Integrating Syntax Highlighting into a Digital NoSQL Database Teaching Tool

Vanessa Meyer[1], Jaison John Palakat[1] and Lena Wiese[1,2]

[1]Institute of Computer Science Goethe University Frankfurt, Robert-Mayer-Str.10, Frankfurt am Main, 60325, Germany
[2]Fraunhofer Institute for Toxicology and Experimental Medicine, Nikolai-Fuchs-Str.1, Hannover, 30625, Germany

Abstract
We present our NoSQLconcepts tool as a digital learning platform with a focus on a learner-centered approach in database education, where learners can control their educational journey, accessing the provided educational content and four databases under one common interface and engaging in interactive activities at their own pace. Our contribution is providing more technical details on the implementation – in particular, details on the implementation of syntax highlighting for the different database query languages MongoDB Query Language (MQL), Neo4J's Cypher, and Cassandra Query Language (CQL).

Keywords
NoSQL Databases, Learning Tool, Learning Analytics

1. Introduction

In today's digital learning landscape, online learning platforms play an increasingly important role in delivering educational content. The ability to convey complex concepts and skills in digital environments has the potential to enhance and expand the learning process for a variety of learners. Studies show that e-learning has become quite popular across students. More and more educational institutions are switching to web-based learning. Particularly following the COVID-19 outbreak in 2019, billions of individuals were facing a massive lockdown, which forced the educational institutions to turn to e-learning platforms to ensure continuity in the educational process [1]. There are several aspects which need to be considered regarding a well programmed e-learning platform.

Contributions. Building on our previous work [2], in which we introduced our database teaching platform called *NoSQLconcepts*, in this paper we will demonstrate additional features of the tool that are designed to further enhance the database learning experience.

Our *NoSQLconcepts* tool is used within a practical course at Goethe University to allow students to work on different exercises on four different *NoSQL* (Not only SQL) databases [3] in a unified environment.

LWDA'24: Lernen, Wissen, Daten, Analysen. October 23–25, 2024, Würzbzrg, Germany
v.meyer@em.uni-frankfurt.de (V. Meyer); s9941926@stud.uni-frankfurt.de (J. J. Palakat); lwiese@cs.uni-frankfurt.de (L. Wiese)
0000-0003-3515-9209 (L. Wiese)

The original starting point of the practical course was as follows [4]: Students used a template (excel or csv file) in which they had to enter their solutions manually. In two-week phases, students worked on an exercise sheet with the respective database. To execute database queries, a separate environment was used for each of the databases, such as *pgAdmin* for PostgreSQL.

With the unified environment including the connection to four *NoSQL* databases, as well as components like for example the digital exercise sheets, and a learning analytics dashboard, we already introduced an approach to improve teaching and learning of *NoSQL* databases with the help of a digital learning tool in our previous work [2]. Among the innovations of the *NoSQLconcepts* tool that are currently in the development phase are syntax highlighting for database queries, and additional automated feedback.

Outline. In the next section, we provide an overview of the structure and used technologies of the *NoSQLconcepts* tool, as well as an overview of the frontend components and the backend of the *NoSQLconcepts* tool, which is a react web application. In addition to the components that we have already presented in our previous work [2], we also describe other components that we have integrated into our *NoSQLconcepts* learning tool. Section 3 then shows the modifications and improvements. These include the components currently in the development phase like the syntax highlighting for each database query language and the automated feedback. The accessibility improvements are also discussed here. In Section 4 we survey related work. Finally, Section 5 presents the conclusion and an outlook for the future.

2. NoSQLconcepts Overview

This section briefly describes the technologies used, which we have already described in more detail in our previous work [2]. We opted for React[1] for the development of the front end. React is a framework that is easy to use and implement and, among other things, it allows you to create a visually appealing interface [5]. In addition, React uses a virtual DOM, which increases the overall efficiency. Furthermore, external dependencies can be easily installed with the *Node Package Manager*[2] *(NPM)* [6]. The so-called *MERN stack* consists of a *MongoDB* database, *Express*[3], *React* and *Node*[4], which are all open source technologies based on JavaScript [7]. The react components of the *NoSQLconcepts* tool are presented in subsection 2.1. The *MERN* stack is extended in our tool. In addition to the *MongoDB* database, we also integrate the *PostgreSQL*, *Cassandra* and *Neo4J* databases with the database drivers provided for this purpose, which can be seen in Figure 1. We use *Axios* to manage *HTTP* requests to external resources.

