Graph Database

Advantages

those were taken from their website [http://neo4j.com/why-graph-databases/]

- **performance** : performance tends to stay constent as your data grows
- **flexibility**: the structure and schema may flexes as applications & industry changes; easy to maintain and change schema
- **agility**: aligns with today's agile and test-driven development techniques; easy to evolve with

Data Modeling

taken form [http://neo4j.com/blog/data-modeling-basics/] and [http://neo4j.com/developer/graph-db-vs-rdbms/]

NorthWind dataset migration to neo4j : [http://neo4j.com/developer/guide-importing-data-and-etl/]

Comparing to the traditional relational model

foreign key, join tables = layers of complexity (required in a traditional relational model)

in graph based relational model: we enrich by adding labels and attributes & relationships

relational databases have *rigid* schemas and complex data modeling process- they are not suited for rapid change

Graph-based databases

Composed of two elements :

- **node** : represents and entity (person, place, thing, category) - are the data records in the graph
- **relationship** : how two nodes/entities are associated

Neo4J as a graph database

- **properties**: named data values - stores the data associated witg nodes
- **labels**: attributed to nodes to group multiple nodes together
    - *example*: in a social graph, apply label "persons" to multiple nodes
    - nodes may have multiple label associated
- **relationship**: are connecting two nodes
    - **properties**: relationship can also have data attached with *propeties*.
    - *example:* Ian Knows Email since 5 yrs

Schema

Neo4j is a schema-optional graph database, you can you a schema or not.

Cypher - Neo4j query language

```
CREATE (ee:Person { name: "Emil", from: "Sweden", klout: 99 })
```

`CREATE` clause creastes a new node

`ee:Person` creates a new node ee labelled as a *Person*.

`{}` properties associated with the node

```
MATCH (ee:Person) WHERE ee.name = "Emil" RETURN ee;
```

`MATCH` clause matches a specified node or relationship

`(ee:Person)` specifies the pattern we are looking for (ie. single node with label 'Person') and will assign variable *ee* to it.

```
MATCH (ee:Person) WHERE ee.name = "Emil"
CREATE (js:Person { name: "Johan", from: "Sweden", learn: "Surfin'" }),
           (jp:Person { name: "Jérémie", from: "Canada", learn: "Somethin'"
})
       (ee)-[:KNOWS { since: 2001 }]->(jp),
       (jp)-[:KNOWS { since: 1994 }]->(js)
```

Matches *ee* with a person named Emil

Creates a person *js* names Johan and *jp* named *Jérémie* and

- associates nodes *ee* with *jp* with a KNOWS relationship and a property since
- associates nodes *jp* with *js* with a KNOWS relationship and a property since

```
MATCH (ee:Person)-[:KNOWS]-(friends)
WHERE ee.name = "Emil" RETURN ee, friends
```

matches people associated with *Emil* that have a KNOWS relationship associating them with him.

```
MATCH (js:Person)-[:KNOWS]-()-[:KNOWS]-(surfer)
WHERE js.name = "Johan" AND surfer.hobby = "surfing"
RETURN DISTINCT surfer
```

```
MATCH (cust:Customer)-[:PURCHASED]->(:Order)-[o:ORDERS]->(p:Product),
      (p)-[:PART_OF]->(c:Category {categoryName:"Produce"})
RETURN DISTINCT cust.contactName as CustomerName, SUM(o.quantity) AS
TotalProductsPurchased
```

- very easy and natural to query on relationships within entities
- very natural query language