Travel Agency Management System

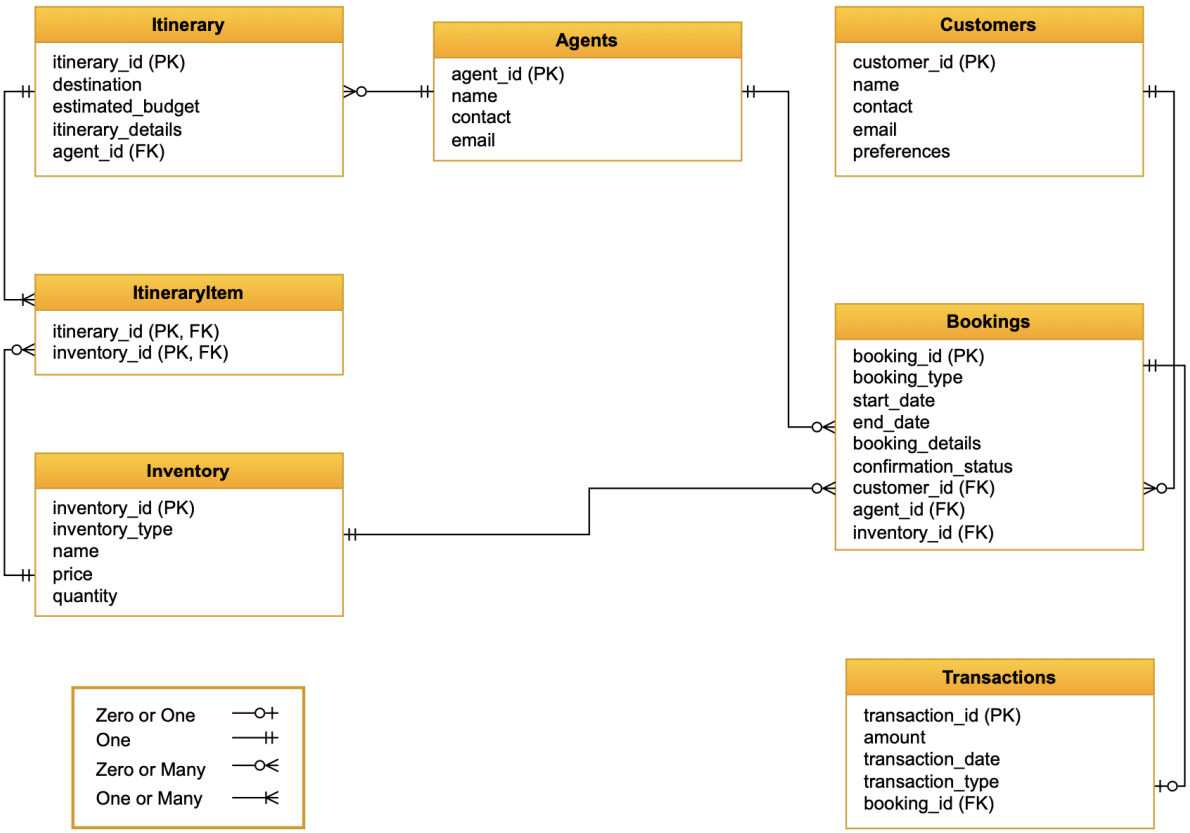
Overview

We designed and developed a comprehensive database management system for a travel agency to efficiently manage its operations, such as customer information, booking details, itinerary management, and financial transactions.

We developed a normalized database, ensuring efficient transactions and benchmarked each type of operation. Additionally, we integrated the ChatCompletion API by OpenAI.

Furthermore, we created a ChatBot to handle all complex query operations, integrating natural language conversion to SQL via AI. This eliminates the need for users to be proficient in SQL.

Entity Relationship Diagram



Stored Procedures

We incorporated the following stored procedures into our database to help AI perform complex operations directly:

Update Agent

For a given agent ID, it can update any of the agent details as mentioned by the user.

Update Customer

For a given customer ID, it can update any of the customer details as mentioned by the user.

