Data Processing and Analytics (DISS-DPA)

Principles of Data Quality – Data Quality Rules for Graphs

Christopher Spinrath

Database Group (BD) – CNRS – LIRIS – Université Lyon 1

Fall 2025

Motivation

Previously

- Data quality is an important problem in data management
- Dirty data is everywhere and costly
- A principled approach to detect and repair errors
  - Using quality improving dependencies (QIDs)
  - Capturing conditional functional dependencies (CFDs), matching dependencies (MDs), etc.

Motivation

Previously

- Data quality is an important problem in data management
- Dirty data is everywhere and costly
- A principled approach to detect and repair errors
  - Using quality improving dependencies (QIDs)
  - Capturing conditional functional dependencies (CFDs), matching dependencies (MDs), etc.

In this Episode

Can we transfer these techniques from the relational setting to graphs?

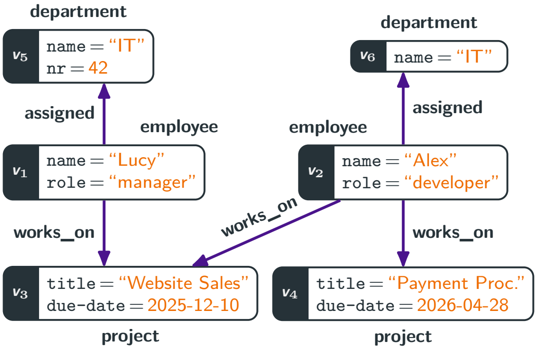
- Instead of a relational database consisting of tables, a database is represented by a graph

Data Model: Graphs with Labels and Properties

Data Model

In this chapter, we represent a database by directed graphs $G = (V, E, L, P)$ where

- V is a set of nodes
- E is a set of edges

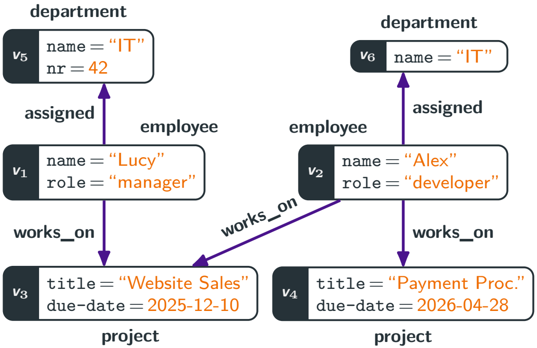


Data Model: Graphs with Labels and Properties

Data Model

In this chapter, we represent a database by directed graphs $G = (V, E, L, P)$ where

- V is a set of nodes
- E is a set of edges
- L is a function that assigns
  - a label $L(v)$ to every node $v \in V$
  - a label $L(e)$ to every edge $e \in E$

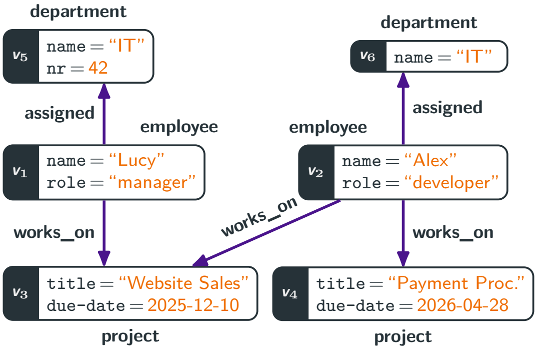


Data Model: Graphs with Labels and Properties

Data Model

In this chapter, we represent a database by directed graphs $G = (V, E, L, P)$ where

- V is a set of nodes
- E is a set of edges
- L is a function that assigns
  - a label $L(v)$ to every node $v \in V$
  - a label $L(e)$ to every edge $e \in E$
- P is a function that assigns
  - a set of key-value pairs, called properties, to every node $v \in V$



Recall: Ingredients for the Repair Problem

1. Quality dependencies
   - We considered quality improving dependencies
2. A dirty relational database
3. A repair model
   - What kind of operations are allowed to modify the database?
   - Examples: tuple deletions, tuple insertions, value modifications
4. A cost model
   - the repair should differ minimally
   - Examples: number of deletions, edit distance

Goal

A clean database that satisfies all the dependencies

Recall: Ingredients for the Repair Problem

1. Quality dependencies
   - We considered quality improving dependencies
   - Are they applicable to graphs?
2. A dirty graph database
3. A repair model
   - What kind of operations are allowed to modify the database?
   - Examples: ~~tuple deletions, tuple insertions~~, value modifications
4. A cost model
   - the repair should differ minimally
   - Examples: number of deletions, edit distance

Goal

A clean graph that satisfies all the dependencies

Recall: Data Improving Dependencies

Recall: Data Improving Dependencies (QIDs)

Formalism for data quality rules that covers

- Functional Dependencies (FDs)
- Conditional Functional Dependencies (CFDs)
- Matching Dependencies (MDs)

Example (A CFD written as a QID)

"In the UK, the zip code uniquely determines the street"

$$\forall t_1 \forall t_2 \Big(\big(\text{Address}(t_1) \wedge \text{Address}(t_2) \wedge t_1[\text{zip}] = t_2[\text{zip}] \wedge t_1[\text{CC}] = t_2[\text{CC}] \wedge t_1[\text{CC}] = 44\big) \rightarrow t_1[\text{street}] = t_2[\text{street}] \Big)$$

Recall: Data Improving Dependencies

Quality Improving Dependency (QID)

A quality improving dependency (QID) is a first-order sentence of the following form

$$\forall t_1 \forall t_2 \Big(\big(R(t_1) \wedge S(t_2) \wedge \bigwedge_{i \in [1,n]} t_1[A_i] \ \mathrm{op}_i \ t_2[B_i] \big) \rightarrow \bigwedge_{j \in [1,m]} t_1[C_j] \ \mathrm{op}'_j \ t_2[D_j] \Big)$$

where the operators op_i and op'_j form the signature of the dependency

Recall: Data Improving Dependencies

Quality Improving Dependency (QID)

A quality improving dependency (QID) is a first-order sentence of the following form

$$\forall t_1 \forall t_2 \Big(\big(R(t_1) \wedge S(t_2) \wedge \bigwedge_{i \in [1,n]} t_1[A_i] \ \mathrm{op}_i \ t_2[B_i]\big) \rightarrow \bigwedge_{j \in [1,m]} t_1[C_j] \ \mathrm{op}'_j \ t_2[D_j] \Big)$$

where the operators op_i and op'_j form the signature of the dependency

Operators

- Equality: $t_1[A] = t_2[B]$ iff attribute A of t_1 and B of t_2 have the same value
- Equality with constant: $t_1[A] =_c t_2[B]$ iff attribute A of t_1 and B of t_2 have value c
- Similarity: $t_1[A] \sim t_2[B]$ iff the values of attribute A of t_1 and B of t_2 are similar relative to some similarity relation $\sim$

Recall: Data Improving Dependencies

Quality Improving Dependency (QID)

A quality improving dependency (QID) is a first-order sentence of the following form

$$\forall t_1 \forall t_2 \Big(\big(R(t_1) \wedge S(t_2) \wedge \bigwedge_{i \in [1,n]} t_1[A_i] \ \mathrm{op}_i \ t_2[B_i] \big) \rightarrow \bigwedge_{j \in [1,m]} t_1[C_j] \ \mathrm{op}'_j \ t_2[D_j] \Big)$$

where the operators op_i and op'_j form the signature of the dependency

Do QIDs Apply to Graphs?

