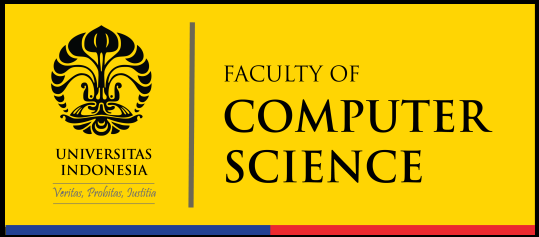


Semantic Web 02: Graph-based Data Models

Adila Krisnadhi – adila@cs.ui.ac.id Faculty of Computer Science, Universitas Indonesia

Outline

Knowledge graph (KG): Definition

- The term “knowledge graph” has been used in the literature since 1973 with many, sometimes conflicting, definitions proposed. (See discussion on Appendix A of the KG Book).
- A knowledge graph is a graph of data intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent relations between these entities.
  - The data graph conforms to a graph-based data model.
  - Knowledge refers to something that is known, perhaps accumulated from external sources, or extracted from the graph itself.

“Knowledge” in KG

- May be composed of
  - simple statements, e.g., “Jakarta is the capital of Indonesia”
    - simple statements can form edges in the data graph
  - quantified statements, e.g., “all capitals are cities”
    - require more expressive way to represent knowledge, e.g., using ontologies or rules.
- To entail and accumulate more knowledge, deductive and inductive methods can be used
  - e.g., we can deduce that the statement “Jakarta is a city” is also true.

Diversity and knowledge change in KG

- The structure and granularity of a KG can be highly diverse since it may be assembled from multiple sources.
- To handle the diversity, we use:
  - schema: high-level structure for the KG;
  - identity: denotes which nodes in the KG (or in external sources) refer to the same real-world entity;
  - context: indicates a specific setting in which some parts of the KG is held true.
- Knowledge changes over time, hence we study these effective methods for KG:
  - extraction
  - enrichment
  - quality assessment
  - refinement

KG in practice

- Open knowledge graphs: all their content accessible online for the public good.

KG in practice

- Open knowledge graphs: all their content accessible online for the public good.
  - DBPedia, Freebase, Wikidata, YAGO (cover many domains; either extracted from Wikipedia or built by communities of volunteers)

KG in practice

- Open knowledge graphs: all their content accessible online for the public good.
  - DBPedia, Freebase, Wikidata, YAGO (cover many domains; either extracted from Wikipedia or built by communities of volunteers)
  - covering specific domains: BBC KG, Data.gov KG, LinkedGeoData, Bio2RDF, etc.

KG in practice

- Open knowledge graphs: all their content accessible online for the public good.
  - DBPedia, Freebase, Wikidata, YAGO (cover many domains; either extracted from Wikipedia or built by communities of volunteers)
  - covering specific domains: BBC KG, Data.gov KG, LinkedGeoData, Bio2RDF, etc.
- Enterprise knowledge graphs: internal for a particular organization/company and often applied for commercial use-cases, such as KGs created by:

KG in practice

- Open knowledge graphs: all their content accessible online for the public good.
  - DBPedia, Freebase, Wikidata, YAGO (cover many domains; either extracted from Wikipedia or built by communities of volunteers)
  - covering specific domains: BBC KG, Data.gov KG, LinkedGeoData, Bio2RDF, etc.
- Enterprise knowledge graphs: internal for a particular organization/company and often applied for commercial use-cases, such as KGs created by:
  - Google, Bing, Airbnb, Amazon, eBay, Uber, Facebook, LinkedIn, Accenture, Banca d'Italia, Bloomberg, Capital One, Wells Fargo, etc.

KG in practice

- Open knowledge graphs: all their content accessible online for the public good.
  - DBPedia, Freebase, Wikidata, YAGO (cover many domains; either extracted from Wikipedia or built by communities of volunteers)
  - covering specific domains: BBC KG, Data.gov KG, LinkedGeoData, Bio2RDF, etc.
- Enterprise knowledge graphs: internal for a particular organization/company and often applied for commercial use-cases, such as KGs created by:
  - Google, Bing, Airbnb, Amazon, eBay, Uber, Facebook, LinkedIn, Accenture, Banca d'Italia, Bloomberg, Capital One, Wells Fargo, etc.
- Applications of KG include search, recommendation systems, personal agents, advertising, business analytics, risk assessment, automation, etc.