Make Payment

For a given booking ID and transaction type (i.e., cash, card, or check), it creates a new transaction and calculates the total amount, which is the actual amount plus transaction fees as per the transaction type, and enters it into the system.

Get Bookings

For a given customer ID and itinerary ID, it provides a list of all bookings made by that customer for the given itinerary if any bookings exist.

Create Booking

For a given customer ID, agent ID, inventory ID, and all necessary details, we can create a booking. Creating a booking also reduces the quantity as soon as the booking is confirmed. Additionally, to make it more functional and realistic, we also adjust the prices when the inventory reduces to a considerable quantity.

Update Single Booking

For a given booking ID, it can update any of the booking details as mentioned by the user.

Cancel Single Booking

For a given booking ID, cancel the booking and update the database.

Cancel Bookings

For a given customer ID and itinerary ID, cancel all the bookings made on any inventory for that particular itinerary selected by the customer.

Database

Insertion

We utilized a Python library called Faker to generate realistic fake data, including names, mobile numbers, emails, company names, random text, etc. Given that we needed to insert tens of thousands of records, we aimed to optimize our approach.

The insertion script only requires the number of records you want for each category, and it generates all the necessary fake data before inserting it into the respective tables. The parameters required for the insertion script are as follows:

- Number of Agents
- Number of Customers
- Number of Itineraries
- Number of Inventories
- Number of Bookings

Benchmarking

For the above-mentioned numerical parameters, we benchmarked the average insertion time per transaction, i.e., the average amount of time required to insert a record into a particular table.

We doubled the number of records for each table each time and found that the average insertion time wasn't affected at all; it always hovered around a certain number for each table.

Case 1

Customers	Agents	Itineraries	Inventories	Bookings
100	100	500	2000	700

```
Average insertion time per agent: 0.000305180549621582 seconds
Average insertion time per customer: 0.00014914989471435548 seconds
Average insertion time per itinerary: 8.591794967651367e-05 seconds
Average insertion time per inventory: 7.459557056427002e-05 seconds
Average insertion time per itinerary item: 7.153796724022515e-05 seconds
Average insertion time per booking: 8.137057809268727e-05 seconds
Average insertion time per transaction: 7.548792432524244e-05 seconds
```

Case 2

Customers	Agents	Itineraries	Inventories	Bookings
200	200	1000	4000	3500

```
Average insertion time per agent: 0.00019000530242919923 seconds
Average insertion time per customer: 8.30996036529541e-05 seconds
Average insertion time per itinerary: 8.206415176391602e-05 seconds
Average insertion time per inventory: 7.169604301452636e-05 seconds
Average insertion time per itinerary item: 6.969430736884483e-05 seconds
Average insertion time per booking: 8.739055906023298e-05 seconds
Average insertion time per transaction: 7.308996796405578e-05 seconds
```

Case 3

Customers	Agents	Itineraries	Inventories	Bookings
400	400	2000	8000	7000

```
Average insertion time per agent: 0.00013036251068115234 seconds
Average insertion time per customer: 7.944762706756592e-05 seconds
Average insertion time per itinerary: 8.102703094482421e-05 seconds
Average insertion time per inventory: 7.069021463394164e-05 seconds
Average insertion time per itinerary item: 6.962618674330784e-05 seconds
Average insertion time per booking: 8.188772201538086e-05 seconds
Average insertion time per transaction: 7.336244415963426e-05 seconds
```

