Schema and Constraints

Mats Rydberg
mats@neotechnology.com
Schema in Cypher

- Cypher is *schema-optional*
  - Fits well with heterogenous data
  - Makes typing and query planning harder
  - Does not fit well with many existing table-based engines
- Constraints are the only tools to enforce structure in the data

Schema in Cypher is a point where we expect there to be major developments as more actors get involved.
New constraint syntax

- Consistent syntax for all types of constraints (CIP)
- Re-use as much as possible from the rest of the language
- Allow for a large set of future constraints, some of which are vendor-specific
  - This allows vendors to use more strict schema when necessary

```cypher
CREATE CONSTRAINT <name>
FOR <simple pattern>
REQUIRE <constraint expression>
```
Constraints, examples

- Property uniqueness constraint
  
  ```cypher
  CREATE CONSTRAINT unique_person_names
  FOR (p:Person)
  REQUIRE UNIQUE p.firstName, p.lastName
  ```

- Property existence constraint
  
  ```cypher
  CREATE CONSTRAINT person_must_have_firstName
  FOR (p:Person)
  REQUIRE exists(p.firstName)
  ```
Constraints, examples

- Property value constraint
  - CREATE CONSTRAINT roads_must_have_positive_finite_length
    FOR ()-[r:ROAD]-()
    REQUIRE 0 < r.distance < infinity

- Property type constraint
  - CREATE CONSTRAINT people_schema
    FOR (p:Person)
    REQUIRE p.email IS STRING
    REQUIRE p.name IS STRING?
    REQUIRE p.age IS INTEGER?
Constraints, examples

- Cardinality constraints
  - CREATE CONSTRAINT spread_the_love
    FOR (p:Person)
    REQUIRE size((p)-[:LOVES]->()) > 3

- Endpoint constraints
  - CREATE CONSTRAINT can_only_own_things
    FOR ()-[:OWNS]->(t)
    REQUIRE (t:Vehicle) OR (t:Building) OR (t:Object)

- Label coexistence constraints
  - CREATE CONSTRAINT programmers_are_people_too
    FOR (p:Programmer)
    REQUIRE p:Person
That's it!

Questions?
Subqueries

Petra Selmer
petra.selmer@neotechnology.com
Existential subqueries

MATCH (actor:Actor)
WHERE EXISTS {
  (actor)-[:ACTED_IN]->(movie),
  (other:Actor)-[:ACTED_IN]->(movie)
WHERE other.name = actor.name
  AND actor <> other
}
RETURN actor
Nested correlated subqueries

MATCH (f:Farm {id: $farmId})-[[:IS_IN]]->(country:Country)
MATCH {
    MATCH (u:User {id: $userId})-[[:LIKES]]->(b:Brand)
    RETURN b AS item, b.designedDate AS dateOfInterest
    UNION
    MATCH (u:User {id: $userId})-[[:BOUGHT]]->(v:Vehicle)
    WHERE v.leftHandDrive = country.leftHandDrive
    RETURN v AS item, v.manufacturedDate AS dateOfInterest
}
RETURN DISTINCT v.code
ORDER BY dateOfInterest
Nested read/write subqueries

MATCH (r:Root)
DO {
    UNWIND range(1, 10) AS x
    MERGE (c:Child {id: x})
    MERGE (r)-[:PARENT]->(c)
}

- **UNWIND** manages the looping
- **DO** suppresses the increased cardinality from the inner query
- **DO** hides all new variable bindings (e.g. `c`)
Scalar subqueries

MATCH (actor:Actor)
WHERE actor.age > {
    MATCH (director:Director)-[:DIRECTED]->(:Movie)
    RETURN max(director.age)
}
RETURN actor.name, actor.details
That's it!

Questions?
Homomorphic and Isomorphic Pattern Matching

Stefan Plantikow

stefan.plantikow@neotechnology.com
Basic Pattern Matching Semantics

• MATCH \((a) - [r] -\to (b)\)
  \[\equiv \{ (a, r, b) \in V \times E \times V \mid r^\to(a,b) \}\]

• MATCH \((a) - [r_1] -\to (b) - [r_2] -\to (c)\)
  \[\equiv \{ (a, r_1, b, r_2, c) \in V \times E \times V \times E \times V \mid r_1^\to(a,b) \neq r_2^\to(b,c) \}\]

• MATCH \((a) - [r_*] -\to (b)\)
  \[\equiv \{ (a, r, b) \in V \times E^* \times V \mid r_*^\to(a,b) \neq r_1 \neq r_2 \}\]
Default Uniqueness = Relationship Uniqueness

• Why?
  • No uniqueness  Potentially infinitely many matching paths
  • Node uniqueness  Can't match self-loops

• When?
  • All pattern variables in a MATCH
  • And similar constructs (comprehensions etc.)

