Lab 1 Output:

1. Create a Database BankDB:

```
mysql> create database BookDB;
Query OK, 1 row affected (0.01 sec)

mysql> use BookDB
Database changed
```

2. Creating Tables:
   a. Creating customers table:

```
mysql> create table customers (CustomerID int not null auto_increment, Name
varchar(50), Address varchar(100), Phone varchar(15), primary key (CustomerI
D));
Query OK, 0 rows affected (0.02 sec)
```

   b. Creating accounts table:

```
mysql> create table accounts (AccountID int not null auto_increment, Custome
rID int, AccountType varchar(100), Balance Decimal(10,2), primary key (Accou
ntID), foreign key (CustomerID) references customers(CustomerID));
Query OK, 0 rows affected (0.03 sec)
```

   c. Creating transactions table:

```
mysql> create table transactions (TransactionID int not null auto_increment,
 AccountID int, TransactionType varchar(100), Amount decimal(10,2),Transacti
onDate datetime default current_timestamp, primary key (TransactionID), fore
ign key (AccountID) references accounts(AccountID));
Query OK, 0 rows affected (0.03 sec)
```

3. Inserting data into the tables:
   a. Inserting into customers table:

```
mysql> insert into customers (Name, Address, Phone) values ("Ram", "KTM", "9
811111111");
Query OK, 1 row affected (0.01 sec)

mysql> insert into customers (Name, Address, Phone) values ("Hari", "Lalitpu
r", "9822222222");
Query OK, 1 row affected (0.00 sec)
```

   b. Inserting into accounts table:

```
mysql> insert into accounts (CustomerID, AccountType, Balance) values (1, "s
aving", 6000);
Query OK, 1 row affected (0.01 sec)

mysql> insert into accounts (CustomerID, AccountType, Balance) values (2, "c
urrent", 1000);
Query OK, 1 row affected (0.01 sec)
```

c. Inserting into transactions table:

```
mysql> insert into transactions (AccountID, TransactionType, Amount) values
(1, "deposit", 1000);
Query OK, 1 row affected (0.01 sec)

mysql> insert into transactions (AccountID, TransactionType, Amount) values
(3, "withdrawal", 6000);
Query OK, 1 row affected (0.01 sec)
```

4. Retrieve all customers and their details:

```
mysql> select * from customers;
+------------+-------+----------+------------+
| CustomerID | Name  | Address  | Phone      |
+------------+-------+----------+------------+
|          1 | Ram   | KTM      | 9811111111 |
|          2 | Hari  | Lalitpur | 9822222222 |
|          3 | Ronil | Lalitpur | 9833333333 |
+------------+-------+----------+------------+
3 rows in set (0.00 sec)
```

5. List all accounts with their balances:

```
mysql> select * from accounts;
+-----------+------------+-------------+----------+
| AccountID | CustomerID | AccountType | Balance  |
+-----------+------------+-------------+----------+
|         1 |          1 | saving      |  6000.00 |
|         2 |          2 | current     |  1000.00 |
|         3 |          3 | current     | 10000.00 |
+-----------+------------+-------------+----------+
3 rows in set (0.00 sec)
```

6. Find transactions for a specific account:

```
mysql> select * from transactions where AccountID = 102;
+---------------+-----------+-----------------+---------+--------------------+
| TransactionID | AccountID | TransactionType | Amount  | TransactionDate    |
+---------------+-----------+-----------------+---------+--------------------+
|             3 |       102 | withdrawal      | 8000.00 | 2025-01-31 18:58:24 |
+---------------+-----------+-----------------+---------+--------------------+
1 row in set (0.00 sec)
```

7. Get customers with account balances above 5000:

```
mysql> select customers.*, accounts.Balance from customers join accounts on cus
tomers.CustomerID = accounts.CustomerID where accounts.Balance > 5000;
+------------+-------+----------+------------+----------+
| CustomerID | Name  | Address  | Phone      | Balance  |
+------------+-------+----------+------------+----------+
|          1 | Ram   | KTM      | 9811111111 |  6000.00 |
|          3 | Ronil | Lalitpur | 9833333333 | 10000.00 |
|          3 | Ronil | Lalitpur | 9833333333 | 20000.00 |
+------------+-------+----------+------------+----------+
3 rows in set (0.00 sec)
```

8. Count the number of accounts per account type:

```
mysql> select accounts.AccountType, count(AccountType)
    -> from accounts
    -> group by AccountType;
+-------------+--------------------+
| AccountType | count(AccountType) |
+-------------+--------------------+
| saving      |                  2 |
| current     |                  2 |
+-------------+--------------------+
2 rows in set (0.00 sec)
```

9. Withdraw 1000 from Ganesh account (AccountID 104):

```
mysql> insert into transactions (AccountID, TransactionType, Amount) values (10
4, "withdrawal", 1000);
Query OK, 1 row affected (0.01 sec)
mysql> select customers.Name, transactions.* from ((customers join accounts on
customers.CustomerID = accounts.CustomerID) join transactions on accounts.Accou
ntID = transactions.AccountID) where accounts.AccountID = 104;
+--------+---------------+-----------+----------------+---------+--------------------+
| Name   | TransactionID | AccountID | TransactionType | Amount  | TransactionDate    |
+--------+---------------+-----------+----------------+---------+--------------------+
| Ganesh |             4 |       104 | withdrawal     | 1000.00 | 2025-01-31 19:20:48 |
+--------+---------------+-----------+----------------+---------+--------------------+
1 row in set (0.00 sec)
```

Lab 2 Output:

1. Find Customers with Highest Balance:

```
mysql> select c.*, a.balance from customers c join accounts a on c.CustomerI
D = a.CustomerID where a.balance = (Select Max(balance) from accounts);
+------------+---------+---------+------------+----------+
| CustomerID | Name    | Address | Phone      | balance  |
+------------+---------+---------+------------+----------+
|          5 | Krishna | Patan   | 9877777777 | 30000.00 |
+------------+---------+---------+------------+----------+
1 row in set (0.00 sec)
```

2. Find the customers details with lowest balance:

```
mysql> select c.*, a.balance from customers c join accounts a on c.CustomerI
D = a.CustomerID where a.balance = (Select min(balance) from accounts);
+------------+------+----------+------------+---------+
| CustomerID | Name | Address  | Phone      | balance |
+------------+------+----------+------------+---------+
|          2 | Hari | Lalitpur | 9822222222 | 1000.00 |
+------------+------+----------+------------+---------+
1 row in set (0.00 sec)
```

3. List Transactions Greater Than the Average Transaction Amount:

```
mysql> select * from transactions where transactions.amount > (select avg(amoun
t) from transactions);
+---------------+-----------+-----------------+---------+---------------------+
| TransactionID | AccountID | TransactionType | Amount  | TransactionDate     |
+---------------+-----------+-----------------+---------+---------------------+
|             2 |         3 | withdrawal      | 6000.00 | 2025-01-31 18:49:19 |
|             3 |       102 | withdrawal      | 8000.00 | 2025-01-31 18:58:24 |
+---------------+-----------+-----------------+---------+---------------------+
2 rows in set (0.01 sec)
```

