




IE6700 Final Report

# [Explore Secure Travel Insurance]

AKASH GHOSH

4/17/2025



## Table of Contents

<b>1. Executive Summary.....</b>	<b>2</b>
<b>2. Introduction .....</b>	<b>2</b>
<b>3. Problem Statement.....</b>	<b>2</b>
<b>4. Requirement Analysis .....</b>	<b>3</b>
<b>5. System Design.....</b>	<b>3</b>
<b>6. Database Design.....</b>	<b>4</b>
<b>7. Key Use Cases.....</b>	<b>5</b>
<b>8. Conclusion and Future Scope .....</b>	<b>5</b>
<b>9. Milestones (1-5).....</b>	<b>5</b>

## **1. Executive Summary**

The Explore Secure Travel Insurance system is a comprehensive travel insurance platform designed to offer customized and secure insurance services for a diverse range of travelers. It addresses the growing need for personalization in travel coverage, especially for high-risk individuals such as journalists, athletes, and seniors. The system includes robust features like automated quote generation, personalized policy creation, streamlined claims processing, agent commission tracking, and strategic.

## **2. Introduction**

In the ever-expanding global travel market, standard insurance policies often fall short of meeting the specific needs of diverse traveler groups. This project introduces a solution that tailors insurance offerings based on traveler profiles, risk levels, and trip types. By incorporating specialized policy features and a secure digital interface, Explore Secure Travel Insurance enhances the customer experience and operational efficiency for insurers.

## **3. Problem Statement**

Traditional travel insurance systems do not offer the flexibility or personalization required by today's travelers. High-risk categories such as journalists in conflict zones, athletes traveling for events, or elderly travelers with pre-existing conditions face limited coverage and complex claim processes. There is a clear gap for a system that:

- Offers traveler-specific policy customization.
- Supports real-time quote generation and claim submission.
- Facilitates partnerships with travel platforms for seamless integration.
- Tracks agent performance and commissions efficiently.

## 4. Requirement Analysis

Functional Requirements:

- Customer registration and profile management
- Personalized insurance quote generation
- Policy enrollment and tracking
- Claim initiation and status tracking
- Agent management and sales tracking
- Payment processing and receipts
- Travel platform partnerships

Non-Functional Requirements (Future Plan) :

- Secure data handling and encryption
- Scalable cloud-based deployment
- Responsive design for mobile and desktop
- High availability and disaster recovery

## 5. System Design

Enhanced Entity Relationship (EER) Model:

- Entities: Customer, InsurancePolicy, PolicyType, PersonalizedQuote, Claim, Agent, Payment, TravelPlatform, HighRiskCoverage, Partnership, CountryOffers.
- Relationships: Customers purchase policies, submit claims, and interact with agents and platforms. Specialized travelers fall under sub-entities with high-risk flag.

UML Class Diagram:

- Key Classes: Customer, Agent, InsurancePolicy, Claim, Payment, Platform.
- Attributes and Methods: e.g., Customer(name, age, type), generateQuote(), processClaim().
- Associations: Customers linked to Agents; Policies linked to Quotes and Claims; Payments associated with Policies.

## 6. Database Design

Tables include:

- Customer (CustomerID, Name, DOB, Category, etc.)
- PolicyType (TypeID, Name, Description)
- InsurancePolicy (PolicyID, TypeID, CustomerID, StartDate, EndDate, Premium)
- PersonalizedQuote (QuoteID, CustomerID, QuoteAmount, Validity)
- Claim (ClaimID, PolicyID, Date, Status)
- ClaimProcessing (ProcessID, ClaimID, AgentID, Steps, Timestamp)
- Agent (AgentID, Name, Contact)
- AgentSales (SaleID, AgentID, PolicyID, Commission)
- Payment (PaymentID, CustomerID, Amount, Date, Method)
- TravelPlatform (PlatformID, Name, URL)
- Partnership (PartnerID, PlatformID, StartDate)
- HighRiskCoverage (CoverageID, Category, ExtraPremium, Description)
- CountryOffers (CountryID, OfferDetails, Eligibility)

## 7. Key Use Cases

### 1. Customer Quote Generation:

- A senior customer inputs travel details.
- The system suggests high-risk coverage options.
- A personalized quote is generated.

### 2. Claim Submission and Processing:

- A journalist files a claim from abroad.
- An agent is assigned to validate documents.
- The status is updated and customer notified.

### 3. Agent Sales and Commissions:

- An agent onboards new customers.
- Sales are logged and commissions calculated.

### 4. Platform Partnership Integration:

- A travel platform integrates insurance purchase at checkout.
- The system records referral and calculates partner incentives.

## 8. Conclusion and Future Scope

Explore Secure Travel Insurance bridges a critical gap in the travel insurance domain by offering a modern, user-focused, and high-coverage system for today's dynamic travelers. Future improvements may include:

- AI-based risk assessment and fraud detection.
- Integration with wearable health trackers.
- Predictive analytics for customer retention and policy improvement.

## 9. Milestones (1-5)

Please see below all the milestone that has been done throughout the semester.

## Milestone #1

### Introduction

Explore Secure Travel Insurance is designed to handle the creation of managing and processing of Insurance policies, personalized quotes, claims and solve the problems for different type of traveler. The system integrates with agents, travel platforms and partners which also offers various types of coverage and risk levels tailored for specific customers' demand.

### Entity Relationship Model

The core components of the model are captured in the following tables Customers, policies, claims and agent relationships.

- Customer: Customer represents the person who is seeking the insurance. They Key attributes include customer\_id (PK), name, gender, date\_of\_birth, occupation, travel \_purpose and risk\_level.

A customer can purchase multiple policies, receive personalized quotes and file claim or multiple claims.

- **Policy Type:** Policy type represents different type of policies such as health, travel, accident etc. It consists of policy\_type\_id and type\_name.
- **Insurance Policy:** Insurance policy represent specific insurance policies available for customers. Some attributes include policy\_id, policy\_name, coverage\_details and premium. It is lined with a policy\_type\_id.

Policies are related to multiple customers through the CustomerPolicy table. They also have a direct association with claims and personalized quotes.

- **Personalized Quote:** Provides customers with customized insurance quoted based on the customer profile which includes attribute quote\_id, customer\_id, policy\_id, premium, coverage\_adjustments and quote\_date.

A customer can receive multiple personalized quotes for different policy.

- **Customer Policy:** It represents customer with policies they purchase by specifying the purchase date and expiry date.

Each customer can purchase multiple policies, and each policy can be purchased by multiple customers.

- **Claim:** Claim table represent a customer request for insurance compensation which has attributes as claim\_id, customer\_id, policy\_id, claim\_status, claim\_amount and submission\_date.
- **Claim Processing:** Tracks the claim's status, assigned agent and notes which has processing\_id, claim\_id, assigned\_agent and processing\_status.