2.1. Frontend Components

The main structure of the web application consists of a top bar, the content, and a footer. The top bar has menu items such as *Dashboard*, *History*, *Your data*, and *Admin* (in case that the user is an admin). In the content area the main components of the web application are displayed.

[1]https://react.dev/
[2]https://www.npmjs.com/
[3]http://expressjs.com
[4]http://nodejs.org

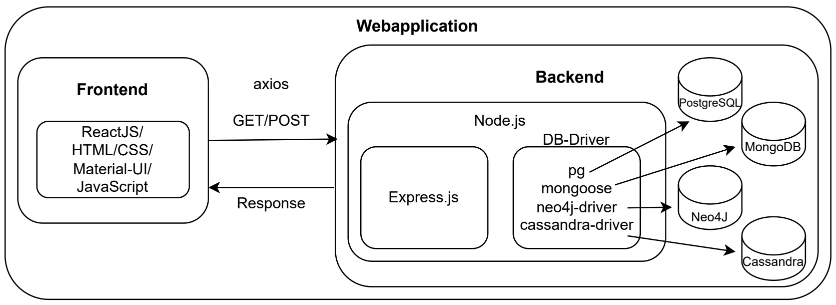


Figure 1: Diagram of the architecture of the web application, introduced in our previous work [2].

The footer is located at the bottom and contains logos of the institutions which are involved in the *NoSQLconcepts* project. The menu items and main components are further described in the following paragraphs.

Main Components. First, we will briefly describe the main components of the web application. The learning analytics dashboard consists of a grid with various contents. First, dashboard cards are displayed for each database (PostgreSQL, Cassandra, Neo4J and MongoDB), as well as for the two computer lab sessions (in *Lab Assignment 1* SQL tasks are tested on site and in *Lab Assignment 2* Cypher tasks), which contain the respective title, a progress circle and a start button to start the exercises.

For individual progress and status reporting, below the dashboard cards are bar charts that show the individual time required for tasks and the number of correct and executable tasks that have already been started. In addition, the current number of active users of the application is shown, as well as a small rating that shows the tasks which were rated as the easiest and the most difficult task. Through a button, users are navigated to a dashboard with course statistics. Here, a line chart shows the number of queries executed for the last 7, 14 or 21 active days.

The dashboard with overall course statistics shows bar charts for the time required for tasks and also contains the number of started, correct and executable tasks with the average values based on all active users of the web application. Pie charts are also used to visualize the overall distribution of perceived difficulty levels for tasks in the respective database.

The database-specific digital task sheets are structured as described below. For each of the databases the same task sheet component is used. The task sheet is a multi-page form. Each page of the form has a link to the download area, where the solutions entered so far can be downloaded as an excel file. In addition, a page can be selected via a select dropdown element and the corresponding task can be jumped to. Accordion elements of the Material-UI[5] library allow the respective database structure to be inspected at any time while working on tasks. This means, for example, that users can take a closer look at the stored database tables (in the case of PostgreSQL) and their attributes. The description of the task is structured in such a way that it contains a general use case title, including a subtitle and a specified maximum processing time. This is followed by a more detailed description of the task. Another button starts a timer

