Dresden Database Systems Group

Graph Analytics

Hannes Voigt

Bio

Dresden Database Systems Group

Current position

- Postdoc at Database System Group, Technische Universität Dresden

Education

- Ph.D. in 2014
- Master in 2008

Experience

- Visiting scholar at SAP Labs, Palo Alto for one year in 2010
- Visiting scholar at University Waterloo, for 4 months in 2007

Interests and Activities

- Graph Data Management and Data Science
- LDBC Graph Query Language Standardization Task Force
- Collaboration with SAP Hana Graph Team

Dresden Database Systems Group

Dresden Database Systems Group

VISIT: HTTPS://WWWDB.INF.TU-DRESDEN.DE/

Trends in Data Management

Everything is Data

[http://blog.acronis.com/posts/data-everything-8-noble-truths]

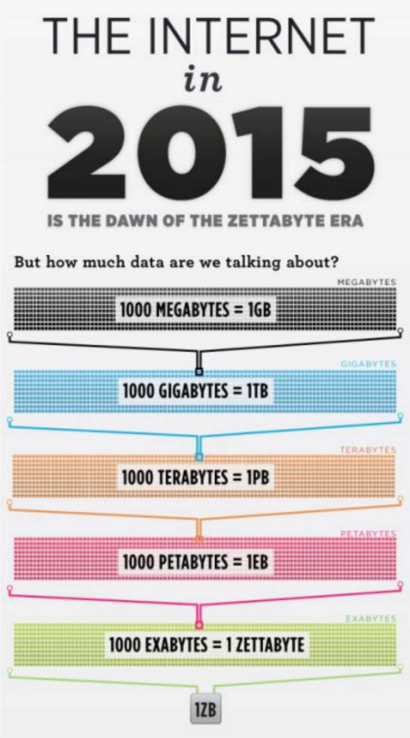
The Zettabyte Age

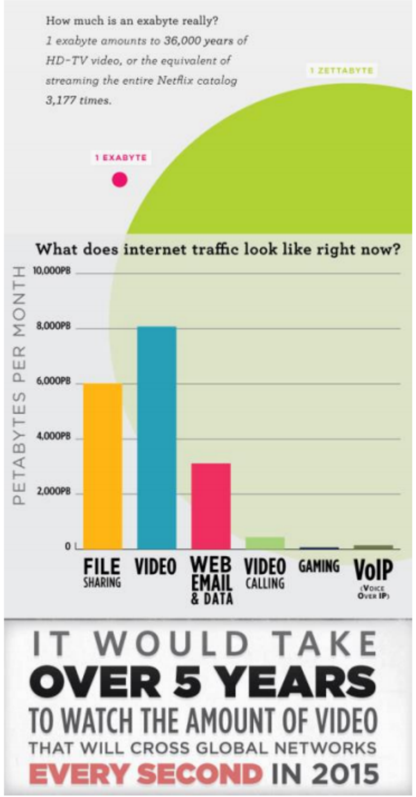
Dresden Database Systems Group

Cisco Visual Networking Index

The internet in 2020

- 26.3 billion networked devices
  - Up from 16.3 billion in 2015
  - 44% of all networked devices will be mobile-connected
- 25.1 GB average traffic per capita per month
  - Up from 9.9 GB in 2015
- 2.3 Zettabytes annual IP-Traffic
  - up from 870.3 Exabytes annual IP-Traffic in 2015
  - One zettabyte = stack of books from Earth to Pluto 20 times





The End of Science

Dresden Database Systems Group

The quest for knowledge used to begin with grand theories. Now it begins with massive amounts of data.

Zetta

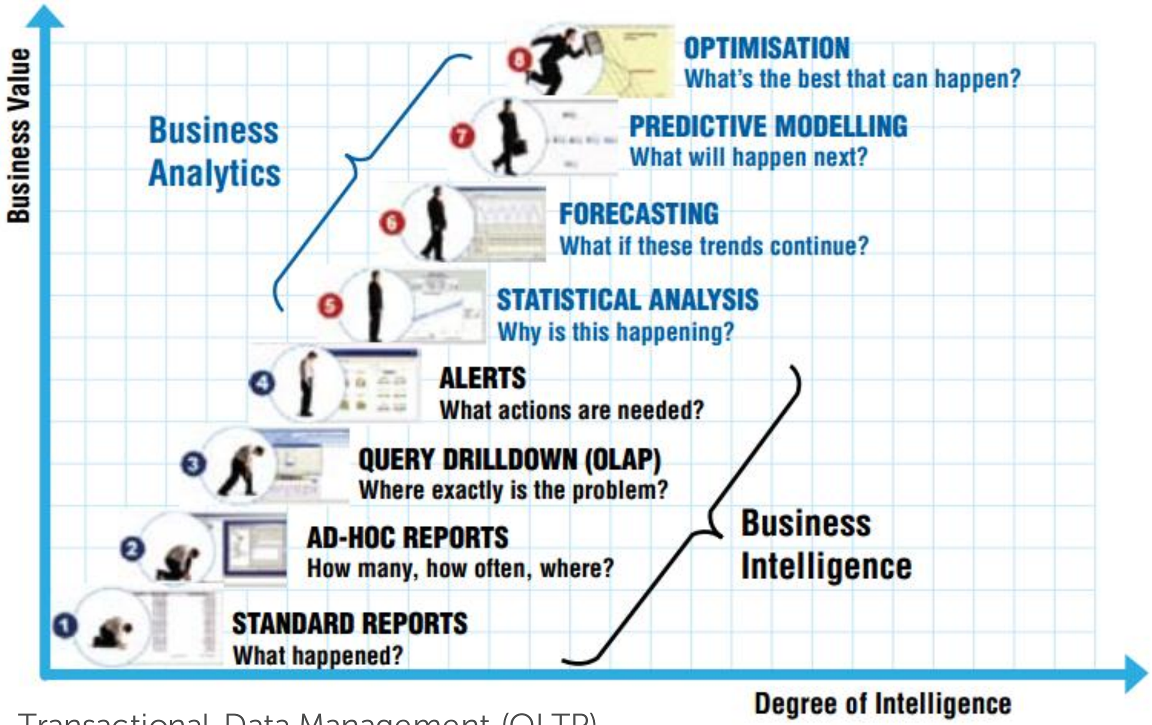
Welcome to the ~~Peta~~byte Age.

New Realities

- Everything is digital data
- Rise of data-driven culture
- High-performant data analytics
- Exploit sophisticated statistical methods

How do we structure/implement/live with this trend?

Level of Analytics



0. Transactional Data Management (OLTP)

[http://www.rosebt.com/blog/eight-levels-of-analytics-business-intelligence-to-business-analytics]

Focus of Interest

Dresden Database Systems Group

Properties of entities

- Captured/measured values
- What are the sales figures/temperatures/etc.?
- Multidimensional data/time series/matrixes

Connections between entities

- Network structure
- What do the friends of your customers buy?
- Graph data

Example: Centrality Measures

Dresden Database Systems Group

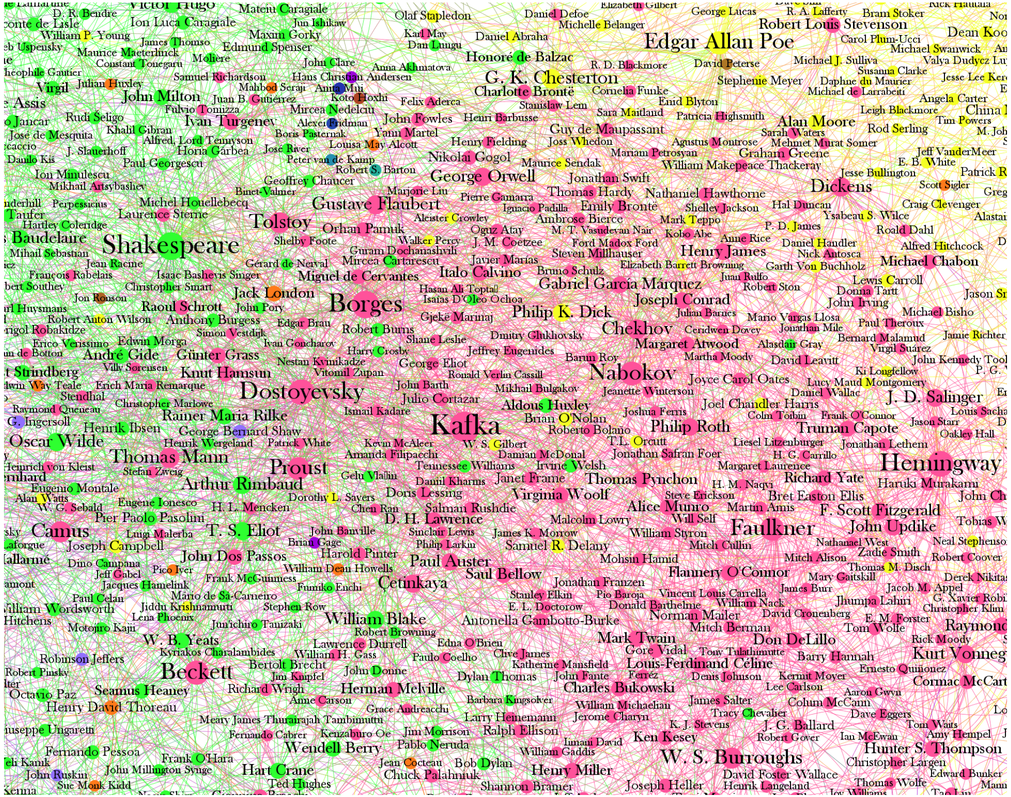
Question: Who are the key players in a graph

- Most social contacts (vaccination schedules)
- Most influential thinkers/papers (reading lists)
- Most important website (web search)
- Most important distributers (supply network)
- etc.

- Can we measure that?

Yes! With centrality Measures!

- Centrality measures identify the most important vertices within a graph



[http://brendangriffen.com/blog/gow-influential-thinkers]

Example: Supply Chain Management

UNITED STATES
CONSUMER PRODUCT SAFETY COMMISSION

Home | Recalls | Safety Education | Regulations, Laws & Standards | Research & Statistics | Busine Manuf

Recall Lawsuits | Violations

Home / Recalls / 2011

HP Expands Recall of Notebook Computer Batteries to Fire Hazard

FOR IMMEDIATE RELEASE
May 27, 2011
Release #11-234

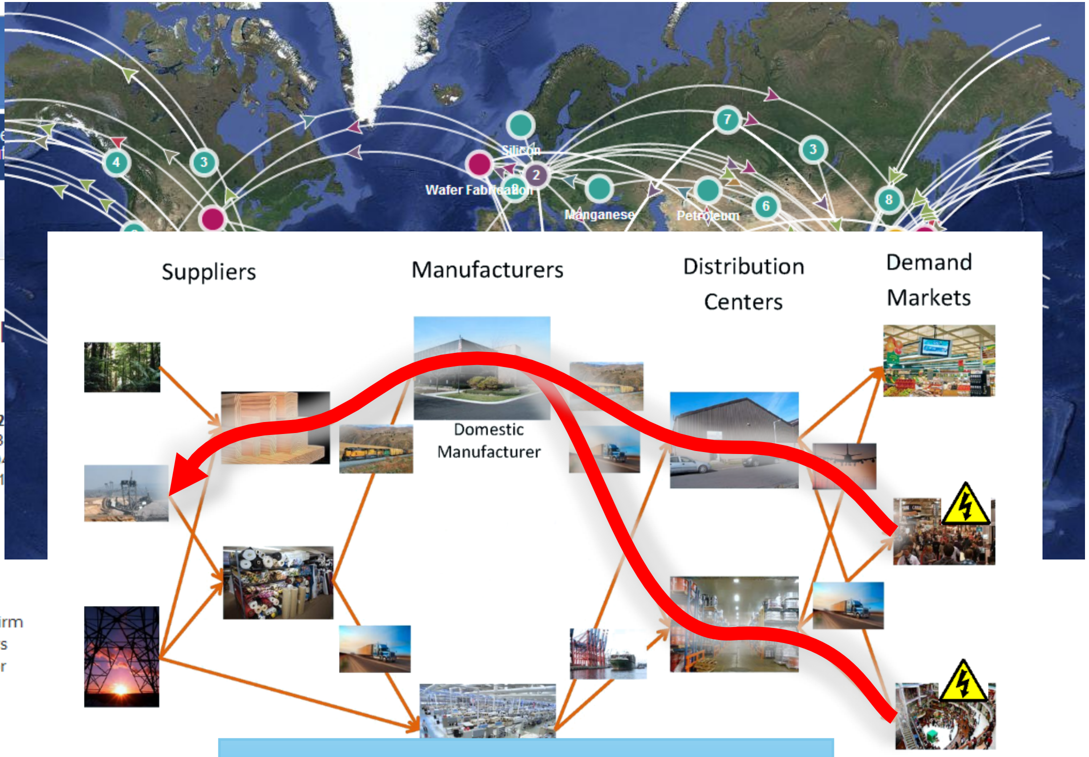
Firm's Recall Hotline: (888) 202
CPSC Recall Hotline: (800) 63
CPSC Media Contact: (301) 50
HP Media Contact: (281) 51

HP Expands Recall of Notebook Computer Batteries Due to Fire Hazard
Consumers urged to recheck notebook models and batteries

WASHINGTON, D.C. - The U.S. Consumer Product Safety Commission, in cooperation with the firm named below, today announced a voluntary recall of the following consumer product. Consumers should stop using recalled products immediately unless otherwise instructed. It is illegal to resell or attempt to resell a recalled consumer product.

Name of Product: Lithium-ion batteries used in HP and Compaq notebook computers

Units: About 162,600 additional batteries (54,000 and 70,000 batteries were previously recalled in May 2010 and May 2009, respectively)



Backward Tracability (Level 3 Analytics)
- Find the parts causing the problem

Example: Supply Chain Management

UNITED STATES
CONSUMER PRODUCT SAFETY COMMISSION

Home | Recalls | Safety Education | Regulations, Laws & Standards | Research & Statistics | Busine Manuf

Recall Lawsuits
Violations

Home / Recalls / 2011

HP Expands Recall of Notebook Computer Batteries to Fire Hazard

FOR IMMEDIATE RELEASE
May 27, 2011
Release #11-234

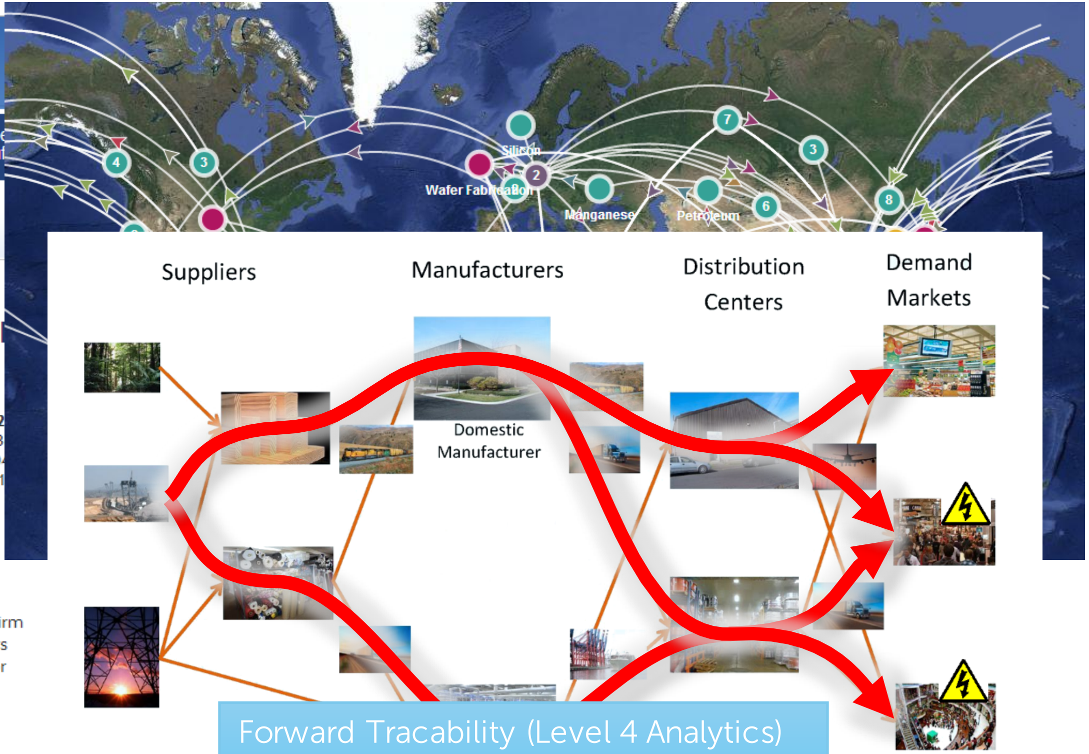
Firm's Recall Hotline: (888) 202
CPSC Recall Hotline: (800) 63
CPSC Media Contact: (301) 50
HP Media Contact: (281) 51

HP Expands Recall of Notebook Computer Batteries Due to Fire Hazard
Consumers urged to recheck notebook models and batteries

WASHINGTON, D.C. - The U.S. Consumer Product Safety Commission, in cooperation with the firm named below, today announced a voluntary recall of the following consumer product. Consumers should stop using recalled products immediately unless otherwise instructed. It is illegal to resell or attempt to resell a recalled consumer product.

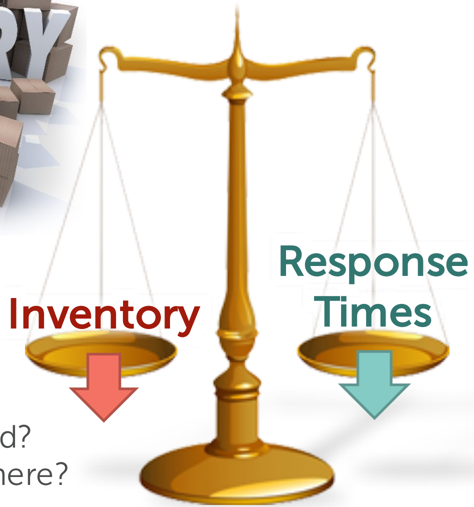
Name of Product: Lithium-ion batteries used in HP and Compaq notebook computers

Units: About 162,600 additional batteries (54,000 and 70,000 batteries were previously recalled in May 2010 and May 2009, respectively)



Forward Tracability (Level 4 Analytics)
- Alert other customer potentially affected by the problem

Example: Supply Chain Management



When to send?
How much to send?
From where to where?

Supply Chain Optimization (Level 8 Analytics)
- Customer A: 10-30% reduction in inventory
- Customer B: 8% reduction in transportation costs

[http://www.logicblox.com/solutions/supply-chain-optimization/]

Business Processes

Dresden Database Systems Group

Business processes are essentially graphs

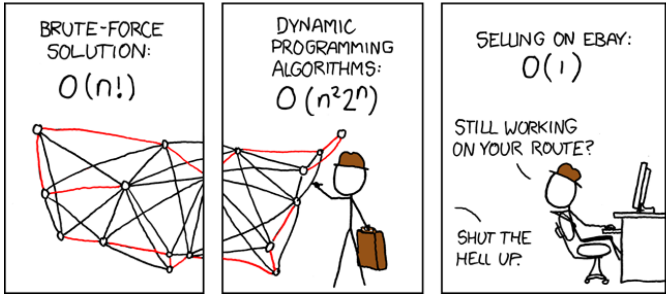
- Who does what in relationship with whom …

[http://integrella.com/services/solutions/business-optimization-bpm/]

The whole analytics stack desired

- From tracking the state of processes (level 0)

- To optimizing the processes (level 8)

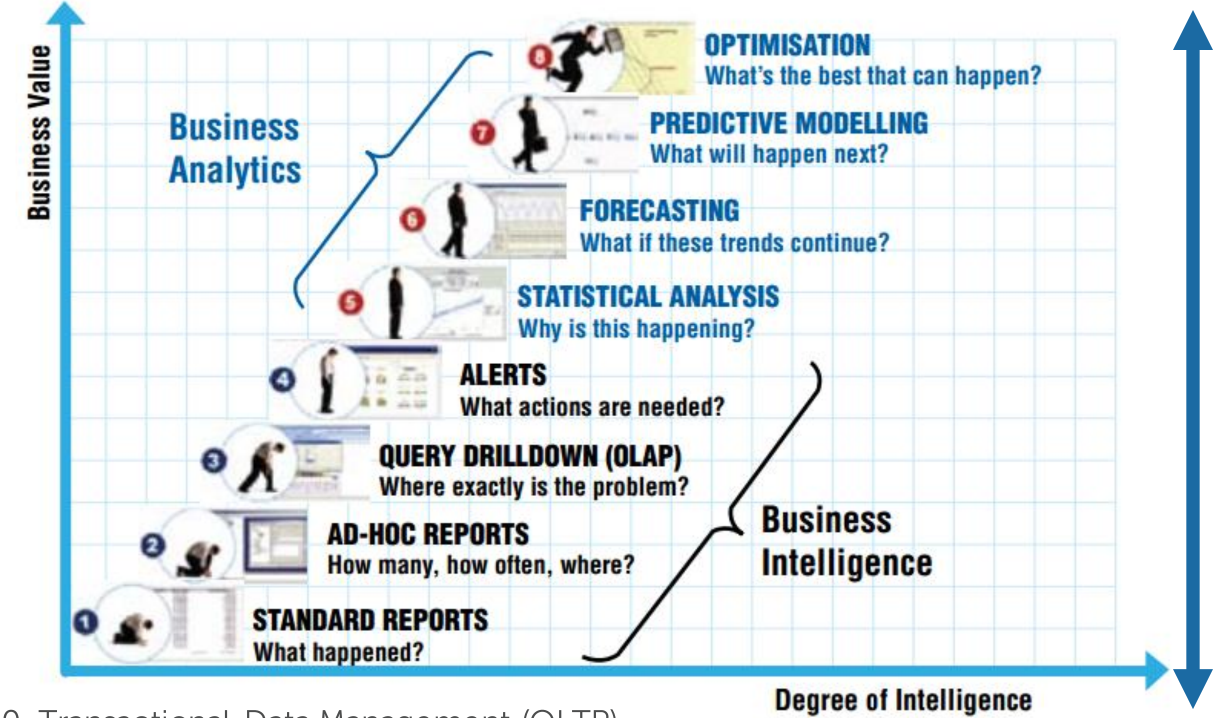


[https://xkcd.com/399/]

Level of Analytics

Dresden Database Systems Group

[http://www.rosebt.com/blog/eight-levels-of-analytics-business-intelligence-to-business-analytics]



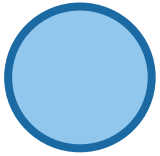
0. Transactional Data Management (OLTP)

Graph Data Management Applications

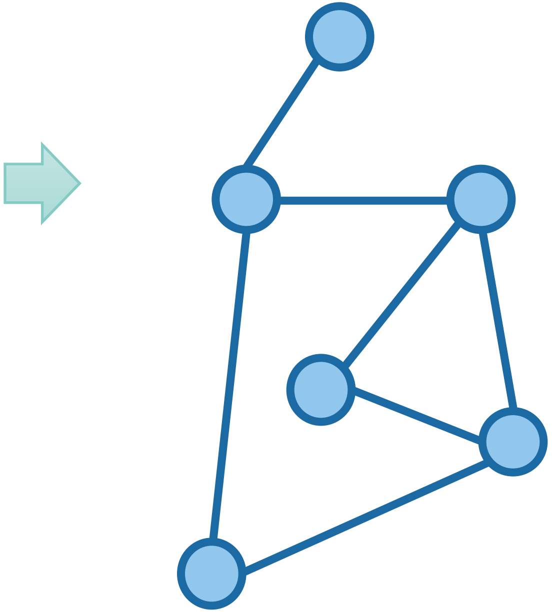
Graph Data Model

Graph Building Blocks

Nodes (Dots)



- Like an entity in ER
- Exist on their own
- Have object identity



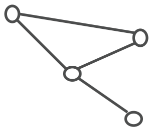
Edges (Lines)

- Like a relationship in ER
- Exist only between nodes
- Identity depends on the nodes they connect

Graph Building Blocks

Vertices & Edges

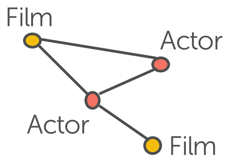
- $G = (V, E)$ with $E \subseteq \{e \mid e \in \mathcal{P}(V) \wedge |e| = 2\}$



- Vertices have identity
- Edge depend on vertices

Vertex Labels

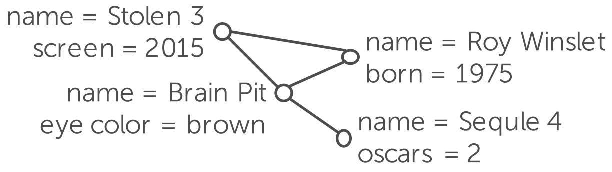
- $G = (V, E, L_V, f_V)$ with $f_V : V \to L_V$ (or $f_V : V \to \mathcal{P}(L_V)$)



- Label are not unique[1]

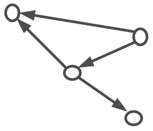
Vertex Properties

- Vertices have set of key-value pairs



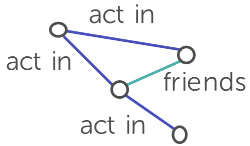
Directionality

- $E \subseteq V \times V$



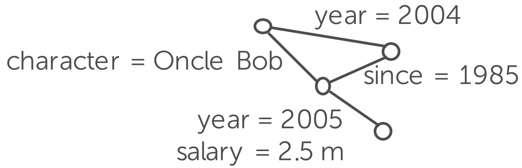
Edge Labels (or weights)

- $G = (V, E, L_E, f_E)$ with $f_E : V \to L_E$



Edge Properties

- Edges have set of key-value pairs

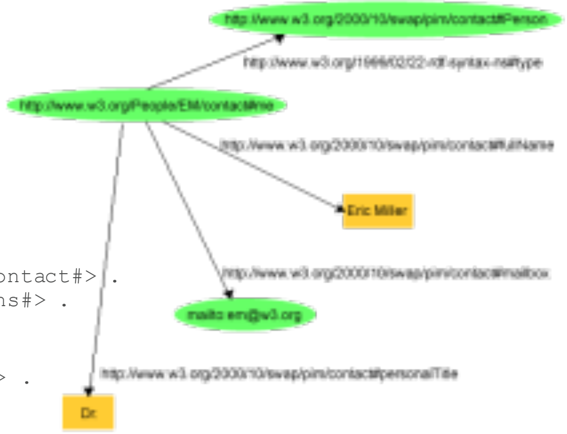


[1] Labels that are required to be unique are not labels but vertex identity

Resource Description Framework (RDF)

Dresden Database Systems Group

- Data descripted in triples subject-predicate-object
- Subjects and objects are vertices (URIs U or value literals L (objects only))
- Predicates are edge labels (URIs U)
- RDF dataset $\subseteq U \times U \times \{U \cup L\}$
- Edges are directed
- No vertex labels
  (note, every literal is per se unique)
- No properties



[http://commons.wikimedia.org/wiki/File:Rdf_graph_for_Eric_Miller.png]

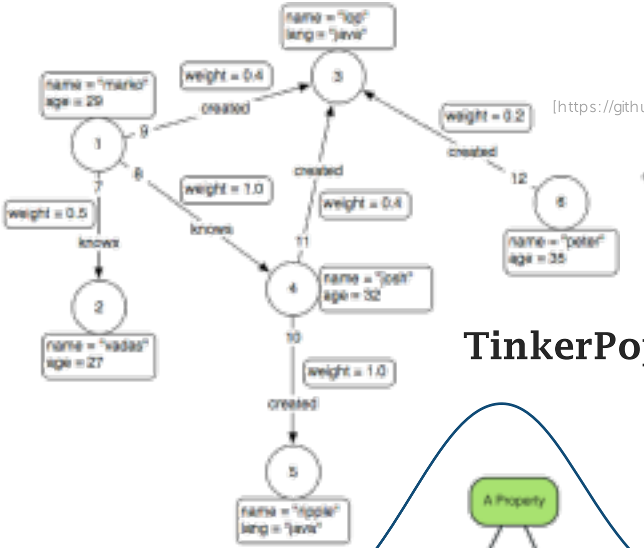
```
@prefix eric: <http://www.w3.org/People/EM/contact#> .
@prefix contact: <http://www.w3.org/2000/10/swap/pim/contact#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

eric:me contact:fullName "Eric Miller" .
eric:me contact:mailbox <mailto:e.miller123(at)example> .
eric:me contact:personalTitle "Dr." .
eric:me rdf:type contact:Person .
```

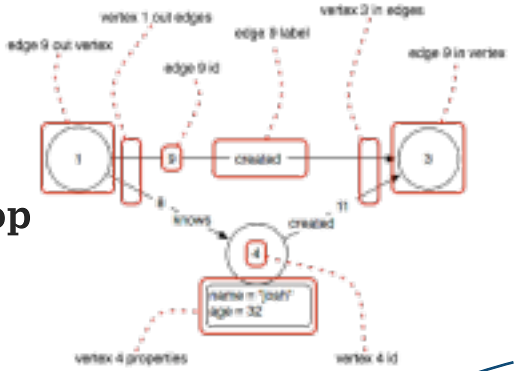
- RDF Schema (RDFS)
- Set of predefined predicates and classes to describe data schema in RDF

[http://lod-cloud.net/]

Property Graph Model

Dresden Database Systems Group

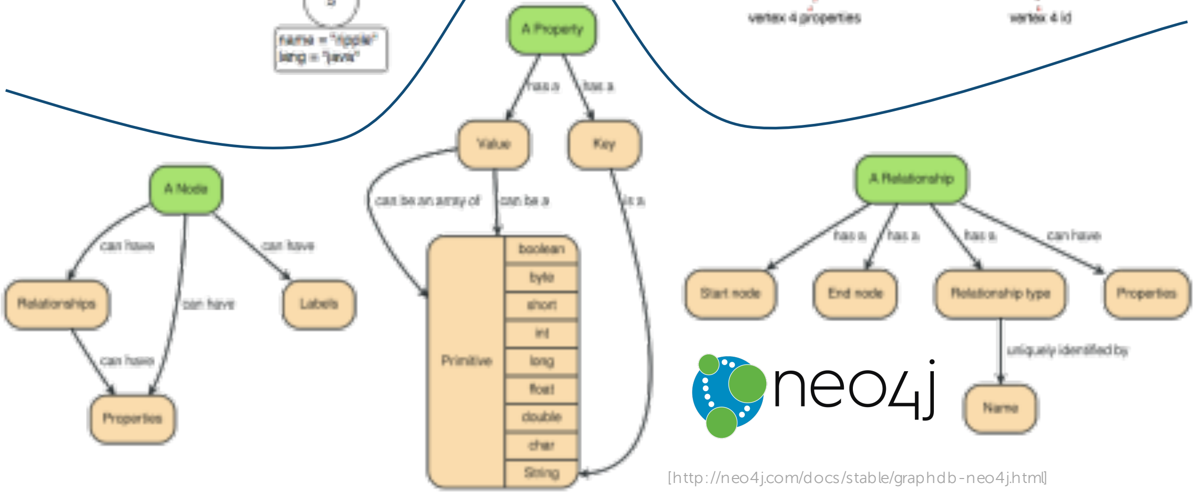




[https://github.com/tinkerpop/blueprints/wiki/Property-Graph-Model]

TinkerPop

- Directed Graph
- Vertices are proper entities with
  - Label (often as type Property)
  - Properties
- Edges are “rich” relationships with
  - Label
  - Properties



neo4j

[http://neo4j.com/docs/stable/graphdb-neo4j.html]

- No standard
- Various different implementations
  - TinkerPop/Gremlin
  - Neo4j (allows multiple vertex labels)
  - Green-Marl (no labels)
  - …

Possible Graph Data Models

Dresden Database Systems Group

		Structure			Plain Data		Structured Data	
		Directionality	Loops & Cycles	Multiple Edges	Vertex Labels	Edge Labels	Vertex Properties	Edge Properties
Pure Structure Graph Models	Basic	○	○	○	○	○	○	○
	DAG	✓	○	○	○	○	○	○
	⋮							
Plain Data Graph Models	RDF	✓	✓	✓	○	✓	○	○
	Pregel Graph Model, Graph-Oriented Object Data model (GOOD)	✓	✓	✓	✓	✓	○	○
Structured Data Graph Models	Green-Marl Graph Model	✓	✓	✓	○	○	✓	✓
	OrientDB	✓	✓	✓	○	✓	✓	✓
	Property Graph, Neo4j, TinkerPop	✓	✓	✓	✓	✓	✓	✓

Relational Representation

RDF

- Triples fits in three-column relational table

Subject	Predicate	Object
<http://www.w3.org/People/EM/contact#me>	<http://www.w3.org/2000/10/swap/pim/contact#fullName>	"Eric Miller"
<http://www.w3.org/People/EM/contact#me>	<http://www.w3.org/2000/10/swap/pim/contact#mailbox>	<mailto:e.miller123(at)example>
<http://www.w3.org/People/EM/contact#me>	<http://www.w3.org/2000/10/swap/pim/contact#personalTitle>	"Dr."
<http://www.w3.org/People/EM/contact#me>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2000/10/swap/pim/contact#Person>

Property Graph

- Two universal tables: one for the vertices, one for the edges

ID	Type	Color	Name	RAM	Nationality
1	Product	black	“Apple iPad MC707LL/A”	64 GB	
...	...	...	...	...	...
4	Category		“Cell Phones & Accessories”		
5	Category		“Phones”		
...	...	...	...	...	...

