NeuPrint Query Library

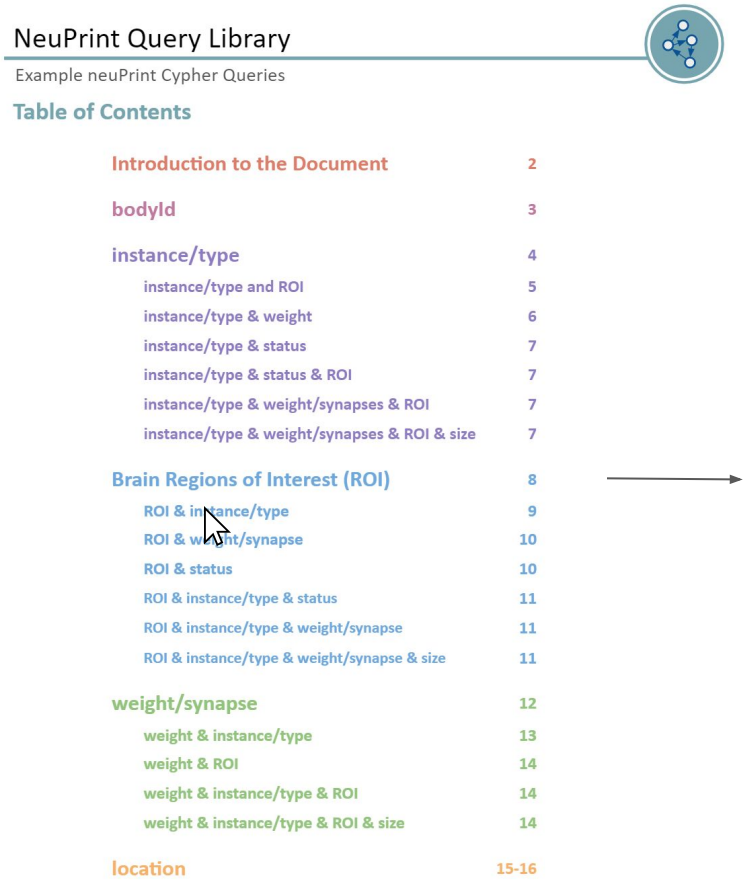
Example neuPrint Cypher Queries

Table of Contents

Intro

The neuPrint Cypher Query Library contains many custom queries that are organized by what is filtered in the WHERE statement (occasionally the filtering is done in the MATCH statement with the format (n:Neuron{`property`:boolean}). Here you can find helpful queries or get inspiration for a query.

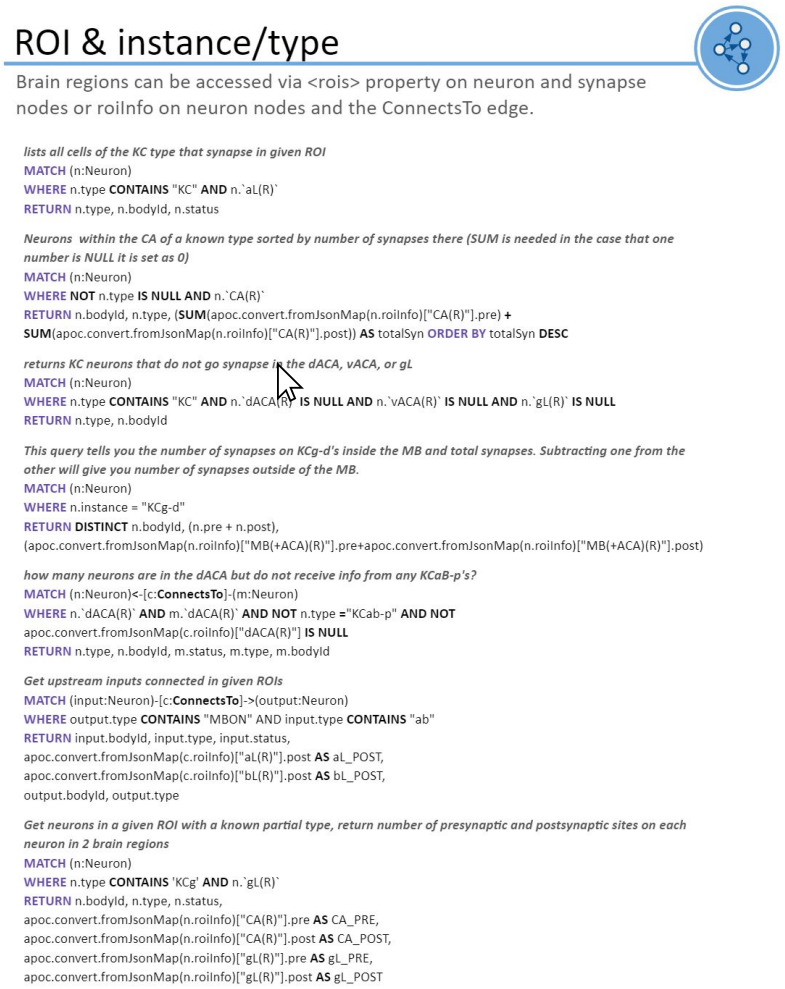
Each section represents a different property filter. Each query will be listed under each of its property filters, thus many queries will be listed multiple times. To navigate, click on the list of filters you want to go browse through in the first page of the document



NeuPrint Query Library

Example neuPrint Cypher Queries

Table of Contents



ROI & instance/type

Brain regions can be accessed via <rois> property on neuron and synapse nodes or roiInfo on neuron nodes and the ConnectsTo edge.

lists all cells of the KC type that synapse in given ROI

```
MATCH (n:Neuron)
WHERE n.type CONTAINS "KC" AND n.`aL(R)`
RETURN n.type, n.bodyId, n.status
```

Neurons within the CA of a known type sorted by number of synapses there (SUM is needed in the case that one number is NULL it is set as 0)

```
MATCH (n:Neuron)
WHERE NOT n.type IS NULL AND n.`CA(R)`
RETURN n.bodyId, n.type, (SUM(apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].pre) +
SUM(apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].post)) AS totalSyn ORDER BY totalSyn DESC
```

returns KC neurons that do not go synapse in the dACA, vACA, or gL

```
MATCH (n:Neuron)
WHERE n.type CONTAINS "KC" AND n.`dACA(R)` IS NULL AND n.`vACA(R)` IS NULL AND n.`gL(R)` IS NULL
RETURN n.type, n.bodyId
```