[5]https://mui.com/material-ui/

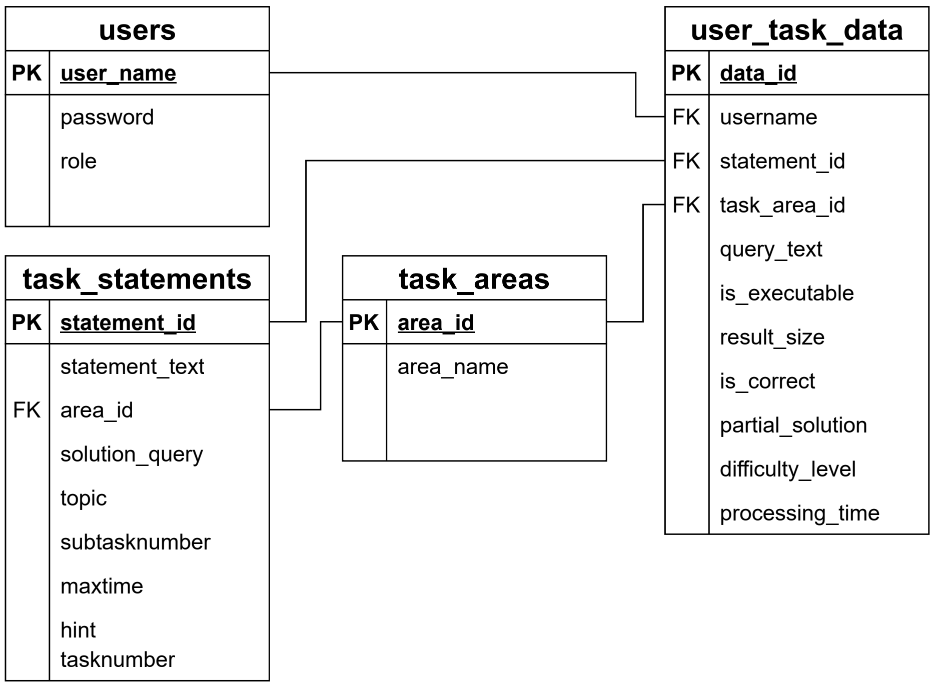


Figure 2: Current structure of the PostgreSQL database used for user data storage, based on our previous work [2].

to automatically measure the time required for the respective task. In addition, the editors, text fields and radio button groups are visible when starting a task, in which users can enter their queries, solution approaches or comments, and the ratings in terms of correctness and perceived level of difficulty. Additional buttons can be used to navigate to the next or previous page of the form.

2.2. Database Backend

To fill the charts of the learning analytics dashboard with data, to create statistics and give users the opportunity to retrieve their own data, the user input data must be saved. To do this, we use a PostgreSQL database with the tables and associated attributes shown in Figure 2. Compared to our paper [2], the table *task_statements* has been extended by several attributes. This table now contains data on the exercises, such as the maximum time that students should spend on an exercise. The maximum time can be adjusted by the course instructor (admin) within the *Exercise Management* component. In addition to the maximum time and the description of the exercise, the admin can also specify a topic, a subtask, or hints for an exercise, which are also stored in the database and retrieved for display within the digital exercise sheets.

3. Modifications and Improvements

As we continue to improve our learning tool, in this section we take a closer look at modifications and improvements that were not yet covered in our previous work [2].

At first, we will briefly discuss improvements to the accessibility of our learning tool. Then we describe an initial approach to the automated feedback function which is used to evaluate the solutions of exercises. We also present the future plans for an improved feedback. Finally, we describe an improved approach for implementing syntax highlighting of database queries in detail. The functionalities described in this section are work in progress.

3.1. Accessibility / Web Disability

In our last paper [2], we already addressed some aspects of diversity and accessibility in our learning tool. In summary, the aspects we dealt with include scenario descriptions in gender-neutral and simple language, the use of clearly structured, legibly and comprehensibly labeled elements, and the use of high-contrast colors.

We will now look at a new function that further improves the accessibility of the learning tool. This function has already been mentioned in Section 2.1 and is discussed again here in relation to accessibility. Within the digital exercise sheets, the underlying data model was displayed as an image next to the task description in our previous work [2]. Alternatively, as an extension we have now added the function to interactively inspect the respective data model. As already mentioned, we use Accordion elements of the MUI library for this purpose. For each table for PostgreSQL and Cassandra, each collection for MongoDB, and each set of nodes and edges for Neo4J, an element labeled with the respective name is created. Users can select any element and view further information such as attribute names with the respective data type. This provides better accessibility than simply displaying the data model as an image with alternative text. We are continuing to work on improvements for better accessibility.

3.2. Automated Feedback Component

To give students feedback on their input queries, feedback is given on executability or errors when the query is executed. A predefined query that generates the expected output is executed at the same time as the query entered by the user is executed. The output of the user's query is compared with this expected output. If the two outputs are the same, a corresponding success message is displayed to the user. This is an initial approach for evaluating the students' queries. However, since a different output than the specified one can also be correct, we plan to replace this function with AI generated feedback in the future.

3.3. Syntax Highlighting

With the key task of the digital learning platform being teaching and testing the students' database knowledge, a fundamental element is the representation and display of database queries. As students will enter queries as a task inside the digital platform it is pivotal for them to get instant feedback. The library which will be investigated in this paper is the React Ace Editor[6], an open-source editor that supports different kinds of languages including PostgreSQL.