• Optional?
  • Not for variable length
Let's fix this: Requirements

- **CIR 2017-174**
  - Defines uniqueness scope precisely
  - Add configurable uniqueness
    - No uniqueness => Graph homomorphism
    - Relationship uniqueness => Graph (relationship) isomorphism
    - Node uniqueness => Graph (node) isomorphism
  - Default uniqueness
  - Cardinality & path matching semantics
Let's fix this: Proposal

- CIP 2017-01-18
  - Uniqueness mode specifiers
    - MATCH ALL $(a)-[r^*]->(b)$
    - MATCH UNIQUE RELATIONSHIPS
    - MATCH UNIQUE NODES
  - Subquery uniqueness mode specifiers
    - MATCH [MODE] { ... }
    - DO [MODE] { ... }
Let's fix this: Proposal

- **CIP 2017-01-18: Path predicates**
  - **MATCH ALL** \( p=()-[:T1*]->()-[r:T2]->() \) WHERE ...
    - node-isomorphic \( \text{nodeUnique}(p) \)
    - relationship-isomorphic \( \text{relUnique}(p) \)
    - has cycles | has no cycles \( \text{open}(p) | \text{closed}(p) \)
    - no immediately repeated rels \( \text{trek}(p) \)
    - no repeated cycles \( \text{NOT} \ \text{redundant}(p) \)
    - ...
That's it!

Questions ?
(Conjunctive) Regular Path Queries

“Regular expressions over Graphs”

Tobias Lindaaker
tobias@neotechnology.com
We’ve been wanting to do this for long

Been trying to find well balanced syntax
Regular Path Patterns

• Disjunctions (unions) between patterns: (a)-/alt1 | alt2 | alt3/- (b)
• Sequence of patterns: (a)-/first second third/- (b)
• Repetition of pattern: (a)<-/(many times)*/- (b)
• Definition of complex pattern predicates:
  MATCH (a)-/complex/- (b)
  PATH (x)-/complex/- (y) IS
    (x)-[:LOVES]->(y), (y)-[:HATES]->(x)
• Shorthand for simple predicates: (a)-/:REGULAR_RELATIONSHIP_TYPE/- (b)
• Combinations of the above
Origins of the syntax in CIP2017-02-06

“Canterbury Style”
(from a meeting in Canterbury, summer 2015)

- In-line pattern expression

MATCH p=(a:Person{a})[
  (x)-[:AUTHORED]->(:Book)<-[:AUTHORED]-(y)
  WHERE x <> y
]*(b:Person{b})
RETURN p ORDER BY length(p) ASC LIMIT 1

Pros
- Pattern visible where used

Cons
- Large patterns, hard to overview

“PGQL Style”
(inspired by the path patterns in PGQL)

- Composition of simple pattern declarations

PATH CO_AUTHORED :=
  (x)-[:AUTHORED]->(:Book)<-[:AUTHORED]-(y)
  WHERE x <> y
MATCH p=(a:Person{a})-/CO_AUTHORED*/-(b:Person{b})
RETURN p ORDER BY length(p) ASC LIMIT 1

Pros
- No nested syntax, each pattern is clear
- Allows reuse of complex sub-pattern
- Pattern definitions align with “path-views”

Cons
- Definition removed from use
- Everything needs to be declared to be used
- Unclear which nodes are input, and which are internal to a PATH pattern
Syntax tradeoffs

Desirable

• Pattern visible where used
• No unnecessary ceremony
• Transparent complexity
• Familiar regular language
• Allow reuse of complex sub-patterns

Undesirable

• Lots of details in patterns - hard to view structure
• Definition of pattern removed from use
• Very many ways to express the same thing
Evolving from Cypher today

• Limited support today:
  • Disjunction on edge label: ( )-[[:A|B|C]->( )
  • Repetition of single edge: ( )-[[:X*]->( )
    • unhelpful when binding: ( )-[x:X*]->( ) - binds as list!
  • (these two forms can be combined)

• Regular Path Patterns
  replace existing repetition syntax
• For CRPQs (Conjunctive Regular Path Queries)
  Cypher need only add Regular Path Patterns,
  it already has conjunctions
Expressive power

Comparing to GXPath (path expressions)