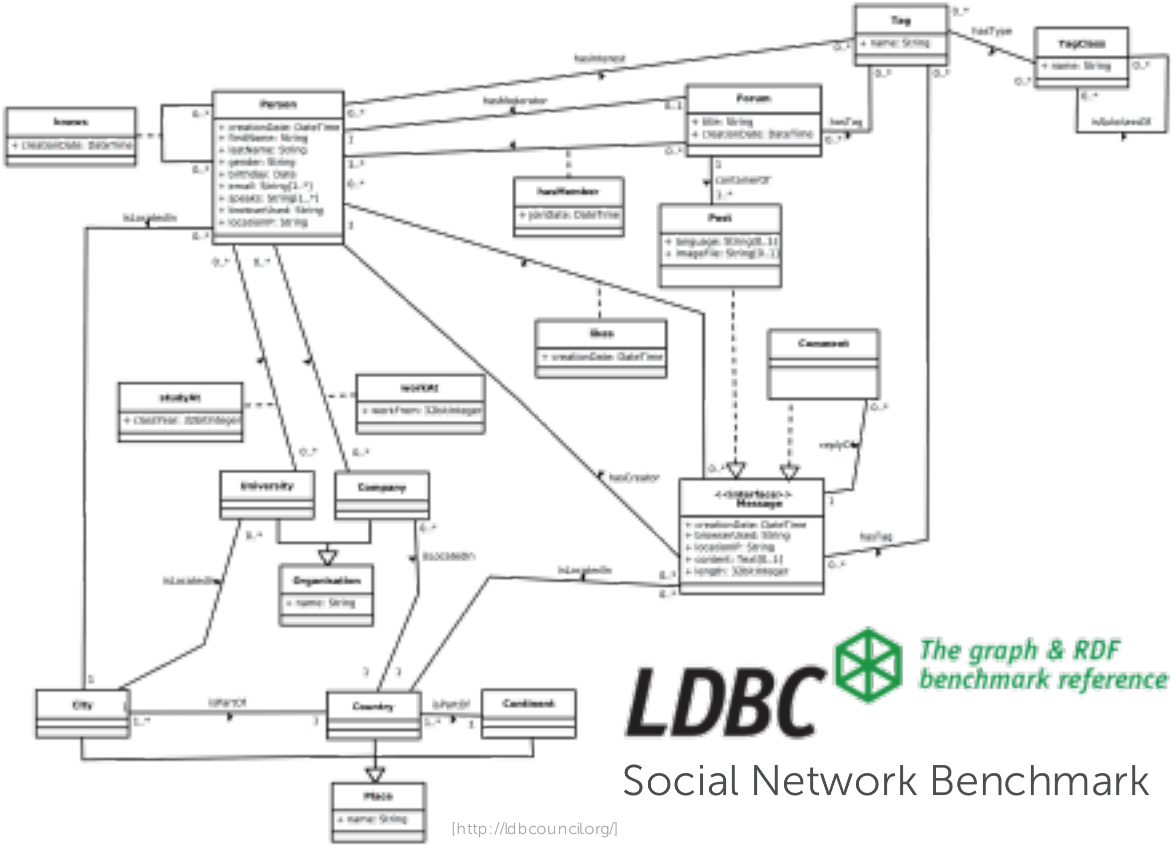
Source	Target	Type	Rating
1	7	in	
...	...	...	...
5	4	part of	
7	6	part of	
...	...	...	...

- Alternative: One universal tables per vertex and edge label (type)

Relational Representation

GRAPH DATA WITH FIXED SCHEMA

- One table for every vertex type and every edge type
- Think of: Two-universal-tables schema partitioned by type
- Edge types representing 1:N relationship can be presented with a simple foreign key

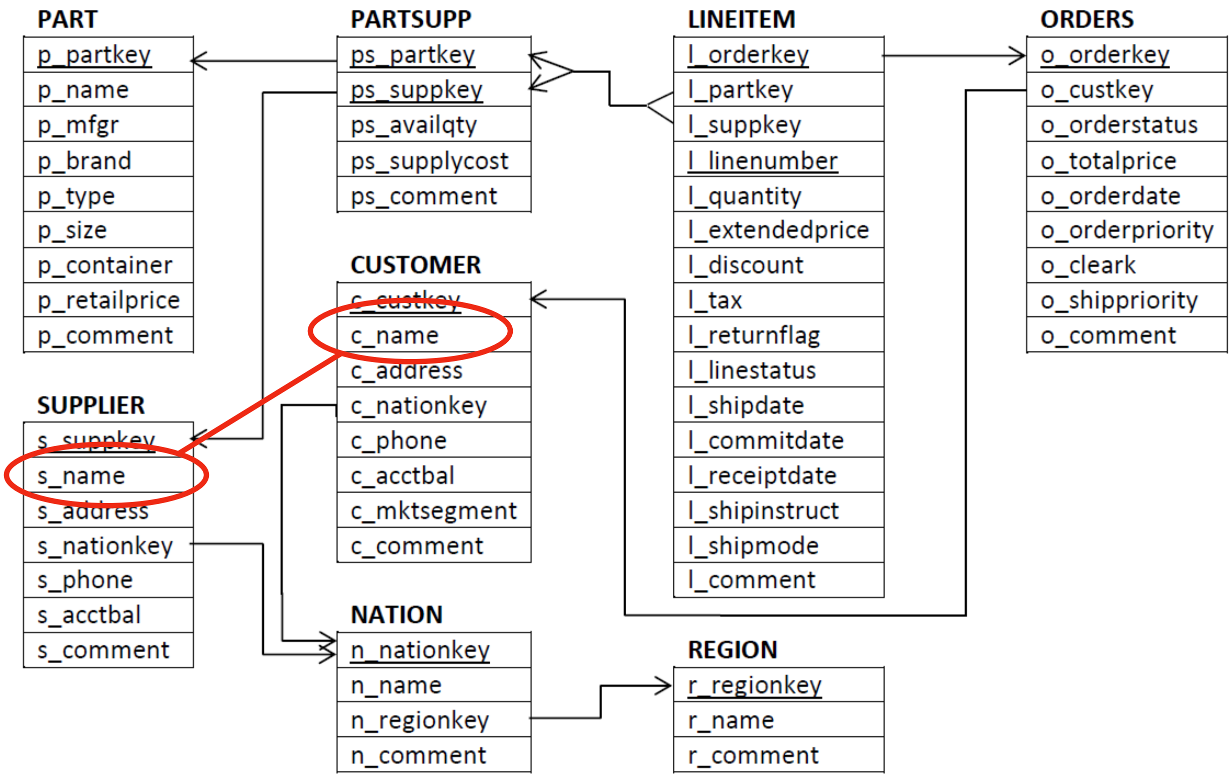


[http://ldbcouncil.org/]

Graph-structure data

Often hidden in non-graph data

- E.g. TPC-H scenario
- Customer that also is a supplier



Graph Models vs. Relational Model

Identity of entities

- Relational: Value-based identity
  - One or more attributes serves as identity and are declared as such per type by primary key constraint
  - Values of identity attributes are user-given
- Graph: Object identity
  - Fixed (visible or hidden) attribute serves as identity
  - Values of identity attribute are either system-generated (e.g. object id) or user-given (e.g. URI)
- Distinction is blurred by bag-semantics of SQL

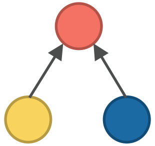
Referencing mechanisms

- Relational: Value-based reference
  - References are expressed by value equality
  - Necessity of referential integrity can be declared by a foreign key constraint
- Graph: Explicit association
  - References are expressed with a dedicated association element -> edges
  - Dedicated association element has referential integrity built in

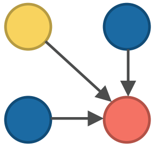
Liberty of referencing

- Relational

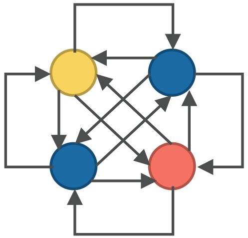
Schema



Data



- Graph



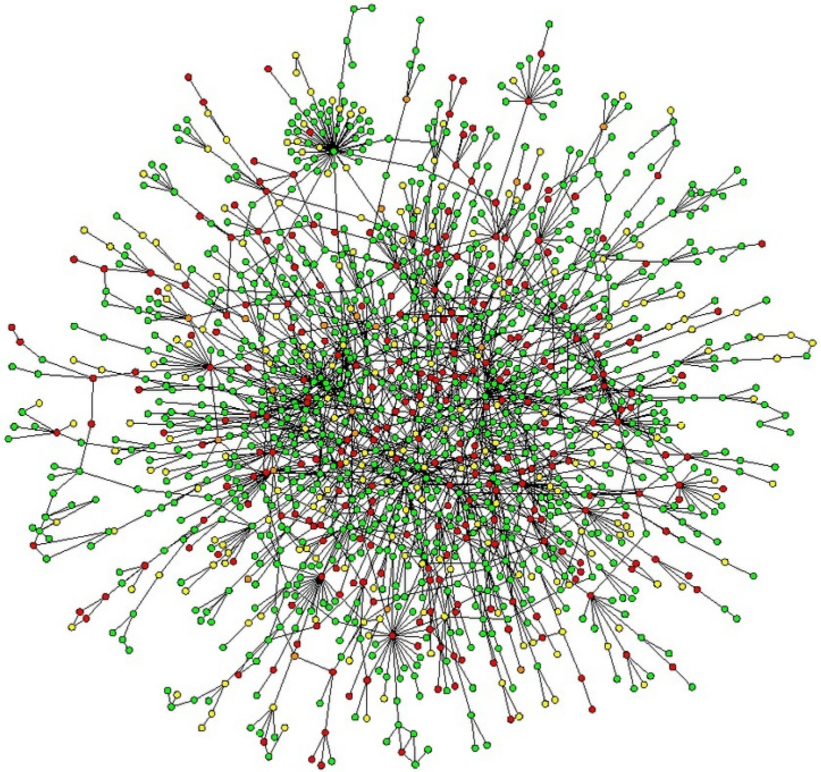
Graph Querying

Querying Graphs

QUESTION

"How is friend of John Doe?"
"Couples where both like 'House of Cards'?"
"Shortest connection between
John Doe and Joe Dohn?"
...

ANSWER

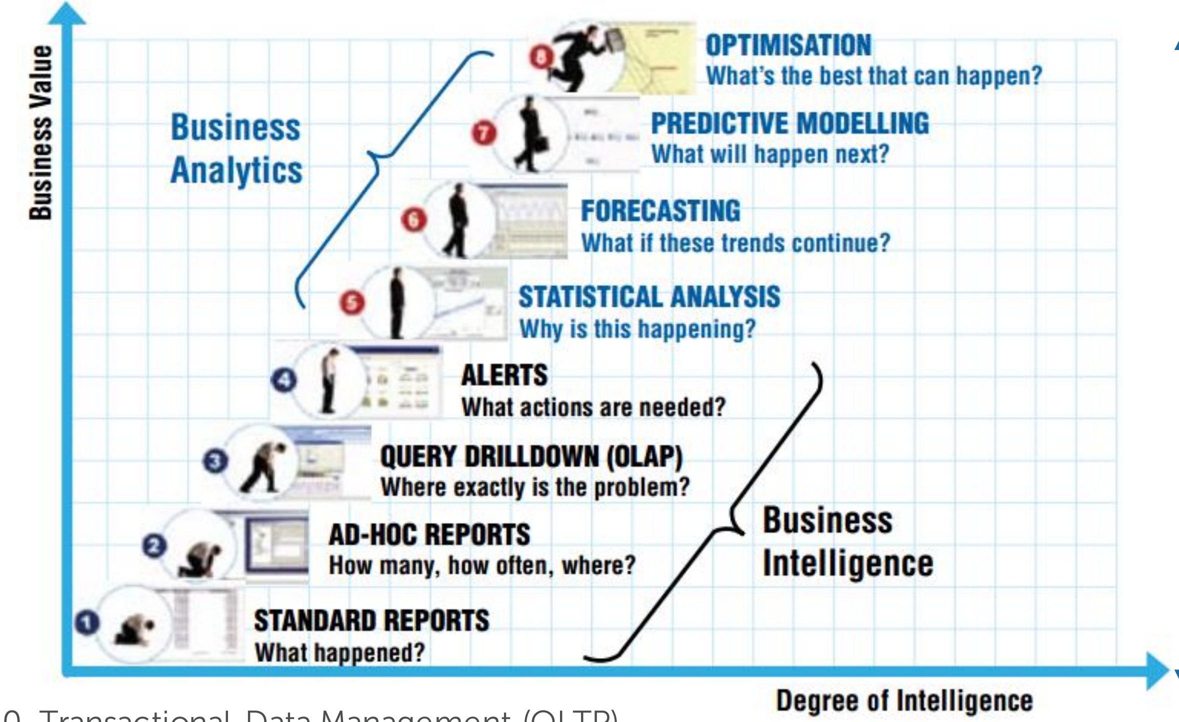


[http://www.bordalierinstitute.com/images/yeastProteinInteractionNetwork.jpg]

Level of Analytics

Dresden Database Systems Group

[http://www.rosebt.com/blog/eight-levels-of-analytics-business-intelligence-to-business-analytics]



0. Transactional Data Management (OLTP)

batch
long running
imperative

Graph Data Management Applications

ad-hoc
short running
declarative

OFFLINE GRAPH ANALYTICS

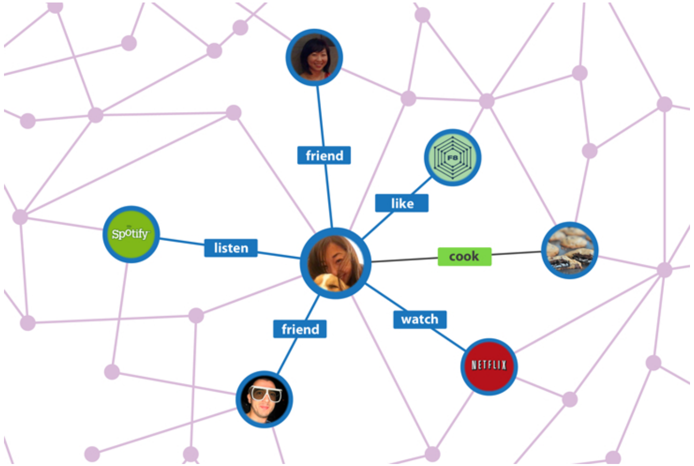
ONLINE GRAPH QUERYING

Graph Workloads

Dresden Database Systems Group

Online Graph Querying

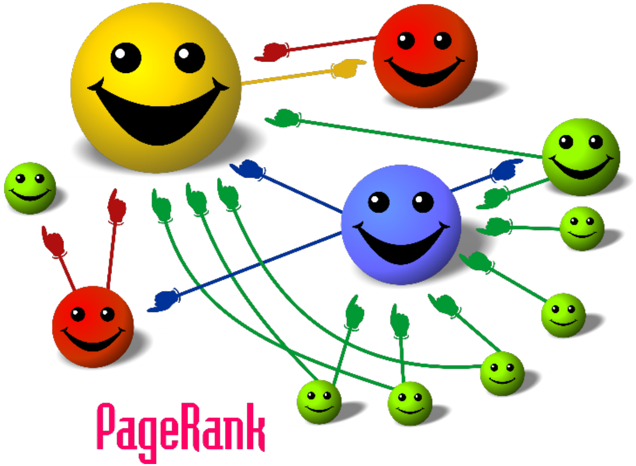
- OLTP-Style
- Short, read and write, high selectivity
- Interest in proximity of one or more start nodes
- E.g., Loading and updating of you Facebook page, Facebook Graph Search



[http://blackfin360.com/2013/01/15/facebook-graph-search/]

Offline Graph Analytics

- OLAP style
- Long, expensive, mainly read, low selectivity
- Topological analysis of whole graph
- E.g. Page rank (centrality), shortest path, connected components, …

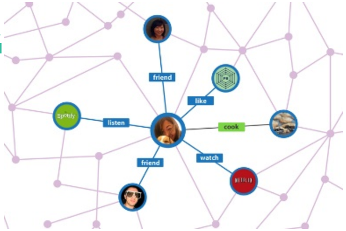


[http://commons.wikimedia.org/wiki/File:PageRank-hi-res.png]

Graph Query Concepts

Dresden Database Systems Group

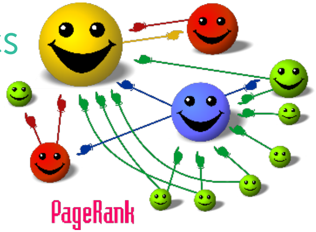
ONLINE GRAPH QUERYING



DECLARATIVE

- DataLog, SPARQL, RQL, Cypher, …
- Focuses on the What
- Abstracts from the How
- Limited compared to normal programming languages
- Allows optimization: Optimizer takes care of the How
- Hides technical, low-level concerns, e.g. selectivities, parallelization, etc.

OFFLINE GRAPH ANALYTICS



IMPERATIVE

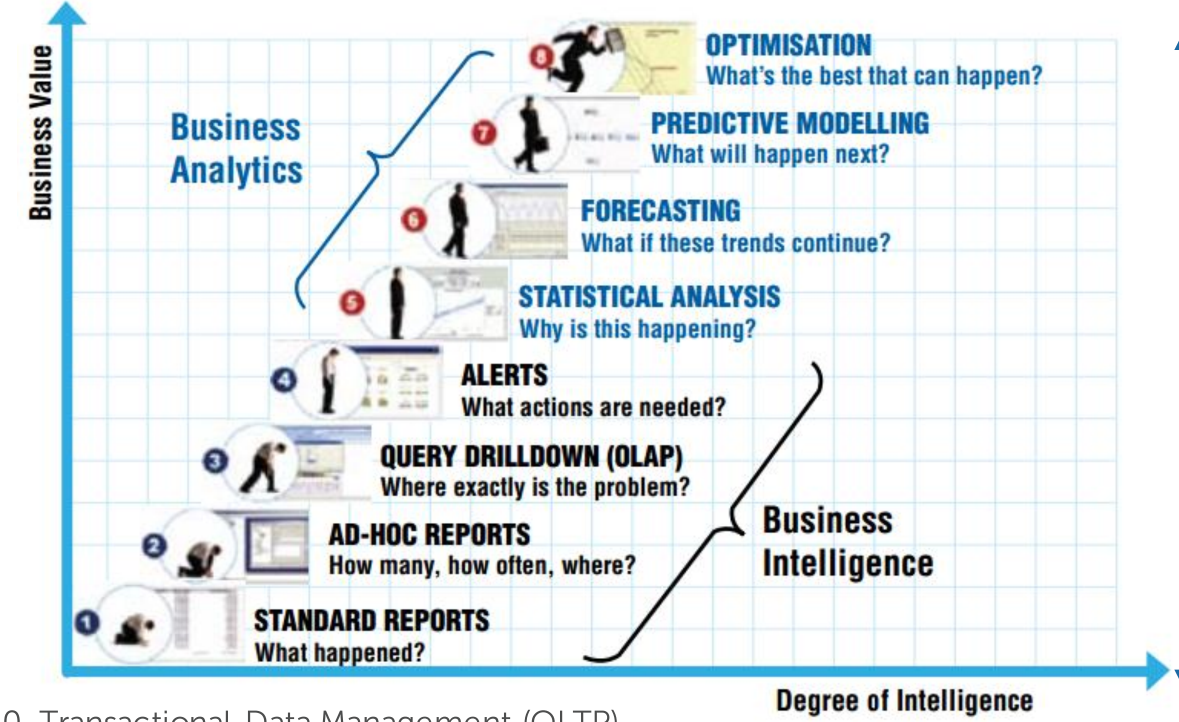
- Gremlin, GreenMarl, Travel, Pregel, GraphLab, …
- DSLs or APIs
- Focuses on the How
- Sets of commands
  - Graph traversal and access
  - General-propose programming language constructs
- (Almost) no restriction in expression power compared to normal programming languages
- Comfortable graph navigation and access

*[declarative sentence spoken by Don Corleone in The Godfather; http://grammar.about.com/od/d/g/declsenterm.htm]

**[Imperative sentences spoken by Clemenza in The Godfather; http://grammar.about.com/od/il/g/impersent09.htm]

Level of Analytics

[http://www.rosebt.com/blog/eight-levels-of-analytics-business-intelligence-to-business-analytics]



0. Transactional Data Management (OLTP)

batch
long running
imperative

Graph Data Management Applications

ad-hoc
short running
declarative

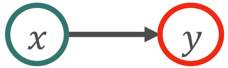
IMPERATIVE GRAPH ANALYTICS

DECLARATIVE GRAPH QUERYING

Online Graph Querying – Pattern Matching

Graph Pattern Matching

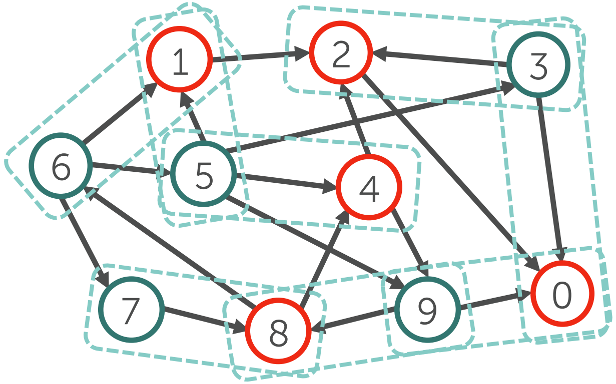
PATTERN p



Graph with place holders (A,B)

MATCHING p ON G

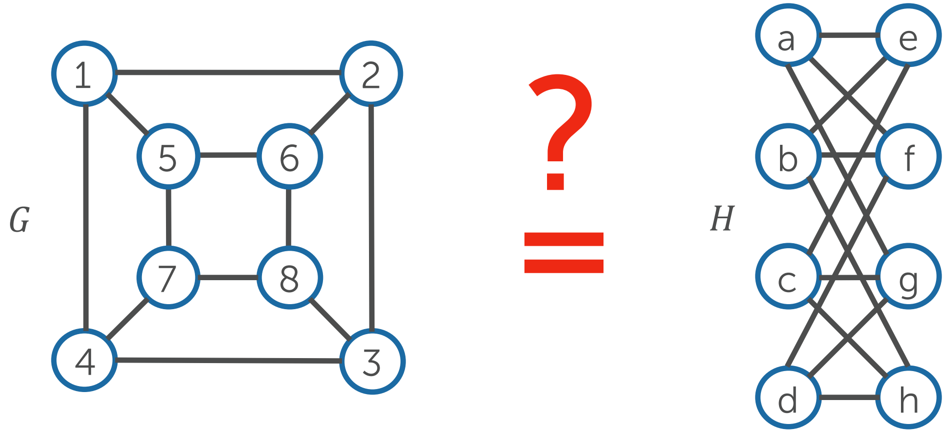
Finds all subgraphs in G that fit to p



Similarity of two Graphs

Dresden Database Systems Group

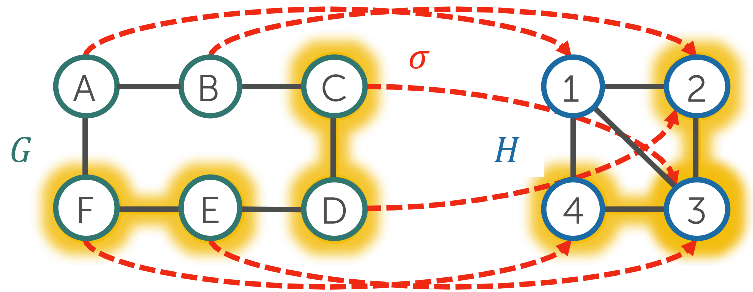
ARE GRAPHS G AND H EQUAL/SIMILAR?

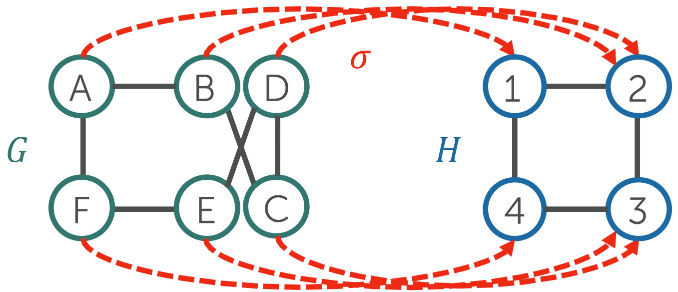


MANY SIMILARITY CRITERIA

- Isomorphism
- Homomorphism
- Simulation
- Bisimilarity
- …

Graphs Homomorphism





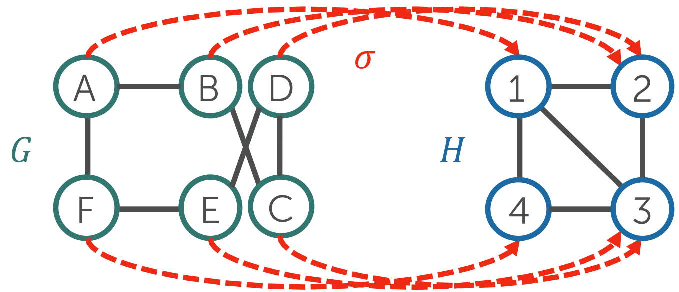
- Given two graphs $G(V_G, E_G)$ and $H(V_H, E_H)$
- G and H are homomorph, if there is a surjective function $\sigma: V_G \rightarrow V_H$ (left-total, right-total, right-unique) such that $(v_i, v_j) \in E_G \rightarrow \left(\sigma(v_i), \sigma(v_j)\right) \in E_H$ (G preserves adjacency of H, i.e. an edge in G has to exist in H as well)

- Note, there may be multiple functions σ, i.e., multiple homomorphism between two graphs

Graph Isomorphism

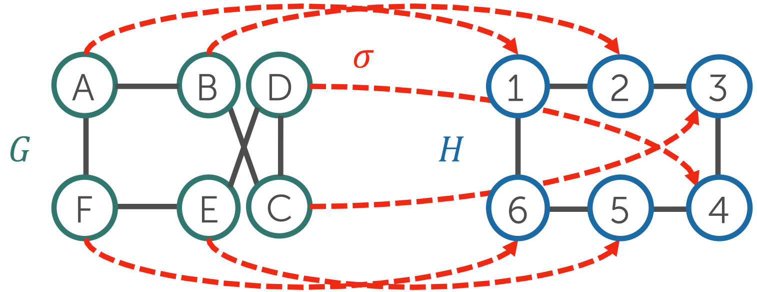
Dresden Database Systems Group

Graph Homomorphism



- Given two graphs $G(V_G, E_G)$ and $H(V_H, E_H)$
- G and H are homomorph,
- if there is a surjective function $\sigma: V_G \to V_H$ (left-total, right-total, right-unique)
  such that $(v_i, v_j) \in E_G \to \left(\sigma(v_i), \sigma(v_j)\right) \in E_H$
  (G preserves adjacency of H)

Graph Isomorphism



- Given two graphs $G(V_G, E_G)$ and $H(V_H, E_H)$
- G and H are isomorph,
- if there is a bijective function $\sigma: V_G \to V_H$ (left-total, left-unique, right-total, and right-unique)
  such that $(v_i, v_j) \in E_G \leftrightarrow \left(\sigma(v_i), \sigma(v_j)\right) \in E_H$
  (G preserves adjacency and non-adjacency of H and vice versa)

Cool!

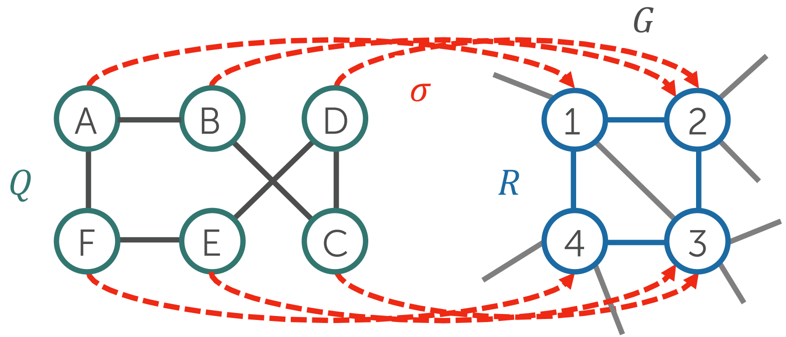
Can we apply this to find subgraphs?

[see also video tutorial on graph gsomorphism by Sarada Herke: https://www.youtube.com/watch?v=yFpRpxOry-A]

TECHNISCHE UNIVERSITÄT DRESDEN

Subgraph Homomorphism Query

Dresden Database Systems Group

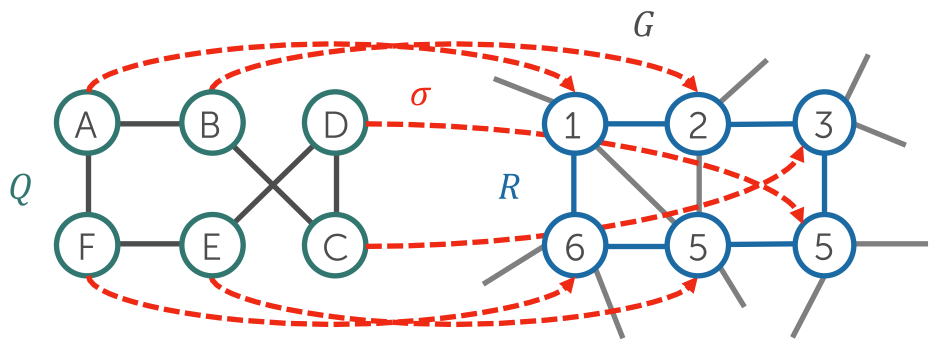


- Given a query graphs $Q(V_G, E_G)$ and data graph $G(V_H, E_H)$
- Graph $R(V_R, E_R)$ is a result for Q if
  - $V_R \subseteq V_G$ and $E_R \subseteq E_G$ (R is a subgraph of G) and
  - there is a surjective function $\sigma: V_G \rightarrow V_H$ (Q and R are homomorph) such that $(v_i, v_j) \in E_Q \leftrightarrow \left(\sigma(v_i), \sigma(v_j)\right) \in E_R$ (Q preserves adjacency of R with no extra edges in R) and $\forall(v_Q, v_R) \in \sigma, v_Q \sim v_R$ (vertex properties match) and $\forall(v_i, v_j) \in E_Q, (v_i, v_j) \sim \left(\sigma(v_i), \sigma(v_j)\right)$ (edge properties match)
- Q can have more vertices and edges than R, i.e., $|V_Q| \geq |V_R|$ and $|E_Q| \geq |E_R|$ holds.

