Institute of Data

2021

© 2021 Institute of Data

What and Why are NoSQL Databases

What

- non-tabular
- non-relational

Why

- scalable
- flexible
- simple

- storage and retrieval of data that is modelled by means other than tabular relations
- not a new idea (1960+)
- supports **distributed** storage and processing
- term ‘NoSQL’ popularised by Facebook, Amazon, Google, etc.

© 2021 Institute of Data

Characteristics of NoSQL Databases

Advantages

- avoid administration & expense of RDBMS
- small footprint
- easy to modify schemas

Use Cases

- simple data requirements
- application-specific data
- start-ups

© 2021 Institute of Data

Characteristics of NoSQL Databases – cont'd

Disadvantages

- many don't support true ACID transactions
  - application code is obliged to try to manage concurrency issues
- easy to modify schemas
  - application developers need to collaborate closely to ensure schema development is under control
- much slower than RDBMS
- lack powerful management & development features of RDBMS

© 2021 Institute of Data

NoSQL Databases Types

- wide column store (https://docs.aws.amazon.com/redshift/latest/dg/c_columnar_storage_disk_mem_mgmnt.html)
  - Formatting and names can vary from row to row inside the same table.
  - Each column is stored separately on disk.
  - > Hadoop / HBase, MapR, BigTable, Hortonworks, Cloudera, Cassandra, Informix
- document store
  - > MongoDB, CouchDB, Azure DocumentDB
- key value / tuple store
  - > DynamoDB, Azure Table Storage, Oracle NoSQL
- graph databases
  - > Neo4j
- multi-model databases
  - > ArangoDB, OrientDB
- object databases
  - > Versant, Objectivity VelocityDB

https://medium.com/analytics-vidhya/sql-vs-nosql-3c948a459335

© 2021 Institute of Data

NoSQL Databases – cont'd

- Document databases
  - MongoDB

© 2021 Institute of Data

Document Databases

- semi-structured data
- records do not all need to have the same fields

XML:

a record is a block of XML tags and values

```
<contact>
  <firstname>Bob</firstname>
  <address>5 Oak St.</laddress>
  <hobby>saling</hobby>
</contact>
```

JSON:

a record is a list of key:value pairs

```
{
   "FirstName": "Bob",
   "Address": "5 Oak St.",
   "Hobby": "sailing"
}
```


© 2021 Institute of Data

MongoDB

- an **open-source** document database (https://docs.mongodb.com/manual/introduction/)
  - no charge for *Community Server* version
- high performance
  - embedded data models reduce I/O activity
  - indexes
    - can include keys from embedded documents, arrays
- high availability
  - replication facility
    - automatic fail-over
    - data redundancy
- automatic scalability (horizontal)

https://www.mongodb.com/try/download/community?tck=docs_server

© 2021 Institute of Data

MongoDB

- CRUD ✓
  - create, read, update, delete
- ACID ?
  - traditionally:
    - document databases are only ACID-compliant only at document level
    - no *transactions* for containing multiple I/O operations
    - application code is obliged to emulate transactions, if required
    - latency results in an **eventual consistency** model
  - MongoDB 4.0
    - introduced transactions

© 2021 Institute of Data

MongoDB: High-Level Objects

Document*:*

- a set of field:value pairs
  - values can be hierarchical
- analogous to RDB row

examples:

{ name: “sue”, age: 26, status: “A”, groups: [“news”, “sports”] }

{ name: “fred”, status: “A”, groups: [“sports”, “hobbies”, “cars”] }

{ name: { first: “fred”, last: “bloggs” }, status: “A”, groups: [“sports”, “hobbies”, “cars”] }

Collection*:*

- a logical group of documents
- analogous to RDB table

© 2021 Institute of Data

Relational vs Document DB

Relational concept	MongoDB equivalent
Database	Database
Tables	Collections
Rows	Documents
Index	Index

© 2021 Institute of Data

MogoDB console

- mongo console
- Creating Document
  - doc = {"id": 1, "first_name": “ABC", "last_name": “XYZ" }

- Creating collection
  - db.customer.insertOne(doc)
  - db.customer.find()
  - db.customer.find().pretty()

