 1    Query complete 00:00:00.145
```



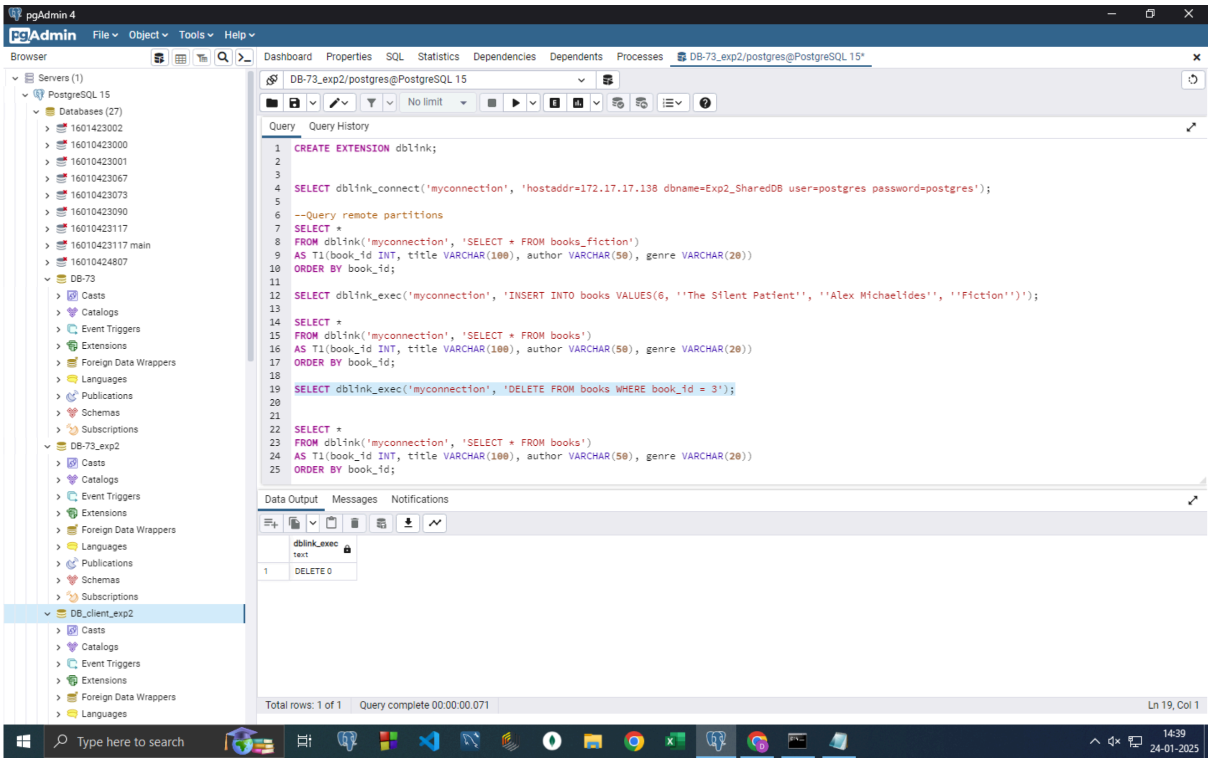
DB-73_exp2/postgres@PostgreSQL 15

```sql
CREATE EXTENSION dblink;


SELECT dblink_connect('myconnection', 'hostaddr=172.17.17.138 dbname=Exp2_SharedDB user=postgres password=postgres');

--Query remote partitions
SELECT *
FROM dblink('myconnection', 'SELECT * FROM books_fiction')
AS T1(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20))
ORDER BY book_id;

SELECT dblink_exec('myconnection', 'INSERT INTO books VALUES(6, ''The Silent Patient'', ''Alex Michaelides'', ''Fiction'')');

SELECT *
FROM dblink('myconnection', 'SELECT * FROM books')
AS T1(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20))
ORDER BY book_id;

SELECT dblink_exec('myconnection', 'DELETE FROM books WHERE book_id = 3');