- Note: R is not given, R has to be determined by the query mechanism -> search problem

Subgraph Isomorphism Query

Dresden Database Systems Group



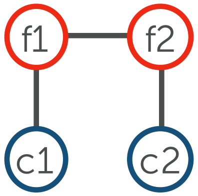
- Given a query graphs $Q(V_G, E_G)$ and data graph $G(V_H, E_H)$
- Graph $R(V_R, E_R)$ is a result for Q if
  - $V_R \subseteq V_G$ and $E_R \subseteq E_G$ (R is a subgraph of G) and
  - there is a bijective function $\sigma: V_G \rightarrow V_H$ (Q and R are isomorph) such that $(v_i, v_j) \in E_Q \leftrightarrow \big(\sigma(v_i), \sigma(v_j)\big) \in E_R$ (Q preserves adjacency of R with no extra edges in R) and $\forall(v_Q, v_R) \in \sigma, v_Q \sim v_R$ (vertex properties match) and $\forall(v_i, v_j) \in E_Q, (v_i, v_j) \sim \big(\sigma(v_i), \sigma(v_j)\big)$ (edge properties match)
- Q and R will have the same number of vertices and edges, i.e., $|V_Q| = |V_R|$ and $|E_Q| = |E_R|$ holds.

Single vertex in V_G can be matched
multiple times in a homomorphic subgraph but only once in an isomorphic subgraph.

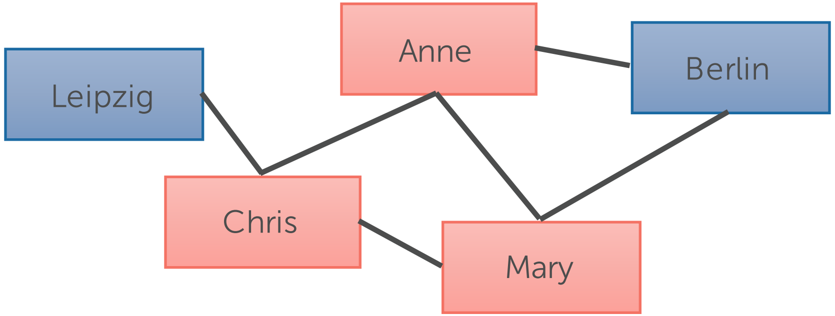
Why Homomorphism is useful…

Example: Look for all pairs of friends and the city each friend lives in

Query



Data



Isomorphism finds only friends living in different cities

- (Leipzig, Chris, Anne, Berlin), (Leipzig, Chris, Mary, Berlin) … and permutation of these

Homomorphism additionally finds friends living in the same city

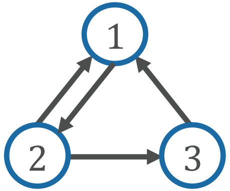
- (Berlin, Mary, Anne, Berlin) … and permutation of these

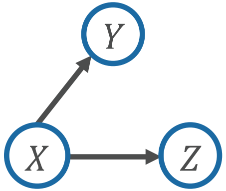
Induced Subgraph Isomorphism

Dresden Database Systems Group

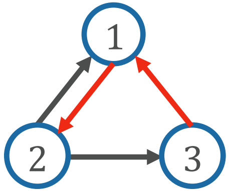
Example

- Data graph: Query graph:





- Does it have a match? How many?
- One solution:



- What about the other edges? Could we forbid them?
- With induced subgraph isomorphism semantics, example query has no match!

Induced subgraph

- Vertex-induced
- Is a subset of the vertices of a graph together with any edges whose endpoints are both in this subset.

Induced subgraph isomorphism

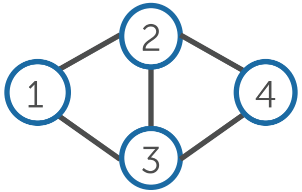
- Stricter isomorphism
- Given query graph $Q(V_Q, E_Q)$ and data graph $G(V_G, E_G)$
- Graph $R(V_R, E_R)$ is a result for Q if
  - $V_R \subseteq V_G$ and $\boldsymbol{E_R = \{(v_i, v_j) | (v_i, v_j) \in E_G \land v_i, v_j \in V_R\}}$ (R is a vertex-induced subgraph of G) and
  - there is a bijective function $\sigma: V_Q \rightarrow V_R$ (Q and R are isomorph) such that …
- In graph query languages, typically explicit negation use instead
- Induced subgraph homomorphism also possible

Duplicate Results

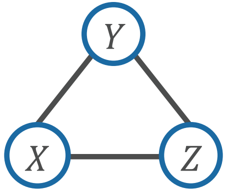
Dresden Database Systems Group

Example: Find triangles

- Data graph:



- Query: All triangles




```
q(X,Y,Z) <- e(X,Y),e(Y,Z),e(Z,X).
```

- Result:

Duplicate results, same subgraph but different isomorphism σ

x	y	z
1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1
2	3	4
2	4	3
3	2	4
3	4	2
4	2	3
4	3	2

Duplicate results

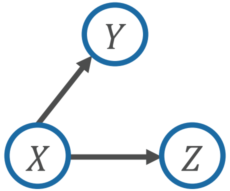
- Given two results $R(V_R, E_R)$ and $S(V_S, E_S)$
- R and S are equivalent iff $V_R = V_S$ and $E_R = E_S$ (both denote the same subgraph)

Subgraph isomorphism w/o duplicates

- Given query graph $Q(V_Q, E_Q)$ and data graph $G(V_G, E_G)$
- Graph $R(V_R, E_R)$ is a result for Q if
  - $V_R \subseteq V_G$ and $E_R \subseteq E_G$ (R is a subgraph of G) and
  - there is a bijective function $\sigma: V_Q \rightarrow V_R$ (Q and R are isomorph) such that $(v_i, v_j) \in E_Q \leftrightarrow \left(\sigma(v_i), \sigma(v_j)\right) \in E_R$ (R preserves adjacency of Q with no extra data) and properties match and $\forall v_i, v_j \in V_Q: \left(v_i <_{V_Q} v_j\right) \leftrightarrow \left(\sigma(v_i) <_{V_R} \sigma(v_j)\right)$ assuming a total order $<_V$ on a vertex set V (allows only one σ per subgraph)

Comparison of Semantics

Dresden Database Systems Group



q(X,Y,Z) <-	INDUCED-SUBGRAPH ISOMORPHISM	⊆	SUBGRAPH ISOMORPHISM	⊆	SUBGRAPH HOMOMORPHISM
w/ duplicates	e(X,Y),e(X,Z), !e(Y,X),!e(Y,Z), !e(Z,Y),!e(Z,X), X!=Y,Y!=Z,Z!=X.		e(X,Y),e(X,Z), X!=Y,Y!=Z,Z!=X.		e(X,Y),e(X,Z).
w/o duplicates	e(X,Y),e(X,Z), !e(Y,X),!e(Y,Z), !e(Z,Y),!e(Z,X), X<Y,Y<Z.		e(X,Y),e(X,Z), X<Y,Y<Z.		e(X,Y),e(X,Z), X<=Y,Y<=Z.

SPARQL

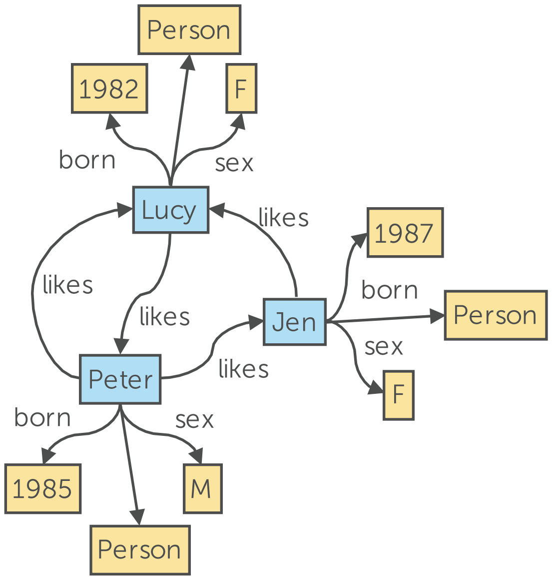
Query language for RDF data

- Selection of subgraph with a triple patterns
- Triple pattern is a set of triples containing variables
- One variable binding of a pattern forms a tuple
- All unique variable binding form a table
- Projection to variable of interest yields query result

```
SELECT   ?p, ?s
WHERE    ?p type Person
         ?p likes ?f
         ?f type Person
         ?f sex ?s
```

?p	?s

S	P	O
Lucy	born	1982
Peter	born	1985
Jen	born	1987
Lucy	sex	F
Peter	sex	M
Jen	sex	F
Lucy	likes	Peter
Peter	likes	Lucy
Jen	likes	Lucy
Peter	likes	Jen



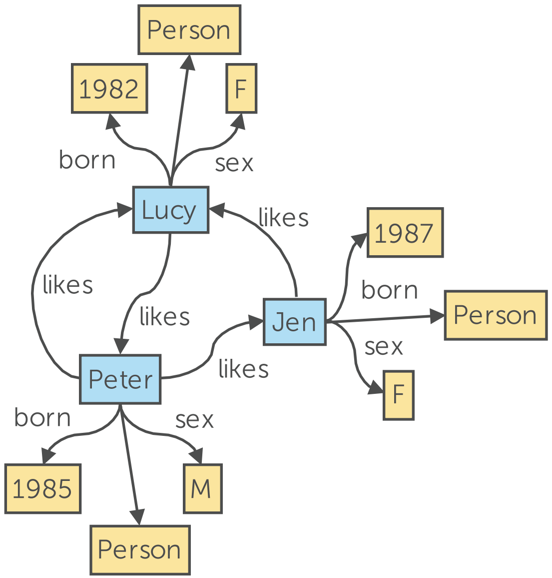
SPARQL

Dresden Database Systems Group

Query language for RDF data

- Selection of subgraph with a triple patterns
- Triple pattern is a set of triples containing variables
- One variable binding of a pattern forms a tuple
- All unique variable binding form a table
- Projection to variable of interest yields query result

S	P	O
Lucy	born	1982
Peter	born	1985
Jen	born	1987
Lucy	sex	F
Peter	sex	M
Jen	sex	F
Lucy	likes	Peter
Peter	likes	Lucy
Jen	likes	Lucy
Peter	likes	Jen




```
SELECT   ?p, ?s
WHERE    ?p type Person
         ?p likes ?f
         ?f type Person
         ?f sex ?s
```

?p	?s
Lucy	M
Peter	F
Jen	F
Peter	F

```
SELECT   ?p, ?fof
WHERE    ?p type Person
         ?p likes ?f
         ?f type Person
         ?f like ?fof
```

?p	?fof

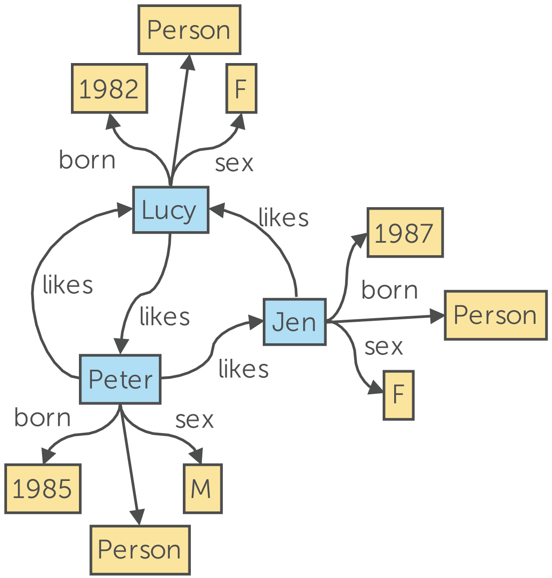
SPARQL

Dresden Database Systems Group

Query language for RDF data

- Selection of subgraph with a triple patterns
- Triple pattern is a set of triples containing variables
- One variable binding of a pattern forms a tuple
- All unique variable binding form a table
- Projection to variable of interest yields query result

S	P	O
Lucy	born	1982
Peter	born	1985
Jen	born	1987
Lucy	sex	F
Peter	sex	M
Jen	sex	F
Lucy	likes	Peter
Peter	likes	Lucy
Jen	likes	Lucy
Peter	likes	Jen




```
SELECT  ?p, ?s
WHERE   ?p type Person
        ?p likes ?f
        ?f type Person
        ?f sex ?s
```

?p	?s
Lucy	M
Peter	F
Jen	F
Peter	F

```
SELECT  ?p, ?fof
WHERE   ?p type Person
        ?p likes ?f
        ?f type Person
        ?f like ?fof
```

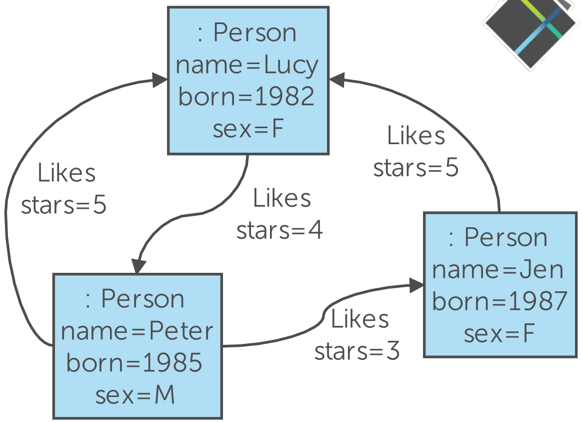
?p	?fof
Lucy	Jen
Peter	Lucy
Peter	Peter
Jen	Peter
Lucy	Lucy

Cypher neo4j

[http://neo4j.com/docs/2.1.6/cypher-query-lang.html]
[http://neo4j.com/docs/stable/cypher-refcard/]

MATCH-Clause

- Primary way of getting data from a Neo4j database
- Allows you to specify the patterns
- Named pattern element, e.g. (p:Person), will be bound to the match instance
- Query can have multiple MATCH-clauses



p.name	f.sex

WHERE-Clause (Optional)

- Allows additional complex predicates in the pattern
- Allows joining two matches

```
MATCH (p:Person)-[:Likes]->(f:Person)
RETURN p.name, f.sex
```

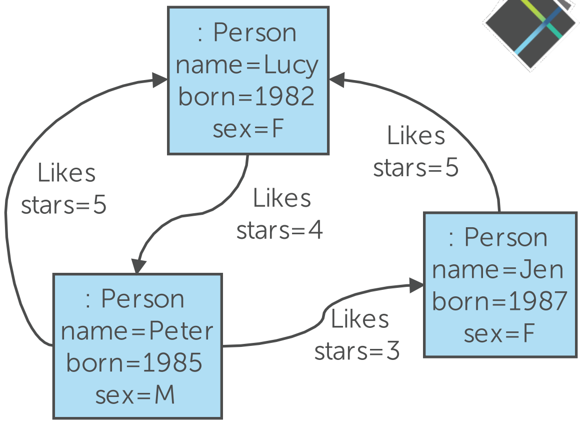
RETURN-Clause

- Projects to the result set
- Allows projection to nodes, edges, and properties

ORDER BY-Clause (like in SQL)

Cypher neo4j

[http://neo4j.com/docs/2.1.6/cypher-query-lang.html]
[http://neo4j.com/docs/stable/cypher-refcard/]



MATCH-Clause

- Primary way of getting data from a Neo4j database
- Allows you to specify the patterns
- Named pattern element, e.g. (p:Person), will be bound to the match instance
- Query can have multiple MATCH-clauses

WHERE-Clause (Optional)

- Allows additional complex predicates in the pattern
- Allows joining two matches

RETURN-Clause

- Projects to the result set
- Allows projection to nodes, edges, and properties

```
MATCH (p:Person)-[:Likes]->(f:Person)
RETURN p.name, f.sex
```

p.name	f.sex
Lucy	M
Peter	F
Jen	F
Peter	F

p.name	fof.name

ORDER BY-Clause (like in SQL)

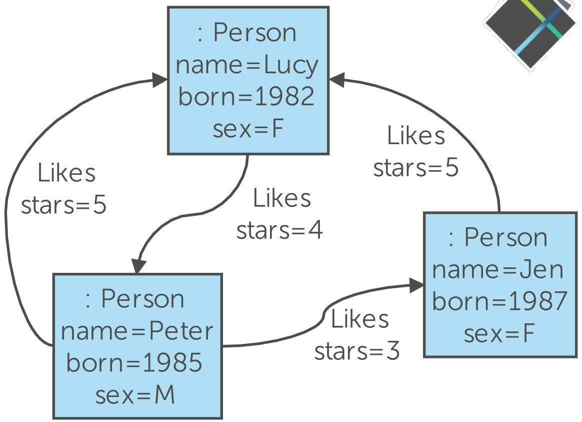
```
MATCH (p:Person)-[:Likes]->(:Person) -[:Likes]->(fof:Person)
RETURN p.name, fof.name
```

Cypher neo4j

[http://neo4j.com/docs/2.1.6/cypher-query-lang.html]
[http://neo4j.com/docs/stable/cypher-refcard/]

MATCH-Clause

- Primary way of getting data from a Neo4j database
- Allows you to specify the patterns
- Named pattern element, e.g. (p:Person), will be bound to the match instance
- Query can have multiple MATCH-clauses



WHERE-Clause (Optional)

- Allows additional complex predicates in the pattern
- Allows joining two matches

RETURN-Clause

- Projects to the result set
- Allows projection to nodes, edges, and properties

```
MATCH (p:Person)-[:Likes]->(f:Person)
RETURN p.name, f.sex
```

p.name	f.sex
Lucy	M
Peter	F
Jen	F
Peter	F

ORDER BY-Clause (like in SQL)

```
MATCH (p:Person)-[:Likes]->(:Person) -[:Likes]->(fof:Person)
RETURN p.name, fof.name
```

p.name	fof.name
Lucy	Jen
Peter	Lucy
Peter	Peter
Jen	Peter
Lucy	Lucy

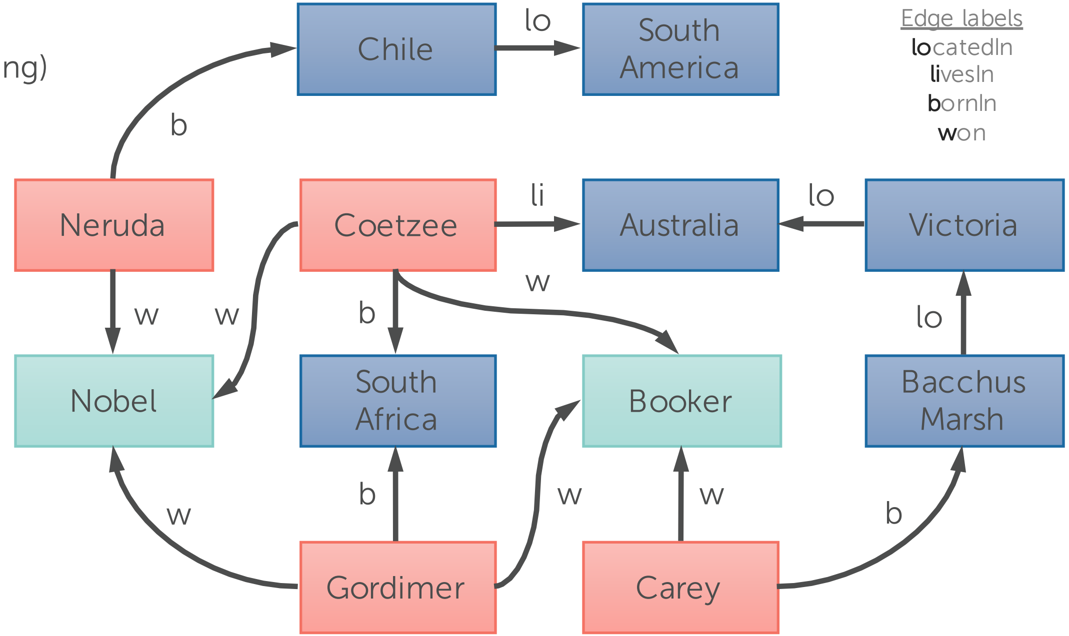
Online Graph Querying – Query Types

Types of Graph Pattern Queries

[Peter T. Wood. Query Languages for Graph Database. SIGMOD Record 41(1), 50–60, March 2012]

Overview

- Conjunctive queries (standard subgraph matching)
- Regular path queries (reachability)
- Conjunctive regular path query
- …



Edge labels
locatedIn
livesIn
bornIn
won

Graph data model for following

- RDF-like data
- Graph $G(V, E, \Sigma)$ with
  - V being the set of vertices,
  - $E \subseteq V \times \Sigma \times V$ being the set of labeled edges,
  - and Σ being the set (or alphabet) of labels

Conjunctive Queries (Std. Matching)

Dresden Database Systems Group

[Peter T. Wood. Query Languages for Graph Database. SIGMOD Record 41(1), 50–60, March 2012]

IDEA

- Query is given as a set of edge predicates
- Each edge predicate consists of a pair vertex variables and an edge label
- A set of variable bindings is a valid answer iff all predicates hold on the data graph

DEFINITION

- Query Q is an expression

$$ans(z_1, \dots, z_n) \leftarrow \bigwedge_{1 \le i \le m} (x_i, a_i, y_i)$$

- Each $x_i \in X$ and $y_i \in Y$ is a vertex variable or a constant from V
- Each $a_i \in \Sigma$ is an edge label
- Each z_i is some x_i or y_i

Correspond to homomorphism

SEMANTICS

- Let $\sigma: X \cup Y \rightarrow V$ be a specific selection of variable bindings, i.e., a mapping to vertices of G
- Say relation $(G, \sigma) \vDash Q$ holds iff $\big(\sigma(x_i), a_i, \sigma(y_i)\big) \in E$ for $1 \le i \le m$, i.e., σ maps the query pattern to valid subgraphs of G
- Then the query result $Q(G)$ is the set of tuples $\big(\sigma(z_1), \dots, \sigma(z_n)\big)$ such that $(G, \sigma) \vDash Q$

Conjunctive Queries (Std. Matching)

Dresden Database Systems Group

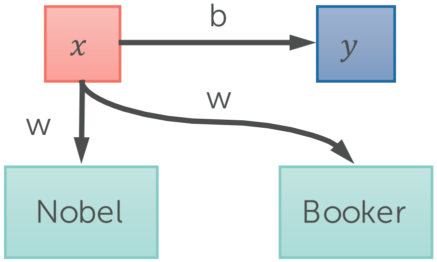
[Peter T. Wood. Query Languages for Graph Database. SIGMOD Record 41(1), 50–60, March 2012]

Example

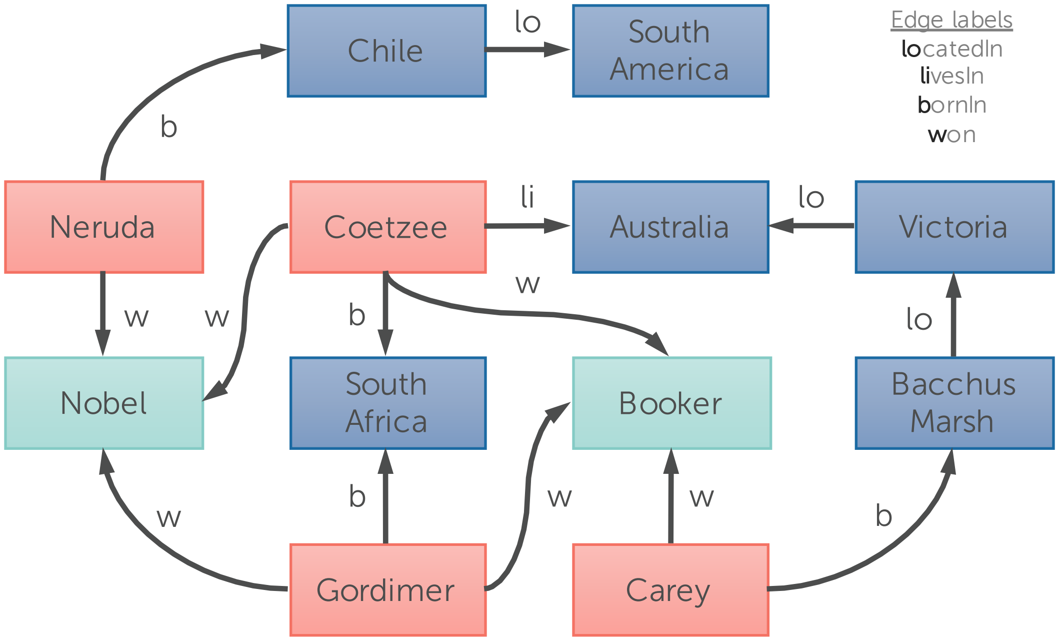
- All authors born in South Africa who have won both the Nobel and Booker prizes

$$ans(x) \leftarrow \begin{array}{c}(x, \text{hasWon}, \text{Nobel}),\\ (x, \text{hasWon}, \text{Booker}),\\ (x, \text{bornIn}, \text{South Africa})\end{array}$$

- Visually:



- Result? Coetzee Gordimer



Conjunctive Queries (Std. Matching)

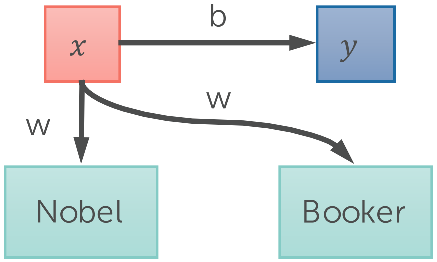
[Peter T. Wood. Query Languages for Graph Database. SIGMOD Record 41(1), 50–60, March 2012]

Example

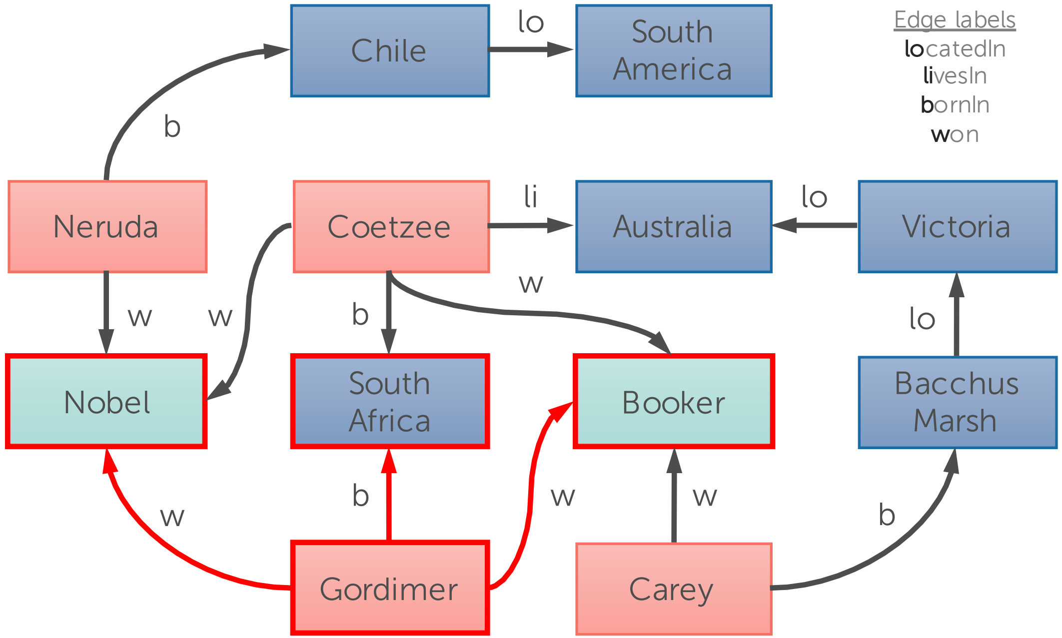
- All authors born in South Africa who have won both the Nobel and Booker prizes

$$ans(x) \leftarrow \begin{array}{c}(x, \text{hasWon}, \text{Nobel}),\\ (x, \text{hasWon}, \text{Booker}),\\ (x, \text{bornIn}, \text{South Africa})\end{array}$$

- Visually:



- Result? Coetzee Gordimer



Edge labels
locatedIn
livesIn
bornIn
won

Conjunctive Queries (Std. Matching)

Dresden Database Systems Group

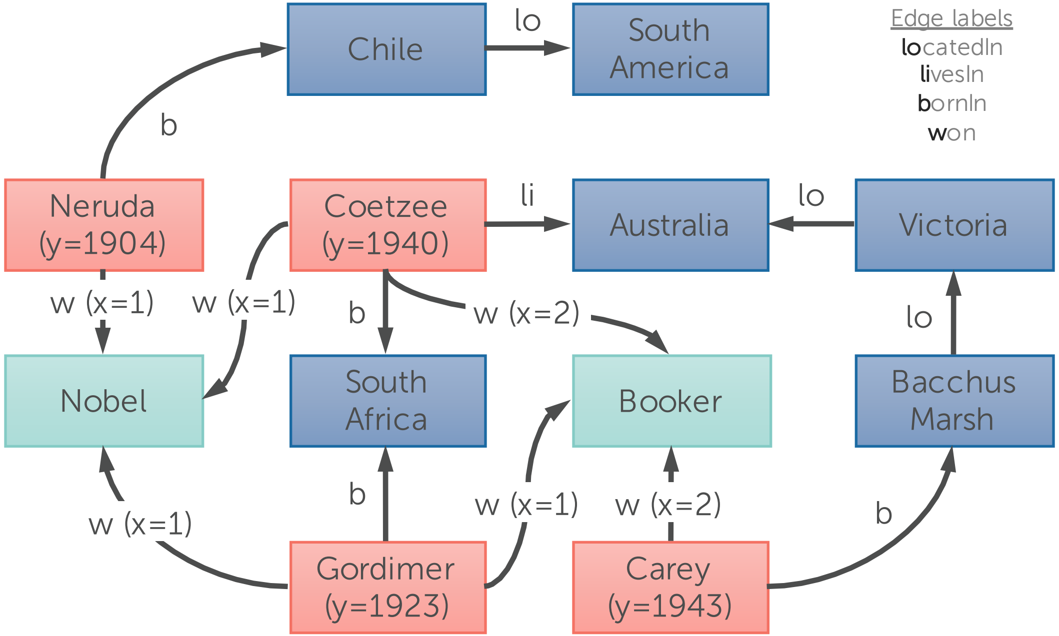
Extension to property graphs

- Add predicates on properties
- For vertex properties
  - All authors born in South Africa before 1930 who have won Nobel and Booker prizes

$$ans(x) \leftarrow \begin{array}{c}(x, \text{hasWon}, \text{Nobel}),\\ (x, \text{hasWon}, \text{Booker}),\\ (x, \text{bornIn}, \text{South Africa}),\\ \color{red}{(x, \text{year}, < 1930)}\end{array}$$

  - Extra syntax required for non-equi predicates
- For edge properties
  - Extra syntax required on existing edge predicates
  - All authors born in South Africa who have won Nobel once and Booker twice

$$ans(x) \leftarrow \begin{array}{c}(x, \text{hasWon}{:}\ \color{red}{(\mathrm{x}=1)}, \text{Nobel}),\\ (x, \text{hasWon}{:}\ \color{red}{(\mathrm{x}=2)}, \text{Booker}),\\ (x, \text{bornIn}, \text{South Africa}),\end{array}$$



Edge labels
locatedIn
livesIn
bornIn
won

Regular Path Query (Reachability)

Dresden Database Systems Group

[Peter T. Wood. Query Languages for Graph Database. SIGMOD Record 41(1), 50–60, March 2012]

Idea

- Query is given as a path predicate consisting of a pair vertex variables and a path expression
- Path expression is a regular expression of edge labels
- A pair of variable bindings is a valid answer iff the respective vertices are connect in hold on the data graph by a path conforming to the path expression

Definition

- Query Q is an expression $ans(x, y) \leftarrow (x, r, y)$
- $x \in V$ and $y \in V$ are vertex variables
- $r \in \Sigma^*$ is a regular expression over alphabet of edge labels Σ

Semantics

- A path p between v_0 and v_m in G is a sequence $v_0 a_0 v_1 a_1 v_2 \dots v_{m-1} a_{m-1} v_m$, with $v_i \in V$, $a_i \in \Sigma$, and $(v_i, a_i, v_{i+1}) \in E$
- Let $\lambda(p) \in \Sigma^*$ be the label of the path p, with $\lambda(p) = a_0 a_1 \dots a_{m-1}$
- Let $L(r)$ be the language denoted by the regular expression r, i.e. set all of all possible path labels denoted by r
- Path p satisfies r if $\lambda(p) \in L(r)$, i.e. the path's label satisfies the regular expression
- Then the query result $Q(G)$ is the set of all pairs of nodes (x, y) in G such there is a path from x to y which satisfies r

Regular Path Query (Reachability)

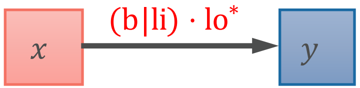
[Peter T. Wood. Query Languages for Graph Database. SIGMOD Record 41(1), 50–60, March 2012]

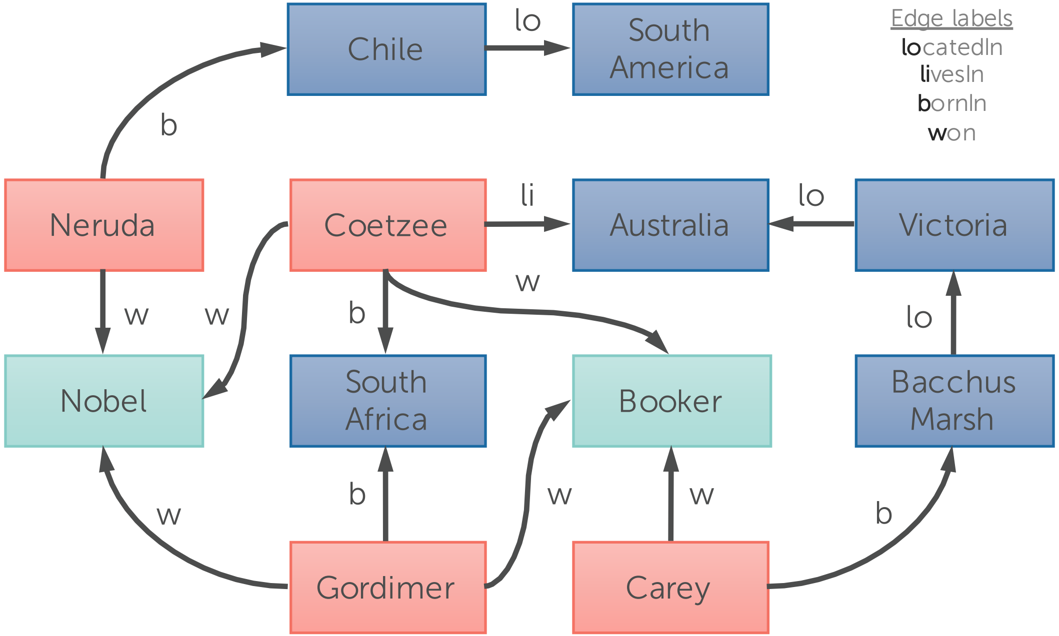
Example

- All authors and where they live in or are born

$$ans(x, y) \leftarrow (x, (\mathrm{b}|\mathrm{li}) \cdot \mathrm{lo}^{*}, y)$$

- Visually:





- Result?

