Advanced Data Management (CSCI 640/490)

Graph Databases

Dr. David Koop
### Time Series Data

- A row of data that consists of a timestamp, a value, optional tags

________

#### Example

<table>
<thead>
<tr>
<th>timestamp</th>
<th>tags</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-07-12T11:51:45Z</td>
<td>&quot;true&quot; &quot;34&quot; &quot;4&quot; &quot;3&quot; saarlouis</td>
<td>465110000</td>
</tr>
<tr>
<td>2016-07-12T11:51:45Z</td>
<td>&quot;true&quot; &quot;34&quot; &quot;-6&quot; &quot;2&quot; saarlouis</td>
<td>0.06196699999999994</td>
</tr>
<tr>
<td>2016-07-12T12:10:00Z</td>
<td>&quot;true&quot; &quot;34&quot; &quot;7&quot; &quot;5&quot; saarlouis</td>
<td>4937000000</td>
</tr>
<tr>
<td>2016-07-12T12:10:00Z</td>
<td>&quot;true&quot; &quot;34&quot; &quot;6&quot; &quot;2&quot; saarlouis</td>
<td>1857300000</td>
</tr>
<tr>
<td>2016-07-12T12:10:00Z</td>
<td>&quot;true&quot; &quot;34&quot; &quot;5&quot; &quot;7&quot; saarlouis</td>
<td>5902300000</td>
</tr>
</tbody>
</table>

[A. Bader, 2017]
Time Series Data

- Metrics: measurements at regular intervals
- Events: measurements that are not gathered at regular intervals
Examples

US Treasury bill contracts

Australian electricity production

Sales of new one–family houses, USA

Annual Canadian Lynx trappings
Examples

**Trend**

- **US Treasury bill contracts**
- **Australian electricity production**
- **Sales of new one-family houses, USA**
- **Annual Canadian Lynx trappings**

[R. J. Hyndman]
Examples

US Treasury bill contracts

Australian electricity production

Sales of new one-family houses, USA

Annual Canadian Lynx trappings

Trend

Trend + Seasonality

[The images depict various time series graphs showing different patterns and trends.]
Examples

Trend

US Treasury bill contracts

Day

price

0 20 40 60 80 100

Trend + Seasonality

Australian electricity production

Year

GWh


Seasonality + Cyclic

Sales of new one-family houses, USA

Time

Total sales


Annual Canadian Lynx trappings

Time

Number trapped

1820 1840 1860 1880 1900 1920

[R. J. Hyndman]
Examples

US Treasury bill contracts

Australian electricity production

Sales of new one-family houses, USA

Annual Canadian Lynx trappings

Trend

Trend + Seasonality

Seasonality + Cyclic

Stationary
Pandas Support for Datetime

- **pd.to_datetime:**
  - convenience method
  - can convert an entire column to datetime
- Has a `NaT` to indicate a missing time value
- Stores in a `numpy.datetime64` format
- **pd.Timestamp:** a wrapper for the `datetime64` objects
Resampling

- Could be
  - downsample: higher frequency to lower frequency
  - upsample: lower frequency to higher frequency
  - neither: e.g. Wednesdays to Fridays

- resample method: e.g. `ts.resample('M').mean()`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>freq</code></td>
<td>String or DateOffset indicating desired resampled frequency (e.g., 'M', 'Smin', or Second(15))</td>
</tr>
<tr>
<td><code>axis</code></td>
<td>Axis to resample on; default axis=0</td>
</tr>
<tr>
<td><code>fill_method</code></td>
<td>How to interpolate when upampling, as in 'ffill' or 'bfill'; by default does no interpolation</td>
</tr>
<tr>
<td><code>closed</code></td>
<td>In downsampling, which end of each interval is closed (inclusive), 'right' or 'left'</td>
</tr>
<tr>
<td><code>label</code></td>
<td>In downsampling, how to label the aggregated result, with the 'right' or 'left' bin edge (e.g., the 9:30 to 9:35 five-minute interval could be labeled 9:30 or 9:35)</td>
</tr>
<tr>
<td><code>loffset</code></td>
<td>Time adjustment to the bin labels, such as '-1s' / Second(-1) to shift the aggregate labels one second earlier</td>
</tr>
<tr>
<td><code>limit</code></td>
<td>When forward or backward filing, the maximum number of periods to fill</td>
</tr>
<tr>
<td><code>kind</code></td>
<td>Aggregate to periods ('period') or timestamps ('timestamp'); defaults to the type of index the time series has</td>
</tr>
<tr>
<td><code>convention</code></td>
<td>When resampling periods, the convention ('start' or 'end') for converting the low-frequency period to high frequency; defaults to 'end'</td>
</tr>
</tbody>
</table>