- Attributes correspond to properties of nodes

Recall: Data Improving Dependencies

Quality Improving Dependency (QID)

A quality improving dependency (QID) is a first-order sentence of the following form

$$\forall t_1 \forall t_2 \Big(\big(R(t_1) \wedge S(t_2) \wedge \bigwedge_{i \in [1,n]} t_1[A_i] \ \mathrm{op}_i \ t_2[B_i] \big) \rightarrow \bigwedge_{j \in [1,m]} t_1[C_j] \ \mathrm{op}'_j \ t_2[D_j] \Big)$$

where the operators op_i and op'_j form the signature of the dependency

Do QIDs Apply to Graphs?

- Attributes correspond to properties of nodes
- But what about tuples? Do they correspond to nodes or edges, or something else?

Recall: Data Improving Dependencies

Quality Improving Dependency (QID)

A quality improving dependency (QID) is a first-order sentence of the following form

$$\forall t_1 \forall t_2 \Big(\big(R(t_1) \wedge S(t_2) \wedge \bigwedge_{i \in [1,n]} t_1[A_i] \ \mathrm{op}_i \ t_2[B_i] \big) \rightarrow \bigwedge_{j \in [1,m]} t_1[C_j] \ \mathrm{op}'_j \ t_2[D_j] \Big)$$

where the operators op_i and op'_j form the signature of the dependency

Do QIDs Apply to Graphs?

- Attributes correspond to properties of nodes
- But what about tuples? Do they correspond to nodes or edges, or something else?
- What about the labels of nodes and edges?

Recall: Data Improving Dependencies

Quality Improving Dependency (QID)

A quality improving dependency (QID) is a first-order sentence of the following form

$$\forall t_1 \forall t_2 \Big(\big(R(t_1) \wedge S(t_2) \wedge \bigwedge_{i \in [1,n]} t_1[A_i] \ \mathrm{op}_i \ t_2[B_i] \big) \rightarrow \bigwedge_{j \in [1,m]} t_1[C_j] \ \mathrm{op}'_j \ t_2[D_j] \Big)$$

where the operators op_i and op'_j form the signature of the dependency

Do QIDs Apply to Graphs?

- Attributes correspond to properties of nodes
- But what about tuples? Do they correspond to nodes or edges, or something else?
- What about the labels of nodes and edges?
- Goal: compare nodes in specific subgraphs instead of tuples

Identifying Subgraphs with Graph Patterns

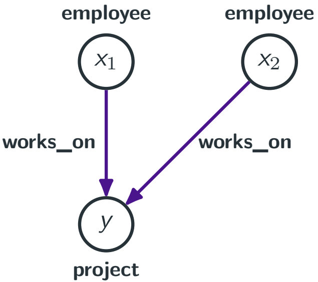
- We use graph patterns to identify subgraphs in our quality dependencies for graphs

Graph Pattern

A graph pattern is a tuple $Q = (X_Q, E_Q, L_Q)$ where

- (X_Q, E_Q) is a directed graph
- the nodes in X_Q are called variables
- L_Q is a function that assigns labels to nodes/variables and edges

Example (Graph Pattern)



All pairs of employees x_1, x_2 working on a common project y

Identifying Subgraphs with Graph Patterns

- We use graph patterns to identify subgraphs in our quality dependencies for graphs

Graph Pattern

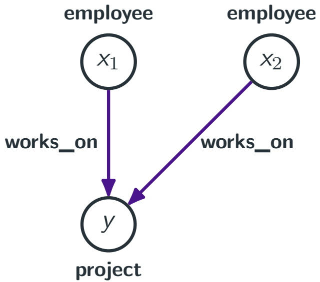
A graph pattern is a tuple $Q = (X_Q, E_Q, L_Q)$ where

- (X_Q, E_Q) is a directed graph
- the nodes in X_Q are called variables
- L_Q is a function that assigns labels to nodes/variables and edges

Note

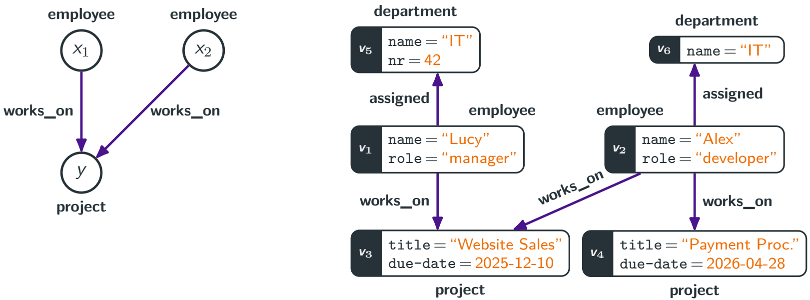
Graph patterns do not refer to properties

Example (Graph Pattern)



All pairs of employees x_1, x_2 working on a common project y

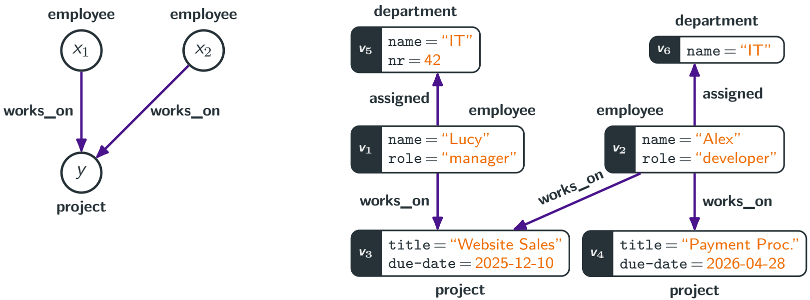
Matching Semantics



Match

A match of a pattern $Q = (X_Q, E_Q, L_Q)$ in a graph $G = (V, E, L, P)$ is a function $h \colon X_Q \to V$ such that

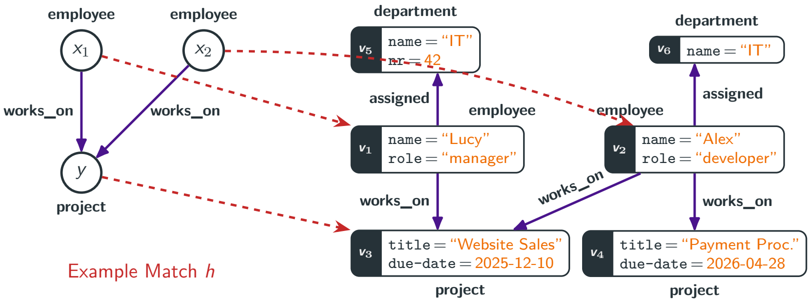
Matching Semantics



Match

A match of a pattern $Q = (X_Q, E_Q, L_Q)$ in a graph $G = (V, E, L, P)$ is a function $h\colon X_Q \to V$ such that

- $L_Q(x) = L(h(x))$ for all $x \in X_Q$
- For all edges $(x, y) \in E_Q$:
  - $(h(x), h(y)) \in E$ is an edge in G
  - $L_Q(x, y) = L(h(x), h(y))$

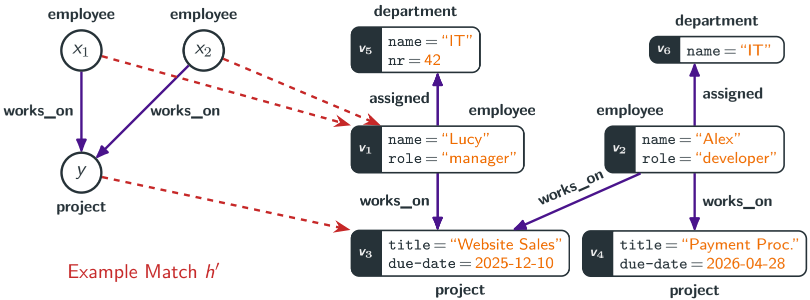


Match

A match of a pattern $Q = (X_Q, E_Q, L_Q)$ in a graph $G = (V, E, L, P)$ is a function $h\colon X_Q \to V$ such that

