SMO_C

Dominique Hausler
University of Regensburg
Data Engineering Group, Faculty of Informatics and Data Science
dominique.hausler@ur.de

September 4, 2023

1 SMO_C for entity types and features

```
gDBelements ::= entityType | feature
entityType ::= nodeEntity | relation
nodeEntity ::= node | nodeWithRels
nodeWithRels ::= connectedNode | multiConnectedNode
feature ::= nodeLabel | relationType | map

relation ::= "[" (relVar | relType | relationCombined) (filter)? "]"
relVar ::= variable
relType ::= relationType (map)? ("," relationType (map)?)*
relationCombined ::= relVar relType

node ::= "(" (nodeVar | nodeLabeled | nodeCombined) (filter)? ")"
nodeVar ::= variable
nodeLabeled ::= nodeLabel (map)? ("," nodeLabel (map)?)*
nodeCombined ::= nodeVar nodeLabeled

(*due to pattern complexity nodeVar and relVar*) can hold one or multiple entity types
    *)

filter ::= WHERE (NOT)? (((combinedNode | relationCombined | function)
    comparisonOperator value) | properties) (booleanOperator (NOT)? (((combinedNode |
    relationCombined | function) comparisonOperator value) | properties))*

connectedNode ::= node ("-" | "←") relation ("-" | "→") node
multiConntectedNode = connectedNode (("-" | "←") relation ("-" | "→") node)*

(*select a pattern*)
select ::= selectPattern | selectSubgraph
selectPattern ::= MATCH nodeEntity ("," nodeEntity)* (OPTIONAL MATCH nodeEntity)*
selectSubgraph ::= CALL APOC.PATH.SUBGRAPHALL "(" nodeVar (*of start node*) "{"
    extendedFilter "}" ")" YIELD NODES "," RELATIONSHIPS
(*all available filterKeys and the valueTypes for each of them can be found in [6] *)
extendedFilter ::= filterKey ":" valueType

alias* ::= AS variable
collect* ::= COLLECT "(" (variable | function | ("[" (nodeVar | relVar | collection) ("
    "," (nodeVar | relVar | collection))* "]" )) ")"
function = (*for a concrete implementation add the function that fits the problem to
    solve best - see [1]*)
```

```
property ::= variable "." propertyName ("=" value)?
map ::= "{" propertyName ":" value ("," propertyName ":" value)* "}"
properties ::= "{" property ("," property)+ "}"
nodeLabel ::= (":" String)+
relationType ::= (":" String)+
nodeVar, relVar ::= Char | String
booleanOperator ::= AND | OR | XOR | NOT
comparisonOperator ::= "=" | "<>" | "<" | ">" | "<=" | ">=" | IS NULL | IS NOT NULL
stringComparisonOperator ::= STARTS WITH | ENDS WITH | CONTAINS | ("=~" regex)
regex ::= "define your regular expression here"
propertyName, newString, oldString, label, type ::= String
```

