Airline Management System

Introduction:

An airline e-ticket is an electronic record of the traveller's airline reservation, containing information such as the time, date and place of the flight, airport, seat assignment and travel class. To book themselves on a flight, travellers can visit any number of Web-based ticketing sites. They can view the options available and use a credit or debit card to pay for their ticket. After placing the order, the electronic record of the ticket goes into the airline's database, where it holds the passenger's spot. E-tickets have virtually replaced traditional paper tickets in the majority of airports and airlines around the world. Recent surveys by the International Air Transport Association estimated that air carriers worldwide achieved more than 98% penetration in E Ticket booking in the US.

This project is on such an Online Airline Reservation System database. This Airline Reservation System project is a design implementation of a general Airline travel ticketing website, which services customers to search the availability, routes and prices of various domestic airline tickets within the US, along with the different options available with the reservations. This project also covers various features like manage all information related to flight, passengers, passenger contact details, reservation, transactions, schedule publishing, airfare payments etc. The major activities include and are not limited to the below:

TICKETING:
All sales transactions are related to flight ticket sales; advance reservations, Reservation cancellations etc. All customers are open to reserve flights to travel in future. Reservations are taken before a set time before the flight.

FLIGHT SCHEDULING:
Day to day flight scheduling, new flight arrangements according to sales potentiality, flight departure delay decisions all takes rooms in its daily flight scheduling activities etc.

AIRCRAFT DETAILS MANAGEMENT:
Management of aircraft vehicle and registration repository that includes tail number, carrier and model details.

PRICING:
Flight rates management, pricing etc.

PAYMENT AND SALES:
Managing Payment status, keeping track of ticket sales etc.

PASSENGER CONTACT DETAILS MANAGEMENT:
Management of passenger and PII data including Names, Email ID, DOB etc.

Logical Database Design:

The major Entities designed for the project are listed below:

- Passengers
- Airports
- Flight Schedules
- Flight Schedule Legs
- Aircrafts
- Aircraft Models
- Reservations
- Itinerary
- Payments
- Flight Fare

Scope and Assumptions:

The following assumptions have been made while designing the database:

- Scope of Airline schedules is within the US and includes only US airport domestic data.
- A Reservation will have one and only one passenger associated with it. Multiple passenger reservations are handled with multiple reservation IDs. Reservation status can have multiple values (Confirmed, Pending, Cancelled) based on the completion of the process.
- Flight Number denotes a route/schedule. Flight schedules mainly comprise of Master Source and destination Airports services by a Route.
- A flight route can have more than one leg, where the same Aircraft transits in one or more airports while still servicing the same route. Unique Leg IDs have been defined to handle these scenarios.
- Each flight route/Schedule is serviced by one aircraft. Though the aircraft might change over time, at a given point of time only one aircraft is used. This occurs only when aircrafts are taken out of schedule for servicing or other downtimes.
- An aircraft can have only one model and is represented by its Tail Number. Aircraft models may be used by zero or more Aircrafts.
- Flight fare is defined for each leg and class. The Business layer of the software calculates the total fare using this core data. The total is not computed at the database level. Flight fares are dependent on legs rather than the schedule routes.
- Flight fares are assumed to be static for a given point of time in the database for a given route leg.
- A Payment is recognized by a unique payment ID and is mapped to a reservation. Total payment data is inserted by the business layer of the application after calculating taxes, discounts at that level.
- Three travel classes are defined in the tables (Economy, Business, First) and the data has been validated and entered to the Reservation and Fare tables where they are used.
- Each passenger is identified by a Synthetic key. (Though email ID can potentially be used as a natural Primary Key). Passenger data is stored in our database and might be reused for future bookings at later points of time.
- A reservation can have multiple legs as part of it, though not vice versa. The legs denote different connections in between.
- The flight schedule assumes to follow the same pattern every day. Hence the day factor has not been taken into consideration.
- Alphanumeric notations are employed for tracking many IDs like Schedule, Leg, Payment etc., as per current Business standards.
- Fares deal with only once currency – USD.

Out of Scope for Project:

- Code Sharing of Aircrafts is not considered where one carrier leverages a partner carrier route with its own route number.
- User Login Credentials and password management functionality.
- Flight seats allocation and management. The project scope handles reservations and tickets only.
- Flight Servicing and Maintenance functionality management.
- Calculation of Taxes and Discounts

Physical Database Design:

The objective of physical database design is to use the knowledge gained from analysing the use and size of the database to improve the efficiency of the database. The efficiency of the database is determined by the requirements of the users and the database designer. For example, an efficient database may be one that executes queries quickly or uses the least amount of disc space.

