

Project

Flight Booking System

Course

Data Intensive Systems

Team: Team Intensive

Name	:	Md Aminul Islam
Email	:	aminul.islam@student.lut.fi
Name	:	Md Raihan Uddin
Email	:	md.uddin@student.lut.fi
Name	:	Arsalan Khan
Email	:	arsalan.a.khan@student.lut.fi
Name	:	Md Iftakhar Jahan
Email	:	md.jahan@student.lut.fi

1. Introduction	3
2. WEB GUI	4
2.1. Login Page	4
2.2. Registration Form	5
2.3. Flight Search	5
2.4. Flight Reservation	6
2.5. Flight Booking	7
2.6. Flight Cancelation	7
3. High-level network model	8
4. Justification of DBMS Architecture Selection	9
4.1. Homogeneous Multiple DBMS	9
4.2. ANSI/SPARC Architecture	10
5. Global and Local Conceptual Schemas	11
6. Top-down Design Strategy	13
6.1. Fragmentation Strategy Rationale and Degree	13
6.2. Correctness Rules of Fragmentation	17
6.3. Data Allocation and Replication Strategy	18
6.4. Cost model	18
7. Transparency	19
8. Local Autonomy	20
9. Data Integration and Access Control	20
9.1. Data Integration	20
9.2. Access Control	20
10. Query Execution	21
10.1. Registering Passenger	22
10.2. Registering Manager on the System	22
10.3. User Account Login	23
10.4. Flight Information Searching	23
10.5. Flight Booking	24
10.6. Flight Cancelations	24
11. Concurrency and Reliability	24
12. Data Security	26
12.1. Infrastructure Security	26
12.2. Service Authentication	26
12.3. User Authentication	27
13. System Expansion	27
Appendix	28

1. Introduction

Air travel is quite prevalent and therefore one of the most common ways of transportation. People who prefer to travel by air today have a variety of airlines and departure times from which to pick. In addition, airline businesses face intense competition in today's market, and customers expect a service that is uncomplicated and streamlined. For this reason, the purpose of our project is to design and develop an application that will automate major airline operations. This includes providing the facilities for the online reservation of air tickets; flight searching; flight scheduling; flight canceling; or any other operations through a practical one-stop solution utilizing a user-friendly interface for a regular passenger who intends to travel through the airways. Specifically, this app is for a regular passenger who wants to fly.

Airline ticketing is more complex than it may seem. It's a complicated process with a lot of systems, interactions, and rules. For example, the database needs to keep track of the flight schedule, the total number of planes, which planes are scheduled for which routes, information about the people on each flight, information about booking and reserving tickets, and so on. These kinds of things happen all the time around the world, and the database needs to keep track of them all without breaking them.

The vast majority of the currently available applications manage their business processes through the use of centralized database systems. Because of this, customers in different regions do not experience the same level of performance, and the service may be interrupted if a server goes down.

To make our service faster, more transparent, and more reliable, we use a distributed database system on our application in which all data can be stored in a different country and multiple machines on a standard communication network. As the airline ticketing system is a global service, a distributed database system ensures the performance and transparency of the service without any interruptions around the globe at an identical speed. Furthermore, if one server goes down, another server will take its place and make the data available.

Flight Ticketing Distributed System Functional Requirements

The proposed system is designed in a way that passengers can get a one-stop service without interruptions through a friendly user interface. Given the initial prerequisites and conditions, the platform should include the following functionality:

- **Registration:** Registering passengers and branch managers within the system user interface.
- **Login:** The registered users have a separate dashboard for following their activities.
- **Flight searching:** People can search for their desired flight with a friendly user interface.
- **Ticket booking:** Passenger can book their ticket Through the system
- **Ticket reservation:** Registered users can reserve a ticket for a period of time without payment.

- **Ticket cancellation:** Passengers can cancel their tickets by following rules and regulations.
- **Ticket rescheduling:** Passengers can reschedule their tickets by following rules and regulations.

2. WEB GUI

In this section, the prototype's web interfaces will be shown, along with a description of how they work and a list of expected improvements for the following stages of project development.

2.1. Login Page

Here passengers will sign in their account and after signing in they will be able to search, reserve and book a flight. If passengers have no account then they can go to registration by clicking sign up button.

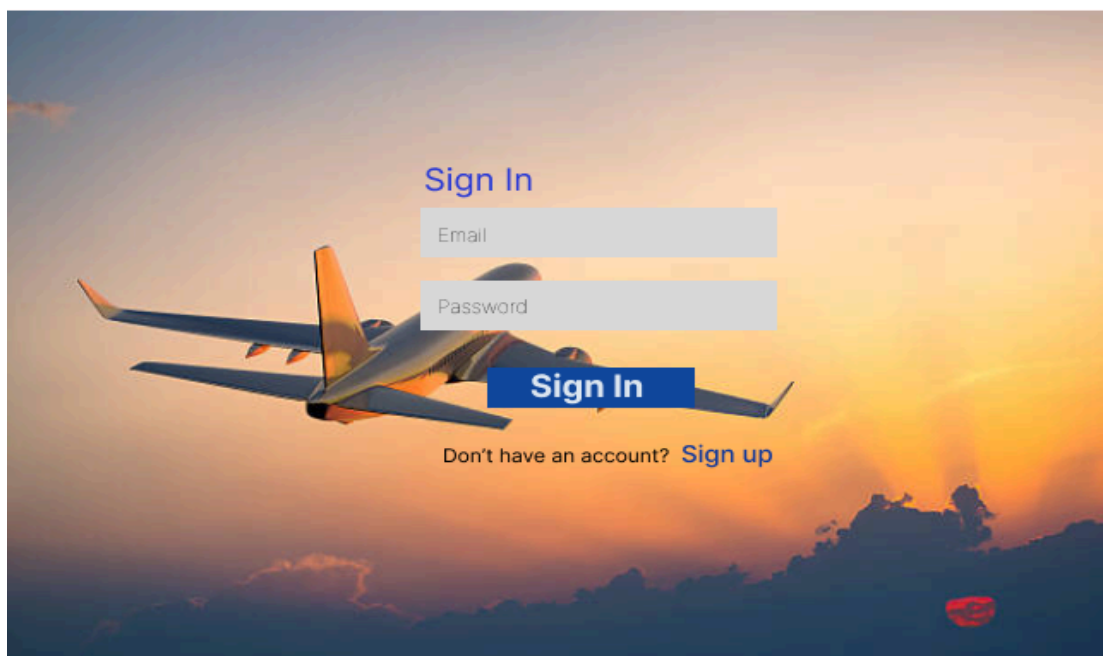
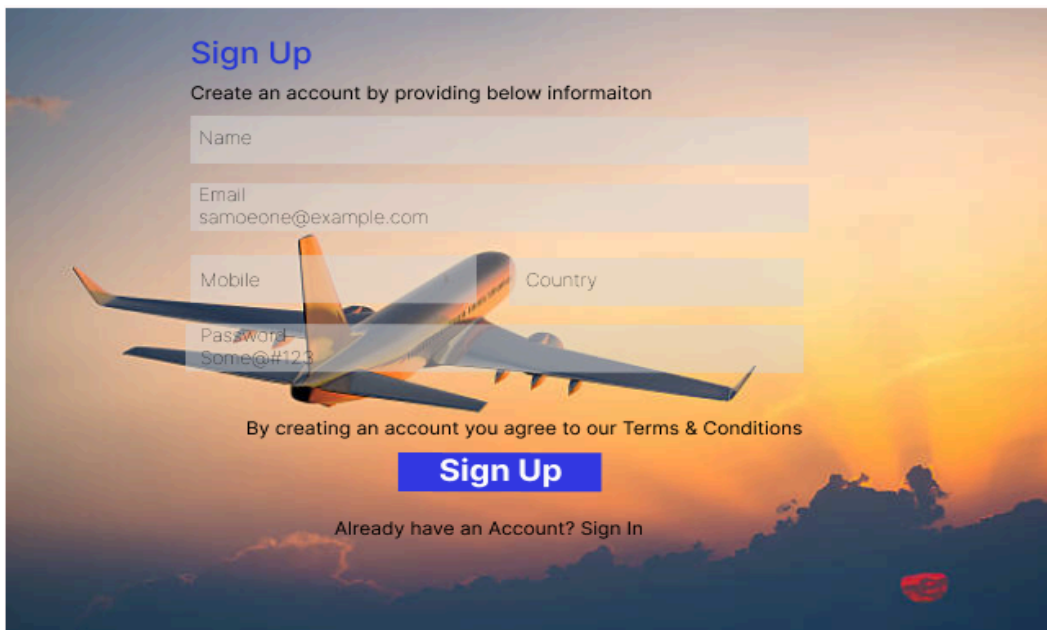


Figure 1. Login Page UI

2.2. Registration Form

Here passengers will give their information. After successfully registering with their information they can login their account.



