UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

SENG365 Web Computing Architecture: #3 Data persistence: SQL/NoSQL, memory stores, GraphDB

Ben Adams

benjamin.adams@canterbury.ac.nz
310, Erskine Building

News

- End-of-term-1 test
  - **Study Needs Agreement: let me know if you have any**

Labs

Term 1 (6 weeks)

- 3 x pre-labs (self-study)
- Week 2: Lab 1
- Week 3: Lab 2
- **Week 4: Lab 3**
- Week 5 & 6: assignment support

Term 2 (6 weeks)

- Week 7: Lab 4
- Week 8: Lab 5
- Week 9: Lab 6
- Week 10 & 11: assignment support
- Week 12: **compulsory** lab
  - Assignment 2 testing

Assessment

The assessment

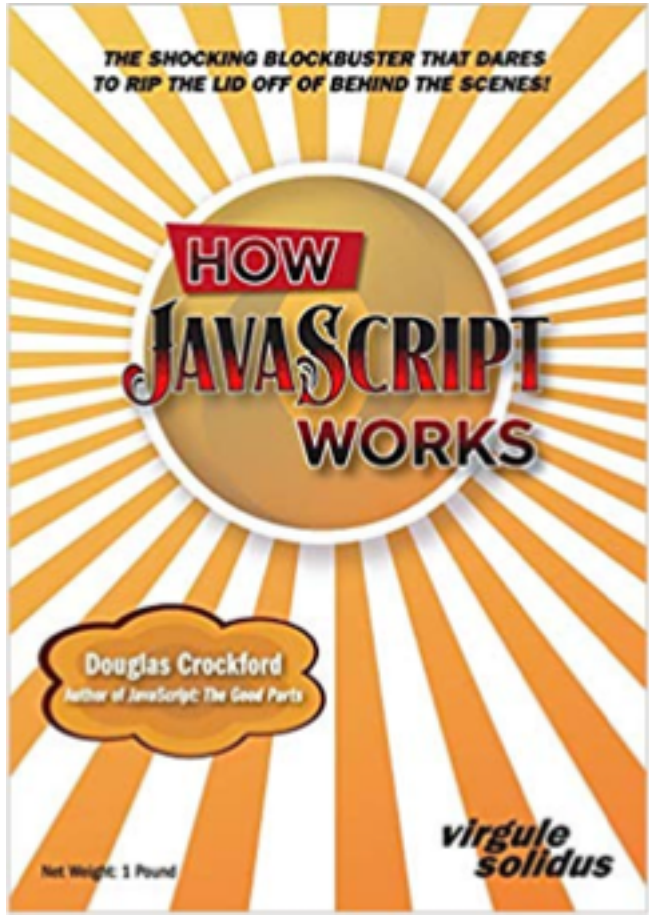
- Assignment 1 (25%)
  - **No extension**
- Mid-semester test (20%)
  - **Wed 24 March, 7pm**
  - **E8 Lecture Theatre**
- Assignment 2 (25%)
  - Compulsory lab in final week
  - **No extension**
- Exam (30%; 2hrs)

Additional information and requirements

- Assignment resources on Learn
- API specification with skeleton project
- Infrastructure
  - eng-git project
  - MariaDB (MySQL) server
  - Docker VM server
  - ‘Behavioural server’

In the lecture this week

- JSON
- Shift from relational databases to NoSQL
- In-memory data stores
- Distributed databases
- Graph databases and triple stores



POJO and JSON

David Crockford's view: https://json.org/

A useful tool: https://json-to-js.com/ with npm version: `npm i -g json-to-js`

Useful tools (but not always accurate): https://tools.learningcontainer.com/

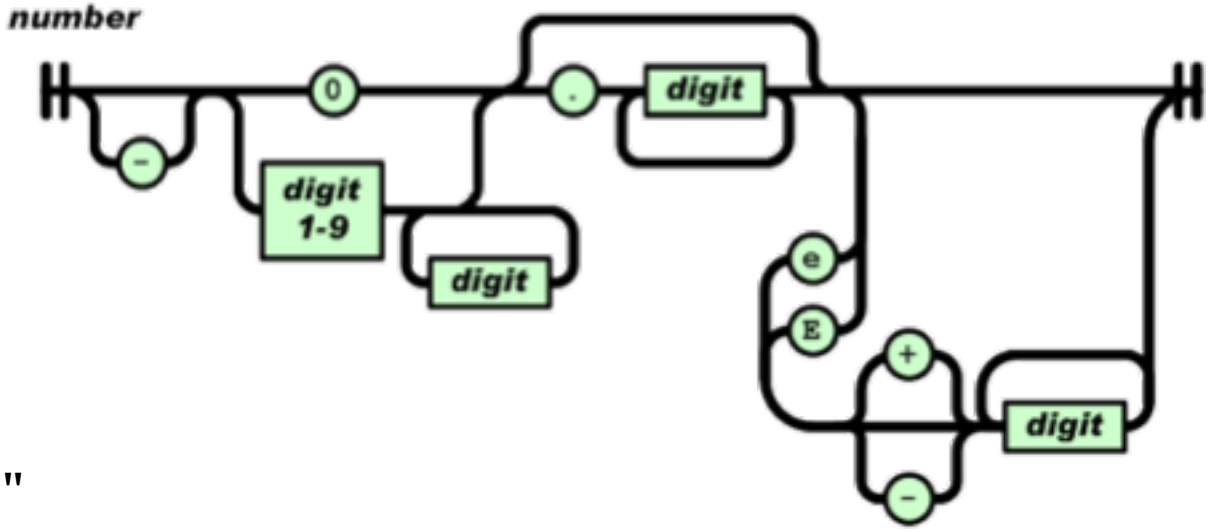
JSON: semi-formal definitions

- JSON is a lightweight **data-interchange** format.
  - https://json.org/
- A syntax for serializing data e.g., objects, arrays, numbers, strings, etc.
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON
- Not specific to JavaScript
  - Was originally intended for data interchange between Java and JavaScript
- No versioning for JSON: why?

