VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

DATABASE MANAGEMENT SYSTEMS (CO3021)

Team Assignment

Big Volume Management

Instructor:
Ph.D. Phan Trong Nhan

Group - Class:
Group 5 - Class CC01

Student:
Vo Truc Son - 2252720
Truong Tran Duy Tan -
Truong Quang Nha - 2152822
Bui Duc Manh Cuong - 2252096
Nguyen Ngoc Phuong Mai - 2211983

HO CHI MINH CITY, APRIL 2025

Contents

1 Member list & workload **2**

2 Theory - Big Volume Management **3**
2.1 Definition . . . 3
2.2 Characteristics . . . 3
2.2.1 Scalability . . . 3
2.2.2 Performance Optimization . . . 4
2.2.3 Fault tolerance and Reliability . . . 5
2.2.4 Automation and Orchestration . . . 5
2.2.5 Security and Compliance at scale . . . 6
2.2.6 Cost optimization for massive workloads . . . 6
2.2.7 High-volume data processing . . . 6
2.3 Benefits . . . 6
2.3.1 Efficient Storage and Retrieval . . . 6
2.3.2 Smarter Decision Making . . . 6
2.3.3 Data Consistency and Security . . . 7
2.3.4 Flexible Scalability . . . 7
2.3.5 Support for Advanced Analytics . . . 7
2.3.6 Cost Savings . . . 7
2.3.7 Ensuring Availability and Reliability . . . 7
2.3.8 Enhanced Data Integration Capabilities . . . 7
2.3.9 Support for Regulatory Compliance . . . 7
2.3.10 Improved Customer Experience . . . 8
2.4 Examples of Big Volume Management applications . . . 8

3 Implementation on DBMS **9**
3.1 Normal query . . . 9
3.2 Partition . . . 12
3.2.1 Result . . . 12
3.2.2 Comparison with other techniques in theory . . . 15
3.3 Indexing . . . 16
3.3.1 Result . . . 16
3.3.2 Comparison with other techniques in theory . . . 19

4 Case study - Wikipedia **20**
4.1 Overview of Wikipedia data . . . 20
4.2 Big Volume Management Strategies in Wikipedia . . . 20
4.3 Benefits Derived from Wikipedia's Big Volume Management . . . 21

A Reference **22**

B Appendix **22**

1 Member list & workload

No.	Full name	Student ID	Assignment	Percentage
1	Vo Truc Son	2252720	DBMS implementation, write report	20%
2	Truong Tran Duy Tan			20%
3	Truong Quang Nha	2152822	Indexing	20%
4	Bui Duc Manh Cuong	2252096	Write report, case study	20%
5	Nguyen Ngoc Phuong Mai	2211983	Write report, case study	20%

2 Theory - Big Volume Management

2.1 Definition

Big Data refers to datasets whose size is beyond the ability of typical database software tools to capture, store, manage, and analyze. In today's environment, the size of datasets that may be considered big data ranges from terabytes (10^{12} bytes) or petabytes (10^{15} bytes) to exabytes (10^{18} bytes). Common 3V characteristics of Big Data are Volume, Velocity, and Variety.

Volume refers to size of data managed by the system. Big volume means massive amount of data generated (mostly automatically) by different sources (for example, sensors, scanning equipment, measuring devices, social media, IoT devices, transactional systems). Big volume management refers to implementing strategies and technologies in order to efficiently store, process, and analyze vast amounts of data. Managing large data volumes is essential for organizations to harness the full potential of their data, leading to better decision-making, increased efficiency, compliance, and a competitive edge.

2.2 Characteristics

2.2.1 Scalability

Scalability describes a system's elasticity. It means to increase the number of materials when the workload is high, and vice versa. For example, e-commerce applications automatically reduce servers at night when traffic is low to save costs. There are 2 types of scalability, horizontal scaling and vertical scaling. The following table describes the properties of them.

Table 1: Horizontal scaling vs Vertical scaling

Property	Horizontal scaling	Vertical scaling
Description	Increase or decrease the number of nodes in a cluster (e.g. Kubernetes cluster) or system to handle an increase or decrease in workload	Increase or decrease the power of a system to handle increased or reduced workload
Example	Add or reduce the number of virtual machines (VM) in a cluster of VMs	Add or reduce the CPU or memory capacity of the existing VM
Execution	Scale in/out	Scale up/down
Workload distribution	Workload is distributed across multiple nodes.Parts of the workload reside on these different nodes	A single node handles the entire workload
Concurrency	Distributes multiple jobs across multiple machines over the network, at a go. This reduces the workload on each machine	Relies on multi-threading on the existing machine to handle multiple requests at the same time
Required architecture	Distributed	Any
Implementation	Takes more time, expertise, and effort	Takes less time, expertise, and effort
Complexity & maintenance	Higher	Lower

Table 1: Horizontal scaling vs Vertical scaling

Property	Horizontal scaling	Vertical scaling
Configuration	This requires modifying a sequential piece of logic in order to run workloads concurrently on multiple machines	No need to change the logic. The same code can run on a higher-spec device
Downtime	No	Yes
Load balancing	Necessary to actively distribute workload across the multiple nodes	Not required in the single node
Failure resilience	Low because other machines in the cluster offer backup	High since it's a single source of failure
Cost	High costs initially; optimal over time	Low-cost initially; less cost-effective over time
Networking	Quick inter-machine communication	Slower machine-to-machine communication
Performance	Higher	Lower
Limitation	Add as many machines as you can	Limited to the resource capacity the single machine can handle
Use case	When traffic spikes, Netflix automatically adds servers to the cluster to handle video streaming requests	Oracle database on a powerful physical server to handle 1 million transactions/second
Tools	AWS Auto Scaling, Google Cloud Instance Groups	Google Cloud SQL, Amazon RDS

2.2.2 Performance Optimization

Database performance optimization is the process of maximizing the efficiency and effectiveness of a database system. It involves ensuring that a database system can handle the required workload while maintaining fast response times and minimizing resource consumption.

