3T: Triple Stores Triggers

Giannis Vassiliou[1], Georgia Eirini Trouli[1], Nikolaos Papadakis[1], and Haridimos Kondylakis[2]

[1] Hellenic Mediterranean University, Heraklion, Greece
{giannisvas,gtrouli,npapadak}@ics.forth.gr
[2] CSD-UOC & FORTH-ICS, Heraklion, Greece
kondylak@ics.forth.gr

Abstract. Database triggers have been introduced since the 1980s, and their value has been widely recognized. They allow for automating several tasks and enforcing business rules at the database level, ensuring data integrity, auditing, and consistent logic across applications. However, notwithstanding their high value, the use of triggers in mainstream triple stores has been mostly overlooked. The dynamic nature of semantic data, the increasing scale of RDF datasets, and the growing demand for real-time processing in graph-based applications demand efficient methods for real-time data validation and integrity enforcement, change tracking and auditing, and automated inference and reasoning. In this demonstration, we recognize triggers as first-class citizens in the semantic web and introduce a new interpreted language to describe them. We present 3T, an integrated development environment (IDE) and a graphical user interface (GUI) enabling triple-store-independent triggers, allowing their real-time, effective, and efficient execution.

Keywords: Triple Stores · RDF · Knowledge Graph · Triggers

1 Introduction

A database trigger is a procedural code that is automatically executed (or "triggered") in response to certain events on a database table or view. Triggers can be defined to run either before or after the triggering event, depending on the desired behavior. They are commonly used to enforce business rules, maintain data integrity, automate system tasks, handle real-time events, or perform auditing. Triggers have been around since the early days of relational databases, emerging in the 1980s as database management systems (DBMS) like Oracle, IBM DB2, and SQL Server started including procedural extensions.

Despite the tangible benefits, triggers have not been explored in triple stores and related RDF engines [1] primarily due to concerns about performance, as adding triggers can slow down systems that are optimized for fast querying and scalability, especially when dealing with large RDF datasets. While some vendors like Blazegraph ignore triggers altogether, others like Stardog offer rule-based alternatives such as SWRL, which require non-scalable reasoning engines and lack real-time responsiveness. Virtuoso advertises triggers, but they are confined

to underlying relational layers, inaccessible to RDF developers. Research efforts, particularly in property graphs [4, 2], have proposed frameworks like PG-Triggers to define and implement triggers, but practical integration remains challenging. Furthermore, the inherent flexibility and schemaless nature of RDF data also make it challenging to define and manage triggers effectively, since the relationships between triples can be dynamic and complex. Instead, triple stores have focused on optimizing query performance [3], prioritizing data integration [5], interoperability, and inferencing.

However, as RDF datasets grow in size and complexity and are used in a more dynamic landscape than ever, there is an urgent need for methods that help to efficiently ensure data integrity, enforce business rules, and automate workflows in real-time. Exploiting triggers could enable automated reasoning, immediate response to data changes, and effective management of dependencies across interconnected triples, making systems more responsive and efficient. Triggers also support tasks like auditing, data validation, and synchronizing with external systems, which are critical as RDF data is increasingly integrated into large-scale, event-driven architectures and applications requiring real-time insights.

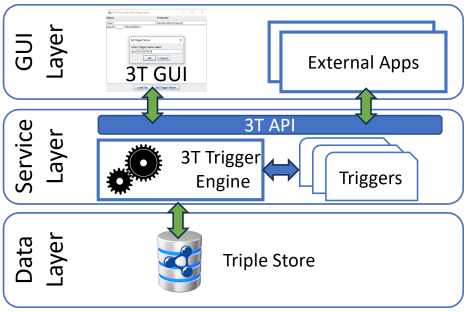
As such, in this paper, we argue that researchers and triple store vendors should capitalize on pre-existing knowledge from the relational world, and enrich the domain ofthe semantic web with the low-hanging fruit of triggers, upgrading them as first-class citizens in the modern world of triple stores. In this demonstration, we present 3T, a system unlocking the trigger functionality for the triple stores, that is triple-store agnostic and enables trigger declaration and execution in an efficient and effective manner. More specifically, our contributions in this paper are the following:

- We demonstrate 3T, the first system enabling the execution of triggers, in a triple-store agnostic method, introducing a new simple interpreted language for describing triggers.
- We also demonstrate an IDE front-end with a GUI allowing the user-friendly management of the stored triggers
- We also provide an API that can be used for interacting with the triple store, hiding the trigger layer and enabling users to build triple-store agnostic programs that are able to exploit the constructed trigger engine.

