DATABASE DESIGN ASSESSMENT 2

BY

MITSIS SPYRIDON 2228036

BSC (HONS) CONPUTING YR. I

DATABASE DESIGN

SWE4004

Prof. RAMMOS DIMITRIOS

WORD COUNT: 2005

MAY 2023

Contents

1 Reversed engineered Wolt schema

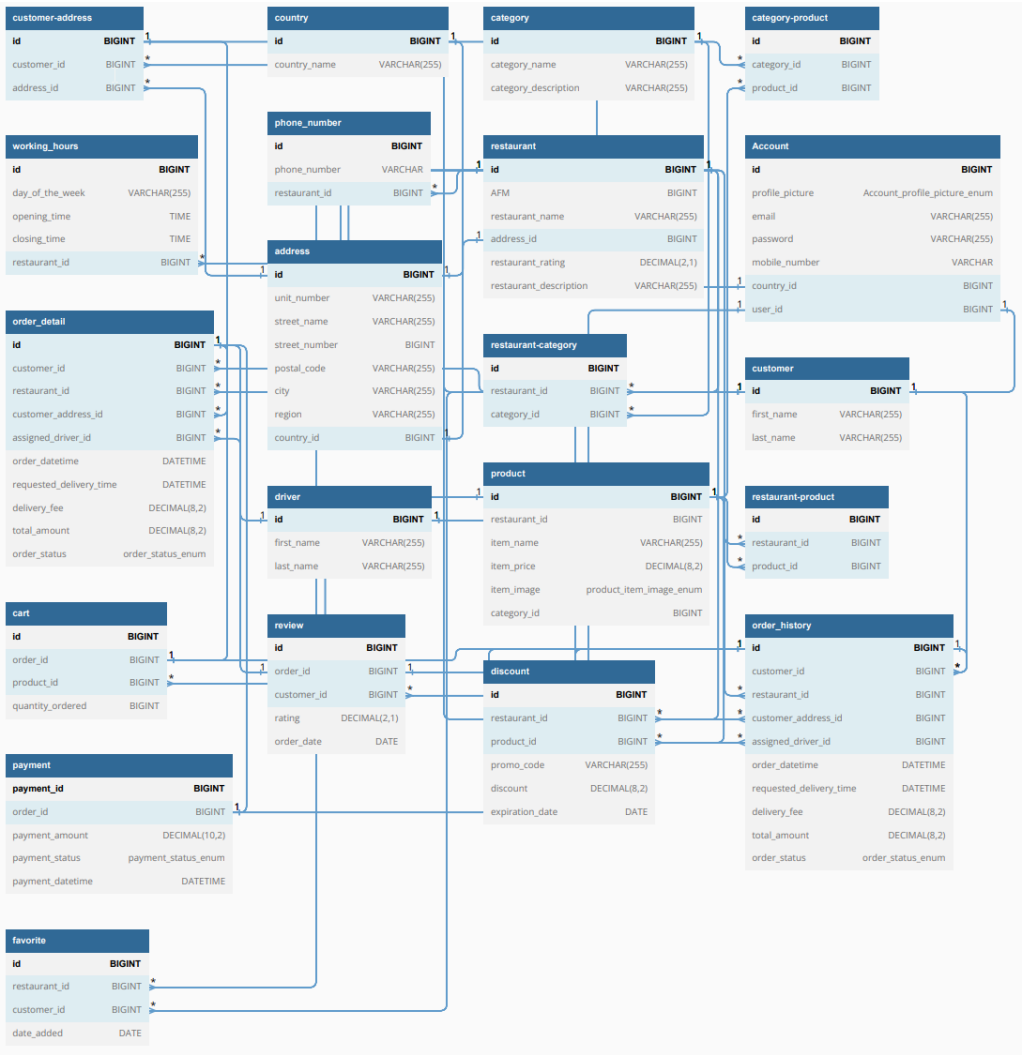

Figure 1: Wolt Schema

The schema is a bit to big to fit in the page. You can visit the website in the footnote that hosts the schema, to login use the password `woltdb10` [1]

[1] `https://dbdocs.io/mitsis.spiros1/Wolt_DB_Mitsis?view=relationships`

2 Potential Problems

Some potential problems that may arise when it comes to Wolt delivery service app are Scalability, loose security measures, and improper or missing use of indexes to maximize query speed.