4. Find Customers with More Than One Account:

```
mysql> select c.*, count(a.accountid) as AccountCount from accounts a join cust
omers c on a.customerid = c.customerid group by a.customerid having count(a.acc
ountid) > 1;
+------------+-------+----------+------------+--------------+
| CustomerID | Name  | Address  | Phone      | AccountCount |
+------------+-------+----------+------------+--------------+
|          3 | Ronil | Lalitpur | 9833333333 |            2 |
+------------+-------+----------+------------+--------------+
1 row in set (0.00 sec)
```

5. Get Accounts with Transactions in the Last 7 Days:

```
mysql> select c.name, a.*, t.transactiondate from accounts a join transactions
t on a.accountid = t.accountid join customers c on c.customerid = a.customerid
where t.transactiondate >= curdate() - interval 7 day;
+--------+-----------+------------+-------------+----------+---------------------+
| name   | AccountID | CustomerID | AccountType | Balance  | transactiondate     |
+--------+-----------+------------+-------------+----------+---------------------+
| Ram    |         1 |          1 | saving      |  6000.00 | 2025-01-31 18:48:56 |
| Ronil  |         3 |          3 | current     | 10000.00 | 2025-01-31 18:49:19 |
| Ronil  |       102 |          3 | saving      | 20000.00 | 2025-01-31 18:58:24 |
| Ganesh |       104 |          4 | current     |  4000.00 | 2025-01-31 19:20:48 |
+--------+-----------+------------+-------------+----------+---------------------+
4 rows in set (0.01 sec)
```

6. Get Customers Whose Balance Is Above the Average of Customers with More Than One Account:
    a. Average balance of customers with more than one account: (subquery)

```
mysql> select avg(balance) from accounts where customerid in (select custome
rid from accounts group by customerid having count(accountid) > 1);
+--------------+
| avg(balance) |
+--------------+
| 15000.000000 |
+--------------+
1 row in set (0.00 sec)
```

```
mysql> select c.*, a.balance, a.accounttype from customers c join accounts a
 on c.customerid = a.customerid where a.balance > (select avg(balance) from
accounts where customerid in (select customerid from accounts group by custo
merid having count(accountid) > 1));
+------------+---------+----------+------------+----------+-------------+
| CustomerID | Name    | Address  | Phone      | balance  | accounttype |
+------------+---------+----------+------------+----------+-------------+
|          3 | Ronil   | Lalitpur | 9833333333 | 20000.00 | saving      |
|          5 | Krishna | Patan    | 9877777777 | 30000.00 | saving      |
+------------+---------+----------+------------+----------+-------------+
2 rows in set (0.00 sec)
```

7. Find Accounts with More Transactions Than the Account with the Least Transactions

```
mysql> SELECT c.Name, a.*, COUNT(t.TransactionID) AS TransactionCount
    -> from accounts a
    -> join customers c on c.customerid = a.customerid
    -> join transactions t on t.accountid = a.accountid
    -> group by a.accountid, c.name
    -> having count(t.transactionid) > (select min(transactioncount) from
    -> (select count(transactionid) as transactioncount
    -> from transactions
    -> group by accountid
    -> )as subquery);
+-------+-----------+------------+-------------+----------+------------------+
| Name  | AccountID | CustomerID | AccountType | Balance  | TransactionCount |
+-------+-----------+------------+-------------+----------+------------------+
| Ronil |       102 |          3 | saving      | 20000.00 |                2 |
+-------+-----------+------------+-------------+----------+------------------+
1 row in set (0.00 sec)
```

Lab 3 Output:

1. Creating composite Type “address”:

```
create type address as (street varchar(100), city varchar(100), postal_code varchar(10));
```

Data Output | Messages | Notifications

```
CREATE TYPE

Query returned successfully in 31 msec.
```

2. Creating table student using the “address” user:

```
create table student (
    sid serial primary key,
    name varchar(20),
    home_address address
);
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 27 msec.
```

3. Inserting data into student

```
Insert into student (name,home_address) values
('Ram', ('Putalisadak', 'Kathmandu', '44600')),
('Shyam', ('New Road', 'Kathmandu', '44600')),
('Hari', ('Anam Marg', 'Pokhara', '33700')),
('John', ('Aarati Path', 'Biratnagar', '56613')),
('Krishna', ('Prabha Path', 'Butwal', '32907'));
select * from student;
```

Data Output | Messages | Notifications

	sid [PK] integer	name character varying (20)	home_address address
1	1	Ram	(Putalisadak,Kathmandu,4460...
2	2	Shyam	("New Road",Kathmandu,4460...
3	3	Hari	("Anam Marg",Pokhara,33700)
4	4	John	("Aarati Path",Biratnagar,56613)
5	5	Krishna	("Prabha Path",Butwal,32907)

4. Displaying name, street, city and postal_code:

```
Select name,
    (home_address).street As street,
    (home_address).city as city,
    (home_address).postal_code as postal_code
from student;
```

Data Output Messages Notifications

	name character varying (20)	street character varying (100)	city character varying (100)	postal_code character varying (10)
1	Ram	Putalisadak	Kathmandu	44600
2	Shyam	New Road	Kathmandu	44600
3	Hari	Anam Marg	Pokhara	33700
4	John	Aarati Path	Biratnagar	56613
5	Krishna	Prabha Path	Butwal	32907

5. Displaying name and city:

```
select name, (home_address).city as City from student;
```

Data Output Messages Notifications

	name character varying (20)	city character varying (100)
1	Ram	Kathmandu
2	Shyam	Kathmandu
3	Hari	Pokhara
4	John	Biratnagar
5	Krishna	Butwal

Lab 4 Output:

1. Creating a parent table people having the attributes id, name, dob:

```sql
CREATE TABLE people (
id SERIAL PRIMARY KEY,
name VARCHAR(100),
dob DATE);
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 46 msec.
```

2. Creating Child Tables student & employee for parent table person:

```sql
CREATE TABLE employee (
department VARCHAR(50),
salary DECIMAL(10,2)
) INHERITS (people);

CREATE TABLE student (
faculty VARCHAR(100)
) INHERITS (people);
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 28 msec.
```

```sql
ALTER TABLE employee ALTER COLUMN id SET DEFAULT nextval('people_id_seq');
```

Data Output | Messages | Notifications

```
ALTER TABLE

Query returned successfully in 36 msec.
```

```sql
ALTER TABLE student ALTER COLUMN id SET DEFAULT nextval('people_id_seq');
```

Data Output | Messages | Notifications

```
ALTER TABLE

Query returned successfully in 32 msec.
```

3. Inserting Data into Child Tables:

```sql
INSERT INTO employee (name, dob, department, salary) VALUES ('Ram Kumar', '2002-03-15', 'Sales', 80000);
INSERT INTO student (name, dob, faculty) VALUES ('Ronil Maharjan', '2002-01-03','Computer Science and Information
Technology');
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 51 msec.
```

4. Query the Parent Table:
    a. Here data inserted into child table are shown since employee and student inherit person,

```
SELECT * FROM people;
```

Data Output Messages Notifications

	id [PK] integer	name character varying (100)	dob date
1	2	Ram Kumar	2002-03-15
2	3	Ronil Maharjan	2002-01-03

5. Query Individual table;

```
select * from student;
```

Data Output Messages Notifications

	id integer	name character varying (100)	dob date	faculty character varying (100)
1	3	Ronil Maharjan	2002-01-03	Computer Science and Information

