Performance, Patterns, and Path from SQL to NoSQL

Meet Solanki (solank94@uwindsor.ca), Dr. Christie Ezeife (cezeife@uwindsor.ca) and Yusriyah Rahman (rahman4n@uwindsor.ca)

School of Computer Science, University of Windsor, Windsor, ON N9B 3P4, Canada

Abstract

Understanding how databases handle different workloads is essential for modern data-driven systems. This project evaluates three storage models—**row-store SQL**, **column-store SQL**, and **MongoDB**—to see how their structures affect performance.

What we tested

- Transactional tasks (OLTP): read/write operations, joins
- Analytical tasks (OLAP): scans, filters, aggregations
- Pattern mining using the **Apriori algorithm**
- Flexibility during migration to **MongoDB**

Key findings

- **Row stores** handle transaction-heavy operations most efficiently.
- **Column stores** deliver faster analytical processing.
- **MongoDB** scales best when workloads mix or grow.
- Overall, the results show that data organization has a direct impact on performance and that each model excels under different conditions.

Overall, the results show that data organization has a direct impact on performance and that each model excels under different conditions.

Introduction

Row Store → Column Store → Apriori Algorithm → MongoDB

Modern data systems must support two competing demands: **fast transactions** and **rich analytical processing**.

Workload types

- **OLTP (Online Transaction Processing)** handles frequent, real-time inserts, updates, and lookups.
- **OLAP (Online Analytical Processing)** supports large-scale filtering, scanning, and aggregation for data insights.

Our approach

To examine this trade-off, we built a **University Database System** using both SQL and NoSQL architectures and evaluated how each responds to different workload types.

What this study shows

Database architecture has a direct impact on **query performance**, **scalability**, and **pattern-mining consistency** across diverse data operations.

Related Work

Research on database performance has explored how different storage models and architectures impact query speed, scalability, and analytics.

Key themes in prior studies

- Comparisons of **row-store vs. column-store** efficiency
- Use of **Apriori** and related algorithms for pattern mining
- Evaluation of **NoSQL systems** for scalability and flexible schema design

Motivation

These works highlight gaps between relational and non-relational approaches, inspiring our comparison of **SQL (row + column)** and **MongoDB** models in both performance and pattern discovery.

Research Work	Focus / Contribution	Limitation
Abadi et al., 2008 (ACM SIGMOD)	Row vs. column store comparison	Limited to analytical queries
Agrawal & Srikant, 1994 (VLDB)	Frequent pattern mining using Apriori	Computationally expensive
Shahnawaz et al., 2025 (ACM Surveys)	Big data analytics & NoSQL overview	Lacked migration testing

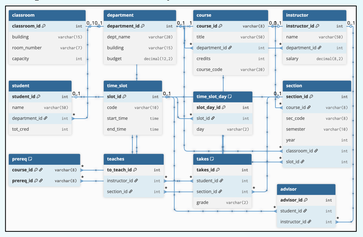
Methodology

We evaluated SQL and NoSQL systems using a controlled benchmarking environment and a shared university schema.

Schema Design and Storage Models

University Information System schema implemented in three forms:

- **Row-Oriented MySQL (univ_row)** — traditional relational layout
- **Column-Oriented MySQL (univ_col)** — decomposed attribute tables
- **MongoDB** — collections mirroring relational entities

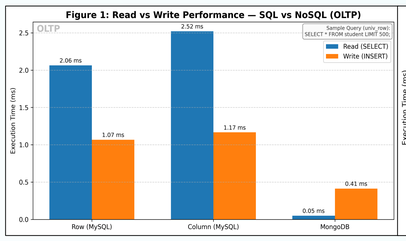


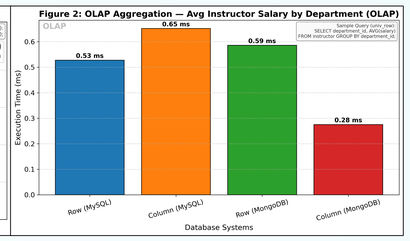
Purpose

Compare how storage format impacts access speed and query execution.