There are some key metrics for optimization:

1. **Low latency:** refers to the time it takes for a database to respond to a query or request. In a big data application, low latency is critical to ensure that the application can respond to user requests in a timely manner.
    - Technology: Redis/Memcached to cache frequently accessed data (reduce latency from 2s to 10ms).
    - For example, Facebook uses a cache layer to load user profiles in a few ms.
2. **High Throughput:** Throughput refers to the number of transactions that a database system can handle per unit of time. High throughput is essential in big data applications, where the system needs to handle a large volume of requests.
    - Solution: use a distributed system like Apache Kafka (processes 1 million messages/second).
    - For example, Uber uses Kafka to handle real-time location of millions of drivers.
3. **Load balancing:** Round Robin, Least Connections, or Geo-based algorithms.

- Tools: NGINX, AWS Elastic Load Balancer.

There are some optimization techniques:

1. **Indexing:** involves creating an index on a column or set of columns in a table. Indexes allow the database system to quickly locate data, improving query response times.
2. **Partitioning:** involves dividing a large table into smaller, more manageable parts. Partitioning allows the database system to access data more efficiently and reduces the risk of data corruption.
3. **Caching:** involves storing frequently accessed data in memory, which can improve query response times and reduce resource consumption.

2.2.3 Fault tolerance and Reliability

Fault tolerance refers to the continued, uninterrupted functioning of systems even when one or more components fail or experience faults. Data reliability refers to the completeness and accuracy of data as a measure of how well it can be counted on to be consistent and free from errors across time and sources. There are some techniques to ensure these characteristics:

1. **Redundancy**
   - Multi-AZ Deployment: Deploy database across multiple Availability Zones (AWS RDS).
   - Example: Google Search Engine replicates data across 3 different data centers.
2. **Backup**
   - Snapshot: Automatically take daily snapshots (e.g. Veeam Backup for VMware).
   - Disaster Recovery: Restore system in 15 minutes with AWS DR Solutions.
3. **Automated Failover**
   - Example: When a Cassandra node dies, the system automatically redirects requests to another node without the user realizing it.

2.2.4 Automation and Orchestration

Automation is programming a task to be executed without the need for human intervention. Orchestration is the configuration of multiple tasks (some may be automated) into one complete end-to-end process or job. Orchestration software also needs to react to events or activities throughout the process and make decisions based on outputs from one automated task to determine and coordinate the next tasks. There are some techinques for automation and orchestration:

1. **Kubernetes**
   - Auto-healing: Automatically restart crashed containers.
   - For example, Spotify uses Kubernetes to manage 10,000+ microservices.
2. **CI/CD**
   - Pipeline: Jenkins/GitLab CI to build, test, and deploy code automatically.
   - For example, Netflix deploys hundreds of updates every day without downtime.

2.2.5 Security and Compliance at scale

- **Multi-Layer Encryption**
  - At Rest: AES-256 (AWS RDS) for petabyte-scale databases.
  - In Transit: TLS 1.3 for high-throughput APIs (1M+ requests/sec).
- **Granular Access Control:** Least Privilege via AWS IAM roles.

2.2.6 Cost optimization for massive workloads

- **Spot instances:** Batch processing at 70% lower cost (vs. on-demand).
- **Auto-shutdown:** Lambda terminates idle EC2 instances after 15 minutes.

2.2.7 High-volume data processing

- **Batch processing:** Hadoop handles Walmart's 1B+ daily transactions for supply chain analytics.
- **Real-Time Stream:** Apache Flink processes 10M events/sec (stock trading systems).
- **Centralized Monitoring:** ELK Stack aggregates logs from 1000+ servers.
- **Proactive Alerts:** Slack/Email alerts for CPU ¿90% or latency ¿500ms.

2.3 Benefits

Effective management of Big Data brings significant advantages to organizations, contributing to enhanced operational efficiency, improved decision-making capabilities, and strengthened competitiveness in the digital era.

2.3.1 Efficient Storage and Retrieval

Big Volume Management enables organizations to leverage modern storage technologies such as data lakes and distributed storage (like Hadoop distributed file system - HDFS, Amazon S3) to optimize storage capacity and accelerate data retrieval processes. *Example:* Facebook, with over 2.9 billion global users, applies Data Lake architecture to store images, videos, and user information efficiently, ensuring data retrieval time within just a few milliseconds.

2.3.2 Smarter Decision Making

Big Data Analytics allows organizations to make data-driven decisions based on real-time information, thereby improving business performance and reducing risks. *Example:* Amazon applies Machine Learning to analyze customer purchasing behavior and generate personalized product recommendations, significantly increasing online sales revenue.

2.3.3 Data Consistency and Security

Big Data Management ensures data consistency and security through various techniques such as:

- Multi-layer Encryption
- Automated Data Backup and Replication
- Granular Access Control Mechanisms

Example: Google Search Engine replicates data across three different data centers globally to prevent data loss in case of system failure.

2.3.4 Flexible Scalability

The system can easily scale horizontally or vertically according to real-time traffic or data growth, ensuring resource efficiency. *Example:* Netflix utilizes AWS Auto Scaling to automatically increase or decrease the number of servers to accommodate changes in user traffic.

2.3.5 Support for Advanced Analytics

Big Data provides a solid foundation for the development of Artificial Intelligence (AI), Machine Learning (ML), and Real-time Analytics, enabling organizations to create intelligent solutions. *Example:* Uber applies Big Data Analytics to predict customer demand and allocate drivers appropriately across different regions.

2.3.6 Cost Savings

Big Data systems optimize operational costs by using Spot Instances or Serverless Architecture for processing large volumes of data. *Example:* Airbnb reduces processing costs by up to 70% through the use of EC2 Spot Instances on AWS Cloud infrastructure.