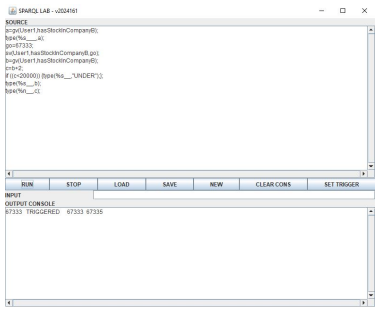
2 3T System Architecture

An overview of the 3T system is shown in Figure 1a and consists of three layers. In the data layer, we find the triple store, where the triples of the knowledge graph are stored. Note that our system is triple store agnostic and as such, it can be connected to any triple store available, upgrading it with triggers. In the service layer, we find the 3T trigger engine that responds to events based on the trigger conditions and executes the appropriate commands. All triggers are available in the main memory of the 3T Trigger Engine; however, for persistence, they are also stored in a text file. An API, the 3T API exposes the whole functionality to the GUI layer. As a proof of concept, we have implemented the 3T

GUI for recording, updating, and executing triggers from a command line. However, the same API can be used by other external applications for interacting with the triple store, allowing triggers to come into play when our API is used. In the sequel, we present in detail the various components of the system.



(a) 3T system architecture.



(b) Executing code that fires the Trigger

Fig. 1: Side-by-side view of system architecture and trigger code.

The 3T GUI functions similarly to an Integrated Development Environment (IDE) and is supported by a custom interpreted language. It is created in Java using ANTLR[3] and is shown in Figure 1b.

The key feature of the GUI is the ability to easily set triggers for a knowledge graph and execute custom code when these triggers are activated. Triggers are monitored based on specific subject-predicate combinations (could also be only one subject or only one predicate) that, when updated, a connected action (a set of code lines) is executed (for example, store a new value in the KG).

After editing their code, users can press the "RUN" button to execute it or use the "STOP" button to halt execution. Other features, such as clearing the console, are also available for ease of use. After pressing the "SET TRIGGER" button, the user is presented with a window where they can add an identifier to the trigger, whereas a text area is provided for writing the code that will be executed when the trigger is activated, with the option to load existing code from a file. Once the trigger is configured, the details are saved to a file, which is automatically loaded the next time the system starts, ensuring the trigger remains active until manually deleted.

When a trigger is set and activated, the associated code (written in our custom language) executes. This code can perform additional operations such as inserting, deleting, or updating triples in the KG. By offering a simple, GUI-driven interface, we provide an effective way to set triggers and automatically maintain the KG in the desired state. To showcase our system, we developed two examples to run by setting triggers in the 3T GUI.

[3] https://www.antlr.org/

Example 1. The first example is related to brokers and stocks. Assume for example, that a broker, "Broker A," can hold a number of stocks in "Company A." We can set a trigger to execute code whenever "Broker A" updates the number of stocks they hold in "Company A." This code could perform various operations supported by the interpreted language, such as updating the number of stocks "Broker A" holds in another company, like "Company B." To set this trigger, the user would select "Broker A" as the subject and "hasStocksInCompanyA" as the predicate, then write or load the associated code through the application's GUI. An example of relevant actual code is shown in Figure 1b. The first line reads (using the function "gv" that gets the value) the number of stocks that "User1" has in "CompanyB" in variable "a" whereas the fourth line updates the stocks of "User1" to 67333 (sets the value using the "sv" function). The seventh line includes a trigger that is fired when the condition is met, printing a string; however, updates on the data could be made here.

Example 2. Another illustrative example comes from the chemistry world. Consider the chemical reaction:

```
1 ACID + 1 BASE → 1 SALT + 1 WATER.
```

In the KG, we might assume the following triples:

```
(ACID, hasMolecules, 50);
(BASE, hasMolecules, 40);
(SALT, hasMolecules, 40);
```

We could define a trigger that fires when the number of molecules of ACID exceeds the number of molecules of BASE. When this condition is met, the trigger would execute code to simulate the chemical reaction, producing SALT and WATER, and leaving some remaining ACID (e.g., 10 moles). In this case, the triple *(SALT, hasMolecules, 40)* would be updated accordingly. For this chemistry example, the trigger is set to monitor the subject *ACID* and the predicate *hasMolecules*. Whenever this value changes, the trigger activates, ensuring the reaction adheres to the chemistry rules and updates the KG accordingly.

