A REPORT ON

Emerging Trend in Database

Submitted as a partial fulfilment of requirement of the curriculum of

Bachelor's 3rd Semester (Software Engineering)

Submitted By:

Rasad Regmi

Pranav Subedi

Srijan Paudel

Aryan Jung Thapa

Under Supervision Of

Amit Shrivastav

DATE:

28-01-2024

Nepal College Of Information Technology

Balkumari, Lalitpur

Nepal

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my dedicated team members, Pranav Subedi, Srijan Paudel, and Aryan Jung Thapa, for their invaluable contributions to the preparation of this report on "Emerging Trend in Database". Our collaborative effort has been the cornerstone of the success of this project.

I extend my heartfelt appreciation to each team member for their diligence and teamwork, which have undoubtedly elevated the quality of our work. This report stands as a testament to the collective effort, dedication, and collaborative spirit of our team. I am truly grateful for the opportunity to work with such a talented and committed group of individuals.

Sincerely,

Rasad Regmi

TABLE OF CONTENT

Assignment	Page Number
NoSQL Databases	1 - 4
Object Oriented Database and ORM	5 - 7
Distributed Databases	8 - 10
Distributed Ledged Technology	11 - 14
Conclusion	15

NoSQL Database

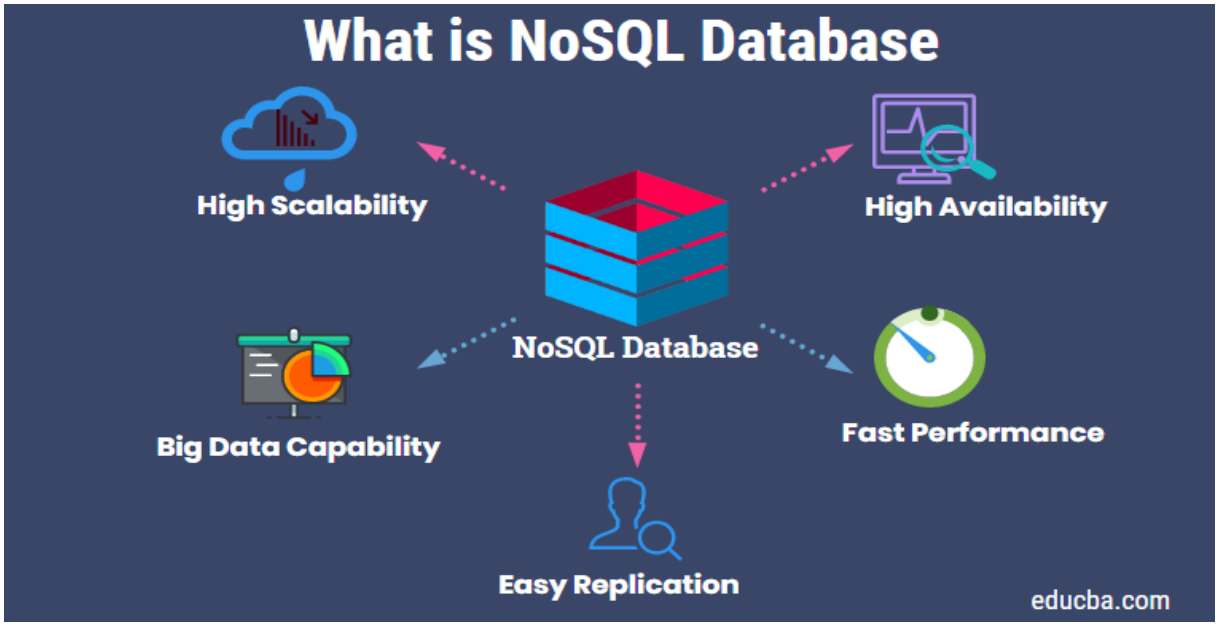
Introduction

NoSQL databases emerged in the late 2000s as the cost of storage dramatically decreased. Gone were the days of needing to create a complex, difficult-to-manage data model in order to avoid data duplication. Modern applications face several challenges that can be solved by NoSQL databases. For instance, applications process a large data volume from disparate sources like social media, smart sensors, and third-party databases.

NoSQL databases are non-tabular databases and store data differently than relational tables. NoSQL databases come in a variety of types based on their data model.

NoSQL is an approach to database management that can accommodate a wide variety of data models, including key-value, document, column and graph formats. A NoSQL database generally means that it is non-relational, distributed, flexible and scalable.

NoSQL occasionally goes easy on consistency in exchange for higher speed, partition tolerance, and availability. This flexibility of NoSQL data structures allows these databases to solve various business problems.