2.3.7 Ensuring Availability and Reliability

Big Data systems are designed with auto-healing capabilities, load balancing, and automated failover mechanisms to ensure continuous service availability. *Example:* Banking systems implement Failover Clusters to ensure that transactions are uninterrupted even when a server encounters an error.

2.3.8 Enhanced Data Integration Capabilities

Big Data Management facilitates the integration of data from multiple sources (like IoT devices, social media, sensors) into a centralized data pool for comprehensive analytics. *Example:* Singapore's Smart City system collects and analyzes data from traffic cameras and environmental sensors to provide real-time alerts and optimize city operations.

2.3.9 Support for Regulatory Compliance

Big Data Management ensures compliance with international data protection standards such as GDPR, HIPAA, and PCI-DSS, thereby minimizing legal risks. *Example:* Financial and banking companies are required to store and process customer data according to GDPR regulations.

2.3.10 Improved Customer Experience

Big Data Analytics enables organizations to personalize customer experiences, thereby increasing customer satisfaction and loyalty. *Example:* E-commerce platforms like Shopee and Lazada analyze customer purchase history to recommend relevant products and personalized promotions.

2.4 Examples of Big Volume Management applications

Organization	Big Data Application	Technologies Used	Achieved Benefits
Netflix	Managing millions of video streams	AWS Auto Scaling, Load Balancer	Improved scalability and minimized downtime during high user traffic periods
Uber	Real-time location data processing of drivers	Apache Kafka, Redis	Optimized driver allocation and reduced waiting time for customers
Walmart	Transaction data analysis and supply chain management	Hadoop, Spark	Efficient supply chain management and accurate demand forecasting
Facebook	User data storage and fast data retrieval	Distributed Database, Cache Layer	Enhanced user experience through rapid data access
Grab	Customer behavior analysis and demand prediction	Real-time Analytics, Machine Learning	Personalized services, optimized pricing strategies, and effective promotional campaigns

3 Implementation on DBMS

3.1 Normal query

```
-- tim mikelong thuong
select * from wiki_views.wiki_people_views
where article_title = 'Mike_Long_(author)';

explain select * from wiki_views.wiki_people_views
where article_title = 'Mike_Long_(author)';
--
```

Figure 1: Normal query

- When we use normal query on a table with big data (in this we use dataset with 23 million records), the DBMS such as MySQL will look through the all the record to get our query, which take a bunch of wasting time. Also query process get more wasting time by many causes, such as:

Result Grid | Filter Rows: | Edit:

id	article_title	view_timestamp	views
1	Mike_Long_(author)	2023-04-01	23
3022419	Mike_Long_(author)	2023-06-01	24
5257712	Mike_Long_(author)	2023-10-01	28
6265587	Mike_Long_(author)	2023-01-01	21
6961967	Mike_Long_(author)	2023-07-01	19
7072543	Mike_Long_(author)	2023-11-01	34
9199867	Mike_Long_(author)	2023-02-01	20
10061890	Mike_Long_(author)	2023-05-01	20
12607259	Mike_Long_(author)	2023-08-01	29
12927906	Mike_Long_(author)	2023-12-01	27
17866059	Mike_Long_(author)	2023-03-01	31
18220391	Mike_Long_(author)	2023-09-01	26
23051366	Mike_Long_(author)	2023-04-01	23
26073784	Mike_Long_(author)	2023-06-01	24
28309077	Mike_Long_(author)	2023-10-01	28
29316952	Mike_Long_(author)	2023-01-01	21
30013332	Mike_Long_(author)	2023-07-01	19
30123908	Mike_Long_(author)	2023-11-01	34
32251232	Mike_Long_(author)	2023-02-01	20
33078221	Mike_Long_(author)	2023-04-01	23
33178790	Mike_Long_(author)	2023-05-01	20
38345559	Mike_Long_(author)	2023-08-01	29
38993881	Mike_Long_(author)	2023-12-01	27
39049714	Mike_Long_(author)	2023-06-01	24

eople_views1 ×

Figure 2: Data duplication

- Because of this problem, we need a lot of time to get all the result, definitely not efficiency.

- As you can see in figure 3, at line 14 in Action Output, we need 82 seconds to get 72 lines with many duplicated information and not arranged data.

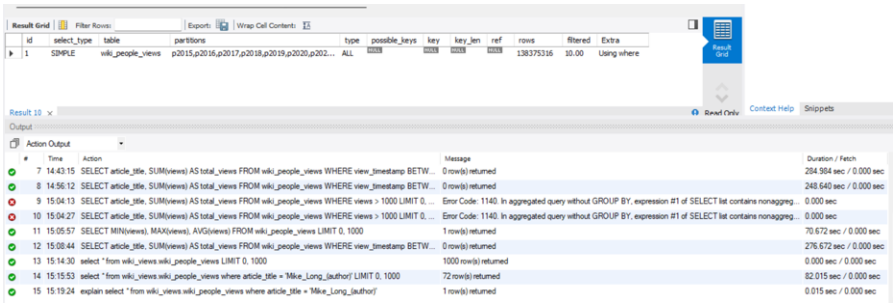

Figure 3: Query time for normal query

- Also if we use more complex query, the DBMS may not give us response due to wait time expired.
- In the figure below, we can see that to query "top 10 most view article in January 2023", it takes us 520 seconds to get the result