Outline

Motivating example

- A tourism board intends to model relevant data about attractions, services, and events. They start with a tabular structure, specifically with the following initial relational schema:

$$Event(\underline{name}, venue, type, \underline{start}, end)$$

where $\underline{name}$ and $\underline{start}$ form the primary key to uniquely identify recurring events.

Motivating example

- A tourism board intends to model relevant data about attractions, services, and events. They start with a tabular structure, specifically with the following initial relational schema:

$$Event(\underline{name}, venue, type, \underline{start}, end)$$

  where $\underline{name}$ and $\underline{start}$ form the primary key to uniquely identify recurring events.

- What do you think the problems will be when the board starts to populate the table with data?

Motivating example (contd.)

Problems with

$$Event(\underline{name}, venue, type, \underline{start}, end)$$

Motivating example (contd.)

Problems with

$$Event(\underline{name}, venue, type, \underline{start}, end)$$

- Events may have multiple names.

Motivating example (contd.)

Problems with

$$Event(\underline{name}, venue, type, \underline{start}, end)$$

- Events may have multiple names.
- Events may have multiple venues.

Motivating example (contd.)

Problems with

$$Event(\underline{name}, venue, type, \underline{start}, end)$$

- Events may have multiple names.
- Events may have multiple venues.
- The board may not yet know the start and end date-times of future events.

Motivating example (contd.)

Problems with

$$Event(\underline{name}, venue, type, \underline{start}, end)$$

- Events may have multiple names.
- Events may have multiple venues.
- The board may not yet know the start and end date-times of future events.
- Events may have have multiple types.

Motivating example (contd.)

Problems with

$$Event(\underline{name}, venue, type, \underline{start}, end)$$

- Events may have multiple names.
- Events may have multiple venues.
- The board may not yet know the start and end date-times of future events.
- Events may have have multiple types.

What would the board do?

Motivating example (contd.)

The board would do the following:

- Create internal identifiers for events, say id.

Motivating example (contd.)

The board would do the following:

- Create internal identifiers for events, say id.
- Adapt the relational schema (e.g., perform normalization) to yield tables like the following:

$$EventName(\underline{id}, \underline{name}) \quad EventStart(\underline{id}, start) \quad EventEnd(\underline{id}, end)$$
$$EventVenue(\underline{id}, \underline{venue}) \quad EventType(\underline{id}, \underline{type})$$

Note: Why do we make $name$, $venue$, and $type$ primary keys?

Motivating example (contd.)

- But, along the way, schema may have to be changed incrementally several times to support new sources of data.

Motivating example (contd.)

- But, along the way, schema may have to be changed incrementally several times to support new sources of data.
  - Costly remodeling (change of schema), reloading, and reindexing of data. Imagine if we work with many tables (not just one)!

Motivating example (contd.)

- But, along the way, schema may have to be changed incrementally several times to support new sources of data.
  - Costly remodeling (change of schema), reloading, and reindexing of data. Imagine if we work with many tables (not just one)!
- Why does the board struggle with relational schema?

Motivating example (contd.)

- But, along the way, schema may have to be changed incrementally several times to support new sources of data.
  - Costly remodeling (change of schema), reloading, and reindexing of data. Imagine if we work with many tables (not just one)!
- Why does the board struggle with relational schema?
  - because they do not know, in advance, what data to be modeled or what sources to be used.

Motivating example (contd.)

- But, along the way, schema may have to be changed incrementally several times to support new sources of data.
  - Costly remodeling (change of schema), reloading, and reindexing of data. Imagine if we work with many tables (not just one)!
- Why does the board struggle with relational schema?
  - because they do not know, in advance, what data to be modeled or what sources to be used.
- Once the latter/adapted relational schema is obtained, further integration of new data sources could be done without more changes.

Motivating example (contd.)