2 SMO_C – Grammar for Evolution operations

```
(*categorization of evolution operations into single-type and multi-type*)
evoOp ::= singleTypeEvoOp | multiTypeEvoOp
singleTypeEvoOp ::= add | delete | rename | transform
multiTypeEvoOp ::= copy | merge | split | move

-------------------------------------------------------------------------------------
ADD
-------------------------------------------------------------------------------------
add ::= set | addUnique

(*create unique*)
addUnique ::= addNodesUnique | addRelsUnique
addNodesUnique ::= MERGE node (ON CREATE setPropery)? (ON MATCH setPropery)?
addRelsUnique ::= MERGE connectedNode (ON CREATE setPropery)? (ON MATCH setPropery)?

(*create without uniqueness check*)
set ::= nativeSet | nonNativeSet
nativeSet ::= addNode | addRel | setLabel | setType | setProperty
addNode ::= CREATE nodeEntity ("," nodeEntity)*
addRel ::= nativeAddRel | addRelOnPath
nativeAddRel ::= CREATE nodeWithRels ("," nodeWithRels)* (*to add nodes and relations
    or just realtions*)
addRelOnPath ::= APOC.NODES.LINK "(" listOfNode "," type "," "{" avoidDuplicates "}" ")
    "
avoidDuplicates ::= Boolean
setLabel ::= SET nodeVar "." (*old*) label "=" value ("," nodeVar "." (*old*) label "="
     value)*
setType ::= SET relVar "." (*old*) type "=" value ("," relVar "." (*old*) type "="
    value)*
setProperty ::= SET property ("," property)*

nonNativeSet ::= CALL (nonNativeAddNodes | nonNativeAddRels | addPropToNode |
    addPropToRel | addLabels | overwriteLabels)
nonNativeAddNodes ::= APOC.CREATE.NODES "(" listOfStrings (*representing labels*) ","
    listOfMaps (*representing properties*) ")" YIELD NODE
nonNativeAddRel ::= APOC.CREATE.RELATIONSHIP "(" (*start*) nodeVar "," map "," (*end*)
    nodeVar ")" YIELD REL
addPropToNode ::= (APOC.CREATE.SETPROPERTY | APOC.CREATE.SETPROPERTIES) "(" (nodeVar |
    listOfNodes) "," listOfKeys "," listOfValues ")" YIELD NODE
addPropToRel ::= (APOC.CREATE.SETRELPROPERTY | APOC.CREATE.SETRELPROPERTIES) "(" (
    relVar | listOfRels) "," listOfKeys "," listOfValues ")" YIELD NODE
addLabels ::= APOC.CREATE.ADDLABELS "(" (nodeVar | listOfNodes) "," listOfStrings (*
    representing labels*) ")"
overwriteLabelsOf*With* = APOC.CREATE.SETLABELS "(" listOfNodes "," listOfStrings")"
    YIELD NODE
```

```
-------------------------------------------------------------------------------------------
RENAME
-------------------------------------------------------------------------------------------
rename ::= CALL (renameNodeFeatures | renameRelFeatures | renameTypes)
(*for all output options see [4], [2], [3],[5]*)
renameNodeFeatures ::= (APOC.REFACTOR.RENAME.LABEL | APOC.REFACTOR.RENAME.NODEPROPERTY)
      "(" (*old*) String "," (*new*) String "," listOfNodes ")" YIELD
     COMMITEDOPERATIONS
renameRelFeatures ::= (APOC.REFACTOR.RENAME.TYPE | APOC.REFACTOR.RENAME.TYPEPROPERTY) "
     (" (*old*) String "," (*new*) String "," listOfRels ")" YIELD COMMITEDOPERATIONS
listOfRels ::= "[" relation ( "," relation)+ "]"
renameTypes ::= APOC.CREATE.SETTYPE "(" relVar "," (*new*) type ")" YIELD INPUT ","
     OUTPUT

-------------------------------------------------------------------------------------------
DELETE
-------------------------------------------------------------------------------------------
(*delete entity types*)
delete ::= restrictedDelete | cascadeDelete | remove
restrictedDelete ::= DELETE (relation | node)
cascadeDelete ::= nativeCascadeDelete | nonNativeCascadeDelete
nativeCascadeDelete ::= DETACH DELETE (nodeWithRel | relation)
nonNativeCascadeDelete ::= CALL APOC.NODES.DELETE "(" nodeId "," batchSize ")" YIELD
     VALUE

(*delete features*)
remove ::= nativeRemove | nonNativeRemove
nativeRemove ::= REMOVE variable "." String (*of a property key, label or type*)
nonNativeRemove ::= CALL (removeLabel | removeNodeProp | removeRelProp)
removeLabel ::= APOC.CREATE.REMOVELABELS "(" (nodeVar | listOfNodes) "," listOfString
     (*resembling labels*)")" YIELD NODE
removeNodeProp ::= APOC.CREATE.REMOVEPROPERTIES "(" (nodeVar | listOfNodes) ","
     listOfString (*resembling keys*) ")" YIELD NODE
removeRelProp ::= APOC.CREATE.REMOVERELPROPERTIES "(" (relVar | listOfRels) ","
     listOfString (*resembling keys*) ")" YIELD REL

-------------------------------------------------------------------------------------------
TRANSFORM
-------------------------------------------------------------------------------------------
transform ::= nodeToRel | RelToNode | propToNode
nodeToRel ::= CALL APOC.REFACTOR.COLLAPSENODE "(" (nodeVar | listOfNodes) "," type ")"
     YIELD INPUT "," OUTPUT "," ERROR
RelToNode ::= CALL APOC.REFACTOR.EXTRACTNODE "(" (relVar | listOfRels) ","
     listOfStrings (*resembling label*) "," type (*outgoing*) "," type (*ingoing*)")"
     YIELD INPUT "," OUTPUT "," ERROR
propToNode ::= CALL APOC.REFACTOR.CATEGORIZE "(" propertyName (*from selected pattern*)
      "," type "," direction "," label (*of newly created node*) "," propertyName (*new
      property name can be set for new node*) "," (listOfStrings (*to be copied*) | "["
      ε "]" ) "]" "," batchSize ")" ";"
direction ::= boolean (*True resembles an outgoing direction of the relation, false is
     interpreted as ingoing. Ingoing from original node entity the property was
     selected from.*)

-------------------------------------------------------------------------------------------
