1. Implement  object oriented relational model to demonstrate inheritance feature.

create table account (

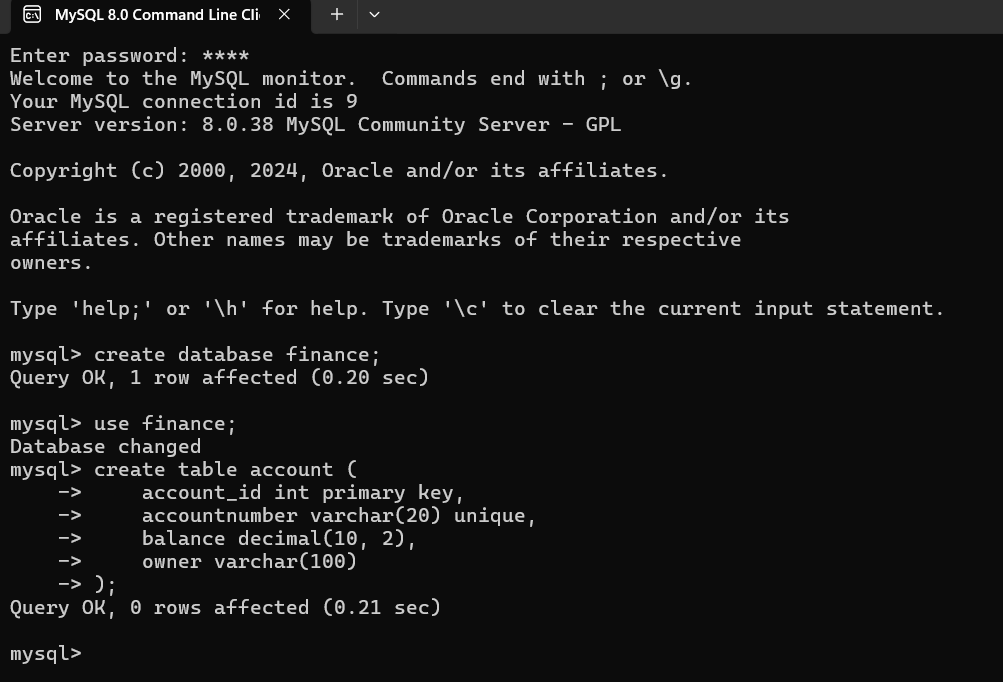
account\_id int primary key,

accountnumber varchar(20) unique,

balance decimal(10, 2),

owner varchar(100)

);



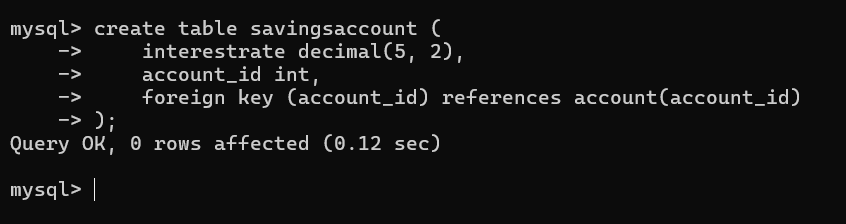
create table savingsaccount (

interestrate decimal(5, 2),

account\_id int,

foreign key (account\_id) references account(account\_id)

);



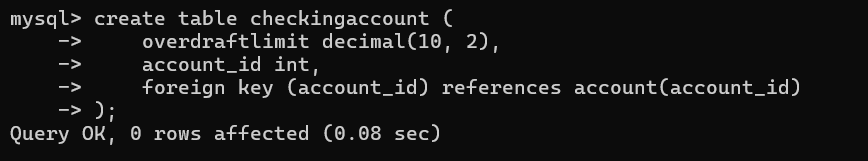
create table checkingaccount (

overdraftlimit decimal(10, 2),

account\_id int,

foreign key (account\_id) references account(account\_id)

);



