NoSql Database Modeling Techniques and Fast Search of Enterprise Data

Upendar Rao Thaduri[1*], **Karu Lal**[2]

[1]ACE Developer, iMINDS Technology Systems, Inc., Pittsburgh, PA 15243, **USA**
[2]Integration Engineer, Ohio National Financial Services, **USA**

[*]Corresponding Contact:
Email: upendarrthaduri@gmail.com

ABSTRACT

There is a need for quick databases that can deal with enormous amounts of data because of the rapid growth of the Internet and the increase in the number of websites that allow users to develop their material, such as Facebook and Twitter. To accomplish this goal, new database management systems, which will be referred to collectively as NoSQL, are currently under development. Because there are various NoSQL databases, each with unique performance, it is essential to evaluate database performance. MongoDB, Cassandra, and Couchbase are the names of the three significant NoSQL databases considered for the performance evaluation. To investigate performance, a variety of workloads were developed. The read and update operations served as the basis for the evaluation that was carried out. The results of this study provide the ability to select the NoSQL database that best meets their requirements in terms of the particular mechanisms and applications.

Key words:
NoSQL, CRUD Operations, MongoDB, Cloud Database, Schema Modeling, Modeling Technique, Enterprise Content Management

5/30/2022 **Source of Support:** None, **No Conflict of Interest:** Declared

INTRODUCTION

Databases are considered an essential component of modern companies and are used virtually every country. Using a standardized data manipulation language called SQL, relational databases make storing, extracting, and manipulating data possible (Ballamudi, 2019c). Until now, relational databases have been the ideal option for enterprise settings. On the other hand, relational databases have several drawbacks that become more apparent as the amount of data saved and analyzed continues to expand. These drawbacks include restrictions on scalability and storage, a reduction in the efficiency of querying the database due to the massive amounts of data, and an increase in the difficulty of storing and managing more extensive databases. NoSQL databases are a fresh new database model designed to overcome these restrictions (Gutlapalli, 2017c). This model was developed using a collection of recent characteristics. Nonrelational databases came about due to a technological breakthrough; these databases can be utilized independently or in addition to relational databases (Desamsetti, 2016a). In other words, relational database management systems (RDBMS) that do not employ SQL are not NoSQL. NoSQL databases are database management systems that do not use the relational paradigm and are incompatible with SQL. A database management system (DBMS) that does not employ the relational paradigm is called a NoSQL DBMS (Desamsetti, 2016b).

NoSQL databases typically share the ability to readily scale horizontally, in addition to being nonrelational and not requiring defined schemas. This is something else that they have in common. Due to the lack of relations, join operations, customary in SQL, are rendered unnecessary here. Aside from that, completely different NoSQL implementations will each have a highly distinctive look. The performance of relational databases is improved by NoSQL thanks to a combination of new characteristics and advantages offered by the technology (Von & Datta, 2012). NoSQL databases offer greater flexibility and are more easily scaled horizontally when compared to conventional databases. NoSQL databases are anticipated to automatically manage and disseminate data, overcome errors, and restore the entire system automatically. The inconsistency of the data kept in NoSQL databases made them famous and gave them their name when the NoSQL technology started gaining traction. There might be a significant barrier for those businesses and systems in which a high level of consistency was necessary.

DATA MODELING TECHNIQUES

Scalability, performance, and consistency are used to compare NoSQL databases (Gutlapalli, 2017b). This element of NoSQL is well-studied in practice and theory since certain non-functional qualities are typically the key justification for its use, and fundamental distributed system conclusions like the CAP theorem apply to NoSQL systems. Unlike relational databases, NoSQL data modeling is poorly explored and lacks coherent theory (Ballamudi, 2019b). We briefly compare NoSQL system families from a data modeling perspective and discuss some standard modeling methodologies in this article. SQL and relational models were created to interface with users long ago. User-centricity had significant implications:

- SQL prioritizes aggregated reporting data over individual data elements because end users prefer it.
- Human users cannot explicitly manage concurrency, integrity, consistency, or data type validity. That's why SQL prioritizes transactional guarantees, schemas, and referential integrity.

However, software applications may not be interested in in-database aggregation and can control integrity and validity themselves. Additionally, removing these features greatly affected storage performance and scalability. This started a new data model evolution:

- Key-value storage is simple but powerful. Many strategies below work well with this paradigm (Pagán et al., 2015).
- Key-value model applicability to critical range processing scenarios is one of its most significant drawbacks. The ordered Key-Value model enhances aggregation and overcomes this issue.

- The ordered Key-Value model is powerful but lacks a value modeling framework. Applications can model values, but BigTable-style databases model values as a map-of-maps-of-maps of column families, columns, and timestamped versions.
- Document databases boost BigTable in two ways. First, values with schemes of any complexity, not just a map of maps. The second is database-managed indexes in some systems. Complete Text Search Engines offer customizable schema and automatic indexes, making them connected. Document databases group indexes by field names, while Search Engines group by field values (Gutlapalli, 2017a). Key-value stores like Oracle Coherence are moving toward document databases by adding indexes and in-database entry processors (Ballamudi, 2019a).
- Finally, graph data models evolved from Ordered Key-Value models. Hierarchical modeling makes alternative data models competitive in this domain, while graph databases provide transparent business entity modeling. Since many graph databases may model values as maps or documents, they are similar to document databases.