2.1 Scalability

Lack of scalability can become a big problem as the business grows. Wolt uses a MySQL database and with this database schema on peak usage hours, usually during lunch and dinner time, the site can experience slower performance and even downtime resulting in a loss of revenue. In order to fix this problem a radical change must be made. The root of this problem stems from the fact that the database used in this scenario is MySQL. MySQL scales linearly, referring to the fact that in order to increase its throughput it needs more resources to handle the load. The resources in question are CPU RAM or hard disks. This is the easiest method for scaling a database but it is not sustainable because after reaching a certain threshold it becomes exponentially more expensive to upgrade a single server. Another approach to scaling is the horizontal one. Where instead of upgrading a single server to increase throughput an additional server, called node, is added. This philosophy can be found in a SQL database such as CockroachDb but is most commonly found in NoSql Databases, more specifically document-oriented databases like MongoDB and Fire-

store (MongoDB, 2020).

2.2 Data Security

Data security poses a significant concern to the Wolt database. Arguably the most important way to mintage the security risk and diminish the size of the attack surface is data encryption.

2.2.1 Encryption

Transparent data encryption is a technology used by Oracle databases such as *MySQL*. It encrypts the data that is located on the disk. If the database and the Keystore are open then *TDE* will encrypt and decrypt the data moving through the database. *TDE* is a powerful tool that protects data when an attacker attempts to bypass the database and direct access the data from the disk. By implementing *TDE* we can reduce the attack surface of the database by about 15% to 20%.

Another technique used to secure the data inside the database is the use of *whole-disk encryption*. As the name suggests this method encrypts the entire disk, which is great for security, and if someone has access to the operating system even when bypassing the database no data can be retrieved. *TDE* mitigates this problem by forcing the user to go through the database to access any data (Mustafa and Lockard, 2019).

2.2.2 Audits

Auditing is one of the most important tools in the security of the database. Auditing helps us understand whether or not a hacker has compromised our system or detect previously undetected vulnerabilities. Different techniques can be used to audit the database, the most commonly used are through API code, database triggers, normal audits, undefined audits and fine-grained auditing.

2.2.3 API Code

Creating API calls in the database is a highly configurable way to capture data inside the database. This method works by making an API call to the database and storing the audit data.

2.2.4 Trigger Code

By using Triggers we can separate the audit events from the application. Every time a trigger event executes it will be recorded giving us a history log table of all the changes to the database. The downside of auditing with triggers is that the audit data will be stored in many tables and to get a full picture of the audit, we need to span multiple tables. One major downside of both *API code* and *Trigger Code* is that they require to write code and when writing code there is a high possibility of implementing a bug in the system. This creates a need to maintain the code and patch bugs.

2.2.5 Normal Audits

Normal Audits are SQL server audits that enable tracking and logging of selected database activities. They capture information such as login attempts, data modification, and schema changes just to name a few. These types of audits store information in multiple locations.

2.2.6 Undefined Audit

An undefined Audit solves the problem of having data stored in multiple places by storing the data on an internal table. It also gives us the ability to create audit policies that can be fine-tuned to our specific needs.

2.2.7 Fine-grained Audit

This method of auditing gives the ability to fine-tune the based on any number of factors but because code is involved it has the same disadvantage that the *API Code* and *Trigger Code* has (Carter, 2018).

2.3 Indexing and Query Optimization

Indexing and query optimization significantly impact the performance of the database, leading to increased resource consumption and slower response time.