1. Implement hierarchical relational data model

create table service (

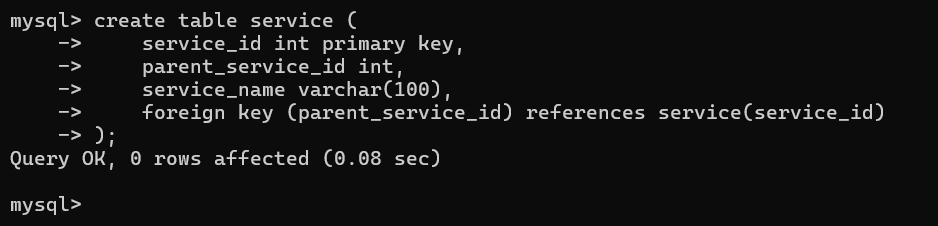
service\_id int primary key,

parent\_service\_id int,

service\_name varchar(100),

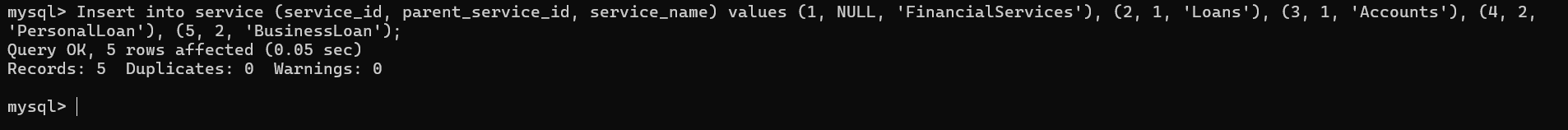
foreign key (parent\_service\_id) references service(service\_id)

);

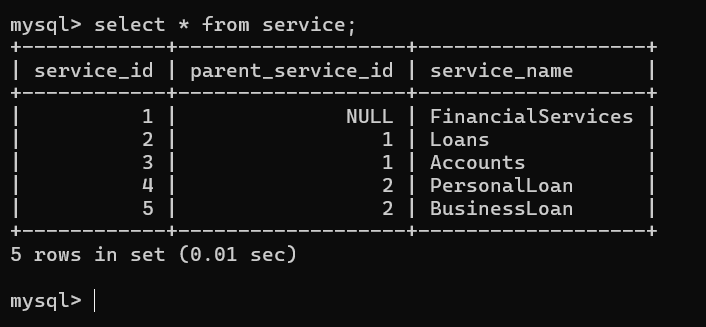


Insert :

Insert into service (service\_id, parent\_service\_id, service\_name) values (1, NULL, 'FinancialServices'), (2, 1, 'Loans'), (3, 1, 'Accounts'), (4, 2, 'PersonalLoan'), (5, 2, 'BusinessLoan');



Output Table:



1. Perform crud operation in mongodb

Create :