CONCEPTUAL TECHNIQUES

The fundamental ideas behind NoSQL data modeling will be covered in this part.

- Denormalization involves transferring data into numerous documents or tables to ease query processing or fit user data into a data model. Most of the methods in this article use denormalization. Denormalization aids these trade-offs:
  - ✓ IO per query vs. total data volume. Denormalization groups all query data together. Various query flows often access the same data in multiple ways. We must duplicate data, increasing data volume.
  - ✓ Total data volume vs. processing complexity. Modeling-time normalization and query-time join complicate query processors, especially in distributed systems. To ease query processing, denormalization stores data in a query-friendly structure (Seera & Jain. 2015).
- Aggregates: Soft schema is supported by all significant NoSQL genres:
  - ✓ Key-value stores and Graph Databases don't confine values; therefore, they can be in any format. Composite keys can change several records for one business entity. A user account can be depicted as a series of composite keys like UserID_name, UserID_email, UserID_messages, etc. User entries are not recorded if they have no email or messages.
  - ✓ BigTable models with flexible column families and cell versions support soft schema.
  - ✓ Document databases are schema-less, but some support user-defined schema validation.
- Soft schema lets one create nested entities and change their structures. This feature offers two main functionalities:
  - ✓ Minimizing one-to-many relationships with nested items reduces joins.
  - ✓ Masking "technical" distinctions between business entities and representing heterogeneous entities with one set of documents or tables.

This shows eCommerce product entity modeling. First, all products have ID, Price, and Description. Next, we learn that different products have different properties, such as Book Author or Jeans Length. These qualities are one-to-many or many-to-many, like Music Album Tracks. Next, fixed types may not model some entities. Jeans characteristics vary by brand and maker. Relational normalized data models can solve all these problems but could be more elegant (Thaduri, 2021). Soft schema lets us model all goods and their properties with one Aggregate (product). Update flows should be carefully considered because embedding with denormalization can affect performance and consistency.

- Application Side Joins: Joins are rarely supported in NoSQL. NoSQL's "question-oriented" structure means joins are generally handled at design time, unlike relational models, which are held at query execution time. Denormalization and Aggregates, or embedding nested items, can often prevent query time joins, which usually hurt performance. Joins are often unavoidable and should be handled by an application. The main uses are:
    - ✓ Links model many-to-many relationships and require joins.
    - ✓ Frequent entity internal changes make aggregates inapplicable. Rather than modifying a value, recording an event and joining the records at query time is better. Message systems can be described as User entities with nested Message entities. If messages are often appended, it may be best to extract Messages as distinct entities and join them to the User at query time:

General Modeling Techniques

This section covers general modeling strategies for NoSQL implementations.

- **Atomic Aggregates:** Not all NoSQL solutions support transactions. Distributed locks or application-managed MVCC can accomplish transactional behavior. However, Aggregates are often used to guarantee some ACID features. Robust transactional machinery is essential in relational databases because normalized data requires multi-place updates. However, Aggregates let one store and edit a business entity as one document, row, or key-value pair (Thaduri, 2020). Data modeling using Atomic Aggregates is not a complete transactional solution, but it can be used provided the store guarantees atomicity, locks, or test-and-set instructions (Hosen et al., 2021).
- **Enumerable Keys:** By hashing the key, an unordered Key-Value data architecture may partition entries across different servers, which is its most prominent feature. Sorting complicates things, but an application can benefit from sorted keys even if storage doesn't. Example: email message modeling. Some NoSQL stores have atomic counters for sequential IDs. UserID_messageID can be used as a composite key to store messages. Previous messages can be traversed with the latest message ID. We can also cross prior and subsequent messages for any message ID. Daily buckets can hold messages. This lets one move through a mailbox from any date or the present date (Chen et al., 2019).
- **Dimensionality Reduction:** Dimensionality Reduction maps multidimensional data to Key-Value or other non-multidimensional structures. Traditional geographic information systems index with Quadtrees or R-trees. Changing these structures in place is costly when data volumes are significant. We may also traverse the 2D structure and flatten it into a list. Geohashes are famous examples of this. Geohashes

fill 2D space with a Z-like scan and encode each motion as 0 or 1 depending on direction. The longitude and latitude bits are interleaved with moves. The image below shows the encoding process, with black and red bits representing longitude and latitude**.** Geohashes use bit-wise coding proximity to estimate the distance between regions, as seen in the. Geohash encoding stores geographical data using fundamental data structures like sorted key values preserving spatial relationships. The BigTable Dimensionality Reduction method was explained (Khan et al., 2017).