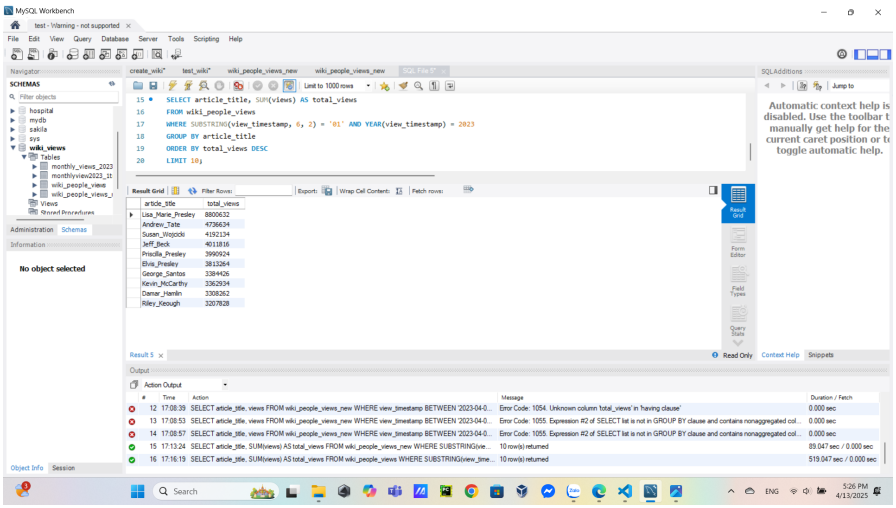

Figure 4: Query time for normal complex query

- Now we perform insert record into table "wiki_people_views", the output is show in figure below, line 23, 24.
- As you can see, it takes us 49 seconds to get the result
- Now we perform delete data from table "wiki_people_views", the output is show in figure below, line 25, 26.
- As you can see, it takes us 51 seconds to get the results.

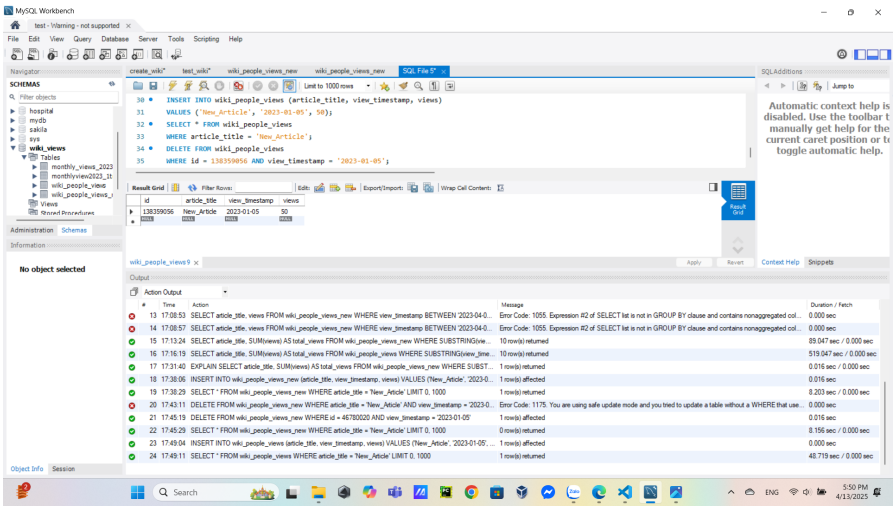

Figure 5: Query time for insert query

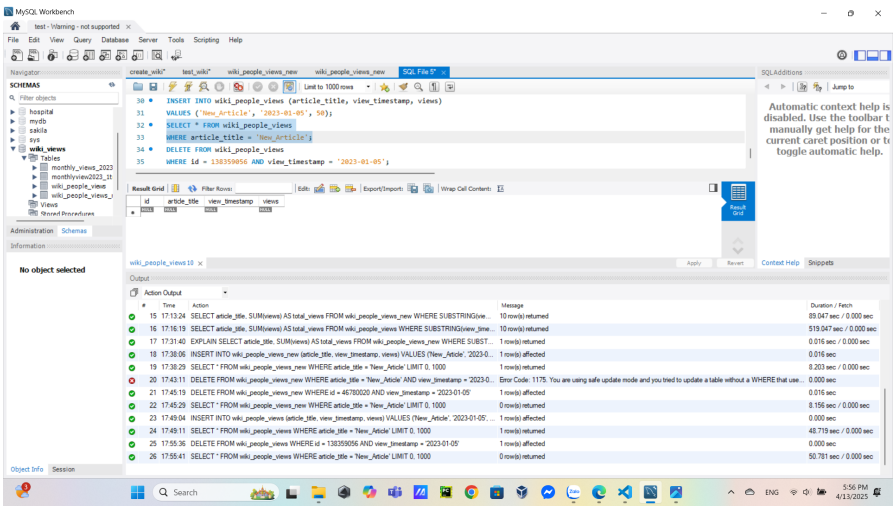

Figure 6: Query time for delete query

- This lead us to a need of a more efficient way to store data

3.2 Partition

3.2.1 Result

- The first approach we use is Partition. By separating the records before insert into table, we get less time in query.

The following is a block of SQL code:

```sql
CREATE TABLE wiki_people_views_new (
    id BIGINT AUTO_INCREMENT,
    article_title VARCHAR(255),
    view_timestamp DATE,
    views INT,
    PRIMARY KEY (id, view_timestamp)
) PARTITION BY RANGE (YEAR(view_timestamp) * 100 + MONTH(view_timestamp)) (
    PARTITION p202301 VALUES LESS THAN (202302), -- Thang 1/2023
    PARTITION p202302 VALUES LESS THAN (202303), -- Thang 2/2023
    PARTITION p202303 VALUES LESS THAN (202304), -- Thang 3/2023
    PARTITION p202304 VALUES LESS THAN (202305), -- Thang 4/2023
    PARTITION p202305 VALUES LESS THAN (202306), -- Thang 5/2023
    PARTITION p202306 VALUES LESS THAN (202307), -- Thang 6/2023
    PARTITION p202307 VALUES LESS THAN (202308), -- Thang 7/2023
    PARTITION p202308 VALUES LESS THAN (202309), -- Thang 8/2023
    PARTITION p202309 VALUES LESS THAN (202310), -- Thang 9/2023
    PARTITION p202310 VALUES LESS THAN (202311), -- Thang 10/2023
    PARTITION p202311 VALUES LESS THAN (202312), -- Thang 11/2023
    PARTITION p202312 VALUES LESS THAN (202401), -- Thang 12/2023
    PARTITION p_future VALUES LESS THAN (MAXVALUE) -- Cac gia tri sau nay
);
```

- With this approach, we can see that the data is organized, duplicated information are less and returned info in alphabetical order (figure 7).

