WG3 YTZ-029R1
DM32.2-2018-00085R1

An overview of the recent history of Graph Query Languages

Title	An overview of the recent history of Graph Query Languages
Authors	Tobias Lindaaker, U.S. National Expert
Date	2018-05-14

Contents

1. Summary

The present day graph query languages, Cypher, PGQL, and G-CORE, share a lot of history. This document aims to lay out and clarify that history for the purpose of putting conversations about graph querying in perspective. Where appropriate, references will be made to earlier history of graph querying.

2. References

[Angles2008]	Renzo Angles and Claudio Gutierrez, *“Survey of Graph Database Models”*, ACM Computing Surveys, Vol. 40, February 2008
[PG2010]	Marko A. Rodriguez and Peter Neubauer, “The Graph Traversal Pattern”, April 2010
[Wood2012]	Peter Wood, “Query Languages for Graph Databases”, SIGMOD Record 2012
[Cypher2018]	Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor, *“Cypher: An Evolving Query Language for Property Graphs”*, SIGMOD 2018
[GCORE2018]	Renzo Angles, Marcelo Arenas, Pablo Barceló, Peter Boncz, George Fletcher, Claudio Gutierrez, Tobias Lindaaker, Marcus Paradies, Stefan Plantikow, Juan Sequeda, Oskar van Rest, and Hannes Voigt, *“G-CORE A Core for Future Graph Query Languages”*, SIGMOD 2018
[Cypher9]	openCypher, *“Cypher Query Language Reference”*, Version 9, Nov. 2017

3. Background

Since the early history of Cypher is tightly coupled to that of Neo4j, this document includes some mentions of the early history of Neo4j in order to provide a backdrop for the origins of Cypher.

For earlier generations of graph models and graph query languages, [Angles2008] and [Wood2012] are good resources.

4. Chronological overview

2000	The founders of Neo4j have the idea to model their data as a network.
2001	The core of what later became Neo4j is implemented.

2007	Neo4j is incorporated, spinning out the database product into a dedicated company.
2009	The early team at Neo4j has the idea to use XPath for querying graphs. Gremlin is created based on this idea.
2010	Neo4j and Marko Rodriguez coin the term Property Graph to describe the data model of Neo4j and Tinkerpop/Gremlin. [PG2010]
2011	Cypher invented. First public release with Neo4j 1.4 in July 2011.
2012	Cypher adds support for writing to the graph. Released with Neo4j 1.8 in October 2012.
2012	Neo4j (version 2.0 in December 2012) adds labels and indexes, and Cypher evolves to be properly declarative.
2015	Oracle creates PGQL as a query language for PGX.
2015	Neo4j opens up Cypher under openCypher.
2015	The LDBC starts its Query Language task force.
2016	The LDBC Query Language task force embarks on designing G-CORE.
2017	WG3 starts an Ad-Hoc group for discussing bringing property graph querying capability into SQL.
2017	The LDBC Query Language task force completes the initial design of G-CORE. [GCORE2018]
2018	Publication of formal semantics of Cypher [Cypher2018]

5. The evolution of Gremlin, Cypher, PGQL, and G-CORE

5.1. The early history of Neo4j

The data model that became Neo4j, and later more broadly accepted as the property graph model, was first conceived of in 2000. The founders of Neo4j were building a media asset management system, where the business requirements kept evolving in ways that entailed substantial change to the schema in the relational database used. The system supported tagging media files in one language and searching for it in other languages, necessitating a semantic linguistic dataset. Access rules were intricate, where media producers could license parts of their portfolio differently to different procurers. In order to better support the required flexibility, Neo4j co-founder Peter Neubauer had the idea to model data in the system as a network of connected concepts with captions on those concepts, inspired by the Cocoon data blade for Informix. Prototyping this idea was entrusted to a group of masters students at IIT Bombay. Neo4j co-founder Emil Eifrém spent a week with the group in Bombay, generalising Peter's idea to a model of nodes connected by relationships, with key-value properties on both nodes and relationships. Together with Emil the Bombay group defined a Java API for interacting with the data model, and then implemented a prototype as an abstraction layer on top of a relational database.