Insert Different documents into the collection

doc = {"id": 1, "first_name": “ABC", "last_name": “XYX", "New": 3 }

© 2021 Institute of Data

.find()

- More like a SELECT statement in SQL

- db.customer.find()
- db.customer.find({"first_name":“ABC"})
- db.customer.find({"first_name":“ABC"}).pretty()
- db.customer.find({"first_name":“ABC", "New": 3})

- Cheat Sheet along with Video explanation

  https://gist.github.com/bradtraversy/f407d642bdc3b31681bc7e56d95485b6
  https://www.youtube.com/watch?v=-56x56UppqQ

© 2021 Institute of Data

Knowledge Check

- Create a Collection with one document containing your information

© 2021 Institute of Data

MongoDB with Python

python -m pip install pymongo

```
# Create a DB called customer

import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")

mydb = myclient[“mydatabase"]

#collection
mycol = mydb["customers"]
```


© 2021 Institute of Data

MongoDB with Python

example: python -m pip install pymongo

```
import pymongo
connection = pymongo.MongoClient("mongodb://localhost")
db = connection.school
students = db.students
cursor = students.find()

# find minimum homework score...
for doc in cursor:
    scores = doc["scores"]
    minhs = 101
    for entry in scores:
        if entry["type"] == "homework":
            if entry["score"] < minhs:
                minhs = entry["score"]
```

- create mongod client
- connect to database ‘school’
- create alias for table ‘students’
- fetch all rows into cursor

- loop through docs in cursor
- get value (doc) associated with key ‘scores’
- initialise min to impossibly large value (> 100%)
- loop through ‘scores’ docs, looking for ‘homework’ keys
- test each corresponding score to find new minimum

© 2021 Institute of Data

Lab 3.1.4: Python with MongoDB (Optional homework)

- Purpose:
  - To develop skills in NoSQL database programming with MongoDB
- Materials:
  - ‘Lab 3.1.4.ipynb’

© 2021 Institute of Data

Institute of Data

(Slides + Jupyter)

NoSQL Databases – cont'd

- Graph Databases
  - Neo4j (***Network Exploration and Optimization 4 Java***)

© 2021 Institute of Data

Degree of Kevin Bacon??

"Degree of Einstein"

friends

friend

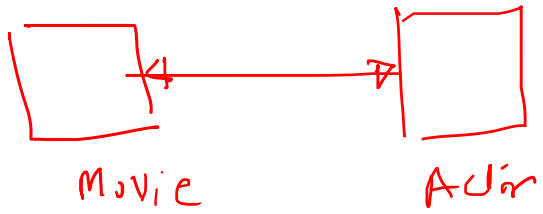
- Nodes

Actors & Movies are "Nodes"
Relationships (Edges) ties two nodes together

© 2021 Institute of Data

Graph Databases

- model members and relationships as a network
- high-level objects:
  - nodes
    - entities (e.g. people, accounts, organisations), Movies
  - edges
    - connections between nodes
  - properties
    - node: differentiates types of nodes
      - roles, classifications, etc.
    - **edge: describes the relationship**
      - 2-way, 1-way, directionless
      - friend, follower, commenter, etc.



