**ISSN:** 2583-9888
**DOI:** doi.org/10.51219/JAIMLD/Naveen-koka/79

URF PUBLISHERS
connect with research world

Journal of Artificial Intelligence, Machine Learning and Data Science

https://urfpublishers.com/journal/artificial-intelligence

**Vol:** 1 & **Iss:** 1

*Research Article*

# Data Evaluation System Utilizing Logic-Based Rules

**Naveen Koka***

Naveen Koka, USA

**Citation:** Naveen Koka. Data Evaluation System Utilizing Logic-Based Rules. *J Artif Intell Mach Learn & Data Sci 2022*, 1(1), 250-253. DOI: doi.org/10.51219/JAIMLD/Naveen-koka/79

**Received:** 02 January, 2022; **Accepted:** 18 January, 2022; **Published:** 20 January, 2022

***Corresponding author:** Naveen Koka, USA

## ABSTRACT

In modern software systems, the dynamic generation of queries plays a crucial role in retrieving relevant data based on user-defined criteria. This paper explores a runtime approach where criteria names are utilized to dynamically construct queries tailored to specific system contexts. The generated queries vary depending on the requirements of the system, ensuring the retrieval of data that meets the specified criteria.

Furthermore, to enhance performance and optimize resource utilization, the paper proposes a caching mechanism for storing generated queries. Once a query is generated, it is cached to eliminate the need for repeated evaluation, thereby reducing overhead and improving response times. This caching strategy ensures that subsequent data retrieval operations can be executed efficiently, contributing to a more responsive and scalable system architecture.

Overall, this paper presents a dynamic query generation and caching framework that facilitates efficient data retrieval in diverse system environments. By leveraging runtime query generation and caching, software systems can achieve enhanced performance and responsiveness, ultimately improving user experience and system scalability.

**Keywords:** Rules Engine, Configured based criteria, Reusable components

## 1. Introduction

Efficient data retrieval is a cornerstone of modern software systems, often reliant on dynamically generated queries tailored to specific criteria. In this context, the runtime generation of queries plays a pivotal role, allowing systems to adapt to varying requirements and user-defined conditions. This paper delves into the dynamic query generation process, focusing on its significance in retrieving pertinent data within diverse system contexts. Furthermore, the paper explores the integration of a caching mechanism to optimize query performance and resource utilization. By caching generated queries, the system mitigates the overhead associated with repetitive query evaluation, thereby enhancing response times and overall system efficiency. Through a combination of dynamic query generation and caching, software systems can achieve heightened agility and scalability, catering to evolving user needs and system demands. This introduction sets the stage for a comprehensive exploration of the dynamic query generation and caching framework, elucidating its role in facilitating efficient data retrieval and system optimization.

## 2. Problem Statement

Many software implementations encounter failure because they overlook the necessity of designing certain crucial components with a generic approach. This oversight proves costly for companies, both in terms of time and finances. When these components lack generality, the process of searching for new tools and adapting them becomes protracted and resource-intensive.

Key components such as dynamic form creation, runtime handle configuration, and criteria evaluation often fall victim to this oversight. Without a generic framework in place, each instance demands individualized development and adaptation efforts. Consequently, the absence of a standardized approach exacerbates the challenges of integrating and optimizing these critical functionalities within the software ecosystem.

Addressing this issue requires a paradigm shift towards prioritizing generic design principles in software development. By imbuing essential components with adaptability and flexibility, companies can streamline their processes, mitigate risks, and enhance overall efficiency. Embracing this approach ensures that future implementations are resilient, cost-effective, and better positioned to meet evolving business needs.

## 3. Let's Interact with an Example

Consider two customers who have purchased home appliances from company A. Customer 1 reports an issue where the system is generating noise, while customer 2 faces a more severe problem with the system being completely down. Clearly, the urgency differs between these cases, with customer 2's issue demanding immediate attention due to its critical nature.

In a conventional implementation relying on if-else statements, assigning priorities and paths for resolution requires explicit coding. For instance, if a status qualifies as high priority, it must be routed through a critical path, while lower priority statuses follow their own resolution paths. However, each addition or modification to these statuses necessitates coding, testing, and deployment, leading to a cumbersome and time-consuming process.

This scenario underscores the limitations of a rigid if-else structure in managing varying priorities effectively. A logic-based rules engine offers a more dynamic solution, where priorities and resolution paths are determined by configurable rules rather than hard-coded conditions. This approach enables easier adaptation to changing requirements and facilitates the seamless incorporation of new statuses without the need for extensive coding efforts. Thus, by implementing a logic-based rules engine, company A can enhance its responsiveness to customer issues while minimizing the overhead associated with software maintenance and updates.

