

IFT 530: Advanced DataBase Management Systems (2023 Spring)

Project Title: Insurance Policy Database Management System

530 Final Project Written Submission

Submitted to Dr. Robert Rucker

By

Devi Subadra Venkatesan (ASU ID: 1228251806)

Balasubramanyam Prema Doddanari (ASU ID: 1227720899)

Praveen Kumar Siddela (ASU ID:1227371004)

On 28th April 2023

Table of Contents

Section 0: Abstract	01
Section 1: Introduction and Importance of area	03
Section 2: ORM Diagram	07
Section 3: Relational Schema	11
Section 5: Relational Tables and Column Constraints.	14
Section 5.1: Answers for Questions	25
Section 6: No SQL – Couchbase	34
Section 6.1: Extended Nested Entities in Couchbase from ORM	35
Section 6.2: Creating Json and inserting documents.	37
Section 6.3: Link Service on the Couchbase to Analytic Service	44
Section 6.4: Queries for questions	46
Section 7: Summary	51
Section 7.1: Conclusion	52
References	53

Introduction for Insurance Policy Database Management System (IPDMS)

An insurance management system is a application designed to help insurance companies manage their policies, claims, and other related activities more efficiently. The system provides a platform for insurers to manage their business operations, from underwriting policies to handling claims, customer service, and compliance with regulatory requirements.

The system is designed to streamline the insurance process, reduce operational costs, and improve customer service. It enables insurers to manage policies more effectively, automate the claims process, and track and analyze data to identify patterns and trends that could lead to potential losses.

Insurance management systems come with a range of features and capabilities, including policy management, claims processing, reporting and analytics, compliance management, and customer relationship management. The system can be customized to meet the specific needs of an insurance company, allowing insurers to tailor the software to fit their unique business requirements.

Importance of the IPDMS

An insurance management system is an essential tool for efficiently managing the complex operations of the insurance industry. Its importance cannot be overstated, as it can streamline processes, ensure compliance, and manage risks effectively. By automating tasks such as data entry and claims processing, insurers can enhance operational efficiency, reduce errors, and optimize resource utilization, leading to cost savings. The system's ability to improve customer service is another significant advantage. Automated claims processing with real-time updates can

result in better customer satisfaction and retention rates, contributing to the long-term success of the company. Furthermore, an insurance management system provides a centralized database for all policies, claims, and relevant information, allowing insurers to manage and monitor data accurately. This, in turn, can help insurers make informed decisions based on precise data analysis, leading to better risk management and profitability. The system's role in ensuring regulatory compliance is crucial as insurance companies are obligated to adhere to strict regulatory requirements, and non-compliance can lead to costly penalties and legal issues. The system can provide alerts and reminders, helping insurers meet deadlines and complete necessary documentation. Lastly, an insurance management system can help insurers manage risks associated with underwriting policies by providing real-time data analysis and identifying potential losses. In summary, an insurance management system is a necessary tool for insurers to stay competitive in today's fast-paced business environment, as it can improve operational efficiency, customer service, compliance, risk management, and contribute to long-term success.

Description of Entities

The entities are "Customer", "Insurer", "Policy", "Policy Payment", "PolicyTypes", "Address". The First entity is Insurer , in which the primary key is insurereID, Foreign key is customerid, there are other attributes which are username, firstname, and lastname they were all self-referenced and marked with Mandatory and added Unique Constraints. The second entity is Customer, in which the primary key is customerid and the foreign key are PhoneNum and EmailId which are connected with Inclusive Or Constraint and addressid which will be connected directly to Address Entity which will store the address info of the Customers with a foreign key as countryCode and stateCode.

The main Entity is Policy as it is connected to the Customer with a Many-to-Many relation, PolicyPayment, and PolicyTypes entities with Policyid as a primary key along with InsuranceStatus which is depicted with Exclusive Or Constraint to represent any one status {Sanctioned,Cancelled, Renewed} is allowed. It has a Ternary relation of PolicyHasPolicyPaymentWhereThePaymentDate through policypaymentid from Policypayment with two subtypes (PaymentStatus and PaymentType) which are differentiated with Exclusive Or Constraint.

The Policy table is connected to the second major Entity which is Types, they are subcategorized into three groups -AutoPolicy(Automobiles), HousingPolicy and LifeInsurance.

AutoPolicy- Checks whether the vehicle has any accidents if so then it would deduct the amount with autoPolicyName and foreign key -accidentid

HousingPolicy- Insurer checks the House conditions with Yearbuilt and any prev policies or claims on the house.

LifeInsurance-Term-MedicalPolicy it covers the medical insurances for customers for certain period of time and UniversalPolicy ensure the permanent insurance covered even for the death of the customer by “checking the basic eligible criteria (i.e) Age limit shouldn’t be greater than 60 years and health condition should be healthy.

List of the queries that are answered as part of the project:

Stored Procedures:

1. Get all the Policy details as a particular procedure.

Example: If we want to get all the details of Policy, instead of executing it every time again and again, we can use a procedure called 'getCustomerPolicyDetails' which stores the SQL query.

2. Is there any Stored Procedure to retrieve Customer Payment policy details? Creates a stored procedure that selects from a particular state from the "Airport" table.

Example: If we want the payment details for a particular customer then this Stored procedure 'getCustomerPaymentPolicyDetails' returns the Customer, Policy and payment details.

Triggers:

1. Triggers to perform Insert operation and that notify Customer if a Policy Payment Fails?

Example: 'CustomerPaymentFailureNotify_Trigger' this trigger will insert the payment failure records in the EventLogger table. This trigger will be triggered whenever a record got inserted in PolicyPayment table.

2. Triggers to perform Insert operation and that notifies the Customer policies which ends in a week?

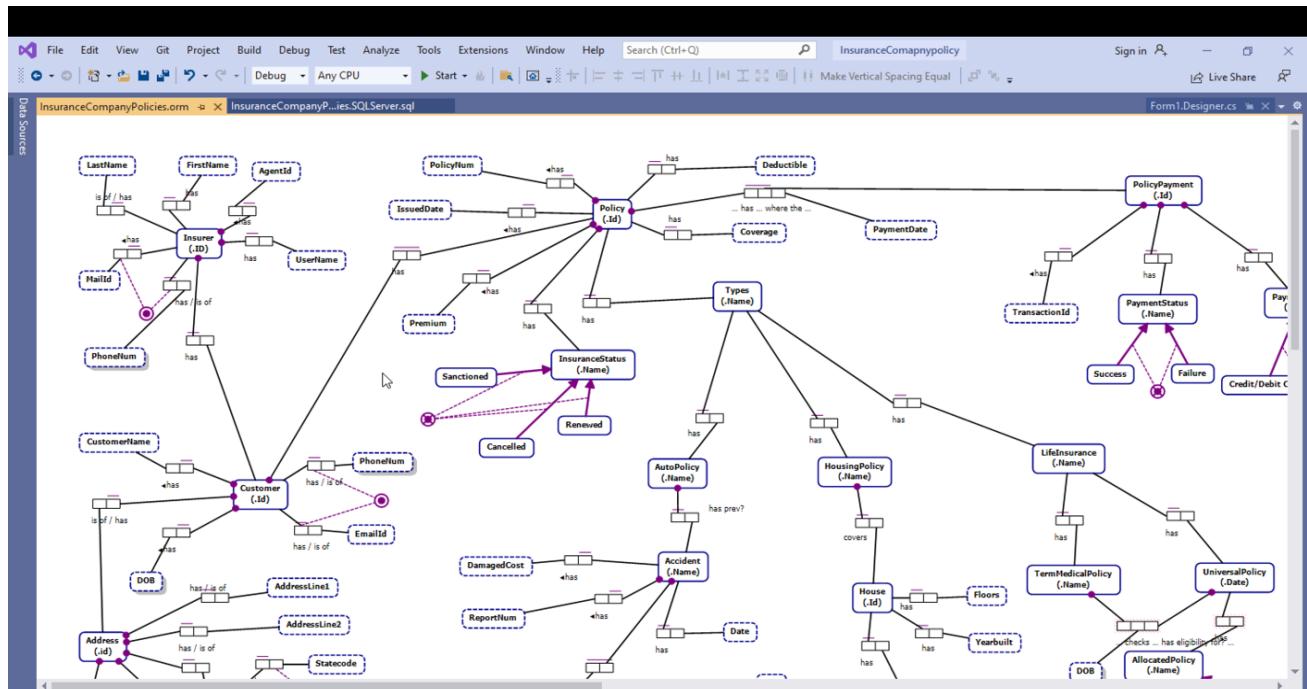
Example: 'CustomerPolicyDeadlineNotify_Trigger' this trigger will insert the Customer policy records which have policy Expire deadline of 7 days. This trigger will be triggered whenever a record get inserted in Policy table

Questions:

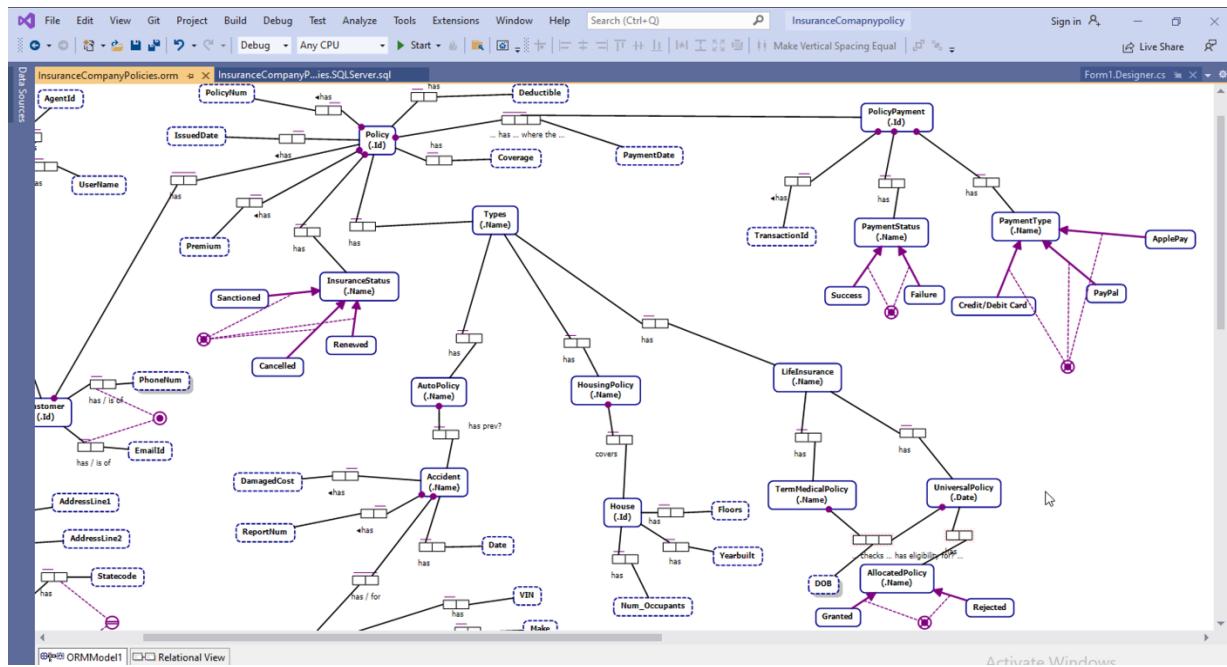
1. How many policies were issued in a particular time period?
2. How many Active Policies are there in the system?
3. Which Customer has the Maximum coverage amount?
4. Find out the Customer details who has paid their policy payment through Netbanking also display the Policy coverage amount?
5. Is there any Stored procedure to retrieve Policy details for a Customer?
6. Is there any Stored procedure to retrieve Customer Payment Policy details?
7. Is there any trigger that notify Customer if a Policy Payment fails?
8. Is there any trigger that notifies the Customer policies which ends in a week?

ORM Diagram

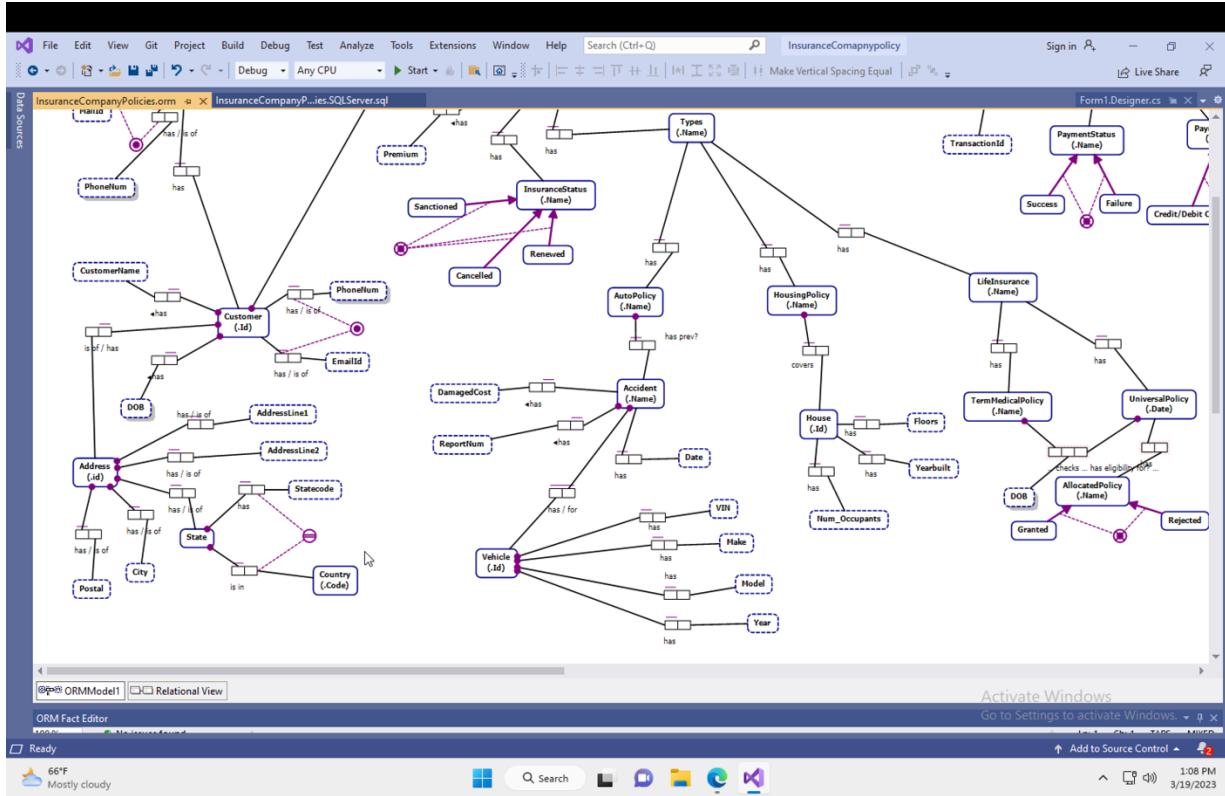
a)



b)



c)



Entities of the database with their attributes:

1. Insurer.ID

- FirstName
- LastName
- AgentId
- UserName
- PhoneNum
- MailId

2. Customer.ID

- CustomerName

- DOB
- PhoneNum
- EmailId

3. Policy(.ID)

- IssuedDate
- PolicyNum
- coverage
- Deductible

4. PolicyPayment(.ID)

- PaymentType
- PaymentStatus
- TransactionId

5. Types(.Name)

- AutoPolicy
- HousingPolicy
- LifeInsurance

6. Accident(.Name)

- DamagedCost
- ReportNum
- Date

7. House(.Name)

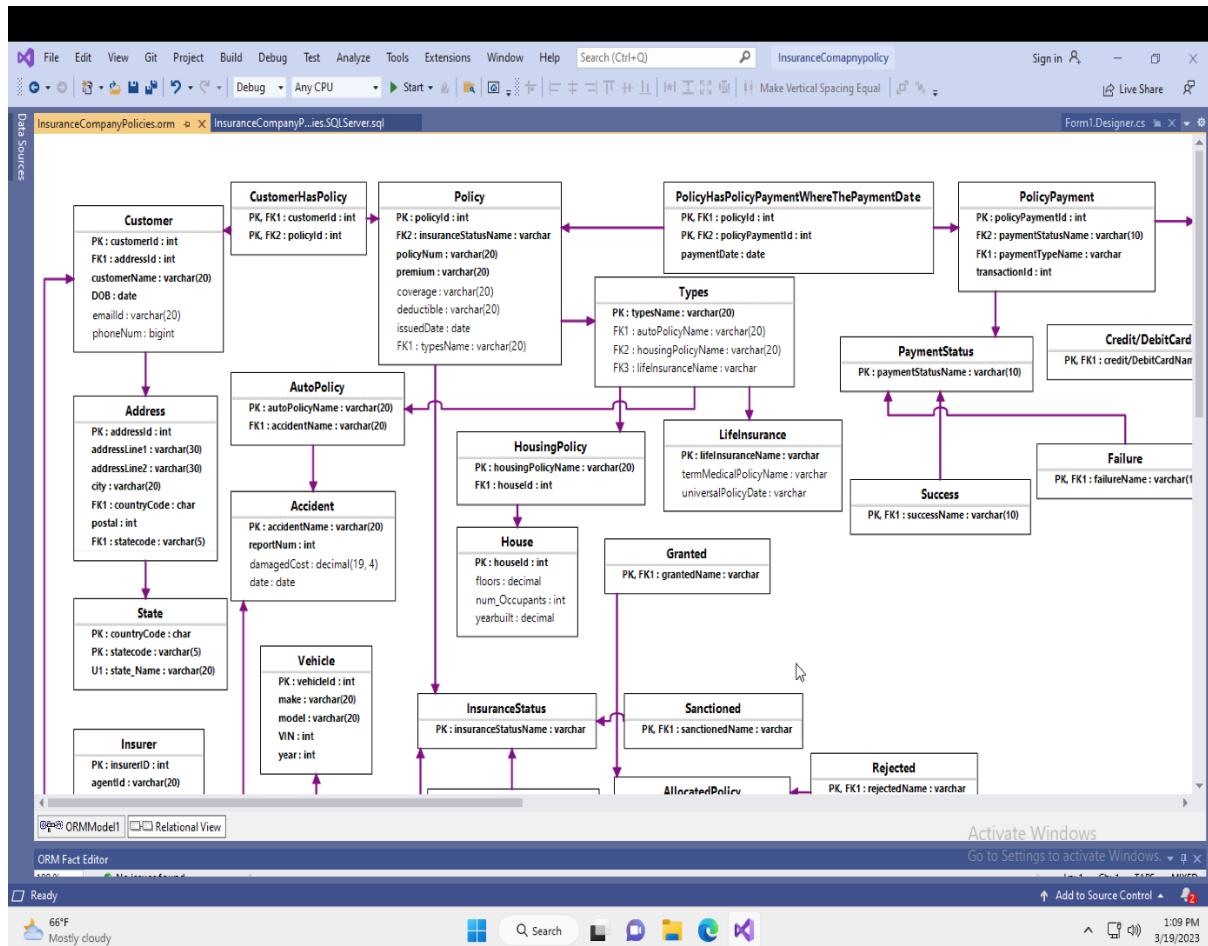
- Floors
- YearBuilt

- occupants

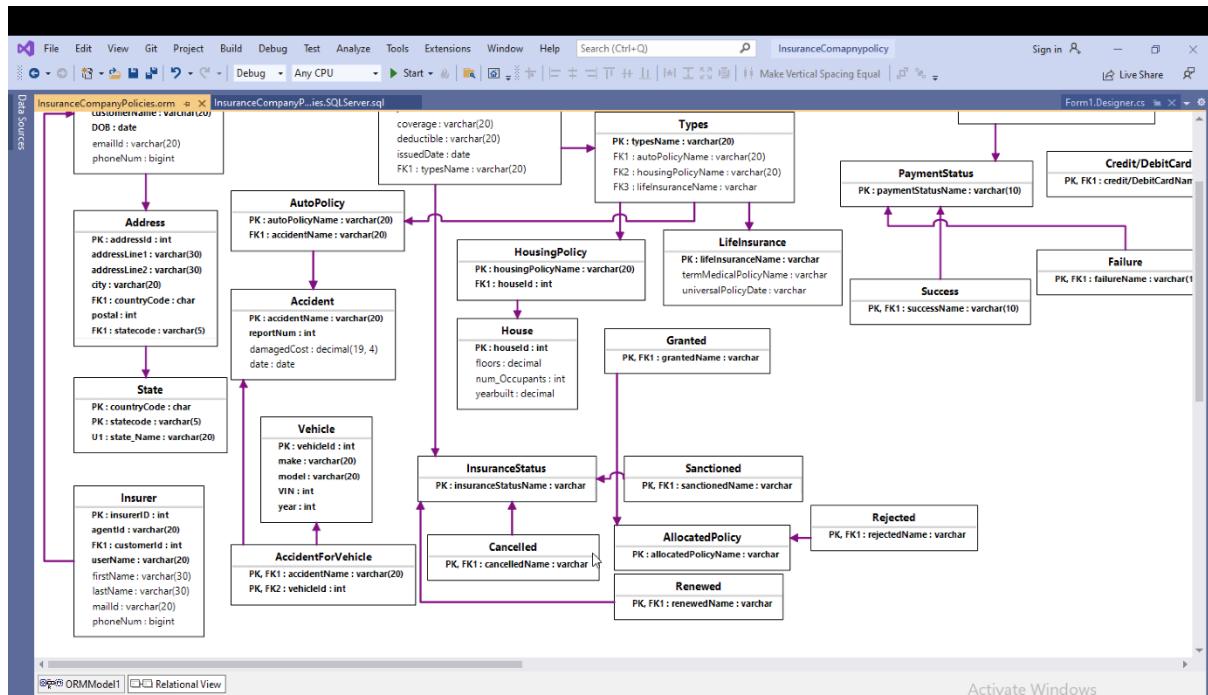
8. Vehicle(ID)

- VIN
- Make
- Model
- Year

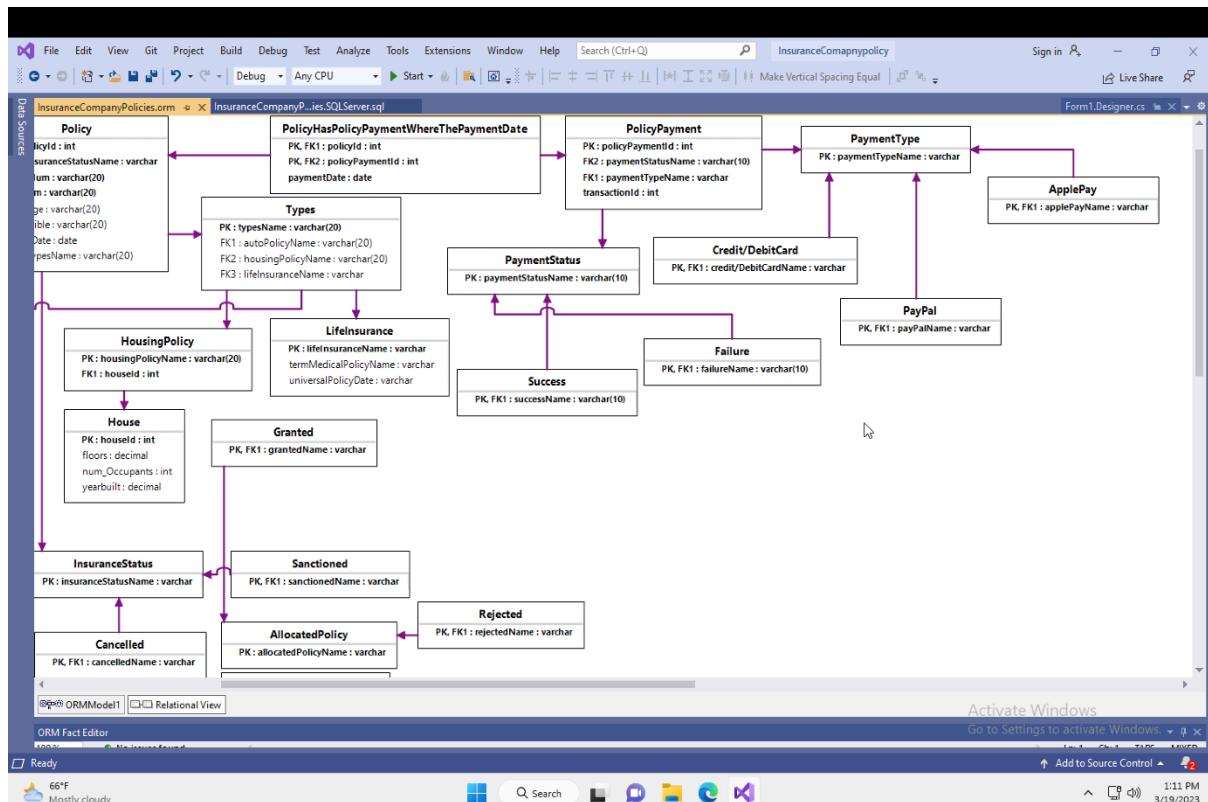
Relational Schema



b)



c)



6. Job Flow:

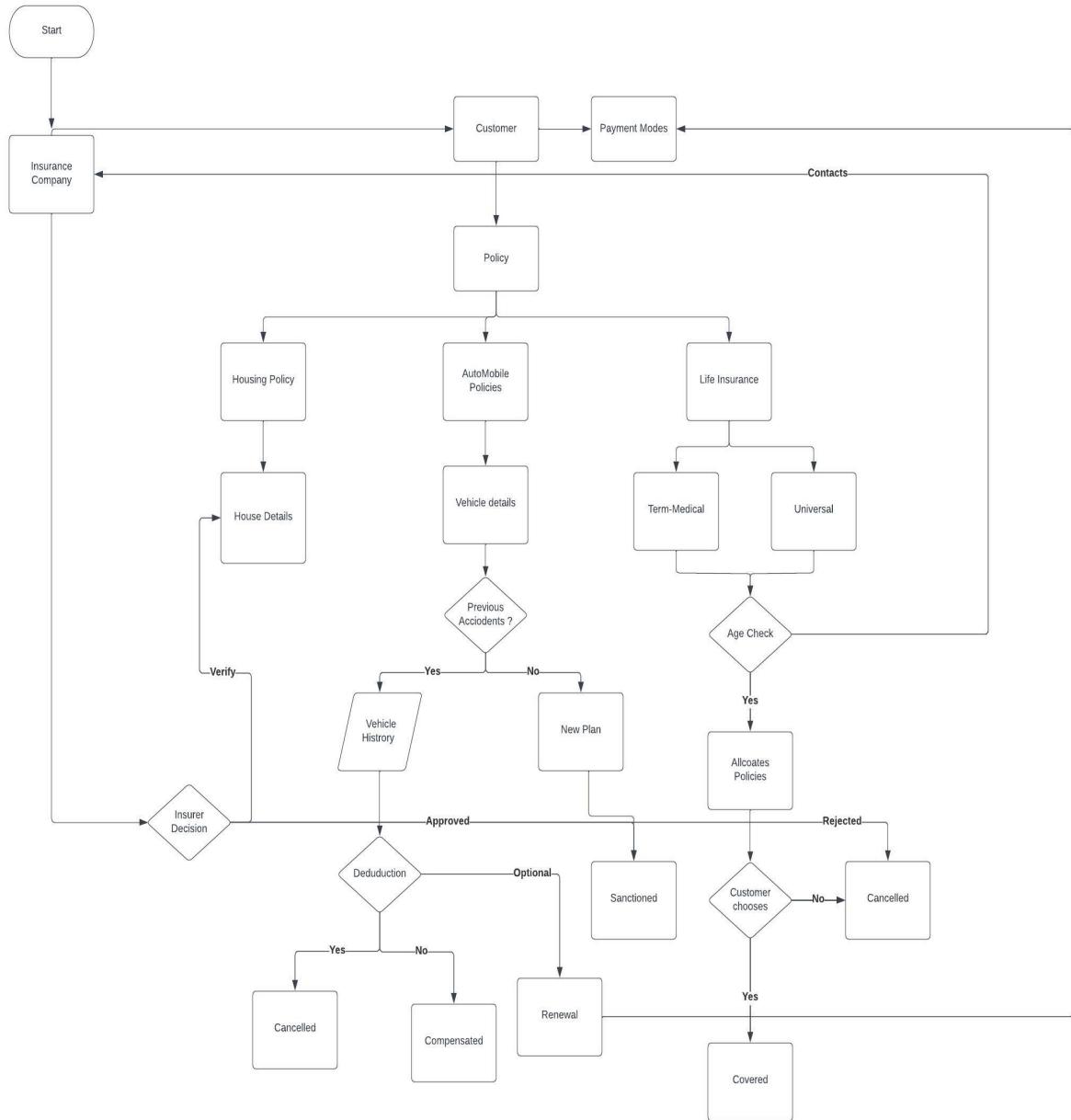


Table Implementation, Datatypes and Column Constraints

In this project, we have used the following column constraints:

1. NOT NULL: The values in this column should not be null.
2. UNIQUE: The UNIQUE constraint ensures that all values in a column are different.
3. PRIMARY KEY: It is a unique key constraint which accepts only unique values in the data records and rejects any duplicate values.
4. FOREIGN KEY: It references the data from another table using a common column attribute to fetch the data.

1. Customer(ID) – Primary Key

- CustomerName: Varchar(20)
- DOB: date
- PhoneNum: bigint
- EmailId :Varchar(20)
- addressId: Int- Foreign Key

2. Policy(ID) – Primary Key

- IssuedDate: date
- PolicyNum: Varchar(20)
- coverage:varchar(20)
- typeName:Varchar(20)- Foreign Key

3. PolicyPayment(ID) – Primary Key

- Paymentmethod: varchar(20)
- PaymentStatus:varchar(20)
- TransactionId:int

4. Types.(Name)-Varchar(20)

- AutoPolicy
- HousingPolicy
- LifeInsurance

DML & DDL

DDL and DML statements are used to define the data in the data bases and perform data manipulation

Data Definition Language (DDL): DDL statements are used to create, modify, or delete database objects such as tables, indexes, and constraints. We have used some DDL commands like, CREATE command for creating the database for the project, ALTER statements to manipulate the structure.

Example: Create a database to store all tables and create tables and then create the tables.

Data Manipulation Language (DML): It is used to manipulate the existing data. We have used statements like INSERT to enter data into the student- university admissions databases and UPDATE to perform any updating in the existing data.

Example: INSERT statement to store Customer details

Database Tables

```
CREATE DATABASE Insurance
```

```
CREATE TABLE Customer
```

```
(  
    CustomerId int IDENTITY (1, 1) NOT NULL,  
    Firstname nvarchar(20) NOT NULL,  
    LastName nvarchar(20) Not NULL,  
    DOBDate date NOT NULL,  
    EmailId nvarchar(20),  
    PhoneNum bigint not null,  
    City nvarchar(20) NOT NULL,  
    State nvarchar(20) Not NULL,  
    ZipCode nvarchar(20) NOT NULL,  
    CONSTRAINT Customer_PK PRIMARY KEY(customerId)  
)  
GO
```

```
CREATE TABLE Policy
```

```
(  
    PolicyId int IDENTITY (1, 1) NOT NULL,  
    PolicyTypeID INT NOT NULL,  
    CustomerID INT NOT NULL,  
    Coverage bigint NOT NULL,  
    IssuedDate DateTime NOT NULL,  
    Expiry DateTime NOT NULL,  
    CONSTRAINT Policy_PK PRIMARY KEY(PolicyId)  
)  
GO
```

```
CREATE TABLE PolicyTypes
```

```
(  
    PolicyTypeID INT IDENTITY (1, 1) NOT NULL,  
    PolicyName nvarchar(20) NOT NULL,  
    CONSTRAINT Types_PK PRIMARY KEY(PolicyTypeID)  
)  
GO
```

```
CREATE TABLE PolicyPayment  
(  
    PaymentId int IDENTITY (1, 1) NOT NULL,  
    PolicyID int NOT NULL,  
    PaymentStatus nvarchar(30) NOT NULL,  
    PaymentType nvarchar(30) NOT NULL,  
    TransactionId bigint NOT NULL,  
    CONSTRAINT PolicyPayment_PK PRIMARY KEY(PaymentId)  
)  
GO
```

```
ALTER TABLE Policy ADD CONSTRAINT Customer_FK FOREIGN KEY (CustomerID) REFERENCES  
Customer(CustomerID) ON DELETE NO ACTION ON UPDATE NO ACTION  
GO
```

```
ALTER TABLE Policy ADD CONSTRAINT PolicyType_FK FOREIGN KEY (PolicyTypeID) REFERENCES  
PolicyTypes(PolicyTypeID) ON DELETE NO ACTION ON UPDATE NO ACTION  
GO
```

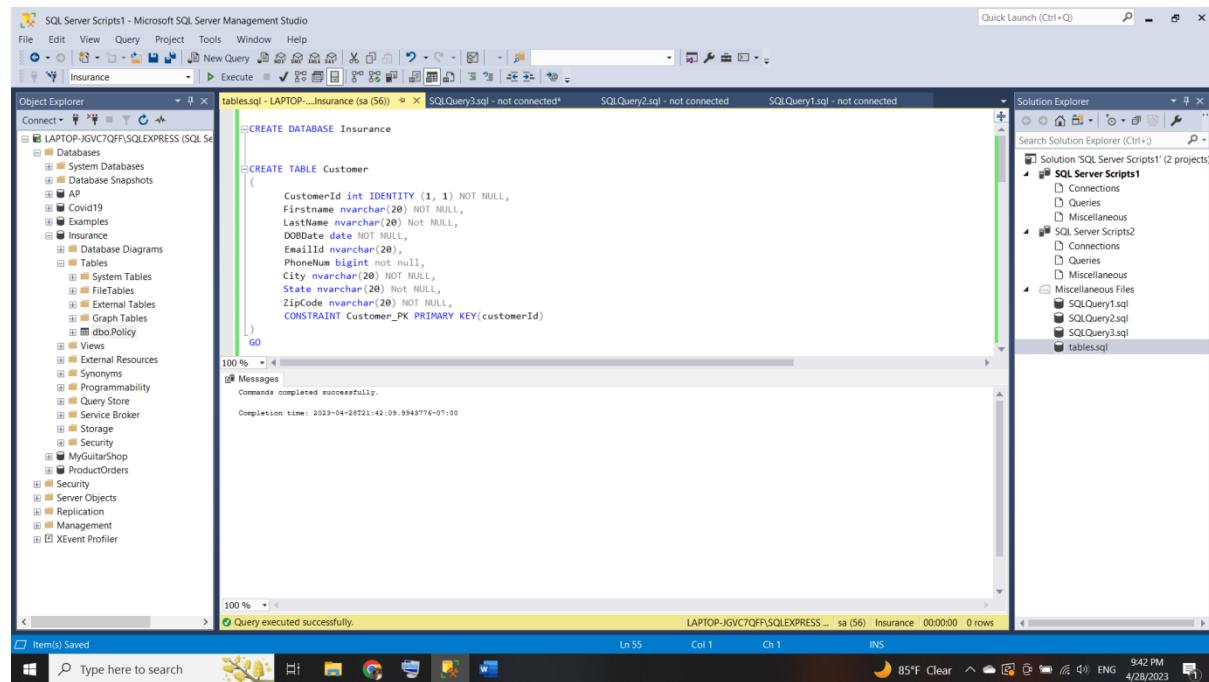
```
ALTER TABLE PolicyPayment ADD CONSTRAINT PolicyPayment_FK FOREIGN KEY (PolicyID)  
REFERENCES Policy(PolicyID) ON DELETE NO ACTION ON UPDATE NO ACTION  
GO
```

```
CREATE TABLE EventLogger
```

```
(CustomerId INT ,
PolicyID INT,
Logtime DateTime,
[Message] NVARCHAR(100))
GO
```

```
CREATE TABLE CustomerPolicyDeadline
```