- $L_Q(x) = L(h(x))$ for all $x \in X_Q$
- For all edges $(x, y) \in E_Q$:
  - $(h(x), h(y)) \in E$ is an edge in G
  - $L_Q(x, y) = L(h(x), h(y))$

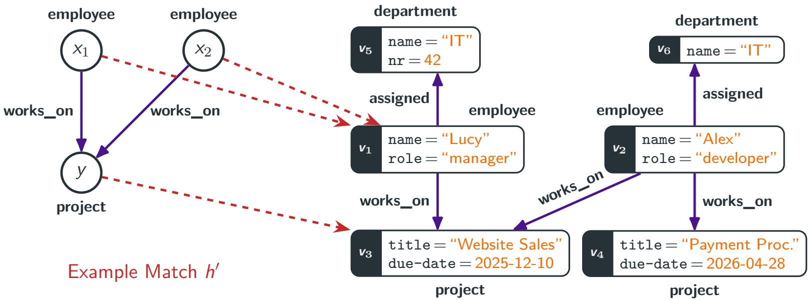
Matching Semantics



Notes

- Multiple variables can be mapped to the same node

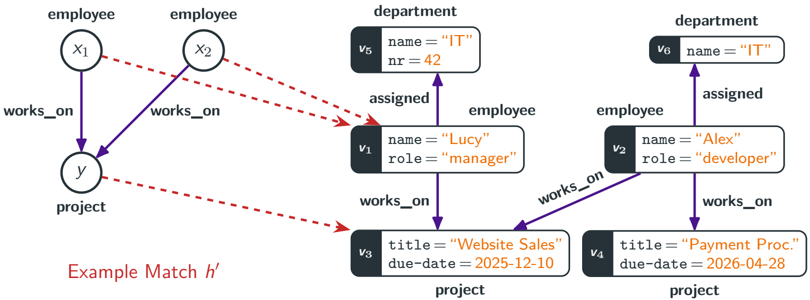
Matching Semantics



Notes

- Multiple variables can be mapped to the same node
- A function h satisfying the definition of match is called a homomorphism
- We thus consider homomorphic matches

Matching Semantics



Notes

- Multiple variables can be mapped to the same node
- A function h satisfying the definition of match is called a homomorphism
- We thus consider homomorphic matches
- There are many alternative matching semantics: Big Graph Processing Systems course

Graph Entity Dependencies[1]

Graph Entity Dependencies (GEDs)

A graph entity dependency (GED) has the following form

$$Q\Big(\bigwedge_{i\in[1,n]} \varphi_i(x_i, y_i) \rightarrow \bigwedge_{j\in[1,m]} \psi_j(z_j, u_j)\Big)$$

- Q is graph pattern
- x_i, y_i, z_j, u_j are variables of Q
- φ and ψ are literals

[1] Fan and Lu, "Dependencies for Graphs", *Proceedings of the 36th ACM Symposium on Principles of Database Systems, PODS 2017*, 2017

Graph Entity Dependencies[1]

Graph Entity Dependencies (GEDs)

A graph entity dependency (GED) has the following form

$$Q\Big(\bigwedge_{i\in[1,n]} \varphi_i(x_i, y_i) \to \bigwedge_{j\in[1,m]} \psi_j(z_j, u_j)\Big)$$

- Q is graph pattern
- x_i, y_i, z_j, u_j are variables of Q
- φ and ψ are literals

Literals

- Property Equality: $x[A] = y[B]$ iff properties A of x and B of y have the same value
- Equality with constant: $x[A] =_c y[B]$ iff properties A of x and B of y have value c
- Node Equality: $x = y$ iff x and y represent the same node

[1] Fan and Lu, “Dependencies for Graphs”, *Proceedings of the 36th ACM Symposium on Principles of Database Systems, PODS 2017*, 2017

Graph Entity Dependencies[1]

Graph Entity Dependencies (GEDs)

A graph entity dependency (GED) has the following form

$$Q\Big(\bigwedge_{i\in[1,n]} \varphi_i(x_i, y_i) \rightarrow \bigwedge_{j\in[1,m]} \psi_j(z_j, u_j)\Big)$$

- Q is graph pattern
- x_i, y_i, z_j, u_j are variables of Q
- φ and ψ are literals

Semantics

A graph G satisfies a GED with graph pattern $Q = (X_Q, E_Q, L_Q)$ if

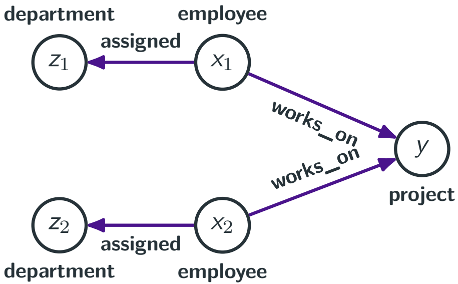
- for every match of h of Q in G:
  - if $(G, h(x_i), h(y_i)) \models \varphi_i(x_i, y_i)$ holds for all $i \in [1, n]$
  - then $(G, h(w_j), h(u_j)) \models \psi_j(x_j, y_j)$ holds for all $i \in [1, m]$

[1] Fan and Lu, "Dependencies for Graphs", *Proceedings of the 36th ACM Symposium on Principles of Database Systems, PODS 2017*, 2017

Graph Entity Dependencies – Example

Example

Graph Pattern Q_1



Graph Entity Dependency

$$Q_1\Big(\text{true} \rightarrow z_1 = z_2\Big)$$

If two employees work on the same project, they are assigned to the same department

Example

Graph Pattern Q_2

Graph Entity Dependency

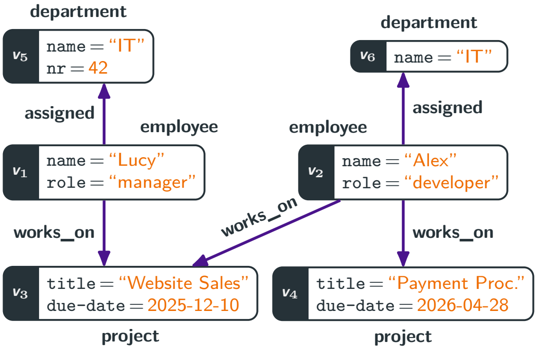
$$Q_2\Big(z_1[\text{name}] = z_1[\text{name}] \rightarrow z_1[\text{nr}] = z_1[\text{nr}]\Big)$$

If two departments have the same name,
they also have the same number

Graph Entity Dependencies – Example

Graph Entity Dependencies

If two employees work on the same project, they are assigned to the same department

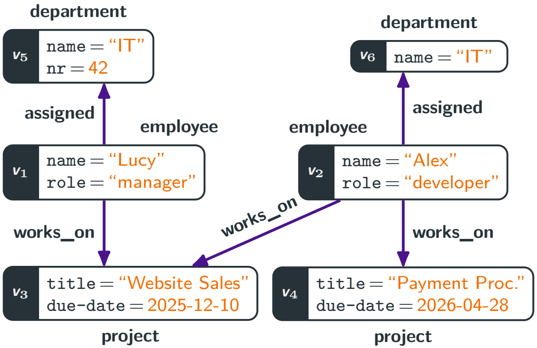


Graph Entity Dependencies – Example

Graph Entity Dependencies

If two employees work on the same project, they are assigned to the same department