MERGE
-------------------------------------------------------------------------------------------
merge ::= nonNativeMerge | innerJoin
nonNativeMerge ::= CALL ((APOC.REFACTOR.MERGENODES "(" listOfNodes) | (APOC.REFACTOR.
     MERGERELATIONSHIPS "(" listOfRels)) "," ((PROPERTIES ":" typeOfJoin) | (
     propertyName ":" typeOfJoin)*) ("," MERGERELS "=" Boolean)? ("," 
     SINGLEELEMENTASARRAY)? ")" YIELD (NODE | REL)
(*In case the typeOfJoin is set for PROPERTIES a right or left join is only available
     for array type values. By declaring each propertyName with a typeOfJoin a left or
     right join can be accomlished. For a fullOuterInclusiveJoin it is sufficiant to
```

```
    set PROPERTIES ":" COMBINE*)
typeOfJoin ::= fullOuterInclusiveJoin | leftJoin |rightJoin
fullOuterInclusiveJoin ::= COMBINE
leftJoin ::= DISCARD
rightJoin ::= OVERRIDE | OVERWRITE
innerJoinOfEntityTypes ::= innerJoinOfNodes | innerJoinOfRels
innerJoinOfNodes ::= CALL APOC.MERGE.NODE "(" listOfStrings (*resembling labels*) ","
    map (*to search for*) "," map (*on create*) "," map (*on match*) ")"
innerJoinOfRels ::= CALL APOC.MERGE.RELATONSHIP "(" (*start*) nodeVar "," type "," map
    (*of properties in pattern*) "," map (*on create*) "," (*end*) nodeVar "," map (*
    on match*) ")"

-------------------------------------------------------------------------------------
COPY
-------------------------------------------------------------------------------------
copy ::= CALL (copyNodes | copySubgraph | copyProperty)
copyNodes* ::= APOC.REFACTOR.CLONENODES "(" listOfNodes ("," withRels ("","
    listOfStrings (*to skip in clone*) )? )? ")" YIELD INPUT "," OUTPUT "," ERROR
withRels = Boolean
copySubgraph ::= ((APOC.REFACTOR.CLONESUBGRAPH "(" listOfNodes) | (APOC.REFACTOR.
    CLONESUBGRAPHFROMPATH "(" varaibale (*of path*) )) "," (relVar | ("[" variable IN
    function (filter)? "]")) "," nodeToCloneTo ")" YIELD INPUT "," OUTPUT "," ERROR
nodesToCloneTo ::= "{" STANDINNODES ":" "[" nodeVar "," nodeVar (*to copy to*) "]" ("","
     SKIPPROPERTIES ":" listOfStrings )? "}" YIELD INPUT "," OUTPUT "," ERROR
copyProperty ::= SET nodeVar (*from entity to overwrite*) "=" (*overwrite with*) ((
    PROPERTIES "(" nodeVar ")") | map (*in case of an empty map it would resemble a
    removal of all properties*) )

-------------------------------------------------------------------------------------
SPLIT WORKAROUND
-------------------------------------------------------------------------------------
split ::= splitNode | splitRel

splitNodesWithAllRels ::=
getInitialNodeAndClones
CALL "{" MATCH node splitKeyValueListAtProperty (*defines key to split at*) "}"
WITH collectOutput aliasOutput "," collectNodeVar aliasInitialNodes "," keysA "," keysB
loopOverKeysARemoveFromOutputAsResultsA"," loopOverKeysBRemoveFromOutputAsResultsB ","
    output "," initialNodes
overwriteMapsResultsAOfNodesOutput
overwriteMapsResultsBOfNodesInitialNodes

getInitialNodeAndClones ::= CALL "{" MATCH node CALL copyNodes RETURN OUTPUT ","
    nodeVar "}"
splitKeyValueListAtProperty ::=
WITH getPropertiesFromNodeVar aliasMap
WITH getKeysFromMap aliasKeys "," "[" getValuesFromMap "]" aliasValues
WITH DISTINCT "[" rangeOfCollectionKeys WHERE KEYS "(" variable ")" "=" String (*define
     key to split at here*) "]" aliasPosKey "," SIZE "(" keys ")" aliasSizeKeys ","
    keys "," values
WITH collectionKeysFromStrartToPosKey aliasKeysNodeA "," collectionKeysFromPosKeyToEnd
    aliasKeysNodeB
WITH collectKeysNodeA aliasKeysA "," collectKeysNodeB aliasKeysB
RETURN keysA "," keysB

(** of upcoming functions refere to their blue components*)
getPropertiesFrom* ::= PROPERTIES "(" variable ")"
getKeysFrom* ::= KEYS "(" map ")"
getValuesFrom* ::= variable IN getKeysFrom* "|" map "[" variable "]"
rangeOfCollection* = variable IN RANGE "(" 0 "," SIZE "(" collection ")" -1 ")"
elementOfCollectionsByVar ::= variable IN "[" collection "[" variable "]" "]"
collection*FromStrartTo* ::= collection "[.." index "[" 0 "]" "]"
collection*From*ToEnd ::= collection "[" index "[" 0 "]" "..]"
loopOver*RemoveFrom*As* ::= WITH "[" rangeOfCollection* "|" removeListOfKeys*FromNodes
```

```
    * "]" alias*
removeListOfKeys*FromNodes* ::= APOC.MAP.REMOVEKEYS "(" listOfNodes "[" variable "]" ,
    listOfKeys "[" variable "]" ")"
overwriteMaps*OfNodes* = FOREACH "(" rangeOfCollection* "|" FOREACH "("
    elementOfCollectionsByVar "|" setPropsToMap* ")" ")"
setPropsToMap* ::= SET nodeVar "=" PROPERTIES "(" map "[" variable "]" ")"

-------------------------------------------------------------------------------
MOVE
-------------------------------------------------------------------------------
move ::= moveNodesWithRels

moveNodesWithAllRels ::= selectSubgraph copySubgraph deleteOldEndNode
deleteOldEndNode :: = CALL "{" selectPattern WITH collectnodeVar aliasoldEndNode
    FOREACH "(" rangeOfCollectionoldEndNode "|" FOREACH "(" elementOfCollectionsByVar
    "|" nativeCascadeDelete ")" ")" "}"

-------------------------------------------------------------------------------
listOfStrings ::= "[" String ("," String)* "]"
listOfMaps ::= "[" map ( "," map )* "]"
listOfNodes ::= "[" node ( "," node)* "]"
listOfKeys ::= "[" propertyName ( "," propertyName)* "]"
listOfValues ::= "[" value ( "," value)* "]"
nodeId ::= ID "(" nodeVar ")"
batchSize ::= Integer
```