### 3.1. Outcome

Once the configuration is established to handle this prioritization and resolution routing, adjusting becomes a swift task. The ability to modify configurations allows for quick adaptations to evolving needs. Compared to the traditional approach of coding, testing, and releasing changes, the time required for such modifications is drastically reduced, by up to 99%. This efficiency stems from the decoupling of business logic from the codebase. With a configuration-driven system, alterations can be implemented through simple adjustments to the configuration settings. This streamlined process eliminates the need for extensive coding and testing cycles, resulting in significant time savings, and enabling rapid responses to changing priorities or requirements.

By harnessing the power of configurable rules and logic-based engines, companies can achieve unparalleled agility in managing and resolving customer issues. This approach not only accelerates problem resolution but also enhances overall system flexibility, empowering organizations to adapt swiftly to market dynamics and customer needs.

## 4. Solution

The In this discussion, we'll explore the concept of making criteria more dynamic to facilitate faster development and reduce the likelihood of errors. By adopting a flexible approach to criteria definition, developers can efficiently apply these concepts across various configurations, promoting reusability and consistency in both frontend and backend development.

Criteria, in essence, encapsulates the conditions used to make decisions in programming constructs like if-else statements. Regardless of the programming language or context, criteria serve as the foundational logic governing the flow of execution. This fundamental concept applies universally, whether in user interface design or backend system architecture.

The proposed solution involves enhancing the dynamism of criteria, enabling developers to leverage them wherever necessary. By embracing this approach, development processes become more streamlined and error resistant. Furthermore, by abstracting criteria into configurable components, developers can build a framework that fosters rapid development and easy adaptation to evolving requirements.

By adopting a mindset that promotes the generic application of criteria, developers can cultivate a more efficient and resilient development environment. This shift towards dynamic criteria empowers teams to accelerate development cycles, reduce error rates, and create solutions that are adaptable to a wide range of contexts and use cases.

"Criteria" refers to the expressions employed for assessing and evaluating the attributes encapsulated by the fields, aiding in decision-making processes within the system.

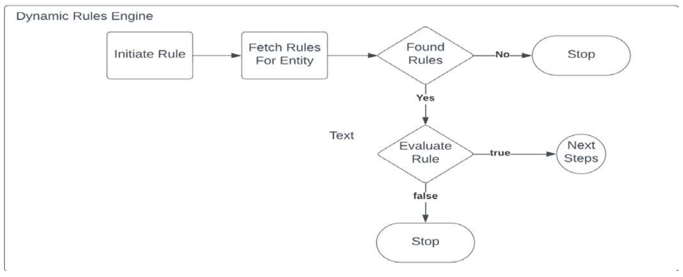

**Figure 1:** Dynamic rules engine.

## 5. Concepts used for Solutioning

The solution for making criteria more dynamic relies on several key concepts. First, it treats combined fields as entities, where a data table represents an entity and its fields represent attributes. This perspective allows for a structured understanding of the data and its characteristics, facilitating more organized development and configuration.

In cases where an entity lacks a physical representation, the solution introduces the concept of virtual entities. These virtual entities are defined through configurations, specifying their attributes and characteristics without requiring a tangible presence in the system. This approach enables developers to work with abstract entities, expanding the scope of what can be modeled and manipulated within the system.

By treating entities and attributes as configurable components, the solution promotes flexibility and adaptability in system design. Developers can define and modify entities and their attributes through configurations, eliminating the need for manual coding and facilitating rapid iteration and experimentation. This configuration-driven approach streamlines development processes and reduces the likelihood of errors, enhancing overall productivity and system reliability.

Overall, these concepts form the foundation of a solution that empowers developers to create dynamic and customizable systems. By leveraging configurations to define entities and attributes, the solution enables developers to build robust, scalable, and adaptable software solutions that meet the evolving needs of users and organizations.

The datatable object represents an entity within the system, named “Account”.

JSON has the representation for entity schema.

```
{
  “$schema”: “http://json-schema.org/draft-04/schema#”,
  “title”: “Account”,
  “description”: “Customer information”,
  “type”: “object”,

  “properties”: {

    “id”: {
      “description”: “The unique identifier for a account”,
      “type”: “string”
    },

    “name”: {
      “description”: “Account Name”,
      “type”: “string”
    },

    “email”: {
      “description”: “Account email”,
      “type”: “string”
    }
  },

  “required”: [“id”, “name”, “email”]
}
```

- The properties object contains the attributes (or fields) of the “Account” entity.
- Each attribute is represented as an json object with properties such as description, type, etc.
- Attributes like id, name, and email are marked as required fields.