This query tells you the number of synapses on KCg-d's inside the MB and total synapses. Subtracting one from the other will give you number of synapses outside of the MB.

```
MATCH (n:Neuron)
WHERE n.instance = "KCg-d"
RETURN DISTINCT n.bodyId, (n.pre + n.post),
(apoc.convert.fromJsonMap(n.roiInfo)["MB(+ACA)(R)"].pre+apoc.convert.fromJsonMap(n.roiInfo)["MB(+ACA)(R)"].post)
```

how many neurons are in the dACA but do not receive info from any KCaB-p's?

```
MATCH (n:Neuron)<-[c:ConnectsTo]-(m:Neuron)
WHERE n.`dACA(R)` AND m.`dACA(R)` AND NOT n.type ="KCab-p" AND NOT
apoc.convert.fromJsonMap(c.roiInfo)["dACA(R)"] IS NULL
RETURN n.type, n.bodyId, m.status, m.type, m.bodyId
```

Get upstream inputs connected in given ROIs

```
MATCH (input:Neuron)-[c:ConnectsTo]->(output:Neuron)
WHERE output.type CONTAINS "MBON" AND input.type CONTAINS "ab"
RETURN input.bodyId, input.type, input.status,
apoc.convert.fromJsonMap(c.roiInfo)["aL(R)"].post AS aL_POST,
apoc.convert.fromJsonMap(c.roiInfo)["bL(R)"].post AS bL_POST,
output.bodyId, output.type
```

Get neurons in a given ROI with a known partial type, return number of presynaptic and postsynaptic sites on each neuron in 2 brain regions

```
MATCH (n:Neuron)
WHERE n.type CONTAINS 'KCg' AND n.`gL(R)`
RETURN n.bodyId, n.type, n.status,
apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].pre AS CA_PRE,
apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].post AS CA_POST,
apoc.convert.fromJsonMap(n.roiInfo)["gL(R)"].pre AS gL_PRE,
apoc.convert.fromJsonMap(n.roiInfo)["gL(R)"].post AS gL_POST
```

Example (above): Say you are looking for a query that will help you pull out a list of neurons of a certain type that do NOT have synapses in a couple brain regions. This query would filter by ROI and type, so click on that option in the table of contents

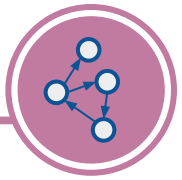
Now you can copy and paste that query, changing it to fit your requirements:

```
MATCH (n:Neuron)
WHERE n.type CONTAINS "KC" AND n.`dACA(R)` IS NULL AND n.`vACA(R)` IS NULL AND n.`gL(R)` IS NULL
RETURN n.type, n.bodyId
```



```
MATCH (n:Neuron)
WHERE n.type CONTAINS "MBON" AND n.`aL(R)` IS NULL AND n.`bL(R)` IS NULL AND n.`a'L(R)` IS NULL
RETURN n.type, n.bodyId
```

bodyId



bodyId is the unique identifier of a body in neuPrint. A body is a segmentation piece with at least 1 synapse.

bodyId is part of so many queries, that this property will not be listed as a filter.

Ways to use bodyId in a custom Cypher query:

```
WITH [123,456,789] AS TARGETS
```

```
WHERE x.bodyId = "123"
```

```
MATCH (x:Neuron{bodyId:123})
```

```
WHERE x.bodyId IN [123,456,789]
```

```
RETURN x.bodyId
```

Count number of postsynaptic sites on a neuron

```
MATCH (n:Neuron)
WHERE n.bodyId = 1078874826
RETURN n.post
```

Get the location of synapses on a neuron, as a list

```
MATCH (n:Neuron)-[:Contains]->(ss:SynapseSet)-[:Contains]->(s:Synapse)
WHERE n.bodyId = 1078874826
RETURN DISTINCT [s.location.x, s.location.y, s.location.z] AS Location
```

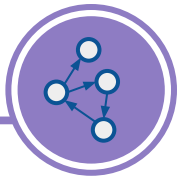
generate list of neurons in the and give synapse count in CA

```
MATCH (n:Neuron{`CA(R)`:true})
RETURN n.bodyId, n.type, n.status, apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].pre AS CA_PRE,
apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].post AS CA_POST,
(SUM(apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].pre) +
SUM(apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].post)) AS total_CA_Syn
```

Number of synapses in given ROIs using roiInfo property of a neuron node

```
WITH [1078874826, 5813092931, 5813114700] AS IDS
UNWIND IDS AS ID
MATCH (n:Neuron)
WHERE n.bodyId = ID
RETURN n.bodyId, n.instance,
SUM(apoc.convert.fromJsonMap(n.roiInfo)["SMP(R)"].pre) AS SMP_PRE,
SUM(apoc.convert.fromJsonMap(n.roiInfo)["SMP(R)"].post) AS SMP_POST,
SUM(apoc.convert.fromJsonMap(n.roiInfo)["SIP(R)"].pre) AS SIP_PRE,
SUM(apoc.convert.fromJsonMap(n.roiInfo)["SIP(R)"].post) AS SIP_POST,
SUM(apoc.convert.fromJsonMap(n.roiInfo)["SLP(R)"].pre) AS SLP_PRE,
SUM(apoc.convert.fromJsonMap(n.roiInfo)["SLP(R)"].post) AS SLP_POST
```

instance/type (of a neuron)



type contains the neuron type of the body. instance indicates a name that indicates a more specific instance of a neuron type. type is also a property of synapses that indicates pre or postsynaptic (not included here).