2.1 The Service Layer

Syntax In this layer, we find the 3T engine, which, in essence, for each incoming statement checks all triggers loaded in main memory and executes the body of the triggers. The syntax of the language that the engine supports is shown in Listing 1.1.

In our notation, upper-case letters are reserved for terminal symbols; nonterminals are in a lower case, alternatives are separated by the | symbol. The language includes many features like variable declarations, assignments, and arrays. Control flow structures like 'if-then-else' statements and 'for' and 'while' loops are fully supported, enabling conditional logic and iterative processes. Additionally, users can define functions, which can be invoked within the code or even call themselves recursively.

Listing 1.1: 3T Syntax

```
prog: block+
 | (COMMENT)* (function)* stat+;

function: FUNCTION ID '(' ID (',' ID)* ')' block

stat:
 COMMA # ecommand
 | SV '(' ID ',' ID ',' ID ')' COMMA # setvalue
 | COMMENT # comment
 | ID '(' expr (',' expr)* ')' COMMA # procedure
 | ID '=' expr COMMA # assign
 | ID '[' expr ']' '=' expr COMMA # assignarr
 | RETURN expr COMMA # retstat
 | PPRINT '(' printitem+ (',' expr)+ ')' COMMA
 | IF relexpr ifthen (ELSE ifthen)? # ifstat
 | FOR '(' ID expr TO expr ')' ifthen # forstat
 | WHILE relexpr 'DO' block # whilestat
 | ARRAY ID '[' expr ']' COMMA # array;

block: '{' stat+ '}' # blocking;

printitem: (STRPRINT | NUMBERPRINT | NEWL | SPAC);

ifthen: (block COMMA | stat) # blockstat;

expr: expr op=('*' | '/') expr # MulDiv
 | expr op=('+' | '-') expr # AddSub
 | ID '(' expr (',' expr)* ')' # functcall
 | FLOAT # float
 | INT # int
 | ID # id
 | ID '[' expr ']' # idvar
 | STRINGLITERAL # vstring
 | '(' expr ')' # parens
 | GV '(' ID ',' ID ')' # getvalue;

relexpr: '(' ('(' expr op=(EQ | GT | GTE | LT
 | LTE | DIF) expr ')' | relexpr)
 (opp=(LAND | LOR) ('(' expr op=(EQ
 | GT | GTE | LT | LTE | DIF) expr ')'
 | relexpr))* ')';
```

Semantics In our case, triggers are executed when an insert/update/delete operation is executed, explicitly mentioning the subject/object of the trigger condition. Then the body of the condition is executed. Our language is quite flexible to be able to declare multiple lines of code in the body of the trigger, including additional conditions etc. We next describe the trigger semantics along the classical dimensions.

Granularity. We assume that each trigger execution is linked to a subject and/or predicate that are monitored and that the trigger is fired as soon as relevant data is inserted, updated, or deleted through our API. For each incoming statement, all triggers are checked to identify whether relevant conditions are met in order to execute the trigger body. In the trigger body, all actions are executed as a set.

Action Time. Trigger execution of our engine occurs after the insertion, update, or deletion of information in the triple store, which is executed using our API. In essence, execution takes place after a successful commit and operates within an autonomous transaction.

Chained Execution. Triggers are checked in the order they have been inserted. Multiple triggers might be executed in the order they are found. So far, we have disabled recursive execution of triggers; however, the topic is well explored in relational databases by techniques limiting recursion that will be incorporated in the next version of the system.

API. Furthermore, the service layer includes an API that can be used in order to insert, update or delete triples from the connected triple store, allowing external apps to exploit the trigger mechanisms introduced by 3T.

2.2 The Data Layer

As already stated our system is triple-store independent as long as the triple store is supported by the Apache Jena API. As such it currently supports TDB and TDB2 native stores, Fuseki, Virtuoso, Stardog, and Blazegraph. Although other APIs can also be used for the time being we only support Apache Jena.

