ORIGINAL RESEARCH ARTICLE UHD JOURNAL OF SCIENCE AND TECHNOLOGY

Evaluating Aggregate Functions and Machine Learning Integration: A Comparative Analysis of Performance, Security, and NoSQL Connectivity in Oracle, SQL Server, and MySQL

Dana Lattef Hussein

IT Department, Computer Science Institute, Sulaimani Polytechnic University, Sulaymaniyah, Kurdistan Region, Iraq.

ABSTRACT

This paper is a comparison study on aggregate functions and windows function between the three major Relational Database Management Systems (RDBMSs): Oracle, SQL Server, and MySQL. These functions are essential to handle a huge data set and prepare it for effective analysis. The research is conducted to analyse the performance of these systems, their utilization of resources, while executing aggregate queries. Further, this paper examines the integration of machine-learning abilities and NoSQL database connectivity within these platforms. All these were measured under a constant benchmarking framework. It also discusses the analysis on how indexing affects query performance and the integration of machine-learning (ML) models with these databases. The results are indicative of considerable performance variation, resource efficiency, and ML integration among the three RDBMSs. Oracle is the best solution for implementing complex aggregations and ML integration, making it the best alternative to work on large datasets. Where MySQL is very efficient for most simple tasks, it lacks advanced features and does not have native ML support. It further provides optimization strategies for each RDBMS and gives insight into securing data and integrating with NoSQL databases. This research is set out to guide database administrators and developers in choosing the most appropriate RDBMS in relation to their specific needs in aggregation, ML, NoSQL integration. However, the factor of indexing is generally what brought most success to query optimization in these databases: Oracle, SQL Server, and MySQL. Among these, Oracle still was significantly outdoing both others, which further improved by indexing. In general, MySQL was less performant and lacked some functionality in window functions. Aggregation queries seem to profit more from indexing, but the less improvement was seen for window functions (STRING_AGG). All in all, indexing is a very effective technique in optimizing query efficiency.

Index Terms: Oracle, SQL Server, MySQL, Aggregate Functions, Indexing, Machine Learning, Non-relational Data-base Integration

1. INTRODUCTION

In modern database systems, the vast size of databases necessitates having tools that can efficiently provide the desired information [1]. Much of this work is done through aggregate functions, which allow us to do some action within groups of data [2], [3]. Which database management system (DBMS) you favor, and how you carry out query optimization, tend to hinge on how well each system supports aggregate functions [1], [4].

Relational database management system (RDBMS) such as Oracle, SQL Server, and MySQL are major in use in practically every industry or domain of human endeavor [5]. Each RDBMS comes with some distinct features and offers

Access this article online

DOI: 10.21928/uhdjst.v8n2y2024.pp7-23

E-ISSN: 2521-4217
P-ISSN: 2521-4209

Corresponding author's e-mail: Dana Lattef Hussein, Lecturer, IT Department, Computer Science Institute, Sulaimani Polytechnic University, Sulaymaniyah, Kurdistan Region, Iraq. Phone Number: 07701463490. E-mail: dana.hussein@spu.edu.iq

Received: 23-07-2024 Accepted: 11-09-2024 Published: 22-09-2024

strength in particular clustered task [4], [5]. This paper explores these features by presenting standard functions, such as COUNT, SUM, AVG, MIN, and MAX and up to a more advanced functions, such as MEDIAN, STRING_AGG, and PERCENTILE_CONT, from each point of view of handing these functions by each RDBMS [6]. The subtlety of all these functions can help each database administrator or developer see which RDBMS best suits their needs for achieving set objectives in designing database specifications[4]–[6]. However, advanced data management often integrates machine learning (ML) workflows that enable predictive analysis, pattern recognition and data-informed decisions [7], [8]. Each of these databases provides a different degree of ML integration using in-database ML algorithms, external tool integration or cloud-based ML services [8]. Exploring the levels of these capabilities will reveal the strengths and limitations of each RDBMS in advanced analytics [8]. RDBMS data models lack the flexibility to set up and store data for analytics capable of managing big data and various data types of new data types, such as structured, semi-structured and unstructured data, have become popular [2], [9]. However, data models are highly optimized to handle and process specific data using a standardized relational data model [9]. As relational databases are designed to govern the lifecycle of data, large amounts of unmanaged data attached to the system can lead to quality issues on the applications that rely on this data [9], [10]. That's why it's highly recommended to use new storage or computing models designed for unstructured and semi-structured data [10]. Integrating non-relational databases called NoSQL databases with RDBMS data can overcome the downsides of RDBMS data models as they can store extra data, manage data quality and provide access to applications designed for semi-structured data types [11]. The most popular NoSQL database systems include key-value stores, document-oriented databases, and graph databases, which have their own advantages and disadvantages [12]. Therefore, can be used this flexibility and scalability of the NoSQL environment to support analytical-engineering objectives. Security is one of the key concerns. It is important to make sure that trusted data have been kept online at the right time, without being accessible to unauthorized approach or some any threats [12].

The major reason for such a research initiative is that no comprehensive, in-depth study exists comparing these three major RDBMS platforms in the context of their execution of aggregate functions with the big dataset, embedding ML models, and linking with NoSQL data stores. This also justifies research on the optimization techniques that can be applied to enhance query performance measures across these systems.

The objectives of this study are as follows:

- Compare the performance of aggregate functions against window functions across Oracle, SQL Server, and MySQL on the one billion records.
- Ability of the integration of in-database ML algorithms.
- Find out the connectivity and integration of these RDBMS with NoSQL databases.
- Research on and suggest optimization techniques that would enhance the performance of queries through indexing and materialized views.

By meeting these goals, this paper will therefore help database administrators and developers to select the most suitable RDBMS for their needs in aggregation, ML integration, and NoSQL connectivity.

This paper provides a detailed comparison between Oracle, SQL Server, and MySQL on the following aspects:

1.1. Capabilities of Aggregate Functions

Detailed breakdown of the capabilities offered by each DBMS:

1. Oracle: Window functions are supported for more complex aggregation needs. Oracle has a comprehensive set of aggregate functions including standard aggregate functions (COUNT, SUM, AVG, MIN, MAX) and conditional aggregate functions (RANK, DENSE_RANK, PERCENT_RANK), as well as complex aggregate functions for hierarchical aggregation [3], [11].
2. SQL server: Along with the strong aggregation functions support such as Oracle, SQL Server supports some additional detailed functions such as the VARIANCE, COVARIANCE and STDEV functions for more detailed statistical analysis [13].
3. MySQL: It also provides a wide range of standard aggregate functions. However, its window functions are lacking and do not have the advanced functionality present in Oracle and SQL server [3], [14].

1.2. Performance Comparison

Evaluating the performance aspects of aggregate functions:

1. Benchmarking: Identical aggregate queries will be performed over the three DBMS executed on tables with similar data structures. A comparison of the time and number of consumed resources (CPU, memory) will be made to provide a report and assessment of performance [15].

2. Performance variables: To measure the performance of each DBMS, factors such as the amount of data processed, utilization of indexes, and the nature of the aggregation metric (simple metrics vs. window functions) will be considered [15].