© 2021 Institute of Data

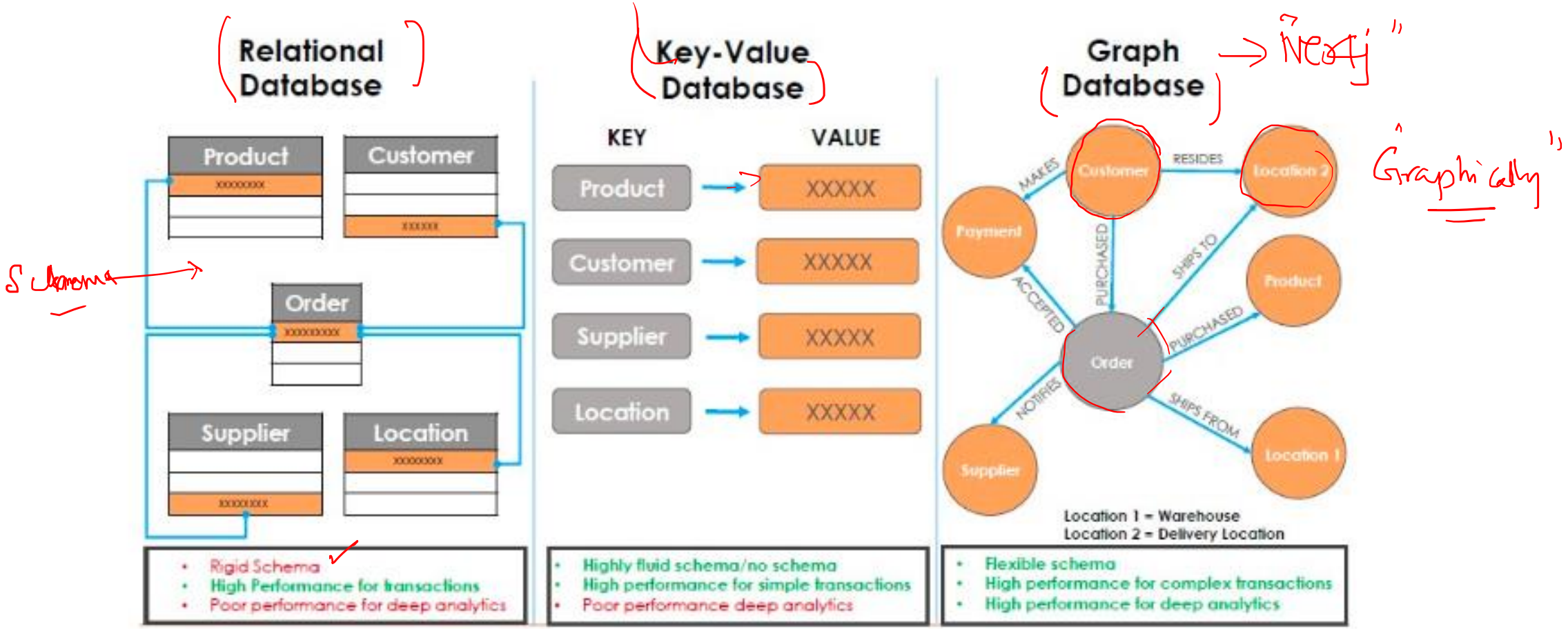
Comparison with SQL

SQL world	Neo4j world
Table	Node label
Row	Node
Column	Node property
Foreign key	Relationship
Join table	Relationship