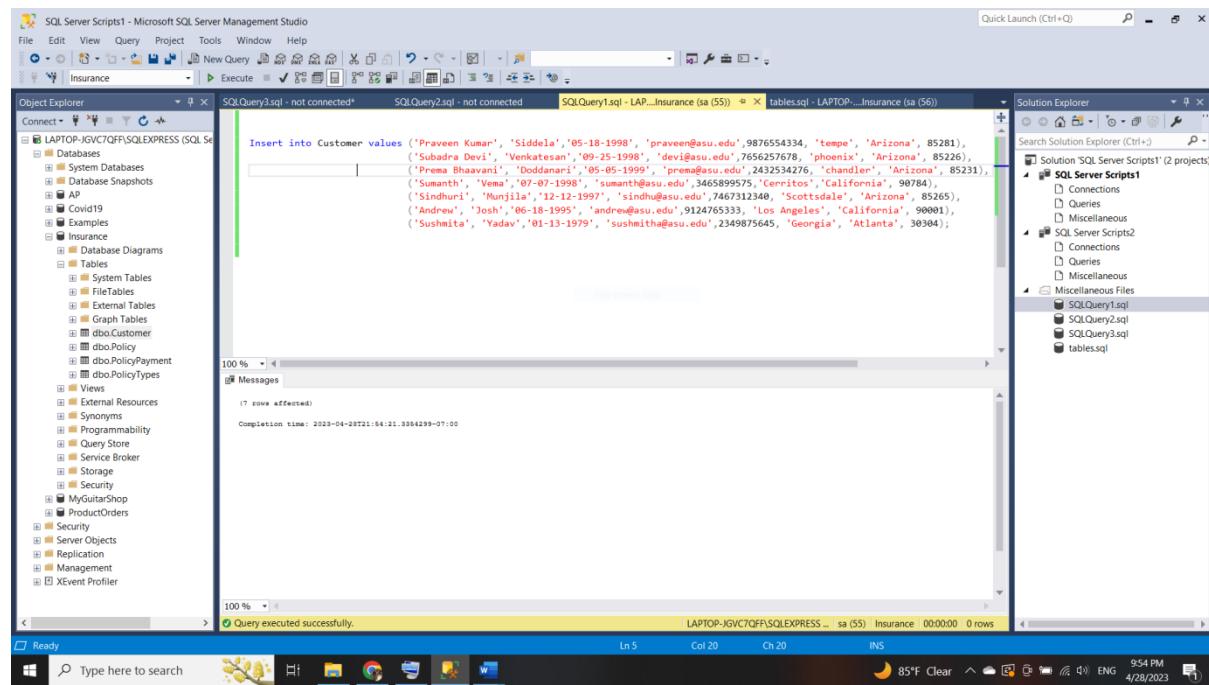
```
(CustomerId INT ,
Firstname nvarchar(20) NOT NULL,
LastName nvarchar(20) Not NULL,
PolicyID INT,
Coverage bigint NOT NULL,
IssuedDate DateTime NOT NULL,
Expiry DateTime NOT NULL,
Logtime DateTime,
[Message] NVARCHAR(100))
GO
```



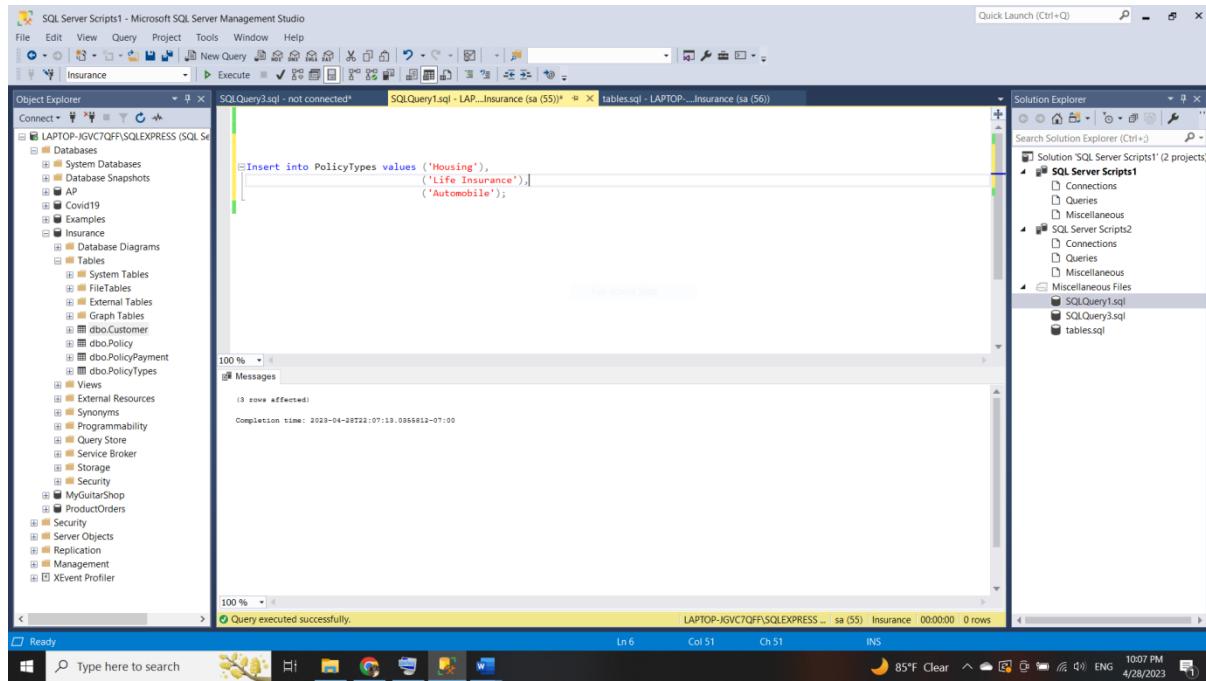
Inserting Data into Tables:

Insert into Customer values

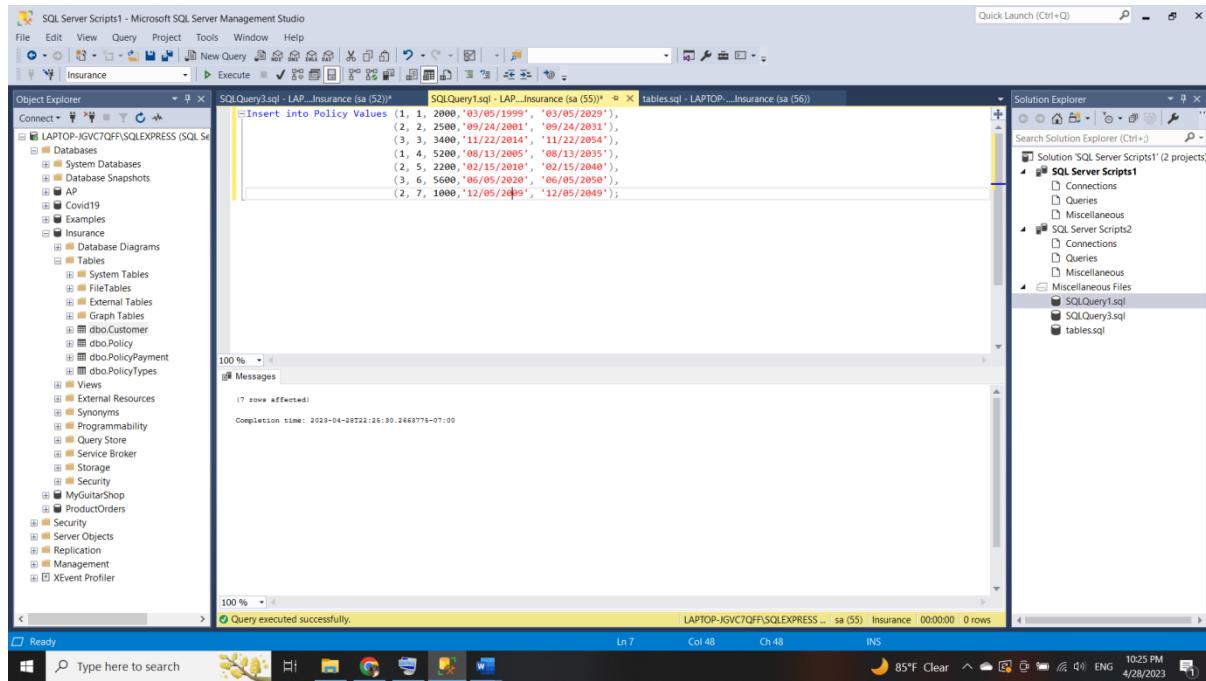
```
('Praveen Kumar', 'Siddela', '05-18-1998', 'praveen@asu.edu', 9876554334, 'tempe', 'Arizona', 85281),
('Subadra Devi', 'Venkatesan', '09-25-1998', 'devi@asu.edu', 7656257678, 'phoenix', 'Arizona', 85226),
('Prema Bhaavani', 'Doddanari', '05-05-1999', 'prema@asu.edu', 2432534276, 'chandler', 'Arizona', 85231),
('Sumanth', 'Vema', '07-07-1998', 'sumanth@asu.edu', 3465899575, 'Cerritos', 'California', 90784),
('Sindhuri', 'Munjila', '12-12-1997', 'sindhu@asu.edu', 7467312340, 'Scottsdale', 'Arizona', 85265),
('Andrew', 'Josh', '06-18-1995', 'andrew@asu.edu', 9124765333, 'Los Angeles', 'California', 90001),
('Sushmita', 'Yadav', '01-13-1979', 'sushmitha@asu.edu', 2349875645, 'Georgia', 'Atlanta', 30304);
```



```
INSERT INTO PolicyTypes VALUES ('Housing'),
('Life Insurance'),
('Automobile');
```



```
INSERT INTO PolicyTypes VALUES (1, 1, 2000, '03/05/1999', '03/05/2029'),
(2, 2, 2500, '09/24/2001', '09/24/2031'),
(3, 3, 3400, '11/22/2014', '11/22/2054'),
(1, 4, 5200, '08/13/2005', '08/13/2035'),
(2, 5, 2200, '02/15/2010', '02/15/2040'),
(3, 6, 5600, '06/05/2020', '06/05/2050'),
(2, 7, 1000, '12/05/2009', '12/05/2049);
```



```
Insert into PolicyPayment values(1,'Success','Cash',9127),
(2,'Success','Credit/Debit',7389),
(3,'Success','Netbanking',2739),
(4,'Success','Cash',1935),
(5,'Fail','Credit/Debit',4792),
(6,'Success','Credit/Debit',6500),
(7,'Fail','Netbanking',2794);
```

```

-- Insert into PolicyPayment values
(1,'Success','Cash',9127),
(2,'Success','Credit/Debit',7389),
(3,'Success','Netbanking',2739),
(4,'Success','Cash',1000),
(5,'Fail','Credit/Debit',4792),
(6,'Success','Credit/Debit',6580),
(7,'Fail','Netbanking',2794);

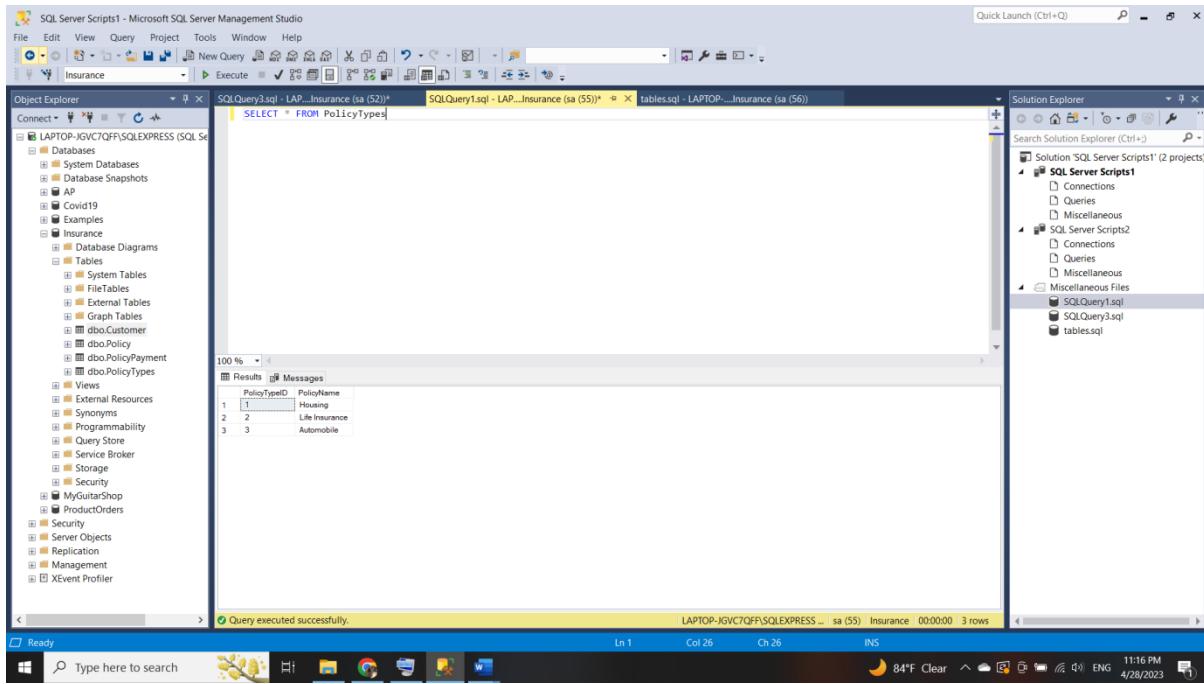
```

Table Contents:

`SELECT * FROM Customer`

CustomerId	FirstName	LastName	DOBDate	EmailId	PhoneNum	City	State	ZipCode
1	Praween	Sudha	1990-05-18	praween@sus.edu	407085034	Phoenix	Arizona	85220
2	Subarna	Deo	1999-05-25	ben@sus.edu	7665257878	Phoenix	Arizona	85220
3	Premra	Bhavani	1999-05-05	premra@sus.edu	242534278	Phoenix	Arizona	85231
4	Sumanth	Vennu	1999-05-07	sumanth@sus.edu	3405089575	Cerritos	California	90784
5	Sindhuri	Munjila	1997-12-12	sindhuri@sus.edu	7467312340	Scottsdale	Arizona	85265
6	Andrew	Josh	1995-06-18	andrew@sus.edu	9124765333	Los Angeles	California	90001
7	Sushmita	Yadav	1979-01-13	sushmita@sus.edu	2349875645	Georgia	Atlanta	30304

SELECT * FROM PolicyTypes

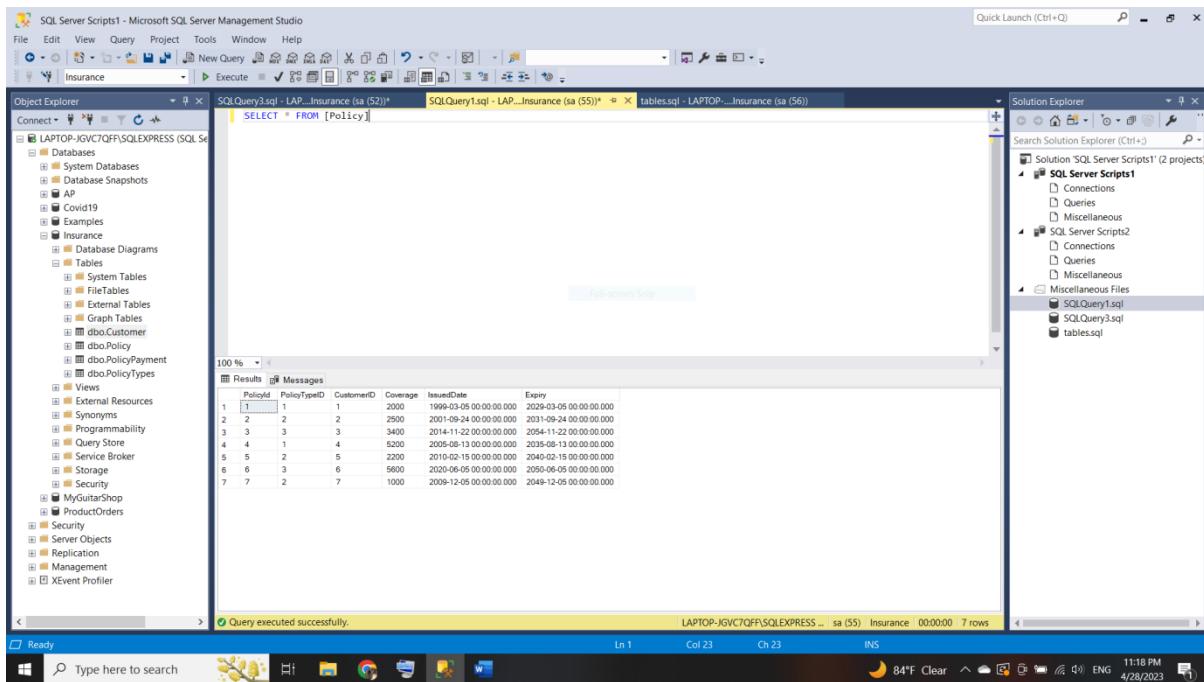


The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows a database named 'Insurance' with various objects like 'Tables', 'Views', and 'Procedures'. The Solution Explorer on the right shows a solution named 'SQL Server Scripts1' with two projects: 'SQL Server Scripts1' and 'SQL Server Scripts2', each containing 'Connections', 'Queries', and 'Miscellaneous' files. The central Results pane displays the output of the query 'SELECT * FROM PolicyTypes'. The results are as follows:

PolicyTypeID	PolicyName
1	Housing
2	Life Insurance
3	Automobile

Below the results, a message box says 'Query executed successfully.' The status bar at the bottom shows 'LAPTOP-JGVC7QFF\SQLEXPRESS ... | sa (55) | Insurance | 00:00:00 | 3 rows'.