Example

- Document: MongoDB and CouchDB
- Key-value: Redis and DynamoDB
- Wide-column: Cassandra and HBase
- Graph: Neo4j and Amazon Neptune.

Features of NoSQL Database

Each NoSQL database has its own unique features. At a high level, many NoSQL databases have the following features:

- Flexible schemas: NoSQL databases are highly flexible as they can store and combine any type of data.
- Horizontal scaling
- Fast queries due to the data model
- Ease of use for developers
- High availability
- Distributed architecture

Types of NoSQL Database

Four major types of NoSQL databases:

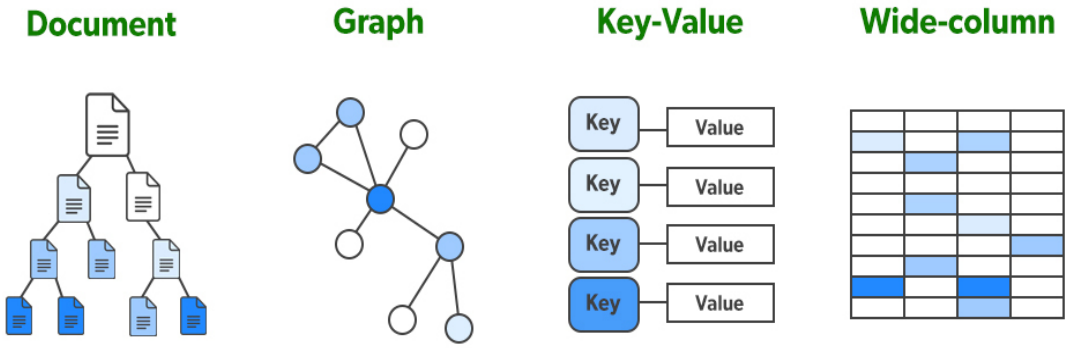
- **Document Databases**
  Document databases store data in documents similar to JSON (JavaScript Object Notation) objects. Each document contains pairs of fields and values. The values can typically be a variety of types including things like strings, numbers, Booleans, arrays, or objects. Document NoSQL databases store data such as JSON objects. Users need not define the data, thus enhancing flexibility.

- **Key-Value Databases**
  Key-value databases are a simpler type of database where each item contains keys and values. Data storage takes place in the form of a hash table with unique keys. Values can be stored in several formats, including string Key-value NoSQL is conceptually based on hash tables using a unique key and a pointer to a specific data item. Key-value storage can include entirely new sets of key-value pairing captured as objects. However, despite featuring the most straight forward implementation process, it is inefficient for querying or updating partial values.

- **Wide-Column Stores**
  Wide-column stores store data in tables, rows, and dynamic columns, with each column being treated independently. Due to ready data availability in the column form, this database type is preferred for high-performance delivery in cases involving aggregation queries such as COUNT, SUM, MIN, and AVG.

- **Graph Databases**

  Graph databases store data in nodes and edges. Nodes typically store information about people, places, and things, while edges store information about the relationships between the nodes. This database model is not held back by the disadvantages associated with the row-and-column tables and inflexibly structured SQL. Instead, it uses a flexible graph model that one can scale across numerous systems.



Advantages of NoSQL

- Scalability: NoSQL databases can scale to accommodate any type of data growth while maintaining low cost.
- Simplicity:
- Less code
- Easy maintenance
- Efficient
- Flexible: NoSQL databases are highly flexible as they can store and combine any type of data.
- High Performance: NoSQL databases are built for great performance.
- Ease of development
- Cost effectiveness

Object Oriented Object and ORM

Introduction

One of the challenges of using object-oriented programming (OOP) languages and databases is the complexity of aligning the programming code with database structures. **Object-relational mapping (ORM)** is a technique that creates a layer between the language and the database, helping programmers work with data without the OOP paradigm.

The challenge OOP developers have is the need to understand and code in the structured query language (SQL) in order to connect their application to an SQL database. Developers who know SQL can write data-access code. This raw SQL coding can be extremely time-consuming because it requires the developer to extract the code strings' data elements. SQL query builders add a layer of abstraction to the SQL code to provide more information about the data. However, developers still need to understand and write SQL.

ORMs are popular and controversial at the same time. Advocates of ORMs claim they increase productivity, improve application design, reuse code and maintain the application over time. According to detractors, a negative aspect of ORMs is performance. This article will provide an overview of ORMs, compare them against SQL tools, and review the pros and cons of these tools so that you can decide if ORMs will help or hurt your database application development efforts.

What is an ORM?

An object-relational mapper provides an object-oriented layer between relational databases and object-oriented programming languages without having to write SQL queries. It standardizes interfaces reducing boilerplate and speeding development time.