Ways to use instance or type in a custom Cypher query:

```
WHERE x.instance = "KCg-d"
```

```
WHERE x.instance STARTS WITH "KC"
```

```
WHERE x.instance =~ '.*Delta6.*
```

```
WHERE x.type CONTAINS "KC"
```

```
RETURN x.type
```

returns all neurons that contain KC in their name, returns name and body ID

```
MATCH (n:Neuron)
WHERE n.type CONTAINS "KCab-p"
RETURN n.type AS Type, n.bodyId AS Body_ID
```

sorts a list of neurons by the number of synapses onto KCab-p's

```
MATCH (n:Neuron)-[s:ConnectsTo]->(m:Neuron)
WHERE n.bodyId IN [5812980567,359586213,735282133] AND m.type CONTAINS
"KCab-p"
RETURN n.type, n.bodyId, COUNT(s) AS count
ORDER BY count DESC
```

returns the number of PAM neurons and the number of all PAM synapses connected to specified output Body Ids

```
MATCH (input:Neuron)-[w:ConnectsTo]->(output:Neuron)
WHERE output.bodyId IN [1078874826, 5813092931, 5813114700] AND
input.type=~"PAM.*"
RETURN COUNT(input.bodyId), SUM(w.weightHP), output.bodyId
```

instance/type & ROI

type contains the neuron type of the body. instance indicates a name that indicates a more specific instance of a neuron type. type is also a property of synapses that indicates pre or postsynaptic (not included here).

lists all cells of the KC type that synapse in given ROI

```
MATCH (n:Neuron)
WHERE n.type CONTAINS "KC" AND n.`aL(R)`
RETURN n.type, n.bodyId, n.status
```

Neurons within the CA of a known type sorted by number of synapses there (SUM is needed in the case that one number is NULL it is set as 0)

```
MATCH (n:Neuron)
WHERE NOT n.type IS NULL AND n.`CA(R)`
RETURN n.bodyId, n.type, (SUM(apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].pre) +
SUM(apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].post)) AS totalSyn ORDER BY totalSyn DESC
```

returns KC neurons that do not go synapse in the dACA, vACA, or gL

```
MATCH (n:Neuron)
WHERE n.type CONTAINS "KC" AND n.`dACA(R)` IS NULL AND n.`vACA(R)` IS NULL AND n.`gL(R)` IS NULL
RETURN n.type, n.bodyId
```

This query tells you the number of synapses on KCg-d's inside the MB and total synapses. Subtracting one from the other will give you number of synapses outside of the MB.

```
MATCH (n:Neuron)
WHERE n.instance = "KCg-d"
RETURN DISTINCT n.bodyId, (n.pre + n.post),
(apoc.convert.fromJsonMap(n.roiInfo)["MB(+ACA)(R)"].pre+apoc.convert.fromJsonMap(n.roiInfo)["MB(+ACA)(R)"].post
)
```

how many neurons are in the dACA but do not receive info from any KCaB-p's?

```
MATCH (n:Neuron)<-[c:ConnectsTo]-(m:Neuron)
WHERE n.`dACA(R)` AND m.`dACA(R)` AND NOT n.type ="KCab-p" AND NOT
apoc.convert.fromJsonMap(c.roiInfo)["dACA(R)"] IS NULL
RETURN n.type, n.bodyId, m.status, m.type, m.bodyId
```

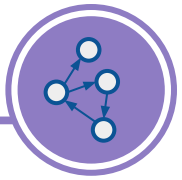
Get upstream inputs connected in given ROIs

```
MATCH (input:Neuron)-[c:ConnectsTo]->(output:Neuron)
WHERE output.type CONTAINS "MBON" AND input.type CONTAINS "ab"
RETURN input.bodyId, input.type, input.status,
apoc.convert.fromJsonMap(c.roiInfo)["aL(R)"].post AS aL_POST,
apoc.convert.fromJsonMap(c.roiInfo)["bL(R)"].post AS bL_POST,
output.bodyId, output.type
```

Get neurons in a given ROI with a known partial type, return number of presynaptic and postsynaptic sites on each neuron in 2 brain regions

```
MATCH (n:Neuron)
WHERE n.type CONTAINS 'KCg' AND n.`gL(R)`
RETURN n.bodyId, n.type, n.status,
apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].pre AS CA_PRE,
apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].post AS CA_POST,
apoc.convert.fromJsonMap(n.roiInfo)["gL(R)"].pre AS gL_PRE,
apoc.convert.fromJsonMap(n.roiInfo)["gL(R)"].post AS gL_POST
```

instance/type & weight



type contains the neuron type of the body. instance indicates a name that indicates a more specific instance of a neuron type. type is also a property of synapses that indicates pre or postsynaptic (not included here).

Get downstream partners of neurons with a known name

```
MATCH (input:Neuron)-[w:ConnectsTo]->(output:Neuron)
WHERE input.instance = "KCab-p" AND w.weightHP >= 10
RETURN input.bodyId, input.instance, w.weightHP, output.bodyId, output.instance
```

Get downstream partners of neurons with a known name. Filter outputs by type and weight

```
MATCH (RH_FB:Neuron)-[w:ConnectsTo]->(output:Neuron)
WHERE RH_FB.type = "FB7A" AND w.weightHP >= 5 AND output.instance CONTAINS "Delta6D"
RETURN RH_FB.bodyId AS FB_ID, RH_FB.instance AS FB_INSTANCE, w.weightHP AS W, output.bodyId AS
OUTPUT_ID, output.instance AS OUTPUT_INSTANCE
ORDER BY W DESC
```

Get upstream partners of a list of body IDs. Filter inputs by partial instance

```
WITH [1078874826, 5813092931, 5813114700] AS RH_IDS
UNWIND RH_IDS AS RH_ID
MATCH (input:Neuron)-[w:ConnectsTo]->(output:Neuron)
WHERE output.bodyId = RH_ID AND w.weightHP >= 5 AND input.instance CONTAINS "PAM"
RETURN input.bodyId AS INPUT_ID, input.instance AS INPUT_NAME, output.bodyId AS OUTPUT_ID,
output.instance AS OUTPUT_NAME, w.weightHP AS W
ORDER BY W DESC
```

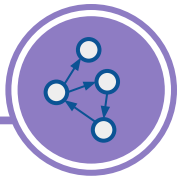
Upstream partners of neurons with a known name. Upstream neurons of the same name are excluded from the results.

```
WITH [1078874826, 5813092931, 5813114700] AS RH_IDS
UNWIND RH_IDS AS RH_ID
MATCH (input:Neuron)-[w:ConnectsTo]->(output:Neuron)
WHERE output.instance = "FB08a(PDL05)_R" AND NOT input.instance CONTAINS "FB08a(PDL05)_R"
AND w.weightHP >= 5
RETURN input.bodyId AS INPUT_ID, input.instance AS INPUT_INSTANCE, w.weightHP AS W,
output.bodyId AS OUTPUT_ID, output.instance AS OUTPUT_INSTANCE
ORDER BY W DESC
```

