Getting information to the user optimally*

Samuel Tvrdoň
Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
xtvrdon@stuba.sk

5. november 2023

Abstract

Getting the user, the information they seek as quickly and efficiently as possible is key when talking about searching in today's world. his article chooses to focus on the ways this is achieved behind the scenes. Comparing different indexing techniques and other optimizations modern database engines employ to provide optimal storage and retrieval efficiency. The article also looks at methods of making retrieval faster by utilizing caching in a way that increases the performance of storage access. Furthermore, querying the database for data the user wants quickly, would not be possible without proper tokenization of the data in the database. The methods for tokenization are also mentioned in the last section of the article.

1 Introduction

With the ever growing expectations from the side of the users, database technologies are always evolving. [7] Fighting for the users attention means bringing up the requested information as quickly as possible. Since the inception of the internet there have been many advancements in the field of information storage and retrieval that are now common place in almost every large application. There are of course benefits and pitfalls each of the performance improvement techniques brings. The first part of article aims to describe indexing its strengths and also multiple ways to create an index for a database table. Moreover, even more performance can be gained by predicting, which data will be used the most and storing it in a faster storage with caching. Additionally the article goes over turning natural language searches into efficient database queries. In this section the article explains how tokenization is used to achieve natural search feeling from the users perspective, while maintaining optimal speeds.

2 Indexing

Indexing is used to make querying data faster and more efficient. Since the data can be any size, and stored in any order, going over every record is unfeasible.

*Semestrálny projekt v predmete Metódy inžinierskej práce, ak. rok 2023/24, vedenie: Ing. Ivan Kapustík

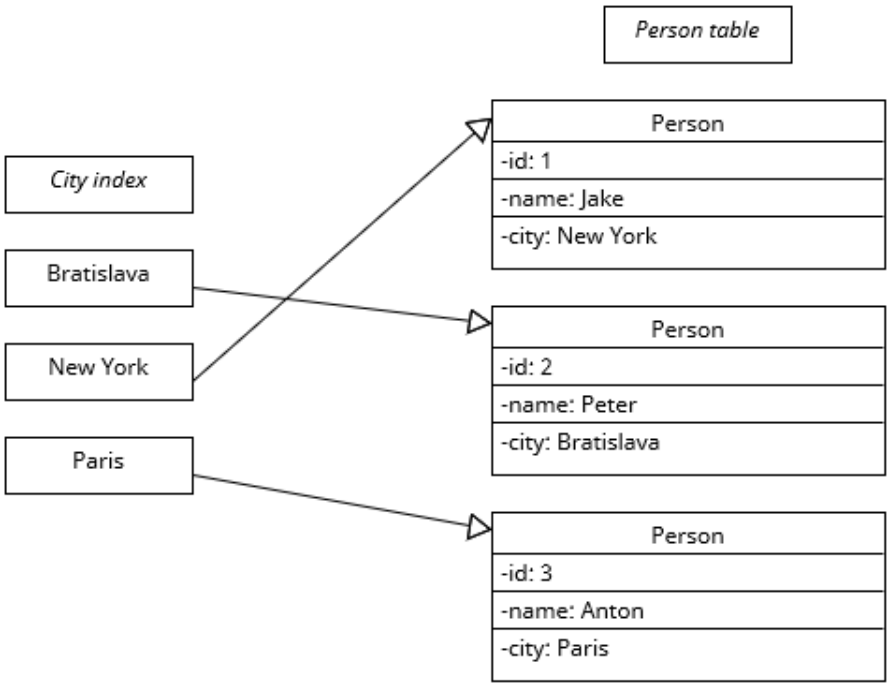

Figure 1: Example of indexing by city name

With the amount of requests users around the globe make, the equipment costs grow rapidly. To solve this issue, indexing has been introduced as a method to help avoid the need of traversing all of the data on each request. From a report by one of the largest search engines in the world Google [2]:

> The Google Search index contains hundreds of billions of webpages and is well over 100,000,000 gigabytes in size. It's like the index in the back of a book — with an entry for every word seen on every webpage we index. When we index a webpage, we add it to the entries for all of the words it contains.

Indices can be created for each of the parameters that needs to be sorted separately. This reduces storage requirements, drastically. Instead of choosing between searching the table each time and having multiple copies of the table ordered by different columns, a third option is now possible. Storing only references to the real table along with the ordered indices. This arrangement is shown in the first figure.1

Another benefit of indices is their reference to the table is only one directional. In other words the table does not care about how many different indices exist, which provides for easy manipulation and flexibility when iterating over database design.

However, indexing isn't one size fits all, there are many options, which cut different corners and make needed optimizations for even faster querying. [8]

2.1 Creating an effective key

As mentioned in the above paragraph, the performance improvement indexing provides is dependent on the data stored in the database. Take the example in Figure 1 again1. The id column is unique for each person, making it a prime candidate for an index key. However, the index must also be a value users look up the data by, making the name or the city column more fitting. The problem is the name and city are not required to be unique for each person. This fact reduces the performance increase they would provide. These facts make indexing a balancing act based on the user's needs and the stored data. The uniqueness of values in a database column is measured as cardinality. And in the recent years there have been attempts to choose optimal columns for indexing using the help of artificial intelligence. [9]

3 Caching

In every system each record is accessed with varying frequency. This fact is leveraged by caching. Caching layer makes for even faster retrieval of the most used records. The most queried records are stored in a temporary memory with faster read speeds than the main larger memory. This helps reduce time as well as the stress on the main storage. [11]

3.1 Accessing cache

The cache layer is a system detached from the main database. This fact allows for different implementations not tied to the underlying database used. This sections aims to look into the most popular arrangements and their pros and cons. [1] [3]

3.1.1 Cache-aside

In this arrangement, users first request from the cache. If there is a cache hit, the cache has the requested data and it is returned. If there is a cache miss, the database needs to be queried and the data subsequently returned and stored in the cache for next request. The benefits are in its simplicity and usefulness in general use cases mainly with high read intensity. Cons arise when writing is involved as the speed benefits are lost when any write has to be synced between cache and the database.