- But, along the way, schema may have to be changed incrementally several times to support new sources of data.
  - Costly remodeling (change of schema), reloading, and reindexing of data. Imagine if we work with many tables (not just one)!
- Why does the board struggle with relational schema?
  - because they do not know, in advance, what data to be modeled or what sources to be used.
- Once the latter/adapted relational schema is obtained, further integration of new data sources could be done without more changes.
  - The latter schema only makes minimal assumptions on multiplicities (1-1, 1-n, etc.)

Motivating example (contd.)

- But, along the way, schema may have to be changed incrementally several times to support new sources of data.
  - Costly remodeling (change of schema), reloading, and reindexing of data. Imagine if we work with many tables (not just one)!
- Why does the board struggle with relational schema?
  - because they do not know, in advance, what data to be modeled or what sources to be used.
- Once the latter/adapted relational schema is obtained, further integration of new data sources could be done without more changes.
  - The latter schema only makes minimal assumptions on multiplicities (1-1, 1-n, etc.)
  - The latter schema ends up with a set of binary relations between entities

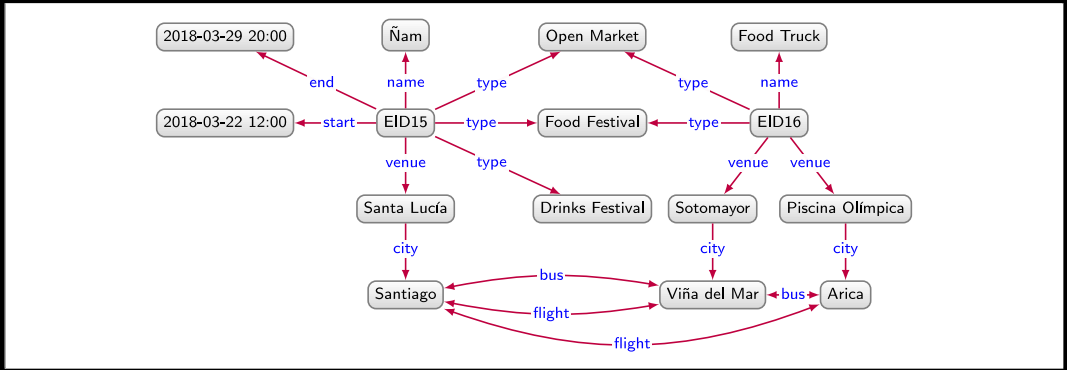
Instances of binary relations naturally form a graph.

Instances of binary relations naturally form a graph.

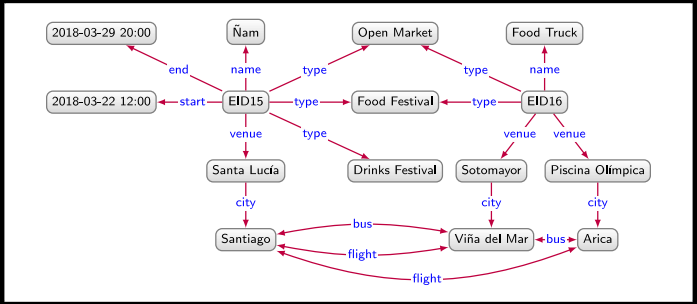
So, why not use **graph** from the start?

Outline

DELG model about events



DELG model about events



- Does adding information to such a graph easy? How?
- Can we represent incomplete information in the graph? How?
- Do we need to define a schema upfront like relational databases?
- Do we require the data to be organized hierarchically like XML or JSON?

Directed edge-labeled graphs

Let $\mathbf{Con}$ be a countably infinite set of constants.

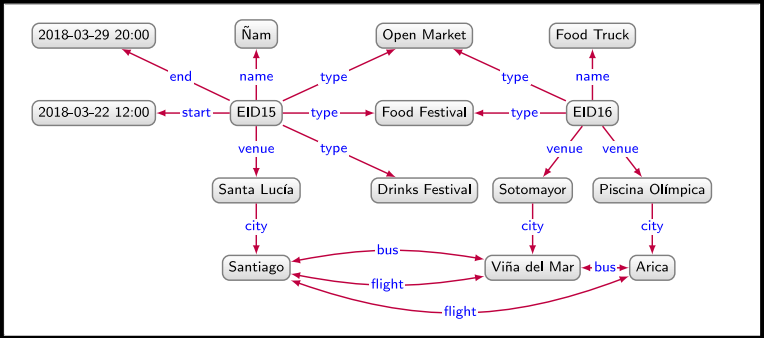
Definition

A directed edge-labeled graph (DELG) is a tuple $G = (V, E, L)$, where

- $V \subseteq \mathbf{Con}$ is the set of nodes,
- $L \subseteq \mathbf{Con}$ is the set of edge labels, and
- $E \subseteq V \times L \times V$ be the set of labeled edges, i.e., triples of the form (node, edge label, node).