1 hop upstream pathways from a neuron with a known body ID. Input filter by partial type

```
MATCH (FB:Neuron)<-[w2:ConnectsTo]-(interneuron:Neuron)<-[w:ConnectsTo]-(input:Neuron)
WHERE FB.bodyId = 1078874826 AND w.weightHP >= 10 AND w2.weightHP >= 10 AND input.type
CONTAINS 'MBON'
RETURN input.bodyId AS INPUT_ID, input.type AS INPUT_TYPE, w.weightHP AS W, interneuron.bodyId
AS INTER_ID, interneuron.type AS INTER_TYPE, w2.weightHP AS W2, FB.bodyId AS FB_ID, FB.type AS
FB_TYPE
ORDER BY W DESC
```

instance/type & status & ROI & weight & size



type contains the neuron type of the body. instance indicates a name that indicates a more specific instance of a neuron type. type is also a property of synapses that indicates pre or postsynaptic (not included here).

instance/type & status

Get neurons of a certain name with a given status

```
MATCH (n:Neuron)
WHERE n.type ='KCab-p' AND n.status = 'Traced'
RETURN n.type AS TYPE, n.bodyId AS ID
```

returns neurons upstream of KCaB-p's (that connect through synapses in the dACA) that do not synapse in the OL and returns the number of synapses that connect it to the KC.

```
MATCH (n:Neuron)-[c:ConnectsTo]->(m:Neuron)
WHERE m.type = "KCab-p" AND NOT apoc.convert.fromJsonMap(c.roiInfo)["dACA(R)"] IS NULL AND
apoc.convert.fromJsonMap(c.roiInfo)["OL(R)"] IS NULL AND n.status ="Traced"
RETURN DISTINCT n.bodyId, n.type, apoc.convert.fromJsonMap(n.roiInfo)["dACA(R)"].pre AS dACA_PRE,
apoc.convert.fromJsonMap(n.roiInfo)["dACA(R)"].post AS dACA_POST
```

instance/type & status & ROI

returns unnamed traced bodies in dACA

```
MATCH (n:Neuron)
WHERE n.`dACA(R)` AND n.instance IS NULL AND n.type IS NULL  AND n.status = "Traced"
RETURN n.bodyId, apoc.convert.fromJsonMap(n.roiInfo)["dACA(R)"].pre AS dACA_PRE,
apoc.convert.fromJsonMap(n.roiInfo)["dACA(R)"].post AS dACA_POST, n.status
```

how many neurons are in the dACA but do not receive info from any KCaB_p's? but pre or post connection is not named KC or pLH (PLP local neuron) and no anchors or orphans or no status.

```
MATCH (n:Neuron{`dACA(R)`:true})<-[c:ConnectsTo]-(m:Neuron{`dACA(R)`:true})
WHERE NOT n.instance CONTAINS "KC" AND NOT n.instance CONTAINS "pLH" AND NOT m.instance CONTAINS
"KC" AND NOT m.instance CONTAINS "pLH" AND m.status = "Traced"
AND NOT apoc.convert.fromJsonMap(c.roiInfo)["dACA(R)"] IS NULL
RETURN DISTINCT m.bodyId, m.status, m.instance
```

instance/type & weight/synapses & ROI

return a list of body IDs in the CA that are fragments with 1-3+ t-bars that are no status that connect to at least 4 PSD that belong to named neurons (timeout)

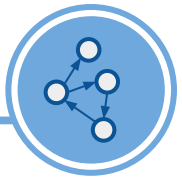
```
MATCH (n:Segment)-[w:ConnectsTo]->(m:Segment)
WHERE n.`CA(R)` AND m.`CA(R)` AND n.pre < 4 AND n.pre > 0 AND n.instance IS NULL AND NOT m.instance IS
NULL
RETURN n.bodyId, n.pre AS PRE, count(m) AS NAMED_PSD
ORDER BY PRE DESC, NAMED_PSD DESC
```

instance/type & weight/synapse & ROI & size

returns a list of tiny 1 t-bar CA segments that are not named and that connect to named CA bodies

```
MATCH (n:Segment{`CA(R)`:true})-[w:ConnectsTo]->(m:Neuron{`CA(R)`:true})
WHERE n.pre = 1 AND n.instance IS NULL AND NOT m.instance IS NULL AND n.size < 3000
RETURN n.bodyId, n.pre AS PRE, count(m) AS NAMED_PSD
ORDER BY PRE DESC, NAMED_PSD DESC
```

ROI (Brain Region of Interest)



Brain regions can be accessed via <rois> property on neuron and synapse nodes or roiInfo on neuron nodes and the ConnectsTo edge.

Ways to use instance or type in a custom Cypher query:

```
MATCH (x:Neuron{`ROI`:true})
WHERE x.`ROI`
RETURN x.roiInfo
```

```
MATCH (s:Synapse{`ROI`:true})
WHERE s.`ROI` IS NULL
MATCH (s:Synapse{`ROI`:true})
```

Note - x.roiInfo will return a json of all ROIs that the node/edge is in. To pull out the synapse number in a specific ROI, use this function:

```
WHERE apoc.convert.fromJsonMap(n.roiInfo)["ROI"].pre > 5
```

```
WHERE apoc.convert.fromJsonMap(c.roiInfo)["ROI"] IS NULL
```

```
RETURN (apoc.convert.fromJsonMap(c.roiInfo)["ROI"].post +
apoc.convert.fromJsonMap(c.roiInfo)["ROI"].pre)
```

Generate a list of bodies in an ROI (replace Neuron with Segment to include really tiny bodies in results)

```
MATCH (n:Neuron{`CA(R)`:true})
RETURN n.bodyId, n.status, n.pre, n.post
```

Count number of neurons in a left hemisphere ROI

```
MATCH (n:Neuron)
WHERE n.`SPS(L)`
RETURN count(n)
```

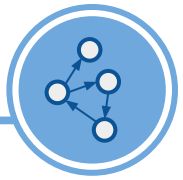
Get list of bodies in one ROI but not in another

```
MATCH (n:Neuron)
WHERE n.EB AND n.FB IS NULL
RETURN n.bodyId, n.instance
```