JSON variants and extensions

- JSON-T: https://developers.squarespace.com/what-is-json-t
- JSON templating: ~JSON + (HTML+JS) templating
- JsonML: JSON Markup Language: http://www.jsonml.org/
- JSON-LD: https://json-ld.org/
- HTML microformats using JSON:
  - http://microformats.org/wiki/h-product

JSON: some rules



https://json.org/

- All key-names are double-quoted
- Values:
  - Strings are double-quoted
  - Non-strings are not quoted
- Use \ to escape special characters, such as \ and "
- Numbers need to be handled carefully
  - e.g. a decimal must have a trailing digit
    - Correct: 27.0
    - Incorrect 27.
    - Correct 27
- Can’t – shouldn't – JSONify functions or methods

- See the following for guidance:
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON
  - https://json.org/
- And this for an… interesting discussion… on JSON syntax:
  - https://stackoverflow.com/questions/19176024/how-to-escape-special-characters-in-building-a-json-string

Example

Paste your JSON object here

```json
{ "firstName" : "Austen",
  "familyName" : "Rainer",
  "title" : "Associate Professor",
  "courses" : {
    "seng365-2019" : "SENG365 2019 Software Engineering",
    "numberOfStudents" : 128,
    "numberOfLectures" : 24,
    "commencedTeaching" : true,
    "lectureTitles" : ["Introduction: it's great",
    "JavaScript: \" The usual parts",
    "Asynchronous behaviour"]
  }
}
```

add trailing comma

JS version appears here

```js
{
  firstName: 'Austen',
  familyName: 'Rainer',
  title: 'Associate Professor',
  courses: {
    'seng365-2019': 'SENG365 2019 Software Engineering',
    numberOfStudents: 128,
    numberOfLectures: 24,
    commencedTeaching: true,
    lectureTitles: [
      'Introduction: it\'s great',
      'JavaScript: " The usual parts',
      'Asynchronous behaviour'
    ]
  }
}
```

Copy result to clipboard

https://json-to-js.com/

Relational databases

- One of the few situations where a theoretical contribution in academic computer science led innovation in industry
- Relational model (E.F. Codd 1970)
  - Data is presented as relations
  - Collections of tables with columns and rows (tuples)
  - Each tuple has the same attributes
  - Unique key per row
  - Relational algebra that defines relational operations: UNION, INTERSECT, SELECT, JOIN, etc.

ACID database transactions

- **Atomicity**—“all or nothing” if one part of a transaction fails, then the whole transaction fails
- **Consistency**—the database is kept in a consistent state before and after transaction execution
- **Isolation**—one transaction should not see the effects of other, in progress, transactions
- **Durability**—ensures transactions, once committed, are persistent

“The end of an architectural era?”

- Traditional RDBMSs evolved from transaction processing systems where ACID properties were the only likely requirement for data handling

- over the past 35 years to so:
  - Moore’s Law—CPU architectures have changed how they acquire speed
  - New requirements for data processing have emerged
  - Stonebraker et al. (2007), suggest that “one size fits all” DBs end up excelling at nothing; complete DB rethink required.
  - Debateable… relational databases are still extremely useful in many cases

Emerging application areas

- Text processing (specialised search engines—Google)
- Data warehouses (column stores)
- Stream processing systems
- Scientific and intelligence databases
- Big Data and the Internet of Things
- Smart mobile devices (phones, tablets, watches, etc.)

‘Shared’ vs. ‘shared nothing’ approaches

- Main RAM sizes—1970: 1MiB … 2020: 4TiB
- Resource control (the RDBMS is its own OS)
- Grid computing (many low-spec versus big machine)
- High availability (single machine recovered from tape backup versus hot standby)
- Tuning (computer time/memory versus user time costs)

- Many areas are moving towards a distributed model...

CAP Theorem

- In distributed computing, **choose two** of:
  - **Consistency**—every read receives the most recent data
  - **Availability**—every read receives a response
  - **Partition tolerance**—system continues if network goes down

- Situation is actually more subtle than implied above
  - Can adaptively chose appropriate tradeoffs
  - Can understand semantics of data to choose safe operations

BASE

- Give up consistency and we can instead get:
  - **Basic Availability**—through replication
  - **Soft state**—the state of the system may change over time
    - This is due to the eventual consistency…
  - **Eventual consistency**—the data will be consistent eventually
    - … if we wait long enough
    - (and probably only if data is not being changed frequently)

ACID versus BASE Example (1/2)

- Suppose we wanted to track people's bank accounts:

  CREATE TABLE user (uid, name, amt_sold, amt_bought)

  CREATE TABLE transaction (tid, seller_id, buyer_id, amount)

- ACID transactions might look something like this:

```
BEGIN
    INSERT INTO transaction(tid, seller_id, buyer_id, amount);
    UPDATE user SET amt_sold=amt_sold + amount WHERE id=seller_id;
    UPDATE user SET amt_bought=amt_bought + amount WHERE id=buyer_id;
END
```

ACID versus BASE Example (2/2)

- If we consider `amt_sold` and `amt_bought` as estimates, transaction can be split:

```
BEGIN
    INSERT INTO transaction(tid, seller_id, buyer_id, amount);
END
BEGIN
    UPDATE user SET amt_sold=amt_sold + amount WHERE id=seller_id;
    UPDATE user SET amt_bought=amt_bought + amount WHERE id=buyer_id;
END
```

- Consistency between tables is no longer guaranteed
- Failure between transactions may leave DB inconsistent

Key-value databases

Overview of key-value databases

- Unstructured data (i.e., schema-less)
- Primary key is the only storage lookup mechanism
- No aggregates, no filter operations
- Simple operations such as:
  - **Create**—store a new key-value pair
  - **Read**—find a value for a given key
  - **Update**—change the value for a given key
  - **Delete**—remove the key-value pair

Advantages

- Simple
- Fast
- Flexible (able to store any serialisable data type)
- High scalability
- Can engineer high availability

Disadvantages

- Stored data is not validated in any way
  - NOT NULL checks
  - colour versus color
- Complex to handle consistency
- Checking consistency becomes the application's problem
- No relationships—each value independent of all others
- No aggregates (SUM, COUNT, etc.)
- No searching (e.g., SQL SELECT-style) other than via key

Examples

- Amazon Dynamo (now DynamoDB)
- Oracle NoSQL Database, ... (eventually consistent)
- Berkeley DB, ... (ordered)
- Memcache, Redis, ... (RAM)
- LMDB (used by OpenLDAP, Postfix, InfluxDB)
- LevelDB (solid-state drive or rotating disk)

“Dynamo: Amazon’s Highly Available Key-value Store”

- Just two operations:
  - put(key, context, object)
  - get(key) → context, object
- Context contains information not visible to caller
  - but is used internally, e.g., for managing versions of the object
- Objects are typically around 1MiB in size

Dynamo design

- Reliability is one of the most important requirements
  - Significant financial consequences in its production use
  - Impacts user confidence
- Service Level Agreements (SLAs) are established
- Used within Amazon for:
  - best seller lists; shopping carts; customer preferences; session management; sales rank; product catalogue

In memory data store (e.g., 

)

- Whole database is stored in RAM
  - Very fast access
  - Useful for cached data on the server
  - E.g. commonly accessed data from RDBMS can be stored in memory store on same computer as the API server.
- Key-value store where the value can be a complex data structure
  - Strings, Bitarrays, Lists, Sets, Hashes
  - Streams (useful for logs)
  - Binary-safe keys
  - Command set for optimized load, storing, and changing data values