Neruda	South America
Coetzee	Australia
Coetzee	South Africa
Gordimer	South Africa
Carey	Australia

Regular Path Query (Reachability)

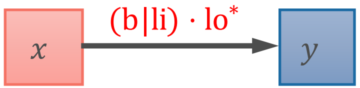
[Peter T. Wood. Query Languages for Graph Database. SIGMOD Record 41(1), 50–60, March 2012]

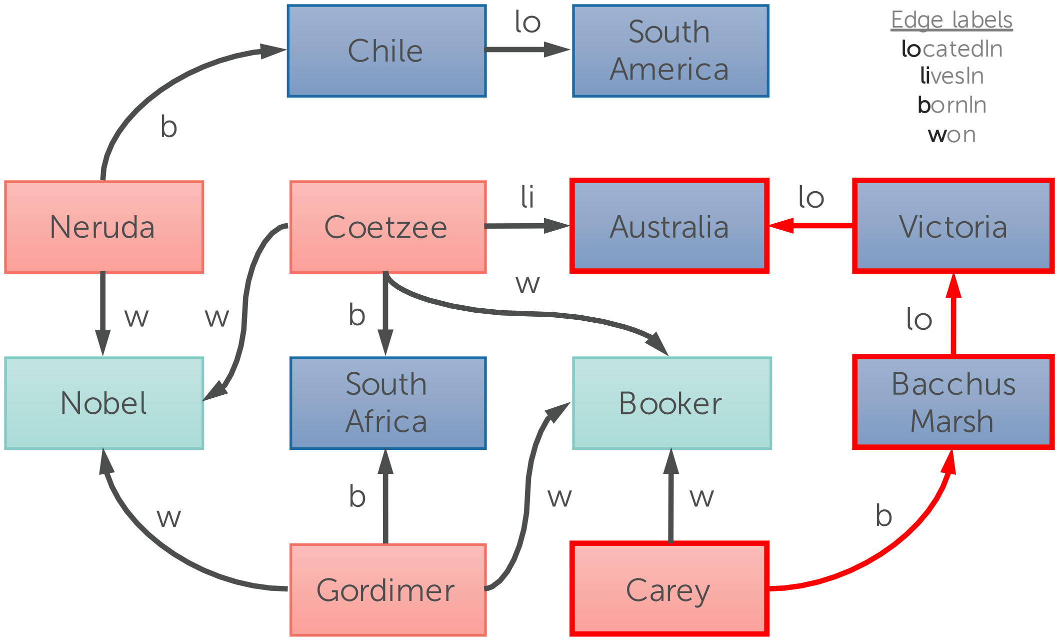
Example

- All authors and where they live in or are born

  $ans(x,y) \leftarrow (x, (\mathrm{b}|\mathrm{li}) \cdot \mathrm{lo}^*, y)$

- Visually:





- Result?

Neruda	South America
Coetzee	Australia
Coetzee	South Africa
Gordimer	South Africa
Carey	**Australia**

Regular Path Query (Reachability)

Dresden Database Systems Group

Extension to vertex label

- Path expressions include vertex labels, e.g., $(\mathrm{bornIn}|\mathrm{livesIn}) \cdot \mathrm{City} \cdot \mathrm{locatedIn}^* \cdot \mathrm{Continent}$
- Let L_V and L_E be the set of vertex and edge labels and λ the labeling function
- A regular expression r is over alphabet of edge and vertex labels pairs $(L_E \times L_V)^*$
- Let the label of a path be $\lambda(p) = a_0 \lambda(v_1) a_1 \dots a_{m-1} \lambda(v_m)$ with $a_i \in L_E$ and $\lambda(v_i) \in L_V$ and $\lambda(p) \in (L_E \times L_V)^*$
- As before: Path p satisfies r if $\lambda(p) \in L(r)$, i.e. the path's label satisfies the regular expression

Extension to properties

- Path expression include property predicates,
  e.g., $\mathrm{livesIn}\colon [\mathrm{since} < 1990] \cdot \mathrm{City}\colon [\mathrm{population} > 10\mathrm{Mio}] \cdot \mathrm{locatedIn}^+ \cdot \mathrm{Continent}$
- Path expressions quickly become hard to read, cf. XPath and XQuery
- In Datalog rules:

$$\begin{aligned} ans(x,y) &\leftarrow livesIn(x,z), eSince(x,z,s), s < 1990, \\ &\qquad city(z), vPopulation(z,p), p > 10\mathrm{Mio}, loStar(z,y), continent(y) \\ loStar(x,y) &\leftarrow locatedIn(x,y) \\ loStar(x,y) &\leftarrow locatedIn(x,z), loStar(z,y) \end{aligned}$$

Conjunctive Regular Path Queries (CRPQs)

[Peter T. Wood. Query Languages for Graph Database. SIGMOD Record 41(1), 50–60, March 2012]

Idea

- Query is given as a set of path predicates
- Each path predicate consists of a pair vertex variables and a regular expression of edge labels
- A set of variable bindings is a valid answer iff all path predicates hold on the data graph

Definition

- Query Q is an expression

$$ans(z_1, \dots, z_n) \leftarrow \bigwedge_{1 \le i \le m} (x_i, r_i, y_i)$$

- Each $x_i \in X$ and $y_i \in Y$ is a vertex variable or a constant from V
- Each $r_i \in \Sigma^*$ is a regular expression over alphabet of edge labels Σ
- Each z_i is some x_i or y_i

Semantics

- Let $\sigma: X \cup Y \rightarrow V$ be a specific selection of variable bindings, i.e., a mapping to vertices of G
- Say relation $(G, \sigma) \vDash Q$ holds iff, for $1 \le i \le m$ there exists a path p_i in G from $\sigma(x_i)$ to $\sigma(y_i)$ such that $\lambda(p) \in L(r)$
- Then the query result $Q(G)$ is the set of tuples $\left(\sigma(z_1), \dots, \sigma(z_n)\right)$ such that $(G, \sigma) \vDash Q$

Conjunctive Regular Path Queries (CRPQs)

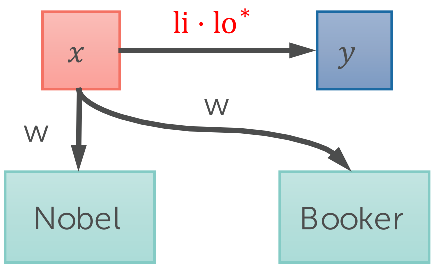
[Peter T. Wood. Query Languages for Graph Database. SIGMOD Record 41(1), 50–60, March 2012]

Example

- All winners of Nobel and Booker and where they live

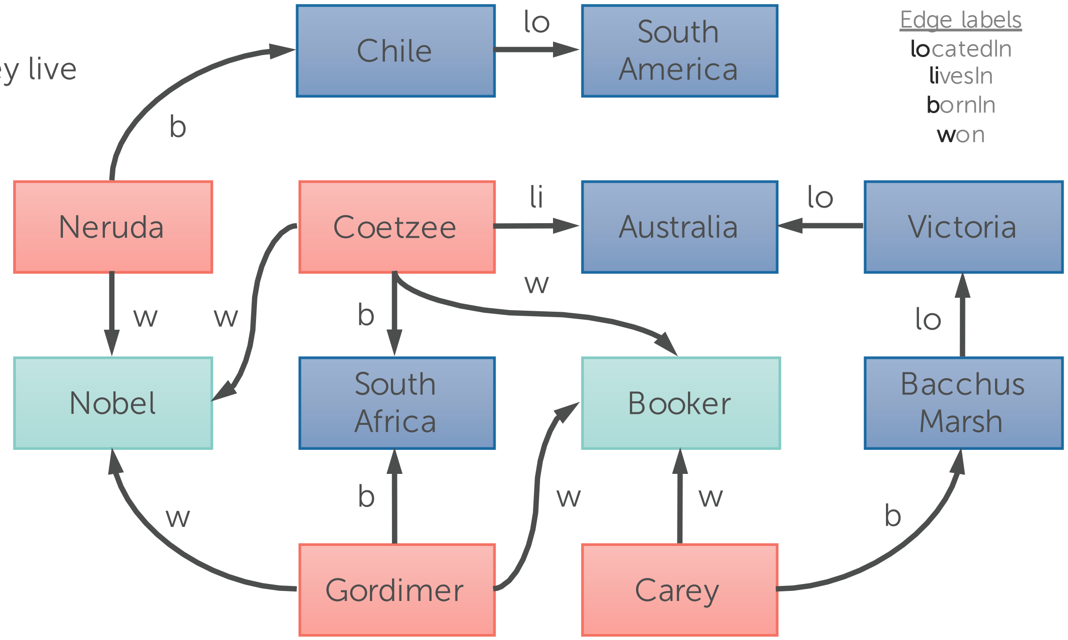
$$ans(x, y) \leftarrow \begin{array}{l}(x, \text{hasWon}, \text{Nobel}),\\ (x, \text{hasWon}, \text{Booker}),\\ (x, \text{li} \cdot \text{lo}^*, y)\end{array}$$

- Visually:



- Result?

Coetzee	Australia



Conjunctive Regular Path Queries (CRPQs)

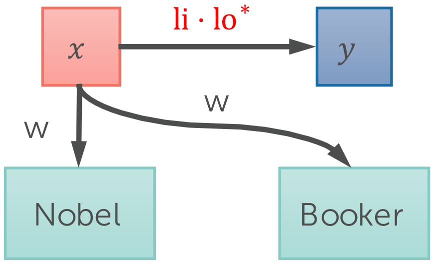
[Peter T. Wood. Query Languages for Graph Database. SIGMOD Record 41(1), 50–60, March 2012]

Example

- All winners of Nobel and Booker and where they live

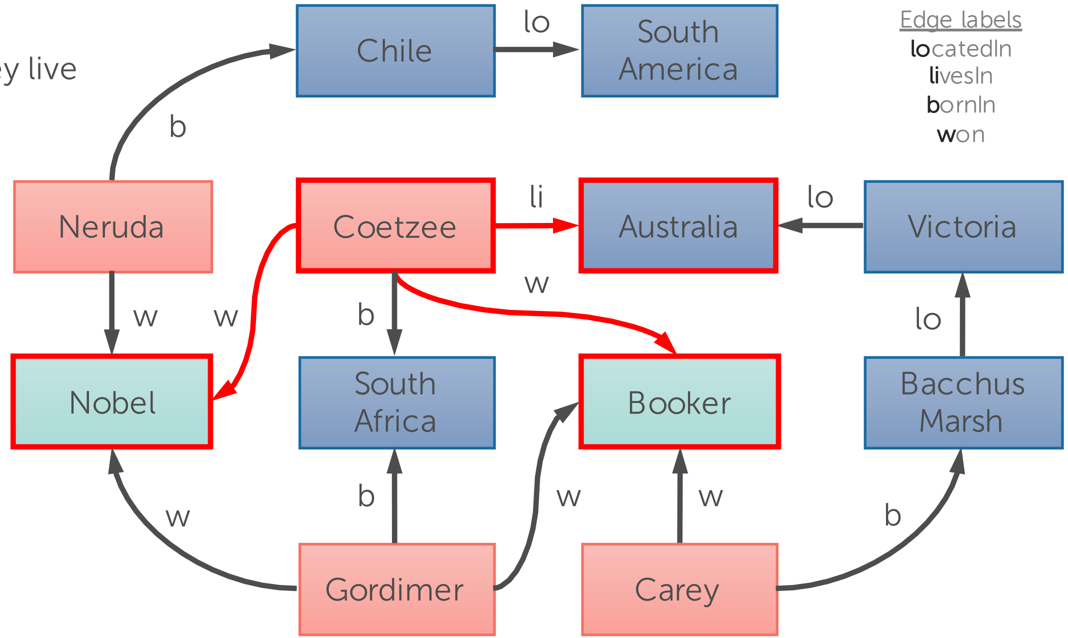
$$ans(x, y) \leftarrow \begin{array}{l}(x, \text{hasWon}, \text{Nobel}),\\ (x, \text{hasWon}, \text{Booker}),\\ (x, \text{li} \cdot \text{lo}^*, y)\end{array}$$

- Visually:



- Result?

Coetzee	Australia



Edge labels
locatedIn
livesIn
bornIn
won

Further Query Types

Dresden Database Systems Group

Union conjunctive Queries (UQs)

- Adds disjunction
- Example: $ans(x) \leftarrow (x, \mathrm{hasWon}, \mathrm{Booker}) \vee (x, \mathrm{hasWon}, \mathrm{Nobel})$ which give all price winners
- Multiple conjunctive queries with intersecting variable sets
- Result is the union of the result each conjunctive query projected to the intersection of all variable sets

Two-way regular path queries (2RPQs)

- Allows to express backward traversal of edge types
- Example: $ans(x, y) \leftarrow (x, \mathrm{hasWon} \cdot -\mathrm{hasWon}, y)$ which gives all author pairs where both have won the same price

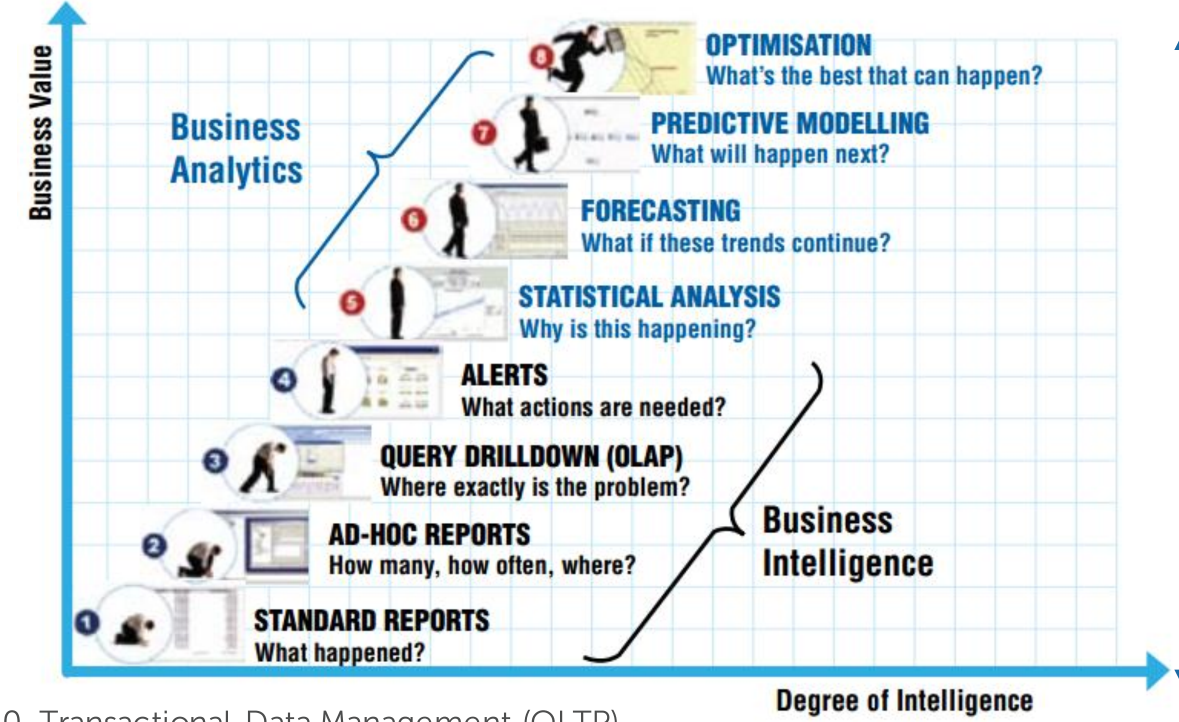
Combination of all: Union conjunctive two-way regular path queries (UC2RPQs)

- Class of Queries that can be expressed with SPARQL 1.1 and Neo4j Cypher (differences in the exact semantics)
- Can also be expressed in the relational world with Datalog or SQL incl. recursive common table expressions

Level of Analytics

Dresden Database Systems Group

[http://www.rosebt.com/blog/eight-levels-of-analytics-business-intelligence-to-business-analytics]



0. Transactional Data Management (OLTP)

batch
long running
imperative

Graph Data Management Applications

ad-hoc
short running
declarative

Aggregations & composability needed

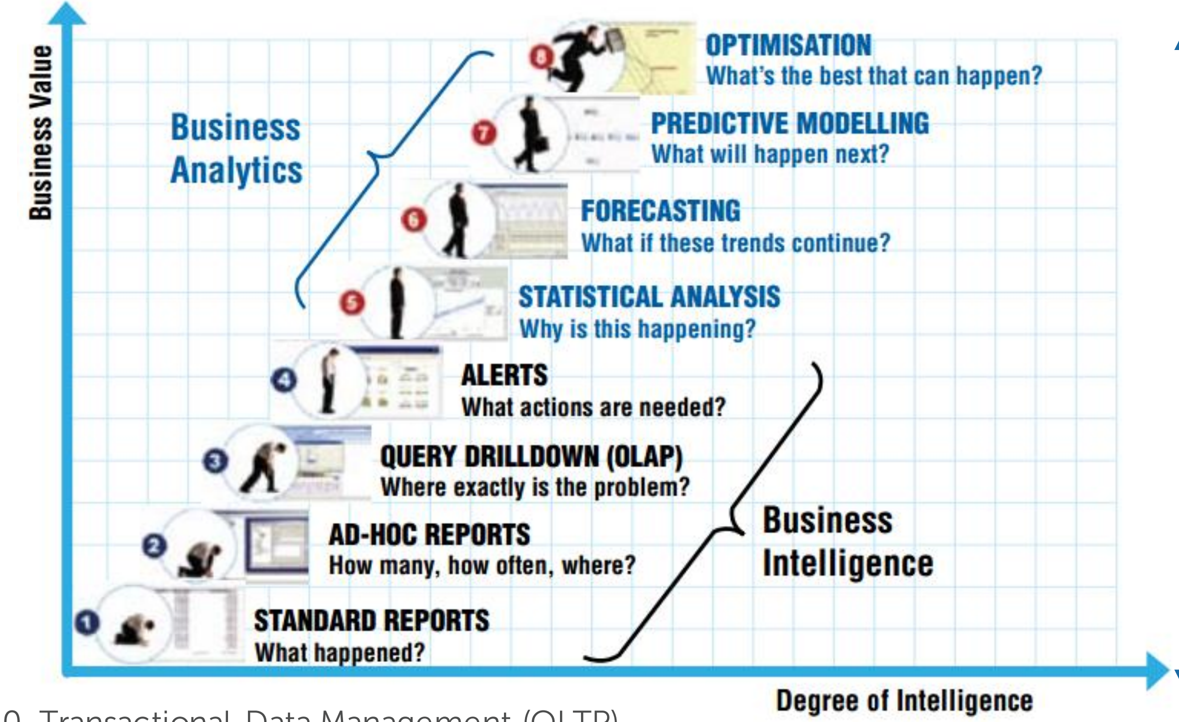
Graph Matching

Break

Level of Analytics

Dresden Database Systems Group

[http://www.rosebt.com/blog/eight-levels-of-analytics-business-intelligence-to-business-analytics]



0. Transactional Data Management (OLTP)

batch
long running
imperative

Graph Data Management Applications

ad-hoc
short running
declarative

Aggregations & composability needed

Graph Matching

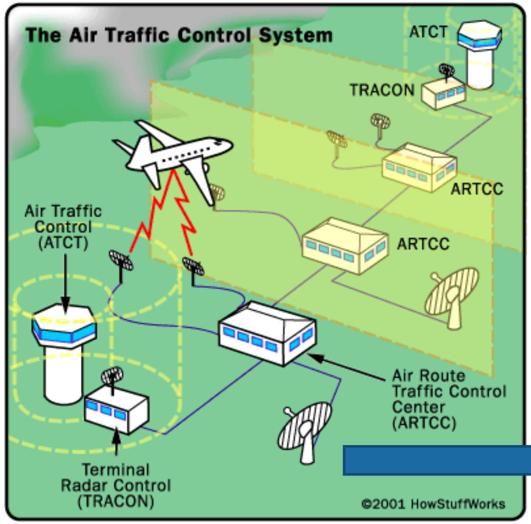
Example: Air Traffic Surveillance

Captured of surveillance data

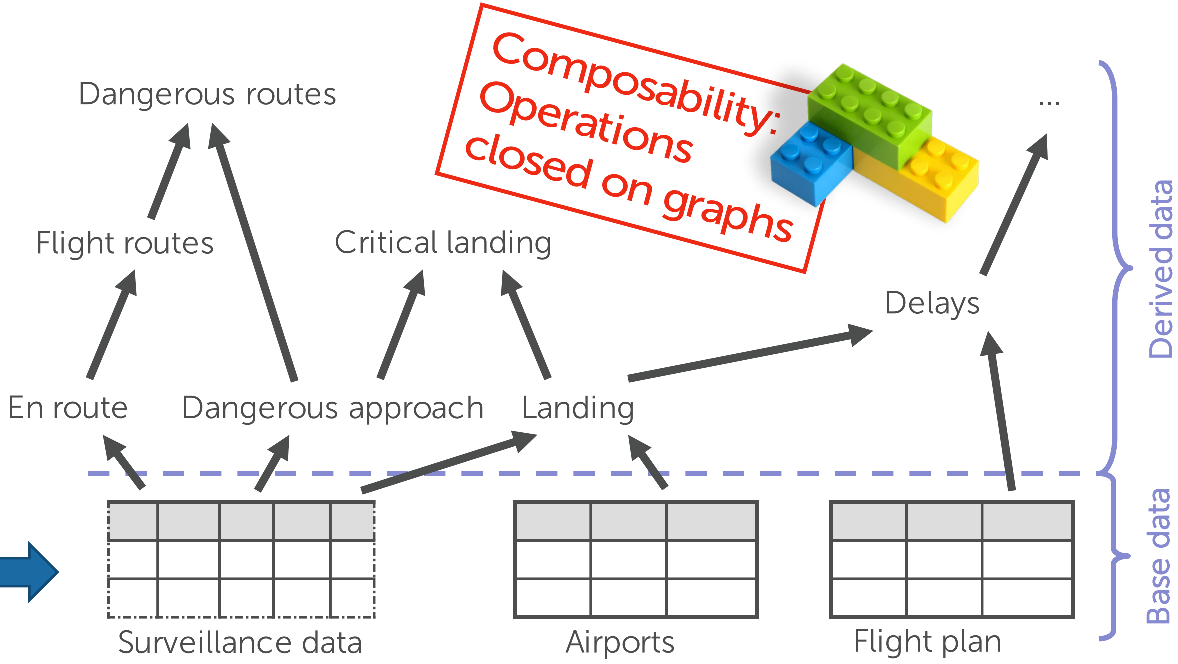
- Fine granularity data
- Low abstraction
- Geo position, timestamp, plane id

Information of interest

- Stepwise abstraction from base data to aggregated information



[http://science.howstuffworks.com/transport/flight/modern/air-traffic-control2.htm]



Composability

Graph Transformation Rules

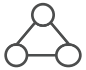
MATCH -> VARIABLE BINDINGS -> PRODUCTION

Match Pattern

Variable Bindings

x_1	…	x_n

Production Pattern



DIFFERENT TYPES OF VARIABLES

- Value variable
- Vertex variable
- (Edge variables)
- (Path variables)
- ((Sub)Graph variables)

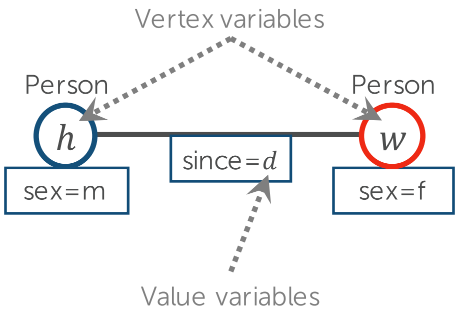
PRODUCTION

- Existing vertices from bound vertex variables
- New vertices with new unbound vertex variables
- Edges either implicit (via vertex variable) or explicit (with edge variables)
- Existing values from bound value variable

Graph Transformation Rule

Example

$$\overbrace{\begin{pmatrix} h:\text{Person}(\text{sex}=\text{m}), \\ w:\text{Person}(\text{sex}=\text{f}), \\ h-w:\text{married}(\text{since}=d) \end{pmatrix}}^{\text{Match Pattern}} \rightarrow \overbrace{\begin{pmatrix} h, w, \\ m:\text{Marriage}(\text{date}=d), \\ m \rightarrow h:\text{husband}, m \rightarrow w:\text{wife} \end{pmatrix}}^{\text{Production Pattern}}$$



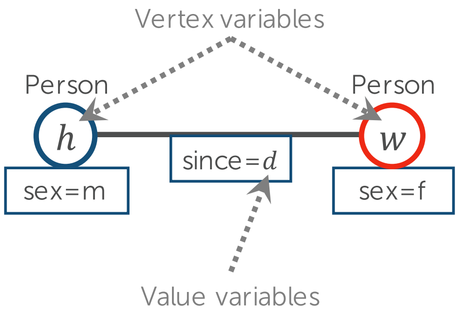
Graph Transformation Rule

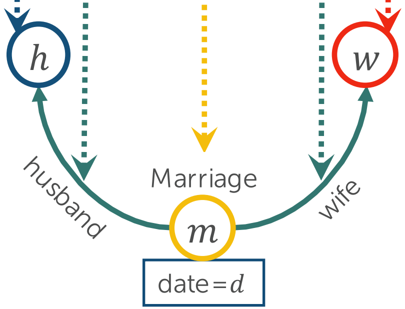
EXAMPLE

Match Pattern

Production Pattern

$$\begin{pmatrix} h\colon \mathrm{Person}(\mathrm{sex} = \mathrm{m}), \\ w\colon \mathrm{Person}(\mathrm{sex} = \mathrm{f}), \\ h - w\colon \mathrm{married}(\mathrm{since} = d) \end{pmatrix} \rightarrow \begin{pmatrix} h, w, \\ m\colon \mathrm{Marriage}(\mathrm{date} = d), \\ m \rightarrow h\colon \mathrm{husband}, m \rightarrow w\colon \mathrm{wife} \end{pmatrix}$$





Unbound vertex variable m produces a new vertex for every match

Unbound vertex pairs $m \rightarrow h$ and $m \rightarrow w$ produce new edges for every match

Result Presentation

Rule (without transformation)

- e.g., Pairs of friends: 

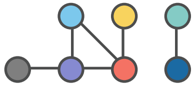
Isolated matches

- Each match separately independently of vertex identity

- Vertices taking part in multiple matches have to duplicated
- Good for querying paths, further combining individual matches and result iteration

Merged matches

- All matches form a (partitioned) graph based



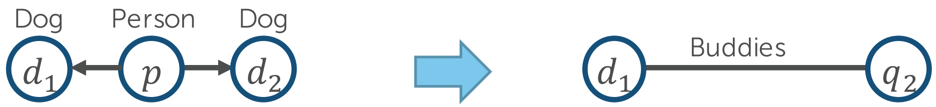
- No vertex duplication necessary
- Keeps topology of source graph

Merged Transformation Results

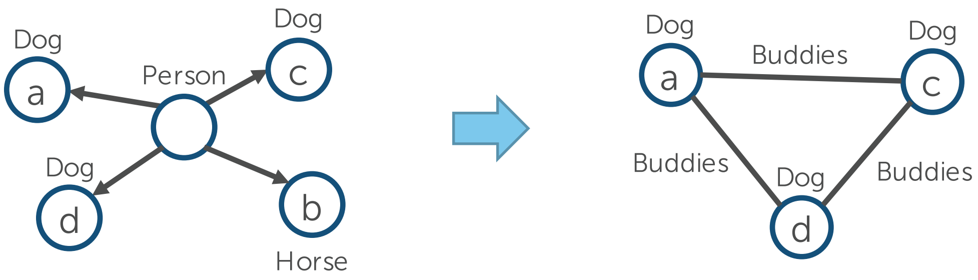
Example

- Rule

$$((p\colon \mathrm{Person}, d_1\colon \mathrm{Dog}, d_2\colon \mathrm{Dog}, p \to d_1, p \to d_2) \to (d_1, d_2, d_1 - d_2\colon \mathrm{Buddies}))$$



- Data



Side Note on Syntax

Dresden Database Systems Group

YOU DO NOT LIKE THE SYNTAX?

