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

<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;</td>
<td>&quot;34&quot;</td>
</tr>
<tr>
<td>2016-07-12T11:51:45Z</td>
<td>&quot;true&quot;</td>
<td>&quot;34&quot;</td>
</tr>
<tr>
<td>2016-07-12T12:10:00Z</td>
<td>&quot;true&quot;</td>
<td>&quot;34&quot;</td>
</tr>
<tr>
<td>2016-07-12T12:10:00Z</td>
<td>&quot;true&quot;</td>
<td>&quot;34&quot;</td>
</tr>
<tr>
<td>2016-07-12T12:10:00Z</td>
<td>&quot;true&quot;</td>
<td>&quot;34&quot;</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**

[R. J. Hyndman]
Examples

Trend

US Treasury bill contracts

Australian electricity production

Sales of new one-family houses, USA

Annual Canadian Lynx trappings

Notes: Seasonal or cyclic?

Time series patterns

Forecasting: Principles and Practice

R. J. Hyndman

D. Koop, CSCI 640/490, Spring 2023
Examples

US Treasury bill contracts

Australian electricity production

Sales of new one-family houses, USA

Annual Canadian Lynx trappings

Trend

Trend + Seasonality

— R. J. Hyndman —

D. Koop, CSCI 640/490, Spring 2023
Examples

- **Trend**
  - US Treasury bill contracts

- **Seasonality + Cyclic**
  - Sales of new one-family houses, USA

- **Trend + Seasonality**
  - Australian electricity production

- **Annual Canadian Lynx trappings**

---

[R. J. Hyndman]
Examples

- **Trend**

  US Treasury bill contracts

  ![Graph of US Treasury bill contracts](image1)

- **Trend + Seasonality**

  Australian electricity production

  ![Graph of Australian electricity production](image2)

- **Seasonality + Cyclic**

  Sales of new one-family houses, USA

  ![Graph of sales of new one-family houses](image3)

- **Stationary**

  Annual Canadian Lynx trappings

  ![Graph of annual Canadian Lynx trappings](image4)

---

[R. J. Hyndman]
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
  - downsampling: higher frequency to lower frequency
  - upsampling: 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>freq</td>
<td>String or DateOffset indicating desired resampled frequency (e.g., 'M', 'Smin', or Second(15))</td>
</tr>
<tr>
<td>axis</td>
<td>Axis to resample on; default axis=0</td>
</tr>
<tr>
<td>fill_method</td>
<td>How to interpolate when upsampling, as in 'ffill' or 'bfill'; by default does no interpolation</td>
</tr>
<tr>
<td>closed</td>
<td>In downsampling, which end of each interval is closed (inclusive), 'right' or 'left'</td>
</tr>
<tr>
<td>label</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>loffset</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>limit</td>
<td>When forward or backward filling, the maximum number of periods to fill</td>
</tr>
<tr>
<td>kind</td>
<td>Aggregate to periods ('period') or timestamps ('timestamp'); defaults to the type of index the time series has</td>
</tr>
<tr>
<td>convention</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>