Another crucial concept in the solution involves defining all objects within the system and providing detailed descriptions for each object. This comprehensive approach ensures that every aspect of the system is accounted for and accessible for configuration. By defining objects, such as data tables, forms, or modules, developers create a structured representation of the system’s components. Each object is accompanied by a detailed description outlining its purpose, functionality, and associated attributes. This descriptive approach serves as a reference point for configuring the system and entering criteria.

Overall, the concept of defining and describing all objects within the system ensures transparency, accessibility, and ease of use during the configuration process. By establishing a comprehensive understanding of the system’s components, developers and users alike can effectively configure criteria to meet specific requirements and objectives.

## 5. Configuration Structure

The configuration structure defines a framework for specifying criteria in a structured manner. Here’s a breakdown of its components.

```
Structure
{
  “criteria_name”: “”,
  “object”: “<entity_name>”,
  “criteria”: {
    “1”: {
      “field_name”: “<field_name>”,
     “operator”: “<equal | contains | greater | many more>”
```

```
        },
        "2": {
          "field_name": "<field_name>",
          "operator": "<equal | contains | greater | many more>"
        },
      },
      "advanced": " 1 or 2 and 3"
    }
```

**criteria_name**: This field represents a descriptive name for the criteria being defined. It serves as a label or identifier for the set of conditions.

**object**: This attribute specifies the entity (or object) to which the criteria apply. It identifies the context within which the criteria are evaluated.

**criteria**: This section contains a set of conditions represented as key-value pairs. Each condition is numbered (e.g., "1", "2", etc.) and consists of:

**field_name**: Indicates the attribute or field within the specified object against which the condition is evaluated.

**operator**: Specifies the comparison operation used in the condition (e.g., "equal", "contains", "greater", etc.).

**advanced**: This field provides an optional expression for combining multiple conditions using logical operators (e.g., "and", "or", etc.). It allows for the creation of more complex criteria by chaining together simpler conditions.

## 6. Runtime

At runtime, the system retrieves the criteria name and dynamically generates a query to fetch the relevant data. The query generated may vary depending on the specific system or context in which the data retrieval is performed. This dynamic query generation ensures that the data fetched meets the criteria specified by the user or application.

Once the query is generated, the system caches it to avoid the need for repeated evaluation. Caching the query improves performance by eliminating the overhead associated with regenerating the query for subsequent data retrieval operations. Instead, the cached query can be quickly accessed and executed whenever the same criteria need to be applied again.

By caching the generated queries, the system optimizes resource utilization and response times, enhancing overall efficiency. This approach ensures that data retrieval operations are performed swiftly and seamlessly, contributing to a more responsive and scalable system architecture.

## 7. Uses

Utilizing the dynamic rules engine enhances versatility across various contexts, owing to its inherent dynamism. This feature allows for widespread applicability and adaptability, optimizing system functionality and flexibility. The dynamic nature of the rules engine ensures seamless integration and efficient utilization across diverse environments.

## 8. Conclusion

Optimizing n conclusion, the dynamic rules engine emerges as a pivotal tool in modern software development, offering unparalleled flexibility and adaptability. By enabling the dynamic generation of queries and criteria, this engine facilitates efficient data retrieval tailored to specific system contexts. Additionally, the integration of a caching mechanism enhances performance by reducing query evaluation overhead and optimizing resource utilization. Through the combined capabilities of dynamic query generation and caching, software systems can achieve heightened responsiveness and scalability, catering to evolving user needs and system demands. The widespread applicability of the dynamic rules engine underscores its significance in diverse domains, from backend data processing to frontend user interactions. Moving forward, continued research and development in this field promise to further enhance the capabilities and effectiveness of dynamic rules engines, driving innovation and efficiency in software engineering practices. Overall, the dynamic rules engine stands as a cornerstone of modern software architecture, empowering developers to create robust, adaptable systems capable of meeting the ever-changing demands of today's digital landscape..

## 10. References

1. Huang Bin, He Zheyuan, Tang You. Application research of business rules engine management system based on drools. Cyber Security Intelligence and Analytics, 2020; 1146.
2. Naser Nrkarami, Junichi Iijima. A Logical Approach for Implementing Dynamic Business Rules. Management Information Systems, 2010; 6: 29-52.
3. https://www.academia.edu/21195403/ACTIVE_RULE_ENGINE_FOR_DYNAMIC_BUSINESS_RULES
4. Bock Alexander, Frank Ulrich. Low-Code Platform. Business & Information Systems Engineering, 2021; 63: 733-740.
5. Woo Marcus. The Rise of No/Low Code Software Development-No Experience Needed?. Engineering, 2020; 6: 960-961.