Check if bodies are in certain ROIs, returns the number of synapses in that ROI

```
WITH [664511977, 5813092931, 5813114700] AS TARGETS
MATCH (n:Neuron)
WHERE n.bodyId IN TARGETS AND n.`bL(R)`
RETURN n.bodyId, (apoc.convert.fromJsonMap(n.roiInfo)["bL(R)"].post +
apoc.convert.fromJsonMap(n.roiInfo)["bL(R)"].pre) AS totalBLSyn
```

ROI & instance/type



Brain regions can be accessed via <rois> property on neuron and synapse nodes or roiInfo on neuron nodes and the ConnectsTo edge.

lists all cells of the KC type that synapse in given ROI

```
MATCH (n:Neuron)
WHERE n.type CONTAINS "KC" AND n.`aL(R)`
RETURN n.type, n.bodyId, n.status
```

Neurons within the CA of a known type sorted by number of synapses there (SUM is needed in the case that one number is NULL it is set as 0)

```
MATCH (n:Neuron)
WHERE NOT n.type IS NULL AND n.`CA(R)`
RETURN n.bodyId, n.type, (SUM(apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].pre) +
SUM(apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].post)) AS totalSyn ORDER BY totalSyn DESC
```

returns KC neurons that do not go synapse in the dACA, vACA, or gL

```
MATCH (n:Neuron)
WHERE n.type CONTAINS "KC" AND n.`dACA(R)` IS NULL AND n.`vACA(R)` IS NULL AND n.`gL(R)` IS NULL
RETURN n.type, n.bodyId
```

This query tells you the number of synapses on KCg-d's inside the MB and total synapses. Subtracting one from the other will give you number of synapses outside of the MB.

```
MATCH (n:Neuron)
WHERE n.instance = "KCg-d"
RETURN DISTINCT n.bodyId, (n.pre + n.post),
(apoc.convert.fromJsonMap(n.roiInfo)["MB(+ACA)(R)"].pre+apoc.convert.fromJsonMap(n.roiInfo)["MB(+ACA)(R)"].post
)
```

how many neurons are in the dACA but do not receive info from any KCaB-p's?

```
MATCH (n:Neuron)<-[c:ConnectsTo]-(m:Neuron)
WHERE n.`dACA(R)` AND m.`dACA(R)` AND NOT n.type ="KCab-p" AND NOT
apoc.convert.fromJsonMap(c.roiInfo)["dACA(R)"] IS NULL
RETURN n.type, n.bodyId, m.status, m.type, m.bodyId
```

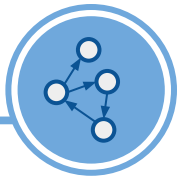
Get upstream inputs connected in given ROIs

```
MATCH (input:Neuron)-[c:ConnectsTo]->(output:Neuron)
WHERE output.type CONTAINS "MBON" AND input.type CONTAINS "ab"
RETURN input.bodyId, input.type, input.status,
apoc.convert.fromJsonMap(c.roiInfo)["aL(R)"].post AS aL_POST,
apoc.convert.fromJsonMap(c.roiInfo)["bL(R)"].post AS bL_POST,
output.bodyId, output.type
```

Get neurons in a given ROI with a known partial type, return number of presynaptic and postsynaptic sites on each neuron in 2 brain regions

```
MATCH (n:Neuron)
WHERE n.type CONTAINS 'KCg' AND n.`gL(R)`
RETURN n.bodyId, n.type, n.status,
apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].pre AS CA_PRE,
apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].post AS CA_POST,
apoc.convert.fromJsonMap(n.roiInfo)["gL(R)"].pre AS gL_PRE,
apoc.convert.fromJsonMap(n.roiInfo)["gL(R)"].post AS gL_POST
```

ROI & weight/synapse & status



Brain regions can be accessed via <rois> property on neuron and synapse nodes or roiInfo on neuron nodes and the ConnectsTo edge.

ROI & weight/synapse

Returns the number of presynaptic sites that are on bodies in the dACA that have at least 40 pre*5+postsynaptic sites.

```
MATCH (n:Neuron)
WHERE (n.pre*5+n.post) > 40 AND n.`dACA(R)`
RETURN sum(n.pre)
```

returns neurons with 5+ post syn connections in the CA ROI

```
MATCH (n:Neuron)
WHERE n.`CA(R)` AND apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].post > 4
RETURN DISTINCT n.bodyId,  apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].post,  n.type as type
```

returns synapses of connections to a body through a specific ROI

```
MATCH (input:`hemibrain_Neuron`)-[c:ConnectsTo]->(output:`hemibrain_Neuron`)
WHERE input.bodyId=1078874826 AND c.weight > 10 AND NOT
apoc.convert.fromJsonMap(c.roiInfo)["bL(R)"] IS NULL
RETURN input.bodyId, output.bodyId, apoc.convert.fromJsonMap(c.roiInfo)["bL(R)"].post AS bL_Syn
```

returns neurons that are upstream of your neuron that pass through the LO ROI

```
WITH [518930199] AS TARGETS
MATCH (n:Neuron)<-[w:ConnectsTo]-(m:Neuron{`LO(R)`:true})
WHERE n.bodyId IN TARGETS AND w.weight >= 5
RETURN m.bodyId, w.weight
```

ROI & status

Count number of neurons in one ROI with a given status

```
MATCH (n:Neuron)
WHERE n.status= 'Assign' AND n.`CRE(R)`
RETURN count(n)
```

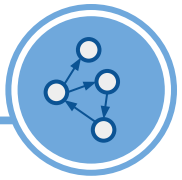
returns all bodies in CA without a status

```
MATCH (n:Neuron)
WHERE n.`CA(R)` AND n.status IS NULL
RETURN n.bodyId, n.pre, n.post
```

finds all orphans in CA

```
MATCH (n:Neuron)
WHERE n.`CA(R)` AND n.status = "Orphan"
RETURN n.instance, n.status, n.bodyId, n.pre, n.post ORDER BY n.pre DESC
```

ROI & weight/synapse & status & instance/type & size



Brain regions can be accessed via <rois> property on neuron and synapse nodes or roiInfo on neuron nodes and the ConnectsTo edge.