© 2021 Institute of Data

Comparison

Documents



© 2021 Institute of Data

Neo4j

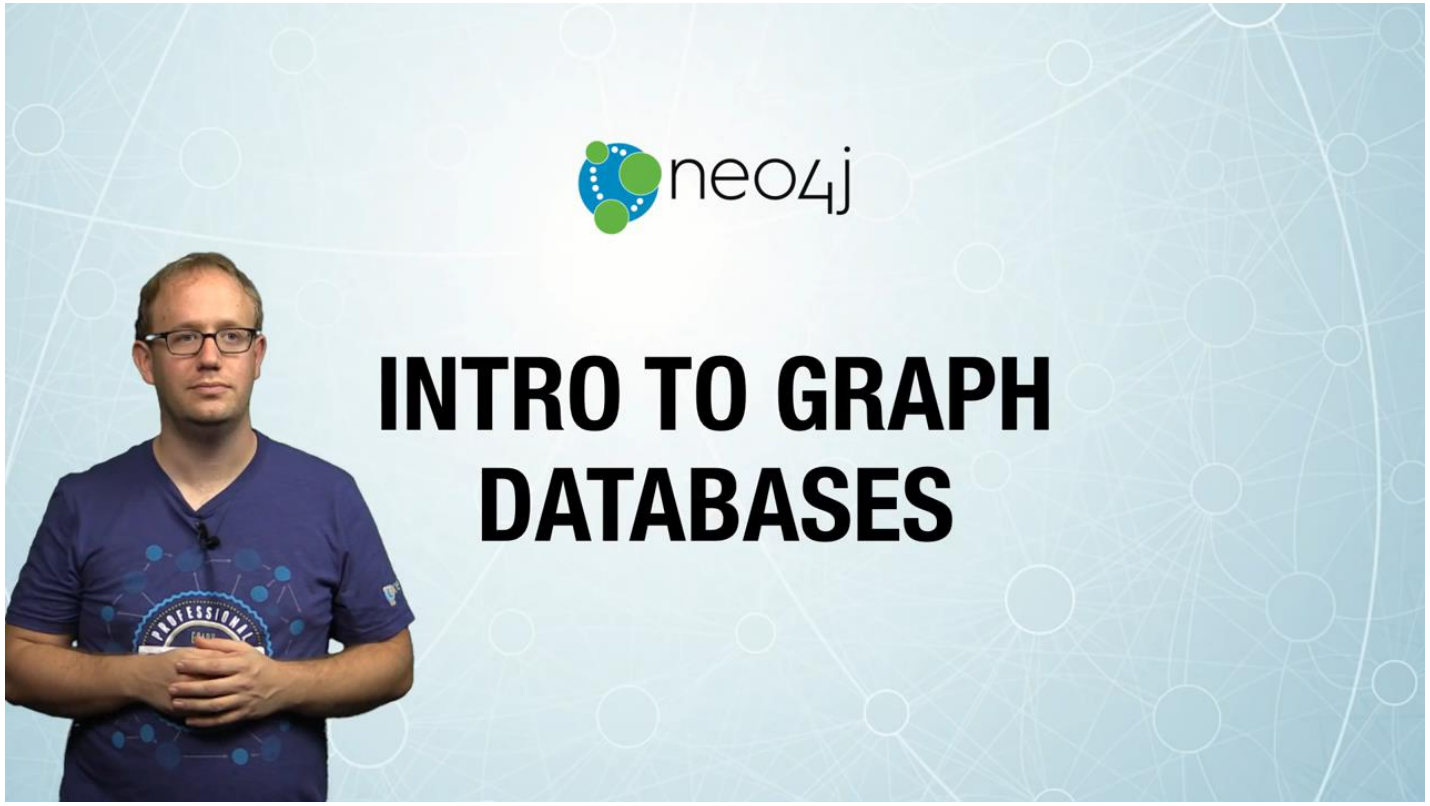
- open source
  - no charge for *Community Server* edition
- ACID-compliant, transactional database with native graph storage & processing
- online backup
- high availability
- most popular graph database

Graph

https://go.neo4j.com/rs/710-RRC-335/images/Neo4j_Top5_UseCases_Graph%20Databases.pdf