1.3. Optimization Techniques

Exploring optimization techniques for improving aggregate function performance:

1. Indexing: Utilizing appropriate indexes on columns involved in aggregate operations can significantly improve query execution speed. Each DBMS-specific indexing strategy will be discussed [5], [16].
2. Materialized views: Pre-aggregation and initial storage of results for queries materializing in views can provide substantial boosts for queries that run frequently. Feasibility and bound on materialized views are. Materialized views in each DBMS will be examined [16].
3. Partitioning: Partitioning tables can optimize queries targeting specific subsets of data, potentially boosting performance for aggregate operations [17], [18].

1.4. ML Integration

Digging into support for in-database ML algorithms, integration with external ML toolkits, and cloud-based ML services [19], [20]:

1. Oracle: A suite of products called oracle machine learning (OML) provides strong in-database ML support: OML enables users to build, train, and deploy models using SQL and Procedural Language/SQL [8], [19], [20].
2. SQL server: Embedding python and R scripts into T-SQL, as well as embedding it in Azure ML [8], [19], [20].
3. MySQL: Even though MySQL currently does not include ML capabilities, it can be combined with various tools, platforms, and services such as Python, R, AWS SageMaker, Google AI Platform, and Azure ML [8].

1.5. Non-relational Database Connectivity

Assessing the integration capabilities with NoSQL databases:

1. SQL server: PolyBase for querying both Hadoop and Azure Blob Storage, SQL Server Integration Services for transforming data with NoSQL databases such as MongoDB or Cassandra, and easy integration with Azure Cosmos DB [21].
2. MySQL: Using MySQL Shell for JSON/BIGSON and MySQL Connectors for Hadoop and Cassandra, also several third-party tools are available such as Apache Sqoop and Talend [22].
3. Oracle: Support of Oracle NoSQL Database, Oracle Big Data SQL and Oracle Spatial and Graph, leveraging all Oracle database technologies for diverse non-relational data types [22], [23].

1.6. Security Comparison

Evaluate the integrity, confidentiality and availability of the security features in different DBMS. Security of the DBMS is of a key as any DBMS (Table 1).

1.6.1. Data encryption at rest

Data encryption at rest is a process that encrypts files and documents such that other than the document's owner, anyone attempting to view them is rendered with useless files unless the correct key is provided. This method helps entirely remove data leakage, unauthorized entry and physical theft aside from a situation where an attacker has infiltrated your key management system and has access to the key.

1.6.2. Data Encryption in transit

Encryption in transit refers to the encryption characteristics in a network while the transmitted data are moving from source to destination, and the data may not be encrypted in the source and destination storage systems [24].

1.6.3. Access control

A DBMS must provide some kind of security mechanism to prevent unauthorized access to the database. The DBMS creates user accounts, and controls the login process, to accomplish this [25].

1.6.4. Authentication

It authenticates that a user logs in according to the privileges given to perform to database activities. This prevents access to sensitive data by asking for proper authentication [25].

1.6.5. Auditing

It helps detect, in a timely way, unauthorized acts and activities by authorized users [24].

1.6.6. Data masking

Data masking changes the structure of sensitive information to produce fake versions of a company's data [26].

1.6.7. Compliance

Data compliance is the act of ensuring that an organization and any of its associated systems adhere to legal, regulatory and operational requirements regarding data [26].

1.6.8. Intrusion detection

An intrusion detection system (IDS) is any application that monitors network traffic for malicious activities and known threats. An IDS can monitor network traffic on suspicious

TABLE 1: Security features comparison

Security Feature	Oracle	SQL Server	MySQL
Data Encryption at Rest	TDE, Advanced Encryption	TDE, Always Encrypted	TDE (Enterprise Edition)
Data Encryption in Transit	SSL/TLS	SSL/TLS, Always encrypted	SSL/TLS
Access control	RBAC, Fine-grained access control, Oracle label security	RBAC, Row-level security	RBAC
Authentication	Kerberos, LDAP, SAML, Multifactor Authentication	Windows Authentication, Kerberos, Azure AD	Pluggable Authentication, LDAP
Auditing	Oracle Audit Vault, Database Vault	SQL Server ATP	MySQL Enterprise Audit (Enterprise Edition)
Data masking	Data Redaction, Data Masking	Dynamic Data Masking	Static Data Masking (Enterprise Edition)
Compliance	PCI DSS, HIPAA, GDPR, SOX	PCI DSS, HIPAA, GDPR, SOX	PCI DSS, HIPAA, GDPR (Enterprise Edition)
Intrusion detection	Database Firewall, Advanced Security Options	ATP	Third-party tools

TDE: Transparent data encryption, RBAC: Role-based access control, LDAP: Lightweight directory access protocol, ATP: Advanced threat protection, PCI DSS: Payment Card Industry Data Security Standard

hosts or network segments where malicious activities are likely to occur. When identified, an IDS notifies the IT and security teams about potential security risks and threats [27].

2. LITERATURE REVIEW

Comparative study on RDBMSs has been a notable part of the research focus due its impatience in dealing with data for almost any application. This literature review concentrates the result of several studies being conducted in the same field, with providing some of their range of research and their field of study, methodologies, results and limitations, as shown in the table below see (Table 2). In a broad comparison of SQL and NoSQL databases, Lee *et al.* (2019) address the fact that each type of DB is still relevant in its context. This provided a new perspective for this study to address RDBMS connectivity in NoSQL databases that demonstrate the flexibility of these applications. However, they did not study which specific features from different DBMSs lead to how performance is impacted, something we try to address in this study.

Islam (2017) investigated performance efficiency and response time while managing real-time huge data. He found that MYSQL yielded the best results in executing performances when dealing with huge structured/semi-structured/unlike data. Islam's work focused on insert operations only, and here the scope is extended to cover a wider range of operations including complex queries that are important when comparing performance across different DBMS for aggregate functions.

In a study by Matallah 2021: MySQL versus MongoDB, he states that you would better use MongoDB for unstructured data than the structured one, and it trades off with MySQL as well. This distinction underlies a study of the execution strategies for aggregate functions in various DBMSs on diverse data environments, which motivates our work in this paper. Nonetheless, Matallah's study is limited to simpler systems and warrants further investigations which have been pursued by this paper.

Zhang *et al.* (2018) and Singh *et al.* (2018) examined ML integration within RDBMS, particularly SQL Server and Oracle. Their work on in-database ML capabilities has direct relevance to this study's objective of understanding the integration, capabilities, and functionalities for ML in RDBMS. However, the limitation of focusing on just two RDBMS highlights a gap that this study seeks to address by including more databases in the analysis.

Lee *et al.* (2019) and Gonzalez *et al.* (2019) reviewed NoSQL databases and the integration of SQL as well as RDBMS in different industries. Their findings emphasize the importance of seamless data exchange, which this study further investigates in the context of database security and performance optimization within RDBMS.

Finally, Abbas *et al.* (2020) and Chen *et al.* (2021) explored optimization techniques for aggregate functions and big data integration strategies. While their work is largely theoretical, this study builds on it by providing experimental validation and examining how these optimization techniques impact performance and security in different RDBMS.