ROI & instance/type & status

returns unnamed traced bodies in dACA

```
MATCH (n:Neuron)
WHERE n.`dACA(R)` AND n.instance IS NULL AND NOT n.status = "Traced"
RETURN n.bodyId, apoc.convert.fromJsonMap(n.roiInfo)["dACA(R)"].pre AS dACA_PRE,
apoc.convert.fromJsonMap(n.roiInfo)["dACA(R)"].post AS dACA_POST, n.status
```

how many neurons are in the dACA but do not receive info from any KCaB_p's? but pre or post connection is not named KC or pLH (PLP local neuron) and no anchors or orphans or no status.

```
MATCH (n:Neuron{`dACA(R)`:true})<-[c:ConnectsTo]-(m:Neuron{`dACA(R)`:true})
WHERE NOT n.instance CONTAINS "KC" AND NOT n.instance CONTAINS "pLH" AND NOT m.instance
CONTAINS "KC" AND NOT m.instance CONTAINS "pLH" AND m.status = "Traced"
AND NOT apoc.convert.fromJsonMap(c.roiInfo)["dACA(R)"] IS NULL
RETURN DISTINCT m.bodyId, m.status, m.instance
```

ROI & instance/type & weight/synapses

return a list of body IDs in the CA that are fragments with 1-3+ t-bars that are no status that connect to at least 4 PSD that belong to named neurons (timeout)

```
MATCH (n:Segment)-[w:ConnectsTo]->(m:Segment)
WHERE n.`CA(R)` AND m.`CA(R)` AND n.pre < 4 AND n.pre > 0 AND n.instance IS NULL AND NOT
m.instance IS NULL
RETURN n.bodyId, n.pre AS PRE, count(m) AS NAMED_PSD
ORDER BY PRE DESC, NAMED_PSD DESC
```

ROI & instance/type & weight/synapse & size

returns a list of tiny 1 t-bar CA segments that are not named and that connect to named CA bodies

```
MATCH (n:Segment{`CA(R)`:true})-[w:ConnectsTo]->(m:Neuron{`CA(R)`:true})
WHERE n.pre = 1 AND n.instance IS NULL AND NOT m.instance IS NULL AND n.size < 3000
RETURN n.bodyId, n.pre AS PRE, count(m) AS NAMED_PSD
ORDER BY PRE DESC, NAMED_PSD DESC
```

weight/synapse

weight indicates the number of synapses in a connection between two neurons. (weight, property of [:ConnectsTo], pre/post properties of (:Neuron))

Ways to use instance or type in a custom Cypher query:

```
WHERE c.weight > 50
```

```
RETURN c.weight
```

```
RETURN SUM(c.weight)
```

Downstream partners - Input: list of body IDs (RH_IDs), Output: list of body IDs (DELTA6DS)

```
WITH [5813055834, 5813061495, 946641212, 946308203] AS RH_IDS, [915605750, 885257867, 886280796] AS DELTA6DS
UNWIND RH_IDS AS RH_ID
MATCH (input:Neuron)-[w:ConnectsTo]->(output:Neuron)
WHERE input.bodyId = RH_ID AND w.weightHP >= 5 AND output.bodyId IN DELTA6DS
RETURN input.bodyId AS INPUT_ID, input.type AS INPUT_TYPE, w.weightHP AS W, output.bodyId AS OUTPUT_ID, output.type AS OUTPUT_TYPE ORDER BY W DESC
```

Upstream partners to a list of body IDs. Connections from one neuron in the list to another neuron in the list excluded

```
WITH [327203386,297183251,5813020735,5813020698,294800293,5813081818] AS RH_IDS
UNWIND RH_IDS AS RH_ID
MATCH (input:Neuron)-[w:ConnectsTo]->(output:Neuron)
WHERE output.bodyId = RH_ID AND w.weightHP >= 5 AND NOT input.bodyId IN RH_IDS
RETURN input.bodyId, input.instance, w.weightHP, output.bodyId, output.instance ORDER BY w.weightHP DESC
```

pulls High Confidence weights for connections between the 2 groups of neurons. Applying >=1 weightHP threshold removes connections that are only Low Confidence.

```
MATCH (input:Neuron)-[w:ConnectsTo]->(output:Neuron)
WHERE input.bodyId IN [674100160,705795727,725839388] AND output.bodyId IN [1317712712,518930199,547552266,5813060726,541127846,417221397,769263962] AND w.weightHP>=1
RETURN input.bodyId, output.bodyId, w.weightHP
```

1 hop pathways with interneuron - change OUTPUT_IDS, INPUT_IDS, and THRESHOLD

```
WITH [5813060787, 1286315478] AS OUTPUT_IDS, [704466265,861665641] AS INPUT_IDS, 5 AS THRESHOLD
UNWIND OUTPUT_IDS AS OUTPUT_ID
UNWIND INPUT_IDS AS INPUT_ID
MATCH (input:Neuron)-[w:ConnectsTo]->(interneuron:Neuron)-[w2:ConnectsTo]->(output:Neuron)
WHERE input.bodyId = INPUT_ID AND output.bodyId = OUTPUT_ID AND w.weight > THRESHOLD AND w2.weight > THRESHOLD
RETURN input.bodyId, input.type, w.weight AS W, interneuron.bodyId, interneuron.type, w2.weight AS W2, output.bodyId, output.type ORDER BY W DESC, W2 DESC
```

Three hops: two-Interneuron that are downstream of MBONg2 and also input to FB02 with High Precision connections of 10 or more, Ordered by weightHP

```
MATCH(n:Neuron)-[w:ConnectsTo]->(m:Neuron)-[ww:ConnectsTo]->(i:Neuron)-[www:ConnectsTo]->(o:Neuron)
WHERE n.bodyId IN[704466265,861665641] AND o.bodyId IN[1255297788,5813060787,1286315478]
AND w.weightHP > 9 AND ww.weightHP > 9 AND www.weightHP > 9
RETURN n.type AS InputType, n.bodyId AS InputBodyId, w.weightHP AS WeightHP, m.type AS Interneuron1Type, m.bodyId AS Interneuron1BodyId, m.status AS Interneuron1Status, ww.weightHP AS wWeightHP, i.type AS Interneuron2Type, i.bodyId AS Interneuron2BodyId, i.status AS Interneuron2Status,  www.weightHP AS wwWeightHP, o.type AS OutputType, o.bodyId AS OutputBodyID
ORDER BY  WeightHP DESC, wWeightHP DESC, wwWeightHP DESC
```

weight & instance/type (of a neuron)

type contains the neuron type of the body. instance indicates a name that indicates a more specific instance of a neuron type. type is also a property of synapses that indicates pre or postsynaptic (not included here).