- How about

```
SELECT NODE d1, NODE d2, UNDIRECTED EDGE d1 TO d2(Buddies)
FROM NODE p(Person), NODE d1(Dog), NODE d2(Dog), EDGE p TO d1, EDGE p TO d2
```

- Or

```
SELECT d1, d2, d1--d2:Buddies FROM p:Person, d1:Dog, d2:Dog, p->d1, p->d2
```

- Or

```
SELECT (d1)-[:Buddies]-(d2) FROM (d1:Dog)<--(:Person)-->(d2:Dog)
```

- Or

```
(V(p,Person),V(d1,Dog),V(d2,Dog),E(p,>,d1),E(p,>,d2)) -> (V(d1),V(d2),E(d1,-,d2)).
```

- There is some syntactical freedom as along as you stick with the language principles

Composability

- Queries with a name (aka views) can be refer to in other queries

Unnamed Query using U and V
„singles liking chocolate but no TV"

$$:: \left((V :: (s\colon \text{Person}), U :: (\text{NOT } s\colon \text{Person}), \text{NOT } s \rightarrow t\colon \text{likes}, t\colon \text{TV}) \rightarrow (s)\right)$$

Query named V
„persons liking chocolate"

$$V :: \left((x\colon \text{Person}, c\colon \text{Chocolate}, x \rightarrow c\colon \text{like}) \rightarrow (x, c, x - c)\right)$$

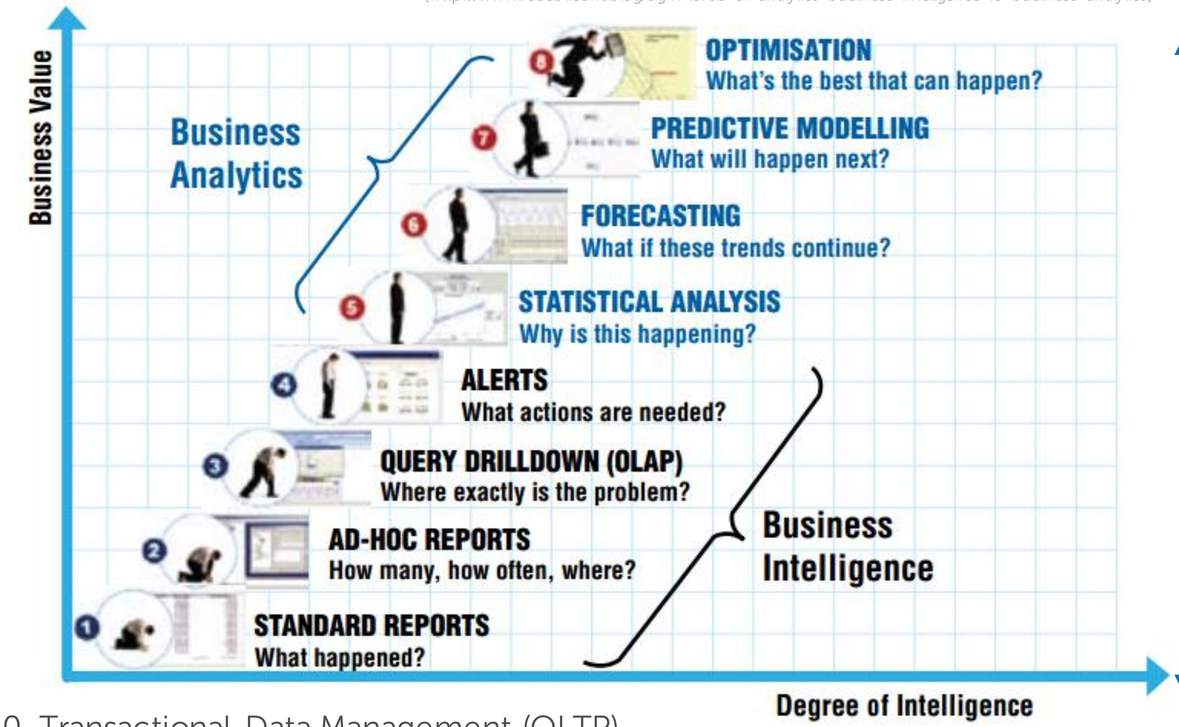
$$U :: \left((a\colon \text{Person}, b\colon \text{Person}, a - b\colon \text{partner}) \rightarrow (a, b, a - b)\right)$$

Query named U
„persons in a partnership"

Level of Analytics

Dresden Database Systems Group

[http://www.rosebt.com/blog/eight-levels-of-analytics-business-intelligence-to-business-analytics]



0. Transactional Data Management (OLTP)

batch
long running
imperative

Graph Data Management Applications

ad-hoc
short running
declarative

Aggregations needed

Composable Graph Matching

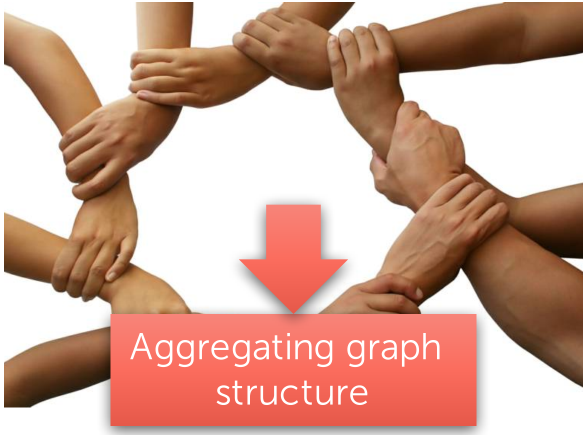
Aggregating Graphs

Properties of entities

- Captured/measured values
- What are the sales figures/temperatures/etc.?
- Multidimensional data/time series/matrixes

Connections between entities

- Network structure
- What do the friends of your customers buy?
- Graph data



Dresden Database Systems Group

Aggregating Graph Data

TECHNISCHE UNIVERSITÄT DRESDEN

Aggregation in Graph Transformation

GERNALE

- Per (vertex) production predicate
- Edges inherit grouping if they connect one or two grouping vertices
- Edges can have own grouping attributes, that are added to the inherit grouping attributes
- Based on match variables
- All match variables not used in grouping can be used in the same predicate only in an aggregation function

SYNTAX

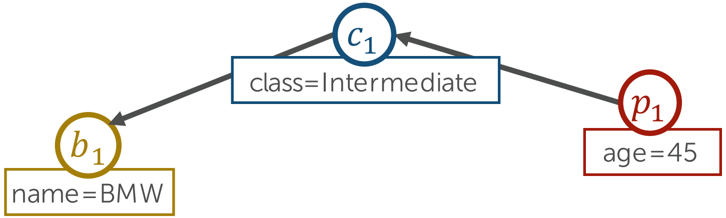
- <productionGroupPredicate> ::= <productionGroupVertexVariable> "@" {<variable> ","} <labels>? <attributes>?
- <variable> ::= <matchVertexVariable> | <valueVariable>

EXAMPLE

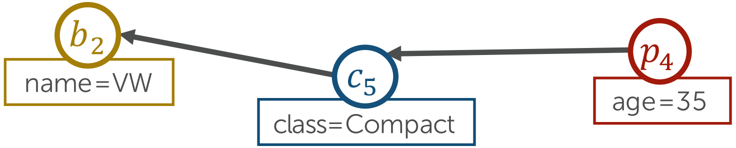
- $\begin{pmatrix} p\colon \mathrm{Person}(\mathrm{age} = a\ ), \\ b\colon \mathrm{Profession}, p \to b\colon \mathrm{works{-}in} \end{pmatrix} \to \begin{pmatrix} g_a @ a(\mathrm{name} = a), \\ g_b @ b(\mathrm{name} = b.\mathrm{name}), \\ g_a - g_b(\mathrm{number} = \mathrm{CNT}(p)) \end{pmatrix}$

Aggregation

$$\begin{pmatrix} p\colon \mathrm{Person}(\mathrm{age} = a), \\ c\colon \mathrm{Car}(\mathrm{class} = k), \\ p \to c\colon \mathrm{drives}, \\ b\colon \mathrm{Brand}(), \\ c \to b\colon \mathrm{madeBy} \end{pmatrix} \to \begin{pmatrix} \quad \end{pmatrix}$$

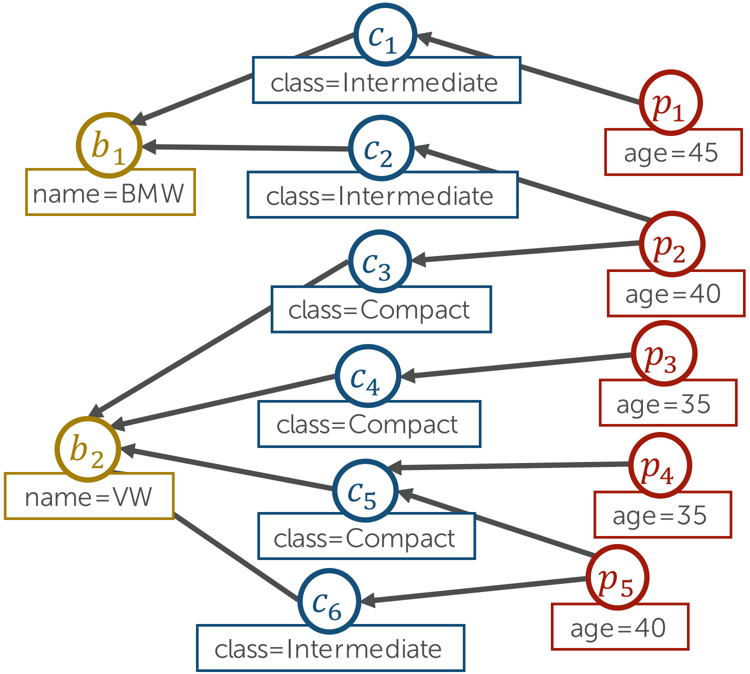


Matches



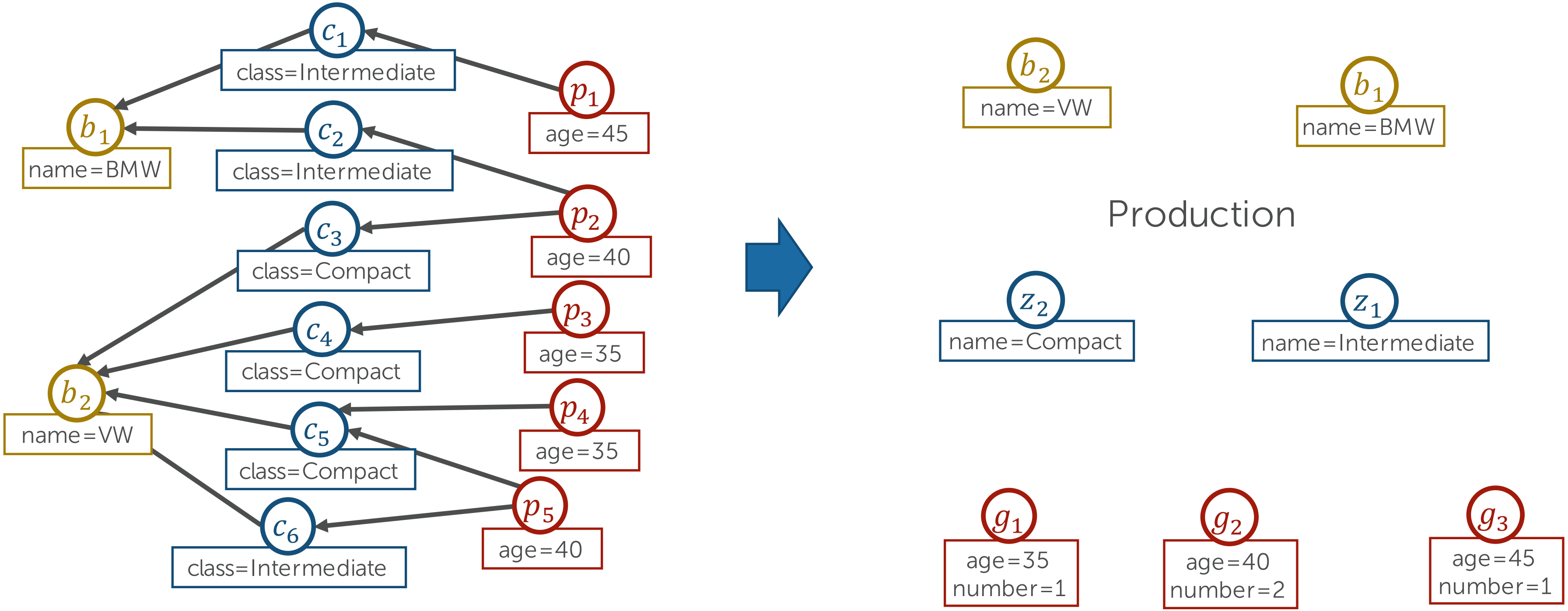
Aggregation

$$\begin{pmatrix} p\colon \mathrm{Person}(\mathrm{age} = a), \\ c\colon \mathrm{Car}(\mathrm{class} = k), \\ p \to c\colon \mathrm{drives}, \\ b\colon \mathrm{Brand}(), \\ c \to b\colon \mathrm{madeBy} \end{pmatrix} \to \begin{pmatrix} \quad \end{pmatrix}$$



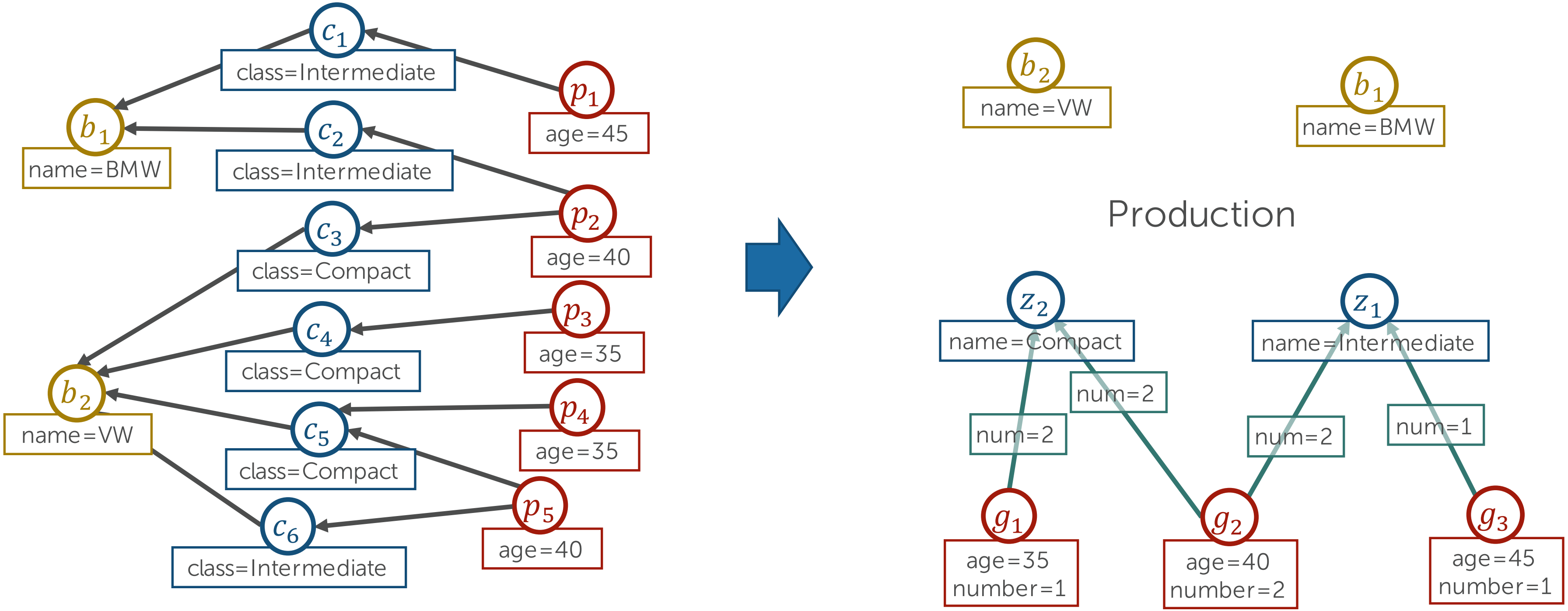
Aggregation

$$\begin{pmatrix} p:\mathrm{Person}(\mathrm{age}=a), \\ c:\mathrm{Car}(\mathrm{class}=k), \\ p \to c:\mathrm{drives}, \\ b:\mathrm{Brand}(), \\ c \to b:\mathrm{madeBy} \end{pmatrix} \to \begin{pmatrix} g@a:\mathrm{AgeGroup}(\mathrm{age}=a, \mathrm{number}=\mathrm{CNT}(p)), \\ z@k:\mathrm{Class}(\mathrm{name}=k), b \\ \\ \\ \\ \end{pmatrix}$$



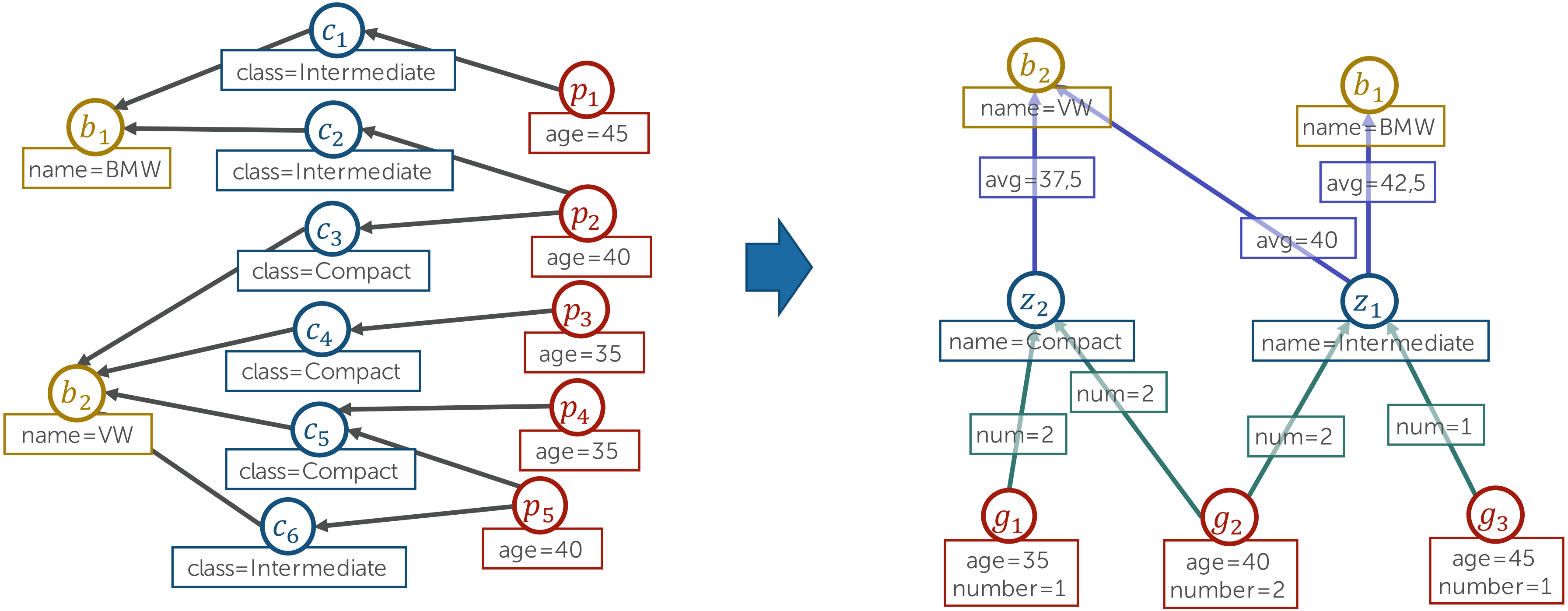
Aggregation

$$\begin{pmatrix} p\colon \mathrm{Person}(\mathrm{age} = a), \\ c\colon \mathrm{Car}(\mathrm{class} = k), \\ p \to c\colon \mathrm{drives}, \\ b\colon \mathrm{Brand}(), \\ c \to b\colon \mathrm{madeBy} \end{pmatrix} \to \begin{pmatrix} g@a\colon \mathrm{AgeGroup}(\mathrm{age} = a, \mathrm{number} = \mathrm{CNT}(p)), \\ z@k\colon \mathrm{Class}(\mathrm{name} = k), b \\ g \to z(\mathrm{numdrivers} = \mathrm{CNT}(p)), \\ \\ \\ \end{pmatrix}$$



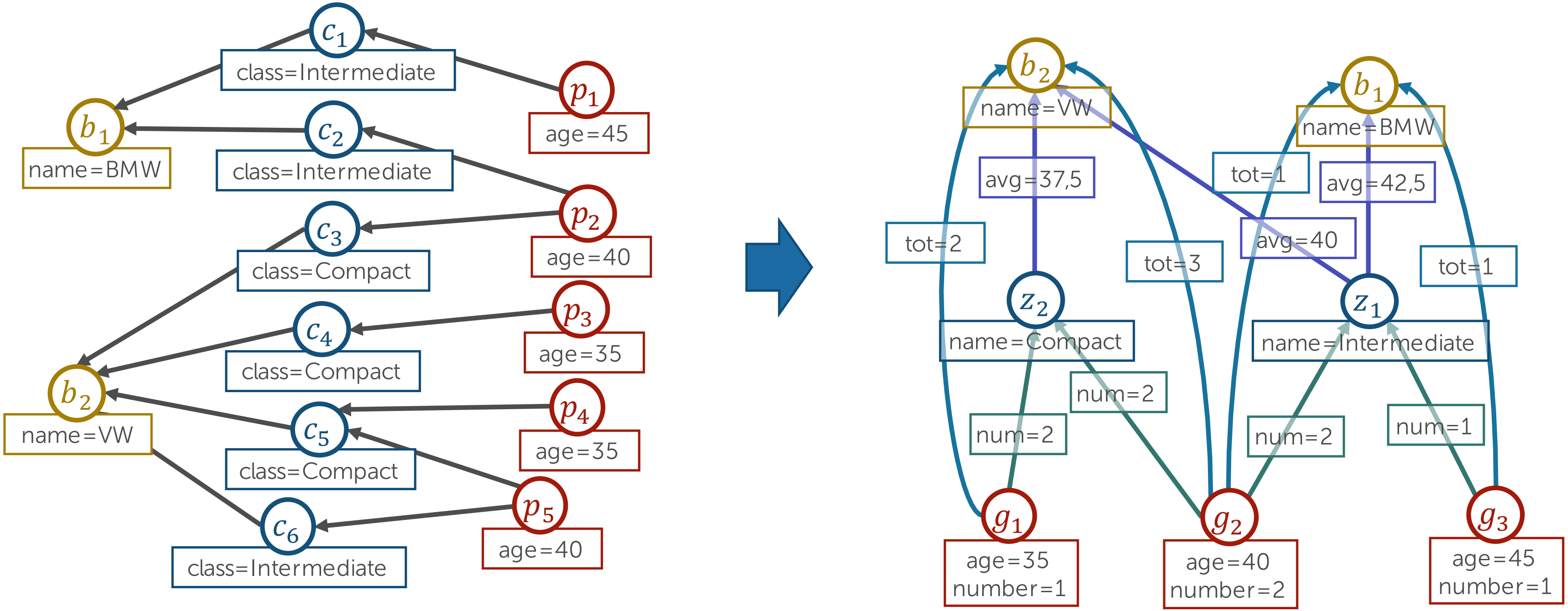
Aggregation

$$\begin{pmatrix} p\!:\mathrm{Person}(\mathrm{age} = a), \\ c\!:\mathrm{Car}(\mathrm{class} = k), \\ p \to c\!:\mathrm{drives}, \\ b\!:\mathrm{Brand}(), \\ c \to b\!:\mathrm{madeBy} \end{pmatrix} \to \begin{pmatrix} g@a\!:\mathrm{AgeGroup}(\mathrm{age} = a, \mathrm{number} = \mathrm{CNT}(p)), \\ z@k\!:\mathrm{Class}(\mathrm{name} = k), b \\ g \to z(\mathrm{numdrivers} = \mathrm{CNT}(p)), \\ z \to b(\mathrm{avgage} = \mathrm{AVG}(p.\mathrm{age})), \end{pmatrix}$$



Aggregation

$$\begin{pmatrix} p\colon \mathrm{Person}(\mathrm{age}=a), \\ c\colon \mathrm{Car}(\mathrm{class}=k), \\ p \to c\colon \mathrm{drives}, \\ b\colon \mathrm{Brand}(), \\ c \to b\colon \mathrm{madeBy} \end{pmatrix} \to \begin{pmatrix} g@a\colon \mathrm{AgeGroup}(\mathrm{age}=a, \mathrm{number}=\mathrm{CNT}(p)), \\ z@k\colon \mathrm{Class}(\mathrm{name}=k), b \\ g \to z(\mathrm{numdrivers}=\mathrm{CNT}(p)), \\ z \to b(\mathrm{avgage}=\mathrm{AVG}(p.\mathrm{age})), \\ g \to b(\mathrm{totalcars}=\mathrm{CNT}(c)) \end{pmatrix}$$



On Tables

$$\begin{pmatrix} p\colon \mathrm{Person}(\mathrm{age} = a), \\ c\colon \mathrm{Car}(\mathrm{class} = k), \\ p \to c\colon \mathrm{drives}, \\ b\colon \mathrm{Brand}(), \\ c \to b\colon \mathrm{madeBy} \end{pmatrix} \to \left(\qquad \right)$$

Dresden Database Systems Group

id	L	...
⋮	⋮	⋮

src	trg	...
⋮	⋮	⋮

Matching

p	a	c	k	b
1	45	1	I	1
2	40	2	I	1
2	40	3	C	2
3	35	4	C	2
4	35	5	C	2
5	40	5	C	2
5	40	6	I	2

Edge Tables

a	g	k	z	$\mathrm{CNT}(p)$
35	1	C	2	2
40	2	C	2	2
40	2	I	1	2
45	3	I	1	1

k	z	b	$\mathrm{AVG}(p.\mathrm{age})$
I	1	1	42,5
I	1	2	40,0
C	2	2	37,5

a	g	b	$\mathrm{CNT}(c)$
35	1	2	2
40	2	2	3
40	2	1	1
45	3	1	1

Vertex Tables

k	z
I	1
C	2

a	g	$\mathrm{CNT}(p)$
35	1	2
40	2	2
45	3	1

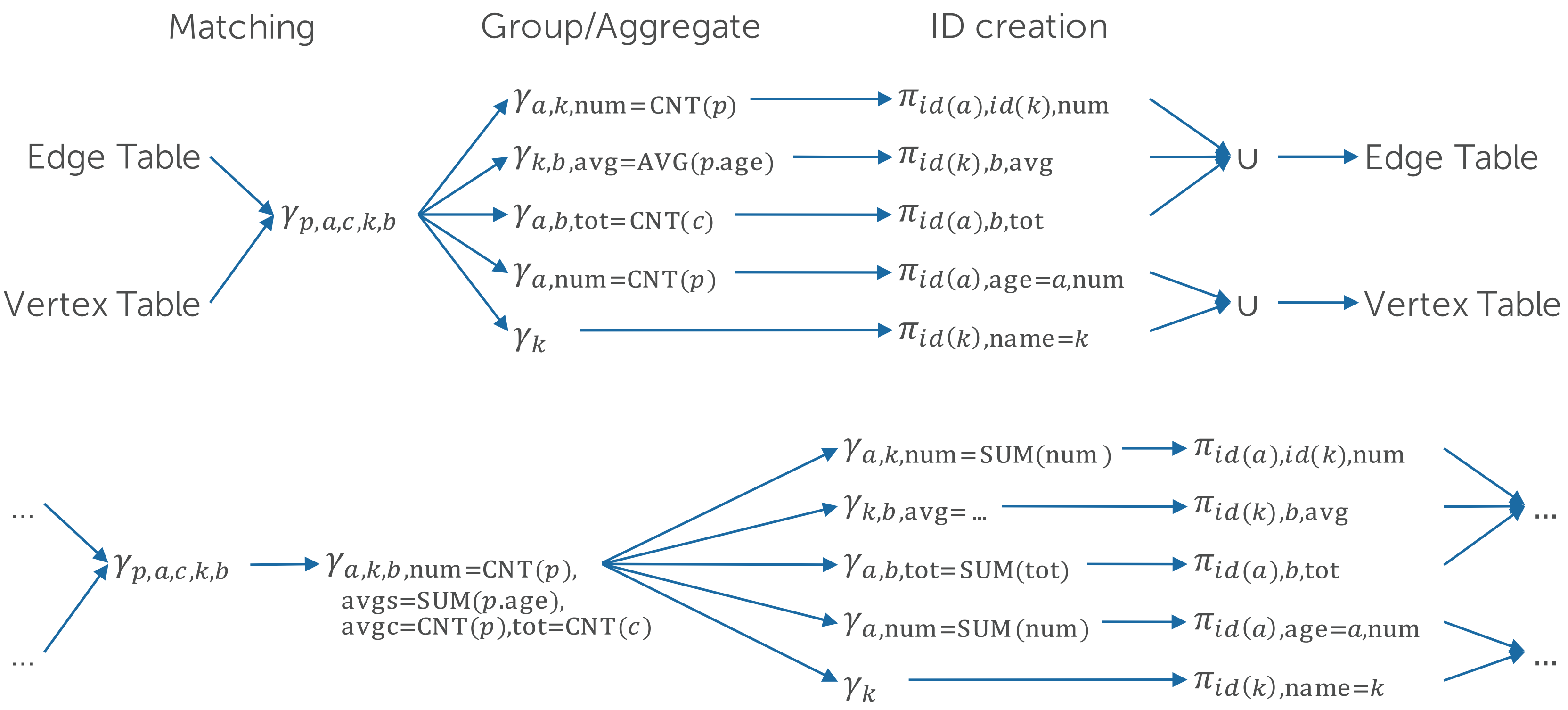
union

src	trg	num	avg	tot
g_1	z_2	2		
g_2	z_2	2		
g_2	z_1	2		
g_3	z_1	1		
z_1	b_1		42,5	
z_1	b_2		40,0	
z_2	b_2		37,5	
g_1	b_2			2
g_2	b_2			3
g_2	b_1			1
g_3	b_1			1

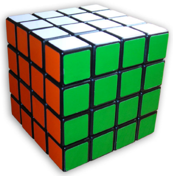
union

id	L	name	age	num
g_1	AgeGroup		35	2
g_2	AgeGroup		40	2
g_3	AgeGroup		45	1
z_1	Class			
z_2	Class			
b_1	Brand	BMW		
b_2	Brand	VW		

Grouping Lattice



Multidimensional Graph Aggregation



[https://commons.wikimedia.org/wiki/File:Rubiks_revenge_solved.jpg]

Fact

c: Car( ),

$\left(\quad \right) \rightarrow \left(\quad \right)$

c_1

c_2

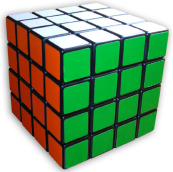
c_3

c_4

c_5

c_6

Multidimensional Graph Aggregation



[https://commons.wikimedia.org/wiki/File:Rubiks_revenge_solved.jpg]

Fact

Dimension *Class*

$$\left(c\colon \mathrm{Car}(\mathrm{class} = k), \right) \rightarrow \left(\right)$$

c_1 class=Intermediate

c_2 class=Intermediate

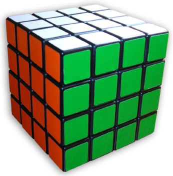
c_3 class=Compact

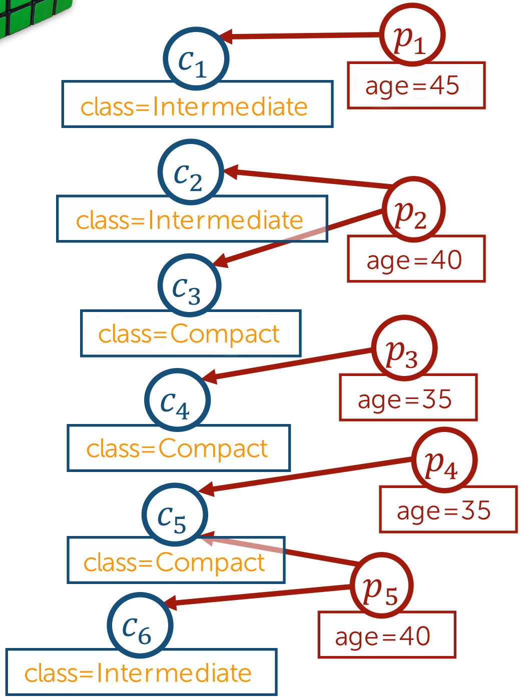
c_4 class=Compact

c_5 class=Compact

c_6 class=Intermediate

Multidimensional Graph Aggregation





Dimension *Class*

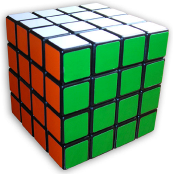
Fact

Dim. *Driver Age*

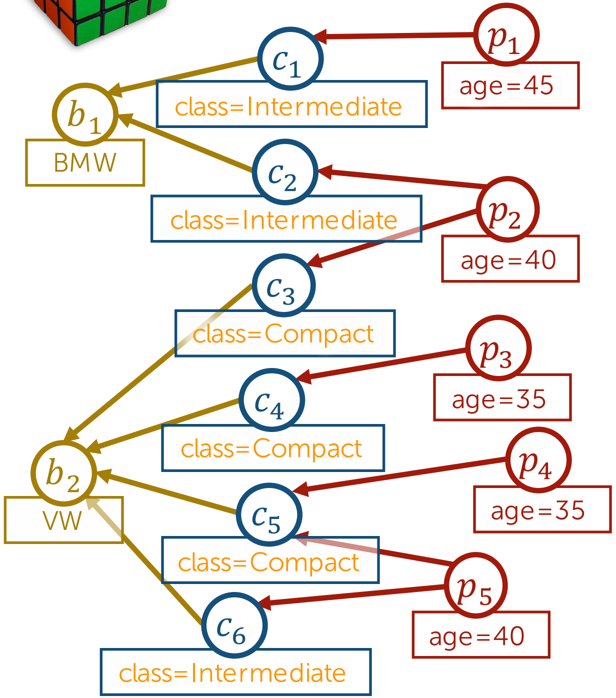
$$\begin{pmatrix} c\!:\mathrm{Car}(\mathrm{class} = k), \\ p\!:\mathrm{Person}(\mathrm{age} = a), p \rightarrow c, \\ \end{pmatrix} \rightarrow \begin{pmatrix} \quad \end{pmatrix}$$