While working with this network model greatly improved productivity, performance was insufficient, so Neo4j co-founder Johan Svensson went through the effort of implementing a native database management system for this model. This implementation is what became Neo4j. Neo4j was successfully used as an internal product for a few years. In 2007 the IP of Neo4j was spun out of the parent company into a separate database company.

From the initial prototype to the first public release, the data model of Neo4j comprised of nodes and typed relationships (i.e. edges with a single label), both having properties which are pairs of a key and a simple value. The early versions of Neo4j did not have any indexes, instead the application would build its own search structures within the graph based on a single well known node in the graph - the root node. As this explicit management of search structures was cumbersome for applications, Neo4j 2.0 (released in December 2013) introduced the concept of a set of labels per node on which indexes could be based. Based on a node label, Neo4j indexes a predefined set of properties for nodes with that label.

The resulting graph model with nodes, edges and properties, where edges have exactly one label and nodes have zero or more labels, is the definition of the Neo4j property graph model. The addition of indexes to Neo4j allowed the query language Cypher to become the primary way of interacting with Neo4j, since updates now could focus on the actual data only and did not have to also update the search structures.

5.2. The creation of Gremlin

The initial mode of querying Neo4j was through use of a Java API. The application would embed the database engine and use this API for querying the graph. This mode proved somewhat difficult for users who were accustomed to the model of the database engine being a separate server to which the application connected remotely.

Around this time, the term "NOSQL" had just recently been coined, and a common pattern among these young database engines was to use REST and HTTP for interaction and querying. Early Neo4j employee Tobias Lindaaker (then Ivarsson) was thinking about good ways of interacting with Neo4j over HTTP in order to make Neo4j more approachable to users. One observation was that a lot of queries could be expressed as tree projections from the graph. Typically starting from a given root traversing a spanning tree from there and returning data of the leaves. Based on this observation it seemed likely that a tree-querying language, such as XPath could be used for graph querying. Furthermore the syntax of XPath is very similar to common URI syntax, meaning that XPath queries could naturally be sent as part of the URI in a HTTP GET request. Neo4j co-founder Peter Neubauer liked the idea and engaged a friend from a Neo4j customer, Marko Rodriguez. Within two days Marko had built a prototype of a language that used XPath for querying the graph, and Groovy to provide looping constructs, branching, and computation. This language was the first prototype of Gremlin, and the first public version was available in late November 2009.

