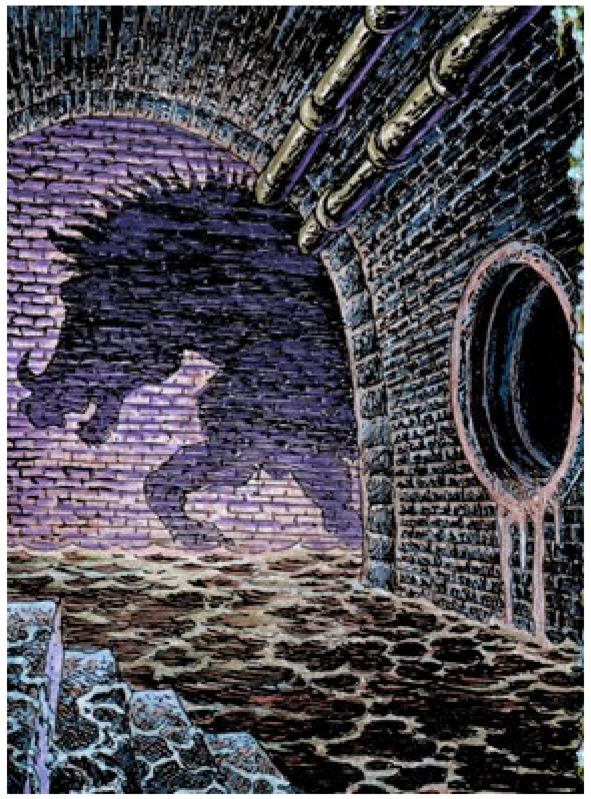
Mark Needham

23 Oct 2014 · NEO4J

Neo4j: Cypher - Avoiding the Eager

Although I love how easy Cypher's **LOAD CSV** command makes it to get data into Neo4j, it currently breaks **the rule of least surprise** in the way it eagerly loads in all rows for some queries even those using **periodic commit**.



This is something that my colleague Michael noted in the **second** of his blog posts explaining **how to use LOAD CSV successfully**:

> The **biggest issue** that people ran into, even when following the advice I gave earlier, was that for large imports of more than one million rows, Cypher ran into an out-of-memory situation. That was **not related to commit sizes**, so it happened even with PERIODIC COMMIT of small batches.

I recently spent a few days importing data into Neo4j on a Windows machine with 4GB RAM so I was seeing this problem even earlier than Michael suggested.

Michael explains how to work out whether your query is suffering from unexpected eager evaluation:

> If you profile that query you see that there is an "Eager" step in the query plan. That is where the "pull in all data" happens.

You can profile queries by prefixing the word 'PROFILE'. You'll need to run your query in the console of */webadmin* in your web

Mark Needham

browser or with the **Neo4j shell**.

I did this for my queries and was able to identify query patterns which get evaluated eagerly and in some cases we can work around it.

We'll use the **Northwind data set** to demonstrate how the Eager pipe can sneak into our queries but keep in mind that this data set is sufficiently small to not cause issues.

This is what a row in the file looks like:

