Applied Data Science

Data Storage

Dr. Niall Twomey

University of Bristol

Recap of Data Ingress

- The main data formats
  - CSV
  - JSON

Recap of Data Ingress

- The main data formats
  - CSV
  - JSON
- Limitation of storage with these data formats
  - Data not efficiently queryiable
  - Cannot directly merge data
  - These data formats do not scale
    - To huge data sizes
    - To multiple servers

Recap of Data Ingress

- The main data formats
  - CSV
  - JSON
- Limitation of storage with these data formats
  - Data not efficiently queryiable
  - Cannot directly merge data
  - These data formats do not scale
    - To huge data sizes
    - To multiple servers
- Databases are a solution to these problems

Selection of databases

- **Databases should be selected to match the data**
- Familiarity with one database is not a sufficient reason to select that database
- **In this lecture, I will discuss**:
  1. What database (or combination of databases) is suitable for your data
  2. How to criticise the appropriateness of database to a dataset

Database types

- In this lecture we will discuss:
  - Structured Query Language (SQL): PostgreSQL[1]
  - Hu*mongo*us Databases: MongoDB[2]
  - Graph databases: Neo4j[3]

[1] https://www.postgresql.org/
[2] https://www.mongodb.com/
[3] https://neo4j.com/
https://en.wikipedia.org/wiki/Category:Types_of_databases

Database types

- In this lecture we will discuss:
  - Structured Query Language (SQL): PostgreSQL[1]
  - Hu*mongo*us Databases: MongoDB[2]
  - Graph databases: Neo4j[3]
- Other databases (not covered in this lecture):
  - Key/value databases: Memcache, Voldemort, Riak, Redis
    - Very quick acquisition; scales well
    - Queries are limited; most interfaces do not expose full control (Erlang)
  - Column-orientated databases: Hyptertable, HBase, Cassandra
    - Scales up well; versioning built-in; active/helpful community
    - Doesn't scale down; many required components; no sorting/indexing

[1] https://www.postgresql.org/
[2] https://www.mongodb.com/
[3] https://neo4j.com/
https://en.wikipedia.org/wiki/Category:Types_of_databases

- Will provide introduction to the databases mentioned in the previous slide
- Will provide links to resources where for further reading

[4] https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

- Will provide introduction to the databases mentioned in the previous slide
- Will provide links to resources where for further reading

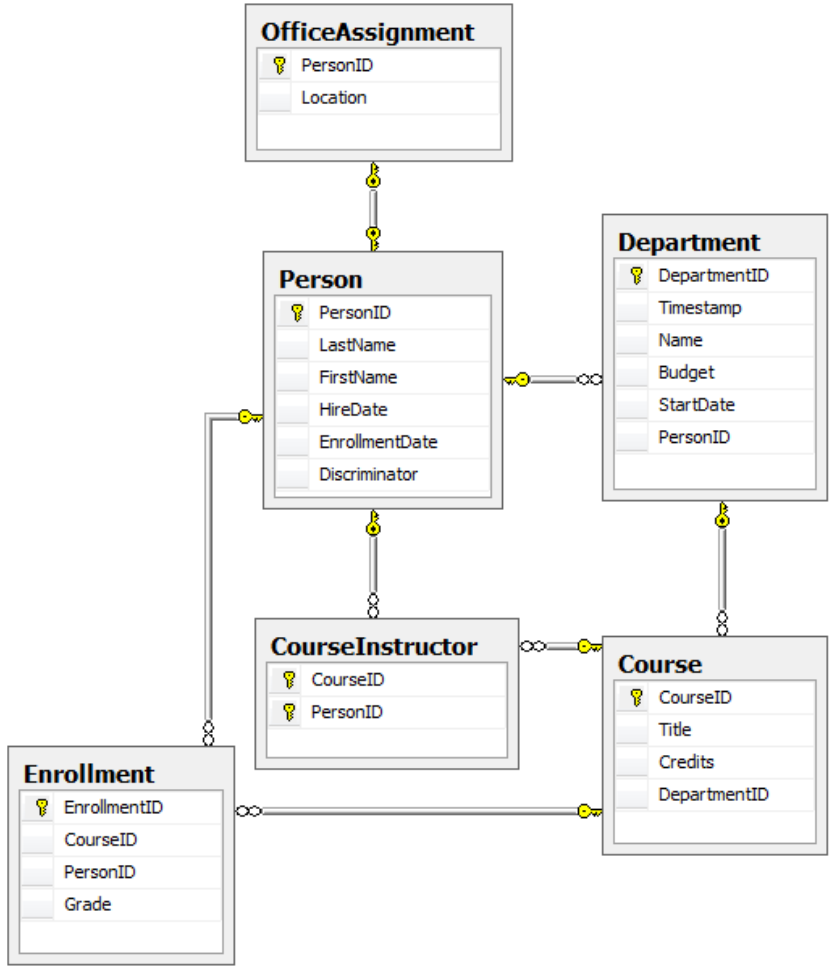
- Interaction: CRUD[4]
  - **C**: Create
  - **R**: Read
  - **U**: Update
  - **D**: Delete
- Some considerations:
  - How flexible is the database?

    In terms of query language, schema, storage, relational
  - Does the database scale to distributed storage/web servers
  - Assessing the performance of databases

[4] https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

SQL: Overview

- Relational
- Consist of rows and columns (like CSV files)
- Strict column types:
  - String, integer, date, binary
- Powerful querying language
- Many implementations available: MySQL, SQLite, PostgrSQL, HSQLDB...



SQL: Create

Database

- Databases manage collections of tables

```
CREATE DATABASE dbname;
```

Table

- Tables manage collections of rows/columns:

```
CREATE TABLE Shippers
(
  ShipperID int
    NOT NULL PRIMARY KEY,
  ShipperName varchar(255),
  Phone varchar(255)
);
```

SQL: Read

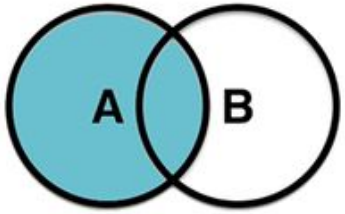
- Example database from W3Schools:
- Stock
  - Food category table
  - Supplier table
- Order
  - Customer table
  - Employee table
  - Shipper table
- Queries:
  - List data
  - Select unique items
  - Join on other table values
  - String wildcards

---

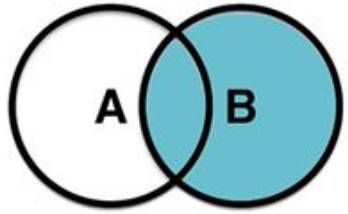
https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_all

GUIA VISUAL

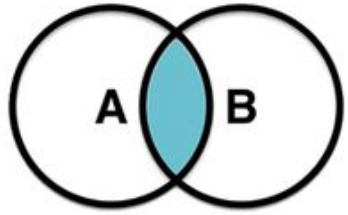
SQL JOINS