Ace Editor. The Ace Editor is an open-source and standalone code editor which is written in JavaScript. The Code is published and accessible on GitHub[7]. The editor is used as the primary editor in Amazon's "Cloud9" IDE and offers language support for over 120 languages. Some of these are Python, JavaScript or TypeScript, but also PostgreSQL, which is one of the query languages that is being used in our tool. The Ace Editor can be extended with further custom languages or languages which are not yet supported by writing an own *mode* with own *rules*, which can then be applied to the editor.

At first sight it might make sense to integrate and deploy those syntax highlighting libraries which support the query languages natively that are used inside our digital learning tool. Then the Ace Editor could be for example used for PostgreSQL and another library for MongoDB and Cypher. But as the students will be performing all their queries on that digital learning tool, it is important to provide them with an appropriate and especially consistent user interface. Inconsistent syntax highlighting might result in confusion. Consequently, it is worthwhile to use one library for all the queries to ensure consistency across the editor for the different query languages. Furthermore, the technical complexity of the digital learning tool would increase with each additional library used for syntax highlighting. This is because knowledge of all used libraries for syntax highlighting would be necessary if any of those libraries needs to be maintained or extended. As PostgreSQL is the only query language for our digital learning tool that is natively supported by the Ace Editor, it is necessary to extend the language support by MongoDB, Cypher and Cassandra.

Implementation of a Mode for MongoDB As the functionality of syntax highlighting within our NoSQLconcepts tool is a major novel contribution, we will first of all describe our approach of implementing a mode for the AceEditor by an example for MongoDB.

Before starting to create a new mode for the ACE Editor, we have to download the source code from the Ace GitHub page[8]. After installing all the dependencies, the first step is to create two new files inside the ACE Editor project:

- mongodb.js
- mongodb_highlight_rules.js

These files are placed inside 'Ace/src/mode/' where other modes and highlighting rules are persisted. The two files have two different purposes: the mode file (mongodb.js file) contains the path to the language's rules, like syntax highlighting or indentation rules. In mongodb_highlight_rules.js the highlighting rules are defined, which are supposed to be applied when writing a query or code inside the editor.

Starting with the mode, Ace recommends extending an existing one and provides a template[9] for a minimal mode. Generally, it is also possible to look at any other mode that is already

[6]https://github.com/securingsincity/react-ace
[7]https://github.com/ajaxorg/ace
[8]https://github.com/ajaxorg/ace
[9]https://github.com/ajaxorg/ace/wiki/Creating-or-Extending-an-Edit-Mode

implemented inside the Ace Editor. With using the template as the base, we created a *mongodb.js* file that is adjusted with the new highlighting rule.

Implementing Highlighting Rules The Ace Editor uses rules to append tokens to highlight specific structures, functions or keywords. These rules are defined through regular expression. That means, if a pattern is matched with a rule's expression, the respective token will be appended and the matched pattern will be highlighted depending on the appended token. The order of the rules is important too, as the rule with the first matching expression will be executed. Additionally, Ace uses states to differentiate between different contexts. Depending on the context, the current state dictates the rules, which can possibly be applied.

A rule is composed of mainly three parameters:

- **Token**: This can be a string, an array or a function. The token represents the CSS token to apply.
- **Regex**: This is either a string or a regular expression. It represents the expression that needs to be matched for the token to be applied and the state to be changed.
- **Next**: This parameter is a string and is optional. It directs to the next state to enter after the token is applied.

Additional rules can be defined inside the state or additional states can be defined after the definition of the first state. Ace requires an initial *start*-state from which the state machine will begin.

Defining States. Let us consider the explicit situation for block commenting in MongoDB. A block-comment starts with '/*' and ends with '*/'. Everything in between needs to be a comment. The rules defined inside the start-state should not be applicable to anything between a block comment. While it is possible to solve it with regular expression, Ace offers states, allowing the context to be switched and only context-dependent rules to be applied. The transition is visualized in Figure 3.