BASH

```bash
$ head -n 2 data/customerDb.csv
OrderID,CustomerID,EmployeeID,OrderDate,RequiredDate,ShippedDate,ShipVia,Freight,ShipName,ShipAddress,ShipCity,ShipRegion,ShipPostalCode,ShipCountry,CustomerID,CustomerCompanyName,ContactName,ContactTitle,Address,City,Region,PostalCode,Country,Phone,Fax,EmployeeID,LastName,FirstName,Title,TitleOfCourtesy,BirthDate,HireDate,Address,City,Region,PostalCode,Country,HomePhone,Extension,Photo,Notes,ReportsTo,PhotoPath,OrderID,ProductID,UnitPrice,Quantity,Discount,ProductID,ProductName,SupplierID,CategoryID,QuantityPerUnit,UnitPrice,UnitsInStock,UnitsOnOrder,ReorderLevel,Discontinued,SupplierID,SupplierCompanyName,ContactName,ContactTitle,Address,City,Region,PostalCode,Country,Phone,Fax,HomePage,CategoryID,CategoryName,Description,Picture
10248,VINET,5,1996-07-04,1996-08-01,1996-07-16,3,32.38,Vins et alcools Chevalier,59 rue de l'Abbaye,Reims,,51100,France,VINET,Vins et alcools Chevalier,Paul Henriot,Accounting Manager,59 rue de l'Abbaye,Reims,,51100,France,26.47.15.10,26.47.15.11,5,Buchanan,Steven,Sales Manager,Mr.,1955-03-04,1993-10-17,14 Garrett Hill,London,,SW1 8JR,UK,(71) 555-4848,3453,\x,"Steven Buchanan graduated from St. Andrews University, Scotland, with a BSC degree in 1976.  Upon joining the company as a sales representative in 1992, he spent 6 months in an orientation program at the Seattle office and then returned to his permanent post in London.  He was promoted to sales manager in March 1993.  Mr. Buchanan has completed the courses ""Successful Telemarketing"" and ""International Sales Management.""  He is fluent in French.",2,http://accweb/emmployees/buchanan.bmp,10248,11,14,12,0,11,Queso Cabrales,5,4,1 kg pkg.,21,22,30,30,0,5,Cooperativa de Quesos 'Las Cabras',Antonio del Valle Saavedra,Export Administrator,Calle del Rosal 4,Oviedo,Asturias,33007,Spain,(98) 598 76 54,,,4,Dairy Products,Cheeses,\x
```

MERGE, MERGE, MERGE

The first thing we want to do is create a node for each employee and each order and then create a relationship between them.

We might start with the following query:

```
USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS FROM "file:/Users/markneedham/projects/neo4j-northwind/data/customerDb.csv" AS row
MERGE (employee:Employee {employeeId: row.EmployeeID})
MERGE (order:Order {orderId: row.OrderID})
MERGE (employee)-[:SOLD]->(order)
```

This does the job but if we profile the query like so…

```
PROFILE LOAD CSV WITH HEADERS FROM "file:/Users/markneedham/projects/neo4j-northwind/data/customerDb.csv" AS row
WITH row LIMIT 0
```

Mark Needham

```
MERGE (employee:Employee {employeeId: row.EmployeeID})
MERGE (order:Order {orderId: row.OrderID})
MERGE (employee)-[:SOLD]->(order)
```

…we'll notice an 'Eager' lurking on the third line:

```
==> +----------------+------+--------+----------------------------------+-----------------------------------------+
==> |       Operator | Rows | DbHits |                      Identifiers |                                   Other |
==> +----------------+------+--------+----------------------------------+-----------------------------------------+
==> |    EmptyResult |    0 |      0 |                                  |                                         |
==> | UpdateGraph(0) |    0 |      0 |    employee, order,   UNNAMED216 |                            MergePattern |
==> |          Eager |    0 |      0 |                                  |                                         |
==> | UpdateGraph(1) |    0 |      0 | employee, employee, order, order | MergeNode; :Employee; MergeNode; :Order |
==> |          Slice |    0 |      0 |                                  |                            {  AUTOINT0} |
==> |        LoadCSV |    1 |      0 |                              row |                                         |
==> +----------------+------+--------+----------------------------------+-----------------------------------------+
```

You'll notice that when we profile each query we're stripping off the periodic commit section and adding a 'WITH row LIMIT 0'. This allows us to generate enough of the query plan to identify the 'Eager' operator without actually importing any data.