db.customers.insert({

customerID: 101,

name: "John Doe",

accountDetails: [

{ accountType: "Savings", balance: 5000 },

{ accountType: "Checking", balance: 3000 }

],

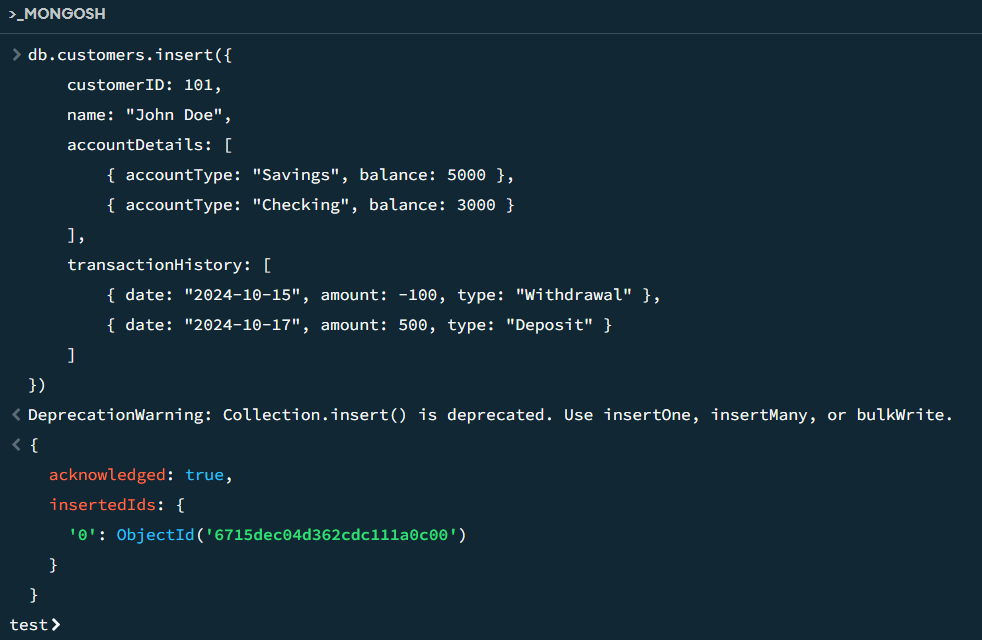
transactionHistory: [

{ date: "2024-10-15", amount: -100, type: "Withdrawal" },

{ date: "2024-10-17", amount: 500, type: "Deposit" }

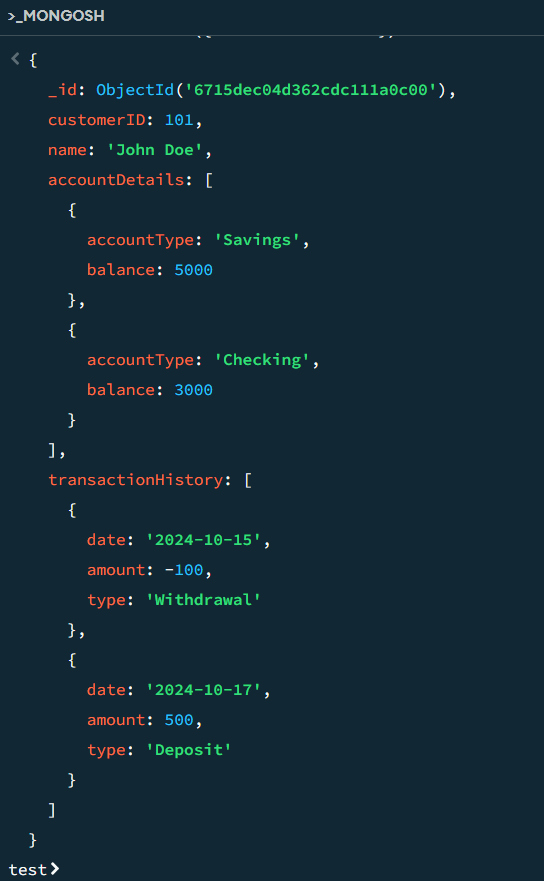
]

})



Read :

db.customers.find({ customerID: 101 })



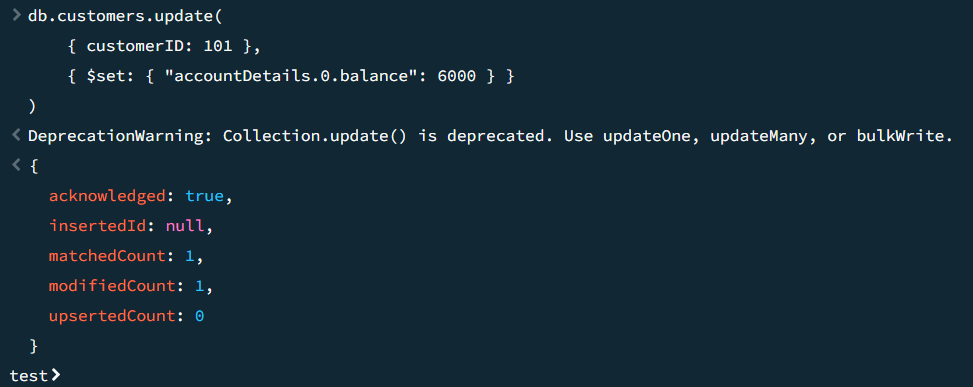
Update :

db.customers.update(

{ customerID: 101 },

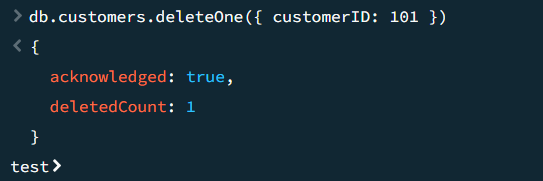
{ $set: { "accountDetails.0.balance": 6000 } }