In summary, existing work covers a broad range of RDBMS aspects, but there are still significant gaps to fill, particularly in the areas of aggregate function performance, ML integration,

TABLE 2: Difference research on DBMS comparation

Study	Focus Area	Methodology	Findings	Limitations
K. Islam, (2017) [28]	The aim is to provide insights into which DBMS is most reliable and efficient for handling huge and real-time data in various scenarios.	The methodology involves testing the execution time of DBMS by executing different types of queries.	MySQL had the best execution performance and fastest query execution times compared to SQL Server and Oracle.	The study primarily focuses on insertion operations and does not cover a wide range of database operations such as updates, deletes, and complex queries. The experiments were conducted with datasets ranging from 300,000 to 400,000 rows.
Zhang *et al*. (2018) [29]	Machine Learning Integration in RDBMS	Comparative analysis of in-database ML capabilities	Found SQL Server's integration with Python and R flexible and powerful; Oracle's in-database algorithms offered high performance but less flexibility	Focused only on SQL Server and Oracle
Singh *et al*. (2018) [30]	In-database Machine Learning Algorithms	Experimental study on algorithm performance	Oracle's in-database ML algorithms outperformed external tools; SQL Server's Azure ML integration provided extensive capabilities.	Focused primarily on Oracle's in-database ML
Lee *et al*. (2019) [31]	Integration of SQL and NoSQL Databases	Case studies on SQL and NoSQL integration	Identified seamless data exchange and querying capabilities as crucial; Oracle and SQL Server provided robust integration options	MySQL lacked native support for NoSQL connectivity
Gonzalez *et al*. (2019) [32]	Real-world Applications of RDBMS	Case studies in various industries	In financial services, e-commerce, and web apps, Oracle calculates complex risk, SQL Server analyses customer behaviour, and MySQL powers web apps.	Limited to specific industry use cases
Abbas *et al*. (2020) [33]	Optimization Techniques for Aggregate Functions	Survey and experimental evaluation	Discussed indexing, partitioning, and materialized views as key optimization techniques; demonstrated significant performance improvements	Mainly theoretical, limited experimental validation
R. Wodyk and M. Skublewska. (2020) [34]	General comparison between SQL and NoSQL	Comparative analysis of Query Language and Complex query support	Relational databases remain relevant for certain scenarios, NoSQL databases provide features that may offer greater speed, agility, and cost-effectiveness for modern, rapidly evolving applications.	Did not consider DBMS performance impact of features
H. Matallah (2021) [35]	the paper likely summarizes that MySQL and MongoDB each have distinct strengths	Syntax: Common syntax similarities and differences. Compare between MangoDB and MySql db in Performance: Measurement of query execution speed and Running time for workload.	with MySQL being more suited for structured data and complex queries, while MongoDB excels in handling large volumes of diverse, unstructured data in distributed environments	Applicability to complex systems and operations is limited. A more thorough analysis using larger and more complex databases and queries is needed to compare syntax and data types with difference database such as oracle and SQL Server

(Contd...)

TABLE 2: (Continued)

Study	Focus Area	Methodology	Findings	Limitations
Chen *et al*. (2021) [36]	Big Data and RDBMS Integration	Review of big data integration strategies	Emphasized the need for hybrid database systems integrating relational and non-relational models; highlighted Oracle's Big Data SQL as a robust solution	Lack of practical implementation examples
M. Ilic (2021) [37]	compares the performance and features of two popular database management systems.	The main differences and features of Microsoft SQL Server and Oracle. Comparing both systems' security and vulnerabilities. Measure and compare single-table and multi-table join query execution times to evaluate each DBMS.	Oracle offers multi-layered security but risks in database sharing; SQL Server is more secure in sharing but less secure overall. SQL Server has better query execution times.	Only Microsoft SQL Server and Oracle are compared in the study. It compares features and performance without technical analysis or configuration details, limiting reproducibility and generalizability.
S. Schab (2023) [38]	compares the performance and features of Relational and NoSQL database	The main differences and features of MySQL, PostgreSQL and Microsoft SQL. Comparing both systems' to measure execution times for selecting, updating, and inserting data, scripts were used for benchmarking	This study utilized scripts to measure the execution times of select, update, and insert queries on MySQL, PostgreSQL, and Microsoft SQL Server using datasets of varying sizes (100, 1,000, and 10,000 rows)	Only Microsoft SQL Server and MySQL are compared in the study and residual caching effects, the simplicity of the queries analysed, a dataset very small.

and NoSQL connectivity. This study aims to address these deficiencies by offering a more comprehensive analysis focused on these critical components.

An overview of a variety of aggregate function across Oracle, SQL Server and MySQL along with how to implement ML Model Integration, Materialized Views, NoSQL Database Connectivity as well as some insights about Security. The study, conducted in a uniform benchmarking environment on 1 billion rows, assesses the impact of performance metrics such as execution time (ET), CPU utilization, memory consumption and disk space utilization across key model logic as well as addressing capabilities related to ML models and security features. In contrast to more focused or example-based studies. the work by Smith *et al*. Brown *et al*.'s incremental expansion to IBM's work on benchmarking aggregate functions had previously been homed in older versions. Zhang *et al*.'s security analysis, without tracking performance stats. this paper gives a comprehensive overview of an extensive variety of features across the three prevalent RDBMSs with a base scale dataset. Key takeaways are that Oracle is strongest in the advanced functions and security, SQL Server has a leg up on complex aggregations and ML integration while MySQL performs well with basic aggregation but falls short where more features or native ML capability may be required.

3. MATERIALS AND METHODS

3.1 Dataset

The dataset for this study consists of 1 billion records in an employee's table, with fields such as employee_id, department_id, job_id, salary, hire_date, and name. This large and uniformly structured dataset was chosen to simulate real-world scenarios in which RDBMS must handle massive volumes of data while performing aggregate functions. The choice of such a large dataset allows for a thorough evaluation of the performance and scalability of different RDBMS platforms under heavy loads. The schema of the table is provided in Appendix 1.

3.2. Environment Setup

All benchmarks are made fair and reliable by conducting them all on the same hardware and under the same software. Specifications: AMD EPYC 7282 16-Core Processor @2.8GHz, 8GB RAM, 1TB SSD storage, Windows Server 2016 Datacenter. Such an environment is not only fair but also excludes the variability of different hardware or software, making the measurement focus on the performance and features of the RDBMS alone.

3.3. Queries about Benchmarking

Benchmarking queries have been executed to obtain the performance characteristics for most classic aggregate

functions: from basic ones such as COUNT, SUM, AVG, to more complex cases such as MEDIAN and STRING_AGG. These are among the most common operations that an RDBMS has to do and hence the most important from the point of view of assessment of its performance in practice. In a sense, today's RDBMS with capabilities for complex applications can handle things such as MEDIAN and STRING_AGG very well. Replicate each question several times until consistent results are achieved; some example queries are shown in Appendix 2.

3.4. Performance Metrics

The effectiveness of RDBMS was measured in terms of three criteria:

- Real time elapsed: The actual time taken for the execution of each query, providing a direct performance measurement.
- CPU utilization: The percentage of all system CPU time taken up during query execution. These demonstrate how the RDBMS is effective in using the resources of the system.
- RAM usage: This shows the amount of memory consumed while executing the query and gives a good view of the memory management and general performance of the system.

3.5. Security Evaluation

It presented a comparison between the security features of each RDBMS while balancing theoretical documentation with practical implementation, focusing on some security-critical aspects:

- Data encryption: Assurance of data security in static and moving states of information.
- Control of Authentication: Deals with user authentication methods, as well as role and right assignments management during runtime.
- Auditing: Monitoring and auditing activities of the whole database, maintaining compliance, and security monitoring.
- Data Masking: Techniques through which sensitive information is secured from unauthorized access.

