Why Graph?

A Look at How Cloud Applications Benefit From Graph Technology

Table of Contents

Introduction

The applications that are changing business today are radically different than ones of prior generations. Today's applications consist of many endpoints including browsers, mobile devices, and/or machines that are geographically distributed, intensely transactional, always available, as well as instantaneously and intelligently responsive no matter the number of users or machines using the application.

Such applications – that DataStax defines as cloud applications – have multi-faceted requirements where their data management is concerned, with success being defined as the application having:

- Continuous availability
- Geographical data distribution
- Operational low latency
- Linear scalability
- Immediate decisiveness
- Functional cohesiveness
- Operational maturity

To meet these requirements, the application's data backend includes numerous pieces of DBMS (DataBase Management System) technology and data-centric engines that must be blended together to act as one – transactional, analytical, search, in-memory, and others – each of which must run in a highly performant way without impacting the other's speed. In other words, the system must be functionally cohesive within a single architecture and also competently manage the varying workloads.

The data flowing through these systems is very complex, ever changing, large in volume, and highly connected (i.e. it possesses a very high number of relationships between data elements). Expectations for the data to be immediately decisive are high, with the need being to instantly answer questions such as:

What products or actions should we recommend to a user based on their preferences and behavioral patterns to maximize sales or user engagement?

Should an initiated transaction be considered fraudulent or malicious based on past user actions and normal patterns of system behavior?

These and similar questions cannot be well serviced by legacy database technology because the number of data relationships coupled with the data distribution, scale, performance, volume, and uptime requirements of the application are not a fit for a relational database. These requirements, however, are addressed natively by a graph database that possesses scale-out and active-everywhere capabilities.

This paper provides an introduction to graph databases and discusses when and where they should be used in today's cloud applications. It also provides a look at DataStax Enterprise (DSE) and DSE Graph, which is a scale-out graph database used to manage complex and highly connected data.

Introduction to Graph Databases

A graph database is used for storing, managing and querying data that is complex and highly connected. A graph database's architecture makes it particularly well suited for exploring data to find commonalities and anomalies among large data volumes and unlocking the value contained in the data's relationships.

Solving Business Problems with Graph

There are a variety of different use cases to which a graph database can be applied and where a graph is a better fit than other DBMS's such as a relational or general NoSQL database.

Recommendation and Personalization

Almost all enterprises need to understand how they can quickly and most effectively influence customers to buy their products and recommend them to others using components in a cloud application such as recommendation, personalization, and network (people or machines) analysis engines.

When utilized properly, graph analysis is the most effective weapon for handling recommendation and personalization tasks in an application and making key business decisions from the value found in the data.

For example, a web retailer needs to recommend products to their customers based on the customer profile as well as previous interactions and purchases. An important point to note is that, to be successful, these actions must be based on the most recent actions taken on the site in the current browsing session, which can all be integrated into one graph.

A graph is well suited to these and similar analytical use cases where recommending products, next actions, or advertising based on a user's data, past behavior, and interactions are important.

Security and Fraud Detection

In a complex and highly interrelated network of users, entities, transactions, events, and interactions, a graph database can help determine which entity, transaction or interaction is fraudulent, poses a security risk, or is a compliance concern. In short, a graph database assists in finding the bad needle in a haystack of relationships and events that involve countless financial interactions.

A graph database is a good fit in this domain because most transactions/interactions/etc., cannot be deemed fraudulent or out of security compliance at face value. Such determinations can only be made in the larger context in which they occur (e.g. previous transactions, past customer behavior).

IoT and Networking

The Internet of Things (IoT) is yet another domain area that is replete with graph-sized problems. The IoT use cases most commonly involve devices or machines that generate time-series information such as event and status data.

A graph works well in this case because the streams from individual points create a high degree of complexity when blended together. Further, analytics involved in tasks such as root-cause analysis, involve numerous relationships that form among the data elements and tend to be of much greater interest when examined collectively than reviewed in isolation.

A graph is also a good model for managing network assets (with their properties or configurations) and how they relate to each other over time. A graph can be used to manage and monitor the network, optimize resource allocation, detect and fix problems, and more.

Master Data Management

A company must understand the data relationships across its multiple business units to create a holistic view of its customers. A graph model is the best way to consolidate that disparate data for use by both business intelligence (BI) tools and other business applications. Other examples include product catalogs, which often have complex hierarchical structures and are overlaid by taxonomies to capture composition or other relationships. In those cases, the data relationships are complex enough to warrant a graph database.