)



Delete:

db.customers.deleteOne({ customerID: 101 })



1. Create a property graph in neo4j  using cypher query language  (OR)  Create a RDF graph in graphdb using sparql.

CREATE (customer1:Customer {name: 'Alice', customerID: 1001}),

(customer2:Customer {name: 'Bob', customerID: 1002}),

(loan1:Loan {loanID: 'LN001', amount: 10000, interestRate: 5.5}),

(loan2:Loan {loanID: 'LN002', amount: 5000, interestRate: 4.7}),

(creditCard1:CreditCard {cardID: 'CC001', limit: 2000, expirationDate: '2025-12-31'}),

(creditCard2:CreditCard {cardID: 'CC002', limit: 5000, expirationDate: '2026-06-30'});

CREATE (customer1)-[:APPLIES\_FOR]->(loan1),

(customer2)-[:APPLIES\_FOR]->(loan2),

(customer1)-[:OWNS]->(creditCard1),

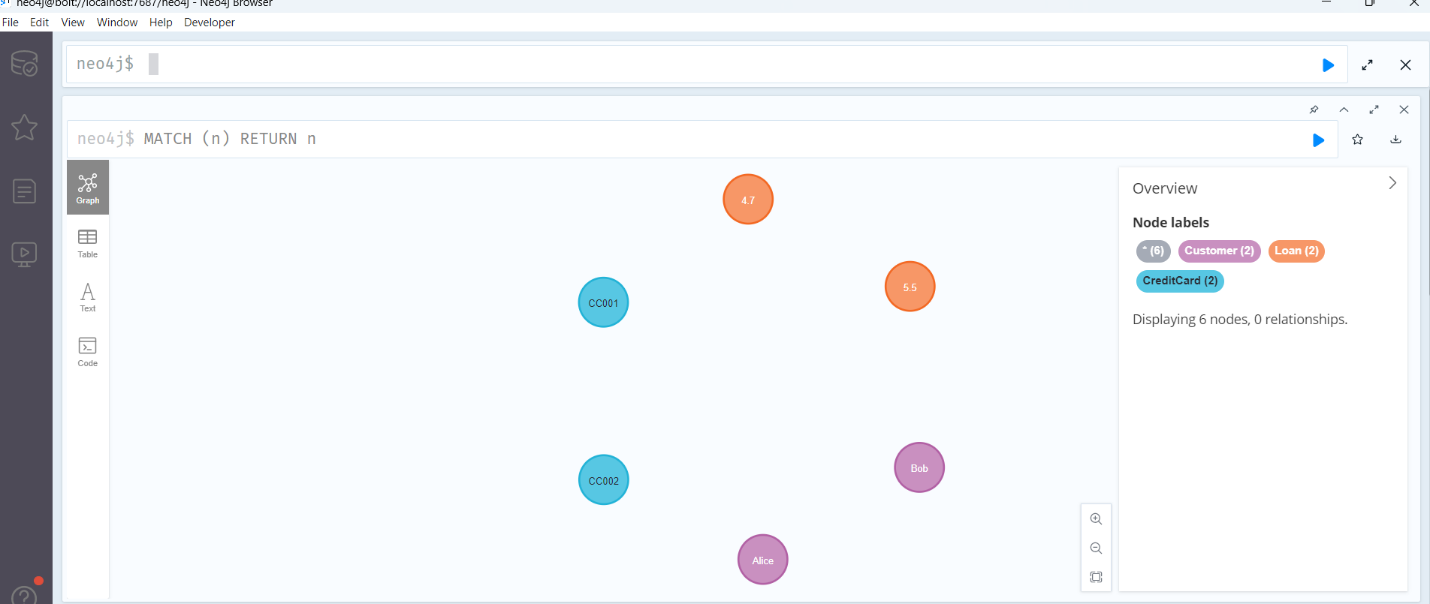
(customer2)-[:OWNS]->(creditCard2);