6. Prevent Inserts into Parent Table:
    a. PostgreSQL allows inserting directly into peple by default, changing the behaviour using:

```
CREATE OR REPLACE FUNCTION prevent_parent_insert()
RETURNS TRIGGER AS $$
BEGIN
    RAISE EXCEPTION 'Inserts are not allowed into the parent table';
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_insert
BEFORE INSERT ON people
FOR EACH ROW EXECUTE FUNCTION prevent_parent_insert();

```

```
Insert into people (name, dob) values ('Raj', '2002-4-23');
```

Data Output Messages Notifications

```
ERROR:  Inserts are not allowed into the parent table
CONTEXT:  PL/pgSQL function prevent_parent_insert() line 3 at RAISE

SQL state: P0001
```

Lab 5 Output:

1. Defining a base table Person:

```sql
Create table Person(
    id Serial Primary Key,
    name text not null,
    birth_date Date
);
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 67 msec.
```

2. Creating two derived tables Employee and Customer using inheritance:

```sql
CREATE TABLE Employee (
department Varchar(50) NOT NULL,
salary DECIMAL(10,2)
)INHERITS (Person);

CREATE TABLE Customer (
loyalty_points INT DEFAULT 0
) INHERITS (Person);
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 42 msec.
```

3. Creating a Product table to store semi-structured data using JSONB:

```sql
CREATE TABLE Product (
    id SERIAL PRIMARY KEY,
    name TEXT NOT NULL,
    attributes JSONB
);
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 63 msec.
```

4. Inserting data to all the above tables:

```sql
INSERT INTO Employee (name, birth_date, department, salary) VALUES
('Ronil Maharjan', '1990-05-15', 'HR', 50000.00),
('Ram Kumar', '1988-08-22', 'IT', 70000.50),
('Krishna Maharjan', '1995-12-10', 'Finance', 65000.75);

INSERT INTO Customer (name, birth_date, loyalty_points) VALUES
('John Brown', '1985-03-25', 120),
('Emma Wilson', '1992-07-14', 300),
('Frank Miller', '1980-11-05', 150);

INSERT INTO Product (name, attributes) VALUES
('Laptop', '{"brand": "Dell", "RAM": "16GB", "Storage": "512GB SSD"}'),
('Smartphone', '{"brand": "Samsung", "RAM": "8GB", "Storage": "128GB"}'),
('Headphones', '{"brand": "Sony", "type": "Wireless", "BatteryLife": "30 hours"}');
```

Data Output | Messages | Notifications

```
INSERT 0 3

Query returned successfully in 45 msec.
```

5. Retrieving all Persons:

```sql
select * from person;
```

Data Output | Messages | Notifications

	id [PK] integer	name text	birth_date date
1	8	John Brown	1985-03-25
2	9	Emma Wilson	1992-07-14
3	10	Frank Miller	1980-11-05
4	5	Ronil Maharjan	1990-05-15
5	6	Ram Kumar	1988-08-22
6	7	Krishna Maharjan	1995-12-10

6. Retrieving Employees details:

```sql
select * from employee;
```

Data Output | Messages | Notifications

	id integer	name text	birth_date date	department character varying (50)	salary numeric (10,2)
1	5	Ronil Maharjan	1990-05-15	HR	50000.00
2	6	Ram Kumar	1988-08-22	IT	70000.50
3	7	Krishna Maharjan	1995-12-10	Finance	65000.75

7. Retrieving Customers with more than 99 loyalty points:

```
64 select * from customer where loyalty_points > 99;
```

Data Output Messages Notifications

	id integer	name text	birth_date date	loyalty_points integer
1	8	John Brown	1985-03-25	120
2	9	Emma Wils...	1992-07-14	300
3	10	Frank Miller	1980-11-05	150

8. Retrieving product attributes in JSON format:

```
66 SELECT name, attributes->>'brand' AS brand FROM Product;
```

Data Output Messages Notifications

	name text	brand text
1	Laptop	Dell
2	Smartphone	Samsung
3	Headphones	Sony

Lab 6 Output:

1. Creating a function calcfactorial for calculating factorial of any given integer number:

```
CREATE OR REPLACE FUNCTION calcfactorial (n INT) RETURNS BIGINT AS $$
DECLARE
result BIGINT := 1;
i INT;
BEGIN
IF n < 0 THEN
RAISE EXCEPTION 'Factorial is not defined for negative numbers';
END IF;
FOR i IN 1..n LOOP
result := result * i;
END LOOP;
RETURN result;
END;
$$ LANGUAGE plpgsql;
```

Data Output | Messages | Notifications

```
CREATE FUNCTION

Query returned successfully in 30 msec.
```

2. Testing the function:

```
SELECT calcfactorial(4);
```

Data Output | Messages | Notifications

	calcfactorial bigint
1	24

3. Creating tables employees & audit_log:

```
create table employees (
emp_id SERIAL PRIMARY KEY,
emp_name VARCHAR(100) NOT NULL,
salary NUMERIC(10,2) CHECK (salary > 0),
department VARCHAR(50) NOT NULL
);
create table audit_log (
log_id SERIAL PRIMARY KEY,
emp_id INT,
action_type VARCHAR(10), -- INSERT, UPDATE, DELETE
old_salary NUMERIC(10,2),
new_salary NUMERIC(10,2),
changed_by VARCHAR(50),
changed_on TIMESTAMP DEFAULT NOW()
);
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 37 msec.
```

4. Creating a function to get the salary of employee by employee_id.

```
CREATE OR REPLACE FUNCTION get_employee_salary(emp_id_param INT)
RETURNS NUMERIC(10,2) AS $$
DECLARE
    emp_salary NUMERIC(10,2);
BEGIN
    SELECT salary INTO emp_salary FROM employees WHERE emp_id = emp_id_param;

    IF emp_salary IS NULL THEN
        RAISE EXCEPTION 'Employee ID % not found', emp_id_param;
    END IF;

    RETURN emp_salary;
END;
$$ LANGUAGE plpgsql;
```

Data Output | Messages | Notifications

```
CREATE FUNCTION

Query returned successfully in 50 msec.
```

a. Testing the function:

```
select get_employee_salary(2);
```

Data Output | Messages | Notifications

	get_employee_salary numeric
1	75000.50

5. Creating a trigger that prevents insertion of employees with salary less than 25000.

```
CREATE OR REPLACE FUNCTION check_salary_before_insert()
RETURNS TRIGGER AS $$
BEGIN
IF NEW.salary < 25000 THEN
RAISE EXCEPTION 'Salary must be at least 25000 ';
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_before_insert_salary
BEFORE INSERT ON employees
FOR EACH ROW
EXECUTE FUNCTION check_salary_before_insert();
```

Data Output | Messages | Notifications

```
CREATE TRIGGER

Query returned successfully in 51 msec.
```

a. Testing the above trigger:

```
INSERT INTO employees (emp_name, salary, department) VALUES ('Ronil', 20000, 'IT');
```

Data Output | Messages | Notifications

```
ERROR:  Salary must be at least 25000
CONTEXT:  PL/pgSQL function check_salary_before_insert() line 4 at RAISE

SQL state: P0001
```