The registration form is overlaid on a background image of an airplane flying over a sunset sky. The form includes a title 'Sign Up', a subtitle 'Create an account by providing below informaiton', and five input fields: 'Name', 'Email' (with the placeholder 'samoeone@example.com'), 'Mobile', 'Country', and 'Password' (with the placeholder 'Some@#123'). Below the fields is a 'Sign Up' button, a link 'Already have an Account? Sign In', and a 'Terms & Conditions' link.

Sign Up

Create an account by providing below informaiton

Name

Email
samoeone@example.com

Mobile

Country

Password
Some@#123

By creating an account you agree to our Terms & Conditions

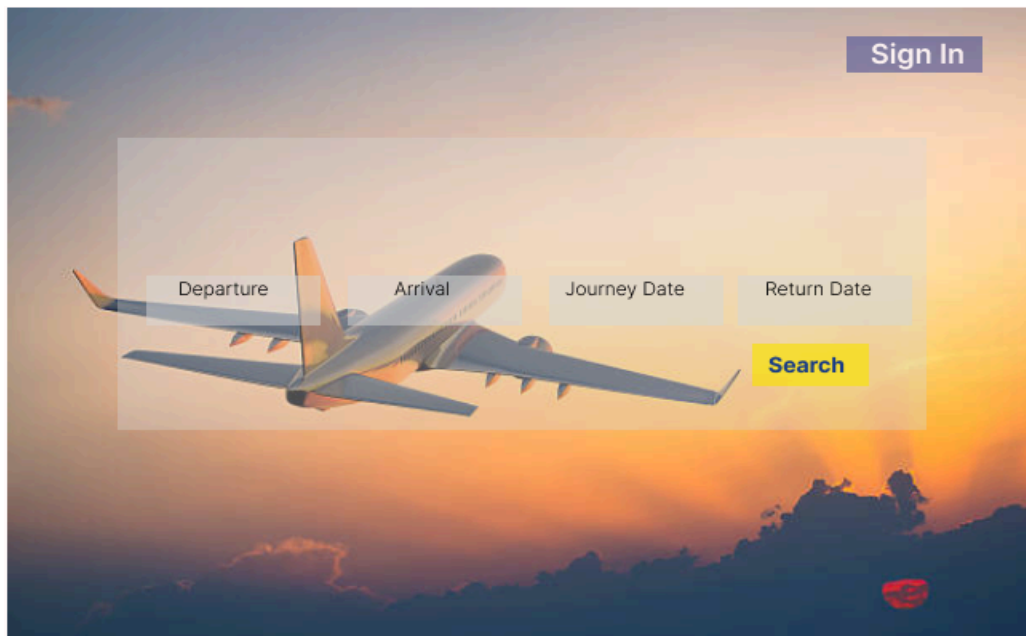
Sign Up

Already have an Account? Sign In

Figure 2. Registration Form UI

2.3. Flight Search

After successfully login passengers can search their desired flight based on departure location, arrival location, departurture time and return time.



The flight search form is overlaid on the same sunset background as the registration form. It includes a 'Sign In' button in the top right corner. The search form itself has four input fields: 'Departure', 'Arrival', 'Journey Date', and 'Return Date'. A yellow 'Search' button is located to the right of these fields.

Sign In

Departure

Arrival

Journey Date

Return Date

Search

Figure 3. Flight Search UI

2.4. Flight Reservation

Before booking a flight passengers can reserve a seat for a certain period of time. Based on their search they will appear in this page and they will check flight information like flight time, fare , flight duration and so on.

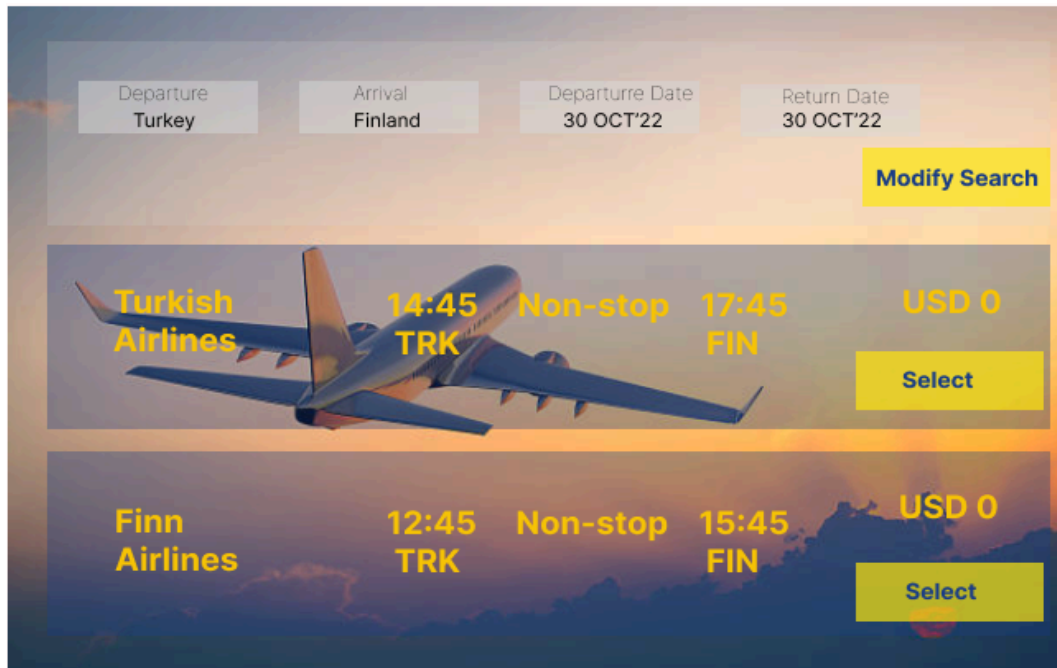


Figure 4. Flight Reservation UI

2.5. Flight Booking

This interface is accessed by clicking on the ticket reservations link button and after selecting the desired flight the summary of the ticket will appear here. Finally, if they want to book a flight they can do it here.

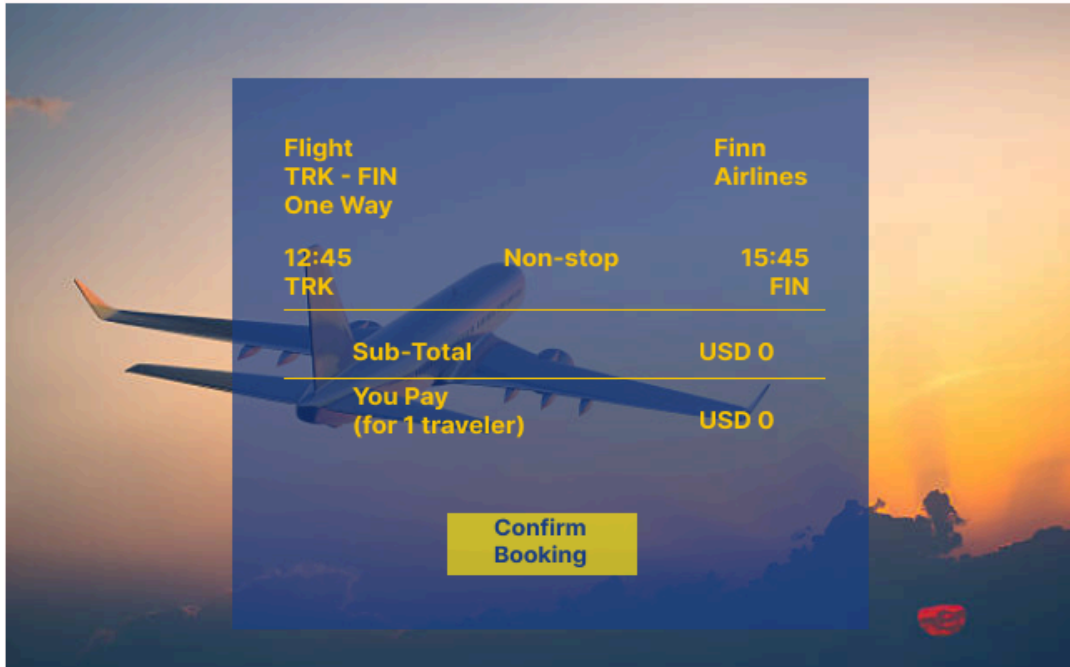


Figure 5. Flight Booking UI

2.6. Flight Cancellation

After booking the flight, passengers can cancel the flight from here any time.