Case 4

Customers	Agents	Itineraries	Inventories	Bookings
800	800	4000	16000	14000

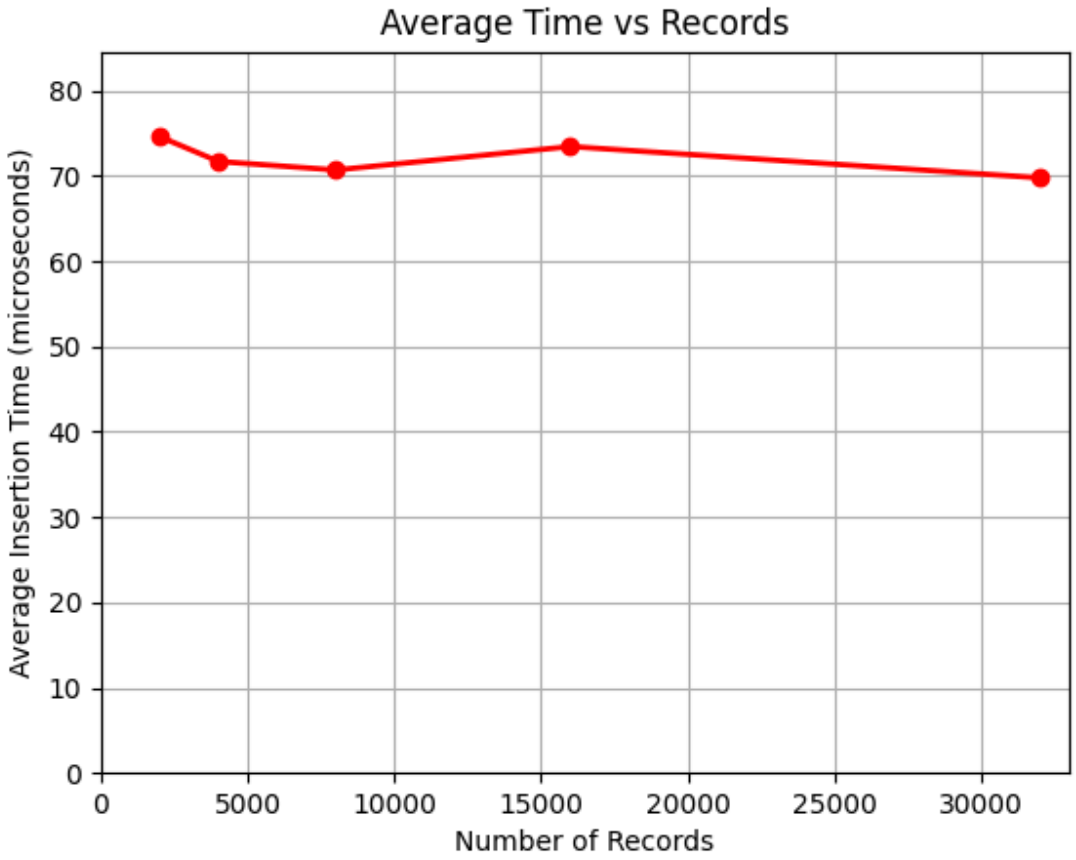
```
Average insertion time per agent: 0.00010303735733032227 seconds
Average insertion time per customer: 8.222281932830811e-05 seconds
Average insertion time per itinerary: 7.601147890090942e-05 seconds
Average insertion time per inventory: 7.344606518745422e-05 seconds
Average insertion time per itinerary item: 6.967848178555224e-05 seconds
Average insertion time per booking: 8.092049189976283e-05 seconds
Average insertion time per transaction: 7.303198404650941e-05 seconds
```

Case 5

Customers	Agents	Itineraries	Inventories	Bookings
1600	1600	8000	32000	28000

```
Average insertion time per agent: 8.315876126289367e-05 seconds
Average insertion time per customer: 7.25911557674408e-05 seconds
Average insertion time per itinerary: 7.55578875541687e-05 seconds
Average insertion time per inventory: 6.977643817663193e-05 seconds
Average insertion time per itinerary item: 6.951668575266439e-05 seconds
Average insertion time per booking: 8.073853594916207e-05 seconds
Average insertion time per transaction: 7.275671100611345e-05 seconds
```

We also generated a graph to observe how the average insertion time changes. Here is the graph showing the number of booking records ranging from 2000 to 32000. As seen in the figure, the average insertion time stayed almost constant, consistently hovering around 70 microseconds.