[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'"
```

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

The key specified in the monitoring data is used to uniquely identify a time series. By sharding all monitoring data based on these unique string keys, each time series dataset can be mapped to a single Gorilla host. Thus, we can scale Gorilla by simply adding new hosts and tuning the sharding function to map new time series data to the expanded set of hosts. When Gorilla was launched to production 18 months ago, our dataset of all time series data inserted in the past 26 hours fit into 1.3TB of RAM evenly distributed across 20 machines. Since then, we have had to double the size of the clusters twice due to data growth, and are now running on 80 machines within each Gorilla cluster. This process was simple due to the share-nothing architecture and focus on horizontal scalability.

Gorilla tolerates single node failures, network cuts, and entire datacenter failures by writing each time series value to two hosts in separate geographic regions. On detecting a failure, all read queries are failed over to the alternate region ensuring that users do not experience any disruption.

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.

[Pelkonen et al., 2015]
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 percent of memory used 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 travel datasets from different institutions (UN World Tourism Office, World Bank, OECD)
  - Integrate information with population
- Record Matching:
  - Which countries are the same?
- Data Fusion:
  - The receipts/expenditures
  - Country names
Test 2

- Upcoming… this coming Monday
- Similar format, but more emphasis on topics we have covered including the research papers
Specific Types of Data
Graphs: Social Networks
What is a Graph?

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

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?

90% of Use Cases

Size

Complexity

Graph Databases

Document Databases

BigTable Clones

Key-Value Store

Relational Databases

90% of Use Cases

[From M. De Marzi, 2012]
Graph Databases Compared to Relational Databases

- Optimized for aggregation
- Optimized for connections

[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]
Compared to Key Value Stores

Optimized for “trees” of data

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

Graph Databases Compared to Document Stores

[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.

[S. Sakr et al.]
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.

Non-Graph Data Sources

Relational Database

Graph Data

Graph Extraction

ETL for Graph Data

Extracted Graphs

Graph OL TP Operations

Graph OLAP Operations

Graph Analytics Engine

Graph Workflow

Graph Algorithm

Graph Model

Data Model

Processing Formalism

Graph Database

Processed Output

Machine Learning

Business Intelligence

Scientific Computing

Augmented Reality and Visualization

Graph-Based Engines

S. Sakr et al.
Graph Databases

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

- Graphs have 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 a graph structure.
- Queries can address directly 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.

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 represent 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 Diagram](image-url)

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

- Hypernode is a directed graph whose nodes can themselves be graphs (or hypernodes), allowing **nesting** of graphs
- **Encapsulates** information

![Diagram of Hypernode Model](image)

[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 Example]

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 Example]

**Figure 5:** Object Exchange Model (OEM). Schema and instance are mixed.

The data is modeled beginning in a root node &pp, with children person nodes, each of them identified by an Object-ID (e.g. &p2). These nodes have children that contain data (name and lastname) or references to other nodes (parent). Referencing permits to establish relations between distinct hierarchical levels. Note the tree structure obtained if one forgets the pointers to OIDs, a characteristic of semistructured data.

Today we can recognize in JSON.

The most popular and elaborated version of the semi-structured model is the XML model. It comprises a rich and flexible data structure [1], a suite of highly refined and standardized query and transformation languages (XPath, XQuery, XSLT) [1] and several other features, that have much to teach graph query language designers.

3.5 Uniform graphs: The RDF model

The Resource Description Framework (RDF) [96] is a recommendation of the W3C designed originally to represent metadata. One of the main advantages (features) of the RDF model is its ability to interconnect resources in an extensible way using graph-like structure for data.

One of the main advantages of RDF is its dual nature. In fact, there are two possible reading of the model. From a knowledge representation language perspective, RDF is a graph-based language for describing resources and their relationships. From a data processing perspective, RDF is used to store and query complex data structures.

[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

![RDF Data Model Diagram](image.png)

[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](image)

[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]
Figure 8: Evolution of graph query languages: G [63], G+ [64], Graphlog, HPQL [104], THQL [141], GRE [37], Hyperlog [123], HNQL [103], PORL [72], SLQL [52], HQL [137], PRPQ [107], GraphQL [85], SPARQL [124], RLV [132], Cypher [14], ECRPQ [43], PDQL [41], GX-Path [106], SPARQL 1.1 [71] and RQ [127].

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.

G evolved into a more powerful language called G+ [64]. The notion of graphical query proposed by G is extended in G+ to define a summary...
Cypher

- Implemented by neo4j system
- Expresses reachability queries via path expressions

\[- p = (a)-[\text{knows}^*]->(b): \text{nodes from } a \text{ to } b \text{ following } \text{knows} \text{ edges}\]

- \text{START } x=\text{node:person(name="John")}
  \text{MATCH } (x)-[\text{friend}]->(y)
  \text{return} \ y.\text{name}\]
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 }

[R. Angles and C. Gutierrez, 2017]
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>

[Table II: Comparing Graph Database Systems: Features]

[D. Koop, CSCI 640/490, Spring 2023]
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>Hypergraphs</td>
<td>Nested graphs</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>

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>

Note: The above tables summarize the features of different graph database systems, including their support for various graph data structures and entity-relation types. The symbols represent the degree of support or implementation: • indicates partial support, and • indicates full support.

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

### Query Support

<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>○</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>

### Types of Queries

<table>
<thead>
<tr>
<th>Graph Database</th>
<th>Adjacency</th>
<th>Reachability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Node/edge adjacency</td>
<td>k-neighborhood</td>
</tr>
<tr>
<td>Allegro</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>

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

Peter Boncz

Keynote, EDBT-ICDT 2022