[D. Koop, CSCI 640/490, Spring 2024]

[W. McKinney, Python for Data Analysis]
Time Series Databases

- Most time series data is heavy **inserts**, few updates
- Also analysis tends to be on ordered data with trends, prediction, etc.
- Can also consider **stream** processing
- Focus on time series allows databases to specialize
- Examples:
  - InfluxDB (noSQL)
  - TimescaleDB (SQL-based)
What is a Time Series Database?

- A DBMS is called TSDB if it can
  - store a row of data that consists of timestamp, value, and optional tags
  - store multiple rows of time series data grouped together
  - can query for rows of data
  - can contain a timestamp or a time range in a query

```
"SELECT * FROM ul1 WHERE time >= '2016-07-12T12:10:00Z'"
```

---

[A. Bader, 2017]
Gorilla Motivation

- Large-scale internet services rely on lots of services and machines
- Want to monitor the health of the systems
- Writes dominate
- Want to detect state transitions
- Must be highly available and fault tolerant
Gorilla Compression

Figure 2: Visualizing the entire compression algorithm. For this example, 48 bytes of values and time stamps are compressed to just under 21 bytes/167 bits.

4.1 Time series compression

In evaluating the feasibility of building an in-memory time series database, we considered several existing compression schemes to reduce the storage overhead. We identified techniques that applied solely to integer data which didn't meet our requirement of storing double precision floating point values. Other techniques operated on a complete dataset but did not support compression over a stream of data as was stored in Gorilla [7, 13]. We also identified lossy time series approximation techniques used in data mining to make the problem set more easily fit within memory [15, 11], but Gorilla is focused on keeping the full resolution representation of data.

Our work was inspired by a compression scheme for floating point data derived in scientific computation. This scheme leveraged XOR comparison with previous values to generate a delta encoding [25, 17].

Gorilla compresses data points within a time series with no additional compression used across time series. Each data point is a pair of 64 bit values representing the time stamp and value at that time. Timestamps and values are compressed separately using information about previous values. The overall compression scheme is visualized in Figure 2, showing how time stamps and values are interleaved in the compressed block.

Figure 2.a illustrates the time series data as a stream consisting of pairs of measurements (values) and time stamps. Gorilla compresses this data stream into blocks, partitioned by time. After a simple header with an aligned time stamp (starting at 2 am, in this example) and storing the first value in a less compressed format, Figure 2.b shows that time stamps are compressed using delta-of-delta compression, described in more detail in Section 4.1.1. As shown in Figure 2.b the time stamp delta of delta is 2. This is stored with a two bit header ('10'), and the value is stored in seven bits, for a total size of just 9 bits. Figure 2.c shows floating-point values are compressed using XOR compression, described in more detail in Section 4.1.2. By XORing the floating point value with the previous value, we find that there is only a single meaningful bit in the XOR. This is then encoded with a two bit header ('11'), encoding that there are eleven leading zeros, a single meaningful bit, and the actual value ('1'). This is stored in fourteen total bits.
Enabling Gorilla Features

- Correlation Engine: "What happened around the time my service broke?"
- Charting: Horizon charts to see outliers and anomalies
- Aggregations: Rollups locally in Gorilla every couple of hours

![Graph showing memory usage over time](image)

Routine process of copying release binary begins

[Pelkonen et al., 2015]
Gorilla Lessons Learned

- Prioritize recent data over historical data
- Read latency matters
- High availability trumps resource efficiency
  - Withstand single-node failures and "disaster events" that affect region
  - "[B]uilding a reliable, fault tolerant system was the most time consuming part of the project"
  - "[K]eep two redundant copies of data in memory"