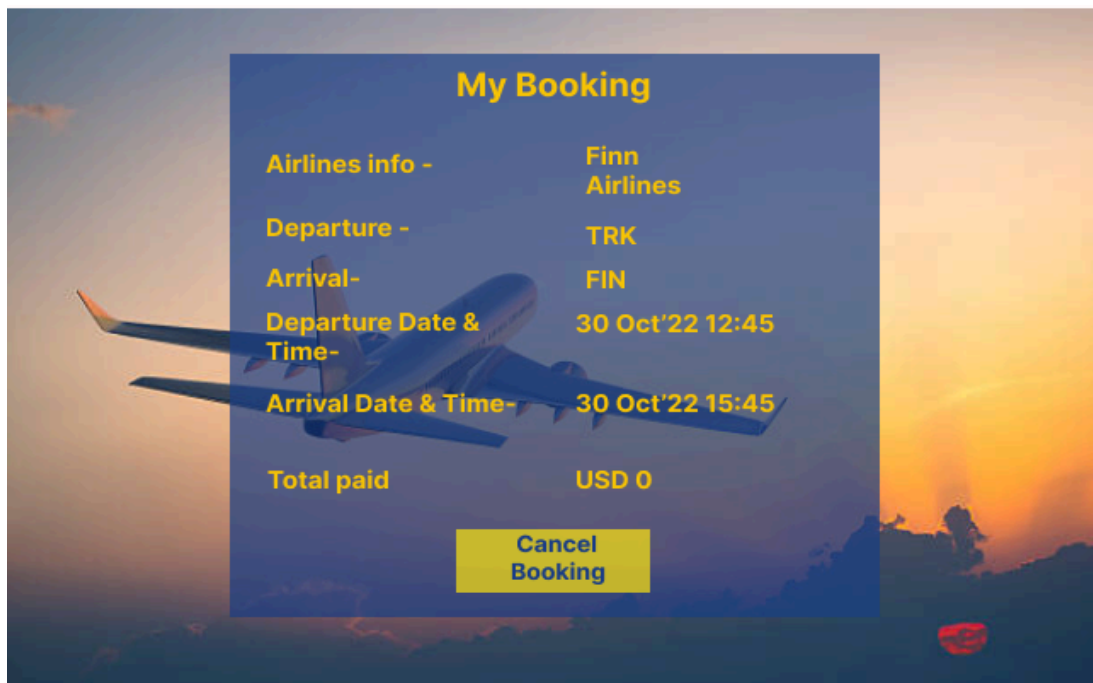


Figure 6. Flight Cancellation UI

3. High-level network model

Traveler or passengers can book ticket from the interface and will be able to see from anywhere of the world. Within a country, there could be multiple site office. Therefore, data will be distributed throughout the country sites or databases also replicated all the site offices within the country. If a single site fails, it does not impact the entire system, as is the case with a centralized database system.

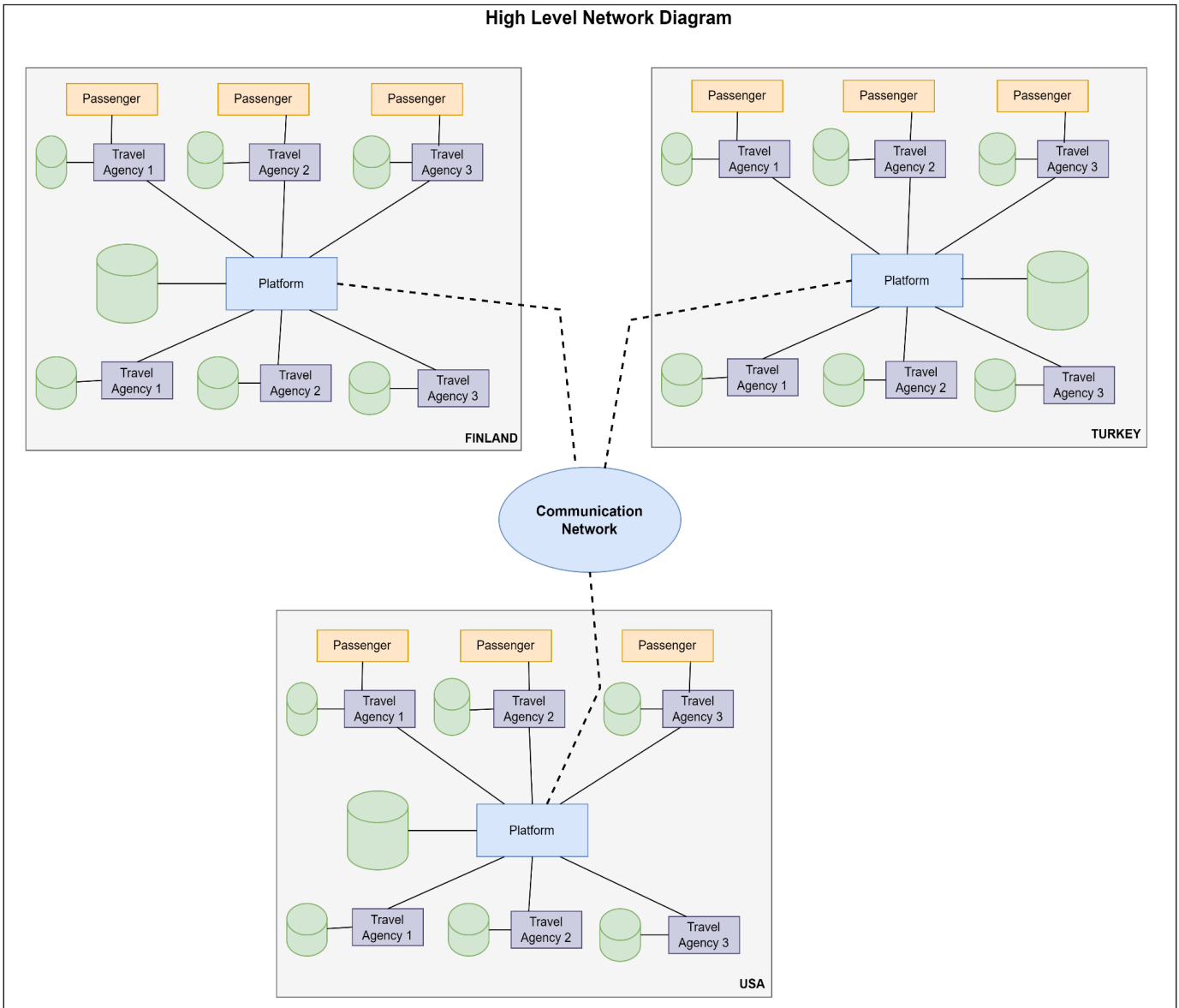


Figure 7. High-level Network Model of Flight Ticket Booking System

4. Justification of DBMS Architecture Selection

4.1. Homogeneous Multiple DBMS

In the proposed architecture for the airline's flight ticketing system, all the sites use identical DBMS and operating systems. Because there will be no external data or system integration required. Hence, the database will be accessed through a single interface as if it is a single database. This will reduce the complexity of the system and also improve its maintainability.

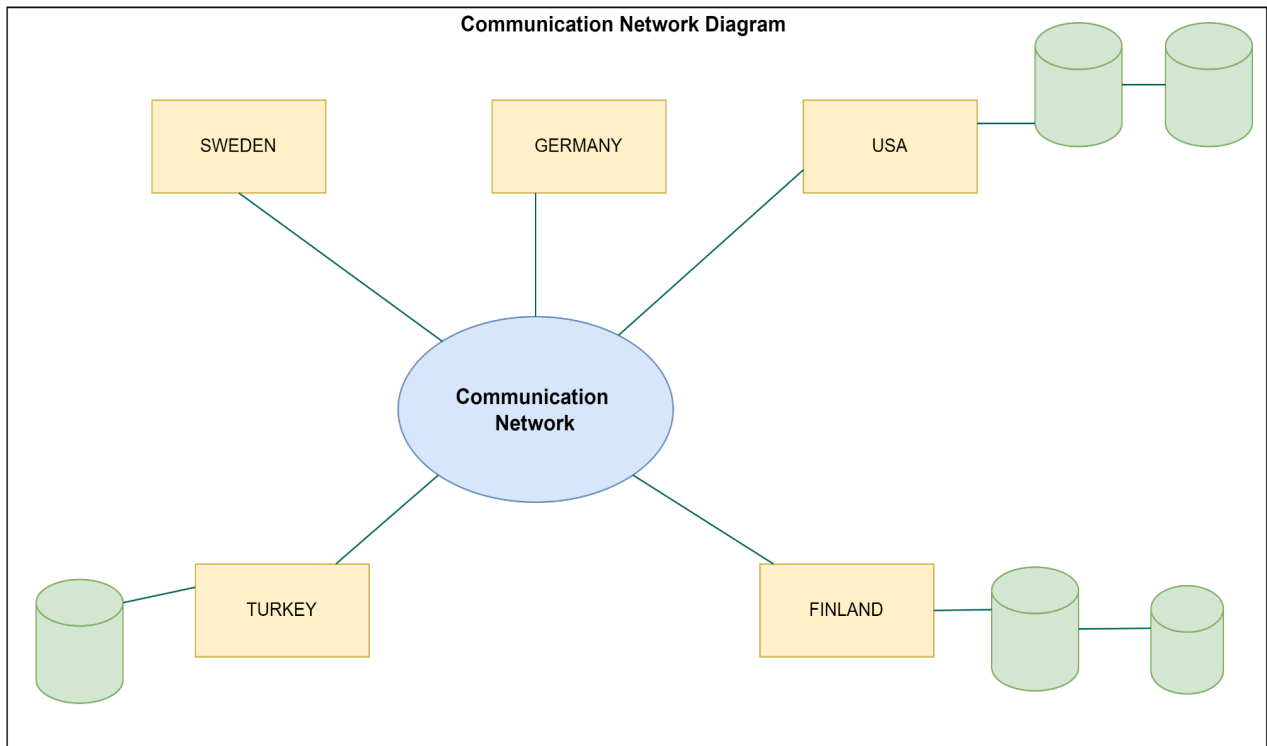


Figure 8. Communication Network Diagram

4.2. ANSI/SPARC Architecture

This airline's flight ticketing system is globally separated and should have a global-level conceptual schema. Then, a fragmentation schema is required for data fragmentation and distribution. Allocation schema will be responsible for data allocation to the sites.