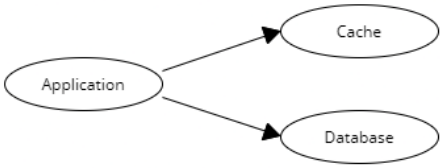

Figure 2: Cache aside

3.1.2 Cache-through

Very similiar to Cache-aside 3.1.1 The main difference is in which party is responsible for populating the cache. While in Cache-aside the cache is written to by the application. In Read-through arrangement the cache is hidden behind the database management system. Every request goes through the cache, which then proceeds to call the database. Write-through solves the issue of syncing write changes with the cache. In this arrangement all writes as well as reads go through the cache, which, then calls the database. This setup hides the slow main storage away from the users and lets him only talk with the fast cache.

Figure 3: Cache through

3.1.3 Cache-back

In this setup each request goes through the cache, utilizing all the speed benefits the faster storage offers. However, the data isn't immediately passed to the main database, instead the cache passes the data in set intervals, while it is free from load.

Figure 4: Cache back

3.2 Invalidating cache

Since the cache is way smaller than the main memory. There comes a point when the stored data needs to be cycled. Various algorithms are employed to maximize cache hits and minimize cache misses. Cache invalidation is a technique virtually every modern database system uses. [4] This section goes over the most common ones, from the simplest to implement to the more complicated ones. [5]

3.2.1 Random replacement

Random replacement is the simplest form of cache invalidation and is therefore preferred for the simplest of applications. The main benefit comes from the randomness of this approach. There is no need to perform any additional book keeping, like the frequency of access to decide which record to replace. Each piece of cached data is assumed to be just as likely to be accessed in the future so, which one gets removed does not matter.

3.2.2 First in first out

The first in first out approach tries to reason about, which data the user might need next. The heuristic reasons that a user is more likely to access data he accessed recently. This algorithm brings in some logic to, which data is replaced, but still keeps it to a minimum, while providing better results.

3.2.3 Least recently used

This strategy chooses to replace items based on the recency of their last access. At its core it is very similar to the First in first out approach, however there is extra logic included. If a piece of data was cached last it can move up the queue when accessed to improve efficiency.

3.2.4 Least frequently used

When invalidating entries with the least frequently used heuristic, the data that is accessed the least is assumed to be least likely used in the future as well. Invalidating based on the access frequency means keeping around data that is accessed lots of times. This method provides a benefit over LRU 3.2.3 in access patterns that would invalidate elements accessed in batches. [6] [10].

4 Tokenization

Tokenization is what ties information retrieval together. It bridges the gap between natural language and databases. Natural language contains many filler words and padding not relevant to the information being searched. To efficiently convert text to a database query it needs to be tokenized. During tokenization the critical parts of each sentence are filtered and used as "tokens". [12]

5 Conclusion

In the world of databases there are methods, which if used correctly result in huge performance increases with little added cost. All of the mentioned methods used together make up large part of today's internet infrastructure. Without these optimizations the internet would not be where it is today. These methods may also help create even better optimizations in the future and move data processing to the next level.

References

[1] AWS Whitepaper caching patterns. `https://docs.aws.amazon.com/whitepapers/latest/database-caching-strategies-using-redis/caching-patterns.html`. Accessed: 2023-11-01.

[2] Google how google search organizes information. `https://www.google.com/search/howsearchworks/how-search-works/organizing-information/`. Accessed: 2023-11-01.

[3] Prisma's Data Guide introduction to database caching. https://www.prisma.io/dataguide/managing-databases/introduction-database-caching. Accessed: 2023-11-01.

[4] Redis cache invalidation. https://redis.com/glossary/cache-invalidation. Accessed: 2023-11-01.

[5] C. Aggarwal, J. Wolf, and P. Yu. Caching on the world wide web. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):94–107, 1999.

[6] M. Chand. A comparative survey on different caching mechanisms in named data networking (ndn) architecture. *International Journal of Emerging Technologies and Innovative Research*, 6(4):264–271, April 2019. Available at SSRN: https://ssrn.com/abstract=3379298.

[7] H. Diakoff. Database indexing: yesterday and today. *The Indexer: The International Journal of Indexing*, 24, 10 2004.

[8] S. J. Kamatkar, A. Kamble, A. Viloria, L. Hernández-Fernandez, and E. G. Cali. Database performance tuning and query optimization. In Y. Tan, Y. Shi, and Q. Tang, editors, *Data Mining and Big Data*, pages 3–11, Cham, 2018. Springer International Publishing.

[9] Y. Li, H. Wang, and X. Liu. One stone, two birds: A lightweight multidimensional learned index with cardinality support, 2023.

[10] A. Osman and N. Osman. A comparison of cache replacement algorithms for video services. *International Journal of Computer Science and Information Technology*, 10:95–111, 04 2018.

[11] Z. Scully and A. Chlipala. A program optimization for automatic database result caching. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL '17, page 271–284, New York, NY, USA, 2017. Association for Computing Machinery.

[12] V. Singh and B. Saini. An effective tokenization algorithm for information retrieval systems. volume 4, 09 2014.