Result Grid | Filter Rows: | Edit:

id	article_title	view_timestamp	views
32928521	Mike_Long_(author)	2023-02-01	20
41594713	Mike_Long_(author)	2023-03-01	31
23727776	Mike_Long_(author)	2023-04-01	23
33790544	Mike_Long_(author)	2023-05-01	20
26751073	Mike_Long_(author)	2023-06-01	24
30690621	Mike_Long_(author)	2023-07-01	19
36335913	Mike_Long_(author)	2023-08-01	29
41949045	Mike_Long_(author)	2023-09-01	26
28986366	Mike_Long_(author)	2023-10-01	28
30801197	Mike_Long_(author)	2023-11-01	34
36656560	Mike_Long_(author)	2023-12-01	27
NULL	NULL	NULL	NULL

e_views_new 1 ×

Figure 7: Using partition in querying

- We also see the huge different in query time by this approach. By using partition in table due to month, we get the exact info in just 19 seconds, less than 4 times we need to query the information (figure 3).

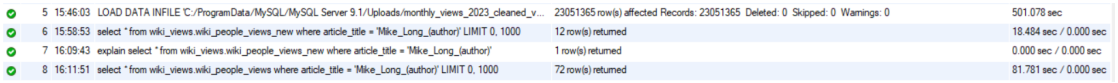

Figure 8: Using partition in querying time

- With more complex query, we still get the result very fast. As you can see in below figure, we need 90s to get the "top 10 most views article in January 2023" results, nearly 6 times faster than normal way (figure 4).

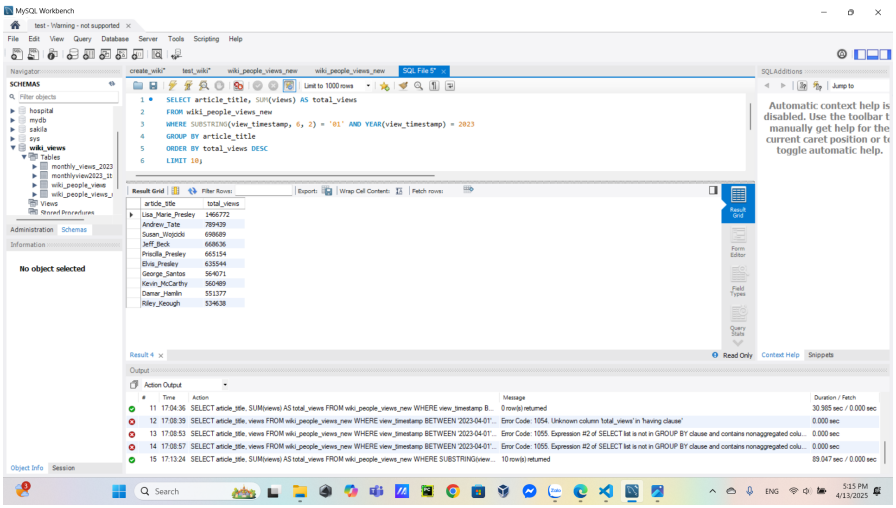

Figure 9: Using partition in complex query

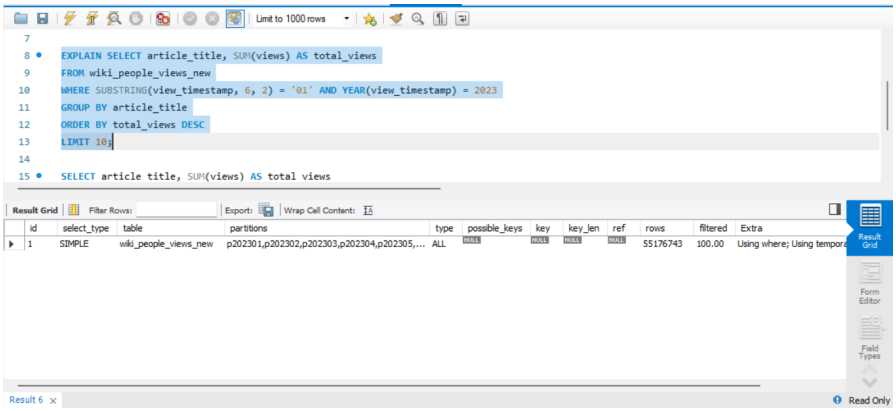

Figure 10: Explain partition in complex query

- Now we perform insertion, you can see the output in figure below (line 18,19)
- We get the results in 8 seconds, rather than 49 seconds with normal ways.