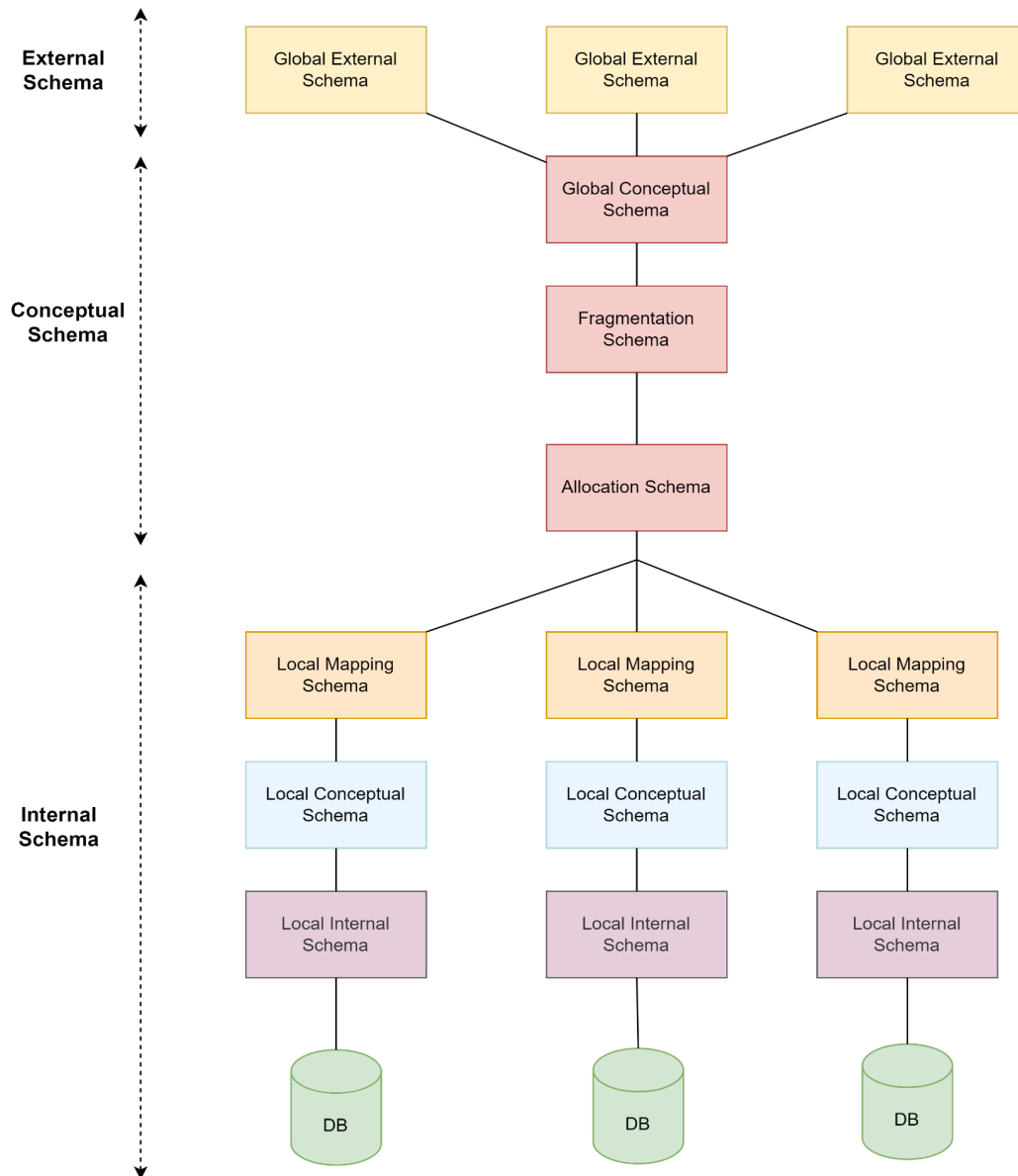


Figure 9. ANSI/SPARC Architecture

5. Global and Local Conceptual Schemas

The strategy is application software doesn't need to connect to each local site. Application will access data using a global schema. Hence, you don't need to know where data is stored, that will ensure location transparency. For retrieving data, some global joins may happen based on the demand of the user and data availability.

In our flight ticket booking system, Global Conceptual Schema (GCS) illustrates the country specific information. All the other relations will bind to the Local Conceptual Schemas (LCS).

The Travel Agency may have multiple branches in a single country. Within the country, all the branches are considered local branches or site offices. Thus, it's a very logical and coherent understanding and also convenient that data will be more available in the country level operations. Each local office or site will be connected to the database that will be managed locally. The system is multi-database and presented in the architecture (section 4). The application must explicitly connect to each site. The Local Schemas are mapped to the Global Schema.