Note that V and L need **not** be disjoint (unlike the typical graphs you learned in the Discrete Math classes).

DELG: Example



List all elements of V, L, and E.

DELG: Example

- $V = \{$2018-03-29 20:00, Ñam, Open Market, Food Truck, 2018-03-22 12:00, EID15, Food Festival, EID16, Santa Lucía, Drinks Festival, Sotomayor, Piscina Olímpica, Santiago, Viña del Mar, Arica$\}$
- $L = \{$name, type, venue, start, end, city, bus, flight$\}$
- $E = \{$(EID15, name, Ñam), (EID, start, 2018-03-22 12:00), (EID15, end, 2018-03-29 20:00) (EID15, venue, Santa Lucía), (EID15, type, Open Market), (EID15, type, Food Festival), (EID15, type, Drinks Festival), (EID16, name, Food Truck), (EID16, type, Open Market), (EID16, type, Food Festival), (EID16, venue, Sotomayor), (EID16, venue, Piscina Olímpica), (Santa Lucía, city, Santiago), (Sotomayor, city, Viña del Mar), (Piscina Olímpica, city, Arica) (Santiago, bus, Viña del Mar), (Viña del Mar, bus, Santiago), (Santiago, flight, Viña del Mar) (Viña del Mar, flight, Santiago), (Santiago, flight, Arica), (Arica, flight, Santiago), (Viña del Mar, bus, Arica), (Arica, bus, Viña del Mar)$\}$

DELG: Remarks

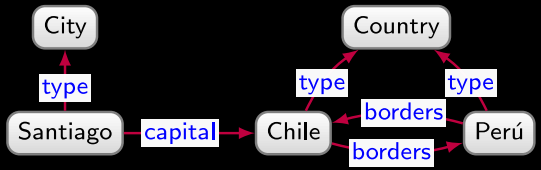
- We can treat the set of edges E as the DELG without explicitly referring to the set of nodes V and labels E.
  - Given a set of edges E, we define the graph induced by E as $G = (V, E, L)$ such that V comprises the nodes at either end of any edge in E and L comprises the labels of any edge in E.

- We can perform set operations on DELGs, which are understood as the application of those operations to their sets of edges.
  - Given DELG $G_1 = (V_1, E_1, L_1)$ and $G_2 = (V_2, E_2, L_2)$, we define the DELG $G_1 \cup G_2$ as the graph induced by $E_1 \cup E_2$
  - W3C standards for DELG data model is Resource Description Framework (RDF).

Outline

Heterogeneous graph: Example

Heterogeneous graphs are like DELGs, except that they treat the `type` edges as part of the graph model directly, instead of as a binary relation.



Heterogeneous graphs

Let $\mathbf{Con}$ be a countably infinite set of constants.

Definition

A heterogeneous graph is a tuple $G = (V, E, L, \ell)$, where

- $V \subseteq \mathbf{Con}$ is a set of nodes,
- $L \subseteq \mathbf{Con}$ is a set of edge/node labels,
- $E \subseteq V \times L \times V$ is a set of labeled edges, and
- $\ell \colon V \to L$ is a function that maps each node to a label.

Edge and node labels in heterogeneous graphs are called types.

Heterogeneous graph: Example

- $V = \{$Santiago, Chile, Perú $\}$
- $L = \{$City, Country$\}$
- $E = \{$(Santiago, capital, Chile), (Chile, borders, Perú), (Perú, borders, Chile)$\}$
- $\ell = \{$Santiago $\mapsto$ City, Chile $\mapsto$ Country, Perú $\mapsto$ Country$\}$

Heterogeneus graphs: Remarks

- An edge in a heterogeneous graph is called homogeneous if it is between two nodes of the same types/labels. Otherwise, it is called heterogeneous. (Which edges are homogeneous and which are heterogeneous in the previous example?)
- Heterogeneous graphs allow for the partitioning of their nodes according to their type — useful, e.g., for the purposes of machine learning tasks.
- Unlike DELG, heterogeneous graphs only support a many-to-one relation between nodes and their types, i.e., each node is associated exactly with one type.

