Project 1 Paper

Hibah Masoom

Online Master of Science in Data Science, Merrimack College

DSE6210: Big DataL SQL and NoSQL

Jerimiah Lowhorn

29th September 2024

When handling customer functions in a relational database management system (RDBMS) like the one used in the airline reservation system, not using views can present several performance challenges. A view in SQL is essentially a virtual table created by a query, and it can significantly optimize data retrieval. Without views, complex queries required by customers, such as retrieving flight information between cities, viewing itineraries, or finding connecting flights, must be repeatedly executed on the raw database tables. This can lead to longer response times, especially when multiple tables need to be joined or filtered for specific data. As the database grows in size, whether in terms of flight schedules, reservations, or passenger data, the query execution time increases exponentially due to the need to scan large tables.

One of the most significant performance constraints is the necessity for the database engine to execute complex joins and filters repeatedly. For example, if a customer wants to find available flights from City A to City B, the database would need to join the flight_schedules, legs, itinerary_reservations, and possibly airports tables, each time the query is executed. If these queries are not optimized using views, the system must compute the same joins and filters every time the query is made, rather than relying on pre-processed or cached data, leading to unnecessary resource consumption, including CPU and I/O. This problem becomes worse when the number of users increases concurrently, leading to longer wait times and potential query timeouts.

Not using views can introduce significant overhead for the database, especially as the dataset scales. As the airline system grows, the number of reservations, customers, and flights will increase rapidly, creating a larger dataset that must be queried for each customer request. Without views, the database must process every query from scratch, rather than reusing previously computed results or pre-aggregated data. For example, repeatedly querying all flights between two cities with specific time constraints would involve scanning potentially millions of rows in real time, without any data caching mechanism in place. This creates substantial overhead as the query optimizer must evaluate the best execution plan for each query independently, further taxing system resources.

Additionally, as the database scales in size, query performance will degrade if not managed properly. Highly transactional web applications, such as an airline reservation system, rely on fast response times to keep customers engaged and to avoid failures during bookings or cancellations. Without views, the raw queries could quickly become bottlenecks, particularly when multiple customers are querying the same data at once. Indexing can alleviate some of this problem, but without predefined views, the database will lack an efficient mechanism to simplify repeated complex queries. Over time, this would negatively affect user experience and system throughput, especially as more customers concurrently access the system.

To mitigate these scaling challenges, a few strategies can be employed. First, creating views for customer-facing queries would offload much of the work from the underlying tables and simplify repeated query execution. Views allow frequently executed queries, like checking flight availability, retrieving passenger itineraries, or viewing flight schedules, to be predefined, improving performance by eliminating the need for the database to repeatedly parse, plan, and execute complex SQL queries. Moreover, materialized views can further enhance performance by storing the results of a view physically on the disk, so that they can be retrieved quickly without recalculating. Materialized views are particularly useful for queries that do not change frequently, such as flight schedules, allowing the system to serve results almost instantly.

Another strategy to enhance scalability is indexing key fields, such as flight_number, passenger_id, and reservation_id. Indexes provide a faster lookup for commonly queried fields and reduce the amount of time the database takes to retrieve records. Indexes on frequently joined columns, such as airport_code and flight_number, would allow the system to access these records quickly, improving performance. Additionally, partitioning large tables based on flight dates or airports can significantly improve performance, particularly for write-heavy tables such as Itinerary_Reservations or Legs. Partitioning helps manage large datasets by splitting them into smaller, more manageable chunks, improving both read and write operations by reducing the number of records that need to be scanned or updated at any given time.

While using a first-normal form (1NF) RDBMS provides the foundation for data integrity and elimination of redundancy, it may not always be the best fit for a highly transactional web application like an airline reservation system. First-normal form ensures that the database schema does not contain repeating groups and that each attribute contains atomic values, which is good for maintaining data consistency. However, in high-transaction environments, 1NF can introduce inefficiencies, especially when dealing with complex transactions like managing reservations, updating itineraries, or handling multi-leg flights.

A potential issue with relying purely on a first-normal form RDBMS is that join-heavy queries required to pull together disparate pieces of information (e.g., passengers, reservations, flights) can be computationally expensive, leading to performance bottlenecks. For instance, in 1NF, reservations, passengers, and flights are stored in separate normalized tables, which means that every query must join multiple tables. In a web application where speed and responsiveness are paramount, these joins can slow down response times. To counteract this, denormalization strategies can be used selectively, where redundant data is stored to reduce the need for costly joins, especially for read-heavy operations. Denormalization can improve read performance by reducing the need for multi-table joins at the cost of potential redundancy, which may be acceptable in high-performance applications.

Moreover, in a highly transactional web application, NoSQL databases or hybrid systems may provide better scalability compared to a pure 1NF RDBMS. NoSQL databases, like MongoDB or Cassandra, are designed to handle massive amounts of transactions with high write throughput. They offer schema flexibility and can scale horizontally, which might be better suited for the dynamic nature of an airline reservation system, where the number of flights, customers, and bookings can fluctuate rapidly. However, the decision to move away from a traditional RDBMS like PostgreSQL depends on the specific performance and scalability requirements of the application.

In conclusion, not using views for customer-facing queries in an airline reservation system could introduce significant performance constraints, particularly as the database scales. Without views, complex queries would need to be executed repeatedly, leading to overhead and slower response times, which would negatively affect customer experience. Additionally, as the number of users and transactions increases, these issues would scale, potentially overwhelming the database. Creating views, indexing key fields, and partitioning large tables would alleviate many of these challenges. Furthermore, while first-normal form RDBMSs offer benefits in terms of data integrity and structure, they might not always be the best choice for a highly transactional web application. In some cases, denormalization or transitioning to a NoSQL system could provide better performance and scalability.

Sources:

1. Elmasri, R. & Navathe, S. *Fundamentals of Database Systems*. Addison-Wesley. (2015).
2. Harrison, G. *Next Generation Databases: NoSQL and Big Data*. Apress. (2015).
3. Silberschatz, A., Korth, H. F., & Sudarshan, S. *Database System Concepts*. McGraw-Hill Education. (2019).