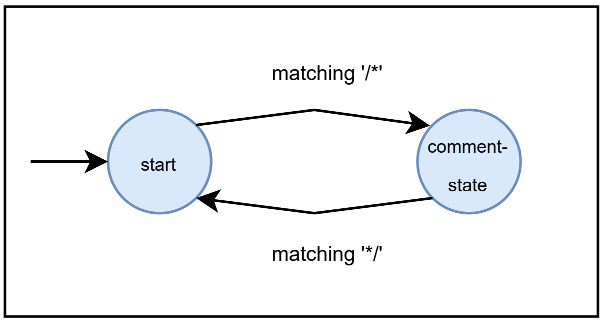
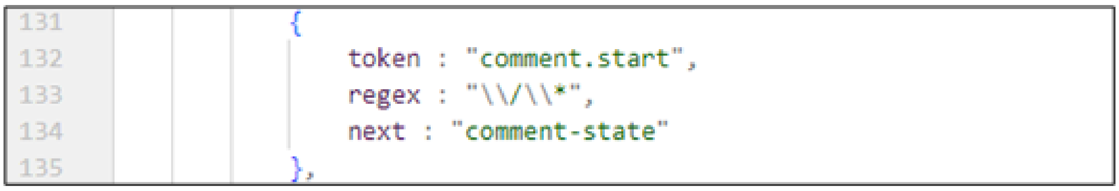


Figure 3: State Change for Block Comment for MongoDB.

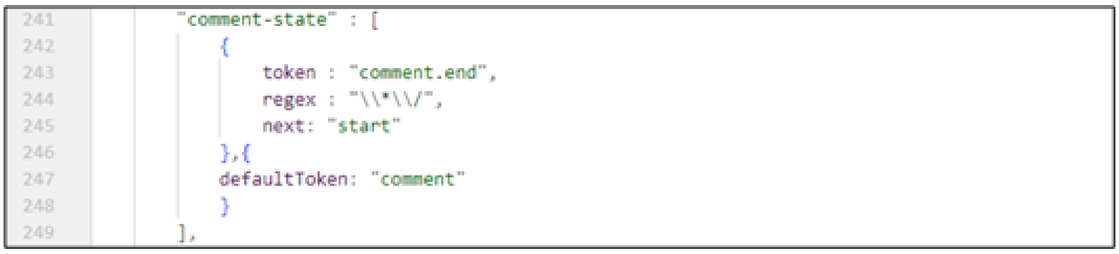
Beginning inside the start-state, we have all the rules for highlighting strings, digits and single line comments. Nonetheless we want the block comment to be only highlighted as a comment and therefore want to enter the comment-state when '/*' is matched and want to return to the start-state when we encounter '*/'. With that in mind, the start-state include the rule shown in Figure 4



```
{
    token : "comment.start",
    regex : "\\/\\*",
    next : "comment-state"
},
```

Figure 4: Rule for Block Comment Start.

When observing the rule, one can see that the next state is defined with the name *comment-state*. That state needs to be implemented with sets of rules, that is going to be applied after the block-comment is starting.



```
"comment-state" : [
    {
        token : "comment.end",
        regex : "\\*\\/",
        next: "start"
    },{
    defaultToken: "comment"
    }
],
```

Figure 5: Comment State for MongoDB.

Figure 5 shows the comment-state, which at first sight only contains one rule, which corresponds to the pattern '*/' and marks the end of the comment. When this pattern is matched, the next state is *start*. That ensures, that all the rules defined in the start-state can then be applied again. But the comment-state also includes another rule, which is not defined with a regular expression. By passing a token value for *defaultToken*, every pattern, that does not match a regular expression defined inside the state, will be applied with that token. That helps to highlight everything between '/*' and '*/' in another style. In this case every other pattern than '*/' will be highlighted in the comment style. This shows the capabilities of context-based highlighting inside the Ace Editor.

Now each language has its own set of reserved words like keywords or method names. For MongoDB, these words can be looked up on their official website[10]. The idea stays the same: When the word is matched, apply a token, and highlight it accordingly. To highlight them, the TextMode, which our class is inheriting from, offers a function called *createKeywordMapper*, which enables us to map multiple keywords that are persisted as a string and separated by a vertical bar with a token. This mapping can then be called inside a rule to mark them accordingly.

[10]https://www.mongodb.com/docs/manual/crud/

Implementation of a Mode for Neo4J The next step was to create an Ace mode for Neo4J's language Cypher. Implementing syntax highlighting for Cypher takes a similar approach as for MongoDB. First, the two files (cypher.js, cypher_highlight_rules.js) are created. The mode is similarly built like the mode for MongoDB but this time pointing to the cypher highlighting rules file. In that file, the states and the rules are defined, according to which the structures and words are highlighted. When looking at the official documentation[11] of Neo4j's Cypher, they mention the three core entities a graph database is consisting of:

- Nodes: These are data entities and are referred to using round parenthesis.
- Relationships: These are the representation of connections between the nodes and they must be enclosed by square brackets.
- Paths: These are the combination of the previously mentioned two entities.