Outline

Motivation

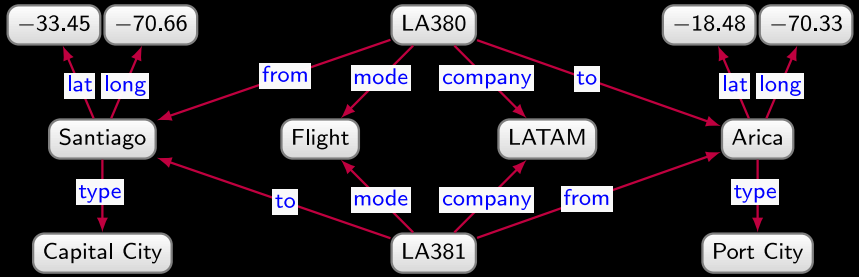
- Consider the case where the tourism board would like to integrate incoming data that provide further details on which companies offer fares on which flights, hence allowing better understanding of available routes between cities.

Motivation

- Consider the case where the tourism board would like to integrate incoming data that provide further details on which companies offer fares on which flights, hence allowing better understanding of available routes between cities.

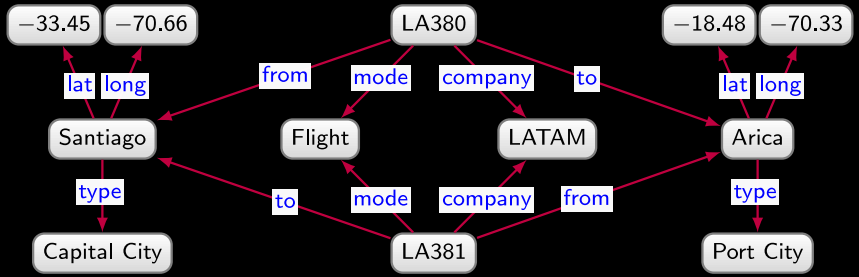
- Using DELG, we **cannot** directly annotate an edge like (Santiago, flight, Arica) with the companies offering that route.

Motivation



- In DELG, we could add a new node denoting a flight, then connect this node with the source, destination, companies, and mode.

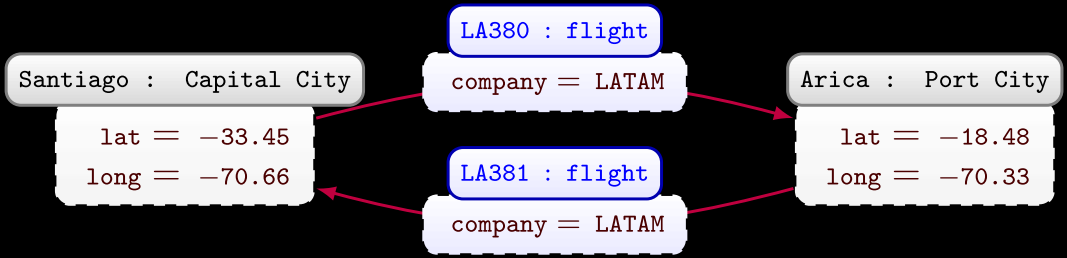
Motivation



- In DELG, we could add a new node denoting a flight, then connect this node with the source, destination, companies, and mode.
- However, applying this to all routes would involve significant changes.

Property graph: Example

As alternative to DELG, property graphs allow a set of property-value pairs to be associated with both nodes and edges, offering more flexibility.