Document databases

An example Document in XML format

- XML can be validated against XML schemas (and previously DTDs)
- JSON has replaced XML in many situations

```
<contacts>
  <contact>
    <firstname>Ben</firstname>
    <lastname>Adams</lastname>
    <phone type="Work”>+64 3 3695710</phone>
    <address>
      <type>Work</type>
      <office>310</office>
      <street1>Erskine Building </street1>
      <city>Christchurch</city>
      <state>Canterbury</state>
      <postcode>8040</postcode>
      <country>NZ</country>
    </address>
  </contact>
  <contact> ... </contact>
</contacts>
```

Overview of document databases

- Semi-structured data model
- Storage of documents:
  - typically JSON or XML
  - could be binary (PDF, DOC, XLS, etc.)
- Additional metadata (providence, security, etc.)
- Builds index from contexts and metadata

Advantages

- Storage of raw program types (JSON/XML)
- Indexed by content and metadata
- Complex data can be stored easily
- No need for costly schema migrations
- (Always remember that your DB is likely to need to evolve!)

Disadvantages

- Same data replicated in each document
- Risk inconsistent or obsolete document structures

Example implementations

- ElasticSearch
- LinkedIn's Espresso
- CouchDB
- MongoDB
- Solr
- RethinkDB
- Microsoft DocumentDB
- PostgreSQL (when used atypically)