- Agent: Agent table represent managing customer policies and sales. The agent has an agent\_id, name and contact\_info.
- Agent Sales: Link agents with policies that sold to customers with sale\_date and commission.

An agent can sell multiple policies, and a customer can buy multiple policies through different agents.

- Payment: Tracks customer payments for the policies that they bought which has payment\_id, customer\_id, policy\_id, amount and payment date.

Payment can be made by customers for specific policies.

- Travel Platform: It represents platform that collaborate with the insurance company to sell policies to their own users which includes platform\_id and name.
- Partnership: Captures the partnership between travel platforms and insurance policies which has commission and date.

A platform can have multiple policies in partnership with the insurance company.

- High Risk coverage: It represent additional coverage for high risk customers who travels to high-risk areas which includes coverage\_id, risk\_type and coverage\_amount.

High risk coverage is linked with insurance policies.

- Country Offers: Specific country that covers the insurance mostly which include discount and special\_terms.

Provide insurance to specific countries.

### Cardinality and Relationships:

- Customer to policy: Many to many relationships through CustomerPolicy table, as a customer have multiple policies, and each policy can be bought multiple customers.
- Customer to Personalized Quote: one to many, a customer can get multiple quotes.
- Customer to Claim: one to many, since a customer can claim multiple claims.
- Agent to customer policy: Many to many relationships through AgentSales where agents can sell multiple policies to many customers.
- Agent to ClaimProcessing: One to many as agents can process multiple claims.
- Policy to Claim: One to many, since policy can have multiple claims.

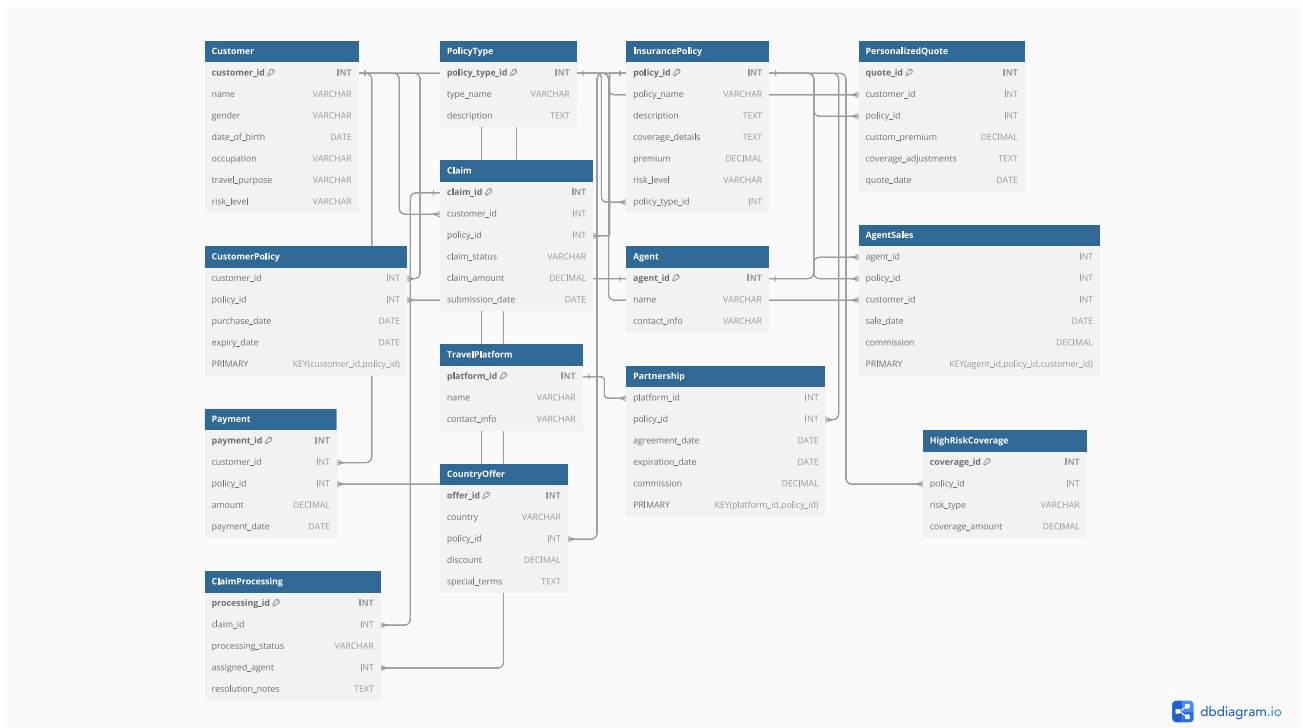
### Business Requirements & Coverage:

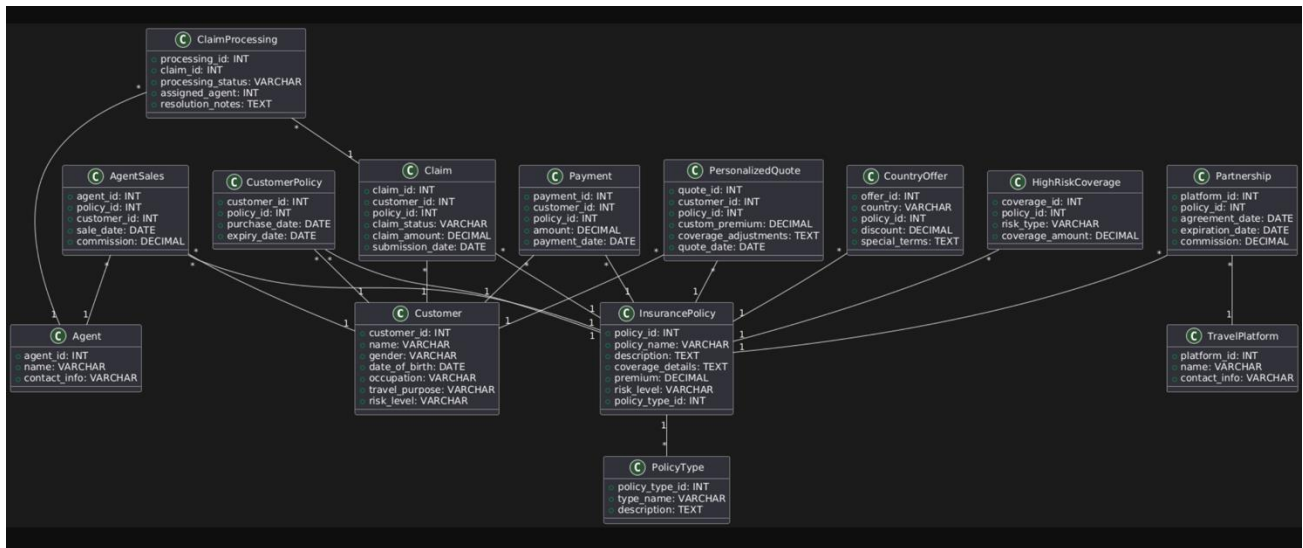
- Risk Level: Each customer may have different risk level policy that can impact the premium and the coverage details provided.
- Claim Status: Claims need to pass through different stages such as submitted, under review and resolved. Design needs to track and update the status.
- Agent Commission: Agents should earn commission based on the policies they are selling. Commission should be tracked for each sale.
- Specific Country Policies: Some policies may offer different type of policy and premium depending on the location and safety.