Property graph model

Property graphs

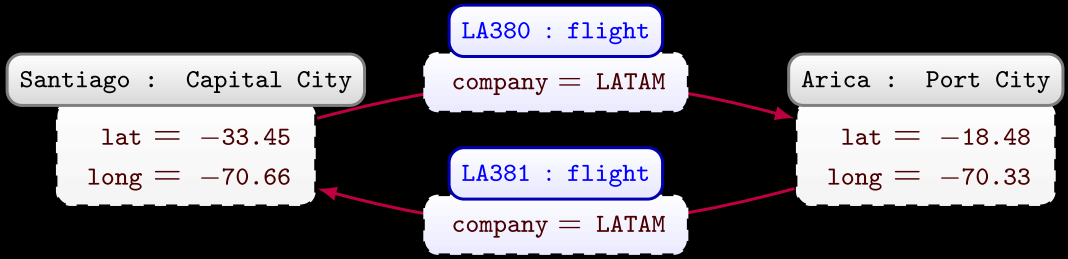
Let $\mathbf{Con}$ be a countably infinite set of constants.

Definition

A property graph is a tuple $G = (V, E, L, P, U, e, \ell, p)$, where

- $V \subseteq \mathbf{Con}$ is a set of node ids,
- $E \subseteq \mathbf{Con}$ is a set of edge ids
- $L \subseteq \mathbf{Con}$ is a set of labels,
- $P \subseteq \mathbf{Con}$ is a set of properties,
- $U \subseteq \mathbf{Con}$ is a set of values,
- $e \colon E \to V \times V$ is a function that maps an edge id to a pair of node ids,
- $\ell \colon V \cup E \to 2^L$ is a function that maps a node or edge id to a **set of labels**, and
- $p \colon V \cup E \to 2^{P \times U}$ is a function that maps a node or edge id to a **set of property-value pairs**.

Property graph: Example



List all elements of V, E, L, P, U, e, ℓ, and p

Property graph: Example

- $V = \{$Santiago, Arica$\}$
- $E = \{$LA380, LA381$\}$
- $L = \{$Capital City, Port City, flight$\}$
- $P = \{$lat, long, company$\}$
- $U = \{$-33.45, -70.66. -18.48, -70.33, LATAM$\}$
- $e = \{$LA380 $\mapsto$ (Santiago, Arica), LA381 $\mapsto$ (Arica, Santiago)$\}$
- $\ell = \{$Santiago $\mapsto \{$Capital City$\}$, Arica $\mapsto \{$Port City$\}$, LA380 $\mapsto \{$flight$\}$, LA381 $\mapsto \{$flight$\}\}$
- $p = \{$Santiago $\mapsto \{$(lat,-33.45), (long,-70.66)$\}$, LA380 $\mapsto \{$(company, LATAM)$\}$ Arica $\mapsto \{$(lat,-18.48), (long,-70.33)$\}$, LA381 $\mapsto \{$(company, LATAM)$\}$ $\}$

Property graph: Remarks

- Property graphs are prominently used in graph databases, such as in Neo4j.
- Property graphs can be converted to/from DELG.
- DELGs offers a more minimal model, while property graphs offer a more flexible one.
- Choice of models are also often dictated by other factors, e.g., the availability of implementation for different models.

Outline

Graph datasets

- We sometimes have to work with multiple KGs. (Why?)

Graph datasets

- We sometimes have to work with multiple KGs. (Why?)
- Two possibilities to handle multiple KGs:

Graph datasets

- We sometimes have to work with multiple KGs. (Why?)
- Two possibilities to handle multiple KGs:
  - merge the graphs together into one monolithic graph by taking their union; or

Graph datasets

- We sometimes have to work with multiple KGs. (Why?)
- Two possibilities to handle multiple KGs:
  - merge the graphs together into one monolithic graph by taking their union; or
  - manage a graph dataset containing multiple graphs that are separated from each other.

Graph datasets (contd.)

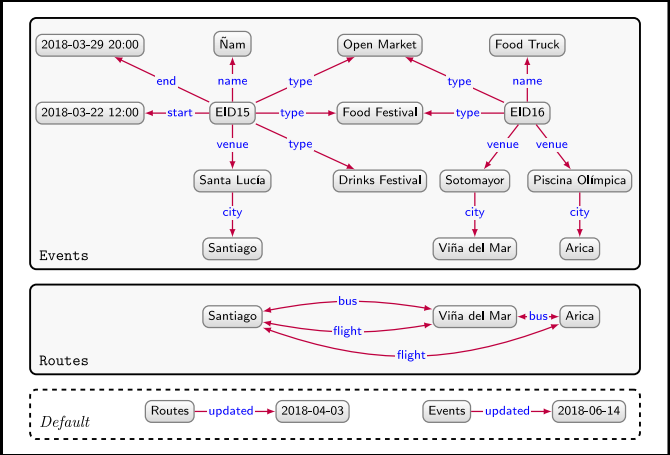
Let $\mathbf{Con}$ be a countably infinite set of constants.