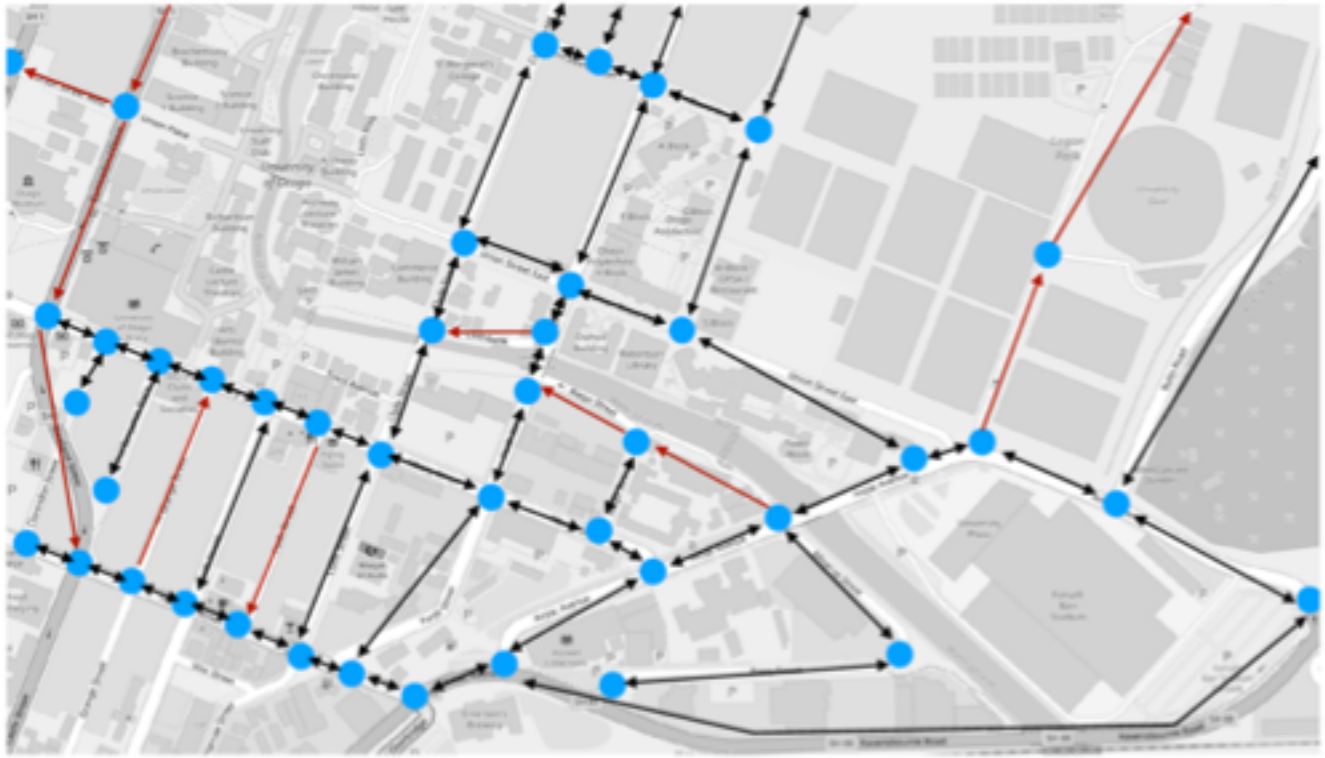
Graph databases

Definitions

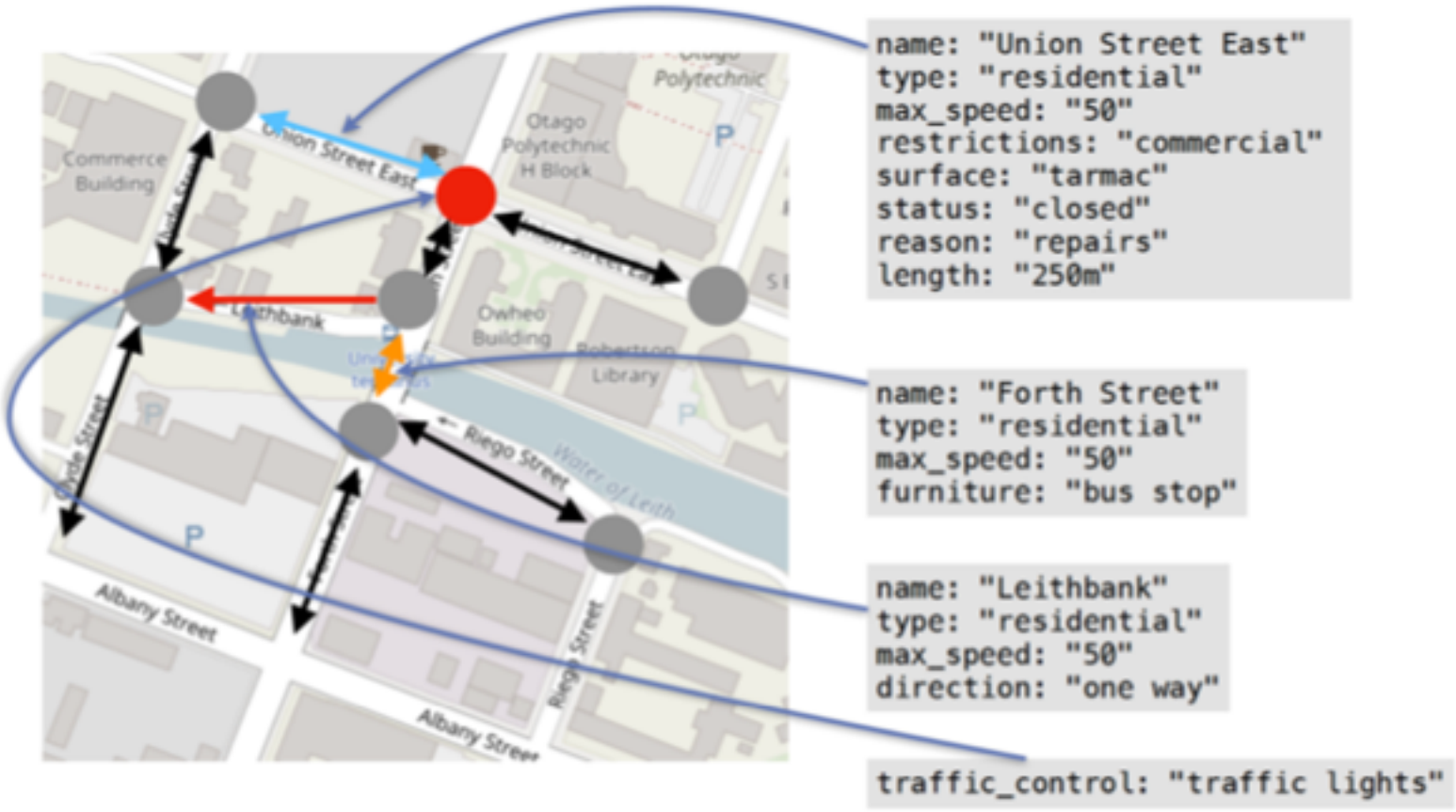
- **Node** (or **vertex**)—represents an entity
- **Edge**—represents relationship between nodes
  - Bidirectional (usually illustrated without arrowheads)
  - Unidirectional (usually illustrated with an arrowhead)
- **Properties**—describe attributes of the node or edge
  - Often stored as a key-value set
- Hypergraph – one edge can join multiple nodes

Street map connectivity is a graph

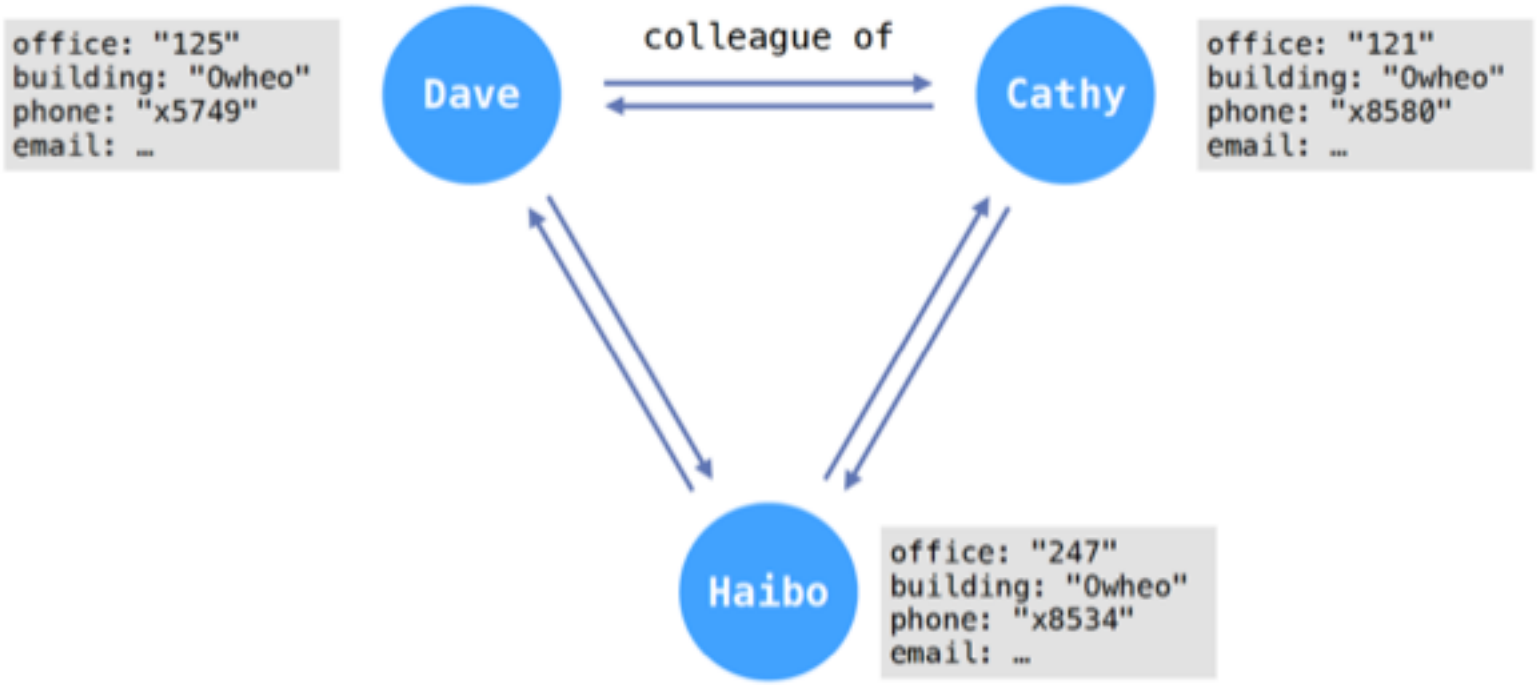
- Node
  - Traffic junction
- Edge
  - Shows traffic flow
  - Can be uni/bidirectional



