Cypher Refresher

(Just in Case)

Resources for learning Cypher

- Cypher Reference Card
  http://neo4j.com/docs/cypher-refcard/
- Cypher Railroad Diagrams
  http://bit.ly/cypher-railroad
- Neo4j Developer Pages
  http://neo4j.com/developer/cypher
- Neo4j Documentation
  http://neo4j.com/docs

Cypher Query Structure

```
MATCH pattern

WHERE predicate

RETURN/WITH expression AS alias ...

ORDER BY expression

SKIP ... LIMIT ...
```

Case sensitivity

- Cypher keywords/clauses are mostly case insensitive
- But, several things in the datastore are case sensitive:
  - Labels
  - Relationship types
  - Property names (keys)
  - Variables you use in Cypher

Labels

- labels are like type tags for nodes
- label-based indexes and constraints
- syntax:

```
CREATE (p:Person)// create labeled node
SET p:Person     // set label on node
REMOVE p:Person  // remove label

MATCH (p:Person) // match nodes with label
WHERE p:Person   // label predicate
RETURN labels(p) // get label collection
```

MATCH

MATCH Patterns

- pattern matching--describe your traversal in a pattern!
- use labels in your pattern to give your query starting points
- create new variables as the query matches the pattern

MATCH Examples

```
... // a simple pattern, with RELTYPE
MATCH (n)-[:LINK]-(m)

... // match a complex pattern
MATCH (n)-->(m)<--(o), (p)-->(m)

... // match a variable-length path
MATCH p=(n)-[:LINKED*]-()

... // use a specialized matcher
MATCH p=shortestPath((n)-[*]-(o))
```

WHERE

WHERE

- use **MATCH** to find patterns, and **WHERE** to filter them
- use patterns as predicates in **WHERE**,

  eg. `WHERE NOT EXISTS ((n)-->())`
- can't create new identifiers in **WHERE**, only predicates

  on existing identifiers defined in **MATCH** or **WITH**

WHERE examples

```
... // filter on a property value
WHERE n.name = "Andrés"

... // filter with predicate patterns
WHERE NOT (n)<--(m)

... // filter on path/collection length
WHERE length(p) > 3

... // filter on multiple predicates
WHERE n.born < 1980 AND n.name =~ "A.*"
```

RETURN

RETURN

- like SQL's SELECT: specify the projection you want to see in results
- alias results with AS
- calculate expressions as they're returned (math, etc.)
- aggregations: collect, count, statistical

RETURN examples

```
… // p aliased to person
RETURN p AS person

… // implicit group by n, count(*)
RETURN n, count(*) AS count

… // collect things into a collection
RETURN n, collect(r)
```

ORDER BY/LIMIT/SKIP

ORDER BY/SKIP/LIMIT

- Cypher doesn't guarantee ordering unless you `ORDER BY`
- `SORT, LIMIT, SKIP`
- things you order by must be in the `RETURN/WITH` clause
- all optional clauses (you can do `LIMIT` without `ORDER BY`, etc.)

ORDER BY/SKIP/LIMIT examples

```
... // descending sort, limit 5
RETURN n, count(*) AS count
ORDER BY count DESC
LIMIT 5

... // get the next 5
RETURN n, count(*) as count
ORDER BY count DESC
SKIP 5
LIMIT 5
```

Exercise: Translate English to Cypher
:play movies

Task: Complex Graph Question

Find all Actors and Movies they acted in

Whose name starts with "T"

Aggregate the frequency and movie titles

Filter by who acted in more than 5 movies

Return their name, birth-year and movie-titles

Ordered by number of movies

Limited to top 10

Compare with Query In SQL


```
SELECT a.name, a.born,
     group_concat(m.title) AS movies,
     count(*) AS cnt
 FROM   actors AS a JOIN actor_movie   ON  (a.id = actor_movie.actor_id)
 JOIN   movies AS m
   ON  (actor_movie.movie_id = m.id)
 WHERE  a.name LIKE "T%"
 GROUP BY a.name, a.born
 HAVING cnt > 5
 ORDER BY cnt DESC
```

Exercise: Write and execute the query in Cypher

Try to remember the query structure

Take one step at a time

Use the Cypher Refcard: http://neo4j.com/docs/cypher-refcard

Solution on Next Slide

Solution: Complex Graph Query

1. find people (click `:Person` in browser)
2. add limit
3. find people where name starts with "T"
4. find people whose name starts with "T", who acted in a movie
5. return aggregation
6. add ordering
7. introduce WITH for in-between filter
8. done

Breakdown

MATCH

describes the pattern

MATCH the Pattern

```
MATCH (a:Person)
RETURN a
LIMIT 10
```

WHERE

filters the result set

Filter using WHERE

```
MATCH (a:Person)
WHERE a.name STARTS WITH "T"
RETURN a
LIMIT 10
```

MATCH

describes the pattern

MATCH the Pattern

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE a.name STARTS WITH "T"
RETURN a
LIMIT 10
```

RETURN

returns the results

RETURN the results

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE a.name STARTS WITH "T"
RETURN a.name, a.born
LIMIT 10
```

Aggregation

with auto-grouping

Aggregation

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE a.name STARTS WITH "T"
RETURN a.name, a.born,
       count(m) AS cnt, collect(m.title) AS movies
LIMIT 10
```

ORDER BY / LIMIT / SKIP

sort and paginate

ORDER BY LIMIT - Paginate

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE a.name STARTS WITH "T"
RETURN a.name, a.born, count(m) AS cnt,
   collect(m.title) AS movies
ORDER BY length(movies) DESC
LIMIT 10
```

WITH + WHERE

computes intermediate results + filter

WITH + WHERE - filter

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE a.name STARTS WITH "T"
WITH a, count(m) AS cnt,
   collect(m.title) AS movies
WHERE cnt > 5
RETURN a.name, a.born, movies
ORDER BY length(movies) DESC
LIMIT 10
```

End of Cypher Refresher

Questions?