They also mention that Nodes, in turn, contain labels, that serve as tags, a name property defined in Cypher by wrapping it in curly brackets and a variable that can be used to reference specific nodes in subsequent clauses. Queries in Cypher begin with a clause keyword like 'MATCH', 'CREATE' or other clause keywords, which can be found on their documentation website[12].

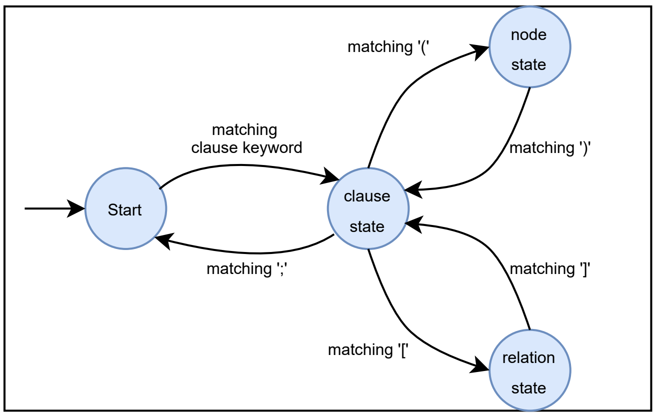


Figure 6: State Machine for Cypher.

Figure 6 represents the state machine of the implemented syntax highlighting rule. First, the machine begins inside the start state, where only constants, comments and numbers are highlighted. The state transition happens when one of the clause keywords is matched. The clause state is the biggest state, as it contains the most rules out of the defined states. It highlights further clause and subclause keywords, different operators and constants. The machine will only switch to the node state, when an opening parenthesis is found, which marks the beginning of a context switch to data entity. In there, labels are being highlighted as expected. The machine returns when a closing parenthesis is matched. To move to the relation state, an opening square

[11]https://neo4j.com/docs/cypher-manual/current/queries/concepts/
[12]https://neo4j.com/docs/cypher-manual/current/syntax/reserved/

bracket is necessary as it indicates the beginning of a relationship pattern. The state highlights the relationship type. Returning to the clause state is done when a closing square bracket is matched. As Cypher query statements are ending with a semicolon, the state machine will switch back to the start state when it is detected. Figure 6 shows the important states of the implementation. There are further states implemented like the block commenting state, which marks everything between '/*' and '*/' as a comment. That state is reachable from every state and will return to the previous state when the ending pattern for the block comment is found.

Implementation of a Mode for CQL Last but not least we had to implement an Ace mode for Cassandra's language CQL. For creating the syntax highlighting for CQL, two files are created for the new mode (cql.js, cql_highlight_rules.js). The first file (mode file) contains the path to the highlighting rules and the second file (highlight rules) contain the rules and definition to highlight CQL queries. Cassandra offers an introduction[13] that describes how a statement is formally built. Especially the data definition and data manipulation queries will be interesting for our tool, as they provide actions for creating, modifying, removing tables, as well as inserting, updating, deleting and querying data.

Figure 7 illustrates the implemented state machine for the CQL mode. The machine starts inside the start state. In there, different rules are defined to switch the context. The semicolon defines the end of a statement. Therefore, all states, that are directly reachable from the start state, will return to it again when a semicolon is matched. The keywords which trigger a context switch are: 'SELECT', 'CREATE TABLE', 'ALTER TABLE', 'UPDATE', 'CREATE KEYSPACE', 'ALTER KEYSPACE', 'DELETE' and 'INSERT INTO'. As CQL is case insensitive for keywords, it does not matter if they are written in lower- or uppercase. Every state has its own context-dependent keyword list. As some words are not context-dependent, they will always be highlighted independent of the current state. Mentionable is the batch state, which is not displayed in the figure, just to keep the graphic clear. The state belongs to a batch-statement which is used to execute multiple insert, update and delete statements in one. The batch state points therefore to the insert, update and delete state and will only return to the start state if the statement ends with an 'APPLY BATCH;', like it is provided on Cassandra's documentation. Also, when observing the figure, it is visible that there is an embedding of the JSON language. When creating an insert statement, it is possible to provide information in a JSON format. The beginning for the JSON format is indicated with a "JSON ' " and it is only highlighted given the state machine's current position is the insert state. It is important to mention that every state also shares some rules like comments and datatypes and block comments. All those rules are represented in every state, just like the context independent reserved keywords. The state machine can easily be extended with further rules, as every statement has its own state.