## Unsupported Business Requirements:

- The current design does not support real time claim processing or claim adjusting.
- The current system also not support dynamic pricing model for the customers based on their data.
- The current database design uses static risk\_level to customers and policies but not supporting real time risk calculations.
- The current database does not handle policy cancellation or any sort of refund policy.

## EER Diagram and UML Diagram:





## Milestone #2

Introduction: The explore secure Travel insurance database system is designed to handle the creation, management, processing of insurance policies, personalized quotes, claims and payments. The database system integrates with agents, travel platforms and partners offering various types of coverage and risk level to customer's needs.

Entity Relationship Model:

The components of the ER Model include in the database system are:

1. **Customers** – Individuals purchasing insurance policies.
2. **Insurance Policies** – Different types of coverage options.
3. **Policy Types** – Categorization of policies (Health, Travel, Accident, etc.).
4. **Personalized Quotes** – Customized pricing for customers based on profile.
5. **Claims** – Insurance compensation requests by customers.
6. **Claim Processing** – Tracking claim status and assigned agents.
7. **Agents** – Representatives selling policies and managing claims.
8. **Agent Sales** – Tracks policies sold by agents and their commissions.
9. **Payments** – Tracks payments made by customers for policies.
10. **Travel Platforms** – External platforms partnering with the insurance company.
11. **Partnerships** – Agreements between insurance companies and travel platforms.
12. **High-Risk Coverage** – Additional coverage for customers traveling to high-risk areas.

### 13. **Country Offers** – Country-specific policy discounts and terms.

#### **Relational Model (Logical Schema):**

Each of the entity is transformed into a relational table with primary keys (PK), foreign keys (FK), and constraints.

- **Customer Table**

Customer (

customer\_id INT NOT NULL,

name VARCHAR(100) NOT NULL,

gender VARCHAR(20) NULL,

date\_of\_birth DATE NOT NULL,

occupation VARCHAR(100) NULL,

travel\_purpose VARCHAR(50) NOT NULL,

risk\_level VARCHAR(20) NOT NULL,

PRIMARY KEY (customer\_id),

CONSTRAINT chk\_risk\_level CHECK (risk\_level IN ('Low', 'Medium', 'High')),

CONSTRAINT chk\_travel\_purpose CHECK (travel\_purpose IN ('Business', 'Leisure', 'Medical', 'Education'))

)

- **Policy Type Table**

**PolicyType (**

**policy\_type\_id INT NOT NULL,**

**type\_name VARCHAR(50) NOT NULL,**

**description TEXT NULL,**

**PRIMARY KEY (policy\_type\_id),**

**CONSTRAINT uniq\_type\_name UNIQUE (type\_name)**

**)**

- **Insurance Policy Table**

**InsurancePolicy (**

**policy\_id INT NOT NULL,**

**policy\_name VARCHAR(100) NOT NULL,**

**description TEXT NULL,**

**coverage\_details TEXT NOT NULL,**

**premium DECIMAL(10,2) NOT NULL,**

**risk\_level VARCHAR(20) NOT NULL,**

**policy\_type\_id INT NOT NULL,**

**PRIMARY KEY (policy\_id),**

**FOREIGN KEY (policy\_type\_id) REFERENCES**

**PolicyType(policy\_type\_id),**

**CONSTRAINT chk\_premium CHECK (premium > 0)**

- **Insurance Policy Table**

```
CREATE TABLE InsurancePolicy (  
  policy_id INT PRIMARY KEY,  
  policy_name VARCHAR(255) NOT NULL,  
  policy_type_id INT NOT NULL,  
  coverage_details TEXT NOT NULL,  
  premium DECIMAL(10,2) NOT NULL,  
  FOREIGN KEY (policy_type_id) REFERENCES PolicyType(policy_type_id)  
);
```

- **Personalized Quote Table**

```
PersonalizedQuote (  
  quote_id INT NOT NULL,  
  customer_id INT NOT NULL,  
  policy_id INT NOT NULL,  
  custom_premium DECIMAL(10,2) NOT NULL,  
  coverage_adjustments TEXT NULL,  
  quote_date DATE NOT NULL,  
  PRIMARY KEY (quote_id),  
  FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),  
  FOREIGN KEY (policy_id) REFERENCES InsurancePolicy(policy_id),  
  CONSTRAINT chk_custom_premium CHECK (custom_premium > 0)  
)
```

- **Customer Policy Table**

```
CustomerPolicy (  
  customer_id INT NOT NULL,  
  policy_id INT NOT NULL,  
  purchase_date DATE NOT NULL,  
  expiry_date DATE NOT NULL,  
  PRIMARY KEY (customer_id, policy_id),  
  FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),  
  FOREIGN KEY (policy_id) REFERENCES InsurancePolicy(policy_id),  
  CONSTRAINT chk_dates CHECK (expiry_date > purchase_date)  
)
```

- **Claim Table**

```
CREATE TABLE Claim (  
    claim_id INT PRIMARY KEY,  
    customer_id INT NOT NULL,  
    policy_id INT NOT NULL,  
    claim_status VARCHAR(50) NOT NULL,  
    claim_amount DECIMAL(10,2) NOT NULL,  
    submission_date DATE NOT NULL,  
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),  
    FOREIGN KEY (policy_id) REFERENCES InsurancePolicy(policy_id)  
);
```

- **Claim Processing Table**

```
CREATE TABLE ClaimProcessing (  
    processing_id INT PRIMARY KEY,  
    claim_id INT NOT NULL,  
    assigned_agent INT NOT NULL,  
    processing_status VARCHAR(50) NOT NULL,  
    FOREIGN KEY (claim_id) REFERENCES Claim(claim_id),  
    FOREIGN KEY (assigned_agent) REFERENCES Agent(agent_id)  
);
```

- **Agent Table**

```
CREATE TABLE Agent (  
    agent_id INT PRIMARY KEY,  
    name VARCHAR(255) NOT NULL,  
    contact_info VARCHAR(255) NOT NULL  
);
```

- **Agent Sales Table**

```
CREATE TABLE AgentSales (  
    agent_id INT NOT NULL,  
    customer_id INT  
    policy_id INT NOT NULL,  
    sale_date DATE NOT NULL,  
    commission DECIMAL(10,2) NOT NULL,  
    PRIMARY KEY (agent_id, policy_id),
```



```
FOREIGN KEY (agent_id) REFERENCES Agent(agent_id),  
FOREIGN KEY (policy_id) REFERENCES InsurancePolicy(policy_id)  
);
```