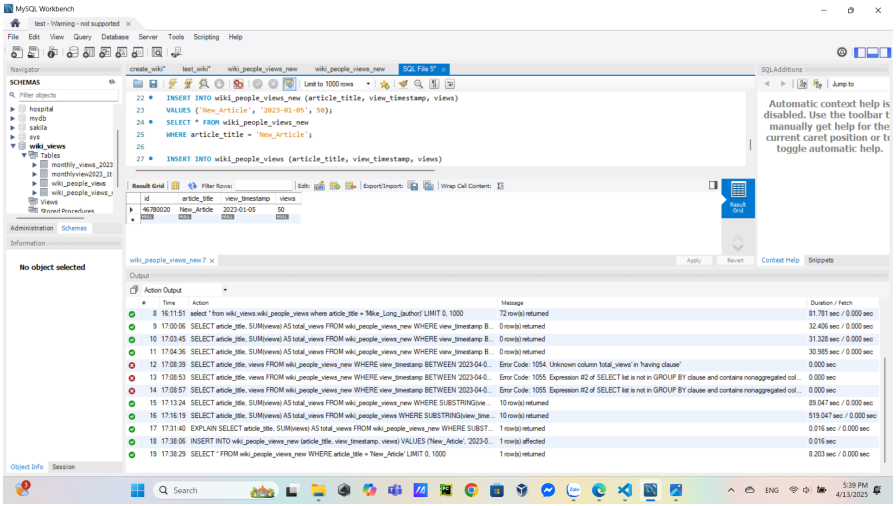

Figure 11: Partition insertion

- Now we perform updating by delete a record, you can see the output in figure below (line 21,22)
- Now you can see, we get the results faster nearly 7 times due to the compare with the normal query ways.

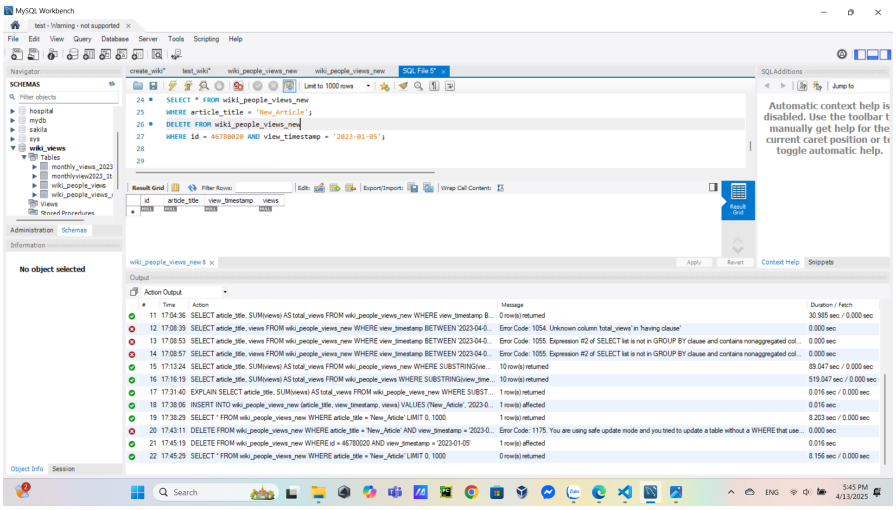

Figure 12: Partition deletion

3.2.2 Comparison with other techniques in theory

Only techniques that may have an impact on optimization are discussed.

- **Vertical scaling:**
  - With SELECT queries, increasing hardware resources can improve performance, but it is unlikely to provide a speedup of 4-6 times due to the limitation in the capability of having more optimized hardware solutions. With INSERT and DELETE operations, having more powerful hardware resources can improve performance, but it depends on the architecture of the system.
  - On the other hand, partitioning is more targeted for SELECT queries and it is likely to provide a more consistent improvement for INSERT and DELETE operations.
- **Horizontal scaling:**
  - With SELECT queries, adding more nodes without sharding might have a limited impact on the performance of a query; the database might be able to distribute the load better, but the query still needs to scan the same amount of data. With INSERT and DELETE operations, horizontal scaling can improve performance if the database can parallelize these operations across multiple nodes. However, the effect depends on the database architecture and the degree of parallelism.
  - On the other hand, partitioning is more targeted for SELECT queries and it is more predictable about the outcome of optimization.
- **Kubernetes:** This technique can help manage and scale the database infrastructure, which could improve performance by ensuring sufficient resources. However, it does not directly optimize the queries and operations.
- **Centralized monitoring:** Monitoring can help identify slow queries, slow operations, or other performance issues, but it does not directly improve queries and operations execution time.

3.3 Indexing

The second approach we apply is Indexing. By creating indexes on the relevant columns before executing queries, we significantly reduce query execution time by enabling faster data retrieval.

The following is a block of SQL code:

```sql
CREATE TABLE wiki_people_views_indexing (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    article_title VARCHAR(255),
    view_timestamp DATE,
    views INT
);

-- Indexing on view_timestamp to speed up date-based queries
CREATE INDEX idx_view_timestamp ON wiki_people_views_indexing(view_timestamp);

-- Indexing on article_title to speed up article-based queries
CREATE INDEX idx_article_title ON wiki_people_views_indexing(article_title);
);
```

3.3.1 Result

- With this approach, we can see that the data are efficiently indexed, there are no redundant information, and the retrieved results are much faster.

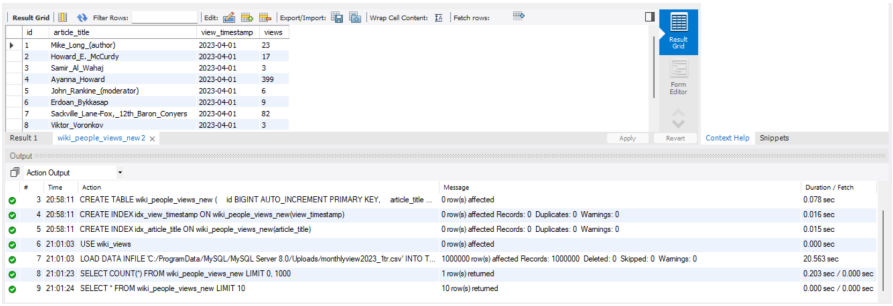

Figure 13: Using indexing in querying

- Before indexing, when we query a record, we need, for example, 11 seconds to get the results.
- When we insert a new record, it takes us the next 11 seconds to find that record back.
- Then, we perform a deletion to that new record. And the DBMS MySQL return us with another 11 seconds to check answer that the output was deleted.