4. Related Work

As we see in recent studies there are aspects and factors for digital learning platforms, which do have an impact on the user satisfaction, especially if we investigate usability. So, it is important to have a guideline regarding a good usability factor to adhere to, if a digital learning

[13]https://cassandra.apache.org/doc/latest/cassandra/developing/cql/ddl.html

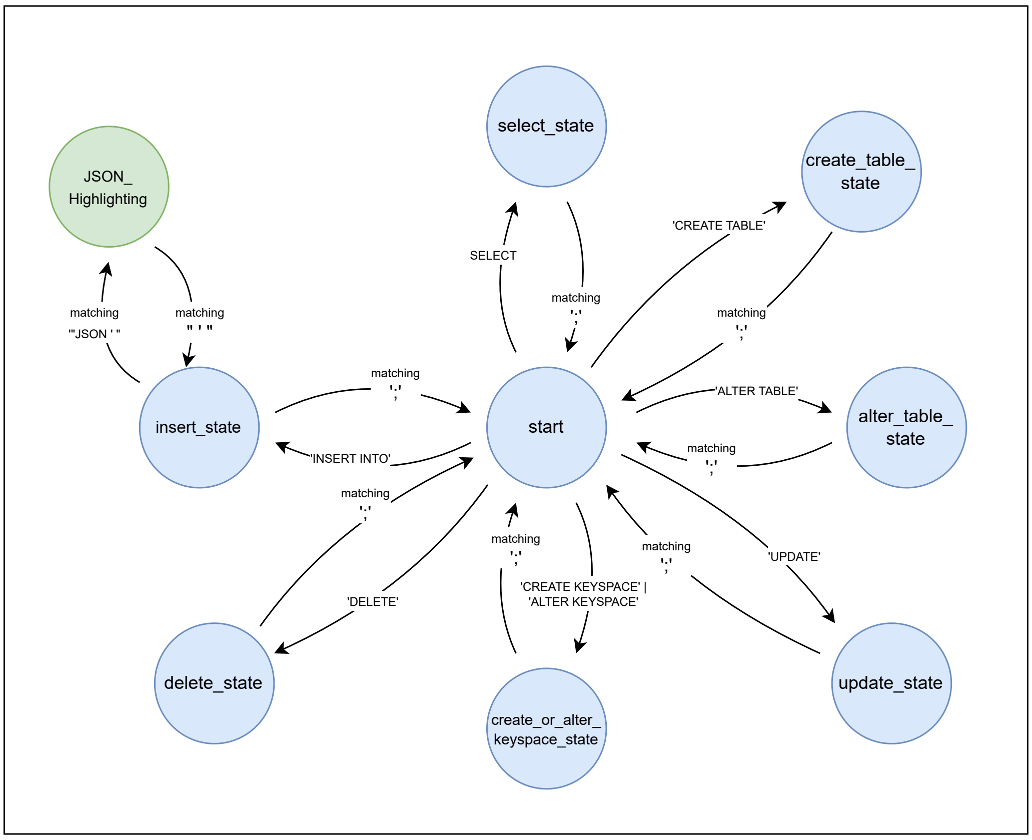


Figure 7: State Machine for CQL.

platform needs to be implemented in regard of the user satisfaction and learning progress. Another characteristic for digital learning platforms is the instructional modalities that it enables. Instructors can provide asynchronous, synchronous and bichronous online modalities to transmit education. Presenting lectures or quizzes about a lecture in an on-demand digital format allows the students to consume the educational content from everywhere any time [8].

When considering the advantages of e-learning, in a recent study [9] most of the students mentioned time efficiency as the main advantage opposed to face-to-face learning. Some other advantages mentioned are convenience, accessibility, lack of need to travel, accessing courses anytime and flexibility. Additionally e-learning is credited with increasing student motivation, improving concentration levels, and offering higher rates of assignment submission compared to traditional face-to-face learning [9]. Despite these advantages, e-learning brings some challenges. The study identified several disadvantages associated with e-learning [10]. The most prominent disadvantage is the lack of interaction with peers and instructors. In regard of these considerations, a hybrid or blended learning approach is being proposed as a potential solution in education. This approach combines the benefits of both e-learning and face-to-face learning, offering flexibility and accessibility while having still the possibility of direct interaction and engagement [10].