We want to split that query into two so it can be processed in a non eager manner:

```
USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS FROM "file:/Users/markneedham/projects/neo4j-northwind/data/customerDb.csv" AS row
WITH row LIMIT 0
MERGE (employee:Employee {employeeId: row.EmployeeID})
MERGE (order:Order {orderId: row.OrderID})
```

```
==> +-------------+------+--------+----------------------------------+-----------------------------------------+
==> |    Operator | Rows | DbHits |                      Identifiers |                                   Other |
==> +-------------+------+--------+----------------------------------+-----------------------------------------+
==> | EmptyResult |    0 |      0 |                                  |                                         |
==> | UpdateGraph |    0 |      0 | employee, employee, order, order | MergeNode; :Employee; MergeNode; :Order |
==> |       Slice |    0 |      0 |                                  |                            {  AUTOINT0} |
==> |     LoadCSV |    1 |      0 |                              row |                                         |
==> +-------------+------+--------+----------------------------------+-----------------------------------------+
```

Now that we've created the employees and orders we can join them together:

```
USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS FROM "file:/Users/markneedham/projects/neo4j-northwind/data/customerDb.csv" AS row
MATCH (employee:Employee {employeeId: row.EmployeeID})
```

Mark Needham

```
MATCH (order:Order {orderId: row.OrderID})
MERGE (employee)-[:SOLD]->(order)
```

```
==> +----------------+------+--------+-------------------------------+-----------------------------------------------------------+
==> |       Operator | Rows | DbHits |                   Identifiers |                                                     Other |
==> +----------------+------+--------+-------------------------------+-----------------------------------------------------------+
==> |    EmptyResult |    0 |      0 |                               |                                                           |
==> |    UpdateGraph |    0 |      0 | employee, order,   UNNAMED216 |                                              MergePattern |
==> |      Filter(0) |    0 |      0 |                               |          Property(order,orderId) == Property(row,OrderID) |
==> | NodeByLabel(0) |    0 |      0 |                  order, order |                                                    :Order |
==> |      Filter(1) |    0 |      0 |                               | Property(employee,employeeId) == Property(row,EmployeeID) |
==> | NodeByLabel(1) |    0 |      0 |            employee, employee |                                                 :Employee |
==> |          Slice |    0 |      0 |                               |                                              {  AUTOINT0} |
==> |        LoadCSV |    1 |      0 |                           row |                                                           |
==> +----------------+------+--------+-------------------------------+-----------------------------------------------------------+
```

Not an Eager in sight!

MATCH, MATCH, MATCH, MERGE, MERGE

If we fast forward a few steps we may now have refactored our import script to the point where we create our nodes in one query and the relationships in another query.

Our create query works as expected:

```
USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS FROM "file:/Users/markneedham/projects/neo4j-northwind/data/customerDb.csv" AS row
MERGE (employee:Employee {employeeId: row.EmployeeID})
MERGE (order:Order {orderId: row.OrderID})
MERGE (product:Product {productId: row.ProductID})
```

```
==> +-------------+------+--------+----------------------------------------------------
+--------------------------------------------------------------+
==> |    Operator | Rows | DbHits |                                        Identifiers |
Other |
==> +-------------+------+--------+----------------------------------------------------
+--------------------------------------------------------------+
==> | EmptyResult |    0 |      0 |                                                    |
|
==> | UpdateGraph |    0 |      0 | employee, employee, order, order, product, product | MergeNode; :Employee; MergeNode; :Order; MergeNode;
:Product |
==> |       Slice |    0 |      0 |                                                    |                                                 {
AUTOINT0} |
==> |     LoadCSV |    1 |      0 |                                                row |
```

Mark Needham

```
|
==> +-------------+------+--------+-----------------------------------------------
+-------------------------------------------------------------
```

We've now got employees, products and orders in the graph. Now let's create relationships between the trio:

```
USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS FROM "file:/Users/markneedham/projects/neo4j-northwind/data/customerDb.csv" AS row
MATCH (employee:Employee {employeeId: row.EmployeeID})
MATCH (order:Order {orderId: row.OrderID})
MATCH (product:Product {productId: row.ProductID})
MERGE (employee)-[:SOLD]->(order)
MERGE (order)-[:PRODUCT]->(product)
```