Get downstream partners of neurons with a known name

```
MATCH (input:Neuron)-[w:ConnectsTo]->(output:Neuron)
WHERE input.instance = "KCab-p" AND w.weightHP >= 10
RETURN input.bodyId, input.instance, w.weightHP, output.bodyId, output.instance
```

Get downstream partners of neurons with a known name. Filter outputs by type and weight

```
MATCH (RH_FB:Neuron)-[w:ConnectsTo]->(output:Neuron)
WHERE RH_FB.type = "FB7A" AND w.weightHP >= 5 AND output.instance CONTAINS "Delta6D"
RETURN RH_FB.bodyId AS FB_ID, RH_FB.instance AS FB_INSTANCE, w.weightHP AS W, output.bodyId AS
OUTPUT_ID, output.instance AS OUTPUT_INSTANCE
ORDER BY W DESC
```

Get upstream partners of a list of body IDs. Filter inputs by partial instance

```
WITH [1078874826, 5813092931, 5813114700] AS RH_IDS
UNWIND RH_IDS AS RH_ID
MATCH (input:Neuron)-[w:ConnectsTo]->(output:Neuron)
WHERE output.bodyId = RH_ID AND w.weightHP >= 5 AND input.instance CONTAINS "PAM"
RETURN input.bodyId AS INPUT_ID, input.instance AS INPUT_NAME, output.bodyId AS OUTPUT_ID,
output.instance AS OUTPUT_NAME, w.weightHP AS W
ORDER BY W DESC
```

Upstream partners of neurons with a known name. Upstream neurons of the same name are excluded from the results.

```
WITH [1078874826, 5813092931, 5813114700] AS RH_IDS
UNWIND RH_IDS AS RH_ID
MATCH (input:Neuron)-[w:ConnectsTo]->(output:Neuron)
WHERE output.instance = "FB08a(PDL05)_R" AND NOT input.instance CONTAINS "FB08a(PDL05)_R"
AND w.weightHP >= 5
RETURN input.bodyId AS INPUT_ID, input.instance AS INPUT_INSTANCE, w.weightHP AS W,
output.bodyId AS OUTPUT_ID, output.instance AS OUTPUT_INSTANCE
ORDER BY W DESC
```

1 hop upstream pathways from a neuron with a known body ID. Input filter by partial type

```
MATCH (FB:Neuron)<-[w2:ConnectsTo]-(interneuron:Neuron)<-[w:ConnectsTo]-(input:Neuron)
WHERE FB.bodyId = 1078874826 AND w.weightHP >= 10 AND w2.weightHP >= 10 AND input.type
CONTAINS 'MBON'
RETURN input.bodyId AS INPUT_ID, input.type AS INPUT_TYPE, w.weightHP AS W, interneuron.bodyId
AS INTER_ID, interneuron.type AS INTER_TYPE, w2.weightHP AS W2, FB.bodyId AS FB_ID, FB.type AS
FB_TYPE
ORDER BY W DESC
```

weight & ROI & instance/type & size

type contains the neuron type of the body. instance indicates a name that indicates a more specific instance of a neuron type. type is also a property of synapses that indicates pre or postsynaptic (not included here).

weight/synapse & ROI

Returns the number of presynaptic sites that are on bodies in the dACA that have at least 40 pre*5+postsynaptic sites.

```
MATCH (n:Neuron)
WHERE (n.pre*5+n.post) > 40 AND n.`dACA(R)`
RETURN sum(n.pre)
```

returns neurons with 5+ post syn connections in the CA ROI

```
MATCH (n:Neuron)
WHERE n.`CA(R)` AND apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].post > 4
RETURN DISTINCT n.bodyId,  apoc.convert.fromJsonMap(n.roiInfo)["CA(R)"].post,  n.type as type
```

returns synapses of connections to a body through a specific ROI

```
MATCH (input:`hemibrain_Neuron`)-[c:ConnectsTo]->(output:`hemibrain_Neuron`)
WHERE input.bodyId=1078874826 AND c.weight > 10 AND NOT
apoc.convert.fromJsonMap(c.roiInfo)["bL(R)"] IS NULL
RETURN input.bodyId, output.bodyId, apoc.convert.fromJsonMap(c.roiInfo)["bL(R)"].post AS bL_Syn
```

returns neurons that are upstream of your neuron that pass through the LO ROI

```
WITH [518930199] AS TARGETS
MATCH (n:Neuron)<-[w:ConnectsTo]-(m:Neuron{`LO(R)`:true})
WHERE n.bodyId IN TARGETS AND w.weight >= 5
RETURN m.bodyId, w.weight
```

weight/synapses & instance/type & ROI

return a list of body IDs in the CA that are fragments with 1-3+ t-bars that are no status that connect to at least 4 PSD that belong to named neurons (timeout)

```
MATCH (n:Segment)-[w:ConnectsTo]->(m:Segment)
WHERE n.`CA(R)` AND m.`CA(R)` AND n.pre < 4 AND n.pre > 0 AND n.instance IS NULL AND NOT
m.instance IS NULL
RETURN n.bodyId, n.pre AS PRE, count(m) AS NAMED_PSD
ORDER BY PRE DESC, NAMED_PSD DESC
```

weight/synapse & instance/type & ROI & size

returns a list of tiny 1 t-bar CA segments that are not named and that connect to named CA bodies

```
MATCH (n:Segment{`CA(R)`:true})-[w:ConnectsTo]->(m:Neuron{`CA(R)`:true})
WHERE n.pre = 1 AND n.instance IS NULL AND NOT m.instance IS NULL AND n.size < 3000
RETURN n.bodyId, n.pre AS PRE, count(m) AS NAMED_PSD
ORDER BY PRE DESC, NAMED_PSD DESC
```

location

location of individual synapses

Ways to use instance or type in a custom Cypher query:

```
WHERE s.location.z < 15000
```

```
WHERE s.location.x = 18000
```

```
RETURN s.location
```

```
RETURN [s.location.x, s.location.y, s.location.z]
```

```
RETURN s.location.y
```