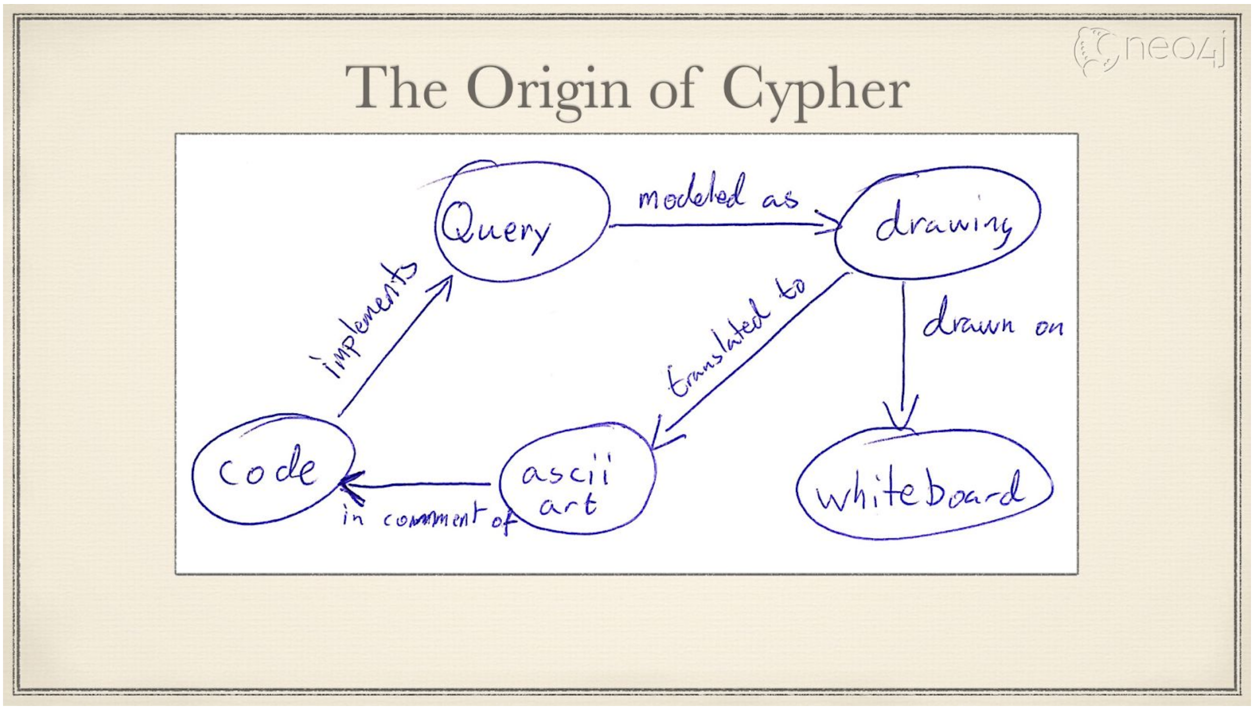
At a later point, Marko found that having two separate language parsers (XPath and Groovy) was more trouble than it was worth and converted Gremlin to an internal DSL in Groovy.

5.3. The creation of Cypher

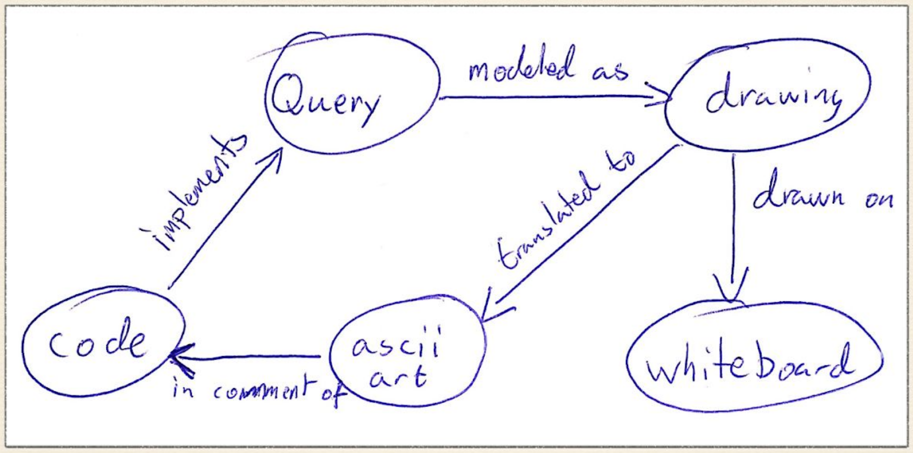
Gremlin did, just like the Neo4j Java API, primarily express a procedural method of querying the database. What it allowed was shorter syntax for expressing queries, and the ability to dynamically express such queries, which was sufficient to solve the problem of being able to query the database from a remote instance on the network. The procedural nature of Gremlin still required the user to know the best way to query the data, an expensive extra burden to put on the application developer. Based on the success of SQL, primarily its ability to separate the declaration of what data to access from the engine’s choice on how to access that data, the Neo4j engineers had a desire for a declarative graph querying language.