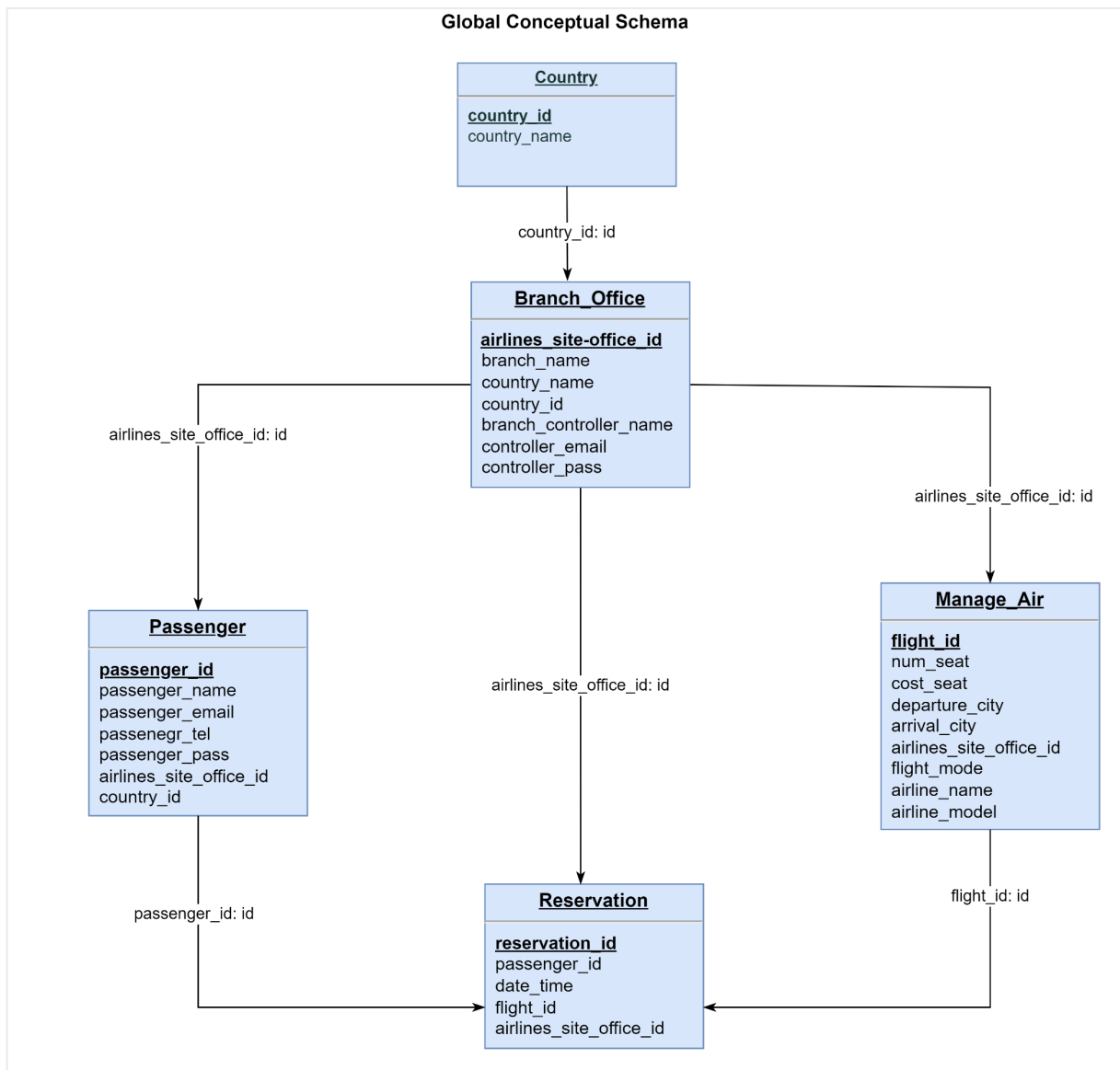


Figure 10. Global Conceptual Schema

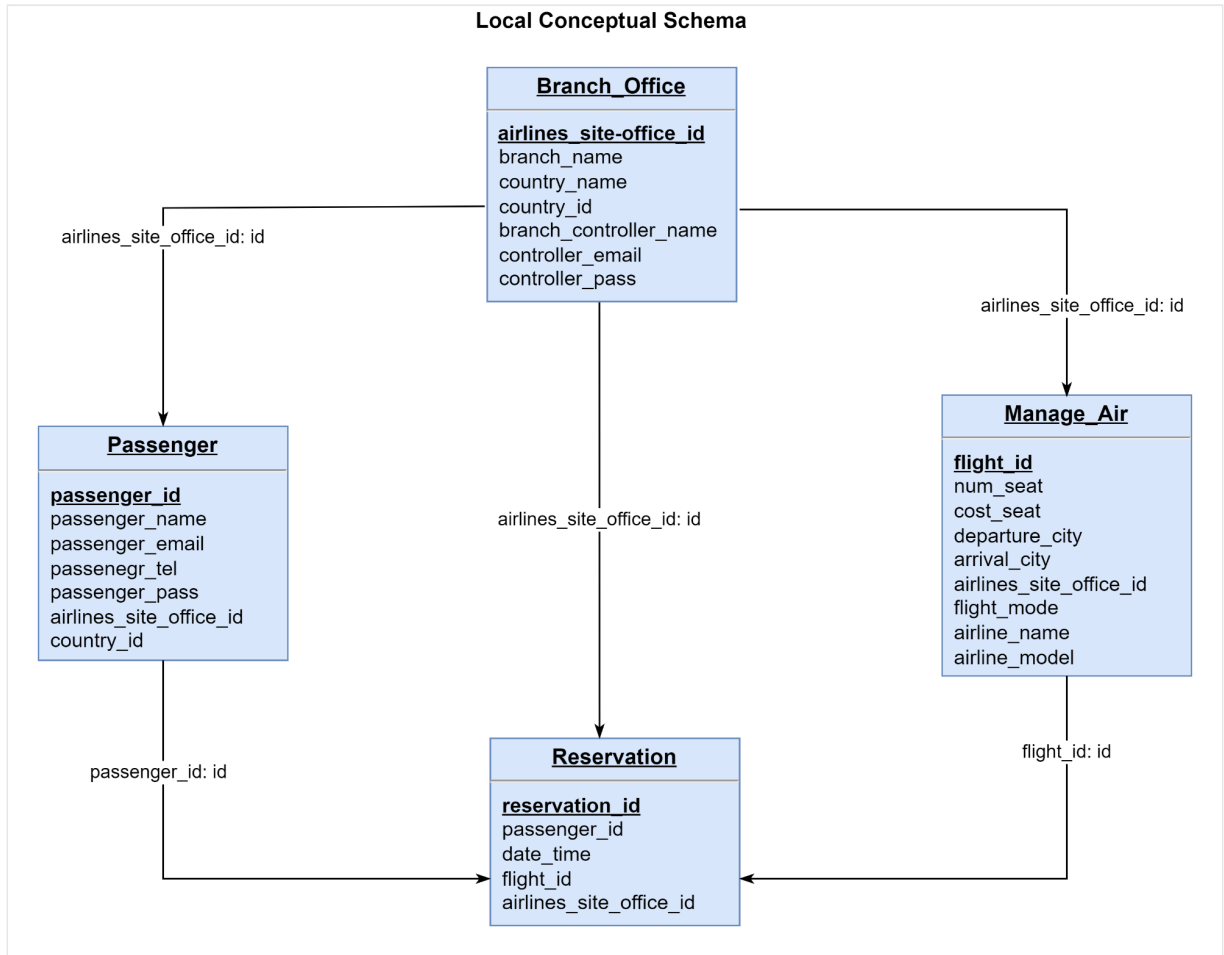


Figure 11. Local Conceptual Schema

6. Top-down Design Strategy

6.1. Fragmentation Strategy Rationale and Degree

Fragmentation Strategy

According to the research, in a distributed system 95% of data users are local users, and the rest of the 5% are global. For example, in a banking system in a local database, most of the hit comes from local customers.

In the flight ticket booking system, for fragmentation, we considered mostly three things:

1. The application will be using the data
2. Query execution in the databases
3. Different Query workload
4. Availability and Safety

Detailed analysis given (section 10).

Based on this analogy and understanding, ensuring the availability and minimizing communication costs or response time, country level data will be stored within the country. For global support, local level data will be integrated into the global schema.

The Fragmentation Algorithm is as follows

Based on the strategy proposed above, here are the fragmentation steps:

1. At the global level, 'country' attributes will be fragmented by country using primary horizontal fragmentation.
2. And then based on the 'country' fragmentation, Branch_Office table will be fragmented based on the site office id as each site id represents one site office identity.
3. To fragment the tuples in other tables (Passenger, Manage_Air and Reservation table) primary horizontal fragmentation will be on the Branch_Office fragmented table based on the site office id
4. These site office data will be stored locally. After that, all the other tables Passenger, Manage_Air and Reservation table will be fragmented by applying derived horizontal fragmentation.

Example Fragmentation

In global level schema, 'country' is the owner table or entity

In local level schema, 'Branch_Office' is the owner table others are member table

DHF

PHF of Country Entity:

Country₁ = $\sigma(\text{country}=\text{"USA"})\text{(Country)}$

select * from country where country="USA";

Country₂ = $\sigma(\text{country}=\text{"FIN"})\text{(Country)}$

select * from country where country="FIN";

Country₃ = $\sigma(\text{country}=\text{"TUR"})\text{(Country)}$

select * from country where country="TUR";

Semi-Join

$\text{Branch_Office}_1 = \text{Branch_Office} \times \text{Country}_1$

$\text{Branch_Office}_2 = \text{Branch_Office} \times \text{Country}_2$

$\text{Branch_Office}_3 = \text{Branch_Office} \times \text{Country}_3$

select * from country inner join branch_office on country.idcountry = branch_office.country_id
where country.country_name='USA';

```
mysql> select * from country inner join branch_office on country.idcountry = branch_office.country_id where country.country_name='USA';
+-----+-----+-----+-----+-----+-----+
| idcountry | country_name | idbranch_office | branch_name | country_id | country_name |
+-----+-----+-----+-----+-----+-----+
| 1 | USA | 1 | Newyork | 1 | USA |
| 1 | USA | 2 | Washington | 1 | USA |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.02 sec)
```

Figure 12. Country Table Fragmentation

PHF of Branch_Office

$\text{Branch_Office}_{11} = \sigma(\text{branch_name}="Newyork")(\text{Branch_Office}_1)$

select * from country where branch_name="Newyork";

$\text{Branch_Office}_{12} = \sigma(\text{branch_name}="Washington")(\text{Branch_Office}_1)$

select * from country where branch_name="Washington";

$\text{Branch_Office}_{21} = \sigma(\text{branch_name}="Helsinki")(\text{Branch_Office}_2)$

select * from country where branch_name="Helsinki";

$\text{Branch_Office}_{22} = \sigma(\text{branch_name}="Lappenranta")(\text{Branch_Office}_2)$

select * from country where branch_name="Lappenranta";

$\text{Branch_Office}_{31} = \sigma(\text{branch_name}="Istambul")(\text{Branch_Office}_3)$

select * from country where branch_name="Istambul";