References

[1] Neo4j, Inc. List functions, 2023. URL https://neo4j.com/docs/cypher-manual/current/functions/list/#functions-tobooleanlist. Accessed: 2023-09-03.

[2] Neo4j, Inc. apoc.refactor.rename.type, 2023. URL https://neo4j.com/labs/apoc/4.3/overview/apoc.refactor/apoc.refactor.rename.type/. Accessed: 2023-05-17.

[3] Neo4j, Inc. apoc.refactor.rename.nodeProperty, 2023. URL https://neo4j.com/labs/apoc/4.3/overview/apoc.refactor/apoc.refactor.rename.nodeProperty/. Accessed: 2023-05-17.

[4] Neo4j, Inc. apoc.refactor.rename.label, 2023. URL https://neo4j.com/labs/apoc/4.3/overview/apoc.refactor/apoc.refactor.rename.label/. Accessed: 2023-05-17.

[5] Neo4j, Inc. apoc.refactor.rename.typeProperty, 2023. URL https://neo4j.com/labs/apoc/4.3/overview/apoc.refactor/apoc.refactor.rename.typeProperty/. Accessed: 2023-05-17.

[6] Neo4j, Inc. apoc.path.subgraphAll, 2023. URL https://neo4j.com/labs/apoc/4.3/overview/apoc.path/apoc.path.subgraphAll/. Accessed: 2023-08-24.