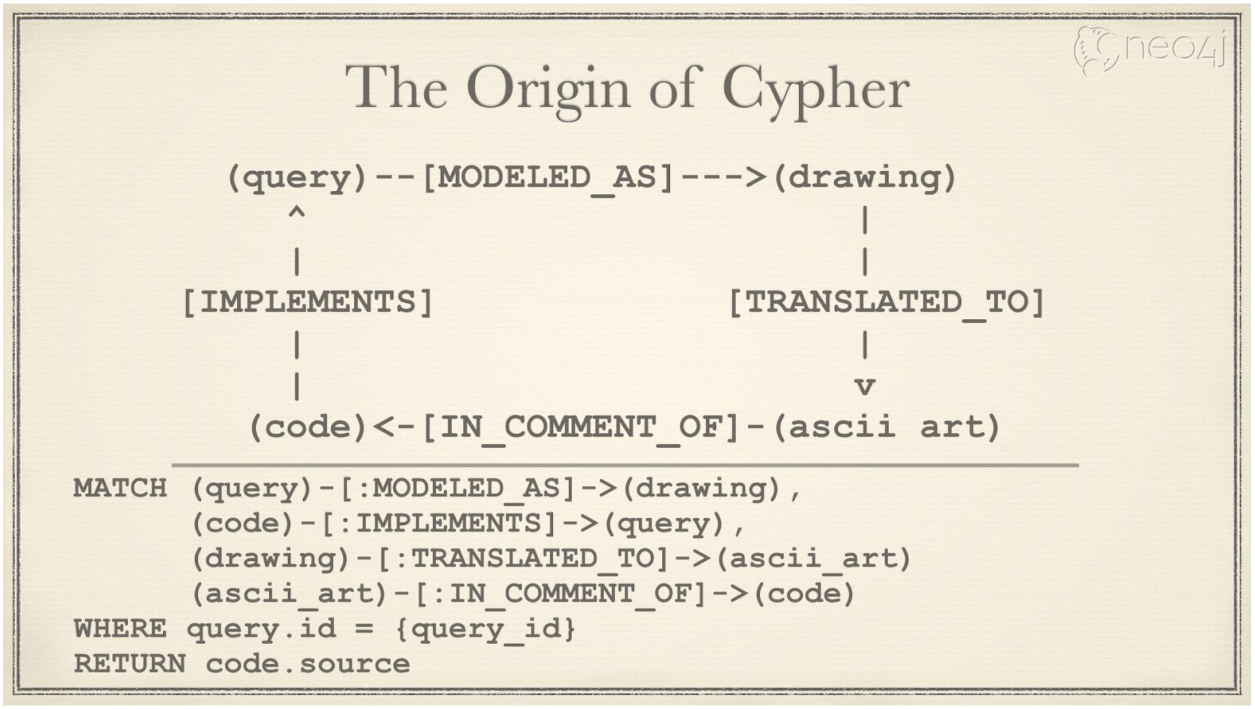
Andrés Taylor joined Neo4j as an engineer in 2010, coming from a background as a SQL DBA, and saw this lack more than anyone. Shortly after joining Neo4j, Andrés started working on a project to create a query language inspired by SQL, but targeted at graph querying. This language, called Cypher, was released with Neo4j version 1.4 in July 2011.

The syntactic foundation of Cypher, to use “ascii art” to describe graph patterns, was based on a practice within the Neo4j engineering team to draw the patterns that a hand-coded query matched as ascii art in source code comments. In order to be able to parse this ascii art, it was simplified from two dimensional drawings to comma separated one dimensional patterns. The following figures describe the evolution of a query from a drawing of the pattern via an ascii art translation of that pattern to a Cypher query expressing the pattern.



The Origin of Cypher





The Origin of Cypher

```
     (query)--[MODELED_AS]--->(drawing)
        ^                          |
        |                          |
   [IMPLEMENTS]             [TRANSLATED_TO]
        |                          |
        |                          v
      (code)<-[IN_COMMENT_OF]-(ascii art)
```

```
MATCH (query)-[:MODELED_AS]->(drawing),
      (code)-[:IMPLEMENTS]->(query),
      (drawing)-[:TRANSLATED_TO]->(ascii_art)
      (ascii_art)-[:IN_COMMENT_OF]->(code)
WHERE query.id = {query_id}
RETURN code.source
```

The first version of Cypher allowed reading of the graph, but required the user to declare the nodes that querying should start from. Based on that set of initial nodes, the query would then be expressed as pattern matching on the graph.