- **Payment Table**

```
CREATE TABLE Payment (  
    payment_id INT PRIMARY KEY,  
    customer_id INT NOT NULL,  
    policy_id INT NOT NULL,  
    amount DECIMAL(10,2) NOT NULL,  
    payment_date DATE NOT NULL,  
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),  
    FOREIGN KEY (policy_id) REFERENCES InsurancePolicy(policy_id)  
);
```

- **Travel Platform Table**

```
CREATE TABLE TravelPlatform (  
    platform_id INT PRIMARY KEY,  
    name VARCHAR(255) NOT NULL  
);
```

- **Partnership Table**

```
CREATE TABLE Partnership (  
    platform_id INT NOT NULL,(PK)  
    policy_id INT NOT NULL,  
    commission DECIMAL(10,2) NOT NULL,  
    partnership_date DATE NOT NULL,  
    PRIMARY KEY (platform_id, policy_id),  
    FOREIGN KEY (platform_id) REFERENCES TravelPlatform(platform_id),  
    FOREIGN KEY (policy_id) REFERENCES InsurancePolicy(policy_id)  
);
```

- **High Risk coverage Table**

```
CREATE TABLE HighRiskCoverage (  
    coverage_id INT PRIMARY KEY, (PK)  
    policy_id INT  
    risk_type VARCHAR(255) NOT NULL,
```

```
coverage_amount DECIMAL(10,2) NOT NULL  
);
```

- **Country Offers Table**

```
CREATE TABLE CountryOffers (  
offer_id INT  
country_name VARCHAR(255) PRIMARY KEY,  
discount DECIMAL(5,2),  
special_terms TEXT  
);
```

**Normalization Analysis:**

1NF :

- All relations have been designed with atomic attributes
- No repeating groups exists in the Database
- All tables have primary keys defined.

2NF:

- All relations are in 1NF
- No partial dependencies exist in relations.
- Customer policy and agent sales table has been structured with composite keys.

3NF:

- All relations are in 2NF
- No transitive dependencies exist
- Policy type have been separated into their own relation
- Claim processing details have been separated.

**Constraints and Business Procedure:**

- Primary keys are defined as not null for the relations
- Unique constraints are applied when needed
- Foreign key relations are properly added
- Delete and update actions are restricted to maintain data integrity
- Date validations (expiry date > purchase date)
- Numeric validations (premium >0)

**Unsupported Constraints:**

- Complex premium calculations
- Time based policy status update
- Commissions calculations rules

- Risk level assessment rule

## Milestone# 3

In **Milestone #3**, I have successfully designed, implemented, and populated the Explore **Travel Insurance Management System** database. This phase involved defining the database schema, creating tables with appropriate constraints, and inserting sample data to validate relationships and ensure data handling.

# 1. Table Creation

I have structured the database by creating 13 interrelated tables, incorporating primary and foreign key constraints to maintain data integrity. The following tables were implemented:

### Core Entities

- Customer: Stores customer details such as name, gender, date of birth, occupation, travel purpose, and risk level. Constraints were added to ensure valid risk levels (Low, Medium, High) and travel purposes (Business, Leisure, Medical, Education).
- 
- Policy Type: Defines different insurance policy types, ensuring uniqueness. I added three new policy types:
  - Travel Blogger Insurance
  - Infant Coverage
  - Teen Coverage

### Policy Management

- Insurance Policy: Stores details about each policy, including name, type, coverage details, and premium amount. A foreign key links this table to the Policy Type table.
- Personalized Quote: Allows customization of insurance policies per customer, storing premium adjustments, coverage modifications, and quote generation dates.

### Customer & Policy Relations

- Customer Policy: Tracks which customers purchased which policies. Includes constraints to ensure the expiry date is always later than the purchase date to prevent invalid entries. This table follows a many-to-many relationship structure.

### Claims & Processing

- Claim: Stores records of insurance claims, including the claim amount, status (Pending, Approved, Rejected), and submission date.
- Claim Processing: Handles claim processing, linking each claim to an assigned agent. Ensures referential integrity by linking to the Agent table.

### Agent & Sales Management

- Agent: Stores insurance agents' details, including name and contact information.
- Agent Sales: Tracks which agents sold which policies, along with the sale date and commission earned.

### Financial Transactions

- Payment: Stores records of premium payments made by customers for their policies, ensuring correct transaction tracking.

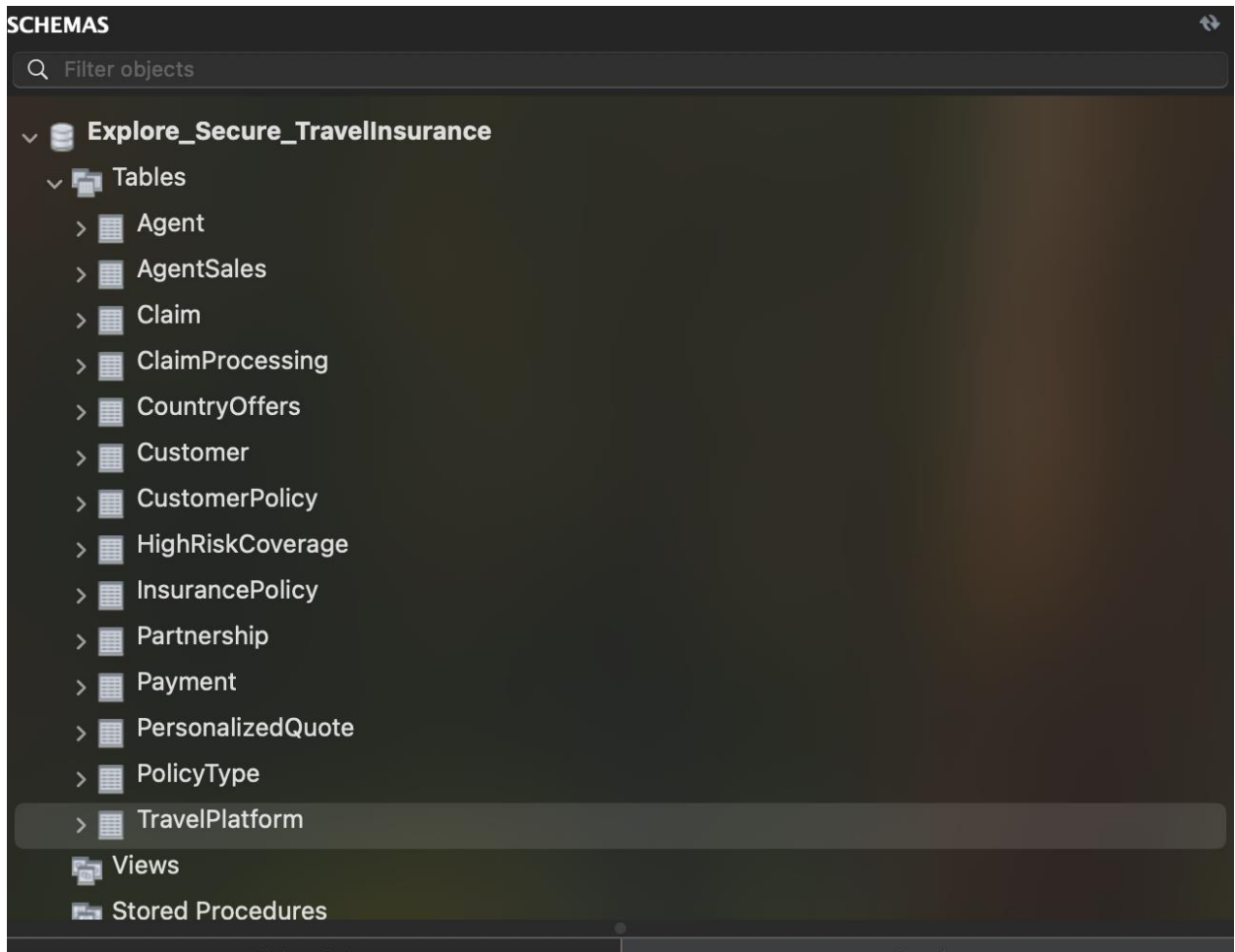
### Business Partnerships & Risk Coverage