© 2021 Institute of Data

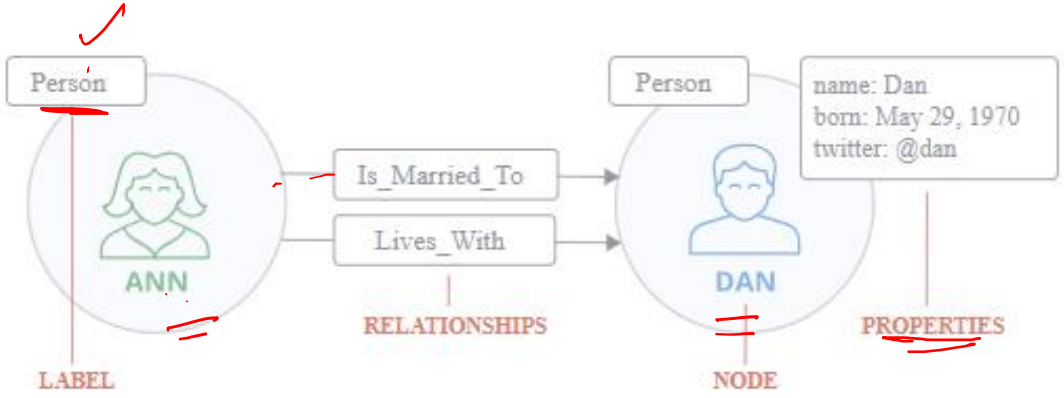
Evolution of DBs

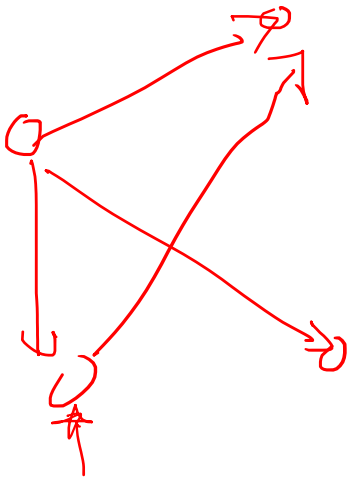


© 2021 Institute of Data

Neo4j: Basics

The Labeled Property Graph Model





Nodes

- Nodes are the main data elements
- Nodes are connected to other nodes via **relationships**
- Nodes can have one or more **properties** (i.e., attributes stored as key/value pairs)
- Nodes have one or more **labels** that describes its role in the graph

Relationships

- Relationships connect two nodes
- Relationships are directional
- **Nodes** can have multiple, even recursive relationships
- Relationships can have one or more **properties** (i.e., attributes stored as key/value pairs)

© 2021 Institute of Data

Creating a node

[CQL] Cypher Query Language

- CREATE (n {id: 1}) return n

name

Cypher: create nodes

CREATE (n:Label {property: $property_value})

CREATE (n {id: 1})

CREATE (n {id: 1, name: "Kevin Bacon", birthdate:"July 8, 1958", birth_city: "Philadelphia", birth_state:"Pennsylvania"})

Properties

CREATE (n:Actor {id: 1, name: "Kevin Bacon", birthdate:"July 8, 1958", birth_city: "Philadelphia", birth_state:"Pennsylvania"})

Property

Name	Kevin Bacon
Birthdate	July 8, 1958
Place of Birth	Philadelphia, PA
id	1

© 2021 Institute of Data

Need to create Nodes for every actor

Vin Diesel
July 18, 1967
Alameda County, CA
5

Tyrese Gibson
December 30, 1978
Los Angeles, CA
3

Kevin Bacon
July 8, 1958
Philadelphia, PA
1

John Turturro
February 28th, 1957
Brooklyn, NY
4

Tom Cruise
July 3, 1962
Syracuse, NY
2

© 2021 Institute of Data

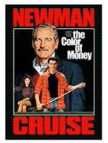
Creating a Node (Movies)

Cypher: *create*

```
CREATE (:Movie {id: 1,   name: "Fast Five",                          release_date:2011})
CREATE (:Movie {id: 2,   name: "Transformers: Revenge of the Fallen", release_date:2009})
CREATE (:Movie {id: 3,   name: "The Color of Money",                 release_date:1986})
CREATE (:Movie {id: 4,   name: "A Few Good Men",                     release_date:1992})
```

Fast Five
2011
1

Transformers: Revenge of the Fallen
2009
2



The Color of Money
1986
3

A Few Good Men
1992
4

Properties

Fast Five
2011
1

Transformers: Revenge of the Fallen
2009
2

The Color of Money
1986
3

A Few Good Men
1992
4

© 2021 Institute of Data

Creating Relationships

- **Cypher:** ***create***
  - *Create is used for relationships, as well as nodes*
  - *CREATE (n) - [:RELATIONSHIP_NAME] -> (m)*

Actor movie

Kevin Bacon	Tom Cruise	Tyrese Gibson	John Turturro	Vin Diesel
1	2	3	4	5

Fast Five	Transformers: Revenge of the Fallen	The Color of Money	A Few Good Men
1	2	3	4