MATCH (c:Customer)-[:APPLIES\_FOR]->(l:Loan)

RETURN c.name AS Customer, l.loanID AS LoanID, l.amount AS LoanAmount;

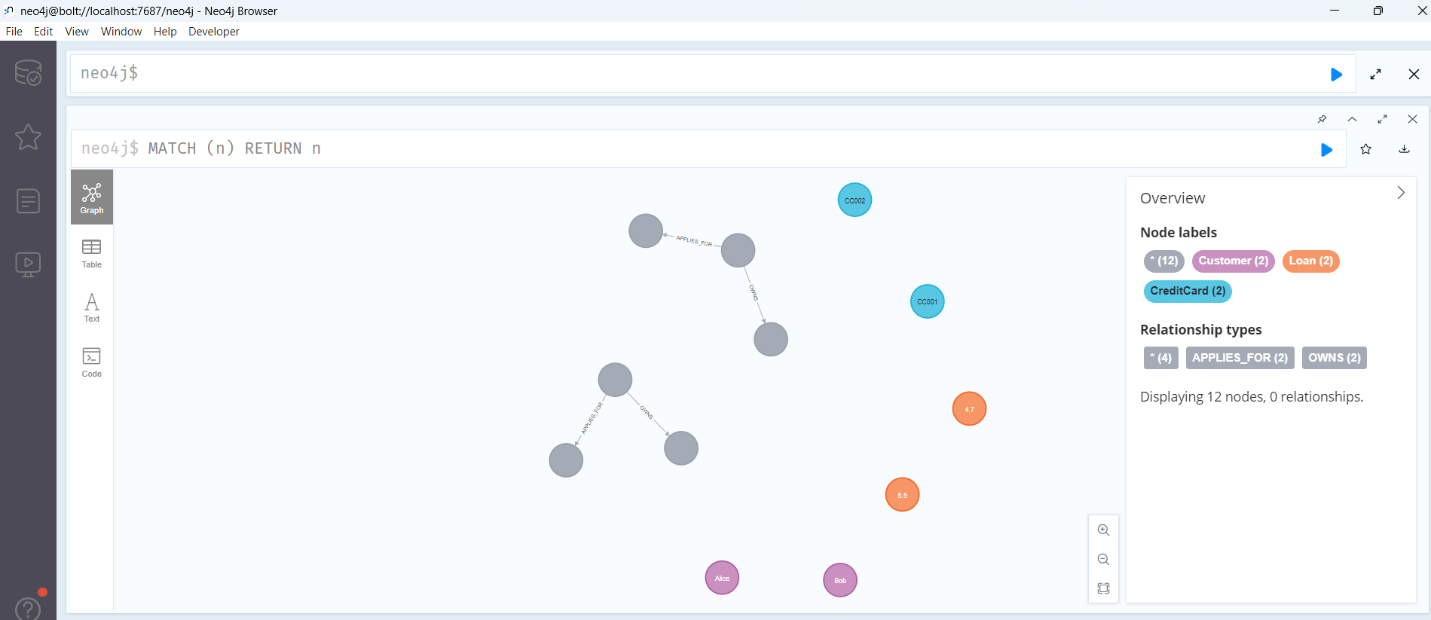
MATCH (c:Customer)-[:OWNS]->(cc:CreditCard)