- Travel Platform: Represents travel-related platforms partnering with the insurance provider.
- Partnership: Manages relationships between travel platforms and insurance policies, including commission rates and partnership start dates.
- High-risk Coverage: Records specialized coverage for high-risk scenarios, such as extreme sports or hazardous travel destinations.

### Promotional Offers

- Country Offers: Tracks country-based discounts and special terms for policies offered in different regions.

Added below a screenshot of the Database with the tables which were created in MySQL.



## 2. Data Insertion

To validate my schema and ensure real-world applicability, I have populated the database with sample data:

### Customers

- Added 20 customers with varied demographics, occupations, travel purposes, and risk levels.
- Ensured diversity in risk categorization (e.g., business travelers categorized as Low-Medium risk, adventure travelers categorized as High risk).

### Insurance Policies

- Inserted 10 insurance policies, linking them to different policy types and defining comprehensive coverage details.

### Customer Policies

- Mapped 20+ customers to their purchased policies, ensuring that expiry dates are valid and match constraints.
- Addressed duplicate primary key errors by carefully handling customer-policy relationships.

#### Claims Management

- Inserted 20+ claims with varying statuses (Approved, Pending, Rejected) and different claim amounts.
- Ensured each claim is linked to a valid policy and customer in the system.

#### Agent Assignments & Sales

- Populated Agent and Agent Sales tables with 10+ insurance agents who are responsible for handling claims and selling policies.
- Tracked sales commissions for agents based on policy sales.

#### Payments & Transactions

- Added 10+ payment records, ensuring all transactions are mapped correctly to customer policies.

#### Business Partnerships & High-Risk Coverage

- Established partnerships between multiple travel platforms and insurance policies to model real-world collaborations.
- Inserted records for high-risk coverage, such as policies covering adventure travel, high-altitude trekking, and extreme sports.

## MILESTONE#4

**I have added the code for MySQL connection with Python and some analysis of existing data.**

```
import mysql.connector
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Establish connection
```

```
conn = mysql.connector.connect(
    host="localhost",    # Host name
    user="root",        # Mysql user name
    password="*",       # SQL password
    database="Explore_Secure_TravelInsurance" # Database name
)
```

In [56]:

In [58]:

```
# Create cursor object
cursor = conn.cursor()

# Execute a query
cursor.execute("SHOW TABLES;")

# Fetch and display results
tables = cursor.fetchall()
print("Tables in the database:", tables)
Tables in the database: [('Agent',), ('AgentSales',), ('Claim',), ('ClaimProcessing',), ('CountryOffers',), ('Customer',), ('CustomerPolicy',), ('HighRiskCoverage',), ('InsurancePolicy',), ('Partnership',), ('Payment',), ('PersonalizedQuote',), ('PolicyType',), ('TravelPlatform',)]
```

In [60]:

```
cursor.execute("SELECT * FROM `Customer`;")
print(cursor.fetchall())
[(1, 'Alice Johnson', 'Female', datetime.date(1985, 6, 15), 'Engineer', 'Business', 'Medium'), (2, 'Bob Smith', 'Male', datetime.date(1990, 9, 23), 'Doctor', 'Medical', 'High'), (3, 'Charlie Brown', 'Male', datetime.date(1987, 3, 10), 'Professor', 'Education', 'Low'), (4, 'Diana Prince', 'Female', datetime.date(1995, 12, 5), 'Journalist', 'Business', 'Medium'), (5, 'Evan Rogers', 'Male', datetime.date(1980, 7, 18), 'Athlete', 'Leisure', 'High'), (6, 'Fiona Davis', 'Female', datetime.date(1992, 11, 30), 'Blogger', 'Leisure', 'Low'), (7, 'George Miller', 'Male', datetime.date(1983, 4, 25), 'Lawyer', 'Business', 'Medium'), (8, 'Helen Carter', 'Female', datetime.date(1975, 5, 22), 'Retired', 'Leisure', 'Low'), (9, 'Ian Thompson', 'Male', datetime.date(1999, 8, 14), 'Student', 'Education', 'Medium'), (10, 'Jessica Lee', 'Female', datetime.date(1988, 2, 17), 'Entrepreneur', 'Business', 'High'), (11, 'Kevin Malone', 'Male', datetime.date(1982, 11, 2), 'Accountant', 'Business', 'Low'), (12, 'Angela Martin', 'Female', datetime.date(1980, 6, 25), 'Financial Analyst', 'Business', 'Medium'), (13, 'Ryan Howard', 'Male', datetime.date(1985, 5, 10), 'Marketing Executive', 'Business', 'High'), (14, 'Kelly Kapoor', 'Female', datetime.date(1987, 2, 13), 'Social Media Manager', 'Leisure', 'Medium'), (15, 'Toby Flenderson', 'Male', datetime.date(1975, 8, 11), 'HR Manager', 'Leisure', 'Low'), (16, 'Creed Bratton', 'Male', datetime.date(1960, 10, 14), 'Retired', 'Leisure', 'Low'), (17, 'Meredith Palmer', 'Female', datetime.date(1978, 4, 22), 'Sales Representative', 'Business', 'Medium'), (18, 'Oscar Martinez', 'Male', datetime.date(1981, 9, 7), 'Tax Consultant', 'Business', 'Low'), (19, 'Jan Levinson', 'Female', datetime.date(1973, 12, 30), 'Senior Manager', 'Business', 'High'), (20, 'David Wallace', 'Male', datetime.date(1968, 5, 17), 'Executive', 'Business', 'High')]
```