Definition

A named graph is a pair (n, G) where G is a data graph and $n \in \mathbf{Con}$ is a graph name/ID.
A graph dataset is a tuple $\mathcal{G} = (G_D, \{(n_1, G_1), \ldots, (n_k, G_k)\})$ with $k \geq 0$ where

- G_D is a data graph, called the default graph, which is allowed to be empty.
- (n_i, G_i) is a named graph for each i, $1 \leq i \leq k$
- if $k > 0$, then for each $1 \leq i, j \leq k$, whenever $i \neq j$, then $n_i \neq n_j$

Graph datasets: Example



Graph dataset with a default graph and two named graphs (`Routes` and `Events`)

Graph datasets: Remarks

- A graph dataset consists of exactly one default graph and zero or more named graphs.
- Named graphs are uniquely named, i.e., no two named graphs in the same graph dataset have the same name.
- The default graph of a graph dataset is the only graph without an ID in that dataset.
- Graph names can be used as nodes in any graph in the dataset.
- Nodes and edges may be repeated across different graphs in the dataset (and they refer to the same entity).
- Using a graph dataset allows us to:
  - update or refine data from different sources separately;
  - distinguish untrustworthy sources from the more trustworthy ones;
  - etc.

Outline

Other graph data models

- Graph data models other than the three already discussed exist, e.g.,
  - graphs with complex nodes (called hypernodes), which may contain individual edges or nested graphs;
  - (hyper)graphs with complex edges that connect sets rather than pairs of nodes.

- Conversion between different graph data models is often possible.

- KG can adopt any of the aforementioned models, but we focus only on DELGs.

Graph stores

- Issue: how do we store and index graphs to facilitate efficient evaluation of queries?

Graph stores

- Issue: how do we store and index graphs to facilitate efficient evaluation of queries?
- DELGs can be stored in relational databases as:

Graph stores

- Issue: how do we store and index graphs to facilitate efficient evaluation of queries?
- DELGs can be stored in relational databases as:
  - a single relation of arity three, i.e., a triple table; or

Graph stores

- Issue: how do we store and index graphs to facilitate efficient evaluation of queries?
- DELGs can be stored in relational databases as:
  - a single relation of arity three, i.e., a triple table; or
  - a collection of binary relations/tables, one for each property, i.e., vertical partitioning; or

Graph stores

- Issue: how do we store and index graphs to facilitate efficient evaluation of queries?
- DELGs can be stored in relational databases as:
  - a single relation of arity three, i.e., a triple table; or
  - a collection of binary relations/tables, one for each property, i.e., vertical partitioning; or
  - a collection of n-ary relations/tables, each contains entities of the same type, i.e., property tables.

Graph stores

- Issue: how do we store and index graphs to facilitate efficient evaluation of queries?
- DELGs can be stored in relational databases as:
  - a single relation of arity three, i.e., a triple table; or
  - a collection of binary relations/tables, one for each property, i.e., vertical partitioning; or
  - a collection of n-ary relations/tables, each contains entities of the same type, i.e., property tables.
- DELGs can also be stored using custom/native storage techniques (not using tables), which may provide more efficient access for finding nodes, edges, and their adjacent elements.

Graph stores

- Issue: how do we store and index graphs to facilitate efficient evaluation of queries?
- DELGs can be stored in relational databases as:
  - a single relation of arity three, i.e., a triple table; or
  - a collection of binary relations/tables, one for each property, i.e., vertical partitioning; or
  - a collection of n-ary relations/tables, each contains entities of the same type, i.e., property tables.
- DELGs can also be stored using custom/native storage techniques (not using tables), which may provide more efficient access for finding nodes, edges, and their adjacent elements.
- Some systems further allow distributing graphs over multiple machines based on NoSQL stores or custom partitioning schemes.