Another interesting facet to this is customer data and the access rights, entitlements, etc., to that data. Many companies struggle with their authentication systems and understanding the inter-workings of their data protection paradigm due to the increasing complexity of data.

Comparing Graph with Other Technologies

Because of its focus on data relationships, it's natural to wonder how a graph database differs from other popular database technologies and exactly when a graph database should be used.

Comparing Graph with an RDBMS

An RDBMS (Relational DataBase Management System) and graph database are similar in that they involve data that contains connections or relationships between data elements. From a data model perspective, their components have the following surface level similarities:

	RDBMS	Graph DB
An identifiable "something" or object to keep track of	Entity	Vertex
A connection or reference between two objects	Relationship	Edge
A characteristic of an object	Attribute	Property

Foundationally, an RDBMS and graph database differ in the underlying engine each uses to store and access data. There are also some data modeling features that RDBMS's and graphs don't share such as graph databases allowing characteristics (i.e. properties) to be assigned to edges, whereas RDBMS relationships have no such ability.

Another key difference between a graph database and an RDBMS is how relationships between entities/vertexes are prioritized and managed. While an RDBMS uses mechanisms like foreign keys to connect entities in a secondary fashion, edges (the relationships) in a graph database are of first order importance.

In other words, relationships are explicitly embedded in a graph data model. Essentially, a graph-shaped business problem is one in which the concern is with the relationships (edges) among entities (vertexes) than with the entities in isolation.

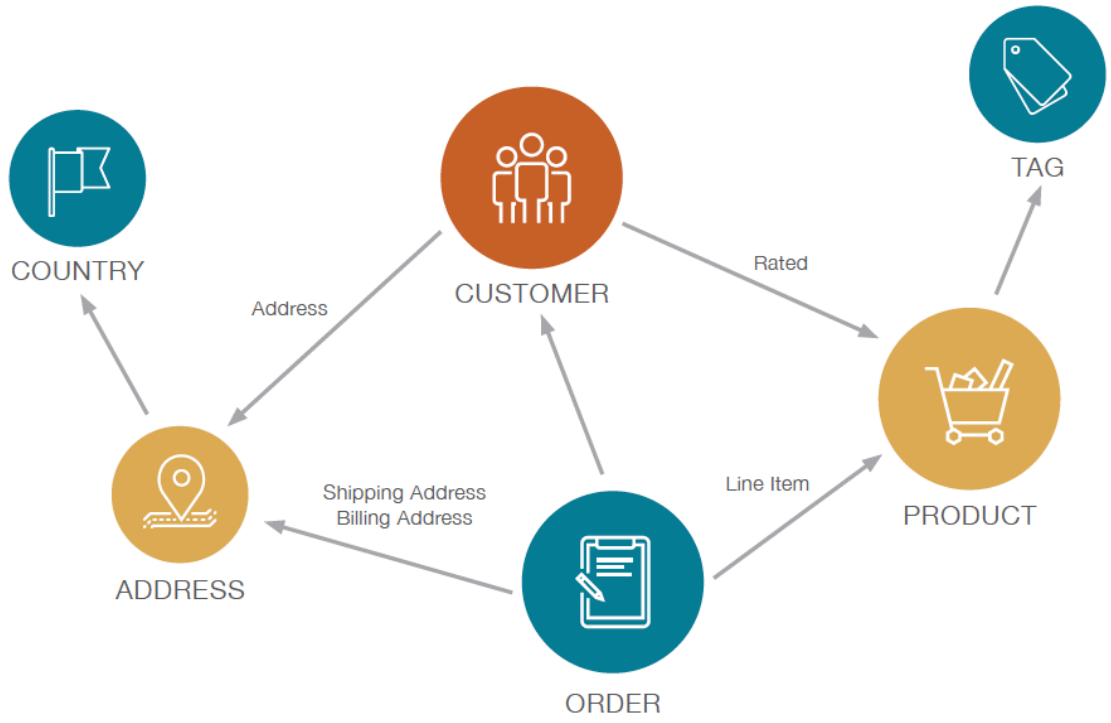


Figure 1 – A simple graph data model.