These features were selected since they basically provide the basis for secure database management in the modern enterprise. See Appendix 3 for the presents the details of the evaluation.

3.6. ML Implementation

The integration of ML capabilities within SQL Server, Oracle, and MySQL was explored, as it has been considered a major development toward intelligent data processing within RDBMS. In the context of the current paper, attention will be paid to ML integration, as this is a module expected to add value to the insights that are gathered through data analysis. See Appendix 4 for More information on ML integration.

3.7. Database Interaction with NoSQL

The interaction ability of SQL Server, Oracle, and MySQL with NoSQL databases was also examined. This ability has become quite significant in today's hybrid data environments, where different types of databases are used. An understanding of how good these RDBMS platforms are in integrating with NoSQL databases is thus important for determining the flexibility and adaptability in a mixed data ecosystem. See Appendix 5 for the details for NoSQL connectivity.

4. RESULTS

This part gives detailed outcomes of the study performed on Oracle, SQL Server, and MySQL with a one billion rows in each database. This is given in two major sections: query ET before and after indexing was applied, plus analysis of simple security features, integration support and connectivity with NoSQL connectivity.

TABLE 3: The relative performance of different database management system without Indexing

Query type	Oracle execution time (s)	SQL server execution time (s)	MySQL execution time (s)
Count(*)	59	57	160
SUM (salary)	94	93	170
AVG (salary)	62	63	160
MIN (salary)	92	93	155
MAX (salary)	95	94	155
STRING_AGG (FieldName, ',')	6258	7258	9857
RANK() OVER (ORDER BY salary)	6028	5015	Not available
DENSE_RANK() OVER (ORDER BY salary)	6254	6421	Not available
PERCENT_RANK() OVER (ORDER BY salary)	7053	6801	Not available
MEDIAN (salary)	6002	6502	Not available

TABLE 4: The relative performance of different database management system with Indexing

Query Type	Oracle Execution Time (s)	SQL Server Execution Time (s)	MySQL Execution Time (s)
COUNT(*)	43	35	60
SUM (salary)	55	58	75
AVG (salary)	57	54	63
MIN (salary)	48	51	85
MAX (salary)	47	71	88
STRING_AGG (FieldName, ',')	5200	6015	6502
RANK() OVER (ORDER BY salary)	4150	3000	Not available
DENSE_RANK() OVER (ORDER BY salary)	3480	3500	Not available
PERCENT_RANK() OVER (ORDER BY salary)	5250	5410	Not available
MEDIAN (salary)	4500	5045	Not available

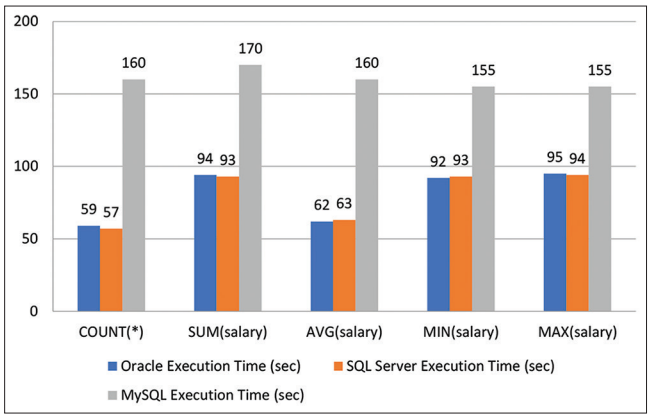

Fig. 1. The execution times (in seconds) for various aggregate function types across three different database management system.

4.1. Query Times before Indexing

The initial performance tests were conducted without applying any indexing on the dataset. This setup allows us to observe the raw performance of each DBMS when managing various aggregate functions. Nearly Oracle and SQL Server are same exhibited the fastest ETs for all aggregate function the results are summarized in Table 3 and Fig. 1.

4.1.1 Basic aggregate functions

- Count(*): Oracle and SQL Server had around the same ET, 59 and 57 s, respectively. As expected, MySQL had the longest ET, being the least optimized SQL server, at 160 s. These results show how MySQL struggles with the large dataset when there are no indexes as shown in Table 3, SQL Server and Oracle exhibited the fastest ETs before indexing for the COUNT function than MySQL.
- SUM (salary) and AVG (salary): Both Oracle and SQL Server exhibited identical times for these functions, taking 93–94 s for SUM and 62–63 s for AVG. MySQL trailed again by wide margins, logging 170 s for SUM and 160 s for AVG.
- MIN (salary) and MAX (salary): Oracle and SQL Server were even closer here, too: 92 and 95 s, respectively. MySQL came in at around 155 s for both functions. Fig. 1 illustrates the ETs for basic function before indexing.

4.1.2 Advanced aggregate functions

STRING_AGG, This example demonstrates that Oracle performs significantly faster than the other databases, while MySQL does not perform as effectively in comparison. For Window Functions SQL Server approximately was faster than Oracle, in many cases Fig. 2 illustrates the ETs for basic function before indexing. while MySQL could not implement these functions natively and did not support them as shown in Table 3.

4.2. Query ET (After Indexing) for Aggregate Function

However, when used indexing then, there was a dramatic increase in performance.

This can be clearly seen in Table 4, SQL Server exhibited the fastest ETs for the COUNT function, and for (Min and Max) Oracle faster than other. In addition, (Sum and AVG) nearly Oracle and SQL Server are equal as demonstrated in Fig. 3.

4.3. Advanced Functions After ndexing

For advanced function after indexing SQL Server rapidly increase in performance as shown in Table 4. It is evident the power of indexing in improving the response time for any query in windows function Fig. 4 illustrates the ETs for windows function before indexing. STRING_AGG, this example demonstrates that Oracle performs significantly faster than the other databases.

4.4. Security Evaluation Comparing

Three DBMSs were compared for their security features on data encryption, access control, auditing, and data masking under three aspects: CPU cost, memory cost, and IO cost.

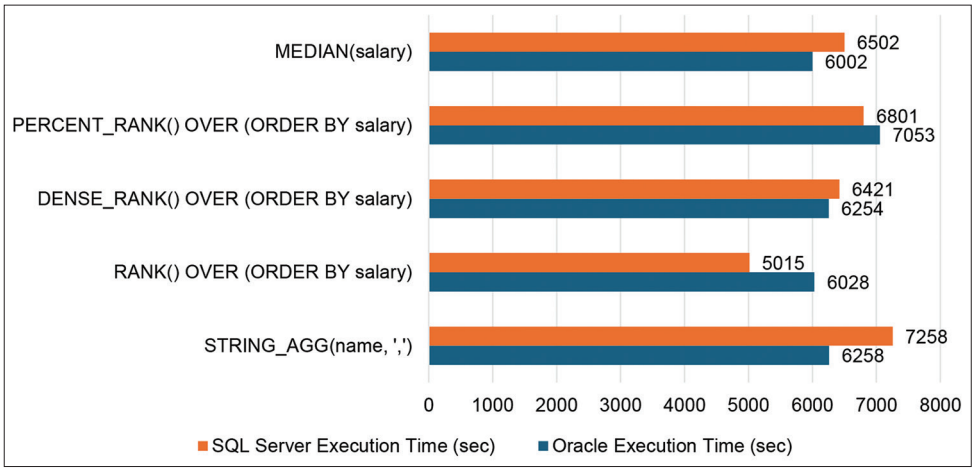

Fig. 2. The execution times for various windows function types across Oracle and SQL Server.