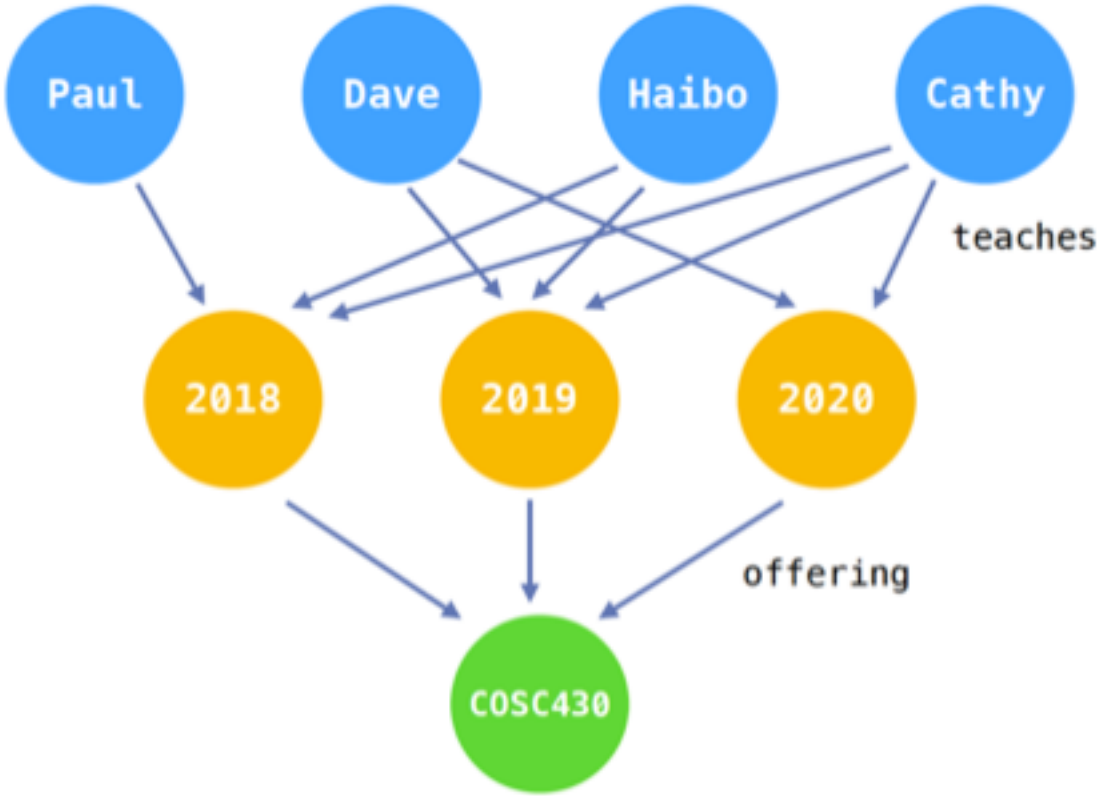
Edges can have properties



Nodes can have properties



Different types of nodes



Different lineages of graph representation

- **Semantic web** style triple-store (**RDF**)
  - Subject-Predicate-Object (“Bob knows Fred”)
  - Linked data on the web
  - Nodes are URIs or values: achieving property lists is arduous
- **Labelled property graphs**
  - Nodes and edges have internal structure: e.g., a key/value set

- Labelled property graphs are more focused on efficient storage than RDF (which is more focused on interchange of information)

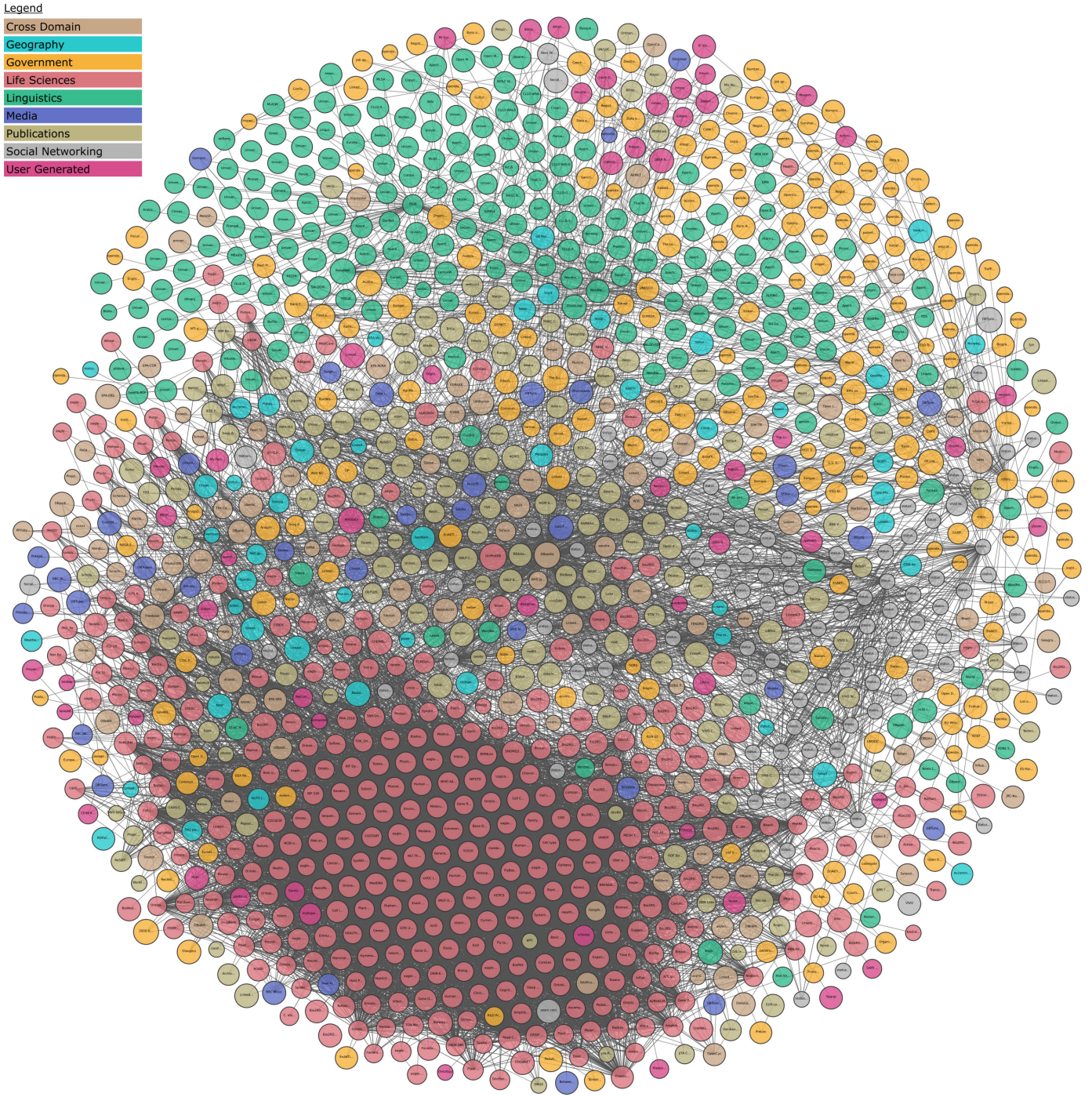
https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/#RDF-property-graph-differences