- **Index Table: An** Index Table is a simple way to use indexes in stores without internal indexes. BigTable-style databases dominate such storage. Create and maintain a custom table with access-pattern-based keys. The master table stores user accounts that can be accessed by ID. An additional table with city as a key can provide a query to fetch all users by city: An index table can be updated with each master table update or in batches. Either way, it increases performance penalties and consistency issues (Thaduri, 2019).
- **Composite Key Index:** Composite key is generic but valuable in a store with ordered keys. Composite keys and secondary sorting create a multidimensional index comparable to Dimensionality Reduction. Example: Take a set of user statistics records. Suppose the database supports partial key matches (like BigTable-style systems do). In that case, we can utilize keys in a format (State:City: UserID) to iterate over records for a state or city to aggregate these statistics.
- **Composite Key Aggregation:** Composite keys can be used for indexing and grouping. Consider an example. Internet users and their site visits are recorded in several log files. We want to count unique users for each Site. The SQL statement below is similar. Keeping all records for one User collocated allows us to get a frame into memory (one User can't make too many events) and eliminate site duplicates using a hash table or something (Bodepudi et al., 2021). One way is to create one entry per person and add sites when occurrences occur. In most implementations, entry modification is less efficient than entry insertion (Gutlapalli, 2016b).
- **Inverted Search—Direct Aggregation:** This method is more data processing than modeling. This tendency also affects data models. This method uses an index to discover data that fulfills requirements and aggregates it using original representation or complete scans. Consider an example (Thaduri, 2017; Mandapuram, 2017b). Internet surfers and their site visits are recorded in several log files. Let each record contain user ID, classifications (Men, Women, Bloggers, etc.), city, and visited Site. The purpose is to characterize the audience that meets some criteria (Site, city, etc.) in terms of unique users for each category. Searching for people who fulfill the criteria is efficient with inverted indexes like {Category -> [user IDs]} or {Site -> [user IDs]}. Such indexes can intersect or unify user IDs (which is efficient if user IDs are stored as sorted lists or bit sets) to find an audience. Describe an audience like an aggregation query. It cannot be efficiently handled by an inverted index with several categories. To address this, create a direct index {UserID -> [Categories]} and iterate over it to generate a final report. See this schema below. Finally, random record retrieval for each audience user ID can be wasteful. This can be solved using batch query processing. This allows precomputed user sets for diverse criteria to compute all reports for this batch of audiences in one direct or inverse index search (Pokorny, 2013; Thaduri, 2018).

ENTERPRISE CONTENT MANAGEMENT

"Enterprise content management" (ECM) "comprises the strategies, processes, methods, systems, and technologies that are necessary for capturing, creating, managing, using, publishing, storing, preserving, and disposing of content within and between organizations" according to Wikipedia. ECM encompasses a comprehensive range of functional areas, including

- Administration of Documents;
- A collaborative effort among the many support systems;
- Management of Content on the Web;
- Administration of Records;
- Workflow Management and Management of Business Processes

ECM systems are currently confronted with various difficulties, the most significant of which is the scaling problem, in which the volume of data grows at a rate greater than that of the available computing capabilities (Gutlapalli, 2016a). When organizations move to cloud-enabled solutions, large multi-tenant databases are created (Mandapuram et al., 2020). Additionally, organizations consume data from social media platforms and data streams generated by devices (Mandapuram, 2017a). Enterprise data and its variety are growing in volume and velocity at an exponential rate. A costly approach is to scale up, which means replacing a smaller machine with a larger one. A 10-terabyte system could cost 100 times more than a 1-terabyte system (Dai et al., 2013).

Lal pointed out in 2015 that the problem is not information overload but rather a breakdown of the filtering system. We are rapidly moving away from outdated data persistence methods since relational database systems have inherent scalability issues. Because of this, a wide variety of solutions known as NoSQL (which stands for "not only SQL") have been developed to address the problems listed above (Ballamudi, 2020).

ELASTICSEARCH

Elasticsearch is the data store we decided to use for our search architecture. The most important advantages are its high scalability and excellent search capabilities. On a laptop, the Elasticsearch database may be installed, and once it is, we are ready to begin working immediately with no configuration required (Lal & Ballamudi, 2017). On the other hand, Elasticsearch may be set up to manage enormous databases when placed on a multi-node cluster and configured appropriately. Elasticsearch is deployed on a 75-node cluster split across two data centers by Synthesio (Lal, 2016). This configuration enables the company to quickly retrieve up to 50 million documents from tens of billions (WEB (d)).

Although there are NoSQL document solutions (such as MongoDB) that provide more functionality and flexibility than Elasticsearch, the capabilities of Elasticsearch are still the most excellent fit for our domain and architecture in mind. In terms of search performance, Elasticsearch is superior to MongoDB and other platforms that offer comparable functionalities, according to the evaluation findings (Desamsetti, 2020). Elasticsearch performs approximately four to five times faster than MongoDB when the request flows consist of fifty percent reading and fifty percent writing. Elasticsearch can conduct search operations against nested objects 20 times quicker and aggregation operations 40 times faster. In addition, the perforations (Ballamudi & Desamsetti, 2017).