Because of this, a graph database can be likened to a pre-joined RDBMS. As such, a graph database is more scalable and performant than an RDBMS when it comes to complex data that is highly connected (e.g. millions or billions of relationships).

Unlike most other ways of representing data, graphs are foundationally designed to express relatedness. This allows graph databases to uncover patterns that are difficult to detect when using traditional representations, such as RDBMS tables.

While an RDBMS may indicate *that* objects are connected, a graph database shows *how* objects are connected.

The following comparison grid can be used to help in the decision making process of whether to use an RDBMS or a graph database like DSE Graph for a particular use case:

RDBMS	DSE Graph
Simple to moderate data complexity	Heavy data complexity
Hundreds of potential relationships	Hundreds of thousands to millions or billions of potential relationships

Moderate JOIN operations with good performance	Heavy to extreme JOIN operations required
Infrequent to no data model changes	Constantly changing and evolving data model
Static to semi-static data changes	Dynamic and constantly changing data
Primarily structured data	Structured and unstructured data
Nested or complex transactions	Simple transactions
Always strongly consistent	Tunable consistency (eventual to strong)
Moderate incoming data velocity	High incoming data velocity (e.g. IoT)
High availability (handled with failover)	Continuous availability (no downtime)
Centralized application that is location dependent (e.g. single location), especially for write operations and not just read	Distributed application that is location independent (multiple locations involving multiple data centers and/or clouds) for write and read operations
Scale up for increased performance	Scale out for increased performance

A graph database like DSE Graph will most times be better than an RDBMS when it comes to identifying commonalities and anomalies in large, complex, and highly connected datasets that are maintained in cloud applications.

From an interface perspective, as SQL is to an RDBMS, *Gremlin* is to a graph database. Gremlin is the open source standard language for all graph databases and is part of the Apache TinkerPop™ graph framework. While different than SQL, Gremlin is an extremely flexible, expressive, and easy-to-learn language that supports both transactional and analytical operations.

From a language comparison perspective, one indicator that a graph database is a better choice than an RDBMS for a target use case is consistently seeing large and non-performant SQL JOIN queries being needed to satisfy application queries. A comparative example between the SQL needed for a recommendation query and the same query accomplished in Gremlin can be found in Appendix B of this paper.

Comparing Graph with NoSQL Databases

The primary difference between a graph data model and those used by other NoSQL databases is that a graph model is purposely built to handle high data complexity and connectedness whereas other NoSQL models are designed to manage data that is simpler in its format and relationships.

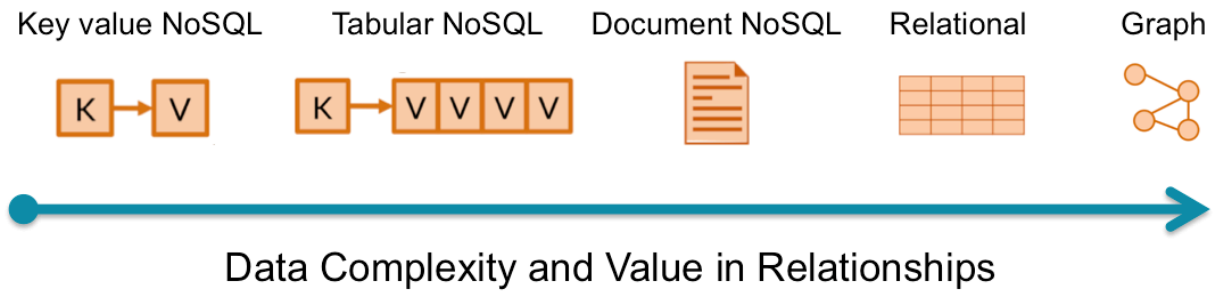


Figure 2 – The data model continuum represented by complexity and data connectedness.

For example, the following comparison grid can be used to help determine when a tabular data model, such as the one found in Apache Cassandra™, should be used vs. a graph data model:

Cassandra Tabular Data Model	Graph Model
Little to no value in data object relationships	Great value in data object relationships
Manual data denormalization easy	Manual data denormalization too complex
Data rarely joined together. If joins occur (e.g. with Spark), performance is acceptable.	Data constantly connected and used to produce end result in performant manner
Write/read heavy	Read heavy; write moderate

A Look at DSE Graph

The requirements of cloud applications and limitations in existing databases leave IT organizations with no choice but to try and create cobbled-together architectures that consist of multiple technologies. Inevitably, these not only end up failing to fully meet their application's requirements, but also prove difficult to develop against, administer, and are cost prohibitive. The solution to this problem is a platform capable of future-proofing the data management requirements of modern cloud applications that need to manage complex and highly connected data.