In [62]:

```
cursor = conn.cursor()

# Fetch all table names
cursor.execute("SHOW TABLES;")
tables = [table[0] for table in cursor.fetchall()]

# Iterate over each table and fetch data
for table in tables:
    print(f"\nFetching data from table: {table}")

    try:
        query = f"SELECT * FROM `{table}`;" # Use backticks for safety
        cursor.execute(query)

        # Get column names
        columns = [desc[0] for desc in cursor.description]

        # Fetch data
        rows = cursor.fetchall()

        # Convert to Pandas DataFrame
```

```

df = pd.DataFrame(rows, columns=columns)

# Display first 5 rows
print(df.head())
except Exception as e:
    print(f"Error fetching data from {table}: {e}")

```

Fetching data from table: Agent

	agent_id	name	contact_info
0	1	Michael Scott	michael@insurance.com
1	2	Pam Beesly	pam@insurance.com
2	3	Jim Halpert	jim@insurance.com
3	4	Dwight Schrute	dwight@insurance.com
4	5	Stanley Hudson	stanley@insurance.com

Fetching data from table: AgentSales

	agent_id	policy_id	sale_date	commission
0	1	101	2025-01-05	50.00
1	2	102	2025-01-10	75.00
2	3	103	2025-02-01	30.00
3	4	104	2025-02-15	20.00
4	5	105	2025-03-01	90.00

Fetching data from table: Claim

	claim_id	customer_id	policy_id	claim_status	claim_amount	submission_date
0	1	1	101	Pending	5000.00	2025-02-20
1	2	2	103	Approved	1500.00	2025-02-25
2	3	3	104	Rejected	800.00	2025-03-01
3	4	4	102	Processing	12000.00	2025-03-10
4	5	5	105	Pending	4500.00	2025-03-15

Fetching data from table: ClaimProcessing

	processing_id	claim_id	assigned_agent	processing_status
0	1	1	2	Under Review
1	2	2	3	Completed
2	3	3	1	Rejected
3	4	4	4	In Progress
4	5	5	5	Pending

Fetching data from table: CountryOffers

	offer_id	country_name	discount \
0	1	USA	10.00
1	2	Canada	5.00
2	3	Germany	7.50
3	4	Australia	12.00
4	5	Japan	6.00

	special_terms
0	10% discount on all travel policies
1	Applicable only for first-time travelers
2	Limited to business travel policies
3	Special discount for students and seniors
4	Coverage includes natural disaster protection

Fetching data from table: Customer

	customer_id	name	gender	date_of_birth	occupation \
--	-------------	------	--------	---------------	--------------



0	1	Alice Johnson	Female	1985-06-15	Engineer
1	2	Bob Smith	Male	1990-09-23	Doctor
2	3	Charlie Brown	Male	1987-03-10	Professor
3	4	Diana Prince	Female	1995-12-05	Journalist
4	5	Evan Rogers	Male	1980-07-18	Athlete

	travel_purpose	risk_level
0	Business	Medium
1	Medical	High
2	Education	Low
3	Business	Medium
4	Leisure	High

Fetching data from table: CustomerPolicy

	customer_id	policy_id	purchase_date	expiry_date
0	1	101	2024-01-01	2025-01-01
1	2	102	2024-02-10	2025-02-10
2	2	103	2024-02-15	2025-02-15
3	3	103	2023-12-05	2024-12-05
4	3	104	2024-03-10	2025-03-10

Fetching data from table: HighRiskCoverage

	coverage_id	policy_id	risk_type	coverage_amount
0	1	101	Extreme Sports Injury	50000.00
1	2	102	War Zone Coverage	75000.00
2	3	103	Pandemic Insurance	60000.00
3	4	104	Terrorism Coverage	80000.00
4	5	105	Political Unrest	70000.00

Fetching data from table: InsurancePolicy

	policy_id	policy_name	policy_type_id \
0	101	Basic Health Plan	1
1	102	Comprehensive Travel Plan	1
2	103	Flight Cancellation Protection	2
3	104	Baggage Protection Plan	3
4	105	Extreme Sports Insurance	4

	coverage_details	premium
0	Covers hospitalization and medical expenses	200.00
1	Includes health, baggage, and flight coverage	500.00
2	Covers full refund on flight cancellation	150.00
3	Covers baggage loss and theft	100.00
4	Covers injuries from sports activities	350.00

Fetching data from table: Partnership

	platform_id	policy_id	commission	partnership_date
0	1	101	20.00	2025-01-01
1	2	102	25.00	2025-01-05
2	3	103	15.00	2025-02-01
3	4	104	10.00	2025-02-15
4	5	105	30.00	2025-03-01

Fetching data from table: Payment

	payment_id	customer_id	policy_id	amount	payment_date
0	1	1	101	200.00	2025-02-20

1	2	2	103	150.00	2025-02-25
2	3	3	104	100.00	2025-03-01
3	4	4	102	500.00	2025-03-10
4	5	5	105	350.00	2025-03-15

Fetching data from table: PersonalizedQuote

	quote_id	customer_id	policy_id	custom_premium \
0	1	1	101	180.00
1	2	2	103	140.00
2	3	3	104	90.00
3	4	4	102	480.00
4	5	5	105	330.00

	coverage_adjustments	quote_date
0	Extended hospitalization coverage	2025-03-01
1	Added trip delay coverage	2025-03-02
2	None	2025-03-03
3	Added higher baggage protection	2025-03-04
4	Extreme skiing coverage included	2025-03-05

Fetching data from table: PolicyType

	policy_type_id	type_name \
0	1	Health Insurance
1	2	Flight Cancellation
2	3	Baggage Loss
3	4	Extreme Sports Coverage
4	5	Senior Citizen Coverage

	description
0	Covers medical expenses during travel
1	Covers cost of canceled flights due to emergen...
2	Covers lost or stolen baggage
3	Covers injuries from adventure sports
4	Special policies for elderly travelers

Fetching data from table: TravelPlatform

	platform_id	name
0	1	Expedia
1	2	Booking.com
2	3	Airbnb
3	4	Skyscanner
4	5	Kayak

In [64]:

*# Fetching the data from claim table create a plot to see the claim status of each one from the data.*

*# Fetch data from the Claim table*

query = "SELECT \* FROM Claim;"

cursor.execute(query)

*# Get column names*

columns = [desc[0] for desc in cursor.description]

*# Fetch data*

rows = cursor.fetchall()

*# Convert to Pandas DataFrame*

```
df_claim = pd.DataFrame(rows, columns=columns)
# Plotting the claim status distribution
plt.figure(figsize=(8, 6))
sns.countplot(data=df_claim, x='claim_status', palette='Set2')
plt.title('Distribution of Claim Status')
plt.xlabel('Claim Status')
plt.ylabel('Count')
plt.grid(True)
plt.show()
```