6. Creating a trigger that automatically records changes to employee salaries.

```
CREATE OR REPLACE FUNCTION audit_salary_changes()
RETURNS TRIGGER AS $$
BEGIN
IF NEW.salary <> OLD.salary THEN
INSERT INTO audit_log (emp_id, action_type, old_salary, new_salary, changed_by, changed_on)
VALUES (OLD.emp_id, 'UPDATE', OLD.salary, NEW.salary, CURRENT_USER, NOW());
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_after_update_salary
AFTER UPDATE ON employees
FOR EACH ROW
EXECUTE FUNCTION audit_salary_changes();
```

Data Output | Messages | Notifications

```
CREATE TRIGGER

Query returned successfully in 39 msec.
```

a. Testing the above trigger:

```
INSERT INTO employees (emp_name, salary, department) VALUES ('Priyansh', 50000, 'Digital')
```

Data Output | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 43 msec.
```

```
UPDATE employees SET salary = 60000 WHERE emp_name = 'Priyansh';
```

Data Output | Messages | Notifications

```
UPDATE 1

Query returned successfully in 43 msec.
```

```
SELECT * FROM audit_log;
```

Data Output | Messages | Notifications

	log_id [PK] integer	emp_id integer	action_type character varying (10)	old_salary numeric (10,2)	new_salary numeric (10,2)	changed_by character varying (50)	changed_on timestamp without time zone
1	1	8	UPDATE	50000.00	60000.00	postgres	2025-02-22 11:15:12.959862

Lab 7 Output:

1. Creating a customers table & loading the customers data into it through customers.csv file:

```sql
CREATE TABLE customers (
    Id serial PRIMARY KEY,
    CustomerId BIGINT,
    Surname TEXT,
    CreditScore INT,
    Geography TEXT,
    Gender TEXT,
    Age INT,
    Tenure INT,
    Balance NUMERIC(15,2),
    NumOfProducts INT,
    HasCrCard BOOLEAN,
    IsActiveMember BOOLEAN,
    EstimatedSalary NUMERIC(15,2),
    Exited BOOLEAN
);
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 54 msec.
```

```sql
copy public.customers FROM 'C:\Homeworks\8th Sem\Advanced DB\customers (1).csv' DELIMITER ',' CSV HEADER;
```

Data Output | Messages | Notifications

```
COPY 10000

Query returned successfully in 54 msec.
```

```sql
select * from customers limit 5;
```

Data Output | Messages | Notifications

	id [PK] integer	customerid bigint	surname text	creditscore integer	geography text	gender text	age integer	tenure integer	balance numeric (15,2)	nu in
1	1	15634602	Hargrave	619	France	Female	42	2	0.00	
2	2	15647311	Hill	608	Spain	Female	41	1	83807.86	
3	3	15619304	Onio	502	France	Female	42	8	159660.80	
4	4	15701354	Boni	699	France	Female	39	1	0.00	
5	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	

2. A simple query to fetch the customers data who are aged 24:

```sql
select * from customers where age = 24;
```

Data Output | Messages | Notifications

	id [PK] integer	customerid bigint	surname text	creditscore integer	geography text	gender text	age integer	ten inte
1	12	15737173	Andrews	497	Spain	Male	24	
2	18	15788218	Henderson	549	Spain	Female	24	
3	20	15568982	Hao	726	France	Female	24	
4	130	15591607	Fernie	770	France	Male	24	
5	167	15724623	Taubman	704	Germany	Female	24	
6	170	15611325	Wood	682	Germany	Male	24	

3. Running EXPLAIN ANALYZE:

```
41 EXPLAIN ANALYZE SELECT * FROM public.customers WHERE age = 24;
```

Data Output Messages Notifications

	QUERY PLAN text
1	Seq Scan on customers (cost=0.00..257.00 rows=132 width=64) (actual time=0.045..2.497 rows=132 loops...
2	Filter: (age = 24)
3	Rows Removed by Filter: 9868
4	Planning Time: 0.126 ms
5	Execution Time: 2.830 ms

The db performed Sequential Scan and the execution time is 2.830ms.

4. Creating index on the age column:

```
43 CREATE INDEX idx_customers_age ON public.customers(age);
```

Data Output Messages Notifications

```
CREATE INDEX

Query returned successfully in 136 msec.
```

```
41 EXPLAIN ANALYZE SELECT * FROM public.customers WHERE age = 24;
42
```

Data Output Messages Notifications

	QUERY PLAN text
1	Bitmap Heap Scan on customers (cost=5.31..143.40 rows=132 width=64) (actual time=0.043..0.122 rows=132 loops=1)
2	Recheck Cond: (age = 24)
3	Heap Blocks: exact=87
4	-> Bitmap Index Scan on idx_customers_age (cost=0.00..5.28 rows=132 width=0) (actual time=0.024..0.024 rows=132 loop...
5	Index Cond: (age = 24)
6	Planning Time: 0.115 ms
7	Execution Time: 0.158 ms

The db now perfoms bitmap heap scan on customers table and bitmap index scan on the newly created idx_cutomers_age and the execution time decreased to 0.158ms.

Materialized Views:

1. Creating a table transactions:

```
Create table transactions
( id SERIAL PRIMARY KEY,
customer_name TEXT,
amount DECIMAL(10,2),
transaction_date DATE default Current_date );
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 59 msec.
```

2. Inserting data into transactions table:

```
insert into transactions (customer_name, amount) values
('Ronil', 30000),
('Ram', 50000);
```

Data Output | Messages | Notifications

```
INSERT 0 2

Query returned successfully in 109 msec.
```

3. Creating materialized view that stores total transaction amounts per customer.

```
CREATE MATERIALIZED VIEW total_transaction_per_customer AS
SELECT
    customer_name,
    SUM(amount) AS total_amount
FROM transactions
GROUP BY customer_name;
```

Data Output | Messages | Notifications

```
SELECT 2

Query returned successfully in 90 msec.
```

```
SELECT * FROM total_transaction_per_customer;
```

Data Output | Messages | Notifications

	customer_name text	total_amount numeric
1	Ronil	30000.00
2	Ram	50000.00

4. Adding new transaction refreshing the materialized view:

```
insert into transactions (customer_name, amount) values
('Ronil', 20000);
REFRESH MATERIALIZED VIEW total_transaction_per_customer;
SELECT * FROM total_transaction_per_customer;
```

Data Output Messages Notifications

	customer_name text	total_amount numeric
1	Ronil	50000.00
2	Ram	50000.00

Lab 8 Output:

1. Creating three fragments partitioning it horizontally based on region Germany, Spain & France:
    - First creating a partition table where table is Partition By List (Geography):

```sql
CREATE TABLE customers_partitioned (
    CustomerId BIGINT NOT NULL,
    Surname TEXT,
    CreditScore INTEGER,
    Geography TEXT NOT NULL,
    Gender TEXT,
    Age INTEGER,
    Tenure INTEGER,
    Balance NUMERIC(15,2),
    NumOfProducts INTEGER,
    HasCrCard BOOLEAN,
    IsActiveMember BOOLEAN,
    EstimatedSalary NUMERIC(15,2),
    Exited BOOLEAN,
    PRIMARY KEY (CustomerId, Geography)
) PARTITION BY LIST (Geography);
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 91 msec.
```

2. Creating the three fragments