[Pelkonen et al., 2015]
Assignment 4

• Work on Data Integration and Data Fusion
• Integrate artist datasets from different institutions (Met, NGA, AIC, CMA)
  - Integrate information based on ids and matching
• Record Matching:
  - Which artists are the same?
• Data Fusion:
  - Names
  - Dates
  - Nationalities
Test 2

- Next Monday… April 8
- Similar format, but more emphasis on topics we have covered including the research papers
Graphs: Social Networks

[Image of a social network graph spanning the globe, labeled "facebook" and dated December 2010 by P. Butler, 2010]
What is a Graph?

• An abstract representation of a set of objects where some pairs are connected by links.

Object (Vertex, Node)

Link (Edge, Arc, Relationship)

[M. De Marzi, 2012]
What is a Graph?

- In computing, a graph is an abstract data structure that represents set objects and their relationships as vertices and edges/links, and supports a number of graph-related operations.
- Objects (nodes): \{A, B, C, D\}
- Relationships (edges):
  \{ (D, B), (D, A), (B, C), (B, A), (C, A) \}
- Operation: shortest path from D to A

[K. Salama, 2016]
Different Kinds of Graphs

- Undirected Graph
- Directed Graph
- Pseudo Graph
- Multi Graph
- Hyper Graph
Graphs with Properties

- Each vertex or edge may have properties associated with it
- May include identifiers or classes

```plaintext
Person
name = 'Tom Hanks'
born = 1956

Person
name = 'Robert Zemeckis'
born = 1951

Movie
title = 'Forrest Gump'
released = 1994

ACTED_IN
roles = ['Forrest']

DIRECTED
```
Types of Graph Operations

• Connectivity Operations:
  - number of vertices/edges, in- and out-degrees of vertices
  - histogram of degrees can be useful in comparing graphs

• Path Operations: cycles, reachability, shortest path, minimum spanning tree

• Community Operations: clusters (cohesion and separation)

• Centrality Operations: degree, vulnerability, PageRank

• Pattern Matching: subgraph isomorphism
  - can use properties
  - useful in fraud/threat detection, social network suggestions
What is a Graph Database?

• A database with an explicit graph structure
• Each node knows its adjacent nodes
• As the number of nodes increases, the cost of a local step (or hop) remains the same
• Plus an Index for lookups
How do Graph Databases Compare?

![Diagram showing the comparison of database types based on complexity and size. Graph Databases have high complexity and are used for 90% of use cases. Document Databases and Key-Value Stores have lower complexity and size. BigTable Clones have lower complexity and size than Document Databases. Relational Databases have the lowest complexity and size.](image-url)
Graph Databases Compared to Relational Databases

Optimized for aggregation

[Diagram of a graph with nodes A1, A2, A3, B1, B2, B3, B4, B5, B6, B7, C1, C2, C3]

Optimized for connections

[Diagram of a more complex graph with nodes A1, A2, B1, B2, B3, B4, B5, B6, B7, C1, C2, C3]

[M. De Marzi, 2012]
Graph Databases Compared to Key-Value Stores

- Optimized for simple look-ups
- Optimized for traversing connected data

[M. De Marzi, 2012]
Graph Databases Compared to Document Stores

Compared to Key Value Stores

Optimized for “trees” of data

Optimized for seeing the forest and the trees, and the branches, and the trunks

[M. De Marzi, 2012]
The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing

S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, and M. T. Özsu
The Future is Big Graphs

S. Sakr et al

CACM
Insights for the Future of Graph Processing

- Graphs are ubiquitous abstractions enabling reusable computing tools for graph processing with applications in every domain.
- Diverse workloads, standard models and languages, algebraic frameworks, and suitable and reproducible performance metrics will be at the core of graph processing ecosystems in the next decade.
Pipeline for Graph Processing

Data flows left to right, from data source to output, via a series of functionally different processing steps. Feedback and loopbacks flow mainly through the blue (highlighted) arrows.
Graph Databases

D. Lembo and R. Rosati
Why Graph Database Models?

- Graphs has been long ago recognized as one of the most simple, natural and intuitive knowledge representation systems.
- Graph data structures allow for a natural modeling when data has graph structure.
- Queries can address direct and explicitly this graph structure.
- Implementation-wise, graph databases may provide special graph storage structures, and take advantage of efficient graph algorithms available for implementing specific graph operations over the data.