DataStax Enterprise meets these requirements by delivering a comprehensive data management layer with its unique always-on architecture that accelerates the ability of enterprises, government agencies, and systems integrators to power their exploding number of cloud applications. DSE well serves cloud applications that require data distribution across datacenters and clouds, through the use of its secure, operationally simple platform that is built on Apache Cassandra.

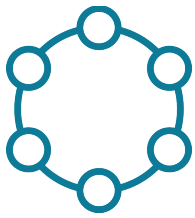
A key component of DSE is DSE Graph, which is part of DSE's multi-model platform. DSE Graph is a graph database built for cloud applications that need to manage complex data and its many relationships. DSE Graph delivers continuous uptime along with predictable performance and scale, while remaining operationally simple to manage.

Built with Apache TinkerPop

Apache TinkerPop is an open source graph computing framework that enables database and data analytic systems to offer graph computing capabilities to their users. The Gremlin graph traversal language is the primary means by which users interface with graph databases that use TinkerPop.

Along with the Gremlin language and virtual machine, TinkerPop provides various supporting tools such as Gremlin Server, data bulk loaders, graph algorithms, visualization tool connectors, and more. Being Apache TinkerPop-enabled, DSE Graph is able to append sophisticated, *standardized* graph computing features to its core foundation and avoids proprietary vendor lock in.

Integrated Deeply with Apache Cassandra



DSE Graph utilizes an enterprise-certified version of Apache Cassandra for its persistent datastore. Unlike traditional RDBMS's and some NoSQL databases, Cassandra sports a masterless "ring" design where every node in a database cluster operates independently with respect to database operations.

Because of its deep integration with Cassandra, DSE Graph inherits all of Cassandra's key benefits including constant uptime, write/read/active-everywhere functionality, linear scalability, predictable low-latency response times, and operational maturity. To that foundation, DSE Graph adds other performance-enhancing capabilities that include an adaptive query optimizer, locality-driven graph data partitioner, distributed query execution engine, and various graph-specific index structures.

Inspired by Titan

DSE Graph's design is inspired by the open source Titan graph database, which is used by many well-known enterprises such as Amazon and others. Titan is a scale-out graph database that has a pluggable storage back end option that allows it to persist data to a variety of databases including Cassandra, HBase, and others.

While DSE Graph uses Titan as a model, it is a completely different set of software that goes much further than Titan's basic scale-out capabilities by deeply integrating with Cassandra, providing key improvements over Titan's core (e.g. automatic data consistency) and including additional commercial software functionality.

Ready for Enterprise Deployments

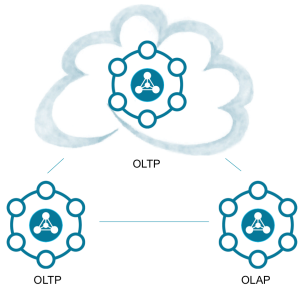
DSE Graph is built on the foundation of TinkerPop and Apache Cassandra and therefore inherits all of the benefits that each open source project provides. Further, DSE Graph incorporates all of the enterprise-class extensions found in DSE that pick up where each open source project leaves off.

These benefits include advanced security protection, built-in analytics and enterprise search functionality, integration with external Hadoop vendors such as Cloudera and HortonWorks, and visual management, monitoring, and development tooling.

Also, as part of the DSE platform, DSE Graph includes around-the-clock support from the graph experts at DataStax, which delivers the confidence needed to run and maintain production graph systems at scale. Support also includes formal end-of-life policies, certified software updates, hot-fixes, and bug escalation privileges.

Automatic Handling of Workload and ETL Management

Effectively running analytics and search operations on transactional data typically causes workload contention in both legacy and NoSQL databases and requires a sharding approach where data is manually extracted and loaded (ETL) into different databases for different workloads. The application is then responsible for targeting a particular database for the right workload.



With DSE Graph, workload management and ETL is built into DSE so everything is automatically handled by the platform. Running multiple workloads is easy and can be carried out in the same database cluster, with the separation of workloads existing in either the same physical location or multiple separate locations including one or more cloud vendors (i.e. hybrid cloud).