Linked data cloud

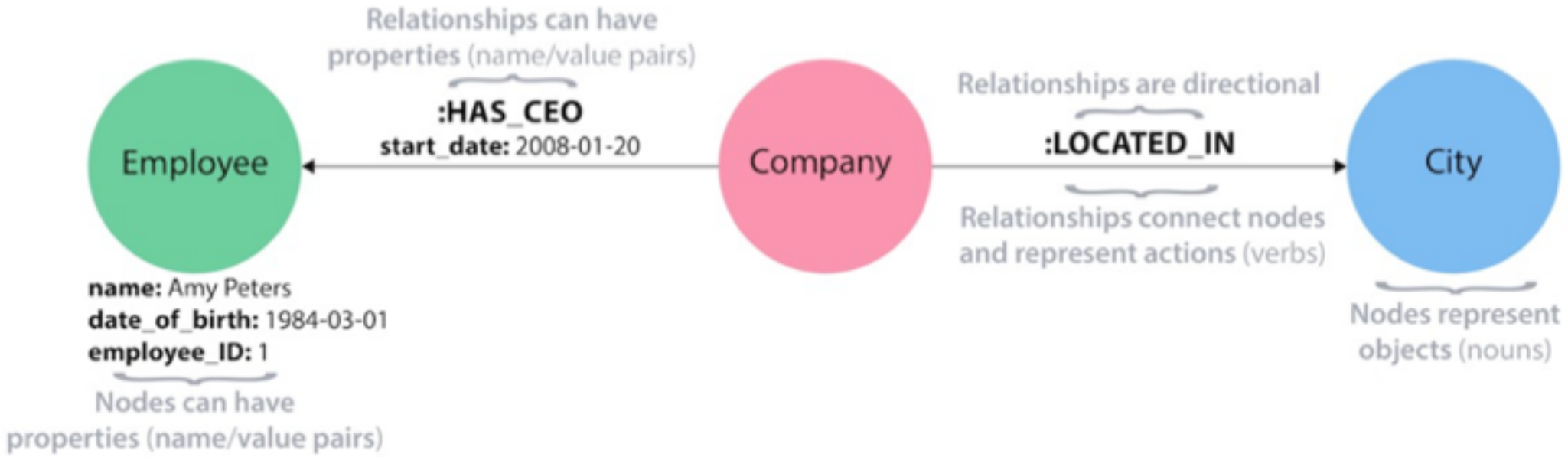
Contains **1276** datasets with **16217** links (as of May 2020)

All dataset contain RDF data with >1000 triples

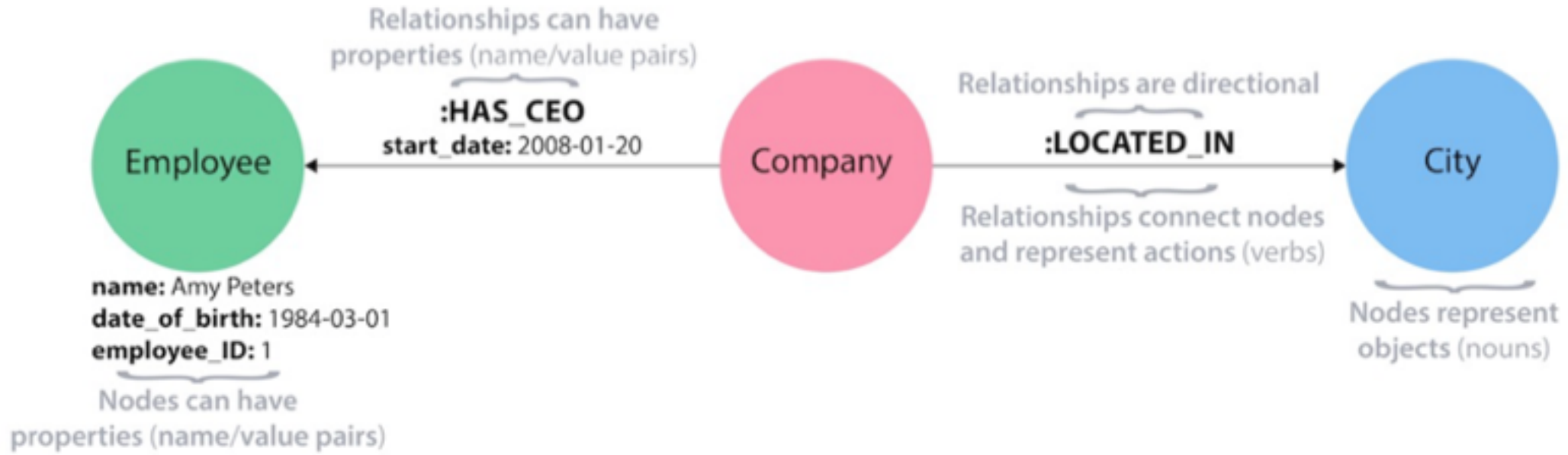
https://lod-cloud.net

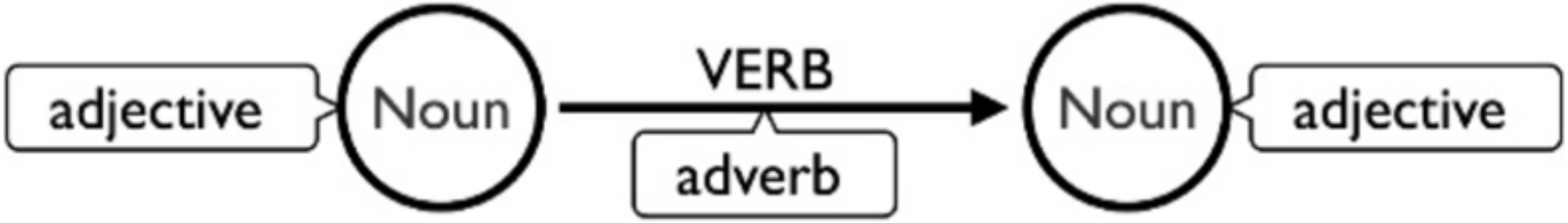


Building blocks of property graph model



Building blocks of property graph model





Why do we need graph databases?