```sql
CREATE TABLE customers_germany PARTITION OF customers_partitioned
    FOR VALUES IN ('Germany');

CREATE TABLE customers_spain PARTITION OF customers_partitioned
    FOR VALUES IN ('Spain');

CREATE TABLE customers_france PARTITION OF customers_partitioned
    FOR VALUES IN ('France');
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 78 msec.
```

3. Running a query on one of the fragment (customers_spain) and confirmed only relevant data is present:

```
69 select * from customers_spain;
```

Data Output Messages Notifications

	customerid [PK] bigint	surname text	creditscore integer	geography [PK] text	gender text	age integer	tenure integer	balance numeric (15,2)	numofproducts integer	hascrcard boolean
1	15647311	Hill	608	Spain	Female	41	1	83807.86	1	false
2	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	true
3	15574012	Chu	645	Spain	Male	44	8	113755.78	2	true
4	15737173	Andrews	497	Spain	Male	24	3	0.00	2	true

```
74 SELECT tableoid::regclass AS partition, * FROM customers_partitioned;
```

Data Output Messages Notifications

	partition regclass	customerid [PK] bigint	surname text	creditscore integer	geography [PK] text	gender text	age integer	tenure integer
5010	customers_france	15719294	Wood	800	France	Female	29	2
5011	customers_france	15606229	Obijiaku	771	France	Male	39	5
5012	customers_france	15569892	Johnstone	516	France	Male	35	10
5013	customers_france	15584532	Liu	709	France	Female	36	7
5014	customers_france	15628319	Walker	792	France	Female	28	4
5015	customers_germ...	15656148	Obinna	376	Germany	Female	29	4
5016	customers_germ...	15643966	Goforth	616	Germany	Male	45	3
5017	customers_germ...	15737452	Romeo	653	Germany	Male	58	1
5018	customers_germ...	15736816	Young	756	Germany	Male	36	2

```sql
CREATE TABLE customers_personal (
    CustomerId BIGINT PRIMARY KEY,
    Surname TEXT,
    Geography TEXT,
    Gender TEXT,
    Age INTEGER
);

CREATE TABLE customers_financial (
    CustomerId BIGINT PRIMARY KEY REFERENCES customers_personal(CustomerId),
    CreditScore INTEGER,
    Balance NUMERIC(15,2),
    EstimatedSalary NUMERIC(15,2)
);

CREATE TABLE customers_activity (
    CustomerId BIGINT PRIMARY KEY REFERENCES customers_personal(CustomerId),
    Tenure INTEGER,
    NumOfProducts INTEGER,
    HasCrCard BOOLEAN,
    IsActiveMember BOOLEAN,
    Exited BOOLEAN
);
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 104 msec.
```

```sql
-- Insert into Personal Details Table
INSERT INTO customers_personal (CustomerId, Surname, Geography, Gender, Age)
SELECT CustomerId, Surname, Geography, Gender, Age FROM customers;

-- Insert into Financial Details Table
INSERT INTO customers_financial (CustomerId, CreditScore, Balance, EstimatedSalary)
SELECT CustomerId, CreditScore, Balance, EstimatedSalary FROM customers;

-- Insert into Activity Details Table
INSERT INTO customers_activity (CustomerId, Tenure, NumOfProducts, HasCrCard, IsActiveMember, Exited)
SELECT CustomerId, Tenure, NumOfProducts, HasCrCard, IsActiveMember, Exited FROM customers;
```

Data Output | Messages | Notifications

```
INSERT 0 10000

Query returned successfully in 509 msec.
```

```sql
select * from customers_personal;
```

Data Output | Messages | Notifications

	customerid [PK] bigint	surname text	geography text	gender text	age integer
1	15634602	Hargrave	France	Female	42
2	15647311	Hill	Spain	Female	41
3	15619304	Onio	France	Female	42

4. Performing the vertical fragmentation considering appropriate attributes & key in the customers data of lab 7.

```sql
CREATE TABLE CustomerIdentity AS
SELECT CustomerId, Surname, Geography, Gender, Age
FROM Customers;

CREATE TABLE CustomerAccount AS
SELECT CustomerId, CreditScore, Tenure, Balance, NumOfProducts, HasCrCard, IsActiveMember
FROM Customers;

CREATE TABLE CustomerStatus AS
SELECT CustomerId, EstimatedSalary, Exited
FROM Customers;
```

Data Output | Messages | Notifications

```
SELECT 10000

Query returned successfully in 95 msec.
```

5. Implementing a Multi-Tier Architecture using PostgreSQL & Python(Flask):

Database Tier:

Query | Query History

```sql
SELECT * FROM public.customers
ORDER BY id ASC LIMIT 100
```

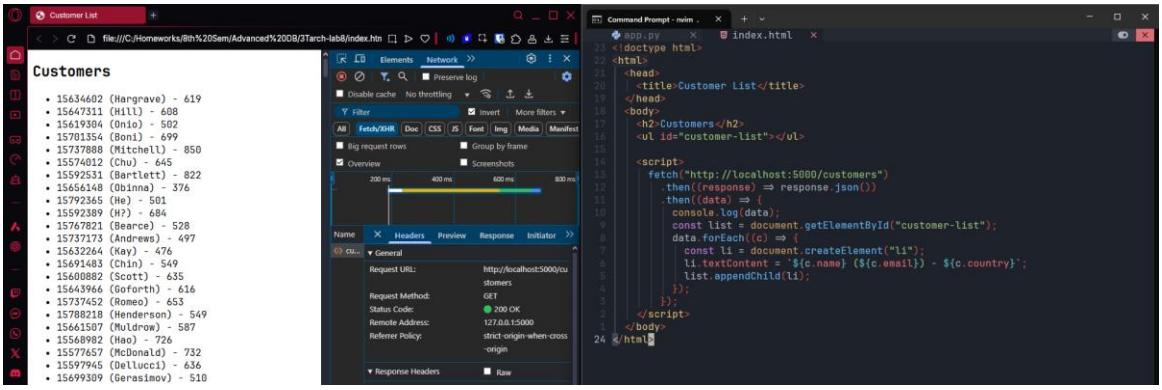
Data Output | Messages | Notifications

	id [PK] integer	customerid bigint	surname text	creditscore integer	geography text	gender text	age integer	tenure integer	balance numeric (15,2)	numofproducts integer	hascrcard boolean	isactivemember boolean	estimatedsalary numeric (15,2)	exited boolean
1	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	true	true	101348.88	true
2	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	false	true	112542.58	false
3	3	15619304	Onio	502	France	Female	42	8	159660.80	3	true	false	113931.57	true
4	4	15701354	Boni	699	France	Female	39	1	0.00	2	false	false	93826.63	false
5	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	true	true	79084.10	false
6	6	15574012	Chu	645	Spain	Male	44	8	113755.78	2	true	false	149756.71	true

Application Tier:

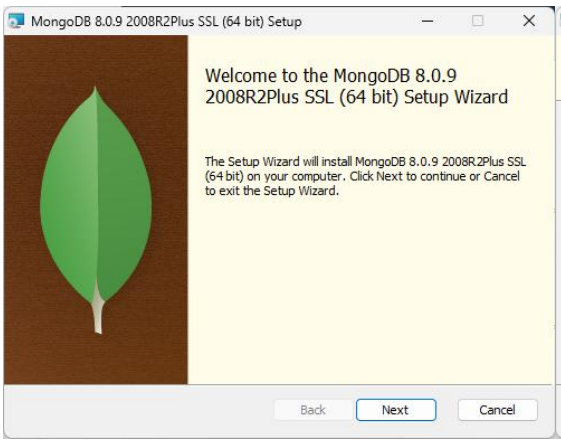
```python
from flask import Flask, jsonify, request
import psycopg2
from flask_cors import CORS


app = Flask(__name__)
CORS(app)  # 👈 Allow cross-origin requests

# Database connection
conn = psycopg2.connect(
    dbname="lab7",
    user="postgres",
    password="root",
    host="localhost",
    port="5432"
)
cur = conn.cursor()