Object-oriented programming includes many states and codes in a format that is complex to understand and interpret. ORMs translate this data and create a structured map to help developers understand the underlying database structure. The mapping explains how objects are related to different tables. ORMs use this information to convert data between tables and generate the SQL code for a relational database to insert, update, create and delete data in response to changes the application makes to the data object. Once written, the ORM mapping will manage the application's data needs and you will not need to write any more low-level code.

How does ORM work?

ORMs create a model of the object-oriented program with a high-level of abstraction. In other words, it makes a level of logic without the underlying details of the code. Mapping describes the relationship between an object and the data without knowing how the data is structured. The model can then be used to connect the application with the SQL code needed to manage data activities. This "plumbing" type of code does not have to be rewritten, saving the developer a tremendous amount of time.

Types of ORMs

ORMs employ two different strategies: active record pattern and data-mapper pattern.

- **Active record pattern**
  This strategy maps data within the structure of objects in the code. You manage data using classes and structures within your programming code. This method has problems since the database structure is tightly connected to the code, making it difficult to remove the database and migrate it to a different application.

- **Data – mapper pattern**
  The data-mapper pattern attempts to decouple the business logic in the objects from the database. This separation can make it easier to change databases and use the same programming logic.

ORM vs. SQL

Developers can use raw SQL code to write a direct interface between the application and the database. Most relational databases support SQL to build data interfaces and applications. It's stable, and since SQL has been used since the 1970s, it's well documented and supported. Programmers maintain a lot of control over their data interface with SQL. It requires a lot of work, but it is more flexible and detailed than an ORM abstraction.

Native Querying with SQL

Using raw SQL also has its drawbacks. For instance, the developer is responsible for the safety and security of the database code. SQL injection is a problem where user input can affect the data state causing issues with the application and data integrity. ORMs sanitizes the code, making it easier to avoid these problems.

SQL Query Builders

Query builders add a layer of abstraction over the raw SQL without masking all of the underlying details. The builders formalize querying patterns and add methods to or functions that add escape items for easier application integration. They add a templating layer to help developers understand the database structure within the same coding application. Template builders still require developers to understand the database structure, requiring them to know SQL.

Distributed Databases

Introduction

A distributed database refers to a database system in which data is stored, processed, and managed across multiple interconnected and geographically distributed computing nodes. Unlike a centralized database, where all data is stored on a single server or a closely connected group of servers, a distributed database distributes data across a network of nodes.

Examples of distributed databases include Google's Bigtable, Amazon DynamoDB, Apache Cassandra, and Hadoop Distributed File System (HDFS). These systems are commonly used in scenarios where large-scale data storage, processing, and high availability are essential requirements.

Advantages of Distributed Databases

There are many advantages to using distributed databases:

- Distributed databases are capable of modular development, meaning that systems can be expanded by adding new computers and local data to the new site and connecting them to the distributed system without interruption.
- When failures occur in centralized databases, the system comes to a complete stop. When a component fails in distributed database systems, however, the system will continue to function at reduced performance until the error is fixed.
- Admins can achieve lower communication costs for distributed database systems if the data is located close to where it is used the most. This is not possible in centralized systems.

Features of Distributed Databases

Some general features of distributed databases are:

- **Location Independency**
  Data is physically stored at multiple sites and managed by an independent DDBMS.

- **Distributed Query Processing**
  Distributed databases answer queries in a distributed environment that manages data at multiple sites. High-level queries are transformed into a query execution plan for simpler management.

- **Distributed Transaction Management**
  Provides a consistent distributed database through commit protocols, distributed concurrency control techniques, and distributed recovery methods in case of many transactions and failures.

- **Seamless Integration**
  Databases in a collection usually represent a single logical database, and they are interconnected.

- **Network Linking**
  All databases in a collection are linked by a network and communicate with each other.

- **Transaction Processing**
  Distributed databases incorporate transaction processing, which is a program including a collection of one or more database operations. Transaction processing is an atomic process that is either entirely executed or not at all.