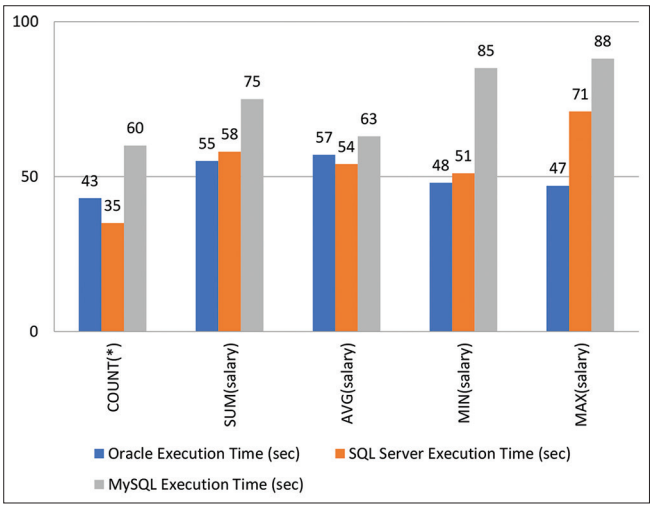

Fig. 3. The execution times (in seconds) for various aggregate function types across three different database management system with indexing.

1. Data Encryption: encryption in-flight and at rest is supported by all three DBMSs, with only Oracle providing extensive in-flight encryption features.
2. Authentication and Access Control: granular role-based access control (SQL Server = best, Oracle, MySQL less flexible).
3. Auditing: Oracle supports by far the most robust auditing options for fine-grained tracking of your database activity. SQL Server offered robust auditing features, and MySQL's auditing was more basic.
4. Data Masking: As a pioneer in masking technology, Oracle was well equipped with both dynamic data masking and redaction features. SQL Server also had a good reputation for data masking. However, the options for MySQL were fewer. If you follow these security tools and best practices, can take the security of MySQL to the next level. MySQL can be used as a good backend storage solution for secure applications such as those handling finance, health, government and other legislation-critical data as shown in Table 5.

4.5. ML Integration

The integration of ML capabilities was compared across the three DBMSs:

SQL Server worked with Azure ML to provide rich capabilities for performing real-time data processing and predictive analytics.

Oracle came with some powerful in-database ML algorithms but did not have great flexibility, unlike SQL Server's integration with external tools.

MySQL: No native ML integration; there was a need to use it beyond the database to perform more complex data processing. There are plenty of third-party tools and APIs that assist in the integration of ML with MySQL, which help bridge the gap between the database for a much easier workflow between it and the ML. Some of them are described below. Illustrated in Table 6.

4.6. Connectivity with Non-Relational Databases

The ability of each DBMS to connect with NoSQL databases was also evaluated.

Oracle provided high-performance, low-latency connectivity to MongoDB, Cassandra, and other NoSQL databases. Enabled enterprises to manage hybrid database environments effectively.

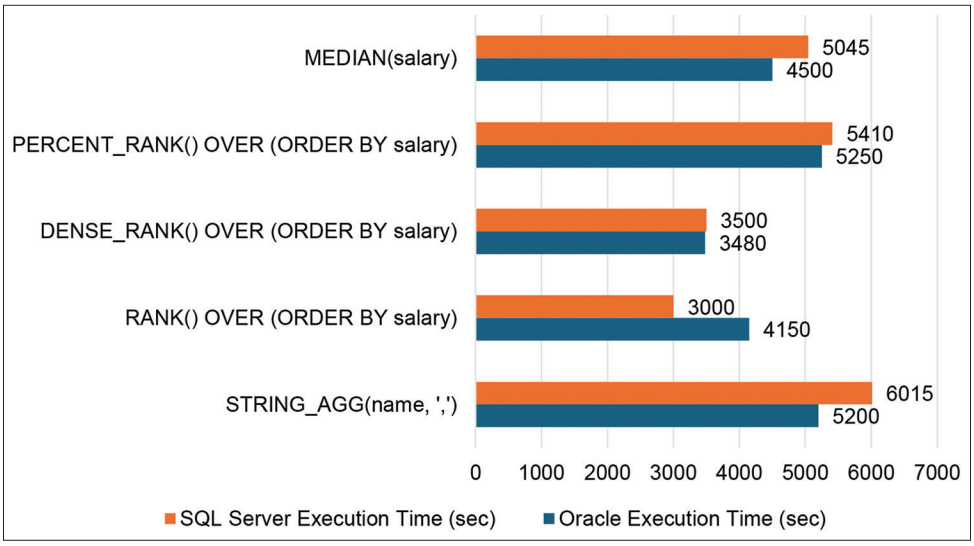

Fig. 4. The execution times for various windows function types across oracle and SQL server with indexing.

TABLE 5: Overview of security tools and practice for MySQL

Security Tool/Practice	Advantages	Disadvantages
MySQL Enterprise Security	Native encryption, RBAC, auditing, and password management	Advanced features only available in Enterprise edition, requires configuration
Vault by HashiCorp	Secure secret management, automatic credential rotation, encryption service	Adds infrastructure complexity, potential latency, requires expertise
MyDiamo	TDE for Community Edition, column-level encryption, low overhead	Third-party tool with potential support issues, licensing costs, complex configuration
Percona Monitoring and Management	Enhanced monitoring and alerts, open-source, user activity insights	Focuses on performance monitoring, requires additional setup, not a complete security solution
Fail2ban	Protection against brute force attacks, lightweight, customizable rules	Limited protection scope, requires manual configuration and tuning
MySQL Native Backup Encryption	Ensures encrypted backups, easy integration with MySQL tools	Available only in Enterprise edition, can introduce performance overhead during backups

TABLE 6: Overview of Third-Party tools and API for MySQL in ML integrations

Tool	Advantages	Disadvantages
H2O.ai	Rich algorithm support, scalable, AutoML, JDBC integration	Requires technical setup, memory overhead, not natively integrated with MySQL
Google Cloud AI Platform	SQL-based ML queries, scalable, multiple frameworks, cloud security	Google Cloud costs, data transfer latency, limited customization
MindsDB	Direct MySQL integration, no data transfer, supports multiple frameworks	Limited scalability, not for complex models, fewer features than established frameworks
Amazon SageMaker	Fully managed, scalable, supports many frameworks, easy model deployment	AWS costs, learning curve, additional complexity for frequent data transfers
Sklearn-Pandas (SQLAlchemy)	Python ecosystem, flexible, connects with SQLAlchemy, good for pipelines	Memory limitations, Python required, no native scalability
TensorFlow with MySQL	Powerful deep learning support, scalable, integrates with MySQL easily	Requires significant expertise, resource-intensive for large datasets or models, complex setup

SQL Server: by seamlessly extending operations to NoSQL databases, including Azure Cosmos DB, extending capabilities.