return all postsynaptic partners of a specific presynaptic site

```
MATCH
(output:Neuron)-[:Contains]->(:SynapseSet)-[Contains]->(post:Synapse)<-[:SynapsesTo]-(pre:Synapse)<-[:Contains]-(:SynapseSet)<-[:Contains]-(input:Neuron)
WHERE pre.location.x = 15723 AND pre.location.y = 20613 AND pre.location.z = 27544 AND input.bodyId = 1503999967
RETURN input.bodyId, output.bodyId
```

returns downstream neurons from a list of neurons where the connection is below a certain z value

```
WITH
[5813011738,512023157,479969106,389536926,454335501,480997558,450597298,5813019543,511634742] AS TARGETS
MATCH
(n:Neuron)-[:Contains]->(:SynapseSet)-[:Contains]->(s:Synapse)-[:SynapsesTo]->(p:Synapse)<-[:Contains]-(:SynapseSet)<-[:Contains]-(m:Neuron)
WHERE s.location.z > 15423 AND n.bodyId IN TARGETS
RETURN m.instance, m.bodyId, COUNT(s) ORDER BY COUNT(s) DESC
```

returns neurons that are upstream of a list of neurons that are connected by synapses in the cube (around vACA) that is defined in the where clause. KCg-d's in TARGETS

```
WITH [975518615, 601388689, 601725569, 631741308, 786260089] AS TARGETS
MATCH
(n:Neuron)-[:Contains]->(:SynapseSet)-[Contains]->(s:Synapse)<-[:SynapsesTo]-(:Synapse)<-[:Contains]-(:SynapseSet)<-[:Contains]-(m:Neuron)
WHERE n.bodyId IN TARGETS AND s.location.x >= 10614 AND s.location.x <= 18066
AND s.location.y >= 15685 AND s.location.y <= 21315
AND s.location.z >= 11845 AND s.location.z<= 19405
RETURN DISTINCT m.bodyId ORDER BY m.bodyId DESC
```

returns a list of postsynaptic sites on a list of neurons within a cube. KCab-p's in TARGETS

```
WITH [456467162, 425086851, 456467162] AS TARGETS
MATCH (n:Neuron)-[:Contains]->(:SynapseSet)-[Contains]->(s:Synapse)
WHERE n.bodyId IN TARGETS AND s.location.x >= 10492 AND s.location.x <= 13948
AND s.location.y >= 17557 AND s.location.y <= 19312
AND s.location.z >= 7210 AND s.location.z<= 12975
RETURN DISTINCT n.bodyId AS KC, s.location.x AS X, s.location.y AS Y, s.location.z AS Z
ORDER BY n.bodyId DESC
```

location

location of individual synapses

get the location of synapses on a set of neurons within of a series of cubes

```
WITH [5812983199, 5812982877, 5812979119, 5812983438, 1234809285, 1203779116] AS TARGETS
MATCH (n:Neuron)-[:Contains]->(:SynapseSet)-[Contains]->(s:Synapse)
WHERE s.type = "post" AND n.bodyId IN TARGETS AND s.location.x >= 7515 AND s.location.x <= 10880
AND s.location.y >= 12200 AND s.location.y <= 16750
AND s.location.z >= 14188 AND s.location.z<= 21080
AND s.vACA IS NULL AND s.CA IS NULL
RETURN DISTINCT s.location.x AS X, s.location.y AS Y, s.location.z AS Z ORDER BY s.location.x DESC
UNION
WITH [5812983199, 5812982877, 5812979119, 5812983438, 1234809285, 1203779116] AS TARGETS
MATCH (n:Neuron)-[:Contains]->(:SynapseSet)-[Contains]->(s:Synapse)
WHERE s.type = "post" AND n.bodyId IN TARGETS AND s.location.x >= 10881 AND s.location.x <= 13228
AND s.location.y >= 12510 AND s.location.y <= 21400
AND s.location.z >= 13210 AND s.location.z<= 19390
AND s.vACA IS NULL AND s.CA IS NULL
RETURN DISTINCT s.location.x AS X, s.location.y AS Y, s.location.z AS Z ORDER BY s.location.x DESC
UNION
WITH [5812983199, 5812982877, 5812979119, 5812983438, 1234809285, 1203779116] AS TARGETS
MATCH (n:Neuron)-[:Contains]->(:SynapseSet)-[Contains]->(s:Synapse)
WHERE s.type = "post" AND n.bodyId IN TARGETS AND s.location.x >= 11160 AND s.location.x <= 11599
AND s.location.y >= 18170 AND s.location.y <= 19010
AND s.location.z >= 11820 AND s.location.z<= 12450
AND s.vACA IS NULL AND s.CA IS NULL
RETURN DISTINCT s.location.x AS X, s.location.y AS Y, s.location.z AS Z ORDER BY s.location.x DESC
UNION
WITH [5812983199, 5812982877, 5812979119, 5812983438, 1234809285, 1203779116] AS TARGETS
MATCH (n:Neuron)-[:Contains]->(:SynapseSet)-[Contains]->(s:Synapse)
WHERE s.type = "post" AND n.bodyId IN TARGETS AND s.location.x >= 13225 AND s.location.x <= 19365
AND s.location.y >= 13535 AND s.location.y <= 20500
AND s.location.z >= 11080 AND s.location.z<= 20295
AND s.vACA IS NULL AND s.CA IS NULL
RETURN DISTINCT s.location.x AS X, s.location.y AS Y, s.location.z AS Z ORDER BY s.location.x DESC
UNION
WITH [5812983199, 5812982877, 5812979119, 5812983438, 1234809285, 1203779116] AS TARGETS
MATCH (n:Neuron)-[:Contains]->(:SynapseSet)-[Contains]->(s:Synapse)
WHERE s.type = "post" AND n.bodyId IN TARGETS AND s.location.x >= 13920 AND s.location.x <= 14830
AND s.location.y >= 12285 AND s.location.y <= 12045
AND s.location.z >= 19100 AND s.location.z<= 19400
AND s.vACA IS NULL AND s.CA IS NULL
RETURN DISTINCT s.location.x AS X, s.location.y AS Y, s.location.z AS Z ORDER BY s.location.x DESC
```