- Can store graphs in RDBMSs, e.g.,
  - Node table
  - Edge table

- But, joins between nodes and edges are common
  - … as the number of hops in a graph increases, this becomes increasingly expensive

- Some problems best suit direct representation in graphs

Centrality measures

- **Degree** centrality—indegree/outdegree
- **Closeness** centrality—average length of all shortest paths
- **Betweenness** centrality—number of times a node acts as a bridge along the shortest paths
- **Eigenvector** centrality—measures the influence of a node in a network (e.g., Google PageRank)

Designing graph databases

- Typical mapping from application's data to a graph:
  - **Entities** are represented as nodes
  - **Connections** are represented as edges between nodes
  - **Connection semantics** dictate directions of edges
  - **Entity attributes** become node properties
  - Link **strength / weight / quality** maps to relationship properties

- Other metadata will also be include in property sets
  - e.g., information about data entry and revision

Graph database systems

- Neo4j
  - https://neo4j.com/developer/graph-database/
- Amazon Neptune
- JanusGraph (scalable, distributed graph database)
- ArangoDB
- OrientDB
- RedisGraph (in memory)
- RDF-specific
  - Virtuoso, BlazeGraph, AllegroGraph
- Others… see https://tinkerpop.apache.org

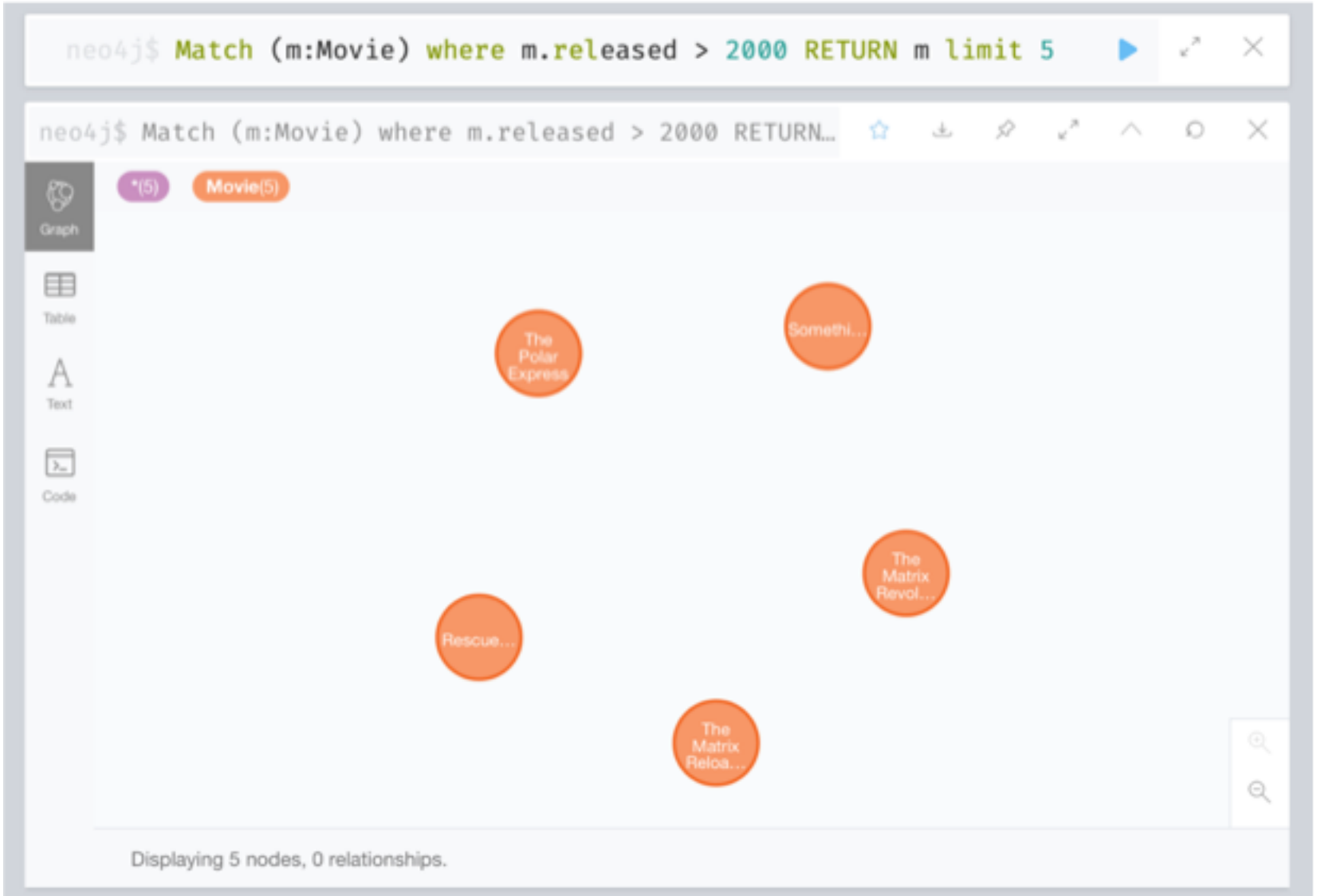
Query languages

- Cypher – developed for neo4j but used by other systems
  - Declarative language like SQL for graph databases
  - Standard CRUD (Create, Read, Update, Delete operations on the elements of the graph)
  - Match patterns in the graph

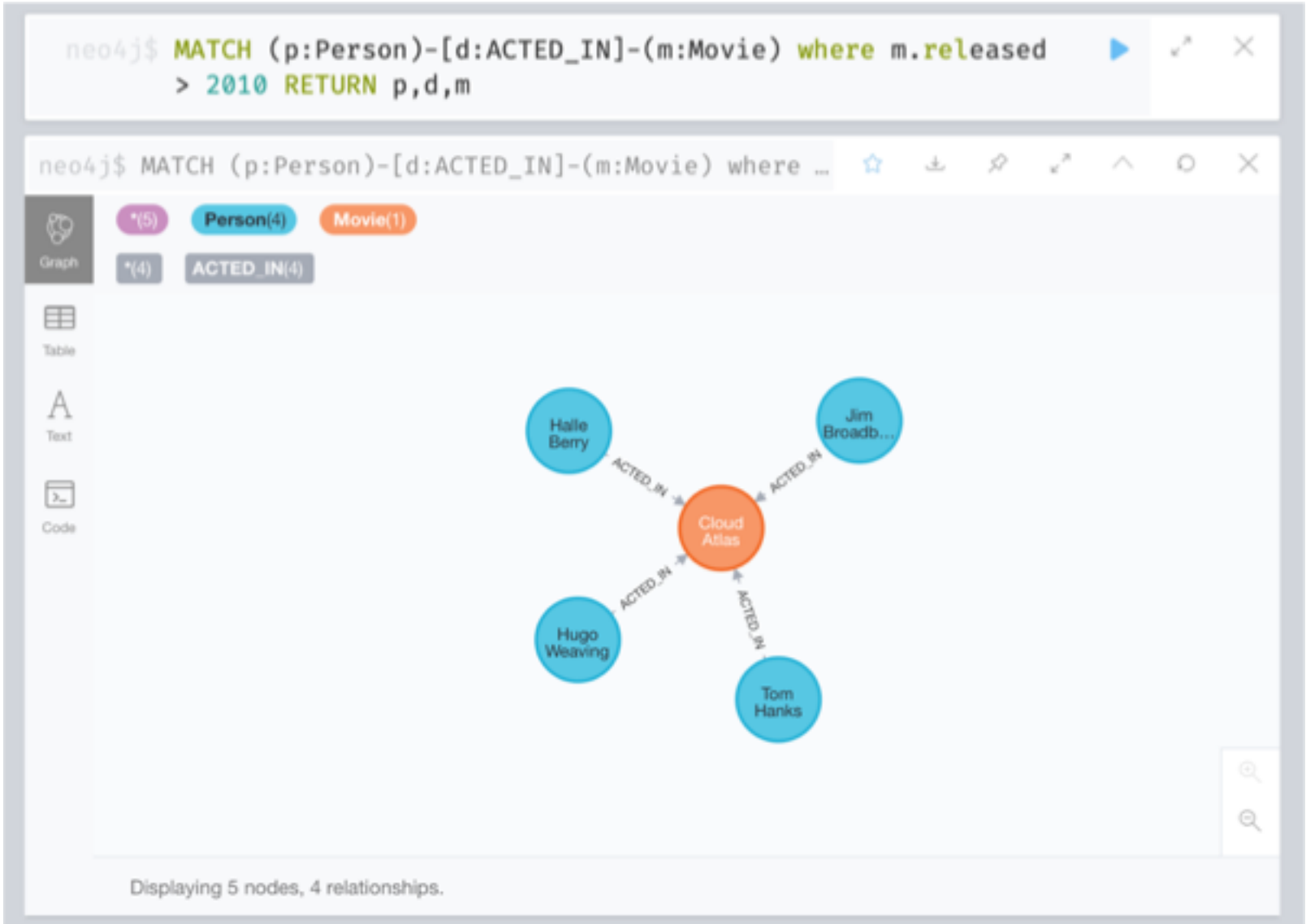
```
(node)-[:RELATIONSHIP]->(node)

(node {key: value})-[:RELATIONSHIP]->(node)
```