If we profile that we'll notice Eager has sneaked in again!

```
==> +----------------+------+--------+--------------------------------+------------------------------------------------------------+
==> |       Operator | Rows | DbHits |                    Identifiers |                                                      Other |
==> +----------------+------+--------+--------------------------------+------------------------------------------------------------+
==> |    EmptyResult |    0 |      0 |                                |                                                            |
==> | UpdateGraph(0) |    0 |      0 |  order, product,   UNNAMED318 |                                               MergePattern |
==> |          Eager |    0 |      0 |                                |                                                            |
==> | UpdateGraph(1) |    0 |      0 | employee, order,   UNNAMED287 |                                               MergePattern |
==> |      Filter(0) |    0 |      0 |                                |    Property(product,productId) == Property(row,ProductID) |
==> | NodeByLabel(0) |    0 |      0 |               product, product |                                                   :Product |
==> |      Filter(1) |    0 |      0 |                                |          Property(order,orderId) == Property(row,OrderID) |
==> | NodeByLabel(1) |    0 |      0 |                   order, order |                                                     :Order |
==> |      Filter(2) |    0 |      0 |                                | Property(employee,employeeId) == Property(row,EmployeeID) |
==> | NodeByLabel(2) |    0 |      0 |             employee, employee |                                                  :Employee |
==> |          Slice |    0 |      0 |                                |                                              {  AUTOINT0} |
==> |        LoadCSV |    1 |      0 |                            row |                                                            |
==> +----------------+------+--------+--------------------------------+------------------------------------------------------------+
```

In this case the Eager happens on our second call to MERGE as Michael identified in his post:

> The issue is that within a single Cypher statement you have to isolate changes that affect matches further on, e.g. when you CREATE nodes with a label that are suddenly matched by a later MATCH or MERGE operation.

We can work around the problem in this case by having separate queries to create the relationships:

```
LOAD CSV WITH HEADERS FROM "file:/Users/markneedham/projects/neo4j-northwind/data/customerDb.csv" AS row
MATCH (employee:Employee {employeeId: row.EmployeeID})
```

Mark Needham

```
MATCH (order:Order {orderId: row.OrderID})
MERGE (employee)-[:SOLD]->(order)
```

```
==> +----------------+------+--------+-------------------------------+-----------------------------------------------------------+
==> |       Operator | Rows | DbHits |                   Identifiers |                                                     Other |
==> +----------------+------+--------+-------------------------------+-----------------------------------------------------------+
==> |    EmptyResult |    0 |      0 |                               |                                                           |
==> |    UpdateGraph |    0 |      0 | employee, order,   UNNAMED236 |                                              MergePattern |
==> |      Filter(0) |    0 |      0 |                               |          Property(order,orderId) == Property(row,OrderID) |
==> | NodeByLabel(0) |    0 |      0 |                  order, order |                                                    :Order |
==> |      Filter(1) |    0 |      0 |                               | Property(employee,employeeId) == Property(row,EmployeeID) |
==> | NodeByLabel(1) |    0 |      0 |            employee, employee |                                                 :Employee |
==> |          Slice |    0 |      0 |                               |                                              {  AUTOINT0} |
==> |        LoadCSV |    1 |      0 |                           row |                                                           |
==> +----------------+------+--------+-------------------------------+-----------------------------------------------------------+
```

```
USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS FROM "file:/Users/markneedham/projects/neo4j-northwind/data/customerDb.csv" AS row
MATCH (order:Order {orderId: row.OrderID})
MATCH (product:Product {productId: row.ProductID})
MERGE (order)-[:PRODUCT]->(product)
```

```
==> +----------------+------+--------+------------------------------+--------------------------------------------------------+
==> |       Operator | Rows | DbHits |                  Identifiers |                                                  Other |
==> +----------------+------+--------+------------------------------+--------------------------------------------------------+
==> |    EmptyResult |    0 |      0 |                              |                                                        |
==> |    UpdateGraph |    0 |      0 | order, product,   UNNAMED229 |                                           MergePattern |
==> |      Filter(0) |    0 |      0 |                              | Property(product,productId) == Property(row,ProductID) |
==> | NodeByLabel(0) |    0 |      0 |             product, product |                                               :Product |
==> |      Filter(1) |    0 |      0 |                              |       Property(order,orderId) == Property(row,OrderID) |
==> | NodeByLabel(1) |    0 |      0 |                 order, order |                                                 :Order |
==> |          Slice |    0 |      0 |                              |                                           {  AUTOINT0} |
==> |        LoadCSV |    1 |      0 |                          row |                                                        |
==> +----------------+------+--------+------------------------------+--------------------------------------------------------+
```