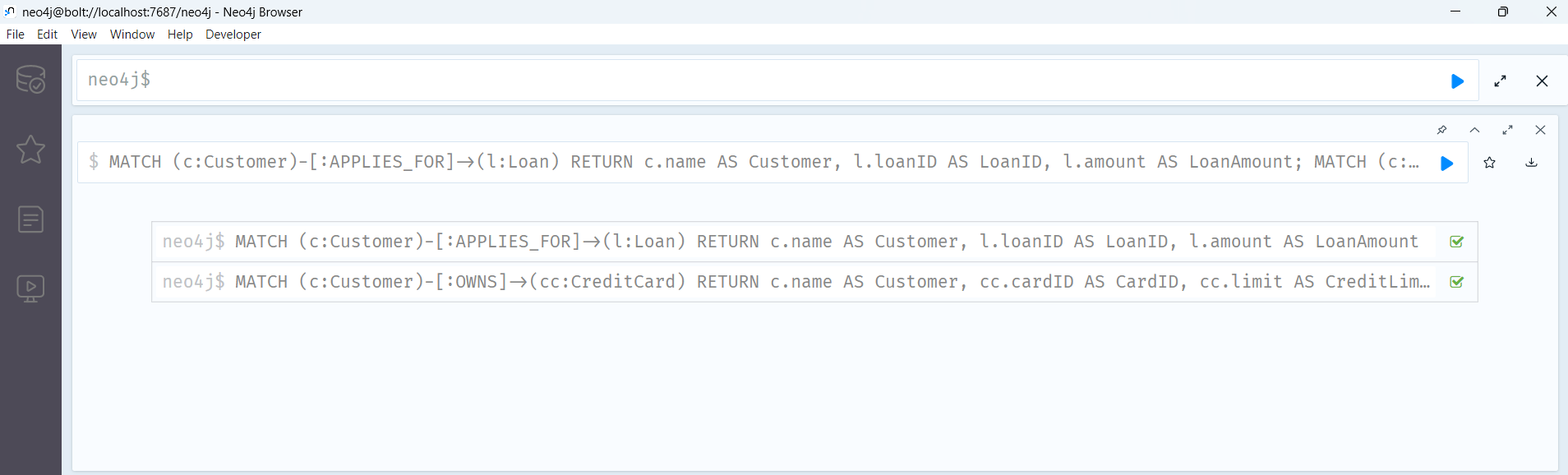
RETURN c.name AS Customer, cc.cardID AS CardID, cc.limit AS CreditLimit;





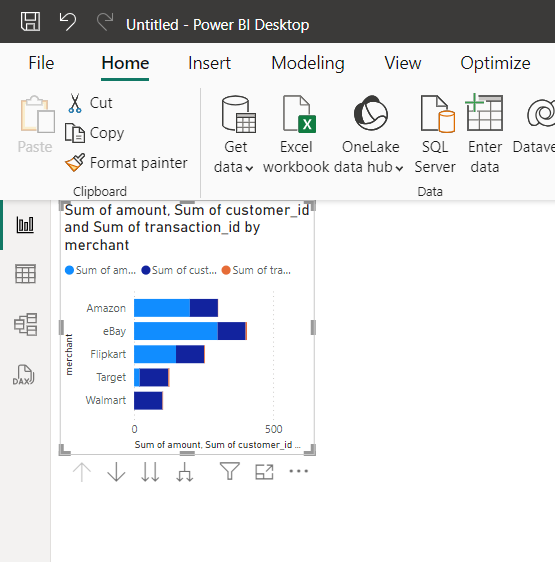




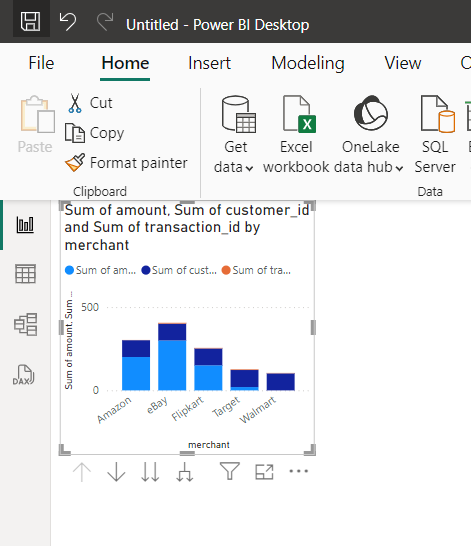


1. Usage of any one DE tools to illustrate the DE flow of your choice - POWER BI

Stacked Bar Chart :



Stacked Column Chart :



Line Chart :