SELECT * FROM [Policy]



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows a database named 'Insurance' with various objects like 'Tables', 'Views', and 'Procedures'. The Solution Explorer on the right shows a solution named 'SQL Server Scripts1' with two projects: 'SQL Server Scripts1' and 'SQL Server Scripts2', each containing 'Connections', 'Queries', and 'Miscellaneous' files. The central Results pane displays the output of the query 'SELECT * FROM [Policy]'. The results are as follows:

PolicyID	PolicyTypeID	CustomerID	Coverage	IssuedDate	Expiry
1	1	1	2000	1999-03-05 00:00:00.000	2029-03-05 00:00:00.000
2	2	2	2500	2001-09-24 00:00:00.000	2031-09-24 00:00:00.000
3	3	3	3400	2014-11-22 00:00:00.000	2054-11-22 00:00:00.000
4	4	1	5200	2005-08-13 00:00:00.000	2035-08-13 00:00:00.000
5	5	2	5	2200	2010-02-15 00:00:00.000
6	6	3	6	5600	2020-06-05 00:00:00.000
7	7	2	7	1000	2009-12-05 00:00:00.000

Below the results, a message box says 'Query executed successfully.' The status bar at the bottom shows 'LAPTOP-JGVC7QFF\SQLEXPRESS ... | sa (55) | Insurance | 00:00:00 | 7 rows'.

`SELECT * FROM [PolicyPayment]`

The screenshot shows the Microsoft SQL Server Management Studio (SSMS) interface. The title bar reads "SQL Server Scripts1 - Microsoft SQL Server Management Studio". The main window displays a query results grid for the "PolicyPayment" table. The table has columns: PaymentId, PolicyId, PaymentStatus, PaymentType, and TransactionId. The data shows 7 rows of payment records. The bottom status bar indicates the query was executed successfully and shows the connection details: "LAPTOP-JGV7QFF\SQLEXPRESS ... | sa (55) | Insurance | 00:00:00 | 7 rows".

PaymentId	PolicyId	PaymentStatus	PaymentType	TransactionId
1	3	Success	Cash	7089
2	2	Success	CreditDebit	2739
3	5	Success	Netbanking	1935
4	6	Success	Cash	4792
5	7	Fail	CreditDebit	6500
6	8	Success	CreditDebit	2794
7	9	Fail	Netbanking	2794

Answers for Questions

1. How many policies were issued in a particular time period?

```
SELECT COUNT(*) AS 'Number of Policies' FROM Policy WHERE IssuedDate BETWEEN '2004-04-21' AND '2023-04-26';
```

SQLQuery1.sql - LAPTOP-JGVC7QF\SQLEXPRESS.Insurance (sa (94)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

SQLQuery1.sql - LAP..._Insurance (sa (94))

SELECT COUNT(*) AS 'Number of Policies' FROM Policy WHERE IssuedDate BETWEEN '2004-04-21' AND '2023-04-26';

Object Explorer

Connect > LAPTOP-JGVC7QF\SQLEXPRESS (SQL Server 16.0.1050 - sa)

Insurance

Tables

Views

Security

Server Objects

Replication

Management

XEvent Profiler

Results

Messages

Number of Policies

10

Query executed successfully.

LAPTOP-JGVC7QF\SQLEXPRESS - sa (94) Insurance 00:00:00 1 rows

Ready

121 %

1 row

Col 68

Ch 68

INS

96°F Sunny 4/29/2023

2) How many Active Policies are there in the system?

A)

```
select count(*) as Active from Policy p
inner join PolicyPayment pp on p.PolicyId=pp.PolicyID where pp.PaymentStatus!='Fail'
and p.Expiry >=GETDATE()
```

3) Which Customer has the Maximum coverage amount?

A)

```
DECLARE @Max int=(Select MAX(Coverage) from policy)
```

```
Select C.CustomerID,C.Firstname,C.LastName,p.Coverage as [Maximum Coverage] from policy p INNER JOIN Customer c on c.CustomerId=p.CustomerID where Coverage=@Max
```

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left displays the database structure, including tables like 'Customer' and 'Policy', and files like 'SQLQuery9.sql'. The central pane shows a query window with the following T-SQL code:

```
DECLARE @Max int=(Select MAX(Coverage) from policy)

Select C.CustomerID,C.Firstname,C.LastName,p.Coverage as [Maximum Coverage] from policy p
INNER JOIN Customer c on c.CustomerId=p.CustomerID where Coverage=@Max
```

The results pane below shows the output of the query:

	CustomerID	Firstname	LastName	Maximum Coverage
1	6	Andrew	Josh	5600
2	2	Subadra Devi	Venkatesan	5600

The status bar at the bottom indicates the query was executed successfully on a local instance of SQL Server.

4) Find out the Customer details who has paid their policy payment through Netbanking also display the Policy coverage amount?

A)

```
SELECT c.CustomerID,c.Firstname,c.LastName,c.EmailId,p.Coverage from Customer c
INNER JOIN Policy p ON c.CustomerId=p.CustomerID
INNER JOIN PolicyPayment pp on p.PolicyId=pp.PolicyID
where pp.PaymentType='Netbanking' AND PaymentStatus='Success'
```

SQLQuery1.sql - LAPTOP-JGVC7QF\SQLEXPRESS.Insurance (sa (94)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

SQLQuery3.sql - LAP...Insurance (sa (62)) SQLQuery1.sql - LAP...Insurance (sa (94))

```
SELECT c.CustomerID,c.Firstname,c.LastName,c.EmailId,p.Coverage from Customer c
INNER JOIN Policy p ON c.CustomerId=p.CustomerID
INNER JOIN PolicyPayment pp on p.PolicyId=pp.PolicyID
where pp.PaymentType='Netbanking' AND PaymentStatus='Success'
```

Results Messages

CustomerID	Firstname	LastName	EmailId	Coverage
3	Prema Bhaavani	Doddanari	prema@asu.edu	3400

Query executed successfully.

LN 6 Col 9 Ch 9 INS

Ready Type here to search 96°F Sunny 2:53 PM 4/29/2023

1) Is there any Stored procedure to retrieve Customer Policy details?

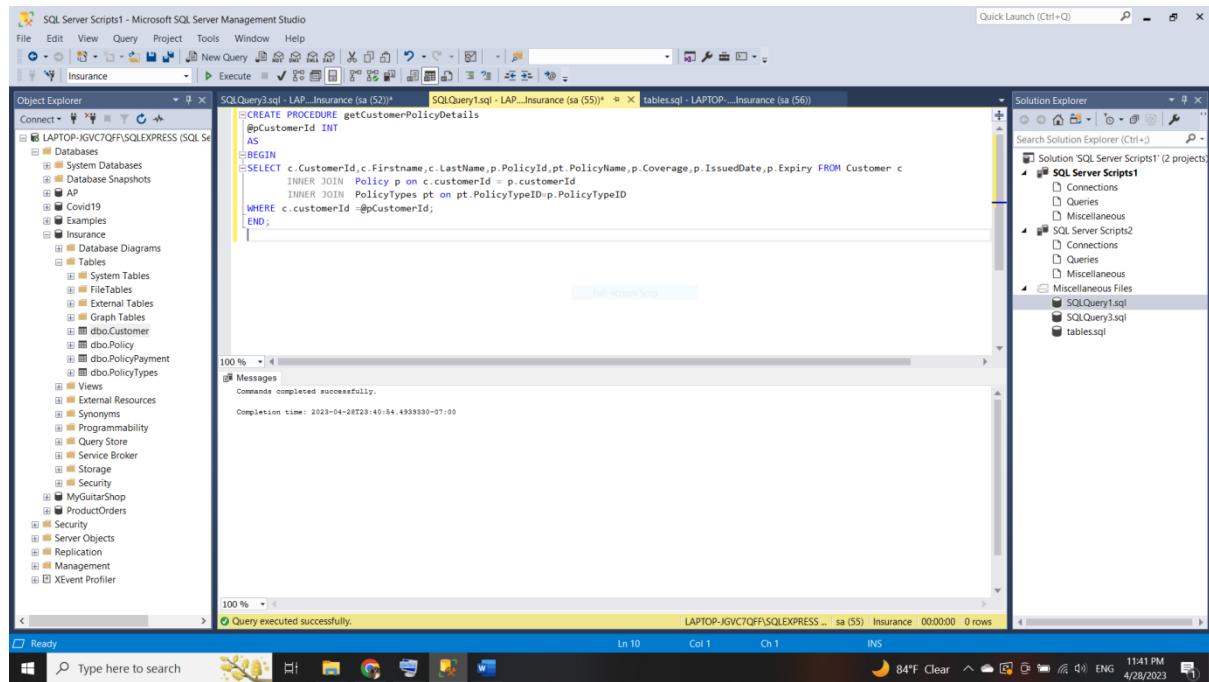
A)

```
CREATE PROCEDURE getCustomerPolicyDetails
    @pCustomerId INT
    AS
    BEGIN
        SELECT
            c.CustomerId,c.Firstname,c.LastName,p.PolicyId,pt.PolicyName,p.Coverage,p.IssuedDate,p.Expiry
        FROM Customer c
        INNER JOIN Policy p on c.customerId = p.customerId
```

```

    INNER JOIN PolicyTypes pt on pt.PolicyTypeID=p.PolicyTypeID
    WHERE c.customerId =@pCustomerId;
  END;

```

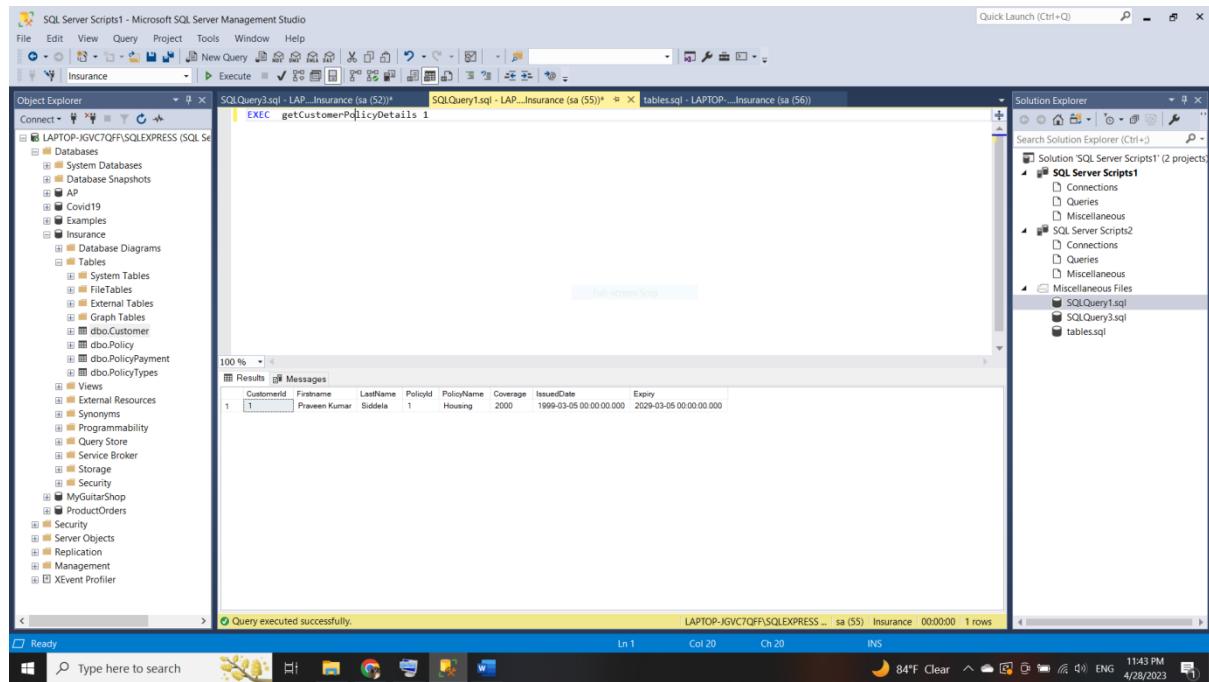


```

CREATE PROCEDURE getCustomerPolicyDetails
    @CustomerId INT
AS
BEGIN
    SELECT c.CustomerId,c.Ffirstname,c.Lastname,p.PolicyId,p.PolicyName,p.Coverage,p.IssuedDate,p.Expiry
    FROM Customer c
    INNER JOIN Policy p on c.customerId = p.customerId
    INNER JOIN PolicyTypes pt on pt.PolicyTypeID=p.PolicyTypeID
    WHERE c.customerId =@pCustomerId;
END;

```

EXEC getCustomerPolicyDetails 1



```

EXEC getCustomerPolicyDetails 1

```

CustomerId	Ffirstname	Lname	PolicyId	PolicyName	Coverage	IssuedDate	Expiry
1	Praween	Kumar	1	Housing	2000	1999-03-05 00:00:00	2029-03-05 00:00:00

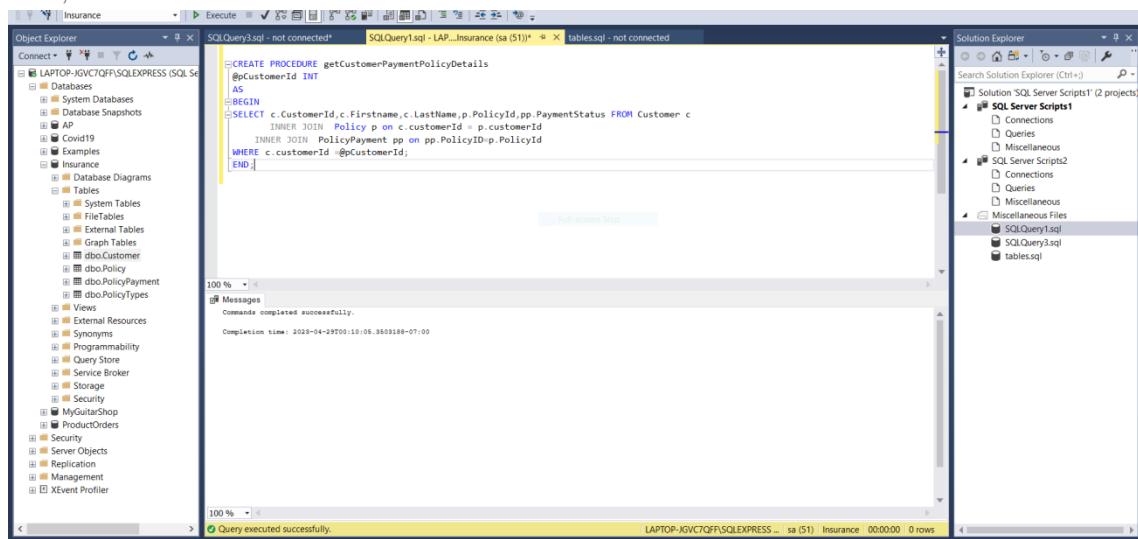
2) Is there any Stored procedure to retrieve Customer Payment Policy details?