- Not satisfied by G ✗



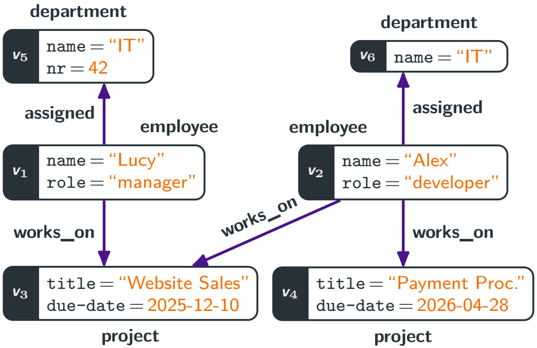
Graph Entity Dependencies – Example

Graph Entity Dependencies

If two employees work on the same project, they are assigned to the same department

- Not satisfied by G ✗

If two departments have the same name, they also have the same number



Graph Entity Dependencies – Example

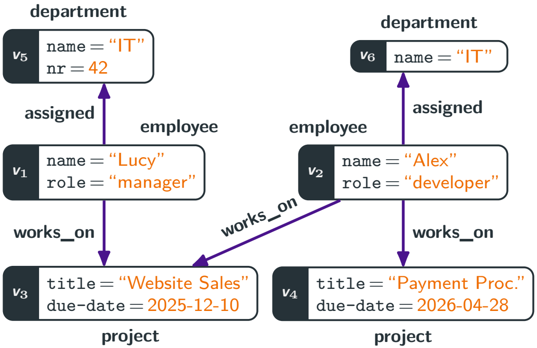
Graph Entity Dependencies

If two employees work on the same project, they are assigned to the same department

- Not satisfied by G ✗

If two departments have the same name,
they also have the same number

- Not satisfied by G ✗



GEDs vs. QIDs

GEDs vs. CFDs

CFDs can be translated into GEDs

- Every tuple in the database is interpreted as a node
  - labelled with the relation it belongs to
- The graph pattern of the GED consists of two disconnect nodes labelled with R and S

GEDs vs. QIDs

GEDs vs. CFDs

CFDs can be translated into GEDs

- Every tuple in the database is interpreted as a node
  - labelled with the relation it belongs to
- The graph pattern of the GED consists of two disconnect nodes labelled with R and S

GEDs vs. MDs

- GEDs do not support similarity operators
- and can therefore not mimic
- But they can be extended accordingly

Ingredients for the Repair Problem

1. Quality dependencies ✔
   - We consider Graph Entity Dependencies
2. A dirty graph database ✔
3. A repair model
   - What kind of operations are allowed to modify the database?
   - Examples: ~~tuple deletions, tuple insertions~~, value modifications
4. A cost model
   - the repair should differ minimally
   - Examples: number of deletions, edit distance

Goal

A clean graph that satisfies all the dependencies

Repair Model

- We discuss a variation of the V-Repair model, adapted for graphs
- It allows for changing values of properties

Repair Model

Repair Model

- We discuss a variation of the V-Repair model, adapted for graphs
- It allows for changing values of properties
- It allows for merging nodes
  - If two nodes v_1, and v_2 are merged into a new node w,
  - then w inherits all outgoing edges from v_1 and v_2,
  - and all incoming edges of v_1 and v_2 are redirected to w

Repair Model

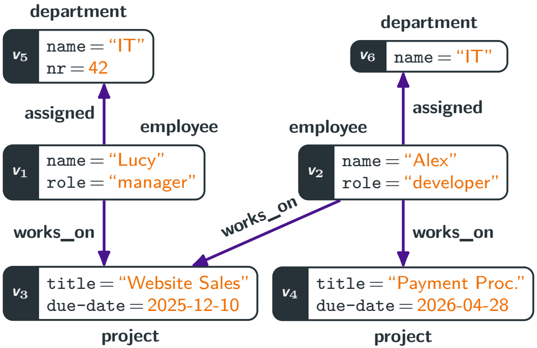
- We discuss a variation of the V-Repair model, adapted for graphs
- It allows for changing values of properties
- It allows for merging nodes
  - If two nodes v_1, and v_2 are merged into a new node w,
  - then w inherits all outgoing edges from v_1 and v_2,
  - and all incoming edges of v_1 and v_2 are redirected to w
- Labels and edges cannot be changed directly

Repair Model – Example

Graph Entity Dependency

> *If two employees work on the same project, they are assigned to the same department*

- Not satisfied by G ✗

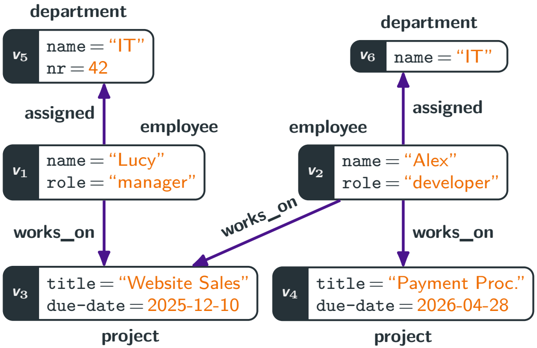


Repair Model – Example

Graph Entity Dependency

If two employees work on the same project, they are assigned to the same department

- Not satisfied by G ✗
- To repair G, we merge v_5 and v_6

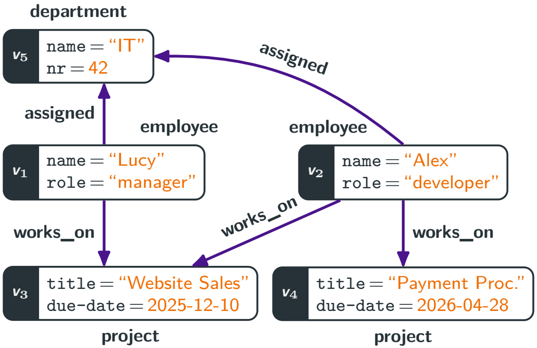


Repair Model – Example

Graph Entity Dependency

If two employees work on the same project, they are assigned to the same department

- Not satisfied by G ✗
- To repair G, we merge v_5 and v_6



Ingredients for the Repair Problem

1. Quality dependencies ✔
   - We consider Graph Entity Dependencies
2. A dirty graph database ✔
3. A repair model ✔
   - Modification of property values, merging of nodes
4. A cost model ✔
   - the repair should differ minimally
   - Examples: number of merges, edit distance

Goal

A clean graph that satisfies all the dependencies

Chasing Graphs

Recall: Chase Procedures

Idea

- We adapt the (extended) chase procedure for graphs and our repair model

Recall: Chase Procedures

The chase takes as input

- a set Σ of data quality rules; and
- an input database D,

and, if the chase terminates successfully, then it outputs a database D' such that $D' \models \Sigma$

Recall: Chase Procedures

Idea

- We adapt the (extended) chase procedure for graphs and our repair model

Recall: Chase Procedures

The chase takes as input

- a set Σ of graph entity dependencies (GEDs); and
- a graph database G,

and, if the chase terminates successfully, then it outputs a clean graph G' satisfying Σ

Recall: Chase Procedures

Idea

- We adapt the (extended) chase procedure for graphs and our repair model

Recall: Chase Procedures

The chase takes as input

- a set Σ of graph entity dependencies (GEDs); and
- a graph database G,

and, if the chase terminates successfully, then it outputs a clean graph G' satisfying Σ

Recall: Implementation

- The chase procedure can fire a dependency σ if σ is not satisfied
- σ is fired for a specific violation, which is then repaired (unless there is a conflict)