Semi-Join on Passenger

$\text{Passenger}_{11} = \text{Passenger} \times \text{Branch_Office}_{11}$

$\text{Passenger}_{12} = \text{Passenger} \times \text{Branch_Office}_{12}$

$\text{Passenger}_{21} = \text{Passenger} \times \text{Branch_Office}_{21}$
 $\text{Passenger}_{22} = \text{Passenger} \times \text{Branch_Office}_{22}$
 $\text{Passenger}_{31} = \text{Passenger} \times \text{Branch_Office}_{31}$

select * from passenger join branch_office on branch_office.idbranch_office =
 passenger.agency_site_office_id where passenger.agency_site_office_id = '1';

```
mysql> select * from passenger join branch_office on branch_office.idbranch_office = passenger.agency_site_office_id where passenger.agency_site_office_id = '1';
```

idpassenger	passenger_name	passenger_email	passenger_tel	agency_site_office_id	country_id	idbranch_office	branch_name	country_id	country_name
1	Isru	isru@gmail.com	1234	1	1	1	Newyork	1	USA
2	Knkur	knkur@gmail.com	3456	1	1	1	Newyork	1	USA
3	Mnkur	mknkur@gmail.com	45656	1	1	1	Newyork	1	USA

3 rows in set (0.00 sec)

Figure 13. Passenger Table Fragmentation

Semi-Join on Manage_Air

$\text{Manage_Air}_{11} = \text{Manage_Air} \times \text{Branch_Office}_{11}$
 $\text{Manage_Air}_{12} = \text{Manage_Air} \times \text{Branch_Office}_{11}$
 $\text{Manage_Air}_{21} = \text{Manage_Air} \times \text{Branch_Office}_{11}$
 $\text{Manage_Air}_{22} = \text{Manage_Air} \times \text{Branch_Office}_{11}$
 $\text{Manage_Air}_{31} = \text{Manage_Air} \times \text{Branch_Office}_{11}$

Semi-Join on Reservation

$\text{Reservation}_{11} = \text{Reservation} \times \text{Branch_Office}_{11}$
 $\text{Reservation}_{12} = \text{Reservation} \times \text{Branch_Office}_{11}$
 $\text{Reservation}_{21} = \text{Reservation} \times \text{Branch_Office}_{11}$
 $\text{Reservation}_{22} = \text{Reservation} \times \text{Branch_Office}_{11}$
 $\text{Reservation}_{31} = \text{Reservation} \times \text{Branch_Office}_{11}$

select * from country inner join branch_office on country.idcountry = branch_office.country_id
 where country.country_name='USA';

Select * from passenger join branch_office on branch_office.idbranch_office =
 passenger.agency_site_office_id where passenger.agency_site_office_id = 'Newyork';

select * from passenger join branch_office on branch_office.idbranch_office =
 passenger.agency_site_office_id where passenger.agency_site_office_id = 1;

Select * from passenger join branch_office on btanch_office.idbranch_office =
 passenger.agency_site_office_id = 2;

6.2. Correctness Rules of Fragmentation

Completeness

Decomposition of relation R into fragments R1, R2, ..., Rn is complete if and only if each data item in R can also be found in some Ri.

In the example above (6.1 Example Fragmentation), 'Branch_Office' entity has been fragmented into five different fragmented table because all the tuples are different based on the predicate. Each of the data items from 'Branch_Office' can be found in at least one of the fragmented tables. This rule ensures that we have not lost any records during the process of fragmentation. Thus, this horizontal fragmentation is complete. Same analogy is applicable for all the fragmentation steps proposed, and no record will be lost.

Reconstruction

After combining using 'union' operation of all the fragmented piece of data, original table can be formed. For example,

```
(select * from country where branch_name="Newyork")
Union
(select * from country where branch_name="Washington")
Union
(select * from country where branch_name="Helsinki")
Union
(select * from country where branch_name="Lappenranta")
Union
(select * from country where branch_name="Istambul")
```

The union operation of the query above will give the actual 'Branch_Office' table.

The same analogy can be applied for all the horizontal fragmentation applied above.

Disjointness

When we perform an Intersect operation between all the above fragments, we will get a set of results or records that are common for all the fragments. For example,

```
(select * from country where branch_name="Newyork")
Intersection
(select * from country where branch_name="Washington")
Intersection
(select * from country where branch_name="Helsinki")
Intersection
(select * from country where branch_name="Lappenranta")
Intersection
(select * from country where branch_name="Istambul")
```


The intersection operation of the query above gives empty set as result.

So we can say that here the disjointness property is satisfied.

6.3. Data Allocation and Replication Strategy

This proposed flight ticket booking system used only horizontal fragmentation for data allocation purpose that discussed in the section 6.1. Correctness rules also discussed that ensures the fragmentation validity. Fragmentation results in the parallel execution of a single query by dividing it into a set of sub queries that operate on fragments. Thus, fragmentation increases the level of concurrency of this system.

Replication strategy ensures that replication of databases are hidden from the users. It enables users to query upon a table as if only a single copy of the table exists. To ensure the availability and performance, we applied the following data allocation and replication strategy:

Replication depending on the ratio of the read-only queries to the update queries:

In a local level, full replication will be applied to ensure the high availability and reliability of service. For example, in the proposed system inside a country data will be fully replicated in all the servers.

However, in the global level partial replication will be applied because most of the local data users are from local users. In summary, this system adopts full and partial replication both in the context.

Example Replication

‘Reservation’ entity has been fragmented based on the fragmentation ‘Branch_Office’ entity. Now within a country there may be several branch offices and from any place of the country user may hit the required data related to his/her. Passengers may want to update his/her profile information based on that reservation information also be required to update. Thus, passenger and corresponding reservation information should be available, it’s obvious that if data replicated in each side in local level that will increase the availability.

Schema images and fragmentation can be found in Section 6.1

6.4. Cost model

We use the following formula to compute the amount of time needed for processing:

$$Total\ time = T_{CPU} * \#insts + T_{I/O} * \#I/Os + T_{MSG} * \#msgs + T_{TR} * \#bytes$$

- T_{CPU} is the time of a CPU instruction.
- $T_{I/O}$ is the time of a disk I/O.
- T_{MSG} is the fixed time of initiating and receiving a message.

- T_{TR} is the time it takes to transmit a data unit from one site to another.

Let us compute the entire amount of time required to transmit one "case" because the "case" is the data piece that takes up the most space in the system's database. After this calculation, it will be possible to give an estimate and a specific number for the most weight the platform can hold.

The results of the hardware testing showed that T_{CPU} and $T_{I/O}$ take up the same amount of time, which is two milliseconds. However, given that the case is loaded simultaneously on three devices and that there are three I/O channels, $T_{I/O}$ needs to be multiplied by 3. The T_{MSG} value can range from 1 to 5 milliseconds; nevertheless, let's take the average of this range and call it 0.0030 seconds. Due to the fact that the computations are only carried out for one data unit, the total number of messages (# msgs) is equal to 1.

Let assume, One data unit is approximately 1000 kilobytes in size. The average Internet speed in the United States, Turkey, and Finland are roughly 50 megabits per second. The packet size equals 8.06 megabits after converting the flies' size kilobytes to megabits. The complete case will take approximately 0.1612 seconds to transfer from one site to another site. The total time equals 0.16 seconds when all values are added together.

Load Calculation

It is very necessary to carry out extra research in order to ascertain the utmost amount of load that can be supported by the platform.

Let's assume,

1. Annually 300,000 passenger travel on the route between the USA and Turkey
3. Out of that 1000,000 travelers utilize our platform throughout the course of the year
4. There will be approximately $(1000,000/365=)$ 2740 passenger application processes carried out each and every day.
5. In the event that this quantity of applications is simultaneously loaded into the system, it will take approximately 10 hours to process them.

Hence, the system is able to manage all requests at peak load for a period of twenty-four hours. This is an estimate of the maximum load, which is based on the assumption that there are five servers available for every passenger traveling along this route. Two of these servers are located in Finland, two are located in the United States, and one is located in Turkey. can accommodate, thereby making the performance both quicker and more reliable.

7. Transparency

Transparency in DDBMS refers to the system's transparent delivery of information to the user. It aids in the concealment of information that the user is to apply.

- **DBMS Transparency:** It conceals the fact that the local DBMS may differ. In this, flight ticketing system followed, logical data independence and physical independence. For logical independence, local and global schema has been introduced (section 5). And physical independence followed the physical storage mechanism, retrieving results and presentation to end user- hiding the details of the storage structure from user applications.
- **Distribution Transparency:** The proposed system dividing each database relation into smaller fragments and treat each fragment as a separate database object. For example, in this proposed system primary horizontal fragmentation and derived horizontal fragmentation has been applied (section 6.1). Fragmentation Transparency hides the fact that the table the user is querying on is actually a fragment or union of some fragments. When a DDBMS exhibits distribution data transparency, the user is not required to be aware that the data is fragmented and may instead perceive the database as a single object or logical entity.

8. Local Autonomy

In our distributed database system, each site's activities are self-contained and not reliant on other sites. Local data security, integrity, and storage representation will be within the authority of the local site. All sites are treated equally. If a single site fails, other sites will be able to continue their operation without any issues.