- $[[\epsilon]]^G = \{(v,v) \mid v \in V\}$ - yes: PATH (a)-/nop/- (a) IS (a)
- $[[\_]]^G = \{(v,w) \mid (v,a,w) \in E \text{ for some } a\} - yes: \text{PATH (a)-/any/-}(b) \text{ IS (a)-/}(b)\$
- $[[a]]^G = \{(v,w) \mid (v,a,w) \in E\} - yes: \text{PATH (a)-/}(a)/-\text{a/-(w)}$
- $[[a^*]]^G = \{(v,w) \mid (w,a,v) \in E\} - yes: \text{PATH (a)-/}(a)/-\lnot a/-(w)$
- $[[\alpha \cdot \beta]]^G = [[\alpha]]^G \sqcap [[\beta]]^G - yes: \text{-/}(\alpha \beta)/-(\text{)}$
- $[[\alpha \cup \beta]]^G = [[\alpha]]^G \cup [[\beta]]^G - yes: \text{-/}(\alpha \cup \beta)/-(\text{)}$
- $[[\neg \alpha]]^G = V \sqcap V - [[\alpha]]^G - maybe:\text{PATH (v)-/not_alpha/-(w) IS (v), (w)}$
  WHERE NOT EXISTS \{ (v)-/\alpha/-\text{(w)} \} \}

Warning: the above is intractable, we might want to restrict to connected patterns

- $[[\phi]]^G = \{(v,v) \mid v \in [[\phi]]^G\}$
  where $\phi$
- $[[\alpha^{n,m}]]^G = \bigcup_{k=n}^{m}([[\alpha]]^G)^k - yes: \text{-/}(\alpha^{n..m})/-(\text{)}$
- $[[\alpha_\rho]]^G = \{(v,w) \in [[\alpha]]^G \mid \rho(v)=\rho(w)\}$
  - yes: PATH (v)-/alpha_eq/-\text{(w)} IS (v)-/\alpha/-\text{(w)} WHERE $v.\rho = w.\rho$
- $[[\alpha_\neq]]^G = \{(v,w) \in [[\alpha]]^G \mid \rho(v) \neq \rho(w)\}$
  - yes: PATH (v)-/alpha_not_eq/-\text{(w)} IS (v)-/\alpha/-\text{(w)} WHERE $v.\rho <> w.\rho$

Conjunctions (for CRPQs):

- $[[\alpha \cap \beta]]^G = [[\alpha]]^G \cap [[\beta]]^G$
  - yes: PATH (v)-/alpha_and_beta/-\text{(w)} IS (v)-/\alpha/-\text{(w)}, (v)-/\beta/-\text{(w)}$
Expressive power

Comparing to GXPath (node tests)

- $[[\alpha]]^G = \{v \mid \exists w (v, w) \in [[\alpha]]^G\}$
  - yes: PATH (v)-/has_alpha/-/-(v) IS (v)-/\alpha/->()

- $[[\neg \phi]]^G = V - [[\phi]]^G$
  - yes: PATH (v)-/not_phi/-/-(v) IS (v) WHERE NOT $\phi$

- $[[\phi \land \psi]]^G = [[\phi]]^G \cap [[\psi]]^G$
  - yes: PATH (v)-/phi_and_psi/-/-(v) IS (v) WHERE $\phi$ AND $\psi$

- $[[\phi \lor \psi]]^G = [[\phi]]^G \cup [[\psi]]^G$
  - yes: PATH (v)-/phi_or_psi/-/-(v) IS (v) WHERE $\phi$ OR $\psi$

- $[[c]]^G = \{v \in V \mid \rho(v) = c\}$
  - yes: PATH (v)-/rho_is_c/-/-(v) IS (v) WHERE $v.\rho = c$

- $[[c\neq]]^G = \{v \in V \mid \rho(v) \neq c\}$
  - yes: PATH (v)-/rho_is_c/-(v) IS (v) WHERE $v.\rho \neq c$

- $[[\alpha = \beta]]^G = \{v \in V \mid \exists w, y (v, w) \in [[\alpha]]^G, (v, y) \in [[\beta]]^G, \rho(w) = \rho(y)\}$
  - yes: PATH (v)-/alpha_eq_beta/-/-(v) IS (v)-/\alpha/-/>(w), (v)-/\beta/-/>(y) WHERE w.\rho = y.\rho

- $[[\alpha \neq \beta]]^G = \{v \in V \mid \exists w, y (v, w) \in [[\alpha]]^G, (v, y) \in [[\beta]]^G, \rho(w) \neq \rho(y)\}$
  - yes: PATH (v)-/alpha_not_eq_beta/-/-(v) IS (v)-/\alpha/-/>(w), (v)-/\beta/-/>(y) WHERE w.\rho \neq y.\rho
Can we do even better?

- Shorthand syntax for node labels
- Shorthand syntax for matching *any edge*
- Shorthand syntax for property predicates

- What is the scope of PATH declarations?
- ...and where in the query should they go?
  - Beginning of query?
  - Sub-clause of MATCH?
  - Anywhere in the query?
- Can we parameterize PATH declarations?
  - ...and use parameters for both input and output...
What other features would you like to see in Cypher?