[R. Angles and C. Gutierrez, 2017]
Relational Model

<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTNAME</th>
<th>PERSON</th>
<th>PARENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>George</td>
<td>Jones</td>
<td>Julia</td>
<td>George</td>
</tr>
<tr>
<td>Ana</td>
<td>Stone</td>
<td>Julia</td>
<td>Ana</td>
</tr>
<tr>
<td>Julia</td>
<td>Jones</td>
<td>David</td>
<td>James</td>
</tr>
<tr>
<td>James</td>
<td>Deville</td>
<td>David</td>
<td>Julia</td>
</tr>
<tr>
<td>David</td>
<td>Deville</td>
<td>Mary</td>
<td>James</td>
</tr>
<tr>
<td>Mary</td>
<td>Deville</td>
<td>Mary</td>
<td>Julia</td>
</tr>
</tbody>
</table>

Figure 1: Example of a genealogy expressed in the relational model (i.e. as tables on the left) and a diagram of its scheme on the right.

Regarding simplicity, one of the most popularized models is the semistructured model, which uses the most simple version of a graph, namely a tree, the most common and intuitive way of organizing our data (e.g. directories). Finally, the most common models are slightly enhanced versions of the plain graphs. One of them, the RDF model, gives a light typing to nodes, and considers edges as nodes, giving uniformity to the information objects in the model. The other, the property graph model, allows to add properties to edges and nodes.

Next, we will present these models and show a paradigmatic example of each. We will use the genealogy toy example modeled as tables and a simple schema in Figure 1.

3.1 The basics: Labeled graphs

The most basic data structure for graph database models is a directed graph with nodes and edges labeled by some vocabulary. A good example is Gram [37], a graph data model motivated by hypertext querying.

A schema in Gram is a directed labeled multigraph, where each node is labeled with a symbol called a type, which has associated a domain of values. In the same way, each edge has assigned a label representing a relation between types (see example in Figure 2). A feature of Gram is the use of regular expressions for explicit definition of paths called walks. An alternating sequence of nodes and edges represents a walk, which combined with other walks conforms other special objects called hyperwalks.

For querying the model (particularly path-like queries), an algebraic language based on regular expressions is proposed. For this purpose a hyper-
Basic Labeled Model (Gram)

- Directed graph with nodes and edges labeled by some vocabulary
- Gram is a directed labeled multigraph
  - Each node is labeled with a symbol called a **type**
  - Each edge has assigned a label representing a **relation** between types

[R. Angles and C. Gutierrez, 2017]
Hypergraph Model (Groovy)

- Notion of edge is extended to **hyperedge**, which relates an arbitrary set of nodes.
- Hypergraphs allow the definition of complex objects (undirected), functional dependencies (directed), object-ID and (multiple) structural inheritance.

![Hypergraph Model Diagram]

[R. Angles and C. Gutierrez, 2017]
Hypernode Model

• Hypernode is a directed graph whose nodes can themselves be graphs (or hypernodes), allowing nested graphs

• Encapsulates information

[R. Angles and C. Gutierrez, 2017]
Semistructured (Tree) Model: (OEM Graph)

- "Self-describing" data like JSON and XML
- OEM uses pointers to data in the tree

**OEM Syntax**

```json
{ person : &p1 { name : "George" ,
  lastname : "Jones" } }  
person : &p2 { name : "Ana" ,
  lastname : "Stone" }  
person : &p3 { name : "Julia" ,
  lastname : "Jones" ,
  parent : &p1 ,
  parent : &p2 }  
person : &p4 { name : "James" ,
  lastname : "Deville" }  
person : &p5 { name = "David",
  lastname : "Deville",
  parent : &p3 ,
  parent : &p4 }  
person : &p6 { name = "Mary",
  lastname : "Deville",
  parent : &p3 ,
  parent : &p4 }  
```