Architecture for Enterprise Search

It is only possible to determine whether a document database model will perform efficiently by reading a description of the model (Desamsetti & Mandapuram, 2017). One must consider how users will query the database, the quantity of data inserted, and the frequency and nature of document updates. Because of this, we think about the patterns of data utilization before making decisions about the data model and the architecture (Thaduri & Lal, 2020). We propose to use two data stores as the foundation for the persistence architecture. The primary store is a SQL database, and both the insertion of new data and the upkeep of the existing data are performed in this database (Mandapuram et al., 2018). A database powered by Elasticsearch serves as the secondary store, and its primary functions are to search for and retrieve data. After being successfully added to the primary database, the newly created data objects are replicated to the secondary storage (Gutlapalli et al., 2019). The following are some of the benefits of taking this approach:

- A lighter strain is placed on the primary store because the secondary store is responsible for managing search and retrieval requests;
- requests for searching and retrieving data do not have to wait for operations to be written for index locks to be released;
- Elasticsearch inverted indexes are utilized for quick searching, including full-text searching;
- Elasticsearch sharding and replication are used to improve performance in response to an increase in the number of data and requests;
- Replication of Elasticsearch is used to ensure that high data availability is maintained.

No locking: Write transactions in relational databases involving various data items (such as tables and indexes). When a transaction is in progress, any objects used must first be "locked," meaning they must be rendered unavailable to other processes. This indicates that other requests must wait until the transaction releases the locks, including read and write. This results in a rapid decline in performance whenever the number of requests increases (Rats, 2017). Our model makes use of the no-locking write functionality that Elasticsearch offers. Since Lucene indexes used by Elasticsearch are immutable, there is no requirement to lock the index before writing data, as Amin & Mandapuram (2021) stated. Instead of merging existing index segments, new index segments are generated to index newly added data; this occurs later in the background. No matter how rapidly new data is being duplicated from the primary store, the Elasticsearch secondary database will always be accessible for searches and retrieving stored information (Desamsetti, 2018).

Scalability: The Elasticsearch database is divided up into a few different shards. When there is a greater need for storage space, additional nodes can be added to the cluster. When other nodes are added, Elasticsearch will immediately move any affected shards (Lal et al., 2018). Therefore, a database with five shards and no replicas can operate on a cluster with anywhere from one to five nodes (Bodepudi et al., 2019). **Availability:** It is possible to set up an Elasticsearch database to support replicas, also known as copies, of the shards. The replication factor of Elasticsearch refers to the number of copies of the index that must be kept (Desamsetti & Lal, 2019). For instance, if the replication factor is set to two, Elasticsearch will generate and keep track of two replicas of each added shard. Because Elasticsearch distributes replicas of a shared among the many nodes that make up a cluster (and because using replication factor 2 requires a cluster with at least three nodes), the minimum size of a cluster required is three.

Replicas can scale Elasticsearch databases to support an arbitrary number of nodes. While fresh data is written to the primary (master) copy of the shard, it is spread among the copies and then copied to the replicas. Search requests are also dispersed among the replicas (Desamsetti, 2021). On display in Elasticsearch cluster consisting of five shards and a clustering factor of two. Here, S(i) refers to a master copy of shard i, R(i,j) represents a replica of shard i, and N1, N2, and N3 are cluster nodes.

Shards and replicas make it possible to process data requests more quickly. In addition, because of the replicas, the cluster can continue to operate normally even if some of the nodes in the cluster are unavailable (Hosen & Gutlapalli, 2021). Although each cluster node contains a full data copy, it is unsafe to allow users in when the connection between all three nodes has been lost. This can lead to a phenomenon known as split brain, which occurs when users interact with three independent systems simultaneously. The sample cluster presented earlier is still fully functional even if one of the cluster nodes becomes unavailable (Karanjekar & Chandak, 2017).

ECM Data Model

ECM systems append metadata to documents to improve the efficiency of business procedures inside an organization. Although the metadata fields can vary, the data model should primarily consist of the following:

- The complete written content of the document
- Title of the document number of the document
- Leader or responsible party
- The beginning of creation
- The cut-off time
- The groups to which the document belongs (such as the case, the folder, or the project).
- Status
- A list of task descriptions that include comments, the author of the task, the person in responsibility for the task, the kind of the task, and the task's status

The named metadata fields contain all the information needed for the data access attributes detailed above or can be generated from those fields (Koehler et al., 2020). Maintaining our focus on optimizing efficiency, we have added an element containing a list of user IDs authorized to access the content (Mandapuram & Hosen, 2018). This is a one-of-a-kind list of user IDs that includes the user ID of the person in charge of the document in addition to the user IDs of the authors of the tasks in the document as well as the user IDs of the persons in charge of the tasks in the document (a user ID is only included in this list once, regardless of how many times the ID appears in the role of a task author or a person in charge of the task or document).