3 Demonstration & Conclusions

The purpose of the demonstration is primarily to highlight to technicians, researchers, and triple store providers the usefulness of triggers in RDF knowledge graphs and argue that we should leverage the established knowledge from the relational database domain, enhancing the semantic web sphere by integrating triggers, elevating them to be primary elements in contemporary triple store technology. The demonstration will proceed in four phases:

Phase 1. Triggers usefulness. The demonstration will start by explaining use-case scenarios from the financial and scientific domains, highlighting the usefulness of triggers.

Phase 2. The language. Then we will present trigger examples, explaining the constructs of the language and the flexibility it introduces. We will use the 3T GUI for visualizing existing triggers, updating them, and writing new ones, whereas we will show how they are fired when the information in the triple store is inserted, deleted, or updated.

Phase 3. The semantics. Then we will explain in detail the semantics of 3T triggers, explaining the granularity, the action time, and the chained execution we support.

Phase 4. "Hands-on". Conference participants will then be allowed to write their own triggers, and then perform update operations in the triple store in order to be executed when triggered. The result will be shown both from the 3T GUI and from the triple store.

4 Conclusions

This paper presents 3T, the first system enabling trigger declaration and execution over RDF triple stores. It facilitates real-time automation, data integrity, and business rule enforcement in knowledge graphs (KGs). 3T includes an interpreted trigger language, a user-friendly GUI-based IDE, and an API for integration with external applications, making it accessible to both technical and non-technical users.

Future work includes support for recursive triggers, inference-based triggers, integration of full SPARQL updates in trigger bodies, and extending 3T to real-time RDF streaming scenarios using platforms like RDF4J.

Acknowledgments. The work reported in this paper is implemented in the framework of H.F.R.I call "Basic research Financing (Horizontal support of all Sciences)" under the National Recovery and Resilience Plan "Greece 2.0" funded by the European Union – NextGenerationEU (H.F.R.I. TALE Project, Number: 16819).

References

1. Agathangelos, G., Troullinou, G., Kondylakis, H., Stefanidis, K., Plexousakis, D.: RDF query answering using apache spark: Review and assess-

ment. In: 34th IEEE International Conference on Data Engineering Workshops, ICDE Workshops 2018, Paris, France, April 16-20, 2018. pp. 54–59. IEEE Computer Society (2018). https://doi.org/10.1109/ICDEW.2018.00016, https://doi.org/10.1109/ICDEW.2018.00016

2. Bellomarini, L., Bernasconi, A., Ceri, S., Gagliardi, A., Magnanimi, D., Martinenghi, D.: Towards a standard for triggers in property graphs. In: Atzori, M., Ciaccia, P., Ceci, M., Mandreoli, F., Malerba, D., Sanguinetti, M., Pellicani, A., Motta, F. (eds.) Proceedings of the 32nd Symposium of Advanced Database Systems, Villasimius, Italy, June 23rd to 26th, 2024. CEUR Workshop Proceedings, vol. 3741, pp. 70–79. CEUR-WS.org (2024), https://ceur-ws.org/Vol-3741/paper02.pdf
3. Bonifati, A., Dumbrava, S., Kondylakis, H., Troullinou, G., Vassiliou, G.: Progressive querying on knowledge graphs. In: Simitsis, A., Kemme, B., Queralt, A., Romero, O., Jovanovic, P. (eds.) Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025. pp. 106–118. OpenProceedings.org (2025). https://doi.org/10.48786/EDBT.2025.09, https://doi.org/10.48786/edbt.2025.09
4. Ceri, S., Bernasconi, A., Gagliardi, A., Martinenghi, D., Bellomarini, L., Magnanimi, D.: Pg-triggers: Triggers for property graphs. In: Barceló, P., Sánchez-Pi, N., Meliou, A., Sudarshan, S. (eds.) Companion of the 2024 International Conference on Management of Data, SIGMOD/PODS 2024, Santiago, Chile, June 9-15, 2024. pp. 373–385. ACM (2024). https://doi.org/10.1145/3626246.3653386, https://doi.org/10.1145/3626246.3653386
5. Kondylakis, H., Plexousakis, D.: Exelixis: evolving ontology-based data integration system. In: Sellis, T.K., Miller, R.J., Kementsietsidis, A., Velegrakis, Y. (eds.) Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011. pp. 1283–1286. ACM (2011). https://doi.org/10.1145/1989323.1989477, https://doi.org/10.1145/1989323.1989477