SELECT *
FROM dblink('myconnection', 'SELECT * FROM books')
AS T1(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20))
ORDER BY book_id;
```

Data Output

	book_id integer	title character varying (100)	author character varying (50)	genre character varying (20)
1	1	1984	George Orwell	Fiction
2	2	To Kill a Mockingbird	Harper Lee	Fiction
3	3	Sapiens: A Brief History of Humankind	Yuval Noah Harari	Non-fiction
4	4	The Great Gatsby	F. Scott Fitzgerald	Fiction
5	5	Educated	Tara Westover	Non-fiction
6	6	The Silent Patient	Alex Michaelides	Fiction

Total rows: 6 of 6 Query complete 00:00:00.209 Ln 14, Col 1



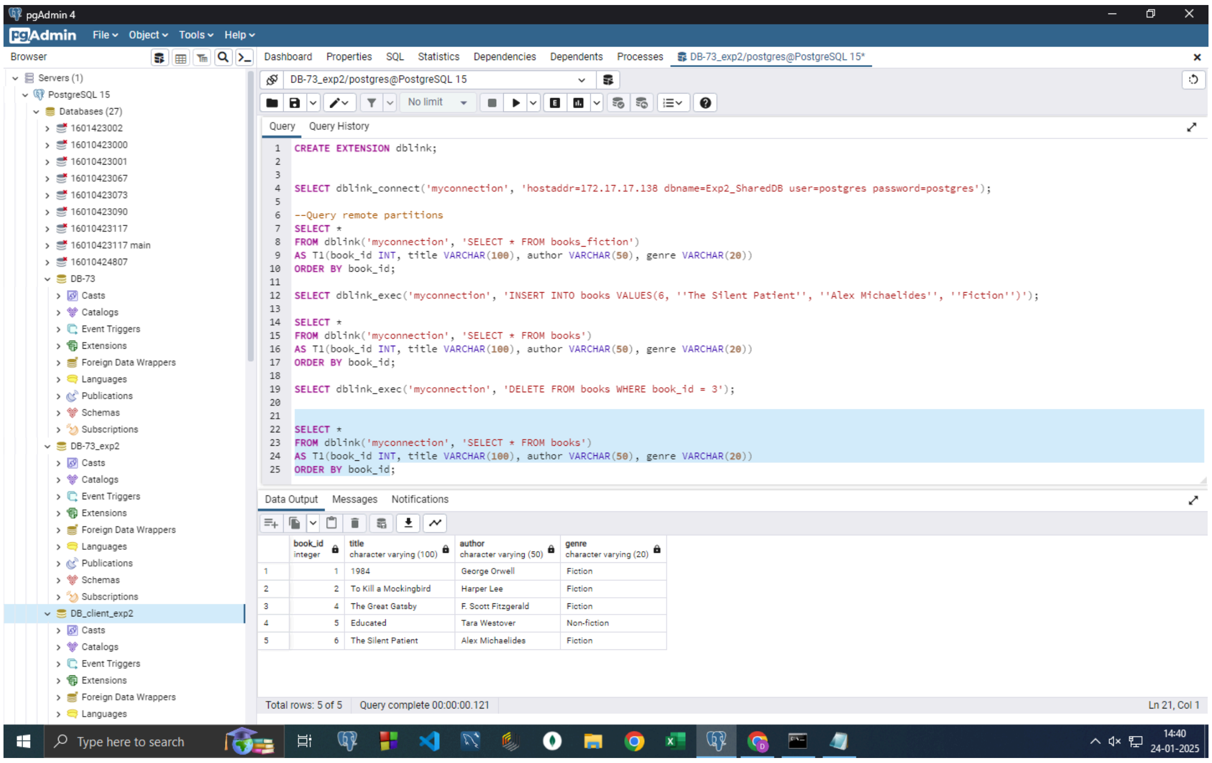
DB-73_exp2/postgres@PostgreSQL 15

```sql
CREATE EXTENSION dblink;


SELECT dblink_connect('myconnection', 'hostaddr=172.17.17.138 dbname=Exp2_SharedDB user=postgres password=postgres');

--Query remote partitions
SELECT *
FROM dblink('myconnection', 'SELECT * FROM books_fiction')
AS T1(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20))
ORDER BY book_id;

SELECT dblink_exec('myconnection', 'INSERT INTO books VALUES(6, ''The Silent Patient'', ''Alex Michaelides'', ''Fiction'')');

SELECT *
FROM dblink('myconnection', 'SELECT * FROM books')
AS T1(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20))
ORDER BY book_id;