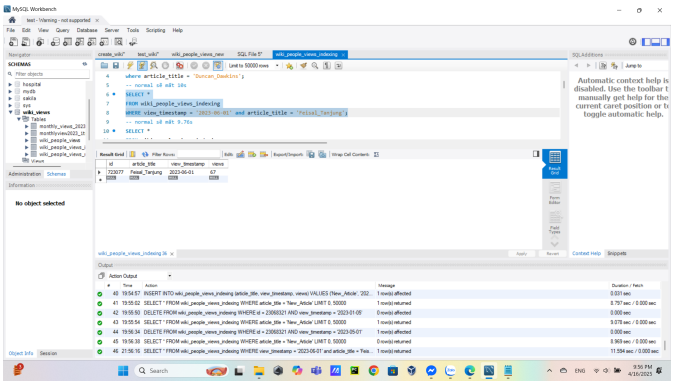

Figure 14: Querying before indexing

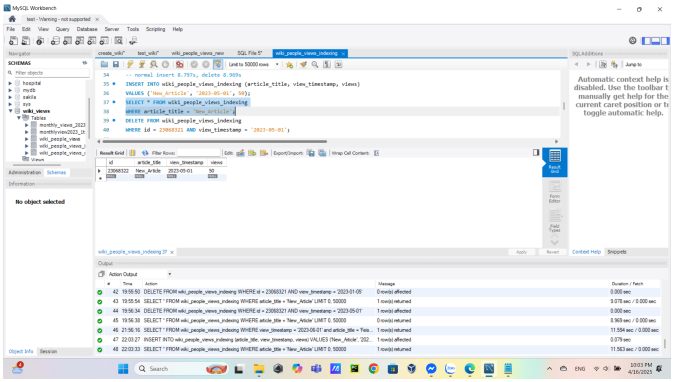

Figure 15: Insertion before indexing

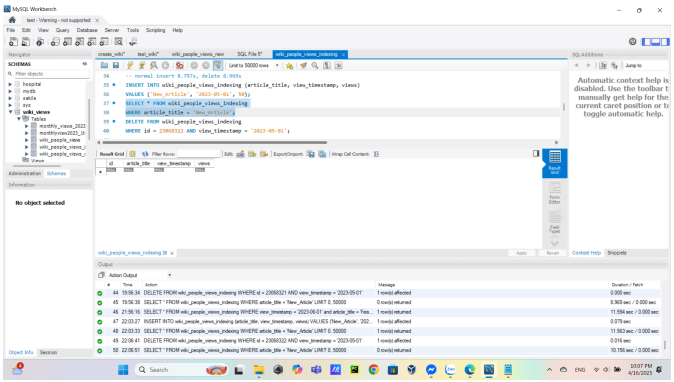

Figure 16: Deletion before indexing

- Now we should create index to hope for a better performances.

- We can see that after indexing, the query waiting time has shorten a quite big range, from 11 seconds to a 0.2 seconds.

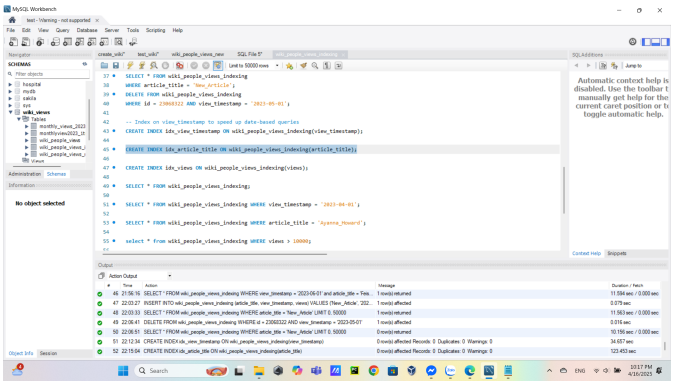

Figure 17: Creating index

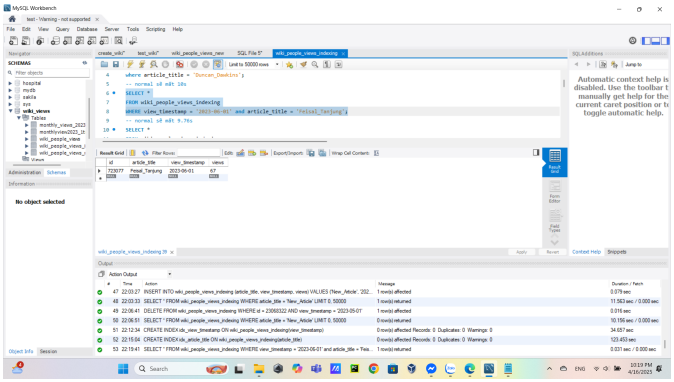

Figure 18: Query after indexing

- We can also see that the insertion and deletion has also increase the performances, the responses time is so quick that seems to be immediatly to the client.