Multidimensional Graph Aggregation

Dresden Database Systems Group



[https://commons.wikimedia.org/wiki/File:Rubiks_revenge_solved.jpg]



Fact

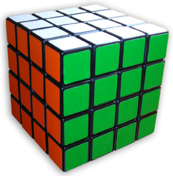
Dimension *Class*

Dim. *Driver Age*

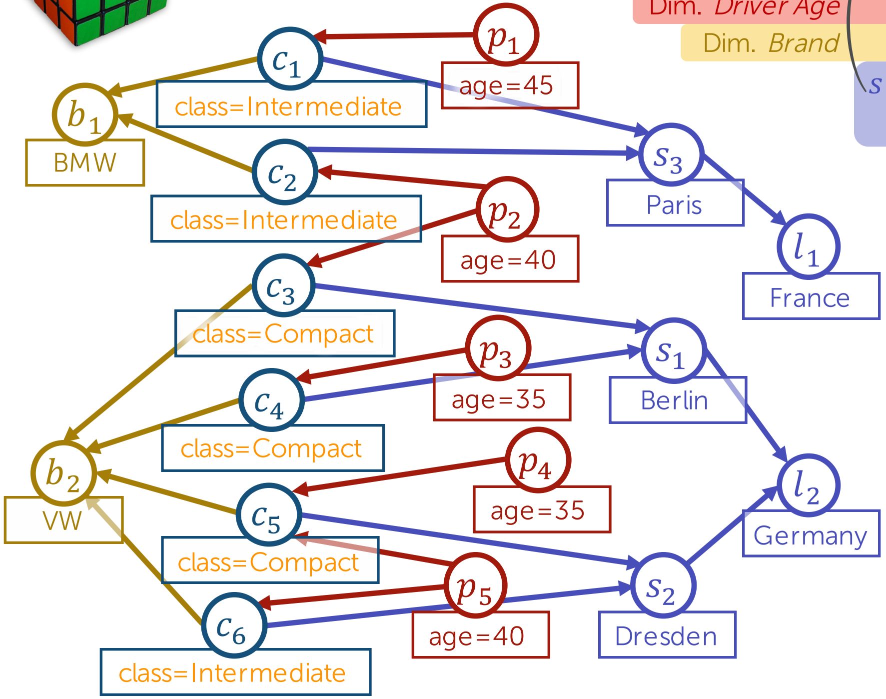
Dim. *Brand*

$$\begin{pmatrix} c\colon \mathrm{Car}(\mathrm{class} = k), \\ p\colon \mathrm{Person}(\mathrm{age} = a), p \to c, \\ b\colon \mathrm{Brand}, c \to b, \end{pmatrix} \to \begin{pmatrix} \quad \end{pmatrix}$$

Multidimensional Graph Aggregation



[https://commons.wikimedia.org/wiki/File:Rubiks_revenge_solved.jpg]



Dimension *Class*

Fact

Dim. *Driver Age*

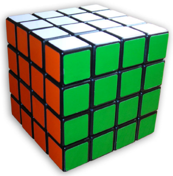
Dim. *Brand*

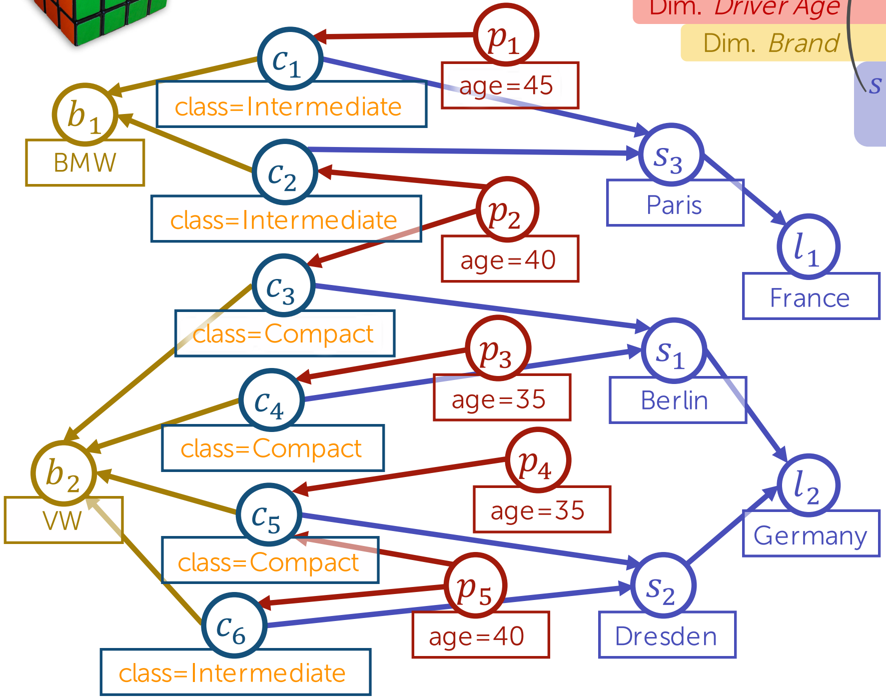
$$\left(\begin{array}{l} c\colon \mathrm{Car}(\mathrm{class} = k), \\ p\colon \mathrm{Person}(\mathrm{age} = a), p \to c, \\ b\colon \mathrm{Brand}, c \to b, \\ s\colon \mathrm{City}, c \to s, l\colon \mathrm{Country}, s \to l \end{array} \right) \to \left(\quad \right)$$

Dimension *Location*

Multidimensional Graph Aggregation

Dresden Database Systems Group



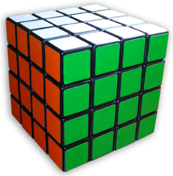


Fact | Dimension *Class* | Dim. *Driver Age* | Dim. *Brand* | Dimension *Location* | Measure *noCars*

$$\left(\begin{array}{l} c\!:\mathrm{Car}(\mathrm{class}=k),\\ p\!:\mathrm{Person}(\mathrm{age}=a), p\rightarrow c,\\ b\!:\mathrm{Brand}, c\rightarrow b,\\ s\!:\mathrm{City}, c\rightarrow s, l\!:\mathrm{Country}, s\rightarrow l \end{array}\right) \rightarrow \left(\begin{array}{l} \quad (\mathrm{noCars}=\mathrm{COUNT}(c)), \end{array}\right)$$

Multidimensional Graph Aggregation

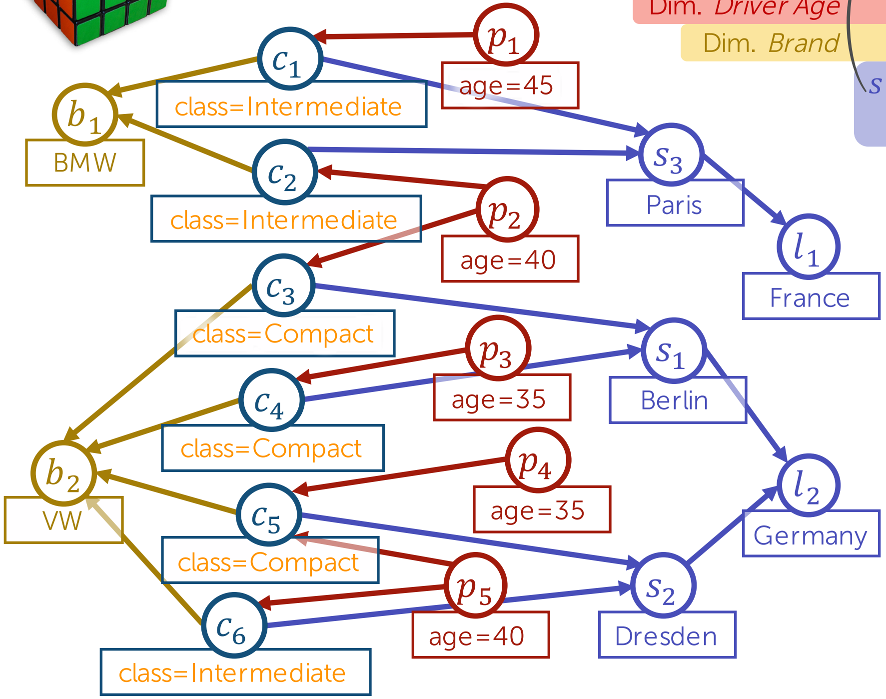
Dresden Database Systems Group

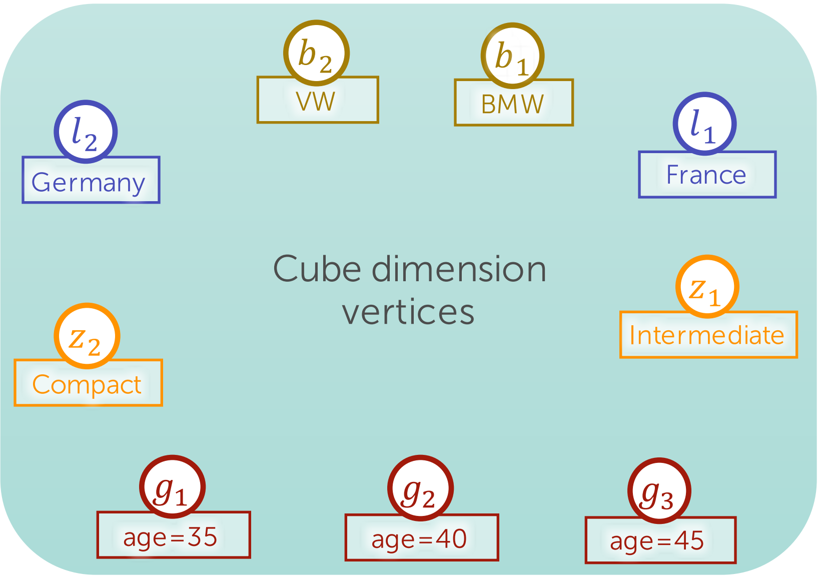


[https://commons.wikimedia.org/wiki/File:Rubiks_revenge_solved.jpg]

Fact · Dimension *Class* · Dim. *Driver Age* · Dim. *Brand* · Dimension *Location* · Measure *noCars*

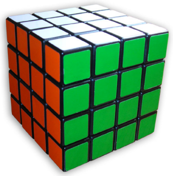
$$\left(\begin{array}{c} c: \mathrm{Car}(\mathrm{class} = k), \\ p: \mathrm{Person}(\mathrm{age} = a), p \to c, \\ b: \mathrm{Brand}, c \to b, \\ s: \mathrm{City}, c \to s, l: \mathrm{Country}, s \to l \end{array} \right) \to \left(\begin{array}{c} (\mathrm{noCars} = \mathrm{COUNT}(c)), \\ g@a: \mathrm{AgeGroup}(\mathrm{age} = a), x \to g, \\ z@k: \mathrm{Class}(\mathrm{name} = k), x \to z, \\ b, x \to b, l, x \to l \end{array} \right)$$





Multidimensional Graph Aggregation

Dresden Database Systems Group



[https://commons.wikimedia.org/wiki/File:Rubiks_revenge_solved.jpg]

Fact

Dimension *Class*

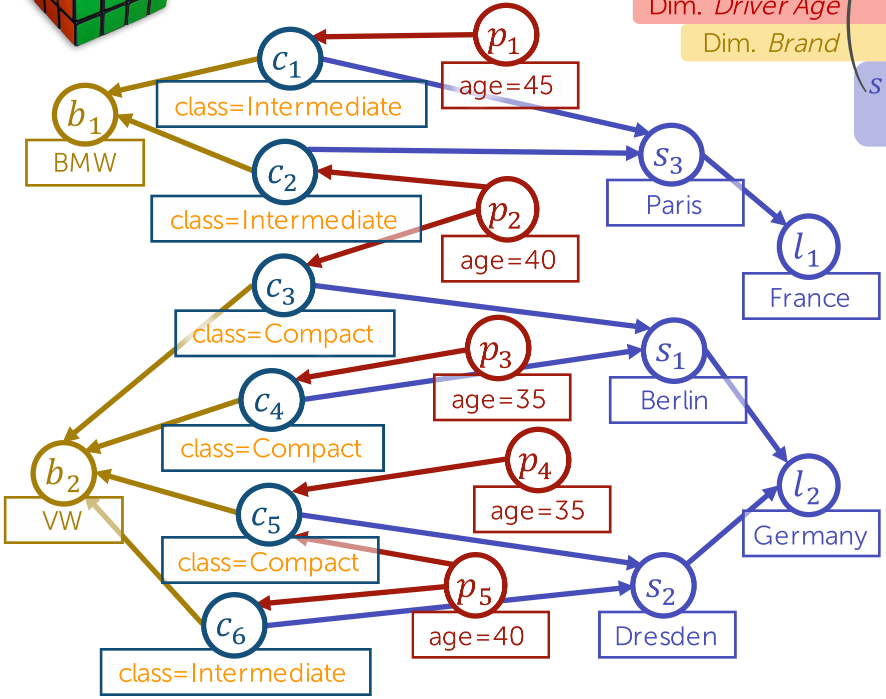
Dim. *Driver Age*

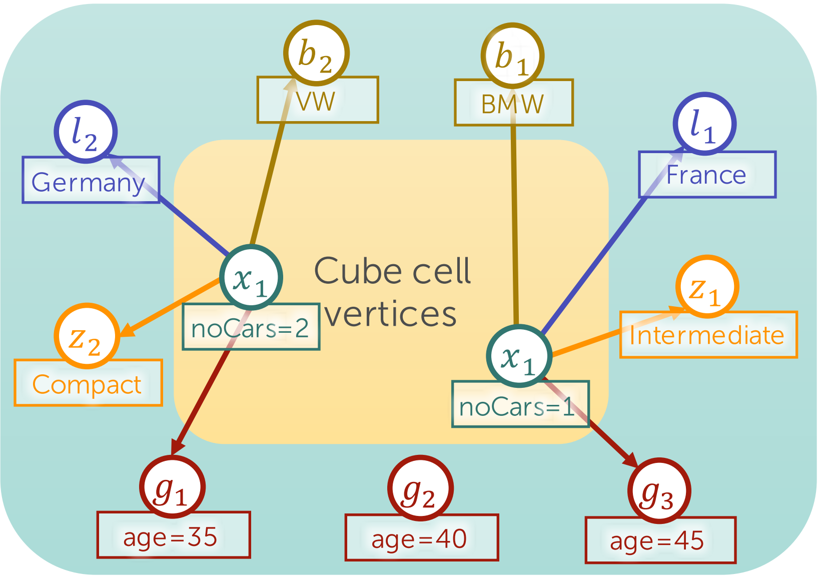
Dim. *Brand*

Dimension *Location*

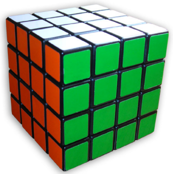
Measure *noCars*

$$\left(\begin{array}{c} c: \mathrm{Car}(\mathrm{class}=k), \\ p:\mathrm{Person}(\mathrm{age}=a), p\to c, \\ b:\mathrm{Brand}, c\to b, \\ s:\mathrm{City}, c\to s, l:\mathrm{Country}, s\to l \end{array}\right) \to \left(\begin{array}{c} x@b,k,a,l(\mathrm{noCars}=\mathrm{COUNT}(c)), \\ g@a:\mathrm{AgeGroup}(\mathrm{age}=a), x\to g, \\ z@k:\mathrm{Class}(\mathrm{name}=k), x\to z, \\ b, x\to b, l, x\to l\end{array}\right)$$



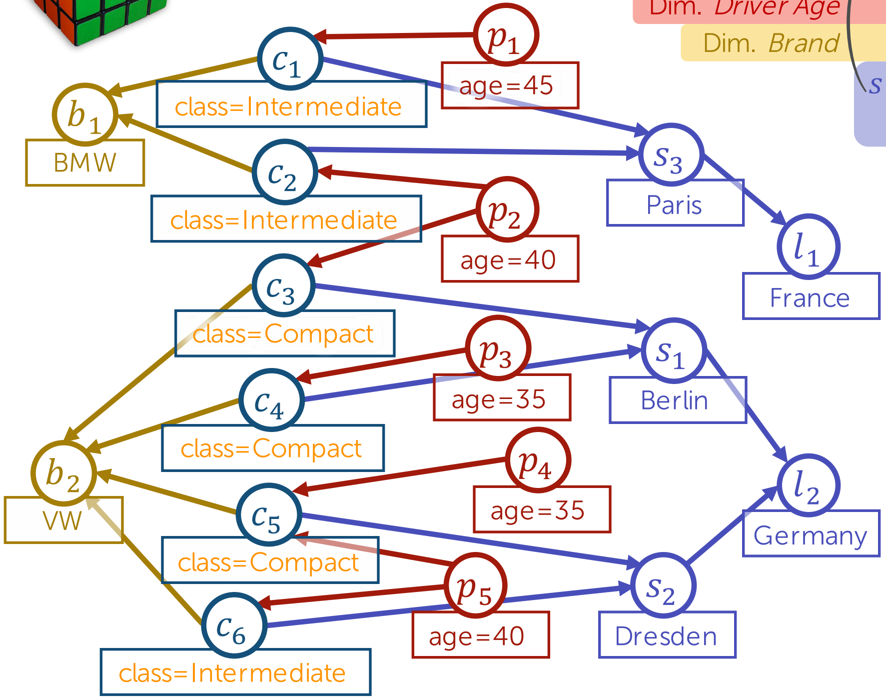


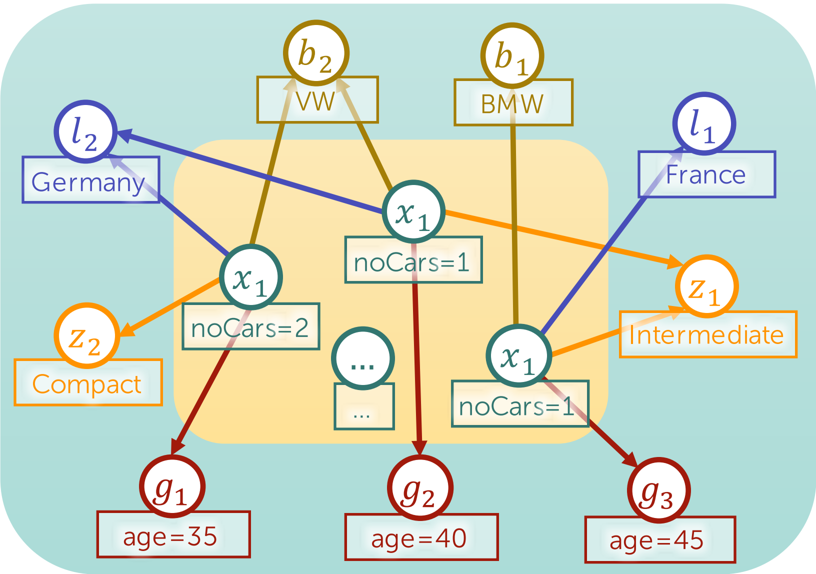
Multidimensional Graph Aggregation



Fact — Dimension *Class* — Dim. *Driver Age* — Dim. *Brand* — Dimension *Location* — Measure *noCars*

$$\left(\begin{array}{c} c\!:\mathrm{Car}(\mathrm{class}=k),\\ p\!:\mathrm{Person}(\mathrm{age}=a), p\to c,\\ b\!:\mathrm{Brand}, c\to b,\\ s\!:\mathrm{City}, c\to s, l\!:\mathrm{Country}, s\to l \end{array}\right) \to \left(\begin{array}{c} x@b,k,a,l(\mathrm{noCars}=\mathrm{COUNT}(c)),\\ g@a\!:\mathrm{AgeGroup}(\mathrm{age}=a), x\to g,\\ z@k\!:\mathrm{Class}(\mathrm{name}=k), x\to z,\\ b, x\to b, l, x\to l \end{array}\right)$$



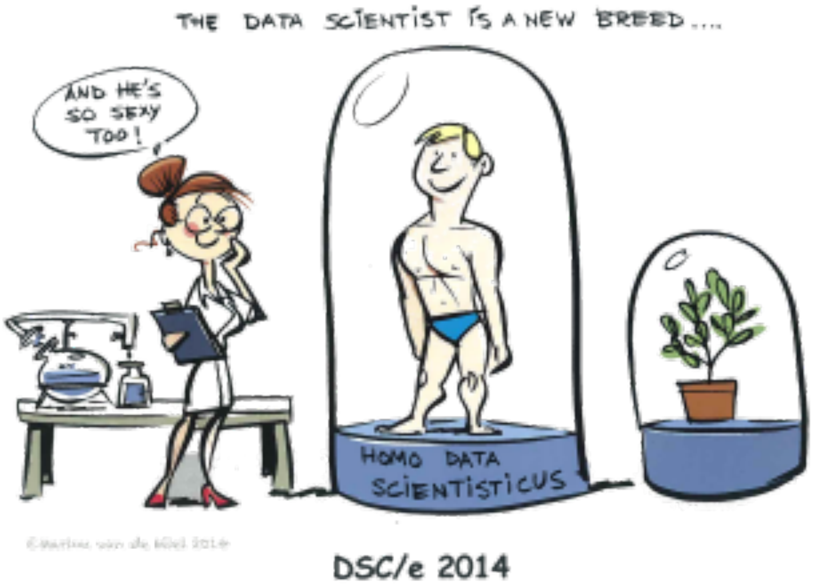


Interactive Multidimensional Graph Exploration

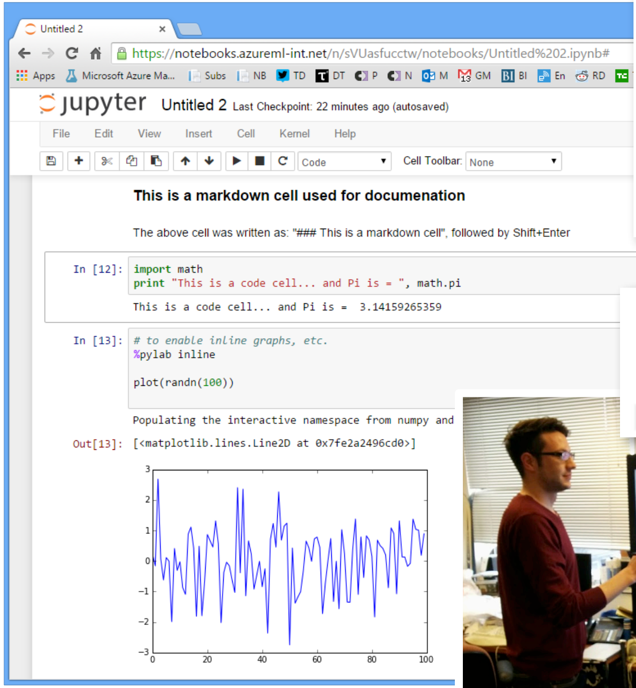
Interactive Exploration

Dresden Database
Systems Group

Consider a data scientist doing multidimensional data exploration



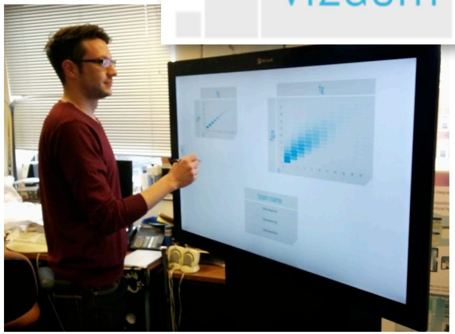
[http://wwwis.win.tue.nl/~wvdaalst/data_science/data_science.html]



jupyter

[http://jupyter.org/]

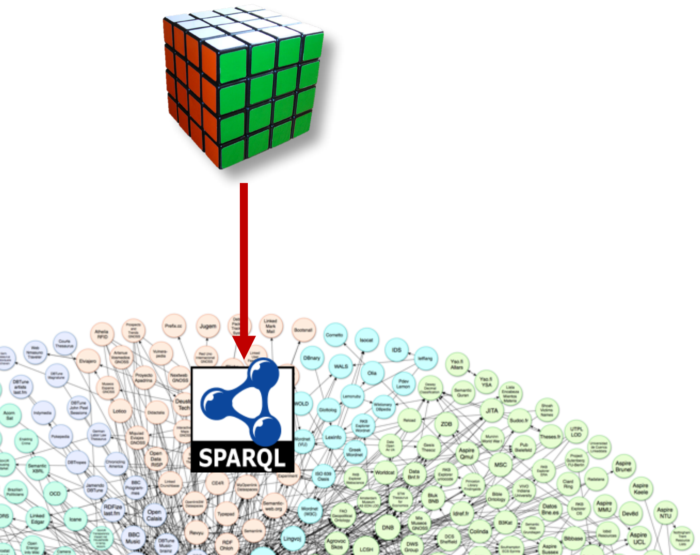
vizdom

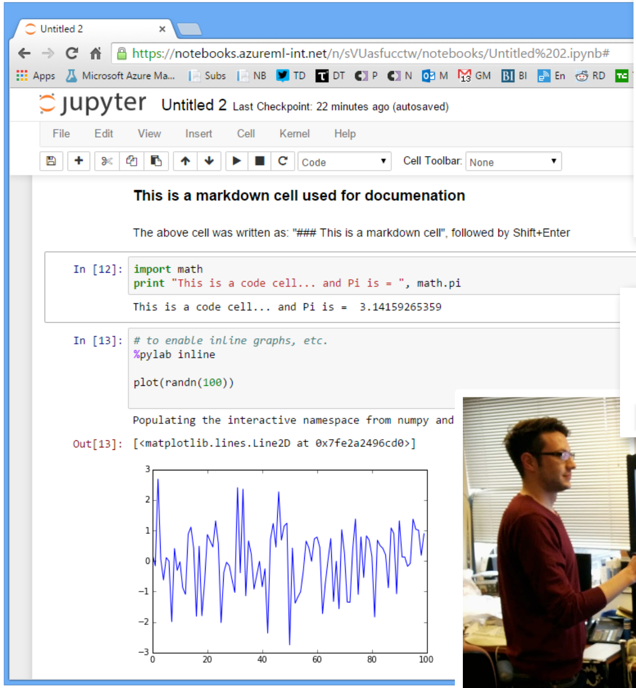


[Crotty et al. Vizdom. VLDB 2015]

Interactive Exploration

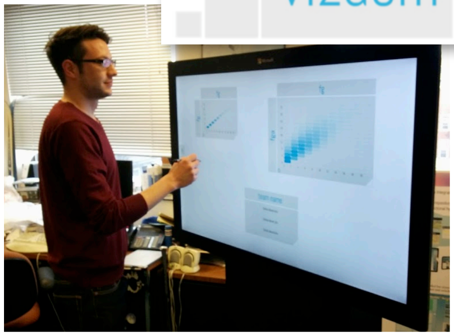
Dresden Database
Systems Group





[http://jupyter.org/]

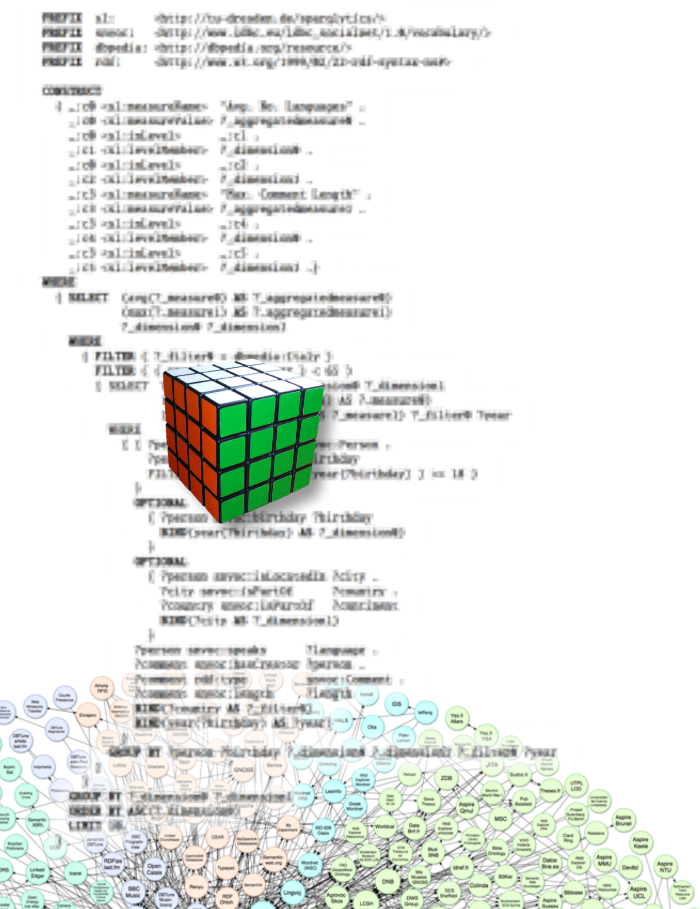
vizdom

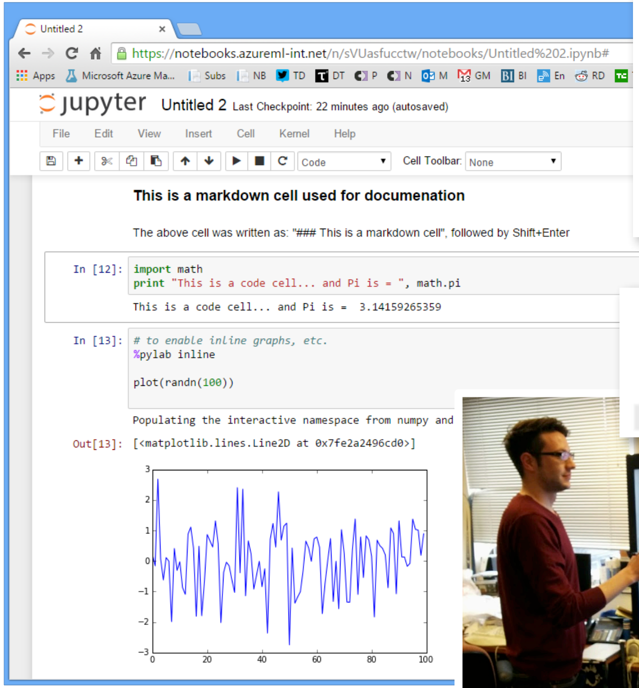


[Crotty et al. Vizdom, VLDB 2015]