© 2021 Institute of Data

Creating Relationship

- **Cypher: create**
  - **Set up our relationships for our actors in the movies**

```
MATCH (a:Actor {id: 1} ), (m:Movie {id: 4}) CREATE (a) - [:ACTED_IN] -> (m)
MATCH (a:Actor {id: 2} ), (m:Movie {id: 4}) CREATE (a) - [:ACTED_IN] -> (m)
MATCH (a:Actor {id: 2} ), (m:Movie {id: 3}) CREATE (a) - [:ACTED_IN] -> (m)
MATCH (a:Actor {id: 3} ), (m:Movie {id: 3}) CREATE (a) - [:ACTED_IN] -> (m)
MATCH (a:Actor {id: 3} ), (m:Movie {id: 2}) CREATE (a) - [:ACTED_IN] -> (m)
MATCH (a:Actor {id: 4} ), (m:Movie {id: 2}) CREATE (a) - [:ACTED_IN] -> (m)
MATCH (a:Actor {id: 4} ), (m:Movie {id: 1}) CREATE (a) - [:ACTED_IN] -> (m)
MATCH (a:Actor {id: 5} ), (m:Movie {id: 1}) CREATE (a) - [:ACTED_IN] -> (m)
```

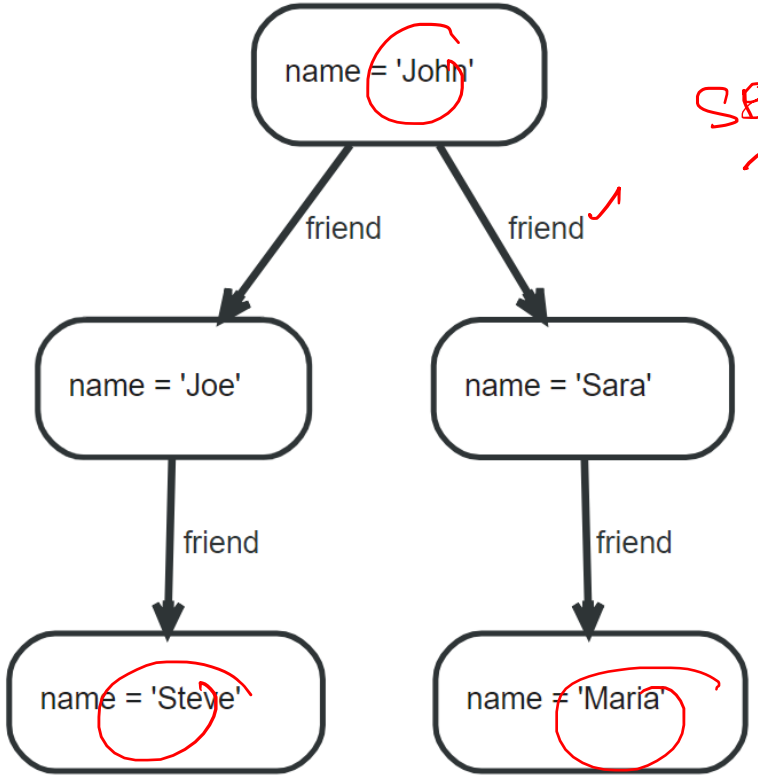
  - In order to relate two nodes (with a create statement), we need to identify which nodes to create.
  - MATCH (n:LABEL {property: {property_value}) return n

Kevin Bacon	Tom Cruise	Tyrese Gibson	John Turturro	Vin Diesel
1	2	3	4	5

Fast Five	Transformers: Revenge of the Fallen	The Color of Money	A Few Good Men
1	2	3	4

© 2021 Institute of Data

Cypher Query Language



SELECT

- example: find friends of friends of John

```
MATCH (john {name: 'John'})-[:friend]->
()-[:friend]->(fof)
RETURN john.name, fof.name
```

output:

```
+----------------------+
| john.name | fof.name |
+----------------------+
| "John"    | "Maria"  |
| "John"    | "Steve"  |
+----------------------+
```


© 2021 Institute of Data

Lab 3.1.5: Neo4j and Python (Optional homework)

- Purpose:
  - To develop familiarity with graph database programming (Neo4j) using:
    - the Neo4j GUI
    - a Python library for Neo4j https://neo4j.com/docs/api/python-driver/current/

- Resources:
  - Neo4j built-in tutorials
  - Cypher cheatsheet
    - https://neo4j.com/docs/cypher-refcard/3.2/

- Materials:
  - 'Lab 3.1.5.ipynb'

© 2021 Institute of Data

Discussion: SQL vs NoSQL

SQL	NoSQL
Traditional rows and columns ● governed data model	No predefined data structure ● database at mercy of developers
Strict structure (incl. primary keys) ● schema changes difficult, risky	Ideal for unstructured data ● schema can change with application requirements
Entire column for each feature	Cheaper hardware
Industry standard	Supports design flexibility & growth ➢ popular among startups
ACID	Application code must manage transactions

© 2021 Institute of Data

Discussion: NoSQL with SQL ?!