A) CREATE PROCEDURE getCustomerPaymentPolicyDetails

```

@pCustomerId INT
AS
BEGIN
SELECT c.CustomerId,c.Firstname,c.LastName,p.PolicyId,pp.PaymentStatus FROM Customer c
INNER JOIN Policy p on c.customerId = p.customerId
INNER JOIN PolicyPayment pp on pp.PolicyID=p.PolicyId
WHERE c.customerId =@pCustomerId;
END;

```

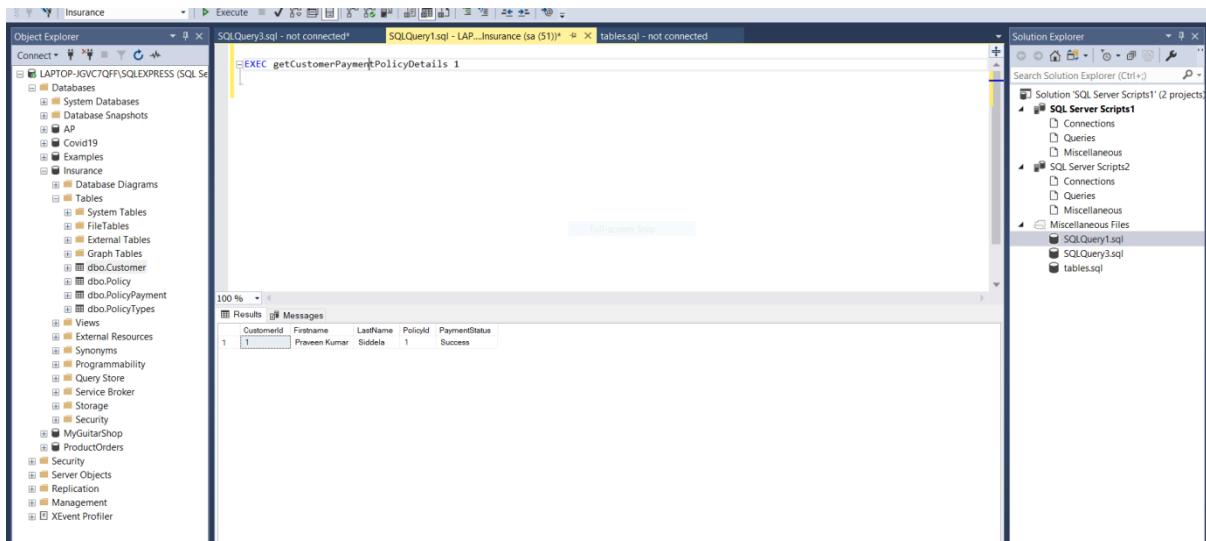


```

CREATE PROCEDURE getCustomerPaymentPolicyDetails
@pCustomerId INT
AS
BEGIN
SELECT c.CustomerId,c.Firstname,c.LastName,p.PolicyId,pp.PaymentStatus FROM Customer c
INNER JOIN Policy p on c.customerId = p.customerId
INNER JOIN PolicyPayment pp on pp.PolicyID=p.PolicyId
WHERE c.customerId =@pCustomerId;
END;

```

EXEC getCustomerPaymentPolicyDetails 1



3) Is there any trigger that notify Customer if a Policy Payment fails?

A)

```

CREATE TRIGGER CustomerPaymentFailureNotify_trigger
ON PolicyPayment
AFTER INSERT
AS
BEGIN

DECLARE @CustomerID INT
DECLARE @PolicyID INT
DECLARE @Logtime DATETIME
DECLARE @Message varchar(100)
DECLARE @Status varchar(20)

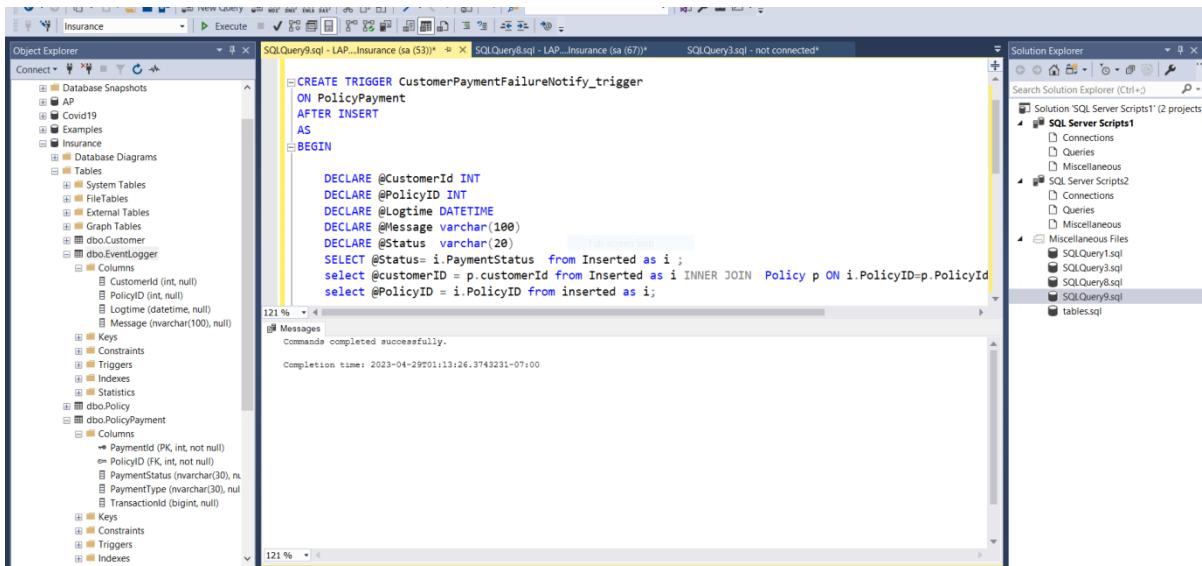
SELECT @Status= i.PaymentStatus from Inserted as i ;
select @customerID = p.customerID from Inserted as i INNER JOIN Policy p ON i.PolicyID=p.PolicyId;
select @PolicyID = i.PolicyID from inserted as i;

IF @Status='Fail'
BEGIN
INSERT INTO EventLogger
VALUES (@CustomerID, @PolicyID, GETDATE(),'Payment Failed');
END

```

```
PRINT 'Customer Payment Failure Notify Trigger Fired'
```

```
END;
```



```
CREATE TRIGGER CustomerPaymentFailureNotify_trigger
ON PolicyPayment
AFTER INSERT
AS
BEGIN

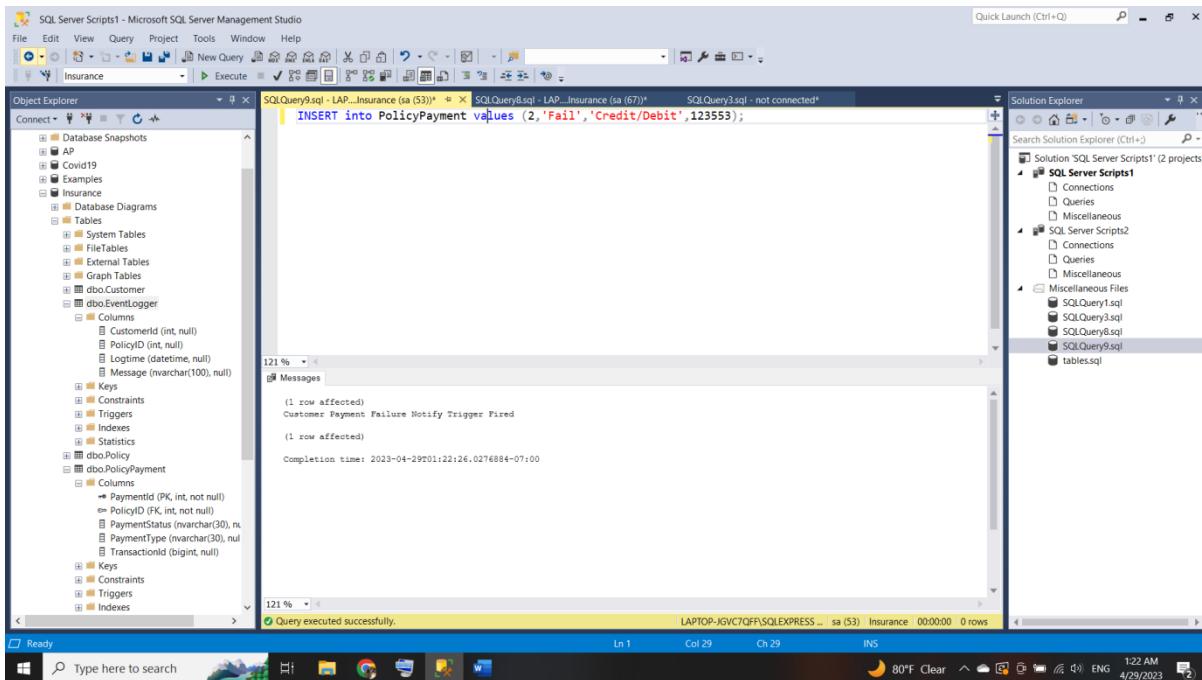
    DECLARE @CustomerId INT
    DECLARE @PolicyID INT
    DECLARE @Logtime DATETIME
    DECLARE @Message varchar(100)
    DECLARE @Status varchar(20)

    SELECT @status= i.PaymentStatus  from Inserted as i ;
    select @CustomerID = p.customerId from Inserted as i INNER JOIN Policy p ON i.PolicyID=p.PolicyId
    select @PolicyID = i.PolicyID from inserted as i;

END
```

Customer Payment Failure Notify trigger will fires whenever there is an Failure insert happens into Policy Payment table, So inserting a record into Policy Payment table to check the trigger.

```
INSERT into PolicyPayment values (2,'Fail','Credit/Debit',123553);
```



```
INSERT into PolicyPayment values (2,'Fail','Credit/Debit',123553);
```

Checking EventLogger Table

```
SELECT * FROM EventLogger
```

SQL Server Scripts1 - Microsoft SQL Server Management Studio

Object Explorer

File Edit View Query Project Tools Window Help

SQLQuery9.sql - LAP...Insurance (sa (53)) SQLQuery8.sql - LAP...Insurance (sa (67)) SQLQuery2.sql - not connected

SELECT * FROM EventLogger

Results Messages

CustomerID	PolicyID	Logtime	Message
2	2	2023-04-29 01:22:25.977	Payment Failed

Query executed successfully.

LAPTOP-JGVC7QFF\SQLEXPRESS sa (53) Insurance 00:00:00 1 rows

Ready Type here to search

4) Is there any trigger that notifies the Customer policies which ends in a week?

```
CREATE TRIGGER CustomerPolicyDeadlineNotify_trigger
ON [Policy]
AFTER INSERT
AS
BEGIN
    DECLARE @CustomerId INT
    DECLARE @PolicyID INT
    DECLARE @Firstname nvarchar(20)
    DECLARE @LastName nvarchar(20)
    DECLARE @Coverage BIGINT
    DECLARE @IssuedDate DATETIME
    DECLARE @Expiry DATETIME
    DECLARE @Logtime DATETIME
```

```

DECLARE @Message varchar(100)

INSERT INTO
CustomerPolicyDeadline(CustomerId,Firstname,LastName,PolicyID,Coverage,IssuedDate,Expiry,Logt
me,[Message])

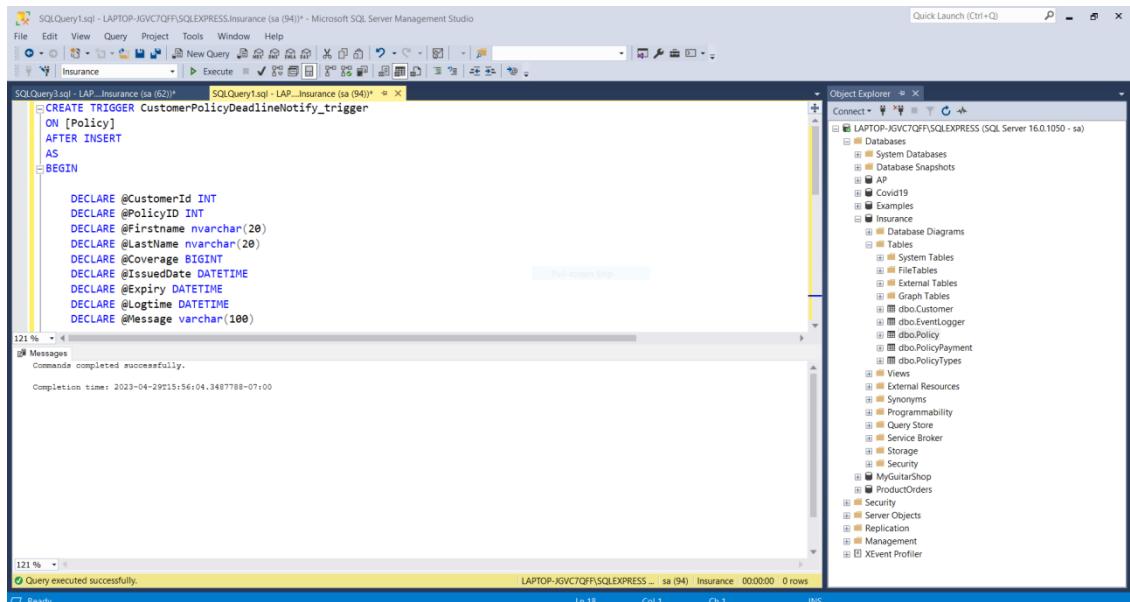
SELECT
p.CustomerID,c.Firstname,c.LastName,p.PolicyId,p.Coverage,p.IssuedDate,p.Expiry,GETDATE(),'Poli
cy expires in 7 days'

from Policy p INNER JOIN Customer c ON c.CustomerID=p.CustomerID
WHERE p.Expiry=CONVERT(DATE,DATEADD(DAY,7,GETDATE()));

PRINT 'Customer Policy Deadline Notify Trigger Fired'

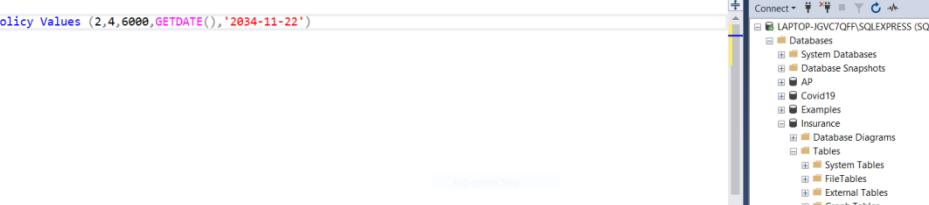
END;

```



Customer policy Deadline Notify trigger will fires whenever there is an insert happens into Policy table it checks all the policies expiration dates and it get triggered if an Expiry date of policy is in 7 days. So, inserting a record into Policy table to check the trigger.

```
INSERT INTO Policy Values (2,4,6000,GETDATE(),'2034-11-22')
```



SQLQuery3.sql - LAP...Insurance (sa (63))
SQLQuery3.sql - LAP...Insurance (sa (62))
SQLQuery1.sql - LAP...Insurance (sa (94))

```
INSERT INTO Policy_Values (2, 4, 6000, GETDATE(), '2034-11-22')
```

121 % 121 %

Messages

```
(1 row affected)
Customer Policy Deadline Notify Trigger Fired

(1 row affected)
```

Completion time: 2023-04-29T16:16:40.5535730-07:00

121 % 121 %

Query executed successfully.

LAPTOP-JGVC7QF\SQLEXPRESS ... sa (94) Insurance | 00:00:00 | 0 rows

Object Explorer

- Connect
- LAPTOP-JGVC7QF\SQLEXPRESS (SQL Server 16.0.1050 - sa)
 - Databases
 - System Databases
 - Database Snapshots
 - AP
 - Covid19
 - Examples
 - Insurance
 - Database Diagrams
 - Tables
 - System Tables
 - FileTables
 - External Tables
 - Graph Tables
 - dbo.Customer
 - dbo.EventLogger
 - dbo.Policy
 - dbo.PolicyPayment
 - dbo.PolicyTypes
 - Views
 - External Resources
 - Synonyms
 - Programmability
 - Query Store
 - Service Broker
 - Storage
 - Security
 - MyGuitarShop
 - ProductOrder
 - Security
 - Server Objects
 - Replication
 - Management
 - XEEvent Profiler

Checking CustomerPolicyDeadline Table

```
SELECT * FROM CustomerPolicyDeadline
```

The screenshot shows the SQL Server Management Studio (SSMS) interface. The Object Explorer on the right lists the database structure for 'LAPTOP-JGVC7QFLSQLEXPRESS'. The 'Tables' node is expanded, showing 'Customer', 'EventLogger', 'Policy', and 'PolicyTypes'. The 'Customer' table is selected. The 'Results' tab in the bottom-left shows a query result grid for the 'CustomerPolicyDeadline' table. The grid has columns: CustomerID, FirstName, LastName, PolicyID, Coverage, IssuedDate, Expir, Logtime, and Message. One row is displayed, showing a message: 'Policy expires in 7 days'.