/var/folders/34/jqpjg\_x508j0x978h6stq2j80000gn/T/ipykernel\_5177/154940699.py:15: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.countplot(data=df_claim, x='claim_status', palette='Set2')
```



In [70]:

```
# Fetching from country tables data to see countries which offers discounts.
# Fetch data from the CountryOffers table
query = "SELECT * FROM CountryOffers;"
cursor.execute(query)
```

```
# Get column names
columns = [desc[0] for desc in cursor.description]
```

```
# Fetch data
rows = cursor.fetchall()
```

```
# Convert to Pandas DataFrame
df_country_offers = pd.DataFrame(rows, columns=columns)
```

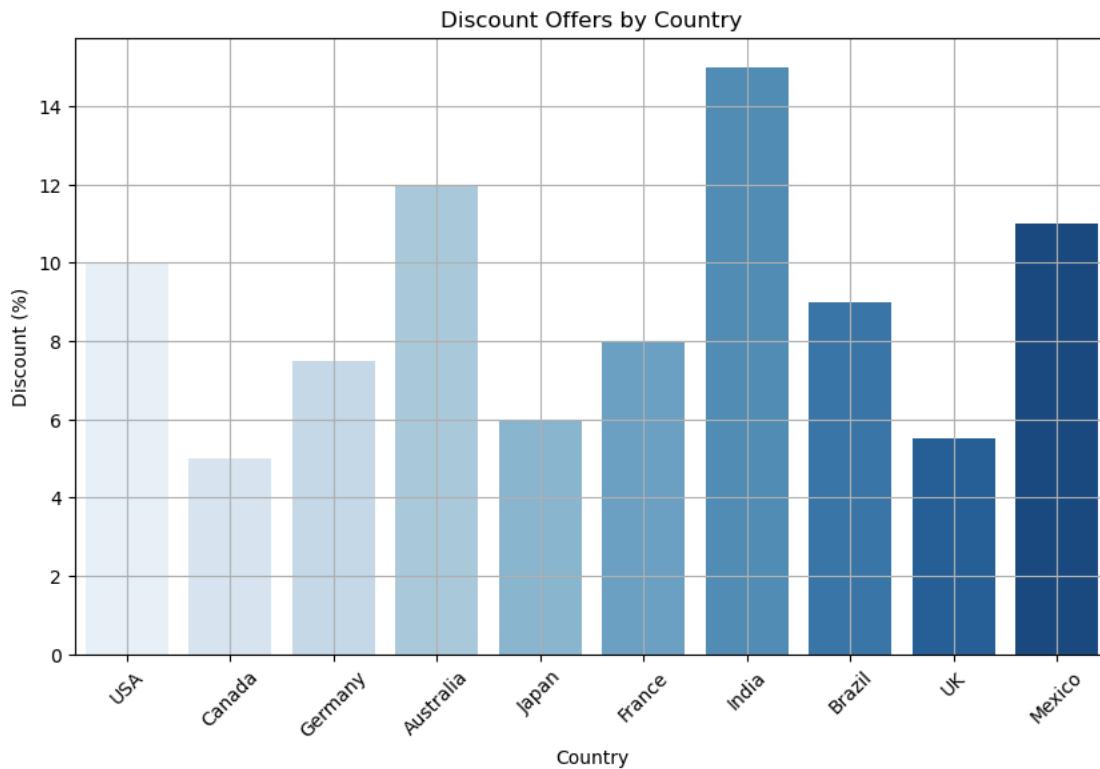
```
# Plot the distribution of discounts by country
plt.figure(figsize=(10, 6))
```

```
sns.barplot(data=df_country_offers, x='country_name', y='discount', palette='Blues')
plt.title('Discount Offers by Country')
plt.xlabel('Country')
plt.ylabel('Discount (%)')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```

/var/folders/34/jqpjg\_x508j0x978h6stq2j80000gn/T/ipykernel\_5177/2443226331.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(data=df_country_offers, x='country_name', y='discount', palette='Blues')
```



In [76]:

```
#Sum of amount customers paid for each policy .
```

```
cursor = conn.cursor()
```

```
# Fetch data from the Payment table
```

```
cursor.execute("SELECT * FROM Payment;")
```

```
columns = [desc[0] for desc in cursor.description]
```

```
rows = cursor.fetchall()
```

```
df_payment = pd.DataFrame(rows, columns=columns)
```

```
# Fetch data from the InsurancePolicy table
```

```
cursor.execute("SELECT * FROM InsurancePolicy;")
```

```
columns = [desc[0] for desc in cursor.description]
```

```
rows = cursor.fetchall()
```

```
df_insurance_policy = pd.DataFrame(rows, columns=columns)
```

```
# Group payments by policy_id and sum the amounts
```

```
df_payment_policy = df_payment.groupby('policy_id')['amount'].sum().reset_index()
```

```
# Merge payment data with insurance policy data to get policy names
```

```
df_payment_policy = pd.merge(df_payment_policy, df_insurance_policy, on='policy_id', how='inner')
```

```
# Plot total payment amount by policy
```

```
plt.figure(figsize=(10, 6))
```

```
sns.barplot(data=df_payment_policy, x='policy_name', y='amount', palette='viridis')
```