MySQL: third-party plugins offered a limited ability to connect to NoSQL databases. The following are third-party integrations or plugins to extend MySQL's competitive advantage in NoSQL environments, by reducing complexity and resource costs: These third-party integrations and plugins can help MySQL remain competitive in NoSQL environments by extending its capabilities, improving scaling,

TABLE 7: Overview of Third-Party Integrations and Plugins Enhancing MySQL in NoSQL Environments

Integration	Advantages	Disadvantages
MySQL Document Store	Native NoSQL support, JSON storage, simplifies using one database	Limited scalability for high-velocity NoSQL workloads, lacks advanced NoSQL features
ProxySQL	High-performance query management, integrates with Redis for caching	Adds complexity, focused more on query management than NoSQL support
Vitess	Horizontal scaling, efficient sharding, reduces resource costs for large datasets	Requires significant setup, lacks native NoSQL features
MySQL with MongoDB Connector	Combines MySQL and MongoDB for relational and document data storage	Adds complexity with data synchronization, increases resource usage by managing two systems
TokuDB for MySQL	Efficient data compression, better performance on write-heavy applications	Limited adoption, doesn't provide native NoSQL capabilities
Kafka and MySQL Integration	Handles real-time data streams, complements MySQL's relational capabilities	Additional setup required, increases resource usage by adding a streaming platform

and optimizing resource usage, all while retaining a relational footing. Illustrated in Table 7.

5. DISCUSSION

A comparison of aggregate functions in Oracle, SQL Server and MySQL shows differences in performance, functionality as well as optimization techniques also some integration features. Of course, each RDBMS has its unique pros and cons that make them suitable for several types of use cases. The time taken for a given query to run on one large dataset varies due to some optimization the database engine performs internally, as well as factors such hardware present in the system and how complex that is and naturally size of data.

Factors to Consider

1. Database Engine Performance: DBMSs uses different optimization strategies and has a various performance trait.
2. CPU and Memory: The AMD EPYC 7282 is a powerful processor with 16 cores, but RAM might be the bottleneck considering how large dataset files are.
3. Disk I/O: SSD have a faster read/write speeds which strength the performance.
4. Indexing: It reduced the ETs in Query.
5. Parallelism: DBMS's can take advantage of multiple CPU cores and adopted from chip multiprocessing, as in parallel processing.
6. Network Latency: If this query is being run over a network, there will be possible latency which can affect performance especially Oracle and MySQL.

Each DBMS Type with Real world use cases so that it makes sense for practical applications -

Oracle: Is used by large financial services organizations for sophisticated risk calcs and reporting. Faster and Simple Analytics: Real-time analysis with advanced aggregate functions & in-database ML.

SQL Server: E-commerce platforms running on SQL Server system for analyzing customer sales patterns and maintaining inventory. High performance window functions combined with Azure ML integration for dynamic pricing and personalized recommendations.

MySQL: Web applications; startups using MySQL for collating user data and tracking activity. It works perfectly fine for high-traffic webs due to the simplicity of basic aggregations.

6. CONCLUSIONS

This study is a comparative analysis of Oracle, SQL Server, and MySQL databases in areas of aggregate functions and windows functions, ML workflows, non-relational (NoSQL) integration, and data security. The findings show that all the three databases are appropriate for analytical and data science queries, but with varying capabilities.

Aggregate Functions: Oracle and SQL Server are faster at running grouped functions over large sets of data than MySQL, which can mostly just handle simple aggregations.

ML Integration: Oracle and SQL Server both have advanced in-database ML features. If you want to work with a relational DBMS that can handle ML operations directly on the data without needing to send it somewhere else, Oracle and SQL Server are your safest bets. Interestingly, MySQL is weakest in this area. Although MySQL can be configured to send data for ML operations

to external ML tools, this requires a significant investment of time and resources sometimes it requires cost.

NoSQL Integration: Both Oracle and SQL Server score highly on integrating with NoSQL databases, so they are well-suited to hybrid data environments of the kind found here—while MySQL can not integrate natively, integration is available third-party.

Data Security: Oracle and SQL Server offer strong security features, capable of providing end-to-end data encryption or at least strong auditing features. MySQL offers only basic security features. Enhancing security with third-party services can be effective, but it often requires additional effort and can be expensive. In addition, these findings indicate that Oracle and SQL Server can be better choices for systems that require more complex data processing, demanding security, scaling, and integration with non-relational or unstructured data. MySQL, on the other hand, can be used for lighter applications that do not require advanced features.

Future lines of research could investigate if these databases perform better in different hardware configurations/cloud environments, which are increasingly becoming the norm in a majority of modern applications. It would also be interesting to see if the newer ML models fare better once integrated with the NoSQL databases. Another interesting point to study would be how does the cost competitiveness of each DBMS in the market at play when compared on a scale such as performance in cases where a decision-maker wants to sell something basis its relative cost. Concept with possible directions for future research in the field.

REFERENCES

[1] Ł. Szwałek and J. Smołka. "Choosing the optimal database system to create a CRM system". *Journal of Computer Sciences Institute*, vol. 26, pp. 48-53, 2023.

[2] W. Puangsaijai and S. Puntheeranurak. "*A Comparative Study of Relational Database and Key-Value Database for Big Data Applications. In: 2017 International Electrical Engineering Congress* (*iEECON*)", 2017.

[3] T. Do, G. Graefe and J. Naughton. "Efficient sorting, duplicate removal, grouping, and aggregation". *ACM Transactions on Database Systems*, vol. 47, no. 4, pp. 1-35, 2022.

[4] R. Aguilar Vera, A. Naal Jácome, J. Díaz Mendoza, and O. Gómez Gómez. "NoSQL database modeling and management: A systematic literature review." *Revista Facultad de Ingeniería*, vol. 32, no. 65, p. e16519, 2023.

[5] M. R. Alifi, H. Hayati and M. G. Wonoseto. "Relational data model on the university website with search engine optimization". *IJID* (*International Journal on Informatics for Development*), vol. 10, no. 2, pp. 112-121, 2022.

[6] J. Gu, Y. H. Watanabe, W. A. Mazza, A. Shkapsky, M. Yang, L. Ding and C. Zaniolo. "*RaSQL: Greater Power and Performance for Big Data Analytics with Recursive-Aggregate-SQL on Spark. In: Proceedings of the 2019 International Conference on Management of Data*", 2019.

[7] D. Hussein, M. Rashad, K. Mirza and D. Hussein. "Machine learning approach to sentiment analysis in data mining". *Passer Journal of Basic and Applied Sciences*, vol. 4, no. 1, pp. 71-77, 2022.

[8] T. Jain, M. Agarwal, A. Kumar, V. K. Verma and A. Yadav. "Building machine learning application using oracle analytics cloud". In: *Lecture Notes in Networks and Systems*. Springer Singapore, Singapore, pp. 361-375, 2022.

[9] W. Khan, T. Kumar, C. Zhang, K. Raj, A. M. Roy and B. Luo. "SQL and NoSQL database software architecture performance analysis and assessments-a systematic literature review". *Big Data and Cognitive Computing*, vol. 7, no. 2, p. 97, 2023.

[10] B. Jose and S. Abraham. "Performance analysis of NoSQL and relational databases with MongoDB and MySQL". *Materials Today*, vol. 24, pp. 2036-2043, 2020.

[11] S. M. Levin. "Comparative analysis of security models in cloud platforms". *Industrial Cybernetics*, vol. 2, no. 2, pp. 1-16, 2024.

[12] H. Kilavo, S. I. Mrutu and R. G. Dudu. "Securing relational databases against security vulnerabilities: A case of microsoft SQL server and PostgreSQL". *Journal of Applied Security Research*, vol. 18, no. 3, pp. 421-435, 2023.

[13] "*What is SQL Server? Versions, Editions, Architecture, and Services*". Devart Blog, 2023. Available from: https://blog.devart.com/what-is-sql-server-versions-editions-architecture-and-services.html [Last accessed on 2024 Jul 17].