- Why has SQL infiltrated the NoSQL paradigm?

© 2021 Institute of Data

Questions?

© 2021 Institute of Data

Appendices

© 2021 Institute of Data

Relational Databases – Normalisation

Codd's 1st-normal form

- the domain of each attribute contains only atomic (indivisible) values
- the value of each attribute contains only a single value from that domain

Customer

Customer ID	First Name	Surname	Telephone Number
123	Pooja	Singh	555-861-2025, 192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53; 182-929-2929
789	John	Doe	555-808-9633

Customer

Customer ID	First Name	Surname	Telephone Number
123	Pooja	Singh	555-861-2025
123	Pooja	Singh	192-122-1111
456	San	Zhang	182-929-2929
456	San	Zhang	(555) 403-1659 Ext. 53
789	John	Doe	555-808-9633

© 2021 Institute of Data

Relational Databases – – Normalisation - cont'd

Codd's 2nd-normal form

- in 1st-normal form
- no non-prime attribute is dependent on any proper subset of any candidate key of the relation
  (an attribute that is not a part of any candidate key of the relation)

Electric Toothbrush Models

Manufacturer	Model	Model Full Name	Manufacturer Country
Forte	X-Prime	Forte X-Prime	Italy
Forte	Ultraclean	Forte Ultraclean	Italy
Dent-o-Fresh	EZbrush	Dent-o-Fresh EZbrush	USA
Kobayashi	ST-60	Kobayashi ST-60	Japan
Hoch	Toothmaster	Hoch Toothmaster	Germany
Hoch	X-Prime	Hoch X-Prime	Germany

Electric Toothbrush Manufacturers

Manufacturer	Manufacturer Country
Forte	Italy
Dent-o-Fresh	USA
Kobayashi	Japan
Hoch	Germany

Electric Toothbrush Models

Manufacturer	Model	Model Full Name
Forte	X-Prime	Forte X-Prime
Forte	Ultraclean	Forte Ultraclean
Dent-o-Fresh	EZbrush	Dent-o-Fresh EZbrush
Kobayashi	ST-60	Kobayashi ST-60
Hoch	Toothmaster	Hoch Toothmaster
Hoch	X-Prime	Hoch X-Prime

© 2021 Institute of Data

Relational Databases – – Normalisation - cont'd

Codd's 3rd-normal form

- in 2nd-normal form
- every non-prime attribute (of a table) is non-transitively dependent on every key (of a table)

Tournament Winners

Tournament	Year	Winner	Winner Date of Birth
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleveland Open	1999	Bob Albertson	28 September 1968
Des Moines Masters	1999	Al Fredrickson	21 July 1975
Indiana Invitational	1999	Chip Masterson	14 March 1977

Tournament Winners

Tournament	Year	Winner
Indiana Invitational	1998	Al Fredrickson
Cleveland Open	1999	Bob Albertson
Des Moines Masters	1999	Al Fredrickson
Indiana Invitational	1999	Chip Masterson

Winner Dates of Birth

Winner	Date of Birth
Chip Masterson	14 March 1977
Al Fredrickson	21 July 1975
Bob Albertson	28 September 1968

© 2021 Institute of Data

Which RDBMS Object to Use When

- queries
  - **ad hoc** queries
  - **stored** or **generated** in application code
- views
  - stored (**reusable**) queries
  - can incorporate **joins** with other views
  - **preferable** to queries in application code
- stored procedures
  - **more powerful than views** (can query and/or modify data)
  - preferred for delivering data to applications (**security, control, maintainability**)
- reports
  - **formatted output** containers based on tables, views
  - text & graphics
  - usually provided via a separate application (designed for but existing outside the RDBMS)
  - may have built-in subscription service

© 2021 Institute of Data

End of Presentation!

© 2021 Institute of Data