Types of Distributed Databases

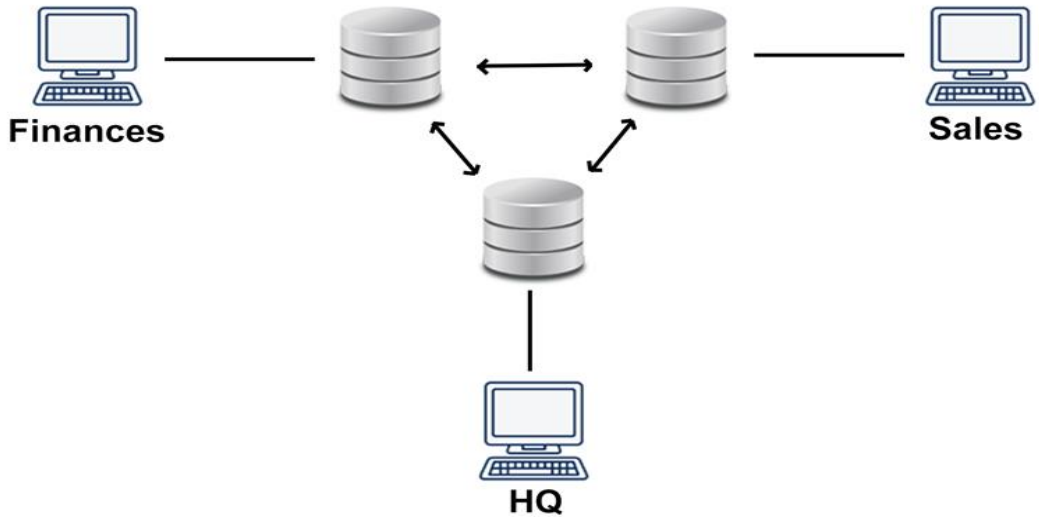
There are two types of distributed databases:

- **Homogenous**

  A homogenous distributed database is a network of identical databases stored on multiple sites. The sites have the same operating system, DDBMS, and data structure, making them easily manageable.

  Homogenous databases allow users to access data from each of the databases seamlessly.

The following diagram shows an example of a homogeneous database:

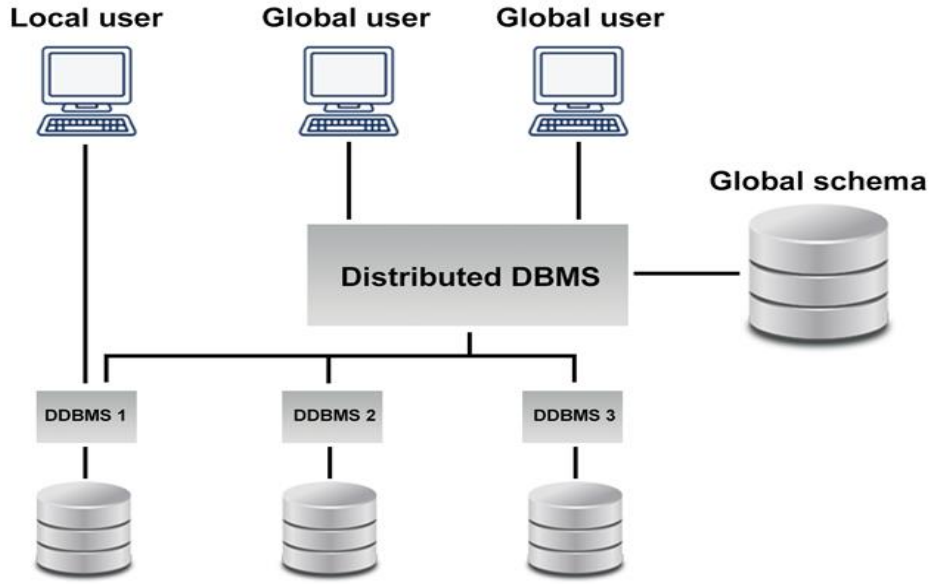


- **Heterogeneous**

  A heterogeneous distributed database uses different schemas, operating systems, DDBMS, and different data models.

  In the case of a heterogeneous distributed database, a particular site can be completely unaware of other sites causing limited cooperation in processing user requests. The limitation is why translations are required to establish communication between sites.

  The following diagram shows an example of a heterogeneous database:



Distributed Ledged Technology

Introduction

Distributed Ledger Technology (DLT) is the technological infrastructure and protocols that allow simultaneous access, validation, and record updating across a networked database. DLT allows users to view any changes and who made them, reduces the need to audit data, ensures data is reliable, and only provides access to those that need it.

Distributed ledgers are maintained by a network of nodes, each of which has a copy of the ledger, validates the information, and helps reach a consensus about its accuracy. Distributed ledgers have been around for decades but have become more well-known, researched, used, and developed since Bitcoin was introduced. All blockchains are distributed ledgers, but not all distributed ledgers are blockchains.

How Distributed Ledger Technology works?

DLTs allow information to be stored securely and accurately using cryptography. The data can be accessed using "keys" and cryptographic signatures. Once the information is stored, it can become an immutable database; the rules of the network, written into the coding of the database programming, govern the ledger.