9. Data Integration and Access Control

9.1. Data Integration

Typical flight booking system connected through multiple airlines. There are different systems, different formats, and different databases for each airline. So they basically use ETL (extract-transform-load) mapping for data integration. In this project, we are implementing a flight booking system where we will not need any external data that will need matching, integration, or mapping. Moreover, our proposed distributed database management system is a network of identical databases stored on multiple sites. So our distributed database management system will be homogeneous. Also, we have done our architectural development based on a top-down design approach. So this project will not need data integration.

However, the system proposed the global schema integrates all local schemas – a view in which case distribution is hidden.

9.2. Access Control

According to our project access control must guarantee that only authorized users perform operations they are allowed to perform on the database. Many different users may have access to a large collection of data under the control of a single distributed system. The distributed DBMS must thus be able to restrict the access of a subset of the database to a subset of the users. So we

are going with the discretionary access control approach where discretionary access control defines access rights based on the users, the type of access and the objects to be accessed.

Subject -entity capable of accessing resources. The introduction of a subject in the system is typically done by a pair (username, password). The username uniquely identifies the users of that name in the system, while the password, known only to the users of that name, authenticates the users. This prevents people who do not know the password from entering the system with only the username.

Object – Resources to which access is controlled.

Access right – Describes the way in which a subject may access an object, for example, read, write, execute, delete and search.

Our access control matrix will be in the below format:

Users\ objects	F1	F2	F3	F4
U1	Own, read, Write	Read		read
U2	read, Write		Own, read, Write	
U3	read	Own, read		read, Write

Table 1. Access control matrix

10. Query Execution

This section examines the core scenarios of user and system interaction through the platform's interface, which are shown visually in the second section. Also, examples of use cases are given so that users can get the right information about how to make queries for the user interface. Each global query will be decomposed into a sequence of SQL statements to be executed at each site.

Example Query Optimization

If a passenger search for flight information at site Newyork that basically need to combine two sites of such as Berlin and Helsinki. Because that person will come to Helsinki and then wants to travel Berlin to Helsinki. Then Berlin and Helsinki data will transmit to Newyork then combined for the final result.

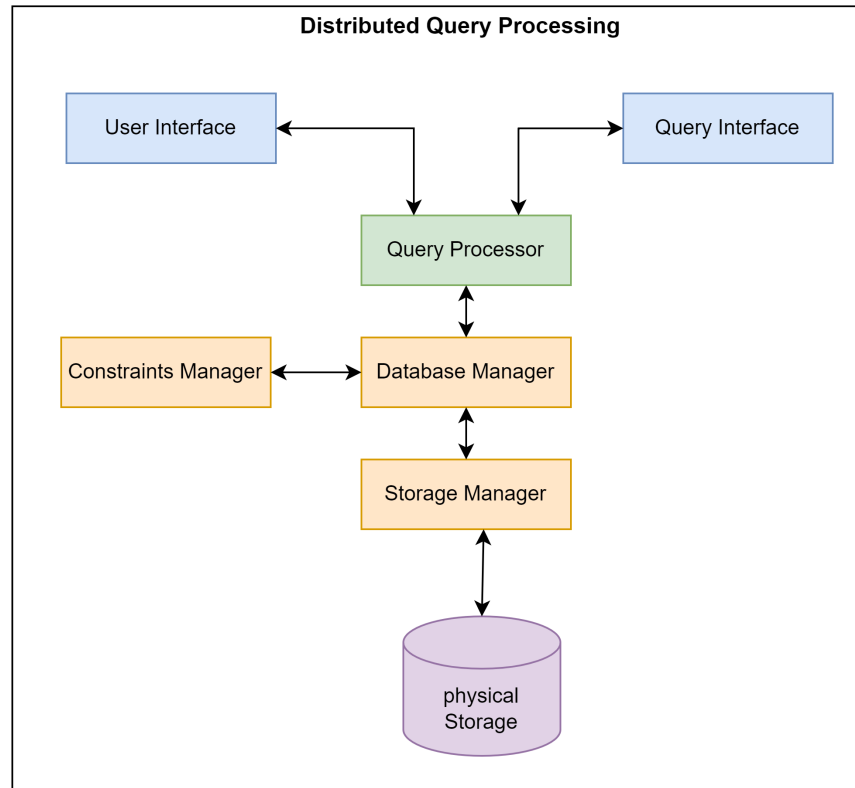


Figure 14. Distributed Query Processing

10.1. Registering Passenger

Actors: Passenger

Passengers can register on the platform by filling out the registration form using the user interface of the system. This form asks for the passenger's name, email address, telephone number, user ID, and password in separate fields. When the user clicks the button and enters their information, a new record for the passenger is created and added to the database.

A sample demo passenger registration query example:

```
INSERT INTO Passenger_table (passenger_id, passenger_name, passenger_telephone_number,
passenger_email, passenger_password) VALUES ('1', 'Jhon Doe', '+3582222222',
'jhon@gmail.com', 'Test123');
```

10.2. Registering Manager on the System

Actors: Branch manager

Managers who maintain the overall system administrator information can register on the platform by completing the registration form using the user interface of the system. This form contains separate fields for the manager's name, email address, phone number, user ID, and password, as well as branch name. When the user hits the button and inputs their information, a new record is created and added to the database for the manager.

A sample demo manager registration query example:

```
INSERT INTO Branch_office_table ( manager_name, branch_name  
manager_telephone_number,manager_email, manager_password) VALUES ('Vernon',  
'Lappeenranta_branch '+3582222222', 'vernon@gmail.com', 'manager123');
```

10.3. User Account Login

Actors: Passenger, manager

The user is required to enter their login credentials, which consist of their username and password, in order to log in to the system. The identification request is submitted to the database that is associated with the platform. In the event that the data that is entered into the credentials matches the data that is kept in the database, access will be permitted.

A sample demo login query example:

```
SELECT username, password from table where username = username AND password =  
password
```

10.4. Searching Flight Information

Actors: Passenger

Through the user interface of the application, anyone can search for flights that are currently available. After selecting the 'Search Flight' option from the user interface, passengers will see a screen with input fields for the city of departure, the city of arrival, the date of departure, and the date of arrival. After entering all the information, passengers will click the "search" button, and the system will show a list of flights that correspond to the passenger's input.

A sample demo flight searching query example,

```
SELECT * FROM reservation WHERE departure_city = departure_city OR arrival_city =  
arrival_city or arrival_date = arrival_date or departure_date = departure_date;
```

10.5. Flight Booking

Actors: Passengers

Through "flight searching," people will be able to see a list of flights where they can reserve or book tickets. But travelers have to sign up before they can reserve or book tickets. Using the "book ticket" feature, passengers can pay for a ticket and book it. After all the information is given, the system will ask the user to confirm. Once the information has been confirmed, the seat is reserved.

```
INSERT INTO reservation (passenger_id, date_time, flight_id, passenger_tel,  
airlines_site_office_id, status) VALUES (1, 'Isru', '8796', 3441121, '+38589765, 2, 1);
```

10.6. Flight Cancelations

Actors: Passenger

Before passengers may cancel their tickets, they must first book a seat. A traveler should then supply the facts regarding the ticket ID and flight ID when utilizing the "cancel ticket" functionality of the system. After that, the system will ask the passenger to confirm their decision. After the information has been confirmed, the cancelation will be made for the seat.

UPDATE reservation

SET status = 0

WHERE passenger_id = idpassenger;

11. Concurrency and Reliability

There are 4 types of concurrency control protocols: Lock-Based Protocols, Two Phase Commit Protocol, Timestamp-Based Protocols, Validation-Based Protocols. We will use a distributed two phase commit protocol (2pc). This solves the atomicity. Here is how it will work:

- Passenger requests for service
- The service commits to main platform
- The platform acts as a coordinator which sends an initiation message to customer and reservation services.
- Then, every server sends a “vote-ready” message to each other.
- Transaction is processed and executed.

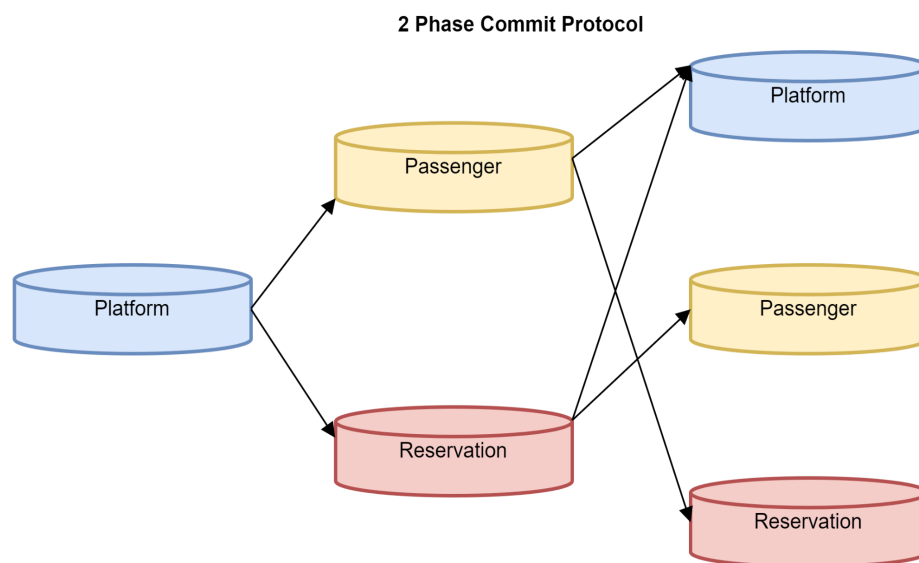


Figure 15. 2 Phase Commit Protocol

This will ensure that all the data across all servers are consistent. But, it can result in a deadlock problem. The figure below elaborates on this issue.