- There are many different tools that the database designer may use when implementing the physical database.
- Indexing increases the speed of accessing the data by storing special data structures which can be processed very quickly.
- Caching uses part of the computer's main memory to store part of the database. Main memory is very quick to access.
- De-normalization is a process of changing the structure of the relations to improve the performance of queries and updates.

Indexing:

When a database relation becomes very large, searching for information in it can be a very slow process. Database management systems allow users to create indexes to speed up certain types of query.

In the example above, an index has been created on the name attribute of the relation. The index is smaller than the relation. This means that it is faster to search the index than it is to search the relation. The index speeds up queries that access individual tuples in a relation, for example, "show Smith's salary".

An index will not improve the speed of queries that access all the tuples in a relation, for example, "show all employee salaries".

Indexing Strategies:

- Index primary keys
  Primary keys are unique and are often used to access individual tuples in a relation. Primary keys are used to join two relations.

- Index foreign keys
  Foreign keys are used to join two relations together. An index will improve the speed of joins between relations.

- Index attributes that restrict queries
  In the query "show the salary where name = 'Smith'" the attribute name is restricting the query to all Smiths. To answer this query all the tuples containing 'Smith' must be retrieved. An index will improve the speed of queries that access individual names.

- Index attributes that are sorted
  Sorting data is a very slow process. Indexes are sorted when they are built. Sorting data using an index simply means reading the index.

- Do not index attributes with few values
  Indexes work best when accessing individual tuples. Attributes with few values will have many duplicates.

- Do not index every attribute
  Indexes are stored on disc. Unnecessary indexes waste space.

Capacity Planning:

Capacity planning acknowledges that the business requirements on the system may increase, and forecasts how much resource must be added to the database system to ensure that the user experience continues uninterrupted.

Typically, the resources you'll add may be CPU power, memory, storage, or network capacity, or more likely a combination of these, depending on how demands are predicted to grow. This makes capacity planning different to performance tuning, because the latter is a reactive process. You tweak the system's parameters to ensure that it does not dip below pre-set levels based on current performance requirements. You plan capacity to ensure that it does not fall below performance levels in the future.

To properly perform capacity planning a cooperative effort must be undertaken between the system administrators, database administrators and network administrators.

- Step 1: Size the Oracle database
- Step 2: Determine number and Type of Users.
- Step 3: Determine hardware requirements to meet required response times and support user load.
- Step 4: Determine backup hardware to support required uptime requirements

The DBA_TEMP_FREE_SPACE dictionary view contains information about space usage for each temporary table space. The information includes the space allocated and the free space. You can query this view for these statistics using the following command.

Architectural Issues:

Architecture Bottlenecks mainly consists of scaling bottlenecks and they are formed due to the following two issues:

- Centralized component:
  A component in application architecture which cannot be scaled out adds an upper limit on the number of requests that entire architecture or request pipeline can handle.
- High latency component:
  A slow component in request pipeline puts lower limit on the response time of the application. Usual solution to fix this issue is to make high latency components into background jobs or executing them asynchronously with queuing.

Issues with distributed database:

- Complexity
  DBAs may have to do extra work to ensure that the distributed nature of the system is transparent. Extra work must also be done to maintain multiple disparate systems, instead of one big one. Extra database design work must also be done to account for the disconnected nature of the database — for example, joins become prohibitively expensive when performed across multiple systems.

- Security
  Remote database fragments must be secured, and they are not centralized so the remote sites must be secured as well. The infrastructure must also be secured (for example, by encrypting the network links between remote sites).

- Difficult to maintain integrity
  In a distributed database, enforcing integrity over a network may require too much of the network's resources to be feasible.

- Inexperience
  Distributed databases are difficult to work with, and in such a young field there is not much readily available experience in "proper" practice.

- Lack of standards
  There are no tools or methodologies yet to help users convert a centralized DBMS into a distributed DBMS.

- Database design more complex
  In addition to traditional database design challenges, the design of a distributed database has to consider fragmentation of data, allocation of fragments to specific sites and data replication.

- Distributed access to data.

- Analysis of distributed data