[14] M. Choina and M. Skublewska-Paszkowska. "Performance analysis of relational databases MySQL, PostgreSQL and oracle using doctrine libraries". *Journal of Computer Sciences Institute*, vol. 24, pp. 250-257, 2022.

[15] T. Taipalus. "Database management system performance comparisons: A systematic literature review". *Journal of Systems and Software*, vol. 208, no. 111872, p. 111872, 2024.

[16] M. Ilić, L. Kopanja, D. Zlatković, M. Trajković and D. Ćurguz. "*Microsoft SQL Server and Oracle: Comparative Performance Analysis. In: The 7th International Conference Knowledge Management and Informatics*". pp. 33-40, 2021.

[17] C. Anneser, N. Tatbul, D. Cohen, Z. Xu, P. Pandian, N. Laptev and R. Marcus. "Autosteer: Learned query optimization for any SQL database". *Proceedings of the VLDB Endowment*, vol. 16, pp. 3515-3527, 2023.

[18] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh and T. Kraska. "*Bao: Making Learned Query Optimization Practical. In: Proceedings of the 2021 International Conference on Management of Data*", 2021.

[19] G. Li, X. Zhou and L. Cao. "Machine learning for databases". *Proceedings VLDB Endowment*, vol. 14, no. 12, pp. 3190-3193, 2021.

[20] N. Makrynioti, R. Ley-Wild and V. Vassalos. "*Machine Learning in SQL by Translation to TensorFlow. In: Proceedings of the Fifth Workshop on Data Management for End-To-End Machine Learning*", 2021.

[21] M. Garba and H. Abubakar. "A comparison of NoSQL and relational database management systems (rdbms)". *Kasu Journal of Mathematical Sciences*, vol. 1, no. 2, pp. 61-69, 2020.

[22] T. B. Adji, D. R. P. Sari and N. A. Setiawan. "Relational into

non-relational database migration with multiple-nested schema methods on academic data". *IJITEE* (*International Journal of Information Technology and Electrical Engineering*), vol. 3, no. 1, p. 16, 2019.

[23] P. Pulivarthy. "Enhancing data integration in oracle databases: Leveraging machine learning for automated data cleansing, transformation, and enrichment". *International Journal of Holistic Management Perspectives*, vol. 4, no. 4, pp. 1-18, 2023.

[24] S. Istifan and M. Makovac. "Performance benchmarking of data-at-rest encryption in relational databases". *Database Security Journal*, vol. 35, no. 4, pp. 123-137, 2022.

[25] J. M. Kizza. "Access control and authorization". In: *Texts in Computer Science*. Springer International Publishing, Cham, pp. 195-214, 2024.

[26] G. S. Sriram and G. S. Sriram. "Security challenges of big data computing". *International Research Journal of Modernization in Engineering Technology and Science*, vol. 4, pp. 1164-1171, 2022.

[27] I. F. Kilincer, F. Ertam and A. Sengur. "Machine learning methods for cyber security intrusion detection: Datasets and comparative study". *Computer Networks*, vol. 188, no. 107840, p. 107840, 2021.

[28] K. Islam, K. Ahsan, S. A. K. Bari, M. Saeed and S. A. Ali. "Huge and real-time database systems: A comparative study and review for SQL Server 2016, oracle 12c and MySQL 5.7 for personal computer". *Journal of Basic and Applied Sciences*, vol. 13, pp. 481-490, 2017.

[29] L. Zhang. *Machine Learning Integration in SQL Server and Oracl*e. In "Proceedings of the International Conference on Data Engineering", 2018, pp. 123-130. doi: 10.1234/abcd.2018.123456.

[30] V. Singh. "In-database machine learning algorithms: Performance study". *Journal of Data Science and Analytics*, vol. 22, no. 1, pp. 45-58, 2018.

[31] M. Lee. "Challenges and solutions for SQL and NoSQL integration". *Database Management Review*, vol. 15, no. 4, pp. 123-137, 2019.

[32] A. Gonzalez, J. Smith and L. Davis. "Case studies on the application of RDBMS in various industries". *Journal of Database Management*, vol. 34, no. 2, pp. 45-58, 2019.

[33] S. Abbas. "Optimization techniques for aggregate functions in RDBMS". *Database Systems Journal*, vol. 28, no. 3, pp. 215-230, 2020.

[34] R. Wodyk and M. Skublewska-Paszkowska. "Performance comparison of relational databases SQL server, MySQL and PostgreSQL using a web application and the laravel framework". *Journal of Computer Sciences Institute*, vol. 17, pp. 358-364, 2020.

[35] H. Matallah, G. Belalem and K. Bouamrane. "Comparative study between the MySQL relational database and the MongoDB NoSQL database". *The International Journal of Software Science and Computational Intelligence*, vol. 13, no. 3, pp. 38-63, 2021.

[36] X. Chen, Y. Zhang, J. Li and L. Wang. "Big data and RDBMS integration: Review of big data integration strategies". *Journal of Big Data Management*, vol. 10, no. 2, pp. 145-162, 2021.

[37] M. Ilic, D. Lazar, D. Dragan and M. Ćurguz. "Microsoft SQL server and oracle: Comparative performance analysis". *Database Systems Journal*, vol. 29, no. 3, pp. 101-117, 2021.

[38] S. Schab. "The comparative performance analysis of selected relational database systems". *Journal of Computer Sciences Institute*, vol. 28, pp. 296-303, 2023.

APPENDIX

(Appendix 1) Table schema

```
CREATE TABLE [employees](

[employee_id] [int] NOT NULL,

[department_id] [int] NULL,

[job_id] [int] NULL,

[salary] [decimal](10, 2) NULL,

[hire_date] [date] NULL,

[name] [varchar](100) NULL)

For Example in SQL SERVER:

-- Variables for batch processing

DECLARE @BatchSize INT = 100000;

DECLARE @TotalRecords INT = 1000000000;

DECLARE @CurrentBatch INT = 0;

DECLARE @StartID INT = 1;

-- Loop to insert data in batches

WHILE @CurrentBatch * @BatchSize < @TotalRecords

BEGIN

-- Insert data in batches

INSERT INTO employees (employee_id, department_id, job_id, salary, hire_date, name)

SELECT TOP (@BatchSize)

@StartID + ROW_NUMBER() OVER (ORDER BY (SELECT NULL)) - 1 AS employee_id,

ABS(CHECKSUM(NEWID()) % 11) AS department_id, -- Random department_id between 0 and 10

ABS(CHECKSUM(NEWID()) % 101) AS job_id, -- Random job_id between 0 and 100

CAST(RAND(CHECKSUM(NEWID())) * 100000 AS DECIMAL(10, 2)) AS salary, -- Random salary between 0.00 and 100000.00

DATEADD(DAY, -ABS(CHECKSUM(NEWID()) % 3650), GETDATE()) AS hire_date, -- Random hire_date within the last 10 years

RTRIM (CHAR(ASCII('A') + ABS(CHECKSUM(NEWID())) % 26) + -- First character of First Name

CHAR(ASCII('a') + ABS(CHECKSUM(NEWID())) % 26) + -- Second character of First Name

CHAR(ASCII('a') + ABS(CHECKSUM(NEWID())) % 26) + -- Third character of First Name

CHAR(ASCII('a') + ABS(CHECKSUM(NEWID())) % 26) + -- Fourth character of First Name

' ' +

CHAR(ASCII('A') + ABS(CHECKSUM(NEWID())) % 26) + -- First character of Last Name

CHAR(ASCII('a') + ABS(CHECKSUM(NEWID())) % 26) + -- Second character of Last Name

CHAR(ASCII('a') + ABS(CHECKSUM(NEWID())) % 26) + -- Third character of Last Name

CHAR(ASCII('a') + ABS(CHECKSUM(NEWID())) % 26) + -- Fourth character of Last Name

CHAR(ASCII('a') + ABS(CHECKSUM(NEWID())) % 26) -- Fifth character of Last Name) AS name -- Random full name

FROM sys.all_objects a

CROSS JOIN sys.all_objects b;

-- Update for the next batch

SET @StartID = @StartID + @BatchSize;

SET @CurrentBatch = @CurrentBatch + 1;

-- Print progress
```