SELECT dblink_exec('myconnection', 'DELETE FROM books WHERE book_id = 3');


SELECT *
FROM dblink('myconnection', 'SELECT * FROM books')
AS T1(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20))
ORDER BY book_id;
```

Data Output

	dblink_exec text
1	DELETE 0

Total rows: 1 of 1 Query complete 00:00:00.071 Ln 19, Col 1



```sql
CREATE EXTENSION dblink;


SELECT dblink_connect('myconnection', 'hostaddr=172.17.17.138 dbname=Exp2_SharedDB user=postgres password=postgres');

--Query remote partitions
SELECT *
FROM dblink('myconnection', 'SELECT * FROM books_fiction')
AS T1(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20))
ORDER BY book_id;

SELECT dblink_exec('myconnection', 'INSERT INTO books VALUES(6, ''The Silent Patient'', ''Alex Michaelides'', ''Fiction'')');

SELECT *
FROM dblink('myconnection', 'SELECT * FROM books')
AS T1(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20))
ORDER BY book_id;

SELECT dblink_exec('myconnection', 'DELETE FROM books WHERE book_id = 3');


SELECT *
FROM dblink('myconnection', 'SELECT * FROM books')
AS T1(book_id INT, title VARCHAR(100), author VARCHAR(50), genre VARCHAR(20))
ORDER BY book_id;
```

	book_id integer	title character varying (100)	author character varying (50)	genre character varying (20)
1	1	1984	George Orwell	Fiction
2	2	To Kill a Mockingbird	Harper Lee	Fiction
3	4	The Great Gatsby	F. Scott Fitzgerald	Fiction
4	5	Educated	Tara Westover	Non-fiction
5	6	The Silent Patient	Alex Michaelides	Fiction

Total rows: 5 of 5   Query complete 00:00:00.121

Questions:

1. **What are the different types of distributed database systems?**
   Homogeneous Distributed Databases:
   These systems consist of databases that are identical in terms of structure, hardware, and software. They work seamlessly together and provide a unified interface to users.

   Heterogeneous Distributed Databases:
   This type includes databases with different architectures, software, or schemas. Middleware is required to bridge the differences and manage interactions.

   Federated Databases:
   Independent databases come together to form a federation. Each database maintains its autonomy while supporting collaboration and data sharing.

   Client-Server Databases:
   The architecture divides tasks between client machines, which make requests, and servers, which store and manage data processing.

   Peer-to-Peer Databases:
   This system operates without a central server, as all nodes are equal participants in data management and sharing.

2. **Give steps to insert and delete records in the remote table.**
   Steps for Inserting Records:
   Initiate a secure connection to the remote database by providing the host address, username, and password.
   Create an INSERT statement using the appropriate syntax to define the target table and the values you want to insert.
   Execute the INSERT command via a database interface, API, or command-line tool.
   Confirm the operation by querying the table to check if the new record is present.

Ex:

```
-- Establish a connection to the remote database first, then execute:
INSERT INTO remote_schema.table_name (column1, column2, column3)
VALUES ('value1', 'value2', 'value3');
```

Steps for Deleting Records:
Open a connection to the remote database using valid authentication credentials.
Write a DELETE query, including conditions to specify which records should be removed (e.g., DELETE FROM table_name WHERE condition).
Execute the DELETE command using a database management tool or interface.
Check the table to ensure that the specified records have been successfully deleted.
Ex:

```
-- Connect to the remote database and execute:
DELETE FROM remote_schema.table_name
WHERE column1 = 'value1' AND column2 = 'value2';
```

Outcomes:
CO1: Design advanced database systems using Parallel, Distributed, Object Relational Databases and its implementation.

Conclusion: (Conclusion to be based on the outcomes achieved):
From this experiment I learned how to implement a distributed database using PostgreSQL and the importance of partitioning data for better performance and manageability. By creating parent and child tables along with rules for data insertion I understood how to organize large datasets efficiently. Using dblink to connect and interact with remote databases demonstrated the practical aspects of distributed systems. This experiment highlighted the benefits of distributed databases in improving query performance and data availability while also showcasing the steps to insert and delete records remotely.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

1. Elmasri and Navathe, “Fundamentals of Database Systems”, Pearson Education

2. https://www.postgresql.org/docs/