@app.route('/customers', methods=['GET'])
def get_customers():
    cur.execute("SELECT * FROM customers;")
    rows = cur.fetchall()
    customers = [{"id": r[0], "name": r[1], "email": r[2], "country": r[3]} for r in rows]
    return jsonify(customers)

@app.route('/customers', methods=['POST'])
def add_customer():
    data = request.get_json()
    if not data:
        return jsonify({"error": "Invalid or missing JSON"}), 400
    cur.execute(
        "INSERT INTO customers (name, email, country) VALUES (%s, %s, %s)",
        (data['name'], data['email'], data['country'])
    )
    conn.commit()
    return jsonify({"message": "Customer added successfully"}), 201

if __name__ == '__main__':
    app.run(debug=True)
```

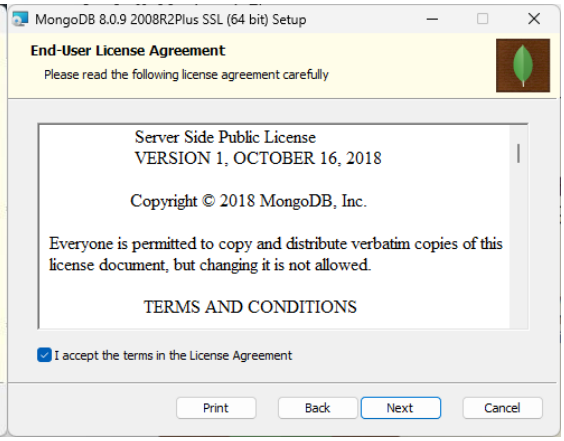
Simple Client Tier (Browser):

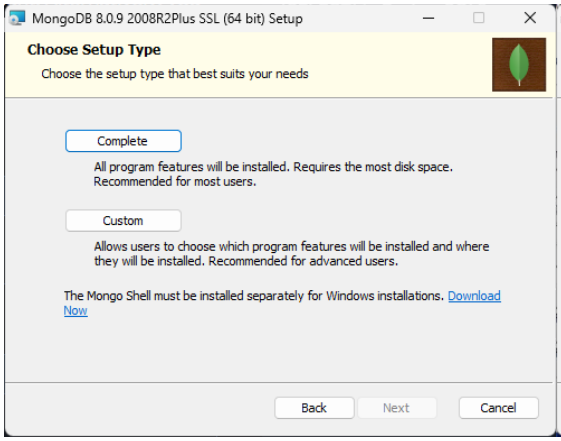


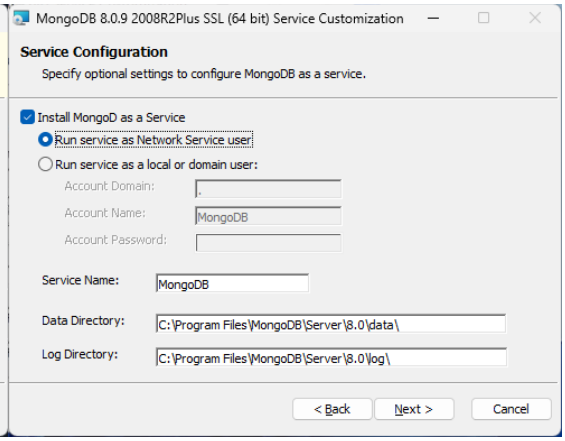
Lab 9 Output:

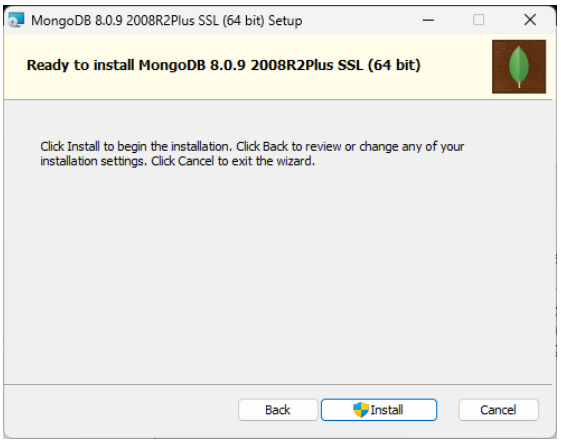
1. Installation of MongoDB:



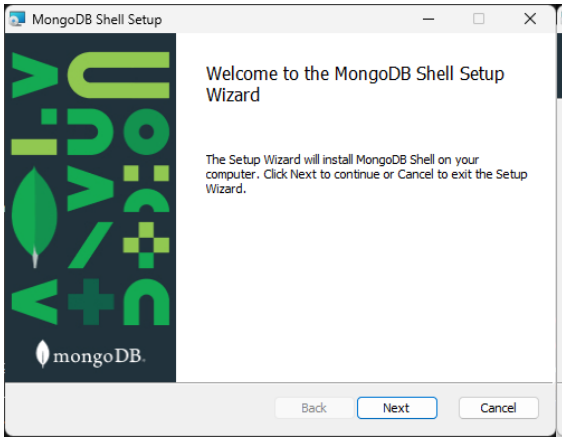


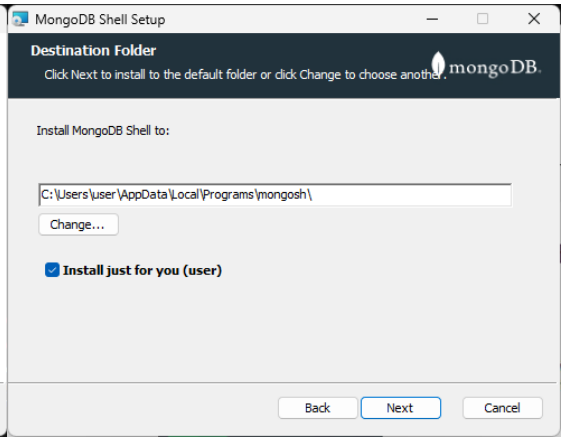


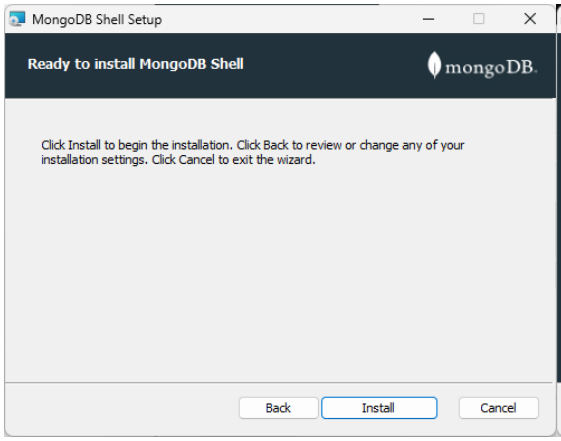


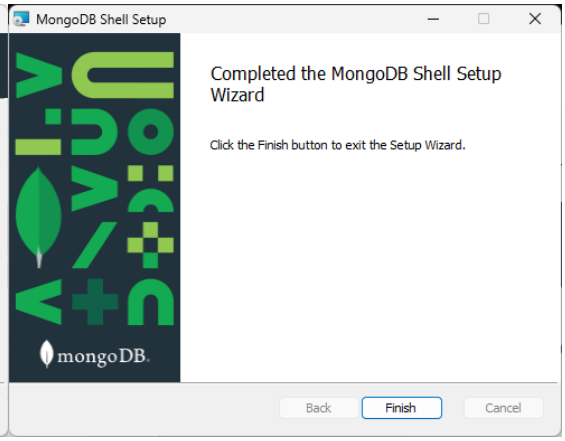


2. Installing MongoDB Shell:









3. Connect to MongoDB using MongoDB Shell