MERGE, SET

I try to make LOAD CSV scripts as idempotent as possible so that if we add more rows or columns of data to our CSV we can rerun the query without having to recreate everything.

This can lead you towards the following pattern where we're creating suppliers:

Mark Needham

```
USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS FROM "file:/Users/markneedham/projects/neo4j-northwind/data/customerDb.csv" AS row
MERGE (supplier:Supplier {supplierId: row.SupplierID})
SET supplier.companyName = row.SupplierCompanyName
```

We want to ensure that there's only one Supplier with that SupplierID but we might be incrementally adding new properties and decide to just replace everything by using the 'SET' command. If we profile that query, the Eager lurks:

```
==> +----------------+------+--------+--------------------+---------------------+
==> |       Operator | Rows | DbHits |        Identifiers |               Other |
==> +----------------+------+--------+--------------------+---------------------+
==> |    EmptyResult |    0 |      0 |                    |                     |
==> | UpdateGraph(0) |    0 |      0 |                    |         PropertySet |
==> |          Eager |    0 |      0 |                    |                     |
==> | UpdateGraph(1) |    0 |      0 | supplier, supplier | MergeNode; :Supplier |
==> |          Slice |    0 |      0 |                    |         {  AUTOINT0} |
==> |        LoadCSV |    1 |      0 |                row |                     |
==> +----------------+------+--------+--------------------+---------------------+
```

We can work around this at the cost of a bit of duplication using 'ON CREATE SET' and 'ON MATCH SET':

```
USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS FROM "file:/Users/markneedham/projects/neo4j-northwind/data/customerDb.csv" AS row
MERGE (supplier:Supplier {supplierId: row.SupplierID})
ON CREATE SET supplier.companyName = row.SupplierCompanyName
ON MATCH SET supplier.companyName = row.SupplierCompanyName
```

```
==> +-------------+------+--------+--------------------+---------------------+
==> |    Operator | Rows | DbHits |        Identifiers |               Other |
==> +-------------+------+--------+--------------------+---------------------+
==> | EmptyResult |    0 |      0 |                    |                     |
==> | UpdateGraph |    0 |      0 | supplier, supplier | MergeNode; :Supplier |
==> |       Slice |    0 |      0 |                    |         {  AUTOINT0} |
==> |     LoadCSV |    1 |      0 |                row |                     |
==> +-------------+------+--------+--------------------+---------------------+
```

With the data set I've been working with I was able to avoid OutOfMemory exceptions in some cases and reduce the amount of time it took to run the query by a factor of 3 in others.

As time goes on I expect all of these scenarios will be addressed but as of Neo4j 2.1.5 these are the patterns that I've identified as being overly eager.

Mark Needham

If you know of any others do let me know and I can add them to the post or write a second part.

EMAIL

About the author

I'm currently working on short form content at **ClickHouse**. I publish short 5 minute videos showing how to solve data problems on YouTube **@LearnDataWithMark**. I previously worked on graph analytics at **Neo4j**, where I also co-authored the **O'Reilly Graph Algorithms Book** with Amy Hodler.

ALSO ON **MARK NEEDHAM**

Hugging Face: Using `max_length`'s …

2 years ago · 2 comments

In this post we'll learn how to set the maximum token length when using the …

Apache Pinot: Exploring indexing …

3 years ago · 1 comment

In this post we'll learn how to use Apache Pinot indexes on a Chicago Crime dataset.