2.3.1 Ineffective Indexing

An Index is a schema object used to speed up the retrieval of rows by implementing pointers. They improve query performance by reducing the number of disk Input/Output operations necessary for the retrieval of data. Improper or missing indexes result in slower query execution times and increased disk I/O operations, especially for “expensive” queries like joins on large tables. To fix this problem we must create indexes on columns to optimize performance, and use composite indexes when multiple columns are often used together.

2.3.2 Query optimization

Poorly written queries that lack optimization techniques such as join conditions or sub-query elimination, lead to slower query response time. To mitigate this problem we must re-write the queries to use explicit join rather than implicit, ensuring that the appropriate join conditions are specified. Utilize query hints to help the query optimizer choose a more efficient execution plan. Avoid overusing sub-queries and opt for `JOIN`s that are more efficient and offer greater performance,

2.4 Conclusion

Scalability is an important factor to consider as the business grows, and the current use of *MySQL* schema, as the company is going to grow, is

going to result in slower performance and higher maintainability costs to keep the service up. To address this problem a different type of database that scales horizontally should be taken into consideration. To increase data security it is vital to implement encryption techniques like *Transparent Data Encryption TDE*, ensuring that data both in traffic and at rest is secured and cannot be accessed by unauthorized third parties. Auditing is essential for detecting security breaches in the system, measures like API code, trigger code, or plain simple audits must be implemented as early as possible. Finally, inefficient indexing and query optimization has led to poor performance. Queries must get analyzed and optimized, ensure proper indexing has been made, and update statistics regularly.

3 concurrency control

Concurrency control plays a fundamental role in any database that grants access to multiple users which make multiple transactions concurrently. A transaction is nothing more than one or more units of works that serve as a way to represent a state change inside a database. A database transaction must follow ACID (Atomicity, Consistency, Isolation, Durability) if a relational database is being used or BASE (Basically Available, Soft state, Eventual consistency) if it is non-relational. These sets of guidelines ensure that any transaction is being processed reliably and accurately. The main purpose of concurrency control is to ensure the validity and integrity of

the data while guaranteeing other transactions access to the same data. Executing transactions simultaneously, without any way to control them, will result in conflicts that need to be fixed. Arguably the most straightforward answer to this problem is to simply use a locking base algorithm (Shebka, 2022).

3.1 Lock-based Algorithms

These types of algorithms lock the data for other users when a user has been granted access to it. The data will remain locked until the lock is opened by a commit, abort, or termination trigger. Unfortunately, this solution although great for applications that require a high degree of serializability, they are very inefficient in terms of time and processing costs. Another major drawback of lock-based algorithms is that they can experience *deadlocks*.

3.1.1 Deadlocks

Deadlock is a situation that occurs when two different transactions have locked some data necessary for the other transaction to access, but to be able to finish execution, both must wait for the other to complete, creating a state of *deadlock* where both transactions are in a stalemate.

3.2 Timestamp-based Algorithms

When opting for the timestamp approach each transaction is given a unique timestamp. Every read/write operation includes the timestamp associated with the transaction that is requesting access to the data. These operations are executed using the *FIFO (First In First Out)* method. Using this algorithm we execute transitions in a serialized manner while minimizing conflicts and avoiding deadlocks. Timestamp-based algorithms also perform some validation to guarantee that the execution order does not break the consistency of the database. One problem that may arise when using these algorithms is by allocating timestamps using a centralized counter, this may result in bottlenecks inside the system (Chaudhry and Yousaf, 2022).

3.2.1 FIFO

The *FIFO* data structure can be considered a queue. A queue is a linear list where new data is inserted at the end of the list (called “tail”) and the deletion of the data is made at the front (called *head*). This data structure can also be referred to as *FCFS (First Come First Served)* (Vijayalakshmi Pai, 2023).