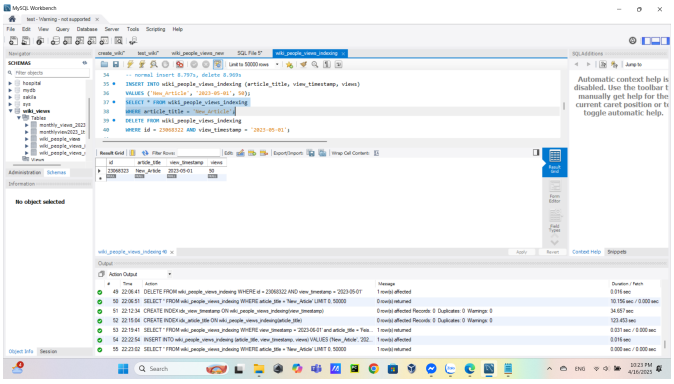

Figure 19: Insert after indexing

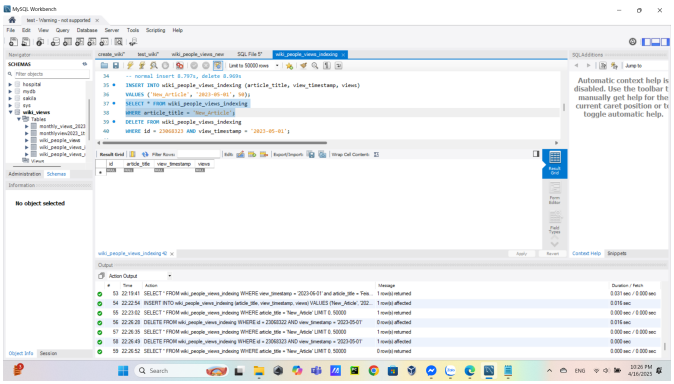

Figure 20: Delete after indexing

3.3.2 Comparison with other techniques in theory

The primary advantage of indexing is speed. It significantly reduces the number of disk I/O operations by allowing the DBMS to quickly locate the data without scanning the entire table, which is especially helpful in large datasets. So, in this case, only techniques that may affect speed are discussed.

- **Sequential Scanning:** Sequential Scanning is the slowest method, as it requires reading every row in a table, resulting in linear time complexity O(n). In contrast, indexing, particularly using B+ + trees or hash indexes, greatly improves speed by allowing the DBMS to locate data in logarithmic time, O(log n), or even constant time, O(1), for exact matches using hash indexes.

- **Hashing:** Hashing can be very fast for exact lookups, often achieving constant-time access. However, its performance can degrade due to collisions, and it does not support range queries, making it slower or unusable in such scenarios where indexing still performs efficiently.

- **Caching:** Caching provides the fastest possible access, constant time, when the data is already in memory (a cache hit), but it is limited by memory size and only benefits frequently accessed data. Unlike indexing, caching does not inherently improve the speed of complex queries or those that access a wide range of data.

- **Vertical scaling:**

Overall, in terms of consistent and reliable speed across a variety of query types, indexing strikes the best balance, offering significant performance improvements over sequential scans and broader applicability than hashing or caching.

4 Case study - Wikipedia

Based on our chosen dataset for implementation on DBMS, we chose Wikipedia as an example of big volume management. Wikipedia is one of the largest and most visited websites in the world, serving millions of users and managing an immense volume of data. The platform relies on its ability to store, process, and deliver data efficiently while ensuring scalability, reliability, and availability. This case study explores how Wikipedia manages its massive volume data and the benefits derived from its strategies.

4.1 Overview of Wikipedia data

Wikipedia operates on a large scale of data. There are more than 210 million (210M) running pages currently, which serves more than 300 language versions. Monthly (from March 2023 to March 2025), there are:

- Over 9 million (9M) content edits made by human users
- Over 900 thousand (900K) pages created
- About 170 thousand (170K) newly self-created registered users
- About 7 to 9 billion (7B - 9B) human user page views on English Wikipedia pages
- Approximately 2.5 to 4 GB net bytes difference made by each edit or revision

4.2 Big Volume Management Strategies in Wikipedia

To manage such a large volume of data, Wikipedia applies several critical big data management strategies:

Strategy	Implementation in Wikipedia	Benefit
Distributed Database Architecture	Wikipedia uses MySQL as its primary relational database, distributed across multiple data centers to balance the load and minimize latency.	Ensures scalability, data consistency, and fast response time.
Caching Layer	Varnish Cache is used for content delivery, significantly reducing database read operations.	Reduces server load, improves page load speed for users worldwide.
Load Balancing	Load balancers distribute incoming traffic across multiple servers based on location and demand.	Increases availability and prevents server overload during traffic spikes.
Redundancy and Backup Systems	Regular backups and database replication are implemented across global servers.	Enhances fault tolerance and disaster recovery capability.
Open-source Infrastructure	Wikipedia utilizes open-source technologies such as MediaWiki, Apache, and MariaDB.	Reduces operational costs while maintaining flexibility for system upgrades and customization.

4.3 Benefits Derived from Wikipedia's Big Volume Management

By applying the strategies mentioned earlier, Wikipedia achieves several operational and strategic benefits:

- High system availability and reliability, even during peak user access periods.
- Optimized storage and retrieval processes, enabling rapid access to large-scale information.
- Cost-effective infrastructure management through the use of scalable and open-source solutions.
- Enhanced user experience through fast content delivery and consistent system performance.
- Support for global data integration across various languages and regions.

A Reference

References

[1] *Wikimedia statistics*. Link: https://stats.wikimedia.org/#/en.wikipedia.org

[2] Gandomi, A., & Haider, M. (2015). *Beyond the hype: Big data concepts, methods, and analytics*. International Journal of Information Management, 35(2), 137–144. Link: https://doi.org/10.1016/j.ijinfomgt.2014.10.007

[3] Chen, M., Mao, S., & Liu, Y. (2014). *Big Data: A survey*. Mobile Networks and Applications, 19(2), 171–209. Link: https://doi.org/10.1007/s11036-013-0489-0

[4] AWS (Amazon Web Services). (2023). *Scaling your architecture: Horizontal and Vertical Scaling*. Link: https://aws.amazon.com/architecture/scaling-best-practices/

[5] IBM. (2022). *What is Big Data?*. Link: https://www.ibm.com/topics/big-data

[6] Netflix Technology Blog. (2021). *How Netflix handles billions of requests every day*. Link: https://netflixtechblog.com/

[7] Uber Engineering. (2020). *Building Reliable Realtime Systems with Apache Kafka at Uber*. Link: https://eng.uber.com/kafka-reliable-realtime/

B Appendix

GitHub repo: https://github.com/VoTrucSonBKiter/Big-Volume-Management.git