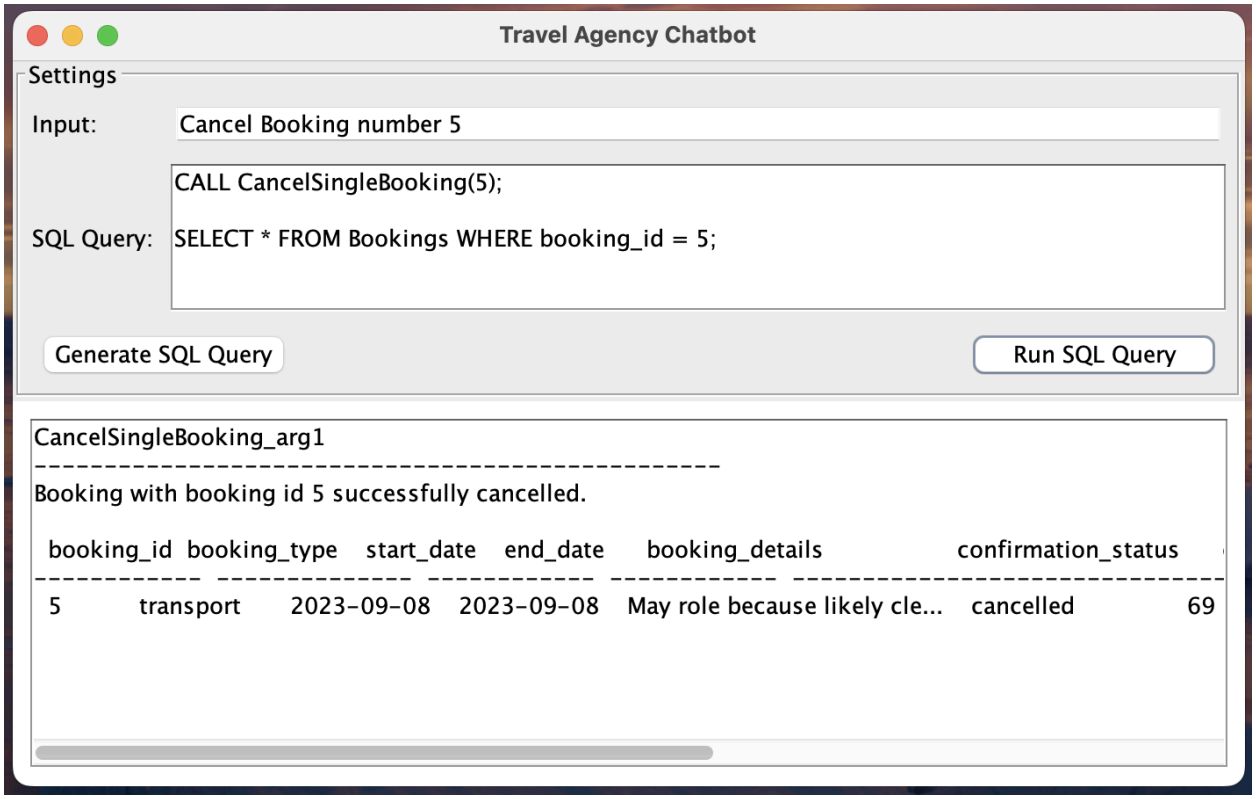


ChatCompletion API Integration

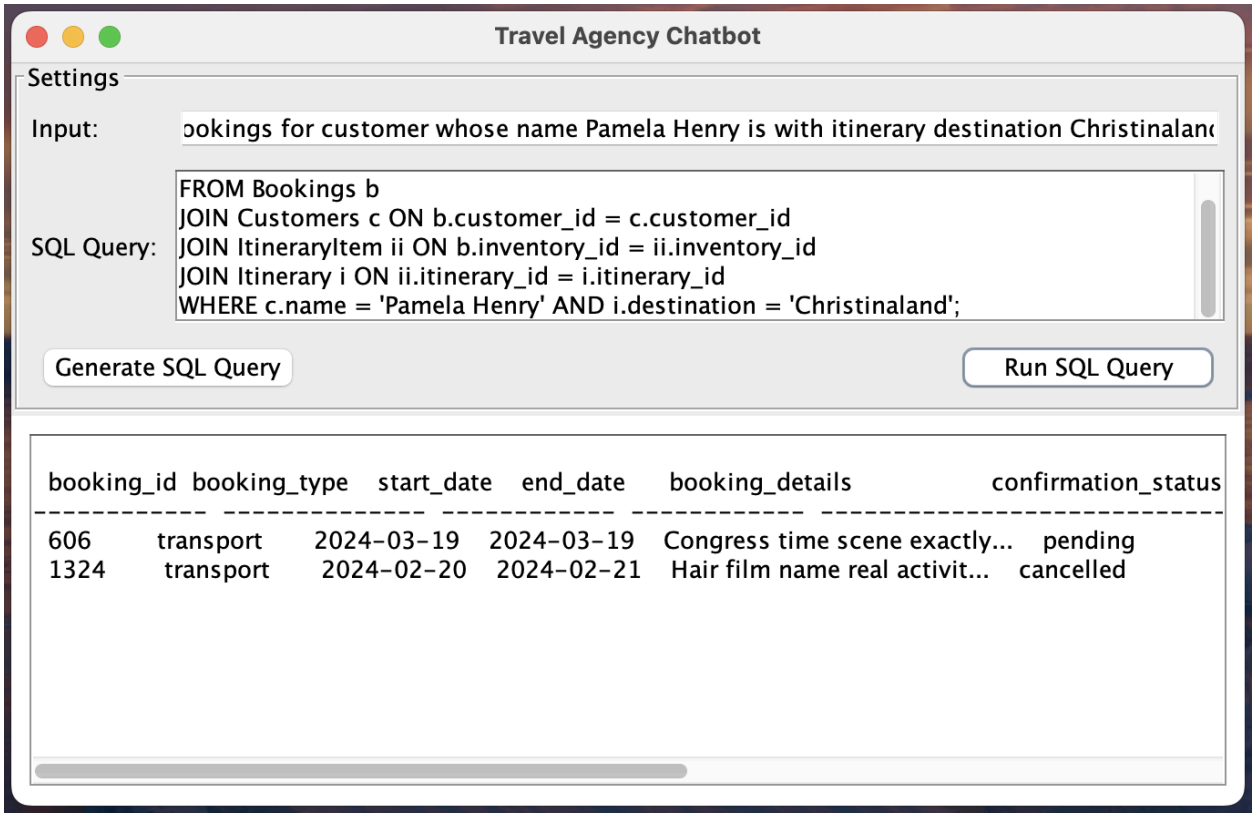
We utilized the ChatCompletion API by OpenAI to generate SQL for a given simple English command or instruction. The ChatCompletion API has the following parameters:

- **Model:** We are currently using the gpt-4-turbo model.
- **Messages:** It can be either System Role or User Role. The System role contains all essential context for AI, and User role guides the AI's actions and outputs. Hence, we have provided all the parsed SQL files such as the database structure and stored procedures into the System content. On the other hand, the user's English input is the User content.
- **Temperature:** This controls the randomness of the generated response, ranging from 0 to 1. As we want it to be strictly controlled to our database, we set it to 0.
- **Max_tokens:** This sets the maximum length of the generated response. We set it to 500.

Example 1: Calling a Stored Procedure



Example 2: Performing a Complex Query



Future Scope

The ChatCompletion API is not always accurate. There is definitely a need for improvement and optimization to generate correct responses. With improved accuracy, we can transition towards a better chat-centric UI. This can be achieved by trying out different LLMs or creating one which is more database-centric. Hence, for now, we are providing the query manipulation feature for user flexibility. In the future, we can remove the query manipulation feature for a more streamlined, standalone chatbot experience.

Wireframe

Figma Wireframe

References

- https://platform.openai.com/docs/models/gpt-4-turbo-and-gpt-4
- https://platform.openai.com/docs/api-reference/chat/create
- https://faker.readthedocs.io/en/master/
- https://pypi.org/project/tabulate/
- https://matplotlib.org/stable/users/explain/animations/animations.html