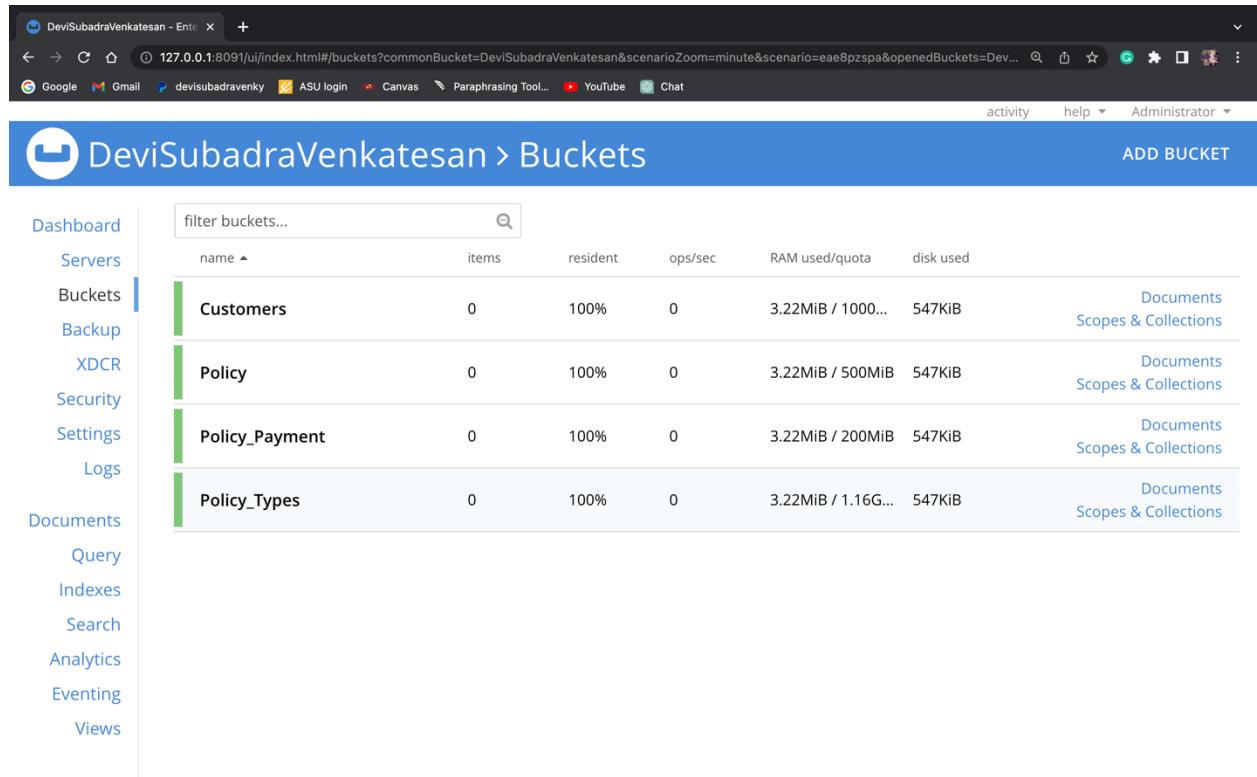
	CustomerID	FirstName	LastName	PolicyID	Coverage	IssuedDate	Expir	Logtime	Message
1	6	Andrew	Josh	6	5000	2020-06-05 00:00:00.000	2023-05-06 00:00:00.000	2023-04-29 16:14:17.503	Policy expires in 7 days

NoSQL – Couchbase

We have created four buckets as mentioned below:

- ## 1. Customer

2. Policy
3. Policy Types
4. Policy Payment



DeviSubadraVenkatesan - Ente +

127.0.0.1:8091/ui/index.html#/buckets?commonBucket=DeviSubadraVenkatesan&scenarioZoom=minute&scenario=eae8pzspa&openedBuckets=Dev...

Google Gmail devisubadravenky ASU login Canvas Paraphrasing Tool... YouTube Chat

activity help Administrator

DeviSubadraVenkatesan > Buckets ADD BUCKET

filter buckets...						
name	items	resident	ops/sec	RAM used/quota	disk used	
Customers	0	100%	0	3.22MiB / 1000MiB	547KiB	Documents Scopes & Collections
Policy	0	100%	0	3.22MiB / 500MiB	547KiB	Documents Scopes & Collections
Policy_Payment	0	100%	0	3.22MiB / 200MiB	547KiB	Documents Scopes & Collections
Policy_Types	0	100%	0	3.22MiB / 1.16GiB	547KiB	Documents Scopes & Collections

Dashboard

Servers

Buckets selected

Backup

XDCR

Security

Settings

Logs

Documents

Query

Indexes

Search

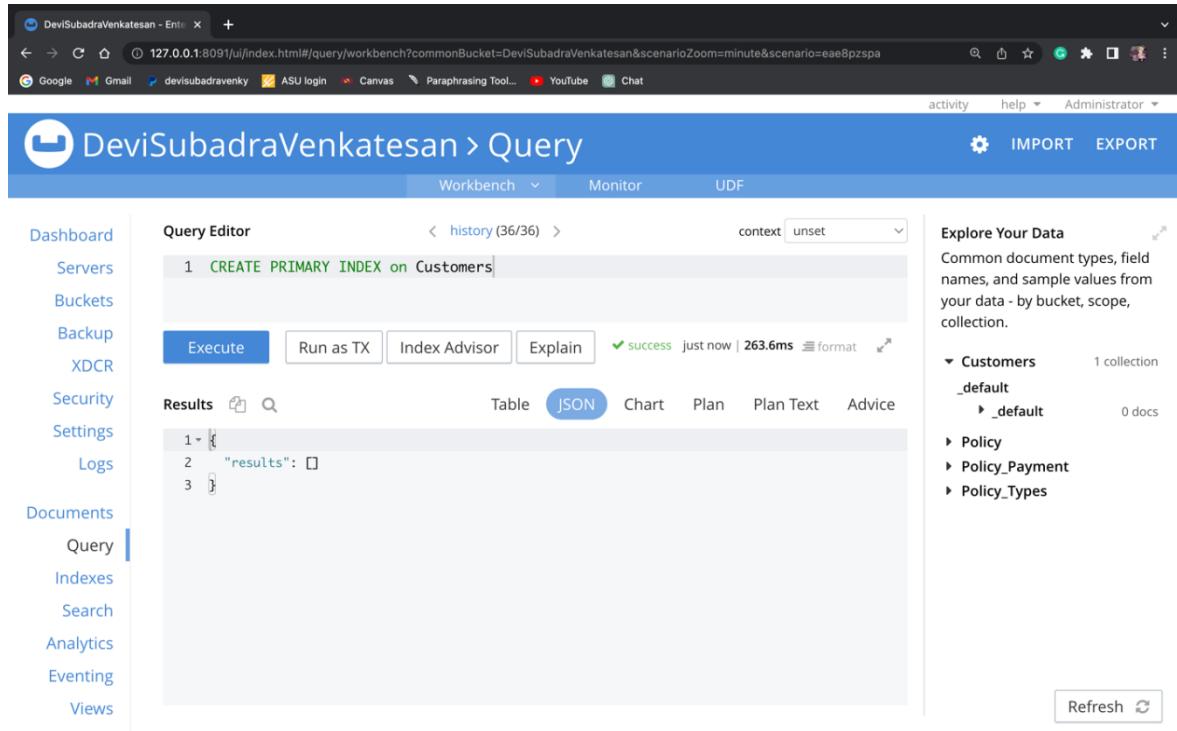
Analytics

Eventing

Views

Creating Primary Indexes:

Query: CREATE PRIMARY INDEX on Customer



The screenshot shows the Apache CouchDB Query Workbench interface. The left sidebar is a navigation menu with the 'Query' option selected. The main area is the 'Query Editor' with the following query:

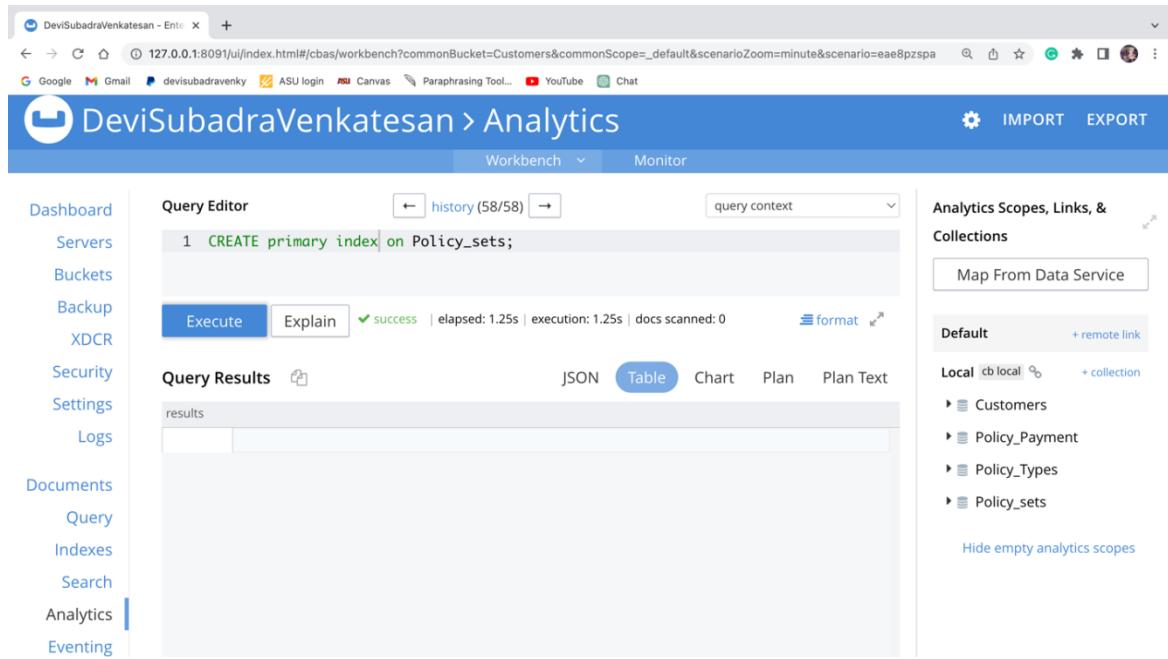
```
1 CREATE PRIMARY INDEX on Customers;
```

The 'Results' tab is selected, showing the response in JSON format:

```
1 {
2   "results": []
3 }
```

On the right, the 'Explore Your Data' section shows the 'Customers' collection with one document and no results. The 'Refresh' button is visible at the bottom right of the results area.

Query: CREATE PRIMARY INDEX on Policy_sets



The screenshot shows the Apache CouchDB Analytics Workbench interface. The left sidebar is a navigation menu with the 'Analytics' option selected. The main area is the 'Query Editor' with the following query:

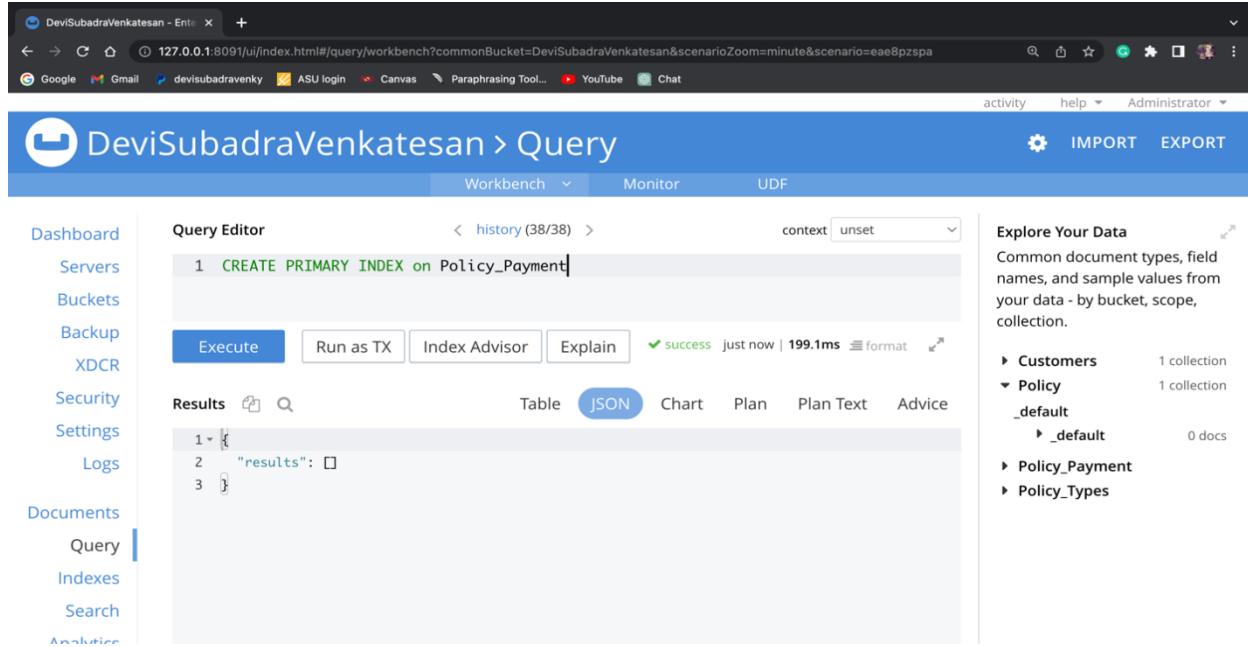
```
1 CREATE primary index on Policy_sets;
```

The 'Query Results' tab is selected, showing the response:

```
results
```

On the right, the 'Analytics Scopes, Links, & Collections' section shows the 'Default' scope with a 'Local' collection named 'Policy_sets'. The 'Hide empty analytics scopes' button is visible at the bottom right.

Query: CREATE PRIMARY INDEX on Policy_Payment



The screenshot shows the Apache CouchDB Query interface. The URL is `127.0.0.1:8091/ui/index.html#/query/workbench?commonBucket=DeviSubadraVenkatesan&scenarioZoom=minute&scenario=eae8pzspa`. The left sidebar is the navigation menu. The main area is the Query Editor with the following content:

```
1 CREATE PRIMARY INDEX on Policy_Payment
```

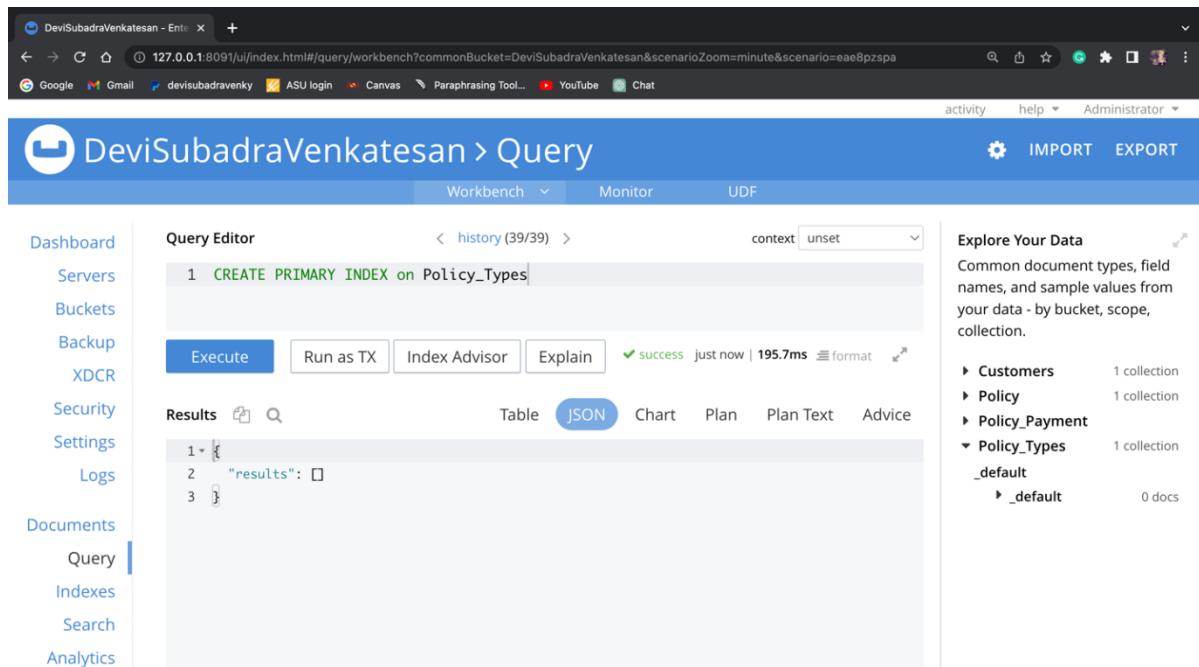
Below the code, there are buttons: Execute, Run as TX, Index Advisor, Explain. The status is success just now | 199.1ms format. The results are displayed in JSON format:

```
1 {
2   "results": []
3 }
```

On the right, the "Explore Your Data" panel shows the following collections and their details:

- Customers: 1 collection
- Policy:
 - _default: 1 collection
 - _default: 0 docs
- Policy_Payment: 1 collection
- Policy_Types: 1 collection

Query: CREATE PRIMARY INDEX on Policy_Types



The screenshot shows the Apache CouchDB Query interface. The URL is `127.0.0.1:8091/ui/index.html#/query/workbench?commonBucket=DeviSubadraVenkatesan&scenarioZoom=minute&scenario=eae8pzspa`. The left sidebar is the navigation menu. The main area is the Query Editor with the following content:

```
1 CREATE PRIMARY INDEX on Policy_Types
```

Below the code, there are buttons: Execute, Run as TX, Index Advisor, Explain. The status is success just now | 195.7ms format. The results are displayed in JSON format:

```
1 {
2   "results": []
3 }
```

On the right, the "Explore Your Data" panel shows the following collections and their details:

- Customers: 1 collection
- Policy: 1 collection
- Policy_Payment: 1 collection
- Policy_Types: 1 collection
 - _default: 1 collection
 - _default: 0 docs