```
mongosh mongodb://127.0.0.
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

C:\Users\user>mongosh
Current Mongosh Log ID: 6824542520344be4b56c4bcf
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverS
Using MongoDB:          8.0.1
Using Mongosh:          2.5.1

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2025-05-14T12:11:05.162+05:45: Access control is not enabled for the database
------

test>
```

4. Creating Database Library

```
test> use Library
switched to db Library
Library>
```

5. Creating Collection books:

```
test> use Library
switched to db Library
Library> db.createCollection("books")
{ ok: 1 }
Library>
```

6. Inserting Documents into the Collection title, author, sno, year.

```
Library> db.books.insertOne({
... title: "Book Title",
... author:"Ronil",
... sno: 1,
... year: 2024
... })
{
  acknowledged: true,
  insertedId: ObjectId('6824585420344be4b56c4bd1')
}
Library>
```

```
Library> db.books.insertMany([
...   { title: "Book A", author: "Author A", sno: 2, year: 2021 },
...   { title: "Book B", author: "Author B", sno: 3, year: 2020 }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6824588020344be4b56c4bd2'),
    '1': ObjectId('6824588020344be4b56c4bd3')
  }
}
Library>
```

7. Query Documents:

- Finding all documents:

```
Library> db.books.find()
[
  {
    _id: ObjectId('6824585420344be4b56c4bd1'),
    title: 'Book Title',
    author: 'Ronil',
    sno: 1,
    year: 2024
  },
  {
    _id: ObjectId('6824588020344be4b56c4bd2'),
    title: 'Book A',
    author: 'Author A',
    sno: 2,
    year: 2021
  },
  {
    _id: ObjectId('6824588020344be4b56c4bd3'),
    title: 'Book B',
    author: 'Author B',
    sno: 3,
    year: 2020
  }
]
```

- Finding specific document by author:

```
Library> db.books.find({ author: "Ronil" })
[
  {
    _id: ObjectId('6824585420344be4b56c4bd1'),
    title: 'Book Title',
    author: 'Ronil',
    sno: 1,
    year: 2024
  }
]
```

- Finding document that satisfies specific conditions like year greater than 2021:

```
Library> db.books.find({ year: { $gt: 2021 } })
[
  {
    _id: ObjectId('6824585420344be4b56c4bd1'),
    title: 'Book Title',
    author: 'Ronil',
    sno: 1,
    year: 2024
  }
]
```

8. Updating and Deleting:

```
Library> db.books.updateOne({ sno: 1 }, { $set: { title: "Updated Title" } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Library> db.books.find({sno: 1})
[
  {
    _id: ObjectId('6824585420344be4b56c4bd1'),
    title: 'Updated Title',
    author: 'Ronil',
    sno: 1,
    year: 2024
  }
]
```

```
Library> db.books.find({sno: 1})
[
  {
    _id: ObjectId('6824585420344be4b56c4bd1'),
    title: 'Updated Title',
    author: 'Ronil',
    sno: 1,
    year: 2024
  }
]
Library> db.books.deleteOne({ sno: 1 })
{ acknowledged: true, deletedCount: 1 }
Library> db.books.find({sno: 1})

Library>
```

9. Creating an Index on the title field:

```
Library> db.books.createIndex({ title: 1 })
title_1
```

10. Checking the indexes on the collection:

```
Library> db.books.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { title: 1 }, name: 'title_1' }
]
```

11. Dropping the Index created in Q.no. 9.:

```
Library> db.books.dropIndex("title_1")
{ nIndexesWas: 2, ok: 1 }
Library> db.books.getIndexes()
[ { v: 2, key: { _id: 1 }, name: '_id_' } ]
Library>
```

12. Counting the total number of books:

```
Library> db.books.countDocuments()
2
```

13. Data Modelling using one to one & one to many relationships:

- One to One Relationship:

```
Library> db.users.insertOne({ _id: 1, name: "User 1", profileId: 1 });
{ acknowledged: true, insertedId: 1 }
Library> db.profiles.insertOne({ _id: 1, address: "123 St." });
{ acknowledged: true, insertedId: 1 }
Library> db.users.aggregate([
...   {
...     $lookup: {
...       from: "profiles",
...       localField: "profileId",
...       foreignField: "_id",
...       as: "profile"
...     }
...   },
...   { $unwind: "$profile" }
... ]);
[
  {
    _id: 1,
    name: 'User 1',
    profileId: 1,
    profile: { _id: 1, address: '123 St.' }
  }
]
Library>
```

- One to Many Relationship:

```
Library> db.books.find()
[
  {
    _id: ObjectId('6824588020344be4b56c4bd2'),
    title: 'Book A',
    author: 'Author A',
    sno: 2,
    year: 2021
  },
  {
    _id: ObjectId('6824588020344be4b56c4bd3'),
    title: 'Book B',
    author: 'Author B',
    sno: 3,
    year: 2020
  }
]
Library> db.users.updateOne(
... {_id: 1},
... { $set: {books:[ObjectId('6824588020344be4b56c4bd2'),ObjectId('6824588020344be4b56c4bd3')]}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