The parent-child relationship that exists between the document and its tasks is represented by the data object that has been described. Relational databases make use of two tables and a parent-child relationship between the two tables (Ballamudi et al., 2021). Aggregated models are preferred by NoSQL databases, meaning tasks should be represented as nested objects and stored along with the content as a single aggregated item. However, there are more effective courses of action than this. Creating two distinct data types—one for documents and one for tasks—and storing a reference to a parent document within a job provides a feasible alternative. For situations like this, Elasticsearch enables users to arrange one data object type to function as a child of another data object type (Kaur & Kanwalvir, 2016).

The two possible configurations—nested and parent-child—have several benefits and drawbacks. The following are some of the many benefits of using a parent-child model (WEB (a)):

- It is possible to make changes to the parent object without having to re-index the children;
- It is possible to create new child objects, modify existing ones, or remove them without having any impact on the parent or the other children;
- When a search is performed, the results may include child documents as possible matches.

The primary benefit of employing the layered model is that, on average, it can be anywhere from five to ten times quicker (WEB (b)).

It is also crucial that ECM systems include the ability to search for tasks (for example, to provide a list of tasks that a specific person is currently working on). As a result, the parent-child model is a more suitable contender for our investigation. Still, there is one more option available to us here (Deming et al., 2018). Since the task data is typically requested for the present data, we might want to think about making the following changes to our model:

- Remove any outdated data from the primary store, and make sure that only the secondary storage has all of the data;
- We should carry out advanced task searches using the principal store.

In this situation, the nested model might be the best option for the secondary store. However, we will continue to focus on the parent-child model and not pursue this particular line of inquiry now (Dekkati et al., 2016).

It is also crucial that ECM systems include the ability to search for tasks (for example, to provide a list of tasks that a specific person is currently working on). As a result, the parent-child model is a more suitable contender for our investigation. Still, there is one more option available to us here (Dekkati & Thaduri, 2017). Since the task data is typically requested for the present data, we might want to think about making the following changes to our model:

- Delete any older data from the primary storage, and only keep complete copies on the secondary store.
- We should carry out advanced task searches using the principal store.

In this situation, the nested model might be the best option for the secondary store. However, we will continue to focus on the parent-child model and not pursue this particular line of inquiry now.

In this scenario, the Document ID and Task ID header properties are the exclusive keys for their respective objects. The Task object contains a reference to its parent Document object under the Document ID property of the Task object (Ballamudi, 2016). In this case, we use denormalization, and the document number is also incorporated into the task object. This can significantly boost efficiency because the document number is frequently utilized in activity-related searches. Due to the infrequency of changes to document numbers, the administrative burden of storing duplicated data is of very minor significance. The User IDs attribute holds the IDs of users with direct access to the document (for more information on this topic, check the prior section of this chapter) (Nadeem et al., 2017).

PERFORMANCE

There are some findings of the evaluation of the performance of NoSQL databases that are available (WEB (c)). The evaluation is carried out on several typical workloads, such as those offered by the YCSB (Yahoo Cloud Serving Benchmark) framework (WEB (e)), with the primary objective of providing a basis for comparing various NoSQL systems (Reddy et al., 2020). We use the methodology created in our prior research supported by the European Union and titled "Definition and Analysis of Models for Advanced Data Visualisation". In contrast to the method taken by YCSB, we begin the definition of workload with user business actions (such as "show my urgent tasks") and the frequency with which they occur. User business tasks are further broken down into sequences of user interactions (i.e., a user request that can be carried out by one or more data requests without user engagement). User interactions can also be broken down into a string of data requests. Because of this, we can construct a workload and estimate the performance of our search model for the specified number of business users (Thaduri et al., 2016). We utilize a list of data request sequences for performance measurement. This list includes search (for example, full-text search inside document content), filtering and processing of aggregates, and document and task creation and update (Dekkati et al., 2019). We make several assumptions on the rates at which an ECM user carries out data request sequences. The research findings referred to above are utilized in making assumptions regarding the frequencies of completion of the data request sequences by an ECM user. The many examples of user activities and interactions used for performance testing are outlined (Thodupunori & Gutlapalli, 2018).

CONCLUSION

We created a multilingual persistence architecture with two data stores: SQL database and Elasticsearch. Multi-user access to data objects, while the secondary store efficiently handles enormous search requests. Inserted/updated into primary store and searched/accessed in secondary store. The primary store enables safe concurrent processing of data through ACID transactions. We will improve our model to capitalize on Elasticsearch's scalability. The primary store is no longer needed if all searchable data is in the secondary store. This would limit primary store growth, which is significant due to SQL's scaling issues. Users of different business roles may execute data request sequences to varying frequencies due to their activity patterns. Future workload generation improvements should consider this.