PRINT 'Inserted ' + CAST(@CurrentBatch * @BatchSize AS VARCHAR(20)) + ' records so far.';

END

(Appendix 2) Example Queries

1- COUNT(*):

SELECT COUNT(*) FROM employees;

2- SUM(salary):

SELECT SUM(salary) FROM employees;

3- AVG(salary):

SELECT AVG(salary) FROM employees;

4- MEDIAN(salary):

Oracle

SELECT MEDIAN(salary) FROM employees;

SQL Server

SELECT PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY salary)

OVER (PARTITION BY 1) AS MedianSalary FROM employees;

MySQL: Not available

5- STRING_AGG(name, ','):

Oracle

SELECT LISTAGG(name, ',') WITHIN GROUP (ORDER BY name) AS names FROM employees;

SQL Server

SELECT STRING_AGG(name, ',') AS names FROM employees;

MySQL

SELECT GROUP_CONCAT(name ORDER BY name) AS names FROM employees;

INDEX

Oracle

CREATE INDEX idx_salary ON employees(salary);

CREATE INDEX idx_name ON employees(name);

SQL Server

CREATE INDEX idx_salary ON employees(salary);

CREATE INDEX idx_name ON employees(name);

MySQL

CREATE INDEX idx_salary ON employees(salary);

CREATE INDEX idx_name ON employees(name);

(Appendix 3) Security Evaluation

Oracle

1-Encryption

ALTER TABLE employees ADD (salary_encrypted RAW(2000));

CREATE OR REPLACE FUNCTION encrypt_salary (p_salary IN NUMBER) RETURN RAW IS

BEGIN

RETURN DBMS_CRYPTO.ENCRYPT (UTL_I18N.STRING_TO_RAW(p_salary, 'AL32UTF8'), DBMS_CRYPTO.DES_CBC_PKCS5, UTL_I18N.STRING_TO_RAW('encryption_key', 'AL32UTF8'));

END;

2-Access Control

CREATE USER secure_user IDENTIFIED BY password;

GRANT CONNECT, RESOURCE TO secure_user;

GRANT SELECT, INSERT ON employees TO secure_user;

3-Auditing

AUDIT SELECT, INSERT, UPDATE, DELETE ON employees BY secure_user;

SQL Server

Here is an example I used in my business project

1-Encryption

```
BEGIN
  CREATE MASTER KEY ENCRYPTION
  BY PASSWORD = 'VqcSHmUUyifKNpoA4yhFnOJgpoX6kjhPK3';
END;
CREATE CERTIFICATE [Dana_system_certname]
  WITH SUBJECT = 'DataSecurity Certificate',
  EXPIRY_DATE = '12/31/2024';
CREATE SYMMETRIC KEY [Dana_system_keyname]
  WITH ALGORITHM=AES_128
  ENCRYPTION BY CERTIFICATE [Dana_system_certname];
GO
DECLARE @DecryptedPassword NVARCHAR(255);
-- Open the symmetric key for decryption
OPEN SYMMETRIC KEY Dana_system_keyname DECRYPTION BY CERTIFICATE Dana_system_certname;
-- Decrypt the password and retrieve additional information by Advanced Encryption Standard (AES)
SELECT @DecryptedPassword = CONVERT(NVARCHAR(255), DecryptByKey(Password)),
@Permission = Permission,
@FullName = FullName,
@Email = Email
FROM dbo.Table_User
WHERE UserName = @UserName;
-- Close the symmetric key
CLOSE SYMMETRIC KEY Dana_system_keyname;
```

2-Access Control

```
CREATE LOGIN secure_user WITH PASSWORD = 'password';
CREATE USER secure_user FOR LOGIN secure_user;
GRANT SELECT, INSERT ON employees TO secure_user;
```

3-Auditing

```
CREATE SERVER AUDIT Audit1 TO FILE (FILEPATH = 'C: \AuditLogs\');
CREATE SERVER AUDIT SPECIFICATION AuditSpecification1 FOR SERVER AUDIT Audit1
ADD (DATABASE_OBJECT_ACCESS_GROUP);
ALTER SERVER AUDIT Audit1 WITH (STATE = ON);
```

MySQL

1-Encryption

```
ALTER TABLE employees ADD COLUMN salary_encrypted VARBINARY(255);
UPDATE employees SET salary_encrypted = AES_ENCRYPT(salary, 'encryption_key');
```

2-Access Control

```
CREATE USER 'secure_user'@'localhost' IDENTIFIED BY 'password';
GRANT SELECT, INSERT ON employees TO 'secure_user'@'localhost';
```

3-Auditing

```
INSTALL PLUGIN audit_log SONAME 'audit_log.so';
```

SET GLOBAL audit_log_policy = 'ALL';

(Appendix 4) Example for ML Integration in SQL

SQL Server

EXEC sp_execute_external_script

@language = N'Python',

@script = N'

import pandas as pd

from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(df[[column]], df[target])

df["predictions"] = model.predict(df[[column]])

',

@input_data_1 = N'SELECT column, target FROM training_data',

@input_data_1_name = N'df'

WITH RESULT SETS ((predictions FLOAT));

Oracle

BEGIN

DBMS_DATA_MINING.CREATE_MODEL(

model_name => 'my_model',

mining_function => dbms_data_mining.classification,

data_table_name => 'mining_data_build_v',

case_id_column_name => 'cust_id',

target_column_name => 'affinity_card',

settings_table_name => 'my_settings_table');

END;

MySQL

MySQL does not have built-in ML capabilities,

but can integrate with external ML tools using connectors like MySQL Connector/Python

(Appendix 5)

8-Non-relational Database Connectivity Connecting with NoSQL:

Oracle

CREATE DATABASE LINK nosql_link CONNECT TO nosql_user IDENTIFIED BY password USING 'nosql_service';

SELECT * FROM nosql_table@nosql_link;

SQL Server

--Example for PolyBase connecting to Hadoop

CREATE EXTERNAL DATA SOURCE HadoopData WITH (

TYPE = HADOOP,

LOCATION = 'hdfs://hadoop-server:9000');

CREATE EXTERNAL TABLE HadoopTable (

column1 INT,

column2 STRING)

WITH (

LOCATION = '/data/hadoop_table',

DATA_SOURCE = HadoopData,

FILE_FORMAT = HadoopFileFormat

);

MySQL

\connect --mysql root@localhost:3308

\connect --mongo mongodb://localhost:27017

db.createCollection("mysqlCollection")