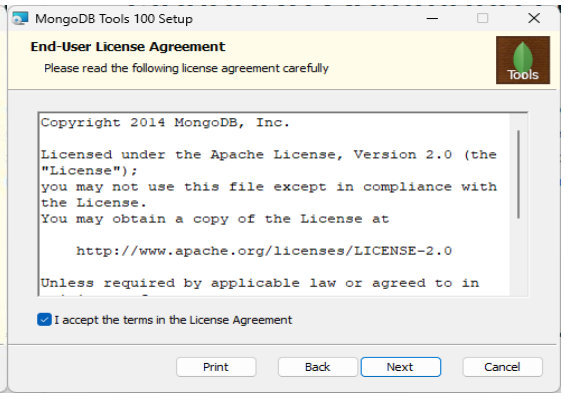
x

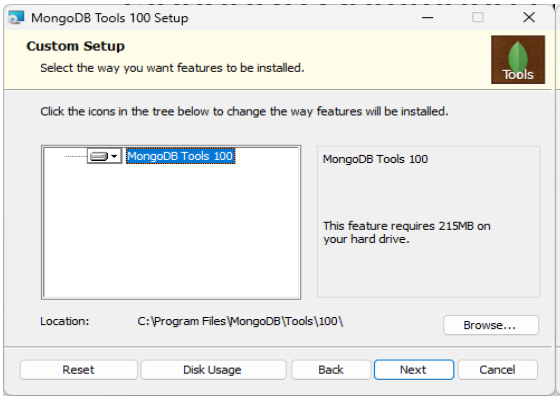
```
Library> db.users.aggregate([
...   {
...     $lookup: {
...       from: "books",
...       localField: "books",
...       foreignField: "_id",
...       as: "books"
...     }
...   },
...   {
...     $lookup: {
...       from: "profiles",
...       localField: "profileId",
...       foreignField: "_id",
...       as: "profileId"
...     }
...   },
... ]);
[
  {
    _id: 1,
    name: 'User 1',
    profileId: [ { _id: 1, address: '123 St.' } ],
    books: [
      {
        _id: ObjectId('6824588020344be4b56c4bd2'),
        title: 'Book A',
        author: 'Author A',
        sno: 2,
        year: 2021
      },
      {
        _id: ObjectId('6824588020344be4b56c4bd3'),
        title: 'Book B',
        author: 'Author B',
        sno: 3,
        year: 2020
      }
    ]
  }
]
```

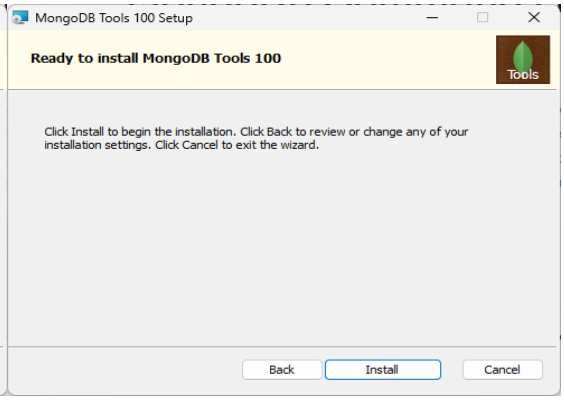
14. Grouping by author and counting the number of books per author.

```
Library> db.books.aggregate([
...   { $group: { _id: "$author", totalBooks: { $sum: 1 } } }
... ])
[
  { _id: 'Author A', totalBooks: 1 },
  { _id: 'Author B', totalBooks: 1 }
]
Library> db.books.insertOne({ title: "Book A1", author: "Author A", sno: 4, year: 2022 })
{
  acknowledged: true,
  insertedId: ObjectId('682460c920344be4b56c4bd4')
}
Library> db.books.aggregate([
...   { $group: { _id: "$author", totalBooks: { $sum: 1 } } }
... ])
[
  { _id: 'Author B', totalBooks: 1 },
  { _id: 'Author A', totalBooks: 2 }
]
Library>
```

15. To Backup and Restore MongoDB Database we first install MongoDB Tools:








```
C:\Users\user>mongodump --version
mongodump version: 100.12.0
git version: 4558399ef8d5aa59a2779d5909fe9713da43b6af
Go version: go1.23.7
    os: windows
    arch: amd64
    compiler: gc
```

- Creating a Backup of Library Database:

```
C:\Users\user>mongodump --db Library --out "C:\Homeworks\8th Sem\Advanced DB\mongoDB_backup"
2025-05-14T15:20:37.281+0545    writing Library.books to C:\Homeworks\8th Sem\Advanced DB\mongo
DB_backup\Library\books.bson
2025-05-14T15:20:37.284+0545    writing Library.profiles to C:\Homeworks\8th Sem\Advanced DB\mo
ngoDB_backup\Library\profiles.bson
2025-05-14T15:20:37.288+0545    done dumping Library.profiles (1 document)
2025-05-14T15:20:37.288+0545    done dumping Library.books (3 documents)
2025-05-14T15:20:37.294+0545    writing Library.users to C:\Homeworks\8th Sem\Advanced DB\mongo
DB_backup\Library\users.bson
2025-05-14T15:20:37.298+0545    done dumping Library.users (1 document)
```

- Restoring the Database:

```
Library> db.dropDatabase()
{ ok: 1, dropped: 'Library' }
Library> use test
switched to db test
test> show dbs
POSTQUICKLY   88.00 KiB
admin         40.00 KiB
config       108.00 KiB
local        168.00 KiB
test>
```

```
C:\Users\user>mongorestore --db Library "C:\Homeworks\8th Sem\Advanced DB\mongoDB_backup\Librar
y"
2025-05-14T15:26:58.812+0545    The --db and --collection flags are deprecated for this use-cas
e; please use --nsInclude instead, i.e. with --nsInclude=${DATABASE}.${COLLECTION}
2025-05-14T15:26:58.814+0545    building a list of collections to restore from C:\Homeworks\8th
 Sem\Advanced DB\mongoDB_backup\Library dir
2025-05-14T15:26:58.814+0545    don't know what to do with file "C:\Homeworks\8th Sem\Advanced
DB\mongoDB_backup\Library\prelude.json", skipping...
2025-05-14T15:26:58.815+0545    reading metadata for Library.books from C:\Homeworks\8th Sem\Ad
vanced DB\mongoDB_backup\Library\books.metadata.json
2025-05-14T15:26:58.815+0545    reading metadata for Library.profiles from C:\Homeworks\8th Sem
\Advanced DB\mongoDB_backup\Library\profiles.metadata.json
2025-05-14T15:26:58.815+0545    reading metadata for Library.users from C:\Homeworks\8th Sem\Ad
vanced DB\mongoDB_backup\Library\users.metadata.json
2025-05-14T15:26:58.845+0545    restoring Library.books from C:\Homeworks\8th Sem\Advanced DB\m
ongoDB_backup\Library\books.bson
2025-05-14T15:26:58.856+0545    restoring Library.users from C:\Homeworks\8th Sem\Advanced DB\m
ongoDB_backup\Library\users.bson
2025-05-14T15:26:58.856+0545    finished restoring Library.books (3 documents, 0 failures)
2025-05-14T15:26:58.861+0545    restoring Library.profiles from C:\Homeworks\8th Sem\Advanced D
B\mongoDB_backup\Library\profiles.bson
2025-05-14T15:26:58.866+0545    finished restoring Library.users (1 document, 0 failures)
2025-05-14T15:26:58.871+0545    finished restoring Library.profiles (1 document, 0 failures)
2025-05-14T15:26:58.871+0545    no indexes to restore for collection Library.books
2025-05-14T15:26:58.872+0545    no indexes to restore for collection Library.profiles
2025-05-14T15:26:58.872+0545    no indexes to restore for collection Library.users
2025-05-14T15:26:58.872+0545    5 document(s) restored successfully. 0 document(s) failed to re
store.
```

- Library is restored:

```
test> show dbs
Library      120.00 KiB
POSTQUICKLY   88.00 KiB
admin         40.00 KiB
config       108.00 KiB
local        168.00 KiB
```