Context Store Feature

Context Store Management

The feature "context" can be added for management context information.

This means, it is possible to store or cache events without using the blocking window operator. First of all, this is useful for slowly changing events (like states in a smart home or in a factory), because in a window-operator you have to wait until the state changes a second time before the first change gets available. Therefore, the context store is a special concept to avoid such a blocking behavior. For each context event (each state), there is a context store needed. This means, for example, that there is one context store for each door in a smart home. After creating a context store, the system can use the store-operator to write into the store and can use the enrich-operator to read from the store. Furthermore, there is a RCP-view to show the actual content of the context store. All these parts work as follows.

Context Store - create and drop

Creating and removing of context stores is done by a Continuous Query Language (CQL)-Statement. To create a new store, you may use the following statement (which is similar to the create stream statement):

```
CREATE CONTEXT STORE <name> (<schema>) AS <type>
```

the variables are used as follows:

- <name> is the name of the store and has to be unique.
- <schema> is a list of attributes and datatypes that describes which schema the context store provides. its a comma-separated list of <attribute-name> <datatype> (like for create stream).
- <type> indicates the type of the store. you can either use
    - SINGLE : this is a store where only one single element is hold (each new state replaces the old one).
    - MULTI SIZE <integer> : this is a store that holds <integer> events. if a new event arrives, then the previous event gets the startimestamp as its new endtimestamp (thus, SINGLE ist equal to MULTI SIZE 1)

An example:

```
CREATE CONTEXT STORE door (timestamp LONG, state DOUBLE) AS MULTI SIZE 10
```

This query creates a context store with name "door" and a schema (long, double). Further, the store holds an amount of 10 events.

You can drop the context store via the query:

```
DROP CONTEXT STORE door IF EXISTS
```

It is possible to omit "IF EXISTS", because it only prevents for throwing an exception if the context store that has to be dropped does not exists.

STORE - writing into a store

The store operator is used for storing data. If you want to store the contents of "inputoperator" into the context store named "door4", the store-operator is used as follows:

```
storing =  STORE({store = 'door4'}, inputoperator)
```

so, you only have to define the name of the store. Notice, that the output-schema of inputoperator has to be union-compatible with the schema of the definied store. Union-compatible means: only the datatypes have to be equal, but not the names of the attributes.

ENRICH - reading from a store

The enrich operator is used for reading data. The enrich has technically two inputs: one for the data that should be enriched and one for the data from the context store. So, if a new event from the normal input gets into the enrich operator, it queries all temporally correlating context information from the context storage and merges it to the incoming event. Since it may happen that the there are two or more context states in the context storage that have to be merged, the enrich could produce also two or more enriched events.

To specify the enrich operator, you can use the simplest form in Procedural Query Language (PQL):

```
enriching = CONTEXTENRICH({store = 'door5'}, inputoperator)
```

As the definition of the store-operator, you only have to define the store you want to read from. In this case, the complete tuple from the store is attached to the current tuple in the context-enrich-operator. Thus, the resulting outputschema is the concatenation of the input-schema of the enrich (or rather the stream that should be enriched by the operator) plus the outputschema of the context store.

Since, the tuple in the context store could have a lot of attributes and you only want to use a subset of them, you can also (optional) specify a list of attributes that are used for the enrichment:

```
enriching = CONTEXTENRICH({store = 'door5', attributes=['state']}, inputoperator)
```

So, in this example, we only get the attribute "state" from the context store (this means that the enrich projects other attributes away from the context-tuple from the context store before it attaches it to the "normal" incoming tuple).

As mentioned before, the enrich only produces results if there is at least one context state in the context store that temporally correlates with the incoming event (which means: they have overlapping timeintervals). In this case, it may happen that the event that has to be enrich is to old and there is no suitable context state, e.g.: the event a is valid at [100;120) and the context state c is valid at [200;oo). Thus, the event a cannot be enriched because it is too old. If you still want to use such a event but without an enriched context state, you can use the outer-option:

```
enriching = CONTEXTENRICH({store = 'door5', attributes=['state'], outer='true'}, inputoperator)
```

if the outer-option is set to "true", the enrich will enrich events with "null"-values instead of discarding them (thus, its like an outer join).

MAP

Together with the Map operator it is possible to retrieve single values from the context store with the function `ContextStore(storeName)`

```
out = MAP({Expressions=["elementAt(ContextStore('door5'),0)","input.x"],input}
```

Remark: The ContextStore functions delivers a tuple in this case and elementAt reads the attribute from the tuple. Else the whole tuple will be added.

View - show the context of stores

if you use the rcp-view, you can use the "Context Store" view. This can be found under the menu windows -> show view -> other... -> odysseus.

The window should look like the following image:

In the upper part, all stores are listed. you can open each store to show its schema and further information like its size. The lower part shows the current content (which is the current valid context state of the store) of the store that is selected in the upper part of the view. so, you have to select,e.g. "door5", if you want to see the current content of the store "door5".