JSON Documents

Customer Bucket:

```

insert into customers(key,value)
values("1",    {
  "customerId": "001",
  "customerName": "Devi",
  "phoneNumber": "123-456-7890"
}),("2",{
  "customerId": "002",
  "customerName": "Praveen",
  "phoneNumber": "555-555-1212"
}),("3",{
  "customerId": "003",
  "customerName": "Prema",
  "phoneNumber": "888-123-4567"
}),("4",{
  "customerId": "004",
  "customerName": "Sindhu",
  "phoneNumber": "555-123-4567"
}),("5",{
  "customerId": "005",
  "customerName": "Sumanth",
  "phoneNumber": "777-555-1234"
}),("6",{
  "customerId": "006",
  "customerName": "Andrew",
  "phoneNumber": "123-456-7890"
}),("7",{

```

```

"customerId": "007",
"customerName": "Sushmita",
"phoneNumber": "123-456-7890"
})

```

Policy_sets Bucket:

insert into Policy_sets (key,value)

```

values("1", {
"customerId": "001",
"Amount": "2000",
"Issued date": "03/05/1999",
"Expiry date": "03/05/2029"
}),("2", {
"customerId": "002",
"Amount": 2500,
"Issued date": "09/24/2001",
"Expiry date": "09/24/2031"
}),("3", {
"customerId": "003",
"Amount": "3400",
"Issued date": "11/22/2014",
"Expiry date": "11/22/2054"
}),("4", {
"customerId": "004",
"Amount": "5200",
"Issued date": "08/13/2005",
"Expiry date": "08/13/2035"
}),("5", {

```

```

"customerId": "005",
"Amount": "2200",
"Issued date": "02/15/2010",
"Expiry date": "02/15/2040"
}), ("6", {
"customerId": "006",
"Amount": "1000",
"Issued date": "03/05/2011",
"Expiry date": "07/22/2030"
}), ("7", {
"customerId": "007",
"Amount": "1050",
"Issued date": "03/12/2003",
"Expiry date": "09/07/2030"
})

```

Policy_Payment Bucket:

```

insert into Policy_Payment (key,value)
values("1", {
"id": 1,
"policy_id": 123,
"amount": 500,
"method": "Cash",
"Transaction id": 9127,
"payment_date": "2022-04-15",
"status": "Success"
}), ("2", {
"id": 2,

```

```
"policy_id": 456,  
"amount": 250,  
"method": "Credit/Debit",  
"Transaction id": 7389,  
"payment_date": "2022-05-01",  
"status": "Success"  
}),("3",{  
"id": 3,  
"policy_id": 789,  
"amount": 750,  
"method": "NetBanking",  
"Transaction id": 2739,  
"payment_date": "2022-06-15",  
"status": "Success"  
}),("4",{  
"id": 4,  
"policy_id": 234,  
"amount": 1000,  
"method": "Cash",  
"Transaction id": 1935,  
"payment_date": "2022-07-01",  
"status": "Success"  
}),("5",{  
"id": 5,  
"policy_id": 500,  
"amount": 1055,  
"method": "Credit/Debit",  
"Transaction id": 4792,
```

```

"payment_date": "2022-08-11",
"status": "Fail"

}),("6",{
"id": 6,
"policy_id": 340,
"amount": 1055,
"method": "Credit/Debit",
"Transaction id": 6500,
"payment_date": "2022-08-11",
"status": "Success"

}),("7",{
"id": 7,
"policy_id": 110,
"amount": 1055,
"method": "NetBanking",
"Transaction id": 2794,
"payment_date": "2022-08-11",
"status": "Fail"
})

```

Policy_Types Bucket:

```

insert into Policy_Types (key,value)
values("O1",
{
"id": 1,
"Type": "Housing"

}),("O2",{
"id": 2,

```

```

  "Type": "Automobile"
}

),("O3", {
  "id": 2,
  "Type": "Automobile"
}

```

Documents

DeviSubadraVenkatesan - Ente

127.0.0.1:8091/uj/index.html#/docs/editor?commonBucket=Customers&commonScope=_default&scenarioZoom=minute&scenario=ea8pzspa

Google Gmail devisubadraVenkatesan ASU login Canvas Paraphrasing Tool... YouTube Chat

DeviSubadraVenkatesan > Documents ADD DOCUMENT

Workbench Import

Keyspace bucket.scope.collection: Customers _default _default 10 0 optional... optional...

Limit 10 Offset 0 Document ID show range N1QL WHERE

7 Results for select meta().id from `Customers`._default._default data order by meta().id

enable field editing < prev batch | next batch >

	id	Document Content
	1	{"customerId": "001", "customerName": "Devi", "phoneNumber": "123-456-7890"}
	2	{"customerId": "002", "customerName": "Praveen", "phoneNumber": "555-555-5555"}
	3	{"customerId": "003", "customerName": "Prema", "phoneNumber": "888-123-4567"}
	4	{"customerId": "004", "customerName": "Sindhu", "phoneNumber": "555-123-4567"}
	5	{"customerId": "005", "customerName": "Sumanth", "phoneNumber": "777-555-5555"}
	6	{"customerId": "006", "customerName": "Andrew", "phoneNumber": "123-456-7890"}
	7	{"customerId": "007", "customerName": "Sushmita", "phoneNumber": "123-456-7890"}

DeviSubadraVenkatesan - Ente + 127.0.0.1:8091/ui/index.html#/docs/editor?commonBucket=Customers&commonScope=_default&scenarioZoom=minute&scenario=eae8pzspa

Google Gmail ASU login ASU Canvas Paraphrasing Tool... YouTube Chat

activity help Administrator

DeviSubadraVenkatesan > Documents

ADD DOCUMENT

Workbench Import

Dashboard Servers Buckets Backup XDCR Security Settings Logs Documents Query Indexes Search Analytics Eventing Views

Keyspace bucket.scope.collection: Policy_Paym... _default _default Limit 10 Offset 0 Document ID optional... show range N1QL WHERE optional...

7 Results for select meta().id from `Policy_Payment` .`_default` .`_default` data order by meta().id limit 10 offset 0

enable field editing < prev batch | next batch >

	id	
	1	{"id": 1, "policy_id": 123, "amount": 500, "method": "Cash", "Transaction id": 9127, "payment_date": "2022-04-15", "status": "Success"}
	2	{"id": 2, "policy_id": 456, "amount": 250, "method": "Credit/Debit", "Transaction id": 7389, "payment_date": "2022-05-01", "status": "Success"}
	3	{"id": 3, "policy_id": 789, "amount": 750, "method": "NetBanking", "Transaction id": 2739, "payment_date": "2022-06-15", "status": "Success"}
	4	{"id": 4, "policy_id": 234, "amount": 1000, "method": "Cash", "Transaction id": 1935, "payment_date": "2022-07-01", "status": "Success"}
	5	{"id": 5, "policy_id": 500, "amount": 1055, "method": "Credit/Debit", "Transaction id": 4792, "payment_date": "2022-08-11", "status": "Fail"}
	6	{"id": 6, "policy_id": 340, "amount": 1055, "method": "Credit/Debit", "Transaction id": 6500, "payment_date": "2022-08-11", "status": "Success"}
	7	{"id": 7, "policy_id": 110, "amount": 1055, "method": "NetBanking", "Transaction id": 2794, "payment_date": "2022-08-11", "status": "Fail"}

DeviSubadraVenkatesan - Ente + 127.0.0.1:8091/ui/index.html#/docs/editor?commonBucket=Customers&commonScope=_default&scenarioZoom=minute&scenario=eae8pzspa

Google Gmail ASU login ASU Canvas Paraphrasing Tool... YouTube Chat

activity help Administrator

DeviSubadraVenkatesan > Documents

ADD DOCUMENT

Workbench Import

Dashboard Servers Buckets Backup XDCR Security Settings Logs Documents Query Indexes Search Analytics Eventing Views

Keyspace bucket.scope.collection: Policy_Types _default _default Limit 10 Offset 0 Document ID optional... show range N1QL WHERE optional...

3 Results for select meta().id from `Policy_Types` .`_default` .`_default` data order by meta().id limit 10 offset 0

enable field editing < prev batch | next batch >

	id	
	1	{"id": 1, "Type": "Housing"}
	2	{"id": 2, "Type": "Automobile"}
	3	{"id": 2, "Type": "Automobile"}

DeviSubadraVenkatesan > Documents

Workbench Import

Keyspace bucket.scope.collection: Policy_sets._default._default

Limit 10 Offset 0 Document ID optional... no indexes available... Retrieve Docs

7 Results for Policy_sets._default._default, limit: 10, offset: 0

	id	Document
	1	{"Amount": "2000", "Expiry date": "03/05/2029", "Issued date": "03/05/1999", "customerId": "001"}
	2	{"Amount": "2500", "Expiry date": "09/24/2031", "Issued date": "09/24/2001", "customerId": "002"}
	3	{"Amount": "3400", "Expiry date": "11/22/2054", "Issued date": "11/22/2014", "customerId": "003"}
	4	{"Amount": "5200", "Expiry date": "08/13/2035", "Issued date": "08/13/2005", "customerId": "004"}
	5	{"Amount": "2200", "Expiry date": "02/15/2040", "Issued date": "02/15/2010", "customerId": "005"}
	6	{"Amount": "1000", "Expiry date": "07/22/2030", "Issued date": "03/05/2011", "customerId": "006"}
	7	{"Amount": "1050", "Expiry date": "09/07/2030", "Issued date": "03/12/2003", "customerId": "007"}

Query Services and Analytics Services

Creation of Datasets:

Query: CREATE DATASET on Customers

DeviSubadraVenkatesan > Analytics

Workbench Monitor

Query Editor

1 create dataset on Customers

Execute Explain ✓ success | elapsed: 392.21ms | execution: 386.08ms | docs scanned: 0 format ↗

Query Results

JSON Table Chart Plan Plan Text

Analytics Scopes, Links, & Collections

Map From Data Service

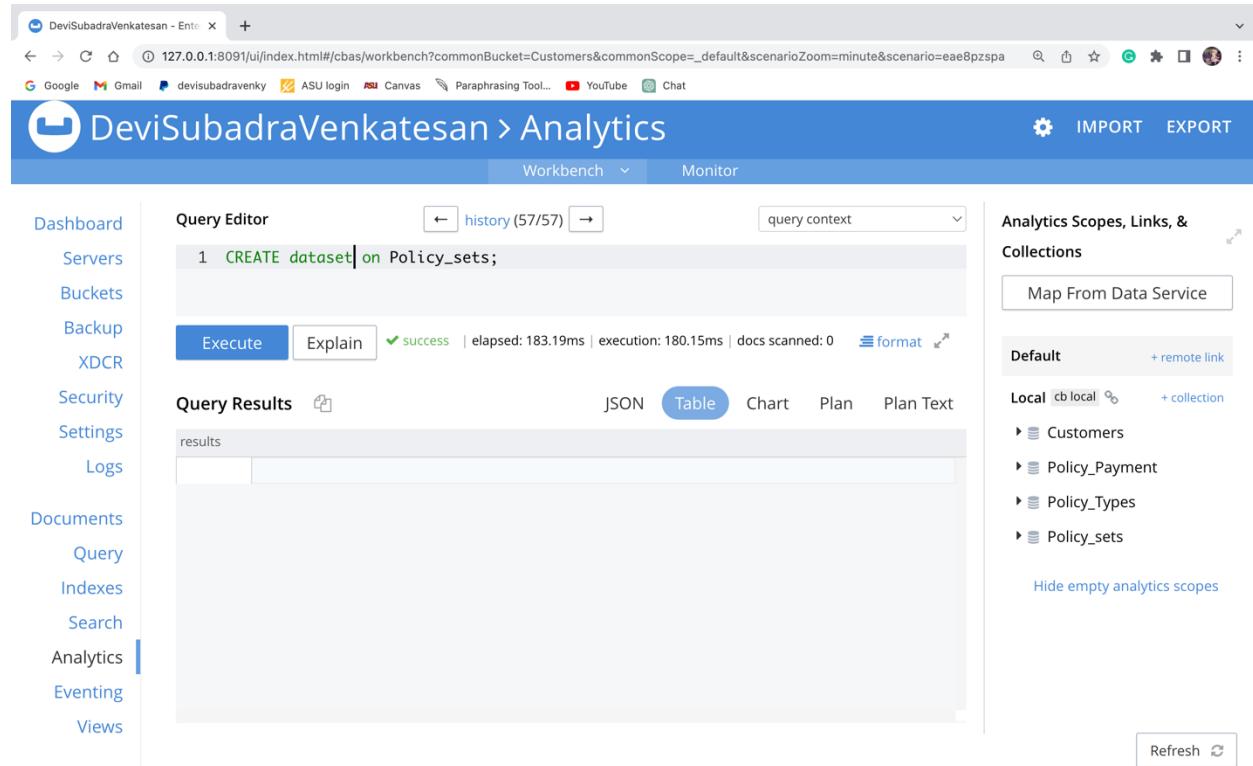
Default + remote link

Local cb local + collection

Customers

Hide empty analytics scopes

Query: CREATE DATASET on Policy_sets

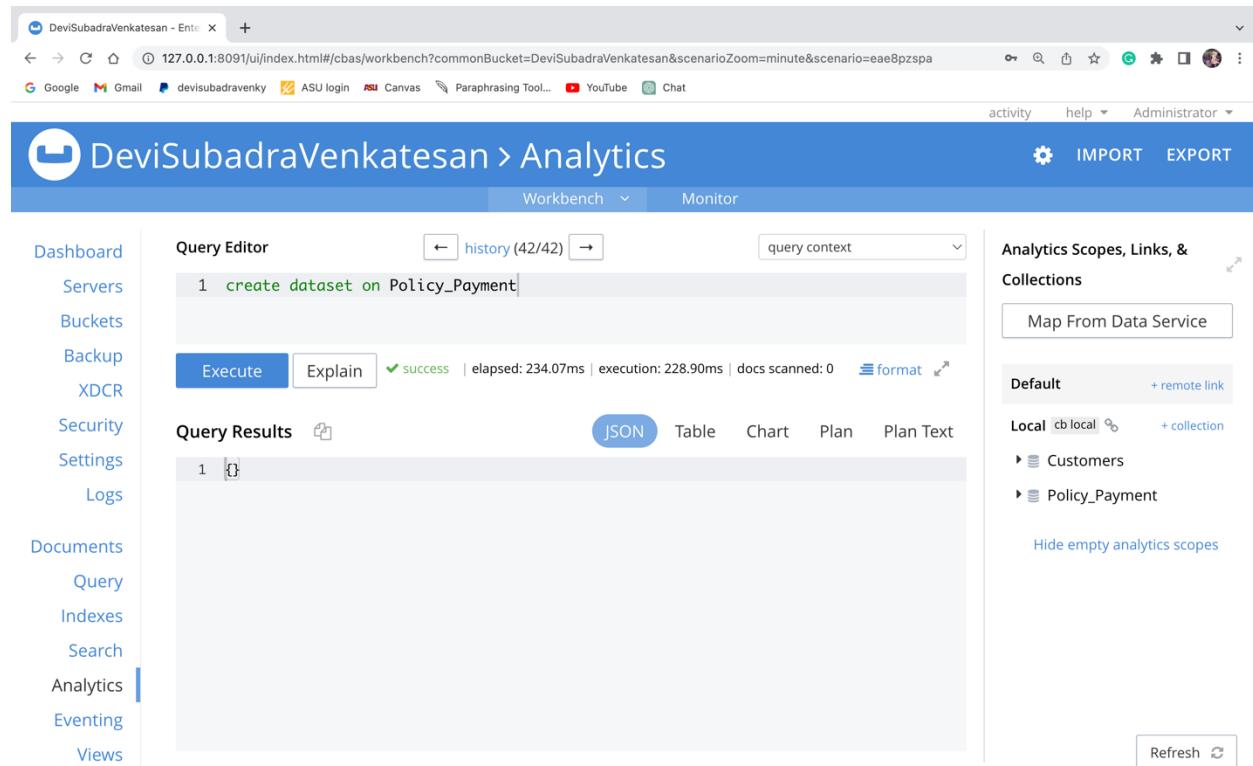


The screenshot shows the Apache CouchDB Analytics interface. The left sidebar is a navigation menu with the 'Analytics' option selected. The main area has a 'Query Editor' with the following query:

```
1 CREATE dataset on Policy_sets;
```

The 'Execute' button is highlighted. Below the editor, the 'Query Results' section shows a table with one row of data. The right sidebar, titled 'Analytics Scopes, Links, & Collections', shows a tree structure of local and remote collections, including 'Default', 'Local', and 'Policy_sets'.