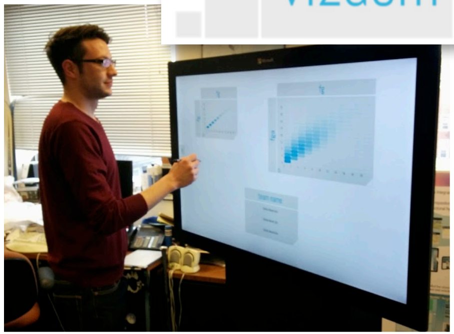
Interactive Exploration

Dresden Database Systems Group



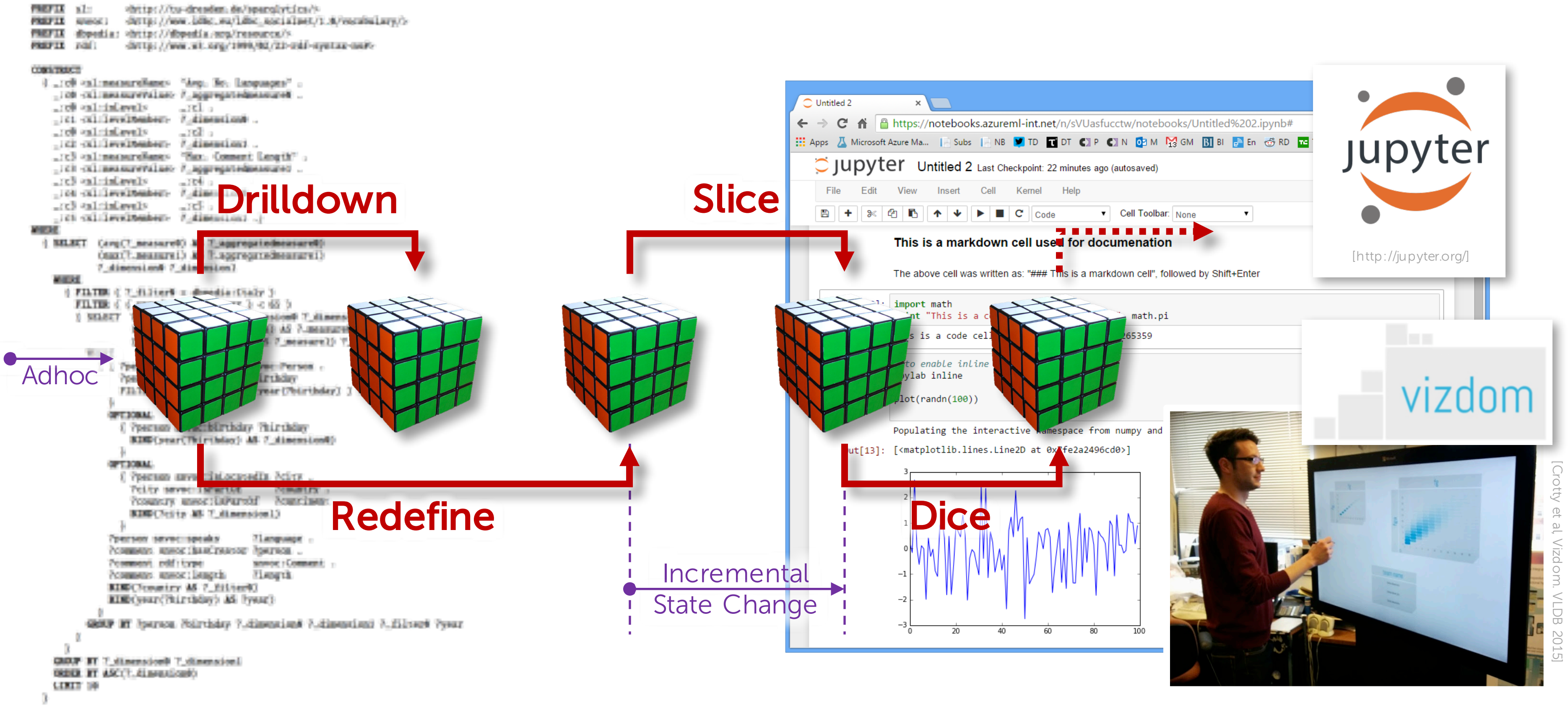


[http://jupyter.org/]

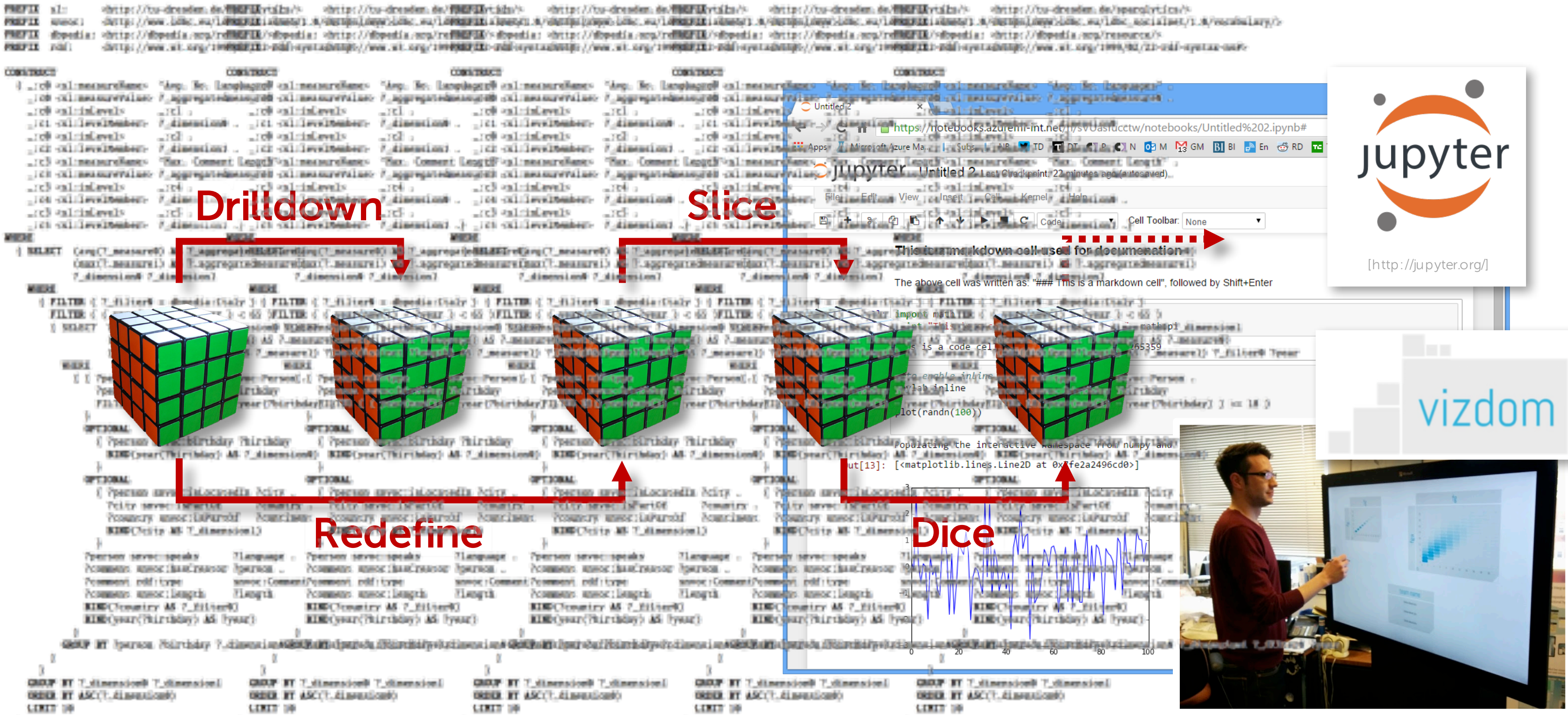


[Crotty et al. Vizdom. VLDB 2015]

Interactive Exploration



Interactive Exploration



[Crotty et al. Vizdom. VLDB 2015]

SynopSys/SPARQLytics

[Michael Rudolf et al. SynopSys: Foundations for Multidimensional Graph Analytics. BIRTE 2014]

Dresden Database Systems Group

Create artifacts

Artifacts Repository

Fact *Car:* $\langle c\colon \mathrm{Car}() \rangle$

Dimension *Class:* $\langle c\colon \mathrm{Car}(\mathrm{class} = k) \rangle$

Dimension *Driver Age:* $\langle p\colon \mathrm{Person}(\mathrm{age} = a), p \to c \rangle$

Dimension *Brand:* $\langle b\colon \mathrm{Brand}, c \to b \rangle$

Dimension *Location:* $\langle s\colon \mathrm{City}, c \to s, l\colon \mathrm{Country}, s \to l \rangle$

Measure *noCars:* $\langle \mathrm{noCars} = \mathrm{COUNT}(c) \rangle$

SynopSys/SPARQLytics

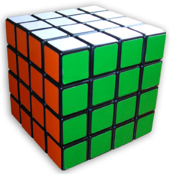
[Michael Rudolf et al. SynopSys: Foundations for Multidimensional Graph Analytics. BIRTE 2014]

Create artifacts

Artifacts Repository

- Fact *Car:* $\langle c\colon \mathrm{Car}() \rangle$
- Dimension *Class:* $\langle c\colon \mathrm{Car}(\mathrm{class} = k) \rangle$
- Dimension *Driver Age:* $\langle p\colon \mathrm{Person}(\mathrm{age} = a), p \to c \rangle$
- Dimension *Brand:* $\langle b\colon \mathrm{Brand}, c \to b \rangle$
- Dimension *Location:* $\langle s\colon \mathrm{City}, c \to s, l\colon \mathrm{Country}, s \to l \rangle$
- Measure *noCars:* $\langle \mathrm{noCars} = \mathrm{COUNT}(c) \rangle$

Define cube by selecting predefined artifact



Rollup/ Drilldown/ etc.

Compute (generates the actual query)

$$\begin{pmatrix} c\colon \mathrm{Car}(\mathrm{class} = k), \\ p\colon \mathrm{Person}(\mathrm{age} = a), p \to c, \\ b\colon \mathrm{Brand}, c \to b, \\ s\colon \mathrm{City}, c \to s, l\colon \mathrm{Country}, s \to l \end{pmatrix} \to \begin{pmatrix} x@b, k, a, l(\mathrm{noCars} = \mathrm{COUNT}(c)), \\ g@a\colon \mathrm{AgeGroup}(\mathrm{age} = a), x \to g, \\ z@k\colon \mathrm{Class}(\mathrm{name} = k), x \to z, \\ b, x \to b, l, x \to l \end{pmatrix}$$

SynopSys/SPARQLytics

[Michael Rudolf et al. SynopSys: Foundations for Multidimensional Graph Analytics. BIRTE 2014]

Create artifacts

Redefine cube/Define another cube

Artifacts Repository

Fact *Car:* $\langle c\colon \mathrm{Car}() \rangle$

Dimension *Class:* $\langle c\colon \mathrm{Car}(\mathrm{class} = k) \rangle$

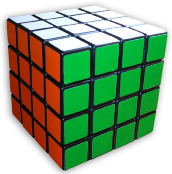
Dimension *Driver Age:* $\langle p\colon \mathrm{Person}(\mathrm{age} = a), p \to c \rangle$

Dimension *Brand:* $\langle b\colon \mathrm{Brand}, c \to b \rangle$

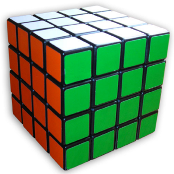
Dimension *Location:* $\langle s\colon \mathrm{City}, c \to s, l\colon \mathrm{Country}, s \to l \rangle$

Measure *noCars:* $\langle \mathrm{noCars} = \mathrm{COUNT}(c) \rangle$

Define cube by selecting predefined artifact



Rollup/ Drilldown/ etc.



Compute (generates the actual query)

$$\begin{pmatrix} c\colon \mathrm{Car}(\mathrm{class} = k), \\ p\colon \mathrm{Person}(\mathrm{age} = a), p \to c, \\ b\colon \mathrm{Brand}, c \to b, \\ s\colon \mathrm{City}, c \to s, l\colon \mathrm{Country}, s \to l \end{pmatrix} \to \begin{pmatrix} x@b, k, a, l\big(\mathrm{noCars} = \mathrm{COUNT}(c)\big), \\ g@a\colon \mathrm{AgeGroup}\,(\mathrm{age} = a), x \to g, \\ z@k\colon \mathrm{Class}(\mathrm{name} = k), x \to z, \\ b, x \to b, l, x \to l \end{pmatrix}$$

SynopSys/SPARQLytics – Experiments

Dresden Database Systems Group

LDBC Social Network Benchmark – BI Workload

Q1 - Posting summary
Q2 - Top tags for country, age, gender, time
Q3 - Tag evolution
Q4 - Popular topics in a country
Q5 - Top posters in a country
Q6 - Most active Posters of a given Topic
Q7 - Most authoritative users on a given topic
Q8 - Related Topics
Q9 - Forum with related Tags
Q10 - Central Person for a Tag
Q11 - Unrelated Replies
Q12 - Trending Posts
Q13 - Popular Tags per month in a country
Q14 - Top thread initiators
Q15 - Social Normals
Q16 - Experts in Social Circle
Q17 - Friend Triangles
Q18 - How many persons have a given number of p
Q19 - Stranger's Interaction
Q20 - High level topics
Q21 - Zombies in a country
Q22 - International Dialog
Q23 - Holiday Destinations
Q24 - Messages By Topic And Continent

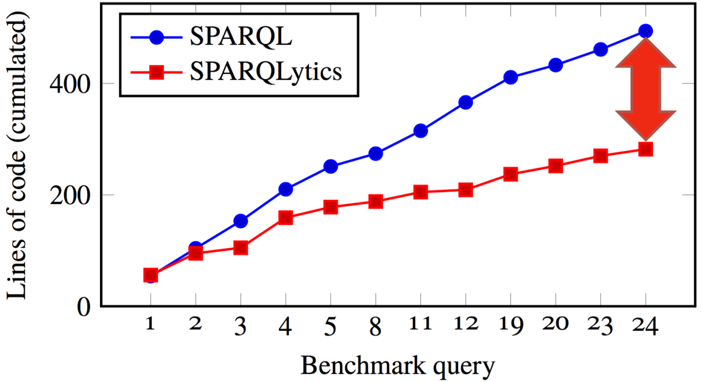
Interactive Exploration

Very low reuse, in practice likely to be much higher!

Unfavorable setting for our approach!

Already considerably fewer lines of code needed (factor 2)

Metric	Maximum	Average
Re-use per dimension	3	1.6
Re-use per measure	5	1.8
Dimensions per cube	8	3.5
Levels per dimension	3	1.5
Measures per cube	4	1.6



Aggregating Graph Structure

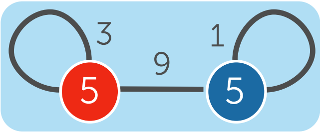
Graph Aggregation

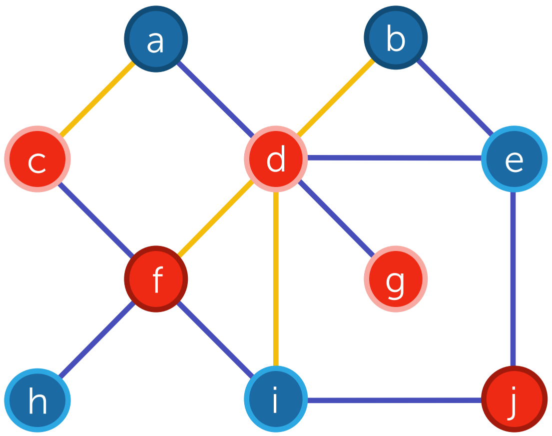
[Peixiang Zhao et al: Graph Cube: On Warehousing and OLAP Multidimensional Networks, SIGMOD 2011]

SUMMARIZE THE STRUCTURE OF A GRAPH IN A SMALLER GRAPH

- Group all vertices and all edge
- Represent the relationship of the groups in a graph

$$\left(\gamma_{\mathrm{gender},\mathrm{COUNT}(*)}V, \gamma_{\emptyset,\mathrm{COUNT}(*)}E\right)$$





Schema: Male/Teacher Female/Teacher Male/Lawyer Female/Lawyer Co-workers Friends

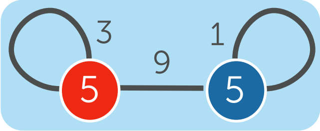
Graph Aggregation

[Peixiang Zhao et al.: Graph Cube: On Warehousing and OLAP Multidimensional Networks, SIGMOD 2011]

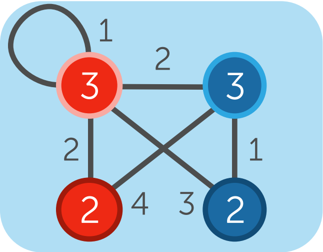
SUMMARIZE THE STRUCTURE OF A GRAPH IN A SMALLER GRAPH

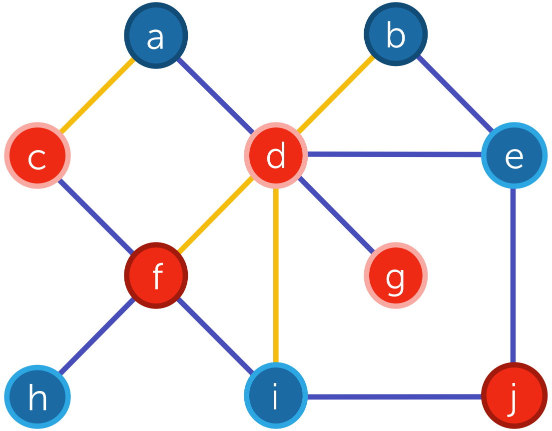
- Group all vertices and all edge
- Represent the relationship of the groups in a graph

$$\left(\gamma^{V}_{\text{gender},\text{COUNT}(*)}, \gamma^{E}_{\emptyset,\text{COUNT}(*)}\right)$$



$$\left(\gamma^{V}_{\text{gender},\text{job},\text{COUNT}(*)}, \gamma^{E}_{\emptyset,\text{COUNT}(*)}\right)$$





Schema: Male/Teacher | Female/Teacher | Male/Lawyer | Female/Lawyer | Co-workers | Friends

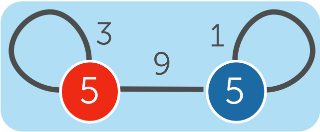
Graph Aggregation

[Peixiang Zhao et al: Graph Cube: On Warehousing and OLAP Multidimensional Networks, SIGMOD 2011]

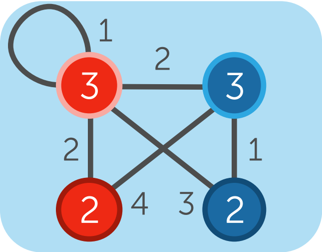
SUMMARIZE THE STRUCTURE OF A GRAPH IN A SMALLER GRAPH

- Group all vertices and all edge
- Represent the relationship of the groups in a graph

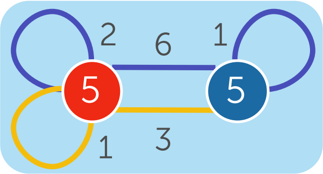
$$\left(\gamma_{\text{gender,COUNT}(*)}V, \gamma_{\emptyset,\text{COUNT}(*)}E\right)$$

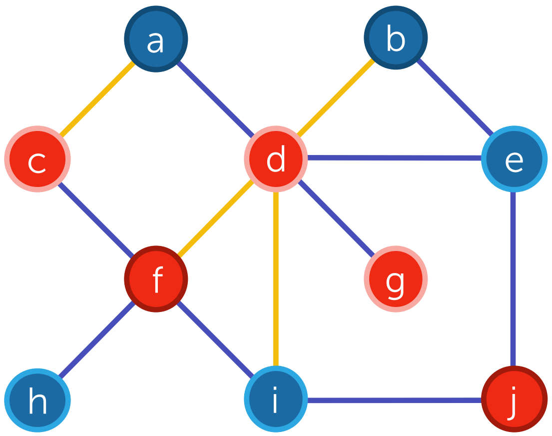


$$\left(\gamma_{\text{gender,job,COUNT}(*)}V, \gamma_{\emptyset,\text{COUNT}(*)}E\right)$$



$$\left(\gamma_{\text{gender,COUNT}(*)}V, \gamma_{\text{status,COUNT}(*)}E\right)$$





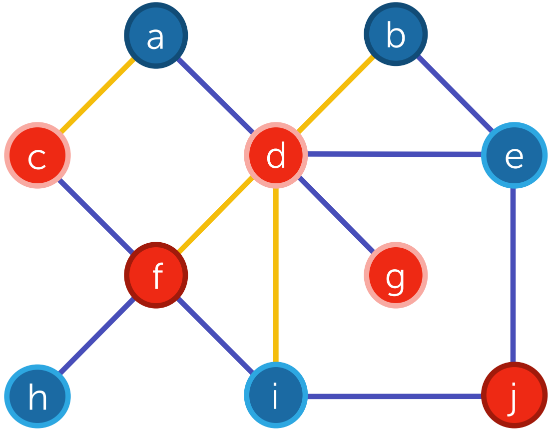
Schema: Male/Teacher | Female/Teacher | Male/Lawyer | Female/Lawyer | Co-workers | Friends

Graph Aggregation

[Peixiang Zhao et al.: Graph Cube: On Warehousing and OLAP Multidimensional Networks, SIGMOD 2011]

Summarize the structure of a graph in a smaller graph

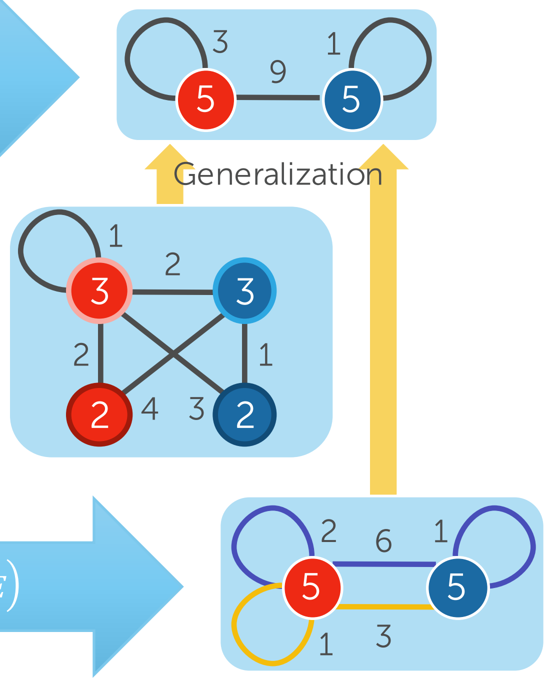
- Group all vertices and all edge
- Represent the relationship of the groups in a graph



$$\left(\gamma_{\text{gender},\text{COUNT}(*)}V, \gamma_{\emptyset,\text{COUNT}(*)}E\right)$$

$$\left(\gamma_{\text{gender},\text{job},\text{COUNT}(*)}V, \gamma_{\emptyset,\text{COUNT}(*)}E\right)$$

$$\left(\gamma_{\text{gender},\text{COUNT}(*)}V, \gamma_{\text{status},\text{COUNT}(*)}E\right)$$



Schema: Male/Teacher | Female/Teacher | Male/Lawyer | Female/Lawyer | Co-workers | Friends

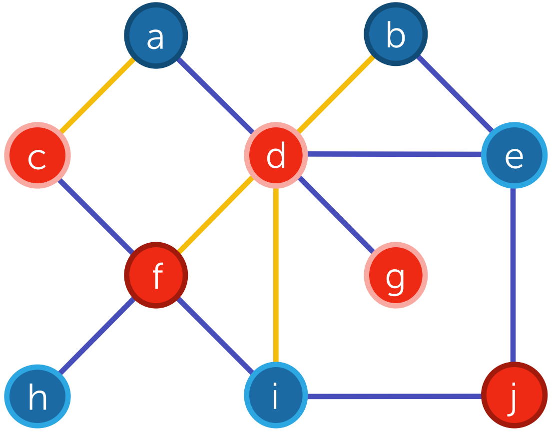
Graph Aggregation

Dresden Database
Systems Group

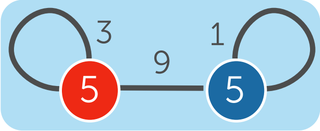
[Peixiang Zhao et al: Graph Cube: On Warehousing and OLAP Multidimensional Networks, SIGMOD 2011]

Summarize the structure of a graph in a smaller graph

- Group all vertices and all edge
- Represent the relationship of the groups in a graph



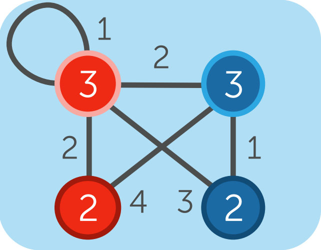
$(\gamma_{gender,COUNT(*)}V, \gamma_{\emptyset,COUNT(*)}E)$



Generalization

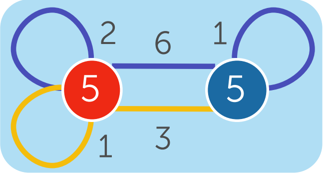
$(\gamma_{gender,SUM(*)}V, \gamma_{\emptyset,SUM(cnt)}E)$

$(\gamma_{gender,job,COUNT(*)}V, \gamma_{\emptyset,COUNT(*)}E)$



$(V, \gamma_{\emptyset,SUM(cnt)}E)$

$(\gamma_{gender,COUNT(*)}V, \gamma_{status,COUNT(*)}E)$



Schema: Male/Teacher | Female/Teacher | Male/Lawyer | Female/Lawyer | Co-workers | Friends

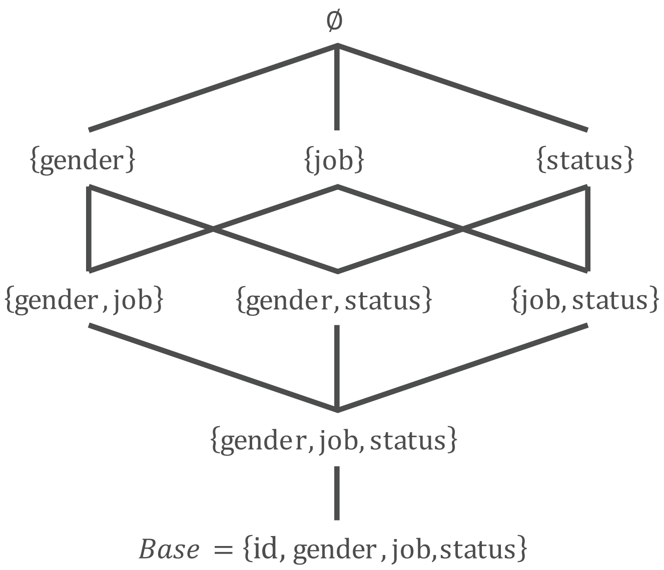
Graph Aggregation

[Peixiang Zhao et al.: Graph Cube: On Warehousing and OLAP Multidimensional Networks, SIGMOD 2011]

Graph cube (cuboid)

- Cube of all possible aggregation of a graph
- Example grouping attributes: $\{\text{id}, \text{gender}, \text{job}, \text{status}\}$

Grouping Lattice

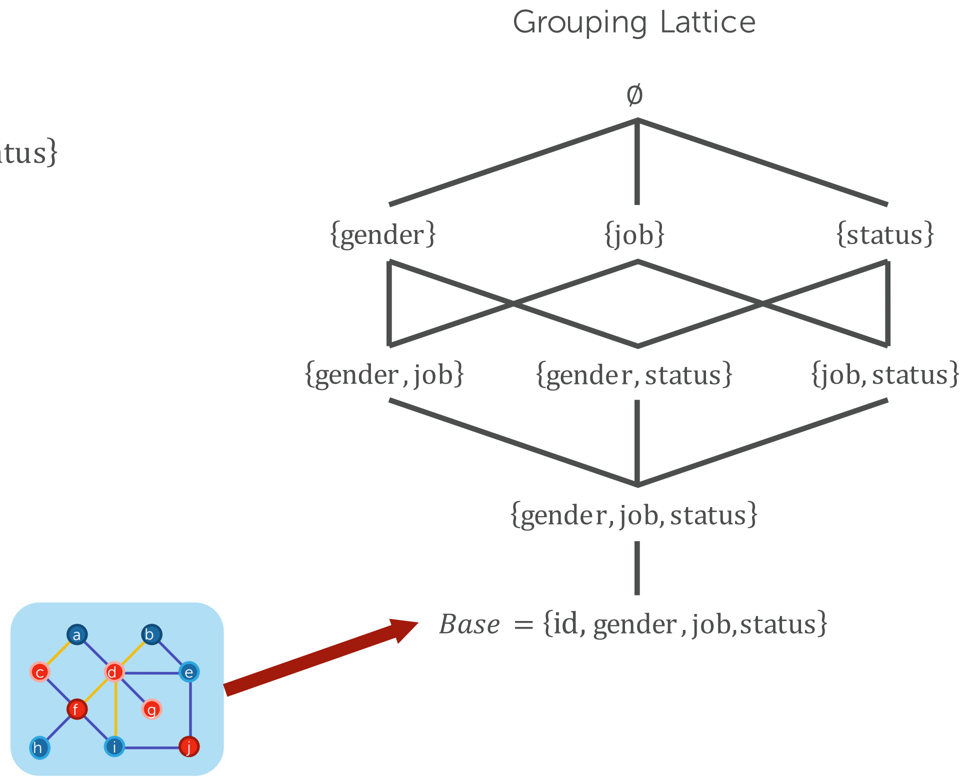


Graph Aggregation

[Peixiang Zhao et al.: Graph Cube: On Warehousing and OLAP Multidimensional Networks, SIGMOD 2011]

GRAPH CUBE (CUBOID)

- Cube of all possible aggregation of a graph
- Example grouping attributes: $\{\text{id}, \text{gender}, \text{job}, \text{status}\}$

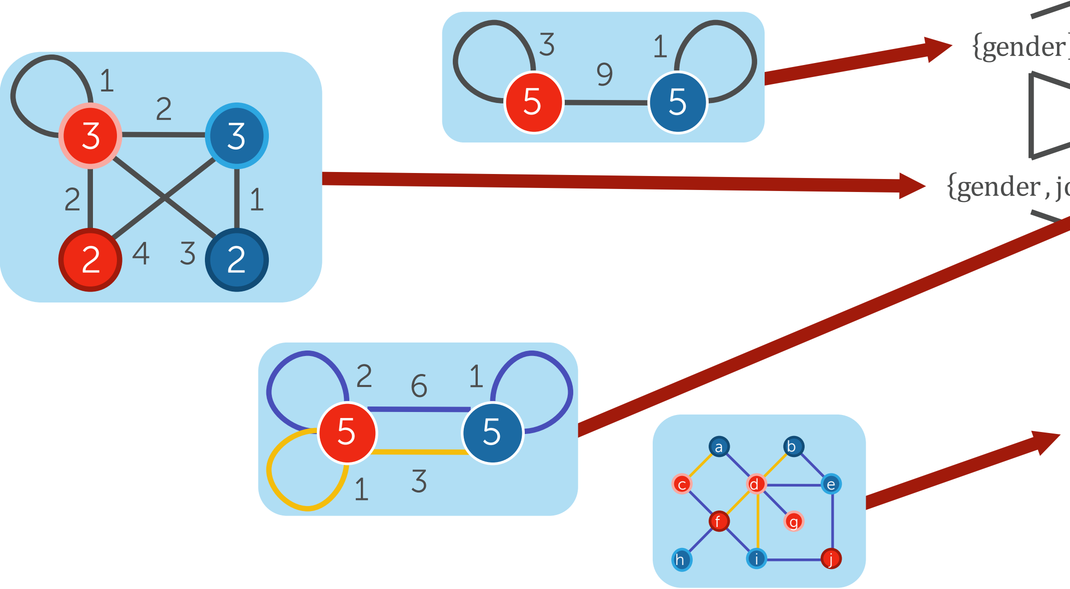


Graph Aggregation

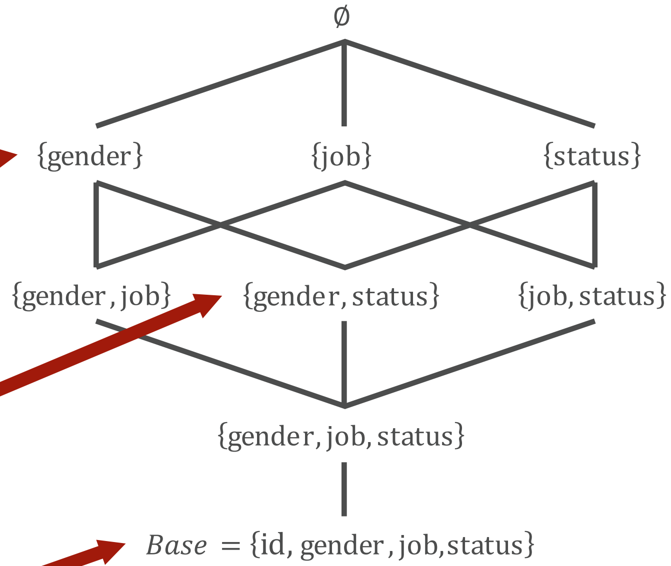
[Peixiang Zhao et al.: Graph Cube: On Warehousing and OLAP Multidimensional Networks, SIGMOD 2011]

Graph Cube (Cuboid)

- Cube of all possible aggregation of a graph
- Example grouping attributes: $\{\text{id}, \text{gender}, \text{job}, \text{status}\}$



Grouping Lattice

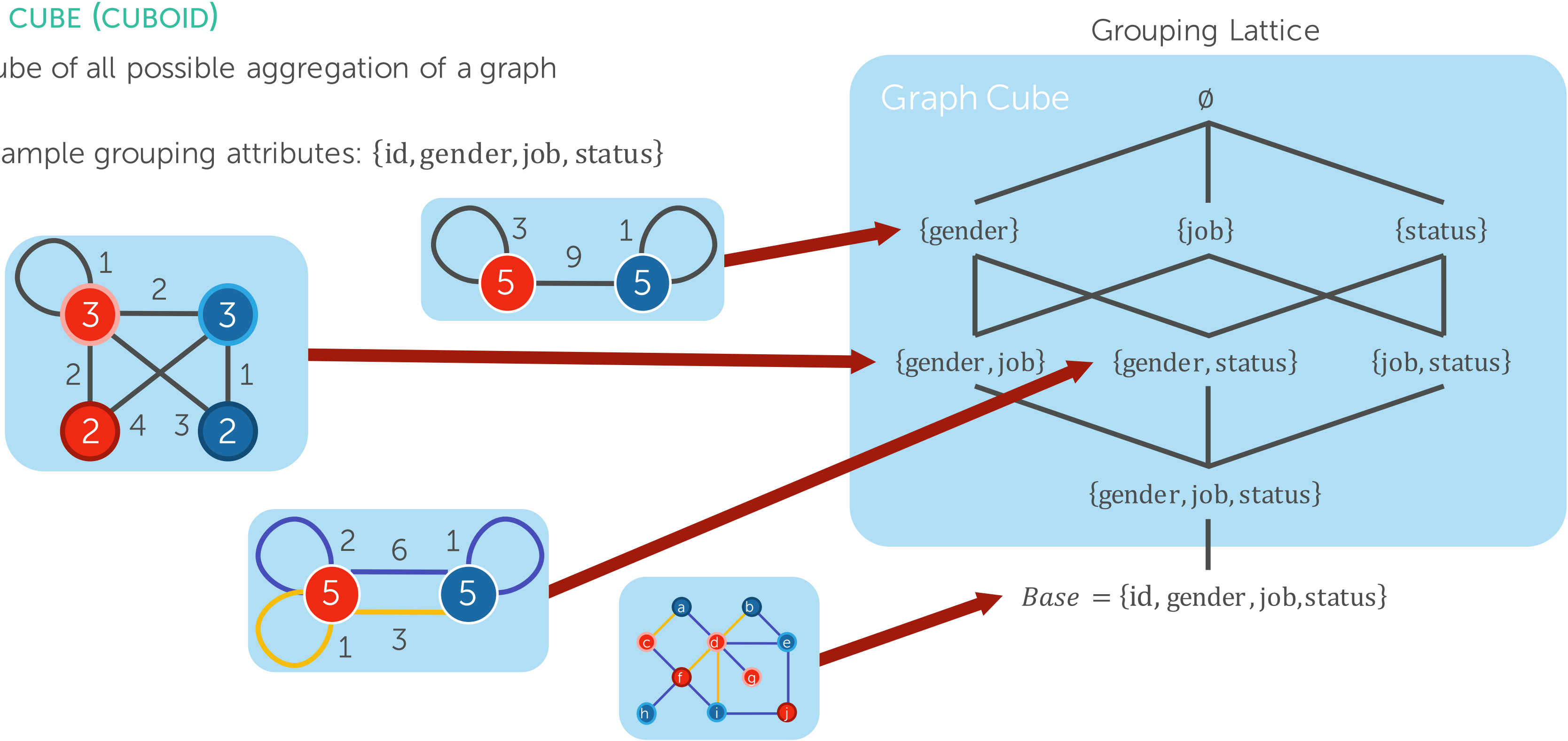


Graph Aggregation

[Peixiang Zhao et al.: Graph Cube: On Warehousing and OLAP Multidimensional Networks, SIGMOD 2011]

Graph Cube (Cuboid)

- Cube of all possible aggregation of a graph
- Example grouping attributes: $\{\mathrm{id}, \mathrm{gender}, \mathrm{job}, \mathrm{status}\}$

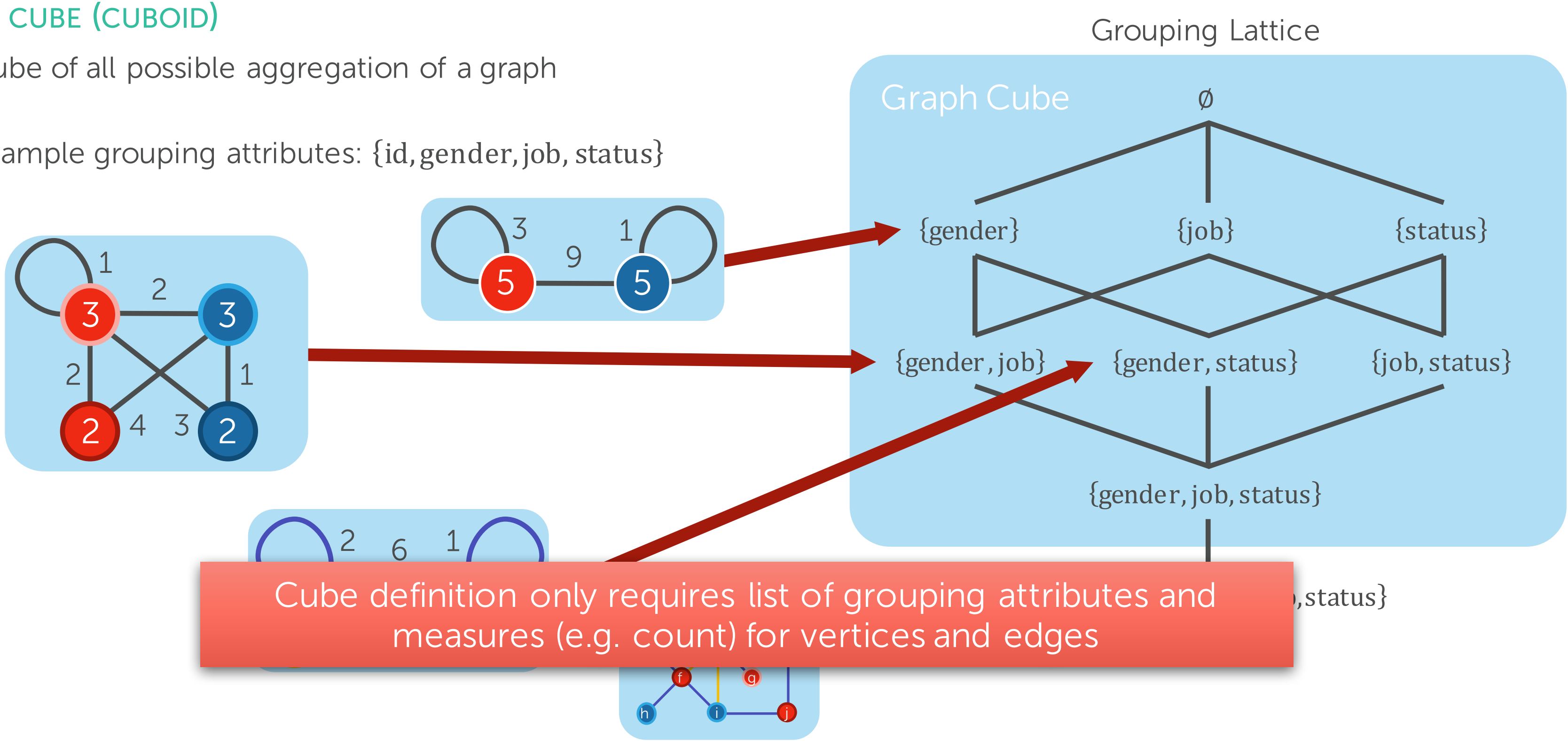


Graph Aggregation

[Peixiang Zhao et al.: Graph Cube: On Warehousing and OLAP Multidimensional Networks, SIGMOD 2011]

GRAPH CUBE (CUBOID)

- Cube of all possible aggregation of a graph
- Example grouping attributes: $\{\text{id}, \text{gender}, \text{job}, \text{status}\}$

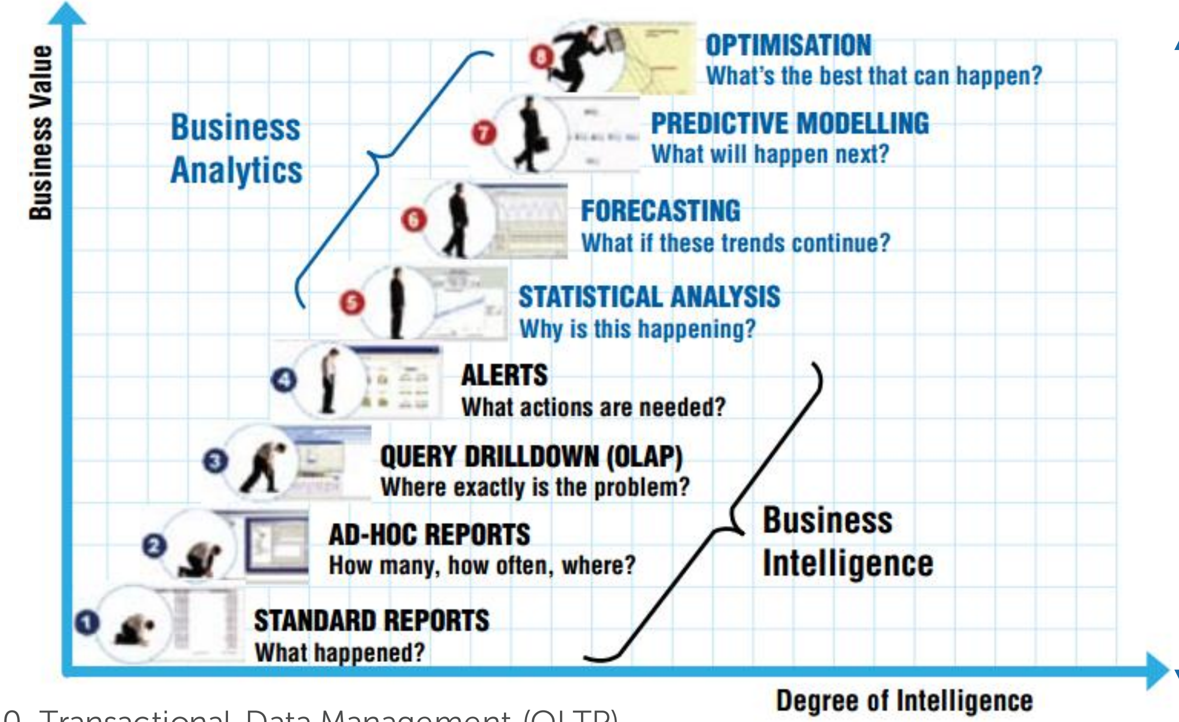


Cube definition only requires list of grouping attributes and measures (e.g. count) for vertices and edges

Level of Analytics

Dresden Database Systems Group

[http://www.rosebt.com/blog/eight-levels-of-analytics-business-intelligence-to-business-analytics]



0. Transactional Data Management (OLTP)

batch
long running
imperative

Graph Data Management Applications

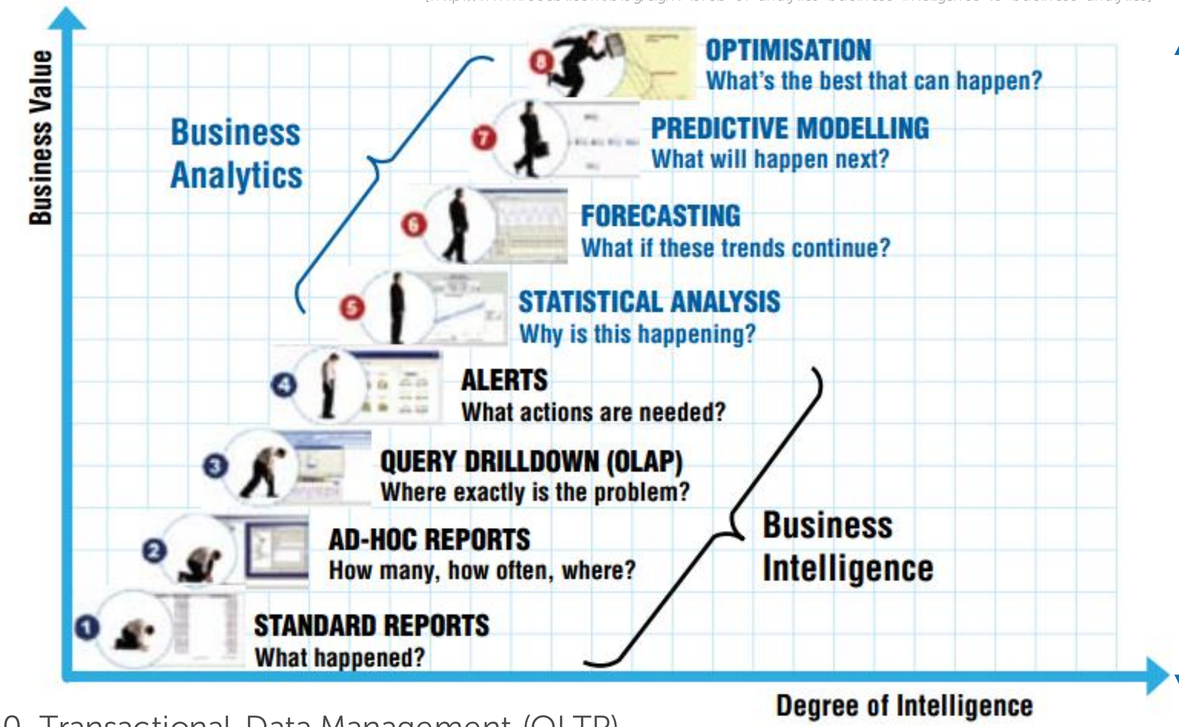
ad-hoc
short running
declarative

Integration with Machine Learning and Optimization Methods

Composable Graph Matching w/ Aggregation

Level of Analytics

[http://www.rosebt.com/blog/eight-levels-of-analytics-business-intelligence-to-business-analytics]



0. Transactional Data Management (OLTP)

batch
long running
imperative

Graph Data Management Applications

ad-hoc
short running
declarative

Vertex-centric Programming

Composable Graph Matching w/ Aggregation

Vertex-centric Programming

Vertex-centric Programming Model

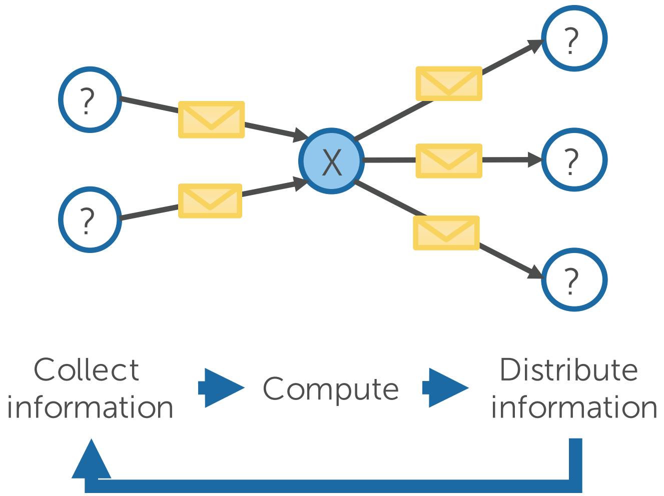
Compute - Communicate

- Based on stateless user-defined function(s)
- Collection data from adjacent vertices
- Compute new state of vertex (update)
- Send data to adjacent vertices
- A vertex can be set to inactive to vote for termination of the whole computation

- Processing terminates when all vertices are simultaneously inactive and there is no data in transit

- User has to provide Compute function
- Depending framework further function are necessary, controlling data collection and sending

Think like a vertex!



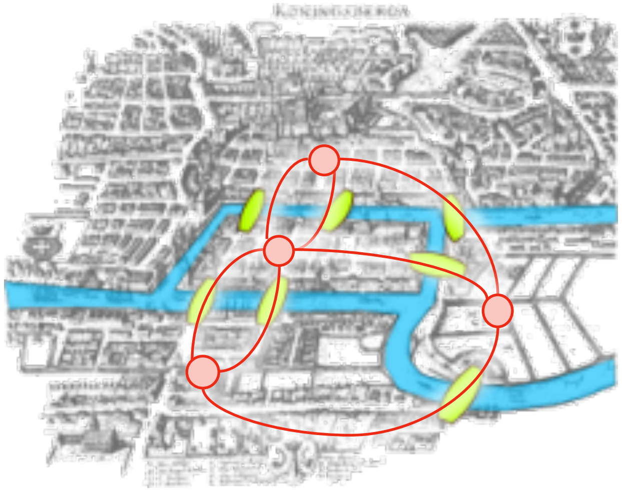
Pregel/Giraph

[Grzegorz Malewicz et al.: Pregel: a system for large-scale graph processing. SPAA 2009/PODC 2009/SIGMOD 2010]

Pregel

- Developed by Google
- "a scalable and fault-tolerant platform with an API that is sufficiently flexible to express arbitrary graph algorithms"
- For directed graphs with vertex and edge labels (byte strings)
- First framework using vertex-centric API
- Vertices exchange instruction messages along edges
- Bulk-Synchronous-Parallel (BSP) processing in super steps

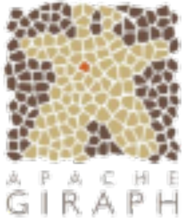
Euler's Seven Bridges of Königsberg (from 1735)



Pregel River

Apache Giraph

- Open source implementation of Pregel



Pregel/Giraph

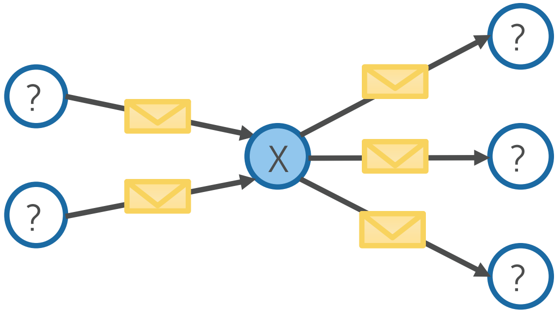
[Grzegorz Malewicz et al.: Pregel: a system for large-scale graph processing. SPAA 2009/PODC 2009/SIGMOD 2010]

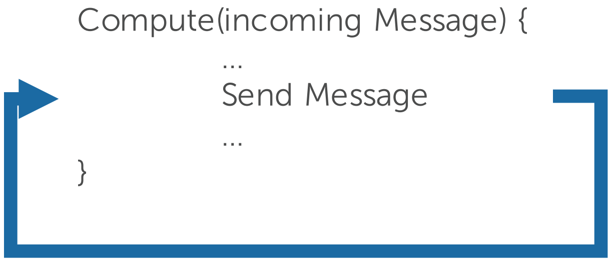
MESSAGE ABSTRACTION

- Data send between vertices, typically neighbors
- Think of message as data store on edges
- Exception
  - Combined message (cf. Combiner)
  - message not send directly, not along edge
- Receiving message = collecting data
- Sending message = sending data
- Message delivery done by framework

SINGLE COMPUTE-FUNCTION

- Gets incoming messages as parameter
  - Think of reading data on incoming edges of current vertex
- Computes new vertex state
- Sends you new messages
  - Think of write data on outgoing edges of current vertex





Pregel/Giraph

[Grzegorz Malewicz et al.: Pregel: a system for large-scale graph processing. SPAA 2009/PODC 2009/SIGMOD 2010]

Writing a vertex-centric program

- Subclassing the predefined `Vertex` class
- Virtual `compute()` method, which will be executed at each active vertex in every super step
- `Vertex` class provides `compute()` helper methods
- Get vertex id with `vertex_id()` and super step with `superstep()`
- Inspect the value associated with its vertex via `GetValue()` or modify it via `MutableValue()`
- Get outgoing edges with `getOutEdgeIterator()`
- Send messages to other vertices with `sendMessageTo(...)`
- Change vertex state from active to halt with `voteToHalt()`

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void compute(MessageIterator* msgs) = 0;
  const string& vertex_id() const;
  int64 superstep() const;
  const VertexValue& getValue();
  VertexValue* mutableValue();
  OutEdgeIterator getOutEdgeIterator();
  void sendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void voteToHalt();
};
```

Pregel/Giraph

[Grzegorz Malewicz et al.: Pregel: a system for large-scale graph processing. SPAA 2009/PODC 2009/SIGMOD 2010]

Combiners

- User-written code
- Combines all message for a vertex V into a single message
- Reduces overhead of message passing
- Enabled by subclassing the `Combiner` class and overriding a virtual `combine()` method
- No guarantees about
  - which (if any) messages are combined,
  - the groupings presented to the combiner, or
  - the order of combining
- Combiners operations should be commutative and associative operations.

Aggregators

- User-written code
- Mechanism for global communication, monitoring, and data
- Each vertex can provide a value to an aggregator in super step S
- Aggregator is used to combine these values to a single value
- resulting value is made available to all vertices in super step $S + 1$
- Predefined aggregators for min, max, sum , etc.
- Enabled by subclassing the `Aggregator` class
- Implementation specifies how
  - aggregated value is initialized from the first input value
  - multiple partially aggregated values are reduced to one
- Aggregator operation should be commutative and associative

Pregel/Giraph

Dresden Database Systems Group

[Grzegorz Malewicz et al.: Pregel: a system for large-scale graph processing. SPAA 2009/PODC 2009/SIGMOD 2010]

EXAMPLE: SINGLE-SOURCE SHORTEST PATHS PROBLEM

- Finding a shortest path between a single source vertex and every other vertex in the graph

```
class ShortestPathVertex : public Vertex<int, int, int> {
  void compute(MessageIterator* msgs) {
    int mindist = IsSource(vertex_id()) ? 0 : INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    if (mindist < getValue()) {
      *MutableValue() = mindist;
      OutEdgeIterator iter = GetOutEdgeIterator();
      for (; !iter.Done(); iter.Next())
        SendMessageTo(iter.Target(), mindist + iter.GetValue());
    }
    VoteToHalt();
  }
};
```

If vertex is source shortest distance is 0

Find shortest distance send with messages

If send distance is shorter that shortest distance already known, remember it

and for each out edge: multiply own distance with edge length and send result to target vertex

Combiner:

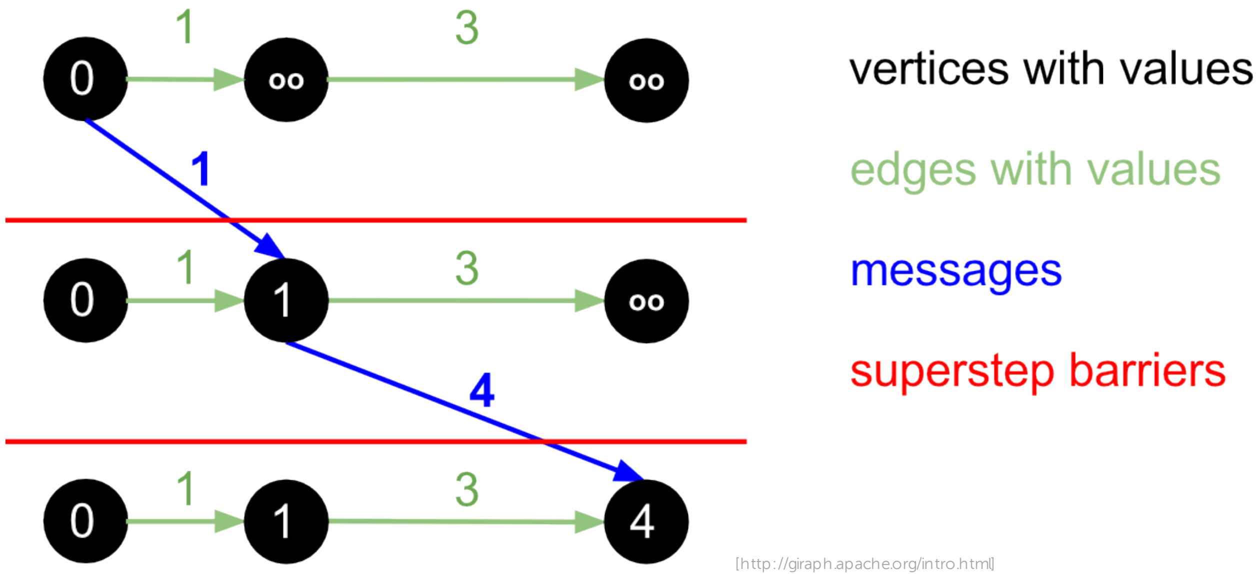
```
class MinIntCombiner : public Combiner<int> {
  virtual void combine(MessageIterator* msgs) {
    int mindist = INF;
    for (; !msgs->Done(); msgs->Next()) mindist = min(mindist, msgs->Value());
    Output("combined_source", mindist);
  }
};
```

Pregel/Giraph

[Grzegorz Malewicz et al.: Pregel: a system for large-scale graph processing. SPAA 2009/PODC 2009/SIGMOD 2010]

Example: Single-Source Shortest Paths Problem

- Finding a shortest path between a single source vertex and every other vertex in the graph



[http://giraph.apache.org/intro.html]

Pregel/Giraph

[Grzegorz Malewicz et al.: Pregel: a system for large-scale graph processing. SPAA 2009/PODC 2009/SIGMOD 2010]

EXAMPLE: SINGLE-SOURCE SHORTEST PATHS PROBLEM

- Finding a shortest path between a single source vertex and every other vertex in the graph

```
public void compute(Iterable<DoubleWritable> messages) {
  double minDist = Double.MAX_VALUE;
  for (DoubleWritable message : messages) {
    minDist = Math.min(minDist, message.get());
  }
  if (minDist < getValue().get()) {
    setValue(new DoubleWritable(minDist));
    for (Edge<LongWritable, FloatWritable> edge : getEdges()) {
      double distance = minDist + edge.getValue().get();
      sendMessage(edge.getTargetVertexId(), new DoubleWritable(distance));
    }
  }
  voteToHalt();
}
```

[http://giraph.apache.org/intro.html]

Pregel/Giraph

Dresden Database Systems Group

[Grzegorz Malewicz et al.: Pregel: a system for large-scale graph processing. SPAA 2009/PODC 2009/SIGMOD 2010]

Example: Page Rank

```
void compute(MessageIterator* msgs) {
  int sum = 0;
  for (; !msgs->Done(); msgs->Next())
    sum = sum + msgs->Value();
  rank = ALPHA + ((1-ALPHA)/N) * sum;
  *MutableValue() = rank;

  if (superstep() < MAX_STEPS) {
    nedges = <count number of out edges with iterator>;
    OutEdgeIterator iter = GetOutEdgeIterator();
    for (; !iter.Done(); iter.Next())
      SendMessageTo(iter.Target(), rank / nedges);
  } else {
    VoteToHalt();
  }
}
```

Sum page rank over incoming messages

$$R[i] = \alpha + \frac{(1-\alpha)}{N} \sum_{(i,j)\in E} \frac{1}{L[j]} R[j]$$

Send new message over outgoing message or terminate

Page Rank

Vertex-centric Frameworks

Dresden Database Systems Group

SCALE-OUT DISK-BASED

- Pregel
  [Grzegorz Malewicz et al.: Pregel: a system for large-scale graph processing. SPAA 2009/PODC 2009/SIGMOD 2010]
- Giraph
  [http://giraph.apache.org/]

APACHE GIRAPH

SINGLE-BOX DISK-BASED

- GraphChi
  [Kyrola. Ligra: GraphChi: Large-Scale Graph Computation on Just a PC. OSDI 2012]
- TurboGraph
  [Han et al.: TurboGraph: A Fast Parallel Graph Engine Handling Billion-scale Graphs in a Single PC. SIGKDD 2013]

SCALE-OUT IN-MEMORY BASED

- GraphLab (asynchronous)
  [Low et al.: Distributed GraphLab: A Framework for Machine Learning in the Cloud. VLDB 2012]
- PowerGraph (Gather, Sum, Apply, Scatter)
  [Gonzalez et al.: PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. OSDI 2012]
- GraphX (on Apache Spark)
  [http://spark.apache.org/graphx/]
- Gelly (on Apache Flink)
  [https://ci.apache.org/projects/flink/flink-docs-master/apis/batch/libs/gelly.html]

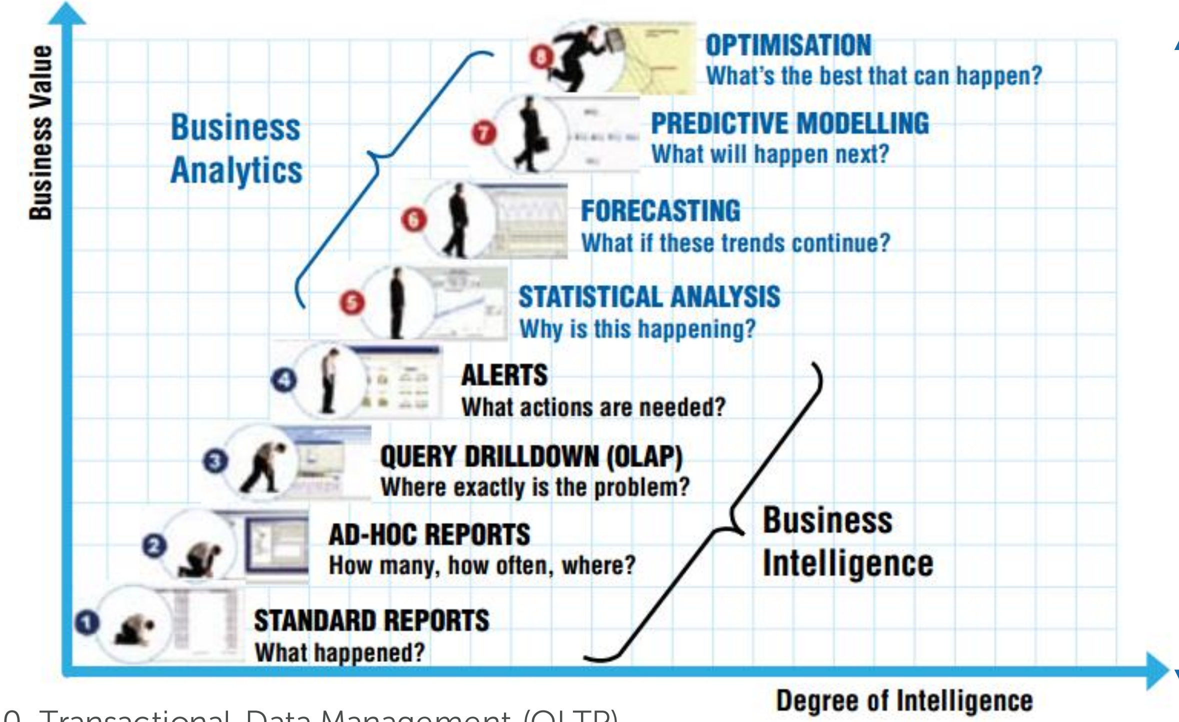
Spark™

SHARED-MEMORY BOX

- Ligra
  [J. Shun and G. E. Blelloch. Ligra: A Lightweight Graph Processing Framework for Shared Memory. PPoPP 2013]
- X-Stream
  [Roy et al.: Ligra: X-Stream: Edge-centric Graph Processing using Streaming Partitions. SIGOPS 2013]
- Polymer (NUMA optimized)
  [Zhang et al.: NUMA-Aware Graph-Structured Analytics. PPoPP 2015]

Level of Analytics

[http://www.rosebt.com/blog/eight-levels-of-analytics-business-intelligence-to-business-analytics]



0. Transactional Data Management (OLTP)

batch
long running
imperative

Graph Data Management Applications

ad-hoc
short running
declarative

Vertex-centric Programming

?

Composable Graph Matching w/ Aggregation

… and many more interesting topics around graphs and data analytics!!!

[http://www.bordalierinstitute.com/images/yeastProteinInteractionNetwork.jpg]

Dresden Database Systems Group

Graph Analytics

Hannes Voigt