**OEM Graph**

![OEM Graph Diagram](image-url)

[R. Angles and C. Gutierrez, 2017]
RDF (Triple) Model

- Interconnect resources in an extensible way using graph-like structure for data
- Schema and instance are mixed together
- SPARQL to query
- Semantic web

[R. Angles and C. Gutierrez, 2017]
Property Graph Model (Cypher in neo4j)

- Directed, labelled, attributed multigraph
- Properties are **key/value pairs** that represent metadata for nodes and edges

![Property Graph Model Diagram]

[R. Angles and C. Gutierrez, 2017]
Types of Graph Queries

• Adjacency queries (neighbors or neighborhoods)
• Pattern matching queries (related to graph mining)
  - Graph patterns with structural extension or restrictions
  - Complex graph patterns
  - Semantic matching
  - Inexact matching
  - Approximate matching
• Reachability queries (connectivity)

[R. Angles and C. Gutierrez, 2017]
Types of Graph Queries (continued)

• Analytical queries
  - Summarization queries
  - Complex analytical queries (PageRank, characteristic path length, connected components, community detection, clustering coefficient)

[R. Angles and C. Gutierrez, 2017]
Graph Structures

- W3C RDF Dataset
  - Hyper vertices
- W3C RDF Graph
  - Multiple edges between a pair of nodes, single edge label
- ISO/IEC SQL Property Graph Model
  - Multiple edge labels
  - Neo4j/Oracle POCOL Property Graph Model
  - Multiple vertex labels
  - Gremlin / MS Cosmos Property Graph Model
  - Single vertex labels, vertex properties, edge properties
- Pregel/Graph-Graph (used for e.g. Node2vec)
  - Data Graph (used for e.g. Page rank, Connected component)
  - Weighted Graph (used for e.g. Shortest path, Louvain modularity)
- Directed Graph
  - Vertex data
  - Edge data
- Simple Graph

[D. Koop, CSCI 640/490, Spring 2024]
For the sake of space we will not present a complete review of graph query languages. Instead we describe some of the languages we consider relevant and useful to show the developments in the area. Moreover, we restrict our review to “pure” GQLs, that is those languages specifically designed to work with graph data models. Figure 8 presents this subset of languages in chronological order.

As we mentioned before, Cruz et al. [63] proposed the query language G. This language introduced the notion of graphical query as a set of query graphs. A query graph (pattern) is a labeled directed multigraph in which the node labels may be either variables or constants, and the edge labels can be regular expressions combining variables and constants. The result of a graphical query $Q$ with respect to a graph database $G$ is the union of all query graphs of $Q$ which match subgraphs of $G$. For instance, Figure 9 presents an example of graphical query containing two query graphs, $Q_1$ and $Q_2$. This query finds the first and last cities visited in all round trips from Toronto (“Tor”), in which the first and last flights are with Air Canada (“AC”) and all other flights (if any) are with the same airline. Note that the last condition is expressed by the edge labeled with regular expression $w^+$. Thanks to the inclusion of regular expressions, G is able to express recursive queries more general than transitive closure. However, the evaluation of queries in G is of high computational complexity due to its semantics based on simple paths.
Cypher

- Implemented by neo4j system
- Expresses reachability queries via path expressions
  
  \[ p = (a)-[:knows*]->(b): \text{ nodes from } a \text{ to } b \text{ following knows edges} \]
- START \textit{x}=node:person(name=\textit{"John"})
  MATCH \textit{(x)}-[[:friend]->(y)
  RETURN \textit{y}.name

[R. Angles and C. Gutierrez, 2017]
SPARQL (RDF)

- Uses SELECT-FROM-WHERE pattern like SQL

```
SELECT ?N
FROM <http://example.org/data.rdf>
WHERE { ?X rdf:type voc:Person . ?X voc:name ?N }
```
Comparing Graph Database Systems: Features

### Data Storage

<table>
<thead>
<tr>
<th>Graph Database</th>
<th>Main Memory</th>
<th>External Memory</th>
<th>Backend Storage</th>
<th>Indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllegroGraph</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>DEX</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Filament</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>G-Store</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>HyperGraphDB</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>InfiniteGraph</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Neo4j</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Sones</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>vertexDB</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