In a later version of Neo4j, version 1.8, released in October 2012, Cypher grew the capability of modifying the graph. The querying still required explicit statement of which nodes the query should start from.

Neo4j version 2.0 in December 2013 introduced the notion of labels that indexes could be based on. With this addition it was made possible for the Cypher execution engine to select which nodes to begin matching the pattern from without explicit guidance from the user.

Through its use in Neo4j, Cypher has been used by a large number of application developers. While the exact extent of utilization is unknown, there have been over eight million downloads of Neo4j since Cypher was introduced into the product, and besides very extensive use of the free open-source edition, there are over 250 commercial customers of Neo4j using Cypher in the construction of production applications, in widely varying industries from IC design to finance to retail.

5.4. openCypher - an open process for evolving and standardizing Cypher

In September 2015 Neo4j decided and publicly announced its intent to open up the governance of the Cypher query language. The new process taking over the governance of Cypher is an open group of implementors of Cypher called the openCypher Implementors Group.

In 2016 the openCypher project published EBNF and ANTLR4 grammars for Cypher features which were felt to be suitable for use in a standard, multi-implementer version of the language. For the same scope, Neo4j contributed a large body of language feature tests as the foundation for the openCypher Technology Compatibility Kit (TCK) under the Apache 2.0 liberal open-source licence. Proposals for language change are also public, and open for submission by anyone. In late 2016 SAP HANA Graph was released using the read-query part of Cypher, and in the course of 2017 two other generally available commercial products, Agens Graph and Redis Graph, have been released with Cypher support. In 2017 a series of public meetings of openCypher implementers, including three meetings in-person, has provided a forum for discussing language changes and agreeing, by a process of consensus, on language extensions.

The openCypher Implementers Group (oCIG) that now controls language changes have decided to use an English-language specification document that is a snapshot of the product documentation for Cypher in Neo4j 3.3.0 (omitting any deprecated or implementation-specific elements not included in openCypher, and incorporating an existing openCypher specification fragment on the data model), plus the openCypher grammar and TCK software artefacts, to collectively form the first specification of the Cypher language under the governance of the oCIG. The language version documented in this way is named Cypher 9 (see [Cypher9]).

5.5. The creation of PGQL

In 2015 Oracle built a graph query language that came to be called PGQL for the PGX engine. PGQL is in many ways similar to Cypher, and early communication with the team behind PGQL indicated that it does indeed draw inspiration from Cypher.

5.6. The creation of G-CORE

The Linked Data Benchmarking Council (LDBC) defines vendor independent benchmarks for graph databases. In the early work on defining these benchmarks, it was noted that there was no standard query language available at the time with good support for expressing graph queries. In order to address this, a task force on query languages was formed with the objective to survey existing graph querying languages and frameworks, identify necessary features of graph querying, then either propose a new query language or propose changes to an existing query language.

Early 2016 a decision was made to design a language rather than propose changes to an existing language. Primarily since this meant that work could continue without restrictions on the model being imposed by a pre-existing language.

The language, G-CORE, that was actually designed by the LDBC Query Language Task Force explicitly maintains syntactic compatibility with Cypher in the cases where G-CORE follows the same semantics as Cypher. Cypher was recognized as a good base, due to it being the syntactic base of other contemporary graph query languages, such as PGQL.

6. Conclusions

It is obvious that both PGQL and G-CORE share a common ancestor in Cypher. Indeed both the syntax and the semantics of these languages are extremely similar. While PGQL is much closer to the early versions of Cypher than G-CORE is, in that they both query graphs and produce tables with almost exactly the same syntax and semantics, G-CORE explicitly in its stated design aims to be syntactically and semantically compatible with Cypher where possible.