REFERENCES

Amin, R., & Mandapuram, M. (2021). CMS - Intelligent Machine Translation with Adaptation and AI. *ABC Journal of Advanced Research*, *10*(2), 199-206. https://doi.org/10.18034/abcjar.v10i2.693

Ballamudi, V. K. R. (2016). Utilization of Machine Learning in a Responsible Manner in the Healthcare Sector. *Malaysian Journal of Medical and Biological Research*, *3*(2), 117-122. https://mjmbr.my/index.php/mjmbr/article/view/677

Ballamudi, V. K. R. (2019a). Artificial Intelligence: Implication on Management. *Global Disclosure of Economics and Business*, *8*(2), 105-118. https://doi.org/10.18034/gdeb.v8i2.540

Ballamudi, V. K. R. (2019b). Road Accident Analysis and Prediction using Machine Learning Algorithmic Approaches. *Asian Journal of Humanity, Art and Literature*, *6*(2), 185-192. https://doi.org/10.18034/ajhal.v6i2.529

Ballamudi, V. K. R. (2019c). Hybrid Automata: An Algorithmic Approach Behavioral Hybrid Systems. *Asia Pacific Journal of Energy and Environment*, *6*(2), 83-90. https://doi.org/10.18034/apjee.v6i2.541

Ballamudi, V. K. R. (2020). Militarization of Space. *Asian Journal of Applied Science and Engineering*, *9*(1), 169–178. https://doi.org/10.18034/ajase.v9i1.38

Ballamudi, V. K. R., & Desamsetti, H. (2017). Security and Privacy in Cloud Computing: Challenges and Opportunities. *American Journal of Trade and Policy*, *4*(3), 129–136. https://doi.org/10.18034/ajtp.v4i3.667

Ballamudi, V. K. R., Lal, K., Desamsetti, H., & Dekkati, S. (2021). Getting Started Modern Web Development with Next.js: An Indispensable React Framework. *Digitalization & Sustainability Review*, *1*(1), 1–11. https://upright.pub/index.php/dsr/article/view/83

Bodepudi, A., Reddy, M., Gutlapalli, S. S., & Mandapuram, M. (2019). Voice Recognition Systems in the Cloud Networks: Has It Reached Its Full Potential?. *Asian Journal of Applied Science and Engineering*, *8*(1), 51–60. https://doi.org/10.18034/ajase.v8i1.12

Bodepudi, A., Reddy, M., Gutlapalli, S. S., & Mandapuram, M. (2021). Algorithm Policy for the Authentication of Indirect Fingerprints Used in Cloud Computing. *American Journal of Trade and Policy*, *8*(3), 231–238. https://doi.org/10.18034/ajtp.v8i3.651

Chen, S., Thaduri, U. R., & Ballamudi, V. K. R. (2019). Front-End Development in React: An Overview. *Engineering International*, *7*(2), 117–126. https://doi.org/10.18034/ei.v7i2.662

Dai, C., Ye, Y., Liu, T. J., Zheng, J. J. (2013). Design of High Performance Cloud Storage Platform Based on Cheap PC Clusters Using MongoDB and Hadoop. *Applied Mechanics and Materials*, *380-384*, 2050. https://doi.org/10.4028/www.scientific.net/AMM.380-384.2050

Dekkati, S., & Thaduri, U. R. (2017). Innovative Method for the Prediction of Software Defects Based on Class Imbalance Datasets. *Technology & Management Review*, *2*, 1–5. https://upright.pub/index.php/tmr/article/view/78

Dekkati, S., Lal, K., & Desamsetti, H. (2019). React Native for Android: Cross-Platform Mobile Application Development. *Global Disclosure of Economics and Business*, *8*(2), 153-164. https://doi.org/10.18034/gdeb.v8i2.696

Dekkati, S., Thaduri, U. R., & Lal, K. (2016). Business Value of Digitization: Curse or Blessing?. *Global Disclosure of Economics and Business*, *5*(2), 133-138. https://doi.org/10.18034/gdeb.v5i2.702

Deming, C., Dekkati, S., & Desamsetti, H. (2018). Exploratory Data Analysis and Visualization for Business Analytics. *Asian Journal of Applied Science and Engineering*, *7*(1), 93–100. https://doi.org/10.18034/ajase.v7i1.53

Desamsetti, H. (2016a). A Fused Homomorphic Encryption Technique to Increase Secure Data Storage in Cloud Based Systems. *The International Journal of Science & Technoledge*, *4*(10), 151-155.

Desamsetti, H. (2016b). Issues with the Cloud Computing Technology. *International Research Journal of Engineering and Technology (IRJET)*, *3*(5), 321-323.