DuckDB: Generate dummy data with …

2 years ago · 1 comment

In this post we'll learn how to create dummy data with DuckDB user defined …

ClickHouse: Unknown setting …

10 months ago · 1 comment

In this post, we'll learn how to use ClickHouse's allow_nullable_key setting.

Runn Whis

7 mont

In this the rei folks a

Mark Needham

6 Comments

1 Login ▾

G

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

1 Share

Best Newest Oldest

Titli Sarkar
7 years ago edited

Hi Mark,
my csv file is >10GB and it is in the format(vertex1,vertex2,edgeinfo). This is my actual query:
USING PERIODIC COMMIT
LOAD CSV FROM "file:///3cok.scalaProcessed/part-00000" AS line
MERGE (n:MyNode {Name:line[0]})
MERGE (m:MyNode {Name:line[1]})
MERGE (n) -[:TO {dist:line[2]}] -> (m)

Here we are using same name "MyNode" for two merge for line[0] and line[1]. My question is: when we create two MERGEs seperately and then join them later, does using same name affect?

2nd Question: Does the data loads in the disk or in memeory> By default, all data are loaded in the memory in LOAD CSV command. I want to save my graph to disk and later find connected components from it. As my file to too large, loading the full data in disk and processing from there does not work.

2 0 Reply Share ›

A

Ami Tabak
5 years ago

Hi Mark
Great article. Worked for me with a 1 GB containing appx 10M simple relationship records which previously run OOM
Are there any plans to achieve the same by "turning of" at the query level the eager (like a hint mechanism) ?

0 0 Reply Share ›

王

王子傲
8 years ago

Hi,there! I am importing a csv file into noe4j. There was a column whose value is like this: "jack,tom,alice,somebody", I want to split them and let them to have relations with other nodes, like: jack->node1, tom->node1, alice->node1,somebody->node1,Then how can I do this? It would be grateful if you can answer my question ,Thank U!

0 0 Reply Share ›

Mark Needham

Felix Victor Münch
10 years ago

Thanks for this, gave me a hint on what's taking so long with my import. I think a good rule of thumb is just not to be too smart and not to do too many things in one query. But looking for eager operations while profiling hit the nail for me. By the way, if I use EXPLAIN instead of PROFILE I don't have to fiddle around with limits.

0 0 Reply Share ›

D

David Peklak
10 years ago

I am running into the same problem with "MATCH, SET". I have a graph with financial instruments, for each instrument I have an identity node and state nodes that describe properties of the instrument that can change over time, as described here: http://www.neo4j.org/graphg...). The state nodes are connected to the identity nodes by relations that have a "from" and a "to" property indicating the dates between the state was/is valid. If a state is currently valid, I set "to" to 9223372036854775807. But I only want to create a new state node if the state that I import is different from the current state. I do this in several steps, the step where the "eager" sneaks is when I try to set the "to" date on the existing relation from the identity node to the current state node:

```
LOAD CSV WITH HEADERS FROM "my_path/INSTRUMENT.csv" AS line
MATCH (:Instrument_Ident{ident:toInt(line.IDENT)})-[r:STATE{to:9223372036854775807}]->(is:Instrument_State)
WHERE NOT (is.name = line.NAME
AND is.reference = line.REFERENCE
AND is.pointValue = toFloat(line.POINT_VALUE)
)
SET r.to = 20150620
;
```

Any hints on how can change this so that it is not eager?
Thanks,
David

0 0 Reply Share ›

Mark Needham Mod ➜ David Peklak
10 years ago

@David Peklak Hey,

Sorry I didn't see your comment until now. With that query the MATCH is eager if it's looking for a path so I think the only way to make it lazy is if you assigned some sort of identifier on the 'Instrument_State' node e.g. a combination of 9223372036854775807 and line.IDENT