OLTP vs OLAP Benchmarking — Comparison Table

Aspect	OLTP Benchmarking (Transactional Workload)	OLAP Benchmarking (Analytical Workload)
Purpose	Simulates real-time operations and frequent transactions	Evaluates analytical query performance and large-scale data processing
Typical Queries	Multiple SELECT and INSERT on student records	Aggregations such as "Average instructor salary per department"
Focus Area	Measures execution time for simple transactions	Measures speed of complex analytical computations
Performance Factors Observed	I/O overhead, commit latency, row access speed	Aggregation efficiency, scan performance, computation time
MySQL Behavior	Good for rapid row-level operations; lower latency	Slower for large aggregations due to row-based layout
MongoDB / Column Store Behavior	Higher I/O overhead for frequent commits	More efficient for analytical tasks due to vertical alignment
Best Use Case	Real-time systems: student records, transactions	Reporting, dashboards, departmental analytics

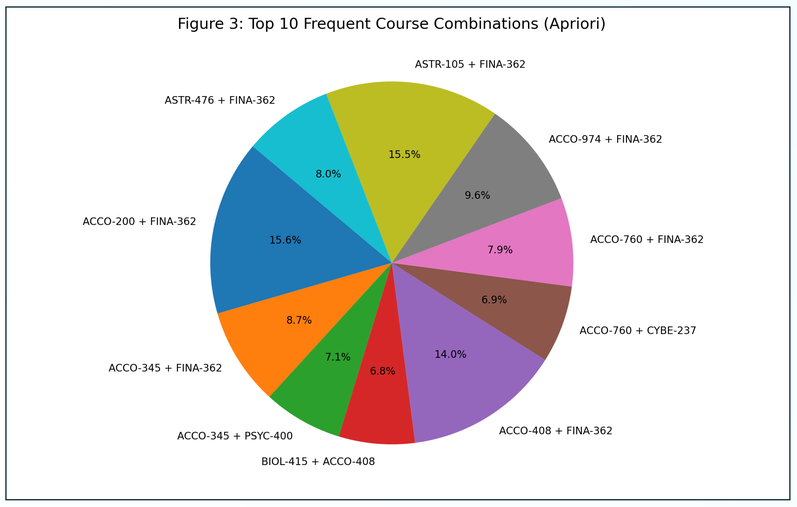




Pattern Mining (Apriori Algorithm)

Process summary:

- Enrollment data exported to CSV
- Preprocessed using **Pandas** + **MLXtend**
- Frequent itemsets + association rules evaluated
- Ensured pattern consistency across all models



Results

Our benchmark tests revealed clear differences in how each database model handled transactional and analytical workloads.

OLTP (Transactional Performance)

- **Row-Oriented MySQL** showed the fastest read/write speeds due to contiguous row storage.
- Column-Oriented MySQL and MongoDB were slower for frequent, small transactions.

OLAP (Analytical Performance)

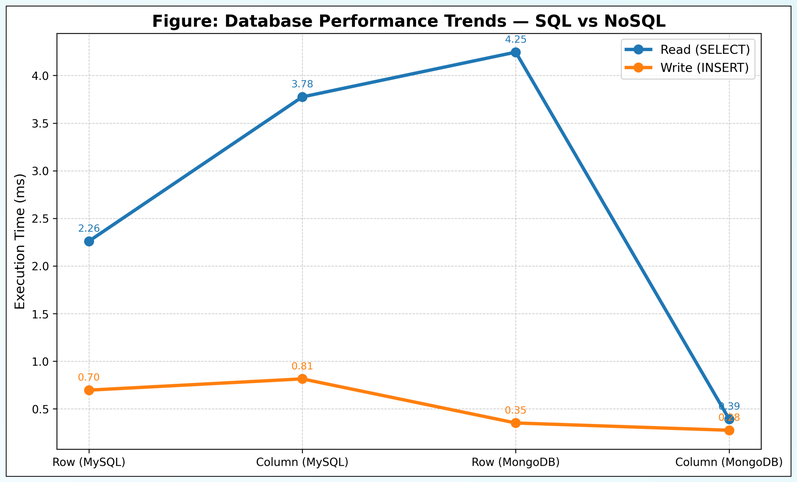
- **Column-Oriented MySQL** outperformed all models on scan-heavy aggregation queries.
- Row-Oriented MySQL was less efficient for large analytical tasks due to row-based alignment.

MongoDB Behavior

- Delivered **moderate performance** on both workloads.
- Provided **superior scalability**, schema flexibility, and adaptability for semi-structured data.

Overall Trend

SQL systems showed **more predictable execution times**, while MongoDB offered **greater elasticity** for larger, mixed, or evolving datasets.



Conclusion

This study shows that database performance is strongly shaped by how data is organized and accessed.

Key takeaways

- **Row-oriented SQL** systems are best suited for OLTP workloads that rely on fast, frequent reads and writes.
- **Column-oriented SQL** systems excel in OLAP scenarios by optimizing scans and aggregations.
- **MongoDB** provides flexible, scalable handling of semi-structured or rapidly evolving data.

Overall insight

No single database model is universally optimal — selecting the right architecture depends on the workload's query profile and data structure.

Future directions

We plan to extend this work by evaluating:

- distributed and cloud-scale deployments
- real-time analytical processing
- hybrid SQL–NoSQL integration strategies

These next steps aim to refine adaptive database design for large-scale systems.