Desamsetti, H. (2018). Internet of Things (IoT) Technology for Use as Part of the Development of Smart Home Systems. *International Journal of Reciprocal Symmetry and Theoretical Physics*, *5*, 14–21. https://upright.pub/index.php/ijrstp/article/view/89

Desamsetti, H. (2020). Relational Database Management Systems in Business and Organization Strategies. *Global Disclosure of Economics and Business*, *9*(2), 151-162. https://doi.org/10.18034/gdeb.v9i2.700

Desamsetti, H. (2021). Crime and Cybersecurity as Advanced Persistent Threat: A Constant E-Commerce Challenges. *American Journal of Trade and Policy*, *8*(3), 239–246. https://doi.org/10.18034/ajtp.v8i3.666

Desamsetti, H., & Lal, K. (2019). Being a Realistic Master: Creating Props and Environments Design for AAA Games. *Asian Journal of Humanity, Art and Literature*, *6*(2), 193-202. https://doi.org/10.18034/ajhal.v6i2.701

Desamsetti, H., & Mandapuram, M. (2017). A Review of Meta-Model Designed for the Model-Based Testing Technique. *Engineering International*, *5*(2), 107–110. https://doi.org/10.18034/ei.v5i2.661

Gutlapalli, S. S. (2016a). An Examination of Nanotechnology's Role as an Integral Part of Electronics. *ABC Research Alert*, *4*(3), 21–27. https://doi.org/10.18034/ra.v4i3.651

Gutlapalli, S. S. (2016b). Commercial Applications of Blockchain and Distributed Ledger Technology. *Engineering International*, *4*(2), 89–94. https://doi.org/10.18034/ei.v4i2.653

Gutlapalli, S. S. (2017a). Analysis of Multimodal Data Using Deep Learning and Machine Learning. *Asian Journal of Humanity, Art and Literature*, *4*(2), 171–176. https://doi.org/10.18034/ajhal.v4i2.658

Gutlapalli, S. S. (2017b). The Role of Deep Learning in the Fourth Industrial Revolution: A Digital Transformation Approach. *Asian Accounting and Auditing Advancement*, *8*(1), 52–56. Retrieved from https://4ajournal.com/article/view/77

Gutlapalli, S. S. (2017c). An Early Cautionary Scan of the Security Risks of the Internet of Things. *Asian Journal of Applied Science and Engineering*, *6*, 163–168. Retrieved from https://ajase.net/article/view/14

Gutlapalli, S. S., Mandapuram, M., Reddy, M., & Bodepudi, A. (2019). Evaluation of Hospital Information Systems (HIS) in terms of their Suitability for Tasks. *Malaysian Journal of Medical and Biological Research*, *6*(2), 143–150. https://mjmbr.my/index.php/mjmbr/article/view/661

Hosen, M. S., & Gutlapalli, S. S. (2021). A Study of Innovative Class Imbalance Dataset Software Defect Prediction Methods. *Asian Journal of Applied Science and Engineering*, *10*(1), 52–55. https://doi.org/10.18034/ajase.v10i1.52

Hosen, M. S., Thaduri, U. R., Ballamudi, V. K. R., & Lal, K. (2021). Photo-Realistic 3D Models and Animations for Video Games and Films. *Engineering International*, *9*(2), 153–164. https://doi.org/10.18034/ei.v9i2.668

Karanjekar, J. B., Chandak, M. B. (2017). Uniform Query Framework for Relational and NoSQL Databases. *Computer Modeling in Engineering & Sciences*, *113*(2), 177-187. https://doi.org/10.3970/cmes.2017.113.177

Kaur, A., Kanwalvir S. D. (2016). Performance Evaluation for Crud Operations in NoSQL Databases. *i-manager's Journal on Cloud Computing*, *3*(2), 1-9.

Khan, W., Ahmed, E., Shahzad, W. (2017). Predictive Performance Comparison Analysis of Relational & NoSQL Graph Databases. *International Journal of Advanced Computer Science and Applications*, *8*(5). https://doi.org/10.14569/IJACSA.2017.080564

Koehler, S., Desamsetti, H., Ballamudi, V. K. R., & Dekkati, S. (2020). Real World Applications of Cloud Computing: Architecture, Reasons for Using, and Challenges. *Asia Pacific Journal of Energy and Environment*, *7*(2), 93-102. https://doi.org/10.18034/apjee.v7i2.698

Lal, K. (2015). How Does Cloud Infrastructure Work?. *Asia Pacific Journal of Energy and Environment*, *2*(2), 61-64. https://doi.org/10.18034/apjee.v2i2.697

Lal, K. (2016). Impact of Multi-Cloud Infrastructure on Business Organizations to Use Cloud Platforms to Fulfill Their Cloud Needs. *American Journal of Trade and Policy*, *3*(3), 121–126. https://doi.org/10.18034/ajtp.v3i3.663

Lal, K., & Ballamudi, V. K. R. (2017). Unlock Data's Full Potential with Segment: A Cloud Data Integration Approach. *Technology & Management Review*, *2*(1), 6–12. https://upright.pub/index.php/tmr/article/view/80

Lal, K., Ballamudi, V. K. R., & Thaduri, U. R. (2018). Exploiting the Potential of Artificial Intelligence in Decision Support Systems. *ABC Journal of Advanced Research*, *7*(2), 131-138. https://doi.org/10.18034/abcjar.v7i2.695