### Operations/Manipulation

<table>
<thead>
<tr>
<th>Graph Database</th>
<th>Data Definition Language</th>
<th>Data Manipulat. Language</th>
<th>Query Language</th>
<th>API</th>
<th>GUI</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllegroGraph</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>DEX</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Filament</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>G-Store</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>HyperGraphDB</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>InfiniteGraph</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Neo4j</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Sones</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>vertexDB</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

[R. Angles, 2012]
## Comparing Graph Database Systems: Representation

### Graph Data Structures

<table>
<thead>
<tr>
<th>Graph Database</th>
<th>Graphs</th>
<th>Nodes</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Simple graphs</td>
<td>Node labeled</td>
<td>Directed</td>
</tr>
<tr>
<td>AllegroGraph</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>DEX</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Filament</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>G-Store</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>HyperGraphDB</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>InfiniteGraph</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Neo4j</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Sones</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>vertexDB</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graph Database</th>
<th>Attributed graphs</th>
<th>Node attribute</th>
<th>Edge labeled</th>
<th>Edge attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllegroGraph</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>DEX</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Filament</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>G-Store</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>HyperGraphDB</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>InfiniteGraph</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Neo4j</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Sones</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>vertexDB</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

### Entites & Relations

<table>
<thead>
<tr>
<th>Graph Database</th>
<th>Schema</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Node types</td>
<td>Property types</td>
</tr>
<tr>
<td>AllegroGraph</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>DEX</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Filament</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>G-Store</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>HyperGraphDB</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>InfiniteGraph</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Neo4j</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Sones</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>vertexDB</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

### Table II

<table>
<thead>
<tr>
<th>Definition</th>
<th>Language</th>
<th>Data Storing Features</th>
<th>Operation and Manipulation Features</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table III

<table>
<thead>
<tr>
<th>Graph Database</th>
<th>Language</th>
<th>API</th>
<th>GUI</th>
<th>Query</th>
<th>Retrieval</th>
<th>Reasoning</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllegroGraph</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>DEX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Filament</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>G-Store</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>HyperGraphDB</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>InfiniteGraph</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Neo4j</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Sones</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>vertexDB</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
</tbody>
</table>

---

[Table III](#table-iii)

---

[R. Angles, 2012]
Comparing Graph Database Systems: Queries

<table>
<thead>
<tr>
<th>Graph Database</th>
<th>Query Lang.</th>
<th>API</th>
<th>Graphical Q. L.</th>
<th>Retrieval</th>
<th>Reasoning</th>
<th>Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllegroGraph</td>
<td>o</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>DEX</td>
<td>●</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Filament</td>
<td>●</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>G-Store</td>
<td>●</td>
<td></td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>HyperGraphDB</td>
<td>●</td>
<td></td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>InfiniteGraph</td>
<td>●</td>
<td></td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Neo4j</td>
<td>o</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Sones</td>
<td>●</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>vertexDB</td>
<td>●</td>
<td></td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Types of Queries</th>
<th>Adjacency</th>
<th>Reachability</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Node/edge adjacency</td>
<td>k-neighborhood</td>
<td>Fixed-length paths</td>
</tr>
<tr>
<td></td>
<td>G-Store</td>
<td>Filament</td>
<td>Regular simple paths</td>
</tr>
<tr>
<td></td>
<td>HyperGraph</td>
<td>G-Store</td>
<td>Shortest path</td>
</tr>
<tr>
<td></td>
<td>DEX</td>
<td>Filament</td>
<td>Pattern matching</td>
</tr>
<tr>
<td></td>
<td>AllegroGraph</td>
<td>Sones</td>
<td>Summarization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vertexDB</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Graph Database</th>
<th>Adjacency</th>
<th>Reachability</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllegroGraph</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>DEX</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Filament</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>G-Store</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>HyperGraph</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>InfiniteGraph</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Neo4j</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Sones</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>vertexDB</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

[R. Angles, 2012]
The (sorry) State of Graph Database Systems

Peter Boncz

Keynote, EDBT-ICDT 2022