Data is automatically replicated between the nodes devoted to differing workloads (e.g. transactional and analytical) so there is no need to carry out manual and error-prone ETL functions.

Solving Business Problems with Multi-Model Support

Because cloud applications involve numerous components that can differ in their data model support requirements, a database that provides adaptive data management (or multi-model) capabilities will deliver a simpler and more agile solution for quickly bringing cloud applications to market.

DSE Graph is part of DSE's multi-model platform, which provides support for Cassandra's key-value and tabular data models, JSON, and graph. Because data from all models are stored in a single persistence layer, each data model inherits Cassandra's benefits as well as DSE's enterprise-grade functionality.

A Full Graph Development and Management Solution

DSE Graph delivers a full solution for developing and managing the graph components of cloud applications. Developers can utilize DataStax Studio, which is a visual web-based development tool, for visualizing graphs and querying their graph databases. Developers can also use integrated drivers from DataStax that contain support for the Gremlin graph language as well as all other API's used in DataStax Enterprise (e.g. CQL for Cassandra).

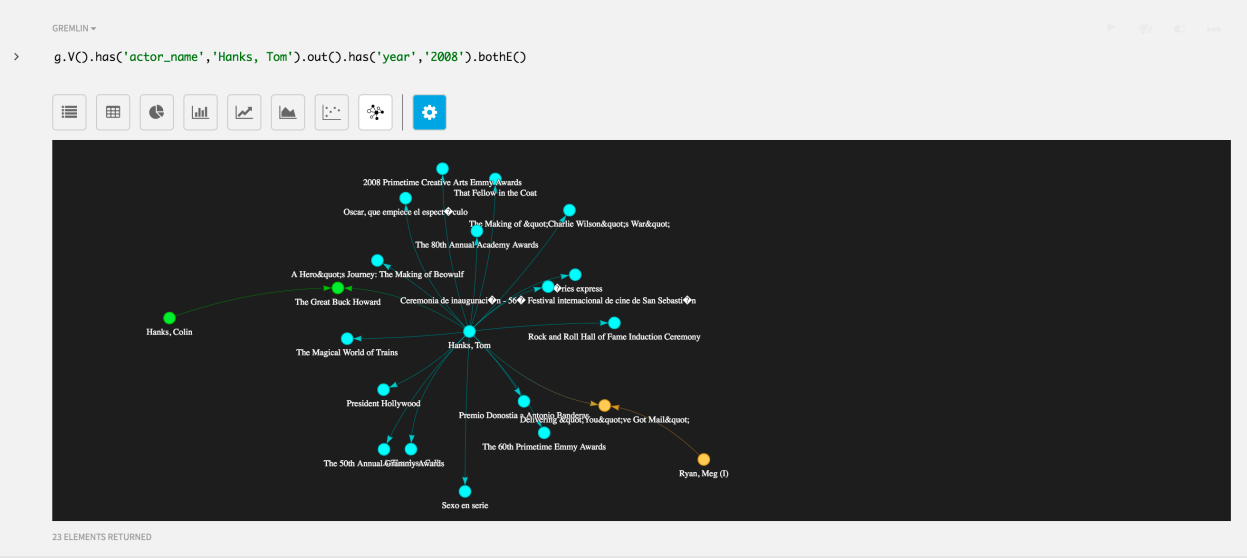


Figure 3 – Visualizing graph data with DataStax Studio.

Administrators and operators can use DataStax OpsCenter to provision, manage, and monitor database clusters that contain DSE Graph and therefore have one tool vs. many to manage all aspects of their multi-model database platform.

Conclusions

Today's cloud applications include data that is constantly changing, large in size, and highly connected. A graph database is best at solving the various business problems that need to make use of this data and is the most capable technology for extracting value from it so key business decisions can be quickly made.

DSE Graph is purpose built for managing data that is complex and sports high degrees of connectedness. As part of DataStax Enterprise, DSE Graph delivers constant uptime along with fast performance and scale for cloud applications that deal with complex and constantly changing data, while remaining very simple to manage.

For more information about DataStax Enterprise and downloads of DataStax software, visit www.datastax.com.