Because they are decentralized, private, and encrypted, distributed ledgers are less prone to cybercrime, as all the copies stored across the network needs to be attacked simultaneously for the attack to be successful.

Every device on a distributed ledger network stores a copy of the ledger. These devices are called nodes—a network can have any number of nodes. Any changes to the ledger, such as moving data from one block to another, are recorded across all nodes. Because each node has a copy of the ledger, each one publishes its version with the latest transactions.

Uses of Distributed Ledger Technology

- **Record Transactions**
  DLT enables secure, transparent and decentralized transactions without the need for a central authority. As DLT is a ledger, it records inflows and outflows. Though this naturally lends itself to financial records, DLT can record any type of transaction even without financial undertones.

- **Secure Identities**
  DLT can be used to create a secure and tamper-proof digital identity for individuals, as the technology can provide a reliable way to verify identities and prevent identify theft.

- **Collect Votes**
  DLT can be used to create a secure and transparent voting system that can prevent voter fraud and ensure the integrity of the voting process. As mentioned above, as transactions (financial or non-financial) are recorded, a transparent, immutable, open ledger of interactions with users is saved. This enhances the equity and believability of a collection of opinions.

- **Enter Contracts**
  DLT allows for small contacts, agreements that automatically execute or complete based on prevailing conditions. For example, an insurance claim may automatically release funds once the claim has been processed. This limits error, and DLTs make it more difficult for precarious activity by bad actors.

- **Demonstrate Ownership**
  DLT can be used to record property transactions, creating a tamper-proof and transparent record of ownership and transfer of property. Though there are some limitations on translating real-world ownership of physical assets to a distributed ledger, the ledger may be able to convey an unchangeable source of truth regarding ownership.

Blockchain

Blockchain is a shared, immutable ledger that facilitates the process of recording transactions and tracking assets in a business network. An *asset* can be tangible (a house, car, cash, land) or intangible (intellectual property, patents, copyrights, branding). Virtually anything of value can be tracked and traded on a blockchain network, reducing risk and cutting costs for all involved.

Properties of Blockchain

- **Immutability**
  If you have stored data on the blockchain it is guaranteed that data cannot be changed later. This is very important when you want to trust something or when you want to make something more trustable. Because if I am building an application and if I want to change data because everything is in my control, I can change the data and change the data in any way I want to. But with blockchain, that is not possible.

- **Decentralized**
  It means that no single entity has control. Blockchain needs a group of nodes to be able to serve that work as points where all the data is stored in a decentralized manner. And these nodes work together to make sure that the correct provenance of data is maintained, which means that the correct timeline of data is maintained.

- **Secure:**
  All the records in the blockchain are individually encrypted. Using encryption adds another layer of security to the entire process on the blockchain network. Since there is no central authority, it does not mean that one can simply add, update or delete data on the network.Every information on the blockchain is hashed cryptographically which means that every piece of data has a unique identity on the network. All the blocks contain a unique hash of their own and the hash of the previous block.

Cryptocurrency

Cryptocurrency is a digital payment system that doesn't rely on banks to verify transactions. It’s a peer-to-peer system that can enable anyone anywhere to send and receive payments. Instead of being physical money carried around and exchanged in the real world, cryptocurrency payments exist purely as digital entries to an online database describing specific transactions. When you transfer cryptocurrency funds, the transactions are recorded in a public ledger. Cryptocurrency is stored in digital wallets.

Advantages of DLT

- Spreads systematic risk around, minimizing the risk of a single point of failure
- Has greater security due to cryptographic algorithms
- Allows for transparency and visibility into operations
- May prove to be more efficient due to smart contract automation
- Offers individuals with limited access to traditional systems potentially greater capabilities

Disadvantages of DLT

- Is more complex compared to more traditional ledger solutions
- Often requires higher energy consumption for operation
- May have difficult scaling as more users/transactions occur
- Still remains risky due to lack of regulation
- May prove to be difficult to reverse fraudulent or erroneous activity

Conclusion

In conclusion, our exploration of Emerging Trend in Database has been both enlightening and rewarding. Through thorough research and collaboration, we have gained a deeper understanding of the significance and applications of these concepts in the realm of database management.

As a team, we recognize the pivotal role Emerging Trend in Database play in enhancing data organization, accessibility, and system efficiency. The insights gained from this study have broadened our perspective on database design and management, highlighting the importance of a flexible and scalable approach.

In the ever-evolving landscape of information technology, the relevance of Emerging Trend in Database cannot be overstated. As we conclude this report, we look forward to applying our newfound knowledge in practical scenarios and contributing to the ongoing discourse in database management.