One of the key aspects, which comes to mind when thinking about usability in a programming environment is syntax highlighting for an increase of usability. Syntax highlighting is an element, that can help developers understand and write code easier. According to different studies, syntax highlighting can significantly improve task completion time [11, 12]. Other studies concluded that colored code is easier to read and easier for identifying keywords [13, 14]. On the other hand, a study of 2018 came to the conclusion that syntax highlighting does not improve the ability to understand source code for programming novices [15]. Looking at most of today's Integrated Development Environments (IDEs) shows that syntax highlighting is a key feature for improving code comprehension and reducing task completion time [11]

5. Conclusion and Future Work

The particular significance of our e-learning tool NoSQLconcepts lies in the enhancement of database learning effectiveness by facilitating the learning process for students through User Interface improvements. The digital platform is different than having a person actively teaching a student, with the possibility of an in-person communication. The student will mostly have interactions with the platform and not the teacher and they will learn and test their knowledge digitally. Therefore, in scope of the digital platform, it is necessary that the platform is implemented with an intuitive, self-explaining UI, which makes the platform easier and more inviting to use for the students. In this article we focus on identifying and exploring available syntax highlighting libraries and tools for the respective database query languages. As students will enter queries as a task inside the digital platform it is pivotal for them to get instant feedback. Our study can provide an insight about the technical landscape for educators, developers and designers involved in digital learning development.

The automated feedback within the digital exercise sheets is still a work in progress. In the future we want to improve the feedback functionality to respond even better to the individual solutions of the students. This should also relieve the tutor of the practical course, who will spend less time on manual evaluation thanks to the automated feedback.

Acknowledgments This paper has been supported by Goethe University Frankfurt's Digital Teaching and Learning Lab (DigiTeLL) in the project NoSQLConcepts and is funded by Stiftung Innovation in der Hochschullehre.

References

[1] V. Gewin, Five tips for moving teaching online as covid-19 takes hold, Nature 580 (2020) 295–296.

[2] V. Meyer, L. Wiese, A.-G. Ahmed, A unified teaching platform for (no)sql databases, presented at the ICEIS 2024 conference (2024).

[3] L. Wiese, Advanced data management: for SQL, NoSQL, cloud and distributed databases, Walter de Gruyter GmbH & Co KG, 2015.

[4] L. Wiese, A. Benabbas, G. Elmamooz, D. Nicklas, One db does not fit it all: Teaching the differences in advanced database systems, Datenbank-Spektrum 21 (2021) 83–89.

[5] C. Rajesh, K. Srikanth, Research on html5 in web development, Int. J. Comput. Sci. Inf. Technol 5 (2014) 2408–2412.

[6] P. Rawat, A. N. Mahajan, Reactjs: A modern web development framework, International Journal of Innovative Science and Research Technology 5 (2020) 698–702.

[7] S. Hoque, Full-Stack React Projects: Learn MERN stack development by building modern web apps using MongoDB, Express, React, and Node. js, Packt Publishing Ltd, 2020.

[8] M. Alenezi, Digital learning and digital institution in higher education, Education Sciences 13 (2023) 88.

[9] V. Gherheș, C. E. Stoian, M. A. Fărcașiu, M. Stanici, E-learning vs. face-to-face learning: Analyzing students' preferences and behaviors, Sustainability 13 (2021) 4381.

[10] J. H. Choi, H.-J. Lee, Facets of simplicity for the smartphone interface: A structural model, International Journal of Human-Computer Studies 70 (2012) 129–142.

[11] A. Sarkar, The impact of syntax colouring on program comprehension., in: PPIG, 2015, p. 8.

[12] G. M. Dimitri, The impact of syntax highlighting in sonic pi., in: PPIG, 2015, p. 9.

[13] T. R. Beelders, J.-P. L. du Plessis, Syntax highlighting as an influencing factor when reading and comprehending source code, Journal of Eye Movement Research 9 (2016).

[14] P. J. Khomokhoana, L. Nel, A framework to assist instructors help novice programmers to better comprehend source code – a decoding perspective, in: International Conference on Computational Science and Its Applications, Springer, 2023, pp. 677–693.

[15] C. Hannebauer, M. Hesenius, V. Gruhn, Does syntax highlighting help programming novices?, Empirical Software Engineering 23 (2018) 2795–2828.