Query: CREATE DATASET on Policy_Payment

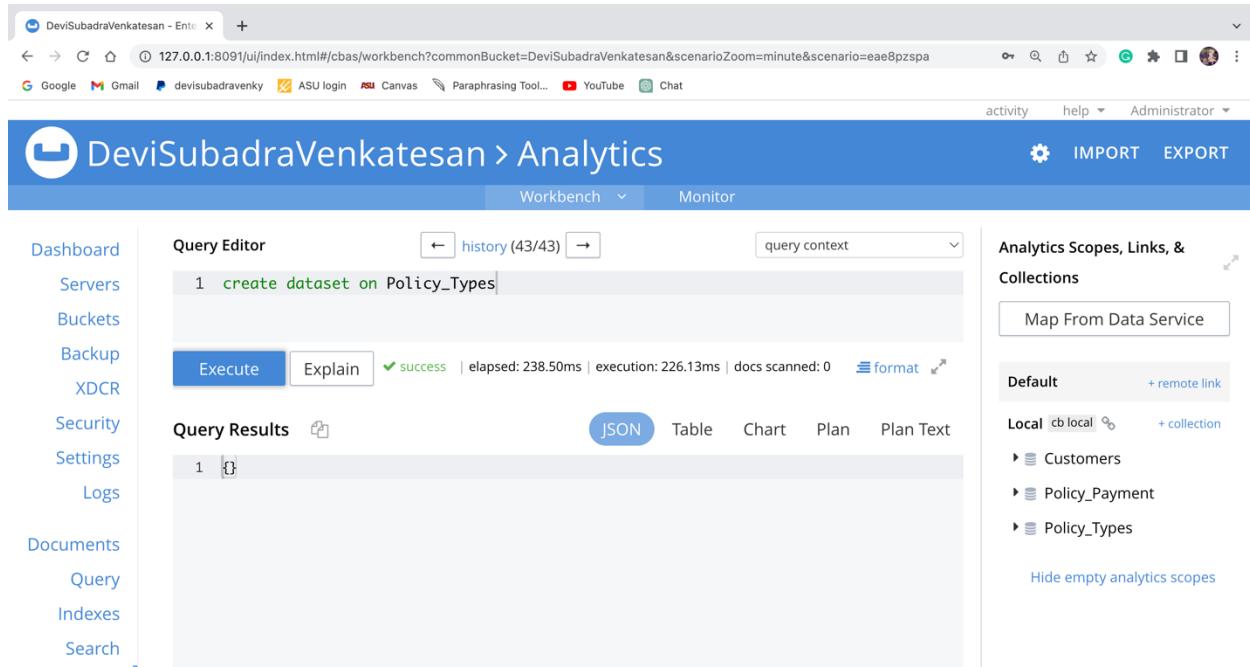


The screenshot shows the Apache CouchDB Analytics interface. The left sidebar is a navigation menu with the 'Analytics' option selected. The main area has a 'Query Editor' with the following query:

```
1 create dataset on Policy_Payment;
```

The 'Execute' button is highlighted. Below the editor, the 'Query Results' section shows a table with one row of data. The right sidebar, titled 'Analytics Scopes, Links, & Collections', shows a tree structure of local and remote collections, including 'Default', 'Local', and 'Policy_Payment'.

Query: CREATE DATASET on Policy_Types



The screenshot shows the DeviSubadraVenkatesan > Analytics interface. On the left, a sidebar lists various navigation options: Dashboard, Servers, Buckets, Backup, XDCR, Security, Settings, Logs, Documents, Query, Indexes, and Search. The main area is the Query Editor, which displays the query '1 create dataset on Policy_Types'. Below the editor, the Query Results section shows a single row with the value '1 [{}]' in JSON format. To the right, the Analytics Scopes, Links, & Collections panel shows the 'Default' scope with 'Local' collections: 'Customers', 'Policy_Payment', and 'Policy_Types'. A 'Map From Data Service' button is also present in this panel.

Queries in Couchbase

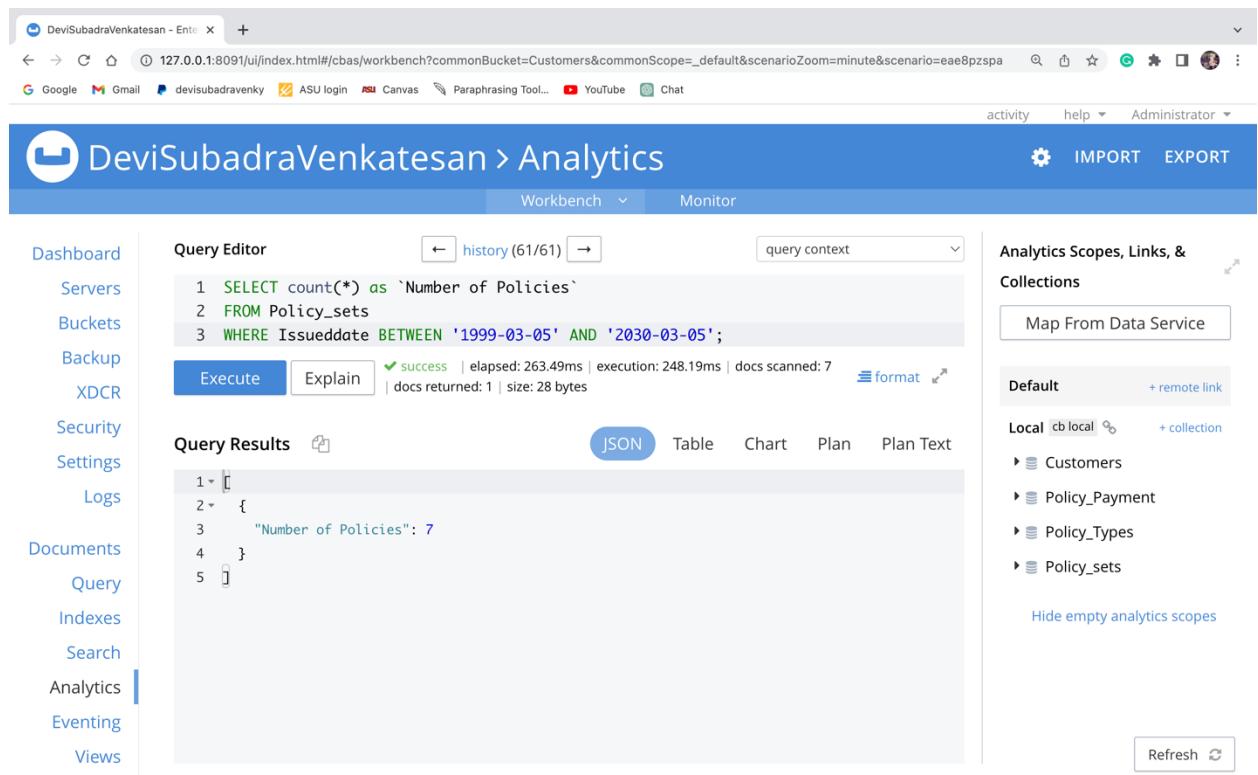
1. How many policies were issued in a particular time period?

```
SELECT count(*) as 'Number of Policies'
```

```
FROM Policy_sets
```

```
WHERE Issueddate BETWEEN '1999-03-05' AND '2030-03-05';
```

Screenshot:



The screenshot shows the DeviSubadraVenkatesan Analytics interface. The left sidebar is a navigation menu with the following items: Dashboard, Servers, Buckets, Backup, XDCR, Security Settings, Logs, Documents, Query, Indexes, Search, Analytics (which is selected), Eventing, and Views. The main area is divided into three sections: Query Editor, Query Results, and Analytics Scopes, Links, & Collections.

Query Editor: Contains the following SQL query:

```

1 SELECT count(*) as `Number of Policies`
2 FROM Policy_sets
3 WHERE Issuedate BETWEEN '1999-03-05' AND '2030-03-05';

```

Query Results: Shows the output of the query in JSON format:

```

1 [
2   {
3     "Number of Policies": 7
4   }
5 ]

```

Analytics Scopes, Links, & Collections: Shows the following scopes:

- Default
- Local cb local (Customers, Policy_Payment, Policy_Types, Policy_sets)

There is also a "Map From Data Service" button and a "Refresh" button.

2) How many Active Policies are there in the system?

A)

```

SELECT COUNT(*) AS Active FROM Policy_sets p
JOIN Policy_Payment pp ON p.PolicyId
WHERE pp.status != 'Fail' AND p.Expirydate >= DATE_PART_STR(NOW_STR(), "2030-07-05");

```

Screenshot:

The screenshot shows the Apache Ignite Workbench interface. The left sidebar contains a navigation menu with items like Dashboard, Servers, Buckets, Backup, XDCR, Security, Settings, Logs, Documents, Query, Indexes, Search, Analytics (which is selected), Eventing, and Views. The main area has a 'Query Editor' with the following SQL query:

```
1 SELECT COUNT(*) AS Active FROM Policy_sets p
2 JOIN Policy_Payment pp ON p.PolicyId
3 WHERE pp.status != 'Fail' AND p.Expirydate >= DATE_PART_STR(NOW_STR(), "2030-07-05");
```

Below the editor, the 'Query Results' section shows the output in JSON format:

```
1 [
2   {
3     "Active": 0
4   }
5 ]
```

On the right, there's a sidebar titled 'Analytics Scopes, Links, & Collections' with a 'Default' section listing 'Customers', 'Policy_Payment', 'Policy_Types', and 'Policy_sets'.

3)Find out the Customer details who has paid their policy payment through

Netbanking also display the Policy coverage amount?

Query:

```
SELECT c.customerId, c.customerName, p.Amount, pp.method
FROM Customers c
JOIN Policy_sets p ON c.customerId = p.customerId
JOIN Policy_Payment pp ON p.policy_id = pp.policy_id
where pp.method="NetBanking" AND pp.status="Success"
```


Screenshot:

The screenshot shows the DeviSubdraVenkatesan Analytics interface. The left sidebar lists various database components: Dashboard, Servers, Buckets, Backup, XDCR, Security, Settings, Logs, Documents, Query, Indexes, Search, Analytics (selected), Eventing, and Views. The main area is the 'Query Editor' with the following SQL query:

```

1 SELECT c.customerId, c.customerName, p.Amount, pp.method
2 FROM Customers c
3 JOIN Policy_sets p ON c.customerId = p.customerId
4 JOIN Policy_Payment pp ON p.policy_id = pp.policy_id
5 where pp.method="NetBanking" AND pp.status="Success"
  
```

Below the query are 'Execute' and 'Explain' buttons. The 'Explain' button is highlighted with a green checkmark and text: 'success | elapsed: 139.99ms | execution: 125.89ms | docs scanned: 21 | docs returned: 1 | size: 91 bytes'. To the right of the query editor is the 'Analytics Scopes, Links, & Collections' panel, which shows 'Default' as the selected scope. Under 'Default', there are entries for 'Local' collections: 'Customers', 'Policy_Payment', 'Policy_Types', and 'Policy_sets'. A 'Refresh' button is located at the bottom right of the main content area.

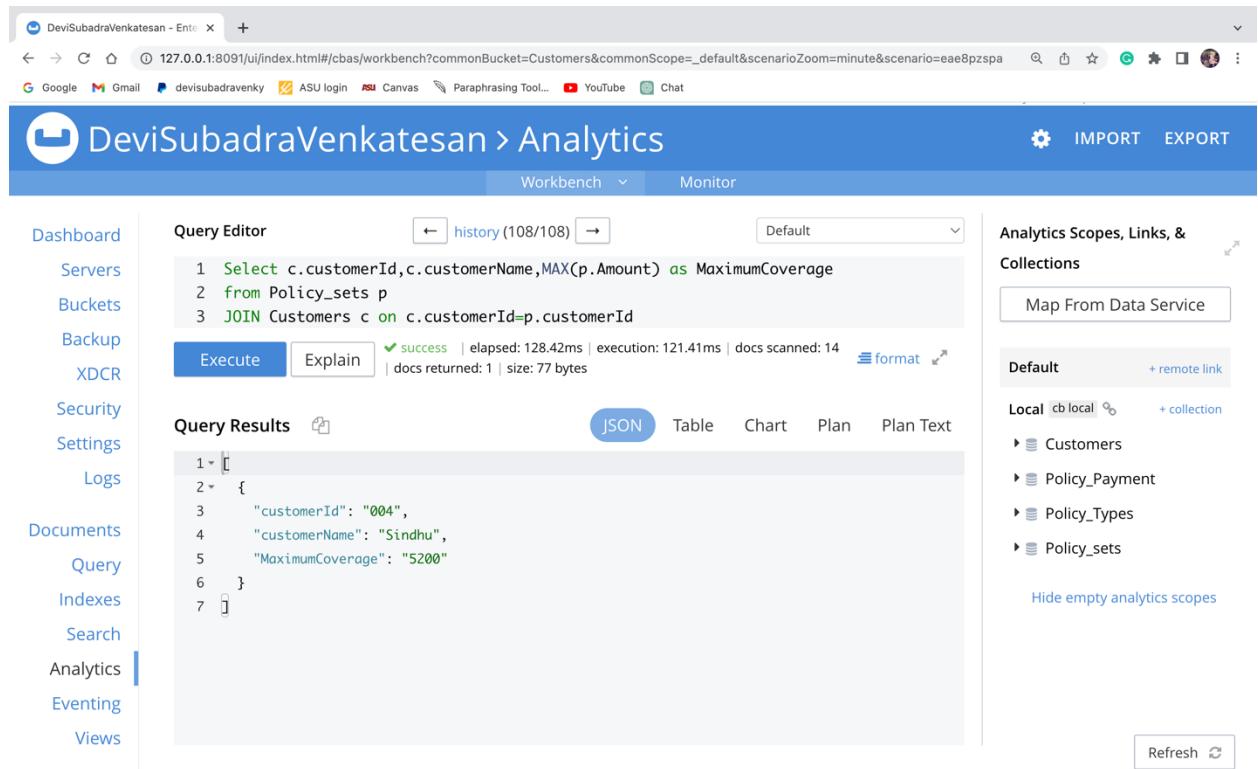
4) Which Customer has the Maximum coverage amount?

Query:

```

Select c.customerId,c.customerName,MAX(p.Amount) as MaximumCoverage
from Policy_sets p
JOIN Customers c on c.customerId=p.customerId
GROUP BY c.customerId,c.customerName
ORDER BY MaximumCoverage DESC
LIMIT 1;
  
```

Screenshot:



The screenshot shows the DeviSubadraVenkatesan Analytics interface. The left sidebar has a 'Analytics' tab selected. The main area has a 'Query Editor' tab open, displaying the following query:

```
1 Select c.customerId,c.customerName,MAX(p.Amount) as MaximumCoverage
2 from Policy_sets p
3 JOIN Customers c on c.customerId=p.customerId
```

Below the query are 'Execute' and 'Explain' buttons, and a success message: 'success | elapsed: 128.42ms | execution: 121.41ms | docs scanned: 14 | docs returned: 1 | size: 77 bytes'. The 'Query Results' section shows the following JSON output:

```
1 [
2 {
3   "customerId": "004",
4   "customerName": "Sindhu",
5   "MaximumCoverage": "5200"
6 }
```

The results are displayed in JSON, Table, Chart, Plan, and Plan Text formats. The right sidebar shows 'Analytics Scopes, Links, & Collections' with a 'Default' scope and a 'Local' collection named 'Customers'. Other collections listed are 'Policy_Payment', 'Policy_Types', and 'Policy_sets'. A 'Refresh' button is at the bottom right of the sidebar.

Summary

The Insurance Policy Database Management System (IPDMS) is a software application that helps insurance companies manage their policies, claims, compliance, and other related activities more efficiently. The system is designed to streamline the insurance process, reduce operational costs, and improve customer service.

IPDMS provides a range of features and capabilities, including policy management, claims processing, reporting and analytics, compliance management, and customer relationship management. It can be customized to meet the specific needs of an insurance company, allowing insurers to tailor the software to fit their unique business requirements.

The importance of an IPDMS cannot be overstated, as it can streamline processes, ensure compliance, and manage risks effectively. By automating tasks such as data entry and claims processing, insurers can enhance operational efficiency, reduce errors, and optimize resource utilization, leading to cost savings.

Moreover, IPDMS's ability to improve customer service is another significant advantage. Automated claims processing with real-time updates can result in better customer satisfaction and retention rates, contributing to the long-term success of the company.

The main agenda of this Insurance Policy Database Management System (IPDMS) is to allocate sustained customized Policies by checking whether any prior accidents, health or natural calamities would have occurred for the vehicle, person and house respectively. The system ensures the entire system is recorded in a suitable manner.

Lastly, an insurance management system can help insurers manage risks associated with underwriting policies by providing real-time data analysis and identifying potential losses. In

summary, an insurance management system is a necessary tool for insurers to stay competitive in today's fast-paced business environment, as it can improve operational efficiency, customer service, compliance, risk management, and contribute to long-term success.

Conclusion

In conclusion, the efficient handling of passenger luggage is critical for the success of airlines and airports. Ultimately, the adoption of Automated Baggage Handling Systems can improve the way airports handle passenger luggage, leading to a better overall travel experience for everyone involved. Regarding further work, it would be beneficial to continue researching and developing innovative luggage handling solutions that can further streamline the process and reduce the likelihood of lost or delayed luggage. Additionally, there is a need to explore the potential environmental impact of luggage handling systems and find ways to make them more sustainable. In terms of the value of NoSQL and SQL in combination, both have their strengths and weaknesses. NoSQL databases are better suited for handling unstructured or semi-structured data and can handle large volumes of data with high velocity. In contrast, SQL databases are better suited for handling structured data and can perform complex queries efficiently. By combining the two, organizations can leverage the strengths of each database type and create a more comprehensive and flexible data management system. This can help organizations to better manage their data and gain valuable insights from it

Reference

1. Terry Haplin, (2015). Object role modeling fundamentals: A Practical Guide to Data Modeling with ORM. Technics Publications
2. Bryan Syverson, Joel Murach , Murach's SQL Server 2016 for developers.