Mandapuram, M. (2017a). Application of Artificial Intelligence in Contemporary Business: An Analysis for Content Management System Optimization. *Asian Business Review*, *7*(3), 117–122. https://doi.org/10.18034/abr.v7i3.650

Mandapuram, M. (2017b). Security Risk Analysis of the Internet of Things: An Early Cautionary Scan. *ABC Research Alert*, *5*(3), 49–55. https://doi.org/10.18034/ra.v5i3.650

Mandapuram, M., & Hosen, M. F. (2018). The Object-Oriented Database Management System versus the Relational Database Management System: A Comparison. *Global Disclosure of Economics and Business*, *7*(2), 89–96. https://doi.org/10.18034/gdeb.v7i2.657

Mandapuram, M., Gutlapalli, S. S., Bodepudi, A., & Reddy, M. (2018). Investigating the Prospects of Generative Artificial Intelligence. *Asian Journal of Humanity, Art and Literature*, *5*(2), 167–174. https://doi.org/10.18034/ajhal.v5i2.659

Mandapuram, M., Gutlapalli, S. S., Reddy, M., Bodepudi, A. (2020). Application of Artificial Intelligence (AI) Technologies to Accelerate Market Segmentation. *Global Disclosure of Economics and Business 9*(2), 141–150. https://doi.org/10.18034/gdeb.v9i2.662

Nadeem, Q. M., Culmone, R., Mostarda, L. (2017). Modeling Temporal Aspects of Sensor Data for MongoDB NoSQL Database. *Journal of Big Data*, *4*(1), 1-35. https://doi.org/10.1186/s40537-017-0068-5

Pagán, J. E., Cuadrado, J. S., Molina, J. G. (2015). A Repository for Scalable Model Management. *Software and Systems Modeling*, *14*(1), 219-239. https://doi.org/10.1007/s10270-013-0326-8

Pokorny, J. (2013). NoSQL Databases: A Step to Database Scalability in Web Environment. *International Journal of Web Information Systems*, *9*(1), 69-82. https://doi.org/10.1108/17440081311316398

Rats, J. (2017). Use of NoSQL Technology for Secure and Fast Search of Enterprise Data. *Baltic Journal of Modern Computing*, *5*(2), 147-163. https://doi.org/10.22364/bjmc.2017.5.2.01

Reddy, M., Bodepudi, A., Mandapuram, M., & Gutlapalli, S. S. (2020). Face Detection and Recognition Techniques through the Cloud Network: An Exploratory Study. *ABC Journal of Advanced Research*, *9*(2), 103–114. https://doi.org/10.18034/abcjar.v9i2.660

Seera, N. K., Jain, V. (2015). Perspective of Database Services for Managing Large-Scale Data on the Cloud: A Comparative Study. *International Journal of Modern Education and Computer Science*, *7*(6), 50-58. https://doi.org/10.5815/ijmecs.2015.06.08

Thaduri, U. R. (2017). Business Security Threat Overview Using IT and Business Intelligence. *Global Disclosure of Economics and Business*, *6*(2), 123-132. https://doi.org/10.18034/gdeb.v6i2.703

Thaduri, U. R. (2018). Business Insights of Artificial Intelligence and the Future of Humans. *American Journal of Trade and Policy*, *5*(3), 143–150. https://doi.org/10.18034/ajtp.v5i3.669

Thaduri, U. R. (2019). Android & iOS Health Apps for Track Physical Activity and Healthcare. *Malaysian Journal of Medical and Biological Research*, *6*(2), 151-156. https://mjmbr.my/index.php/mjmbr/article/view/678

Thaduri, U. R. (2020). Decision Intelligence in Business: A Tool for Quick and Accurate Marketing Analysis. *Asian Business Review*, *10*(3), 193–200. https://doi.org/10.18034/abr.v10i3.670

Thaduri, U. R. (2021). Virtual Reality & Artificial Intelligence in Real Estate Business: A Tool for Effective Marketing Campaigns. *Asian Journal of Applied Science and Engineering*, *10*(1), 56–65. https://doi.org/10.18034/ajase.v10i1.54

Thaduri, U. R., & Lal, K. (2020). Making a Dynamic Website: A Simple JavaScript Guide. *Technology & Management Review*, *5*, 15–27. https://upright.pub/index.php/tmr/article/view/81

Thaduri, U. R., Ballamudi, V. K. R., Dekkati, S., & Mandapuram, M. (2016). Making the Cloud Adoption Decisions: Gaining Advantages from Taking an Integrated Approach. *International Journal of Reciprocal Symmetry and Theoretical Physics*, *3*, 11–16. https://upright.pub/index.php/ijrstp/article/view/77

Thodupunori, S. R., & Gutlapalli, S. S. (2018). Overview of LeOra Software: A Statistical Tool for Decision Makers. *Technology & Management Review, 3*(1), 7–11.

Von D. W. C., Datta, A. (2012). Multiterm Keyword Search in NoSQL Systems. *IEEE Internet Computing*, *16*(1), 34-42. https://doi.org/10.1109/MIC.2011.140

--0--