- Alternatives:
  - SPARQL – querying RDF graphs
  - Gremlin – graph traversal language for Apache Tinkerpop
  - PGQL – Oracle – mix of SQL SELECT-style with graph matching

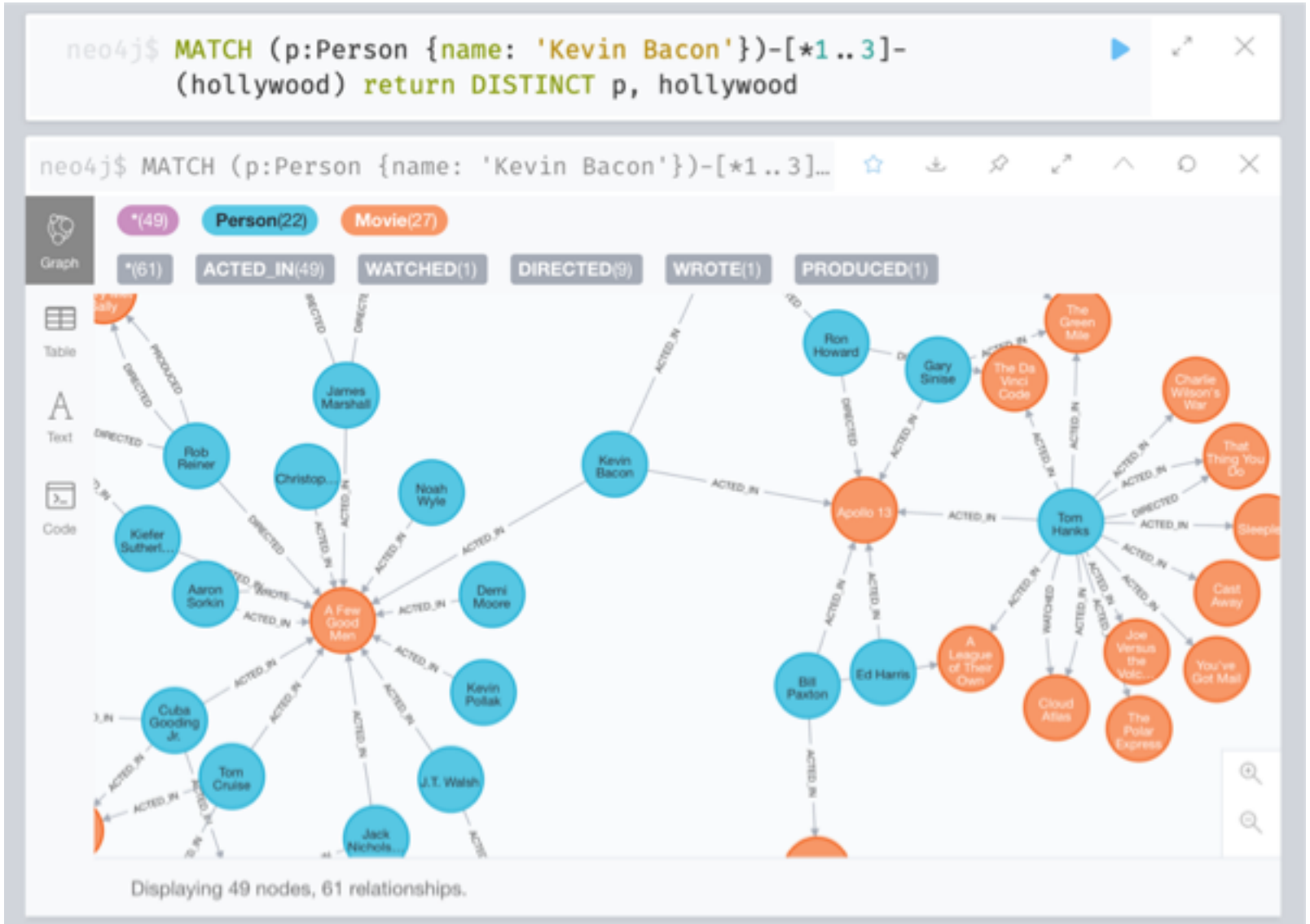
Cypher MATCH and RETURN keywords



Cypher MATCH and RETURN keywords



Complex graph queries



Connecting to neo4j from nodeJS

Shell Copy to Clipboard

```
npm install neo4j-driver
```

JavaScript Copy to Clipboard

```
const neo4j = require('neo4j-driver')

const driver = neo4j.driver(uri, neo4j.auth.basic(user, password))
const session = driver.session()
const personName = 'Alice'

try {
  const result = await session.run(
    'CREATE (a:Person {name: $name}) RETURN a',
    { name: personName }
  )

  const singleRecord = result.records[0]
  const node = singleRecord.get(0)

  console.log(node.properties.name)
} finally {
  await session.close()
}

// on application exit:
await driver.close()
```

UNIVERSITY OF CANTERBURY
Te Whare Wānanga o Waitaha
CHRISTCHURCH NEW ZEALAND

SENG365 Web Computing Architecture: #3 Data persistence: SQL/NoSQL, memory stores, GraphDB

Ben Adams

benjamin.adams@canterbury.ac.nz
310, Erskine Building