```
plt.title("Total Payment Amount by Policy")
```

```
plt.xlabel('Policy Name')
```

```
plt.ylabel('Total Payment Amount')
```

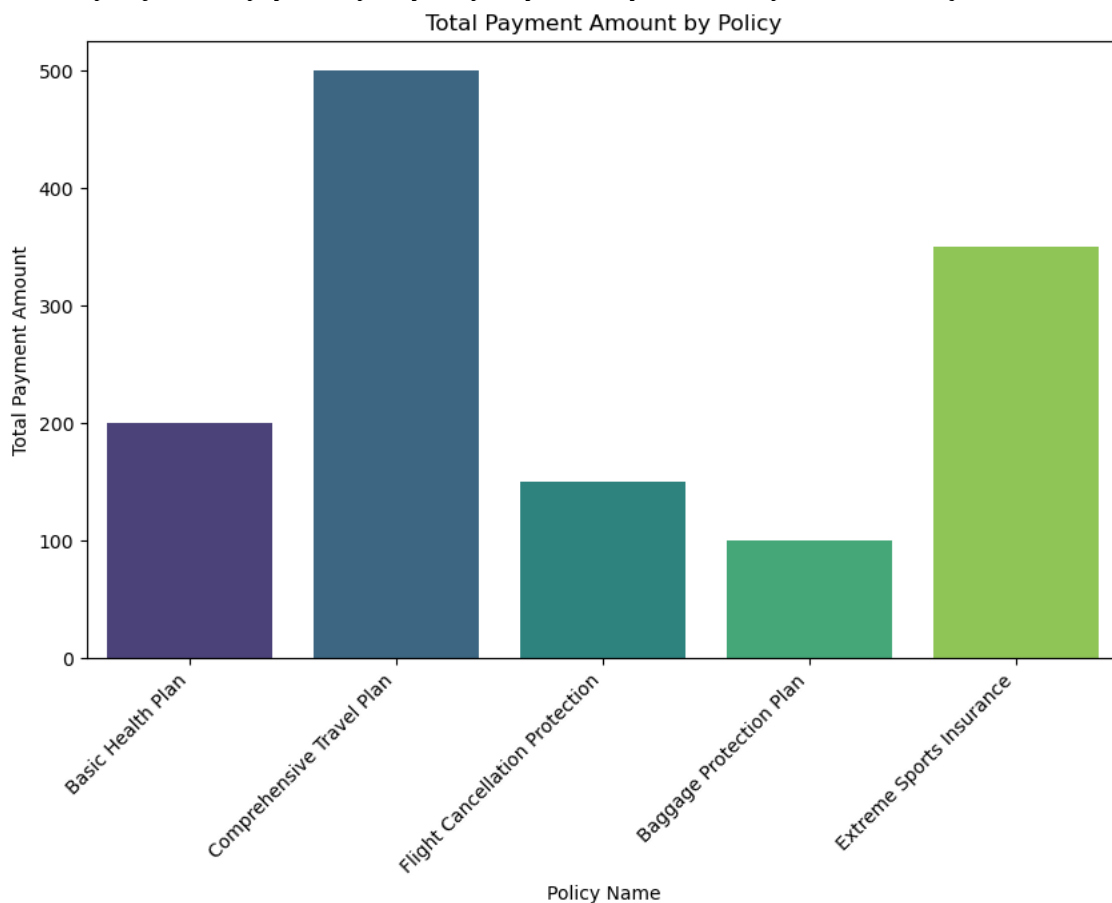
```
plt.xticks(rotation=45, ha='right')
```

```
plt.show()
```

/var/folders/34/jqpjg\_x508j0x978h6stq2j80000gn/T/ipykernel\_5177/779888698.py:23: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(data=df_payment_policy, x='policy_name', y='amount', palette='viridis')
```



To migrate key relational data from the MySQL database of the *Travel Insurance System* to a MongoDB NoSQL structure. The goal is to restructure the dataset for flexibility and scalability, set up MongoDB locally, and run queries to verify successful migration and data retrieval.

## 1. Selection of Tables for Migration

The following MySQL tables were selected for migration based on their relevance and relationships in the system:

- Customers
- Countires\_Offers
- Claims
- Agent\_Sales
- Agents
- Customer\_Policies

## 2. Data Restructuring

Since MongoDB is document-based, the relational structure was converted into embedded documents where applicable:

```
{  
  
  "_id": "cust_001",  
  
  "name": "John Doe",  
  
  "email": "john@example.com",  
  
  "policies": [  
  
    {  
  
      "policy_id": "pol_101",  
  
      "type": "Senior Travel",  
  
      "start_date": "2024-10-01",  
  
      "end_date": "2025-01-01",  
  
      "claims": [  
  
        {  
  
          "claim_id": "cl_501",
```

```

    "amount": 3000,

    "status": "Approved"

  }

]

}

]

}

```

### 3. Query Execution in MongoDB

Verified successful migration with basic and advanced queries:

- **Find all policies of a customer:**

```
db.customers.find( { "name": "John Doe" }, { "policies": 1, "_id": 0 } )
```

List all approved claims:

```

db.customers.aggregate([

  { $unwind: "$policies" },

  { $unwind: "$policies.claims" },

  { $match: { "policies.claims.status": "Approved" } },

  { $project: {

    _id: 0,

    customer: "$name",

    policy: "$policies.policy_id",

    claim: "$policies.claims.claim_id",

    amount: "$policies.claims.amount"

  }}
]

```

D

Total number of claims per customer:

```
db.customers.aggregate([
```

```
{ $project: {
```

name: 1,

```
totalClaims: {
```

$$\text{\$sum: } \{$$
$$\$map: \{$$

```
input: "$policies",
```

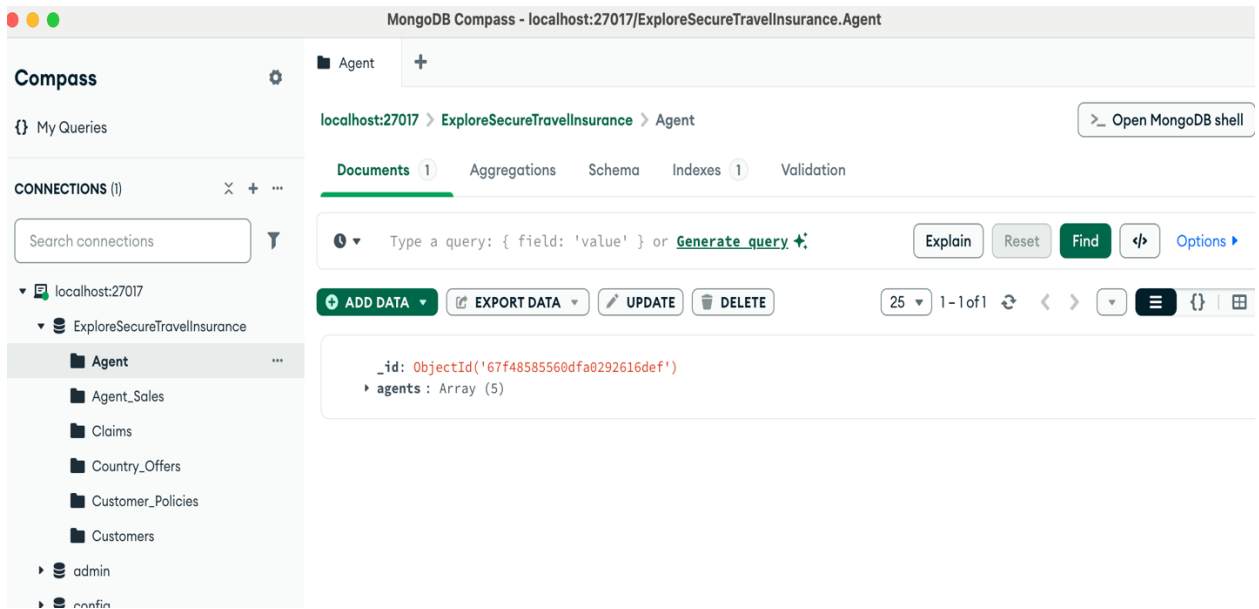
as: "policy",

```
in: { $size: "$$policy.claims" }
```

$$\}$$
$$\}$$
$$\}$$
$$\} \}$$

D





- MongoDB is successfully set up and operational.
- MySQL data has been transformed and migrated to a document-based structure in MongoDB.
- Queries are working as expected, and the data model supports embedded documents for faster access and flexibility.