3.3 Certification-based Algorithms

Certification-based algorithms also known as *Optimistic concurrency control algorithms* are timestamp-based algorithms that go through three phases during their execution. The read phase, the validation phase, and finally the write phase. Each data point is assigned both a read and a write time stamp. These algorithms assume that conflicts between transactions are scarce and most transactions will not impede the execution of another one. All the transactions are allowed to execute without any locks and the database system tracks each change that is made. When the time comes to make a commit and finalize the transaction they report to the database system that will validate whether or not it the transaction can be certified. If the transaction's read timestamp is the same as the global write timestamp then it can be certified, if not it reads the newer write timestamp. In order for a write request to be approved no newer read request must have been approved and committed.

3.4 Multi-version-based algorithms

Multi-version Concurrency Control (MVCC) is an algorithm that considers any write operation to the data item as a creation of a new version, essentially creating different snapshots of the data item at a specific point in time. At the time of a read operation, one of the versions of the data item is selected to be read. The purpose of this algorithm is to maintain multi-

ple versions of a data item without jeopardizing serializability. In contrast with single-version systems, multi-version systems allow bigger concurrency and increase efficiency by allowing both read-only and read-write transactions to execute simultaneously. This can create a serious problem when attempting to synchronize all the changes to the database resulting in an overhead instead of a benefit. To keep track of all the transactions meta-data, containing the increasing transaction numbers alongside information regarding the lifetime of the transaction, a pointer pointing to consecutive neighboring transaction versions is attached to the database. Some popular variations of multi-version concurrency control schemes are timestamp ordering, optimistic concurrency control, serial certifier and two-phase locking. Computing the serialization order can be achieved using various methods such as using thread ID or the local clock.

3.5 Hybrid Algorithms

A Hybrid algorithm uses a combination of both locking-based algorithms and timestamp-based ones. It attempts to take advantage of the benefits that the two algorithms provide. It can lock the data during a write phase and use timestamps during the reading phase. Using this method it can have a high level of serializability while at the same time increasing the overall efficiency of the database. In order to further improve concurrency hybrid algorithms use different types of locks such as shared locks and

exclusive locks (Sheikhan and Ahmadluei, 2015).

3.6 Conclusion

Concurrency functions are the backbone of all databases that support multiple users and transactions. Various algorithms can be implemented, from easy Locking based algorithms that lock the data until a transaction is complete, to more complex ones like Hybrid algorithms that combine the benefits of multiple algorithms minimizing the drawbacks of each and maximizing the benefits.

References

Carter, Peter A. (2018). *Securing SQL Server: DBAs Defending the Database*. English. Berkeley, CA.

Chaudhry, Natalia and Muhammad Murtaza Yousaf (Feb. 2022). “Concurrency control for real-time and mobile transactions: Historical view, challenges, and evolution of practices”. In: *Concurrency and Computation: Practice and Experience* 34.3, e6549. DOI: https://doi.org/10.1002/cpe.6549. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.6549. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.6549.

MongoDB (2020). URL: https://www.mongodb.com/basics/horizontal-vs-vertical-scaling.

Mustafa, Osama and Robert P. Lockard (2019). *Oracle database application security: with Oracle Internet Directory, Oracle Access Manager, and Oracle Identity Manager*. English. New York: Apress.

Shebka, Nasser (Apr. 2022). *International Journal of Advanced and Applied Sciences, 9(7) 2022*. URL: http://science-gate.com/IJAAS/Articles/2022/2022-9-7/1021833ijaas202207016.pdf.

Sheikhan, Mansour and Saeed Ahmadluei (Nov. 2015). “An intelligent hybrid optimistic/pessimistic concurrency control algorithm for centralized database systems using modified GSA-optimized ART neural model”. In: *Neural Computing and Applications* 23.6, pp. 1815–1829. ISSN:

1433-3058. DOI: 10.1007/s00521-012-1147-3. URL: https://doi.org/10.1007/s00521-012-1147-3.

Vijayalakshmi Pai, G. A. (2023). *A Textbook of Data Structures and Algorithms, Volume 1: Mastering Linear Data Structures*. English. Newark: John Wiley Sons, Incorporated. ISBN: 178630869X;9781786308696;