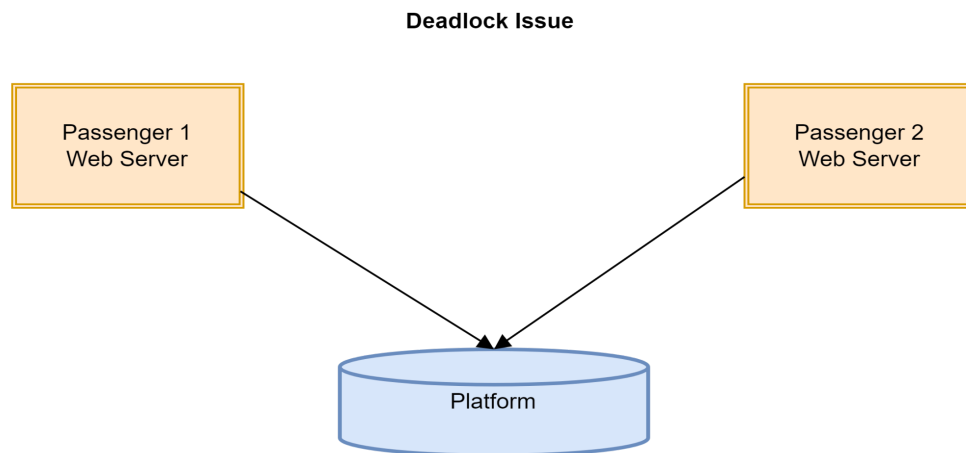


Figure 16. Deadlock Issue

Deadlock becomes an issue when two local servers of customers try to commit to the platform at the same time. But in this case, as the communication between the server will utilize “HTTPS” requests, the platform's own web server will resolve the deadlock issue.

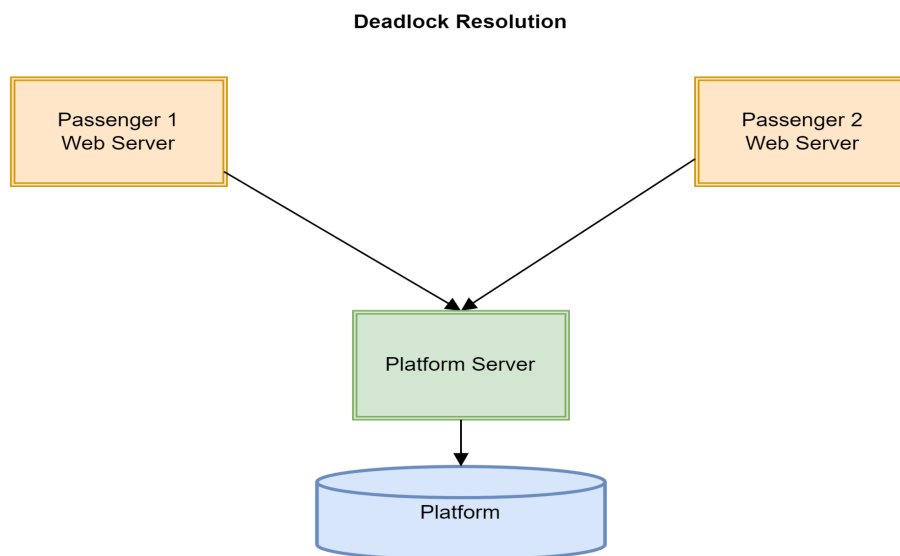


Figure 17. Deadlock resolution

If a request comes from two servers at the same time, we assign a different priority to each one. The web server evaluates the two priorities and performs the higher priority server request first. This does not mean that requests from low-priority servers will not be completed; they will just

be done later. Changing passenger information is handled in the same way, with a two-phase commit protocol and priority setting when two servers are concurrently accessed.

12. Data Security

The data security section addresses the broad principles of safe data management and describes the process involved in implementing this approach in the developed system. The passenger and their bookings data, as well as any data related to persons, are expected to be stored safely and confidentially, preserving the passenger's privacy.

12.1. Infrastructure Security

HTTPS is used to link services. TLS is used to safeguard data transferred via HTTPS. TLS is useful because it verifies that the other person in a connection is who they claim to be, shows if data preserves its original integrity, and offers confidentiality through encryption.

12.2. Service Authentication

Every HTTPS request sent from one service to another contains an access json web token that includes the client's hardware id. This hardware ID is signed with a unique secret key that is distributed throughout the country's network. When the request is signed with an invalid token or the token is missing from the request, the associated message is delivered to the client service.

12.3. User Authentication

Passenger and airline management access credentials are saved in the local organization database and in the platform's hashed with SHA-256 encryption method.

13. System Expansion

In the proposed distributed environment, it is much easier to accommodate and increasing database sizes because of fragmentation and localization. In the context, expansion can be handled by adding processing and storage power to the network, depending on the overhead of distribution. One aspect of easier system expansion is economic feasibility because this system will easily adopt small machines, but the user will get experience a single machine with high computation power system. Adding a new entity to the existing is much easier because global and local schemas are clearly outlined. Based on the relationship, fragmentation can also be done with the proposed algorithm (section 6.1). A new node or server can be added easily to the existing local network by full replication strategy and for global level partial replication can be applied.

Appendix

Database Schema and Data Creation Queries

```
use air_booking;
```

```
INSERT INTO country (idcountry, country_name)
VALUES (1, 'USA'), (2, 'FIN'), (3, 'TUR');
```

```
INSERT INTO branch_office (idbranch_office, branch_name, country_id, country_name)
VALUES (1, 'Newyork', 1, 'USA'), (2, 'Washington', 1, 'USA'),
(3, 'Helsinki', 2, 'FIN'), (4, 'Lappenranta', 2, 'FIN'), (5, 'Istambul', 3, 'TUR');
```

```
INSERT INTO passenger (idpassenger, passenger_name, passenger_email, passenger_tel,
agency_site_office_id, country_id)
VALUES (1, 'Isru', 'isru@gmail.com', '1234', 1, 1), (2, 'Knkur', 'knkur@gmail.com', '3456', 1, 1),
(3, 'Mnkur', 'mnkur@gmail.com', '45656', 1, 1), (4, 'fmnne', 'fmnne@gmail.com', '65456', 2, 1),
(5, 'Zmimmy', 'zmimmy@gmail.com', '9876', 2, 1), (6, 'Kabir', 'kabir@gmail.com', '7657', 3, 2),
(7, 'Ruhi', 'ruhi@gmail.com', '098776', 3, 2), (8, 'Minni', 'minni@gmail.com', '98787', 4, 2),
(9, 'Kanai', 'kanai@gmail.com', '09877', 5, 3);
```

```
mysql> select * from country;
+-----+-----+
| idcountry | country_name |
+-----+-----+
|          1 | USA          |
|          2 | FIN          |
|          3 | TUR          |
+-----+-----+
3 rows in set (0.00 sec)
```

Figure 18. Country table

```
mysql> select * from passenger;
```

idpassenger	passenger_name	passenger_email	passenger_tel	agency_site_office_id	country_id
1	Isru	isru@gmail.com	1234	1	1
2	Knkur	knkur@gmail.com	3456	1	1
3	Mnkur	mknkur@gmail.com	45656	1	1
4	fmnne	fmnne@gmail.com	65456	2	1
5	Zmimmy	zmimmy@gmail.com	9876	2	1
6	Kabir	kabir@gmail.com	7657	3	2
7	Ruhi	ruhi@gmail.com	098776	3	2
8	Minni	minni@gmail.com	98787	4	2
9	Kanai	kanai@gmail.com	09877	5	3

```
9 rows in set (0.00 sec)
```

Figure 19. Passenger Table

```
mysql> select * from branch_office;
```

idbranch_office	branch_name	country_id	country_name
1	Newyork	1	USA
2	Washington	1	USA
3	Helsinki	2	FIN
4	Lappenranta	2	FIN
5	Istambul	3	TUR

```
5 rows in set (0.00 sec)
```

Figure 20. Branch_Office Table