About DataStax

DataStax, the leading provider of database software for cloud applications, accelerates the ability of enterprises, government agencies, and systems integrators to power the exploding number of cloud applications that require data distribution across datacenters and clouds, by using our secure, operationally simple platform built on Apache Cassandra™. With more than 500 customers in over 50 countries, DataStax is the database technology of choice for the world's most innovative companies, such as Netflix, Safeway, ING, Adobe, Intuit, Target and eBay. Based in Santa Clara, Calif., DataStax is backed by industry-leading investors including Comcast Ventures, Crosslink Capital, Lightspeed Venture Partners, Kleiner Perkins Caufield & Byers, Meritech Capital, Premji Invest and Scale Venture Partners. For more information, visit DataStax.com or follow us on @DataStax. 05.02.16

Appendix A – Comparison Between DSE Graph and Other Graph Databases

Below is a quick at-a-glance comparison between DSE Graph and other existing mainstream graph databases.

	DSE Graph	Neo4J	Titan (using backend like Amazon DynamoDB, HBase, etc.)	Titan (using Cassandra backend)
Open Development Language	Yes	Yes	Yes	Yes
Architecture	Masterless	Master-Slave	Master-Slave	Masterless
Consistency Model	Tunable	Tunable	Not tunable	Tunable
Scaling Method	Scale out read/writes	Scale up	Scale out for reads	Scale out reads/writes
Write, Read, Active-Anywhere	Yes	No	No	Yes
Auto Multi-Data Center and Cloud Zone Support	Yes	No	No	Yes
Advanced Security Features	Yes	No	No	No
Integrated Analytics	Yes	No	No	No
Integrated Search	Yes	No	No	No
Automatic workload management and ETL not needed between OLTP, OLAP, and Search systems	Yes	No	No	No
Integrated Multi-Model Platform	Yes	No	No	No

Appendix B – Comparison Between SQL and Gremlin for Recommendation Query

The following compares the SQL needed for a recommendation query and the same request coded with Gremlin.

SQL:

```
SELECT TOP (5) [t14].[ProductName]
    FROM (SELECT COUNT(*) AS [value],
                 [t13].[ProductName]
            FROM [customers] AS [t0]
     CROSS APPLY (SELECT [t9].[ProductName]
                    FROM [orders] AS [t1]
              CROSS JOIN [order details] AS [t2]
              INNER JOIN [products] AS [t3]
                      ON [t3].[ProductID] = [t2].[ProductID]
              CROSS JOIN [order details] AS [t4]
              INNER JOIN [orders] AS [t5]
                      ON [t5].[OrderID] = [t4].[OrderID]
               LEFT JOIN [customers] AS [t6]
                      ON [t6].[CustomerID] = [t5].[CustomerID]
              CROSS JOIN ([orders] AS [t7]
                          CROSS JOIN [order details] AS [t8]
                          INNER JOIN [products] AS [t9]
                                  ON [t9].[ProductID] = [t8].[ProductID])
                   WHERE NOT EXISTS(SELECT NULL AS [EMPTY]
                                      FROM [orders] AS [t10]
                                CROSS JOIN [order details] AS [t11]
                                INNER JOIN [products] AS [t12]
                                        ON [t12].[ProductID] = [t11].[ProductID]
                                     WHERE [t9].[ProductID] = [t12].[ProductID]
                                       AND [t10].[CustomerID] = [t0].[CustomerID]
                                       AND [t11].[OrderID] = [t10].[OrderID])
                     AND [t6].[CustomerID] <> [t0].[CustomerID]
                     AND [t1].[CustomerID] = [t0].[CustomerID]
                     AND [t2].[OrderID] = [t1].[OrderID]
                     AND [t4].[ProductID] = [t3].[ProductID]
                     AND [t7].[CustomerID] = [t6].[CustomerID]
                     AND [t8].[OrderID] = [t7].[OrderID]) AS [t13]
          WHERE [t0].[CustomerID] = N'ALFKI'
       GROUP BY [t13].[ProductName]) AS [t14]
ORDER BY [t14].[value] DESC
```

Gremlin:

```
g.V().has("customerId","ALFKI").as("customer").
  out("ordered").out("contains").out("is").aggregate("products").
    in("is").in("contains").in("ordered").where(neq("customer")).
  out("ordered").out("contains").out("is").where(not(within("products"))).
  groupCount().by("name").
    order(local).by(values,decr).
    select(keys).limit(5)
```