The Chase for GEDs

Let

$$\sigma = Q\Big(\bigwedge_{i \in [1,n]} \varphi_i(x_i, y_i) \rightarrow \psi(z, u) \Big)$$

be a graph entity dependency (GED) with graph pattern $Q = (X_Q, E_Q, L_Q)$

The Chase for GEDs

Let

$$\sigma = Q\Big(\bigwedge_{i \in [1,n]} \varphi_i(x_i, y_i) \rightarrow \psi(z, u)\Big)$$

be a graph entity dependency (GED) with graph pattern $Q = (X_Q, E_Q, L_Q)$

Firing of a GED

The GED σ can be fired on a graph G if there is a match h of Q in G such that

- $(G, h(x_i), h(y_i)) \models \varphi_i$ holds for all $i \in [1, n]$
- but $(G, h(z), h(u)) \models \psi(z, u)$ does not hold

The Chase for GEDs

The Chase Procedure for GEDs

1. Initialize $G' = G$
2. As long as there is
   - a GED $\sigma \in \Sigma$ with consequence $\psi(z, u)$,
   - and a match h

   for which σ can be fired do

The Chase for GEDs

The Chase Procedure for GEDs

1. Initialize $G' = G$
2. As long as there is
   - a GED $\sigma \in \Sigma$ with consequence $\psi(z, u)$,
   - and a match h

   for which σ can be fired do

   2.1 If $\psi(z, u)$ has the form $z[A] = u[B]$ and property A of $h(z)$ exists but property B of $h(u)$ does not, create property B for $h(u)$ and set its value to that of A of $h(z)$

The Chase for GEDs

The Chase Procedure for GEDs

1. Initialize $G' = G$
2. As long as there is
   - a GED $\sigma \in \Sigma$ with consequence $\psi(z, u)$,
   - and a match h

   for which σ can be fired do

   2.1 If $\psi(z, u)$ has the form $z[A] = u[B]$ and property A of $h(z)$ exists but property B of $h(u)$ does not, create property B for $h(u)$ and set its value to that of A of $h(z)$

   2.2 If $\psi(z, u)$ has the form $z[A] = u[B]$ and property A of $h(z)$ exists but property B of $h(u)$ does not, create property B for $h(u)$ and set its value to that of A of $h(z)$

The Chase for GEDs

The Chase Procedure for GEDs

1. Initialize $G' = G$
2. As long as there is
   - a GED $\sigma \in \Sigma$ with consequence $\psi(z, u)$,
   - and a match h

   for which σ can be fired do

   2.1 If $\psi(z, u)$ has the form $z[A] = u[B]$ and property A of $h(z)$ exists but property B of $h(u)$ does not, create property B for $h(u)$ and set its value to that of A of $h(z)$

   2.2 If $\psi(z, u)$ has the form $z[A] = u[B]$ and property A of $h(z)$ exists but property B of $h(u)$ does not, create property B for $h(u)$ and set its value to that of A of $h(z)$

   2.3 If $\psi(z, u)$ has the form $z[A] =_c u[B]$ proceed analogously to cases 1 and 2 but set the value to c and abort if the existing property does not have value c

The Chase for GEDs

The Chase Procedure for GEDs

1. Initialize $G' = G$
2. As long as there is
   - a GED $\sigma \in \Sigma$ with consequence $\psi(z, u)$,
   - and a match h

   for which σ can be fired do

   2.1 If $\psi(z, u)$ has the form $z[A] = u[B]$ and property A of $h(z)$ exists but property B of $h(u)$ does not, create property B for $h(u)$ and set its value to that of A of $h(z)$

   2.2 If $\psi(z, u)$ has the form $z[A] = u[B]$ and property A of $h(z)$ exists but property B of $h(u)$ does not, create property B for $h(u)$ and set its value to that of A of $h(z)$

   2.3 If $\psi(z, u)$ has the form $z[A] =_c u[B]$ proceed analogously to cases 1 and 2 but set the value to c and abort if the existing property does not have value c

   2.4 If $\psi(z, u)$ has the form $z = u$ and the nodes $h(z)$ and $h(u)$ agree on all common properties and have the same label, merge $h(z)$ and $h(u)$
   - The new node inherits all properties of $h(z)$ and $h(u)$

The Chase for GEDs

The Chase Procedure for GEDs

1. Initialize $G' = G$
2. As long as there is
   - a GED $\sigma \in \Sigma$ with consequence $\psi(z, u)$,
   - and a match h

   for which σ can be fired do

   2.1 If $\psi(z, u)$ has the form $z[A] = u[B]$ and property A of $h(z)$ exists but property B of $h(u)$ does not, create property B for $h(u)$ and set its value to that of A of $h(z)$

   2.2 If $\psi(z, u)$ has the form $z[A] = u[B]$ and property A of $h(z)$ exists but property B of $h(u)$ does not, create property B for $h(u)$ and set its value to that of A of $h(z)$

   2.3 If $\psi(z, u)$ has the form $z[A] =_c u[B]$ proceed analogously to cases 1 and 2 but set the value to c and abort if the existing property does not have value c

   2.4 If $\psi(z, u)$ has the form $z = u$ and the nodes $h(z)$ and $h(u)$ agree on all common properties and have the same label, merge $h(z)$ and $h(u)$
   - The new node inherits all properties of $h(z)$ and $h(u)$

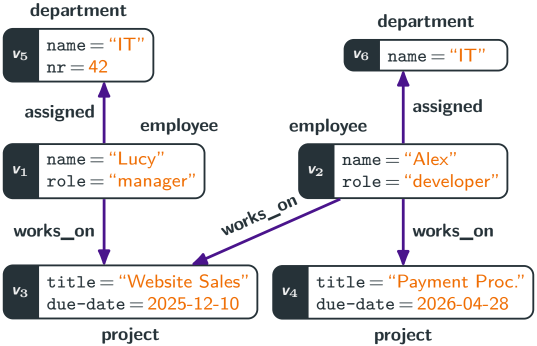
   2.5 If none of the other cases applies, abort

The Chase for GEDs – Example

Graph Entity Dependencies

- GED σ_1:
  If two departments have the same name, they also have the same number

- GED σ_2:
  If two employees work on the same project, they are assigned to the same department



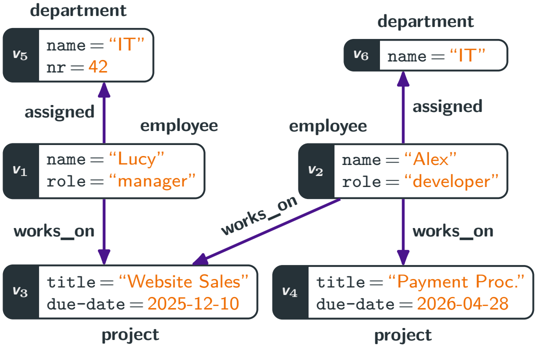
Graph Entity Dependencies

- GED σ_1:
  If two departments have the same name, they also have the same number

- GED σ_2:
  If two employees work on the same project, they are assigned to the same department

The Chase

1. Fire σ_1 with a match for v_5 and v_6
   - Set $v_6[\text{nr}] := v_5[\text{nr}]$

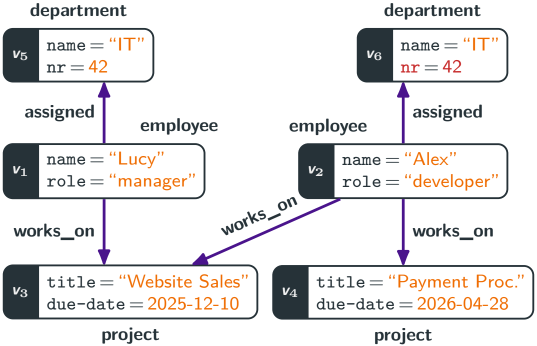


The Chase for GEDs – Example

Graph Entity Dependencies

- GED σ_1:
  If two departments have the same name, they also have the same number

- GED σ_2:
  If two employees work on the same project, they are assigned to the same department



The Chase

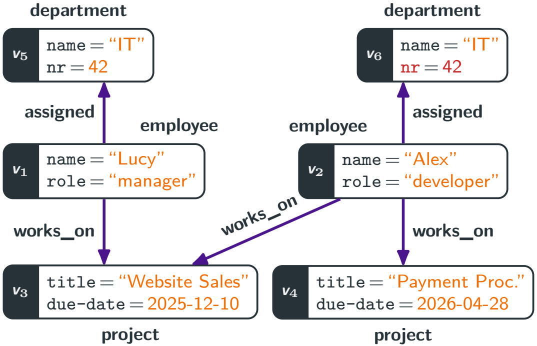
1. Fire σ_1 with a match for v_5 and v_6
   - Set $v_6[\text{nr}] := v_5[\text{nr}]$

The Chase for GEDs – Example

Graph Entity Dependencies

- GED σ_1:
  If two departments have the same name, they also have the same number

- GED σ_2:
  If two employees work on the same project, they are assigned to the same department



The Chase

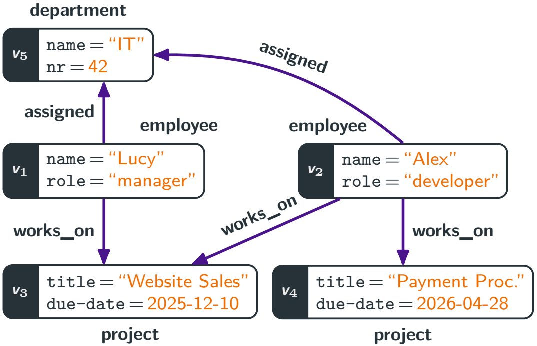
1. Fire σ_1 with a match for v_5 and v_6
   - Set $v_6[\text{nr}] := v_5[\text{nr}]$
2. Fire σ_2 with a match for v_5, v_6, v_1, v_2, v_3
   - Merge v_5 and v_6

The Chase for GEDs – Example

Graph Entity Dependencies

- GED σ_1:
  If two departments have the same name, they also have the same number

- GED σ_2:
  If two employees work on the same project, they are assigned to the same department



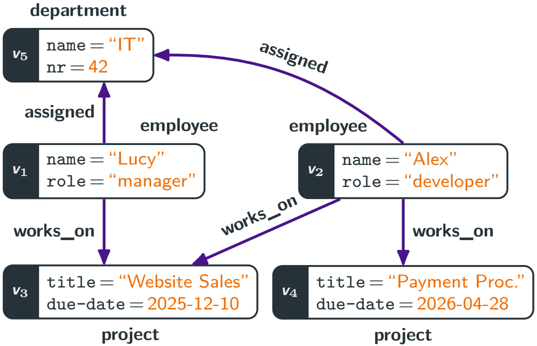
The Chase

1. Fire σ_1 with a match for v_5 and v_6
   - Set $v_6[\mathrm{nr}] := v_5[\mathrm{nr}]$
2. Fire σ_2 with a match for v_5, v_6, v_1, v_2, v_3
   - Merge v_5 and v_6

The Chase for GEDs – Example

Graph Entity Dependencies

- GED σ_1:
  If two departments have the same name, they also have the same number
- GED σ_2:
  If two employees work on the same project, they are assigned to the same department



The Chase

1. Fire σ_1 with a match for v_5 and v_6
   - Set $v_6[\text{nr}] := v_5[\text{nr}]$
2. Fire σ_2 with a match for v_5, v_6, v_1, v_2, v_3
   - Merge v_5 and v_6
3. No GED can be fired
   - The chase terminates successfully

Properties of the Chase for Graphs and GEDs

The chase procedure for graphs and GEDs has two important properties

Properties of the Chase for Graphs and GEDs

The chase procedure for graphs and GEDs has two important properties

Chasing with GEDs is Finite

The chase always terminates after a finite number of steps

Properties of the Chase for Graphs and GEDs

The chase procedure for graphs and GEDs has two important properties

Chasing with GEDs is Finite

The chase always terminates after a finite number of steps

Chasing with GEDs has the Church-Rosser Property

For all graphs G, and a sets Σ of GEDs, the chase either

- always returns the same repair for G and Σ; or
- always aborts due to a conflict

regardless in which order and for which errors GEDs are fired

Reasoning about GEDs

Reasoning about GEDs: The Satisfiability Problem

The Satisfiability Problem

Input: A finite set Σ of GEDs

Question: Is there a non-trivial graph G that satisfies all dependencies in Σ?

Goal

Automatically Verifying that a set of GEDs is consistent

Reasoning about GEDs: The Satisfiability Problem

The Satisfiability Problem

Input: A finite set Σ of GEDs
Question: Is there a non-trivial graph G that satisfies all dependencies in Σ?

Goal

Automatically Verifying that a set of GEDs is consistent

Theorem (Fan and Lu, "Dependencies for Graphs", Theorem 5.4)

The satisfiability problem for GEDs is coNP-complete

- For CFDs in the relational setting, satisfiability is NP-complete

Reasoning about GEDs: The Satisfiability Problem

The Satisfiability Problem

Input: A finite set Σ of GEDs
Question: Is there a non-trivial graph G that satisfies all dependencies in Σ?

Goal

Automatically Verifying that a set of GEDs is consistent

Theorem (Fan and Lu, "Dependencies for Graphs", Theorem 5.4)

The satisfiability problem for GEDs is coNP-complete

- For CFDs in the relational setting, satisfiability is NP-complete
- But in PTIME, if the relational schema does not enforce finite domains for attributes!

Reasoning about GEDs: The Satisfiability Problem

The Satisfiability Problem

Input: A finite set Σ of GEDs
Question: Is there a non-trivial graph G that satisfies all dependencies in Σ?

Goal

Automatically Verifying that a set of GEDs is consistent

Theorem (Fan and Lu, "Dependencies for Graphs", Theorem 5.4)

The satisfiability problem for GEDs is coNP-complete

- For CFDs in the relational setting, satisfiability is NP-complete
- But in PTIME, if the relational schema does not enforce finite domains for attributes!
- Since our graphs have no schema, the difficulty is not inherited from CFDs

The Implication Problem

Input: A finite set Σ of GEDs, a GED σ

Question: Does Σ imply σ?

Reasoning about GEDs: The Implication Problem

The Implication Problem

Input: A finite set Σ of GEDs, a GED σ

Question: Does Σ imply σ?

Goals

Solving the implication problem allows for

- reducing the number of dependencies
- inferring new knowledge

Reasoning about GEDs: The Implication Problem

The Implication Problem

Input: A finite set Σ of GEDs, a GED σ

Question: Does Σ imply σ?

Goals

Solving the implication problem allows for

- reducing the number of dependencies
- inferring new knowledge

Theorem (Fan and Lu, "Dependencies for Graphs", Theorem 5.12)

The implication problem for GEDs is NP-complete

- For CFDs in the relational setting, the implication problem is NP-complete

Reasoning about GEDs: The Error Detection Problem

The Error Detection Problem

Input: A finite set Σ of GEDs, a graph G

Question: Does G not satisfy Σ?

The Error Detection Problem

Input: A finite set Σ of GEDs, a graph G

Question: Does G not satisfy Σ?

Goal

Verify whether a graph satisfies dependencies, or needs to be repaired

Reasoning about GEDs: The Error Detection Problem

The Error Detection Problem

Input: A finite set Σ of GEDs, a graph G

Question: Does G not satisfy Σ?

Goal

Verify whether a graph satisfies dependencies, or needs to be repaired

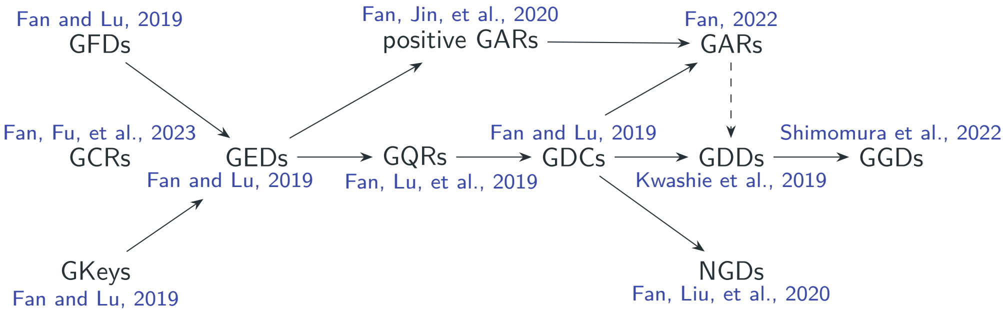
Theorem (Fan and Lu, "Dependencies for Graphs", Theorem 5.16)

The error detection problem for GEDs is NP-complete, even if G is a tree

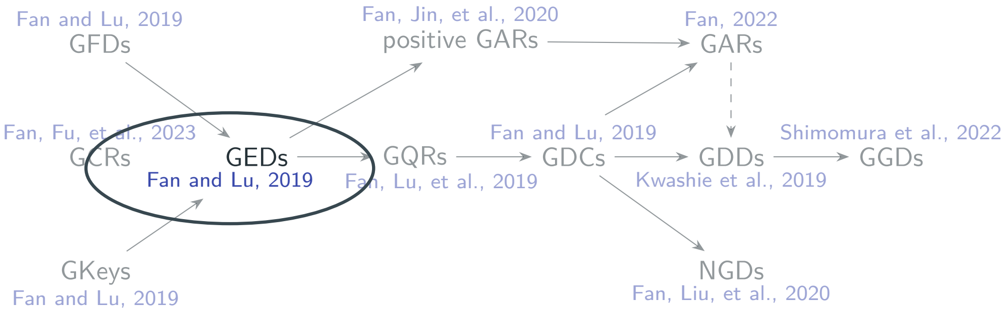
- For CFDs in the relational setting, the error detection problem is in PTIME

The Frontier of Graph Repairs

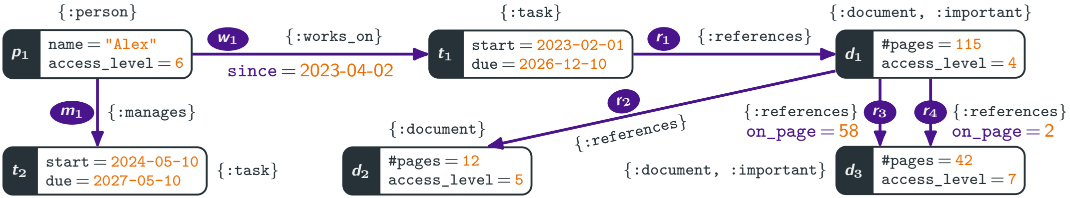
A Family of Graph Dependencies



A Family of Graph Dependencies



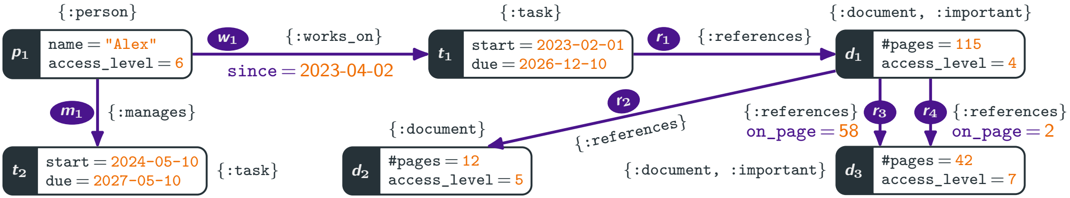
Outlook: Property Graphs



Property Graphs

- Nodes and edges can have multiple labels
- Multi-graphs: there can be more than edge between two nodes
- Nodes and edges can have properties

Outlook: Property Graphs

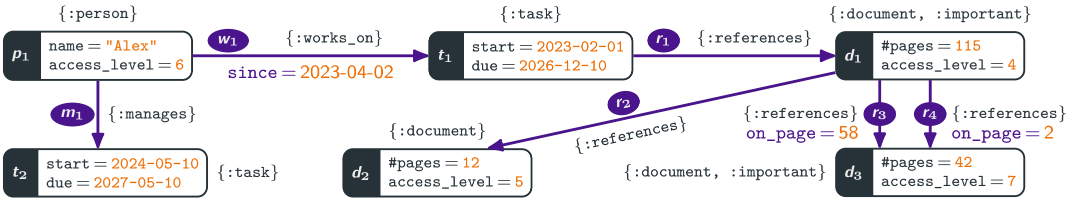


Property Graphs

- Nodes and edges can have multiple labels
- Multi-graphs: there can be more than edge between two nodes
- Nodes and edges can have properties

- Part of the ISO standard for GQL
  - ISO/IEC 39075:2024
  - GQL is a query language for property graphs
  - Published in April 2024

Outlook: Property Graphs



Property Graphs

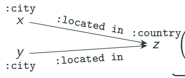
- Nodes and edges can have multiple labels
- Multi-graphs: there can be more than edge between two nodes
- Nodes and edges can have properties

- Part of the ISO standard for GQL
  - ISO/IEC 39075:2024
  - GQL is a query language for property graphs
  - Published in April 2024
- More in the “Big Graph Processing Systems” course in January/February 2026

Outlook: GQL and PG-Constraints

GDCs

and related notions



$$\left(x \neq y \quad \Rightarrow \quad x.\texttt{name} \neq y.\texttt{name}\right)$$

Pattern with variables $x, y, z, \ldots$

Dependency $X \Rightarrow Y$

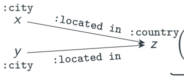
PG-Constraints

```
FOR x, y, z
MATCH (x:city)-[:located in]->(z:country),
      (y:city)-[:located in]->(z)
FILTER x != y
MANDATORY x.name, y.name
FILTER x.name != y.name
```

Outlook: GQL and PG-Constraints

GDCs

and related notions



$\left(x \neq y \Rightarrow x.\mathrm{name} \neq y.\mathrm{name}\right)$

Pattern with variables $x, y, z, \ldots$ — Dependency $X \Rightarrow Y$

- ✔ The satisfiability, validation, and implication problems have been studied[2]
- ✔ Data cleaning has been studied[3] for GDCs without $\leq$ and $\geq$
- ✗ Not a "good" fit for property graphs
  - Single labels, no transitivity, etc.

PG-Constraints

```
FOR x, y, z
MATCH (x:city)-[:located in]->(z:country),
      (y:city)-[:located in]->(z)
FILTER x != y
MANDATORY x.name, y.name
FILTER x.name != y.name
```

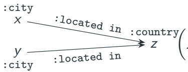
[3] Fan and Lu, "Dependencies for Graphs", *ACM Transactions on Database Systems*, 2019.

[3] Fan, Lu, et al., "Deducing Certain Fixes to Graphs", *Proceedings of the VLDB Endowment*, 2019.

Outlook: GQL and PG-Constraints

GDCs

and related notions



$$\left(x \neq y \quad \Rightarrow \quad x.\mathrm{name} \neq y.\mathrm{name}\right)$$

Pattern with variables $x, y, z, \ldots$

Dependency $X \Rightarrow Y$

- ✔ The satisfiability, validation, and implication problems have been studied[2]
- ✔ Data cleaning has been studied[3] for GDCs without $\leq$ and $\geq$
- ✗ Not a “good” fit for property graphs
  - Single labels, no transitivity, etc.

PG-Constraints

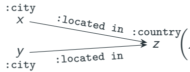
```
FOR x, y, z
MATCH (x:city)-[:located in]->(z:country),
      (y:city)-[:located in]->(z)
FILTER x != y
MANDATORY x.name, y.name
FILTER x.name != y.name
```

- ✔ Designed for property graphs
- ● Expressive and extensive
  - since they are based on GQL
- ✗ No formal results

[3] Fan and Lu, “Dependencies for Graphs”, *ACM Transactions on Database Systems*, 2019.
[3] Fan, Lu, et al., “Deducing Certain Fixes to Graphs”, *Proceedings of the VLDB Endowment*, 2019.

Outlook: GQL and PG-Constraints

GDCs
and related notions



$\left(x \neq y \quad \Rightarrow \quad x.\mathtt{name} \neq y.\mathtt{name}\right)$

Pattern with variables $x, y, z, \ldots$ — Dependency $X \Rightarrow Y$

- ✔ The satisfiability, validation, and implication problems have been studied[2]
- ✔ Data cleaning has been studied[3] for GDCs without $\leq$ and $\geq$
- ✗ Not a “good” fit for property graphs
  - Single labels, no transitivity, etc.

PG-Constraints

```
FOR x, y, z
MATCH (x:city)-[:located in]->(z:country),
      (y:city)-[:located in]->(z)
FILTER x != y
MANDATORY x.name, y.name
FILTER x.name != y.name
```

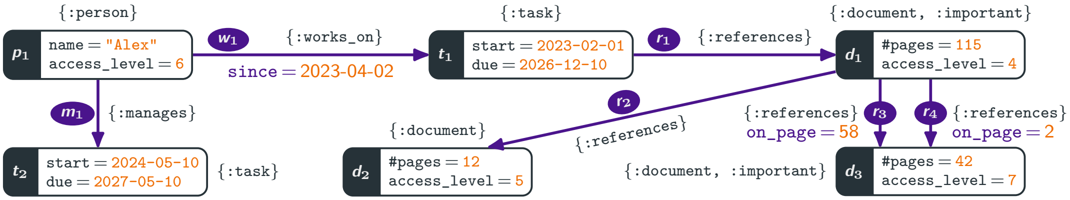
- ✔ Designed for property graphs
- ● Expressive and extensive
  - since they are based on GQL
- ✗ No formal results

---

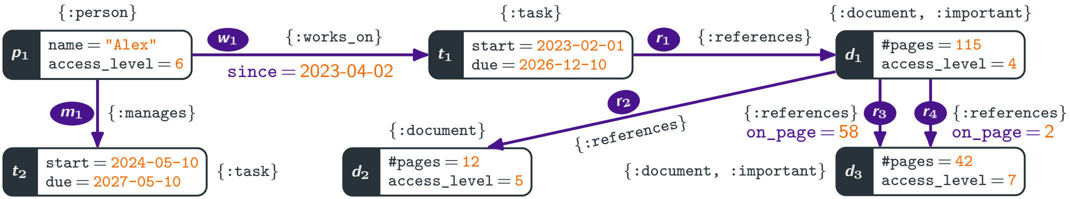
[3] Fan and Lu, “Dependencies for Graphs”, *ACM Transactions on Database Systems*, 2019.

[3] Fan, Lu, et al., “Deducing Certain Fixes to Graphs”, *Proceedings of the VLDB Endowment*, 2019.

Dependencies for Path Constraints



Dependencies for Path Constraints



Constraint
If a person works on a task, which has started and which references directly or indirectly, an important document, then the person's access level is at least as high as the (required) access level of the referenced document.

- This constraint cannot be described by a GED, since it talks about arbitrary length paths

Summary

Data Quality for Graphs

- Graph entity dependencies (GEDs) are a formalism for data quality rules for graphs
  - They cover CFDs from the relational setting (and more)
- Graph patterns are used to identify subgraphs
- The chase can be adapted for GEDs
- Research in this area is still ongoing!

Summary

Data Quality for Graphs

- Graph entity dependencies (GEDs) are a formalism for data quality rules for graphs
  - They cover CFDs from the relational setting (and more)
- Graph patterns are used to identify subgraphs
- The chase can be adapted for GEDs
- Research in this area is still ongoing!

Take away message

Data quality for graphs: a rich source of problems and challenges

References

Fan, Wenfei. “Big Graphs: Challenges and Opportunities”. In: *Proceedings of the VLDB Endowment* 15.12 (2022), pp. 3782–3797. DOI: 10.14778/3554821.3554899.

Fan, Wenfei, Wenzhi Fu, Ruochun Jin, Muyang Liu, Ping Lu, and Chao Tian. “Making It Tractable to Catch Duplicates and Conflicts in Graphs”. In: *Proceedings of the ACM on Management of Data* 1.1 (May 26, 2023), pp. 1–28. DOI: 10.1145/3588940.

Fan, Wenfei, Ruochun Jin, Muyang Liu, Ping Lu, Chao Tian, and Jingren Zhou. “Capturing Associations in Graphs”. In: *Proceedings of the VLDB Endowment* 13.11 (2020), pp. 1863–1876. URL: http://www.vldb.org/pvldb/vol13/p1863-fan.pdf.

Fan, Wenfei, Xueli Liu, Ping Lu, and Chao Tian. “Catching Numeric Inconsistencies in Graphs”. In: *ACM Transactions on Database Systems* 45.2 (June 30, 2020), pp. 1–47. DOI: 10.1145/3385031.

Fan, Wenfei and Ping Lu. “Dependencies for Graphs”. In: *Proceedings of the 36th ACM Symposium on Principles of Database Systems, PODS 2017*. Ed. by Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts. ACM, 2017, pp. 403–416. DOI: 10.1145/3034786.3056114. URL: https://doi.org/10.1145/3034786.3056114.

Fan, Wenfei and Ping Lu. "Dependencies for Graphs". In: *ACM Transactions on Database Systems* (Feb. 13, 2019), pp. 1–40. DOI: 10.1145/3287285.

Fan, Wenfei, Ping Lu, Chao Tian, and Jingren Zhou. "Deducing Certain Fixes to Graphs". In: *Proceedings of the VLDB Endowment* 12.7 (Mar. 2019), pp. 752–765. DOI: 10.14778/3317315.3317318.

Kwashie, Selasi, Lin Liu, Jixue Liu, Markus Stumptner, Jiuyong Li, and Lujing Yang. "*Certus*: An Effective Entity Resolution Approach with Graph Differential Dependencies (GDDs)". In: *Proceedings of the VLDB Endowment* 12.6 (Feb. 2019), pp. 653–666. DOI: 10.14778/3311880.3311883.

Shimomura, Larissa C., Nikolay Yakovets, and George Fletcher. "Reasoning on Property Graphs with Graph Generating Dependencies". In: (2022). DOI: 10.48550/ARXIV.2211.00387. URL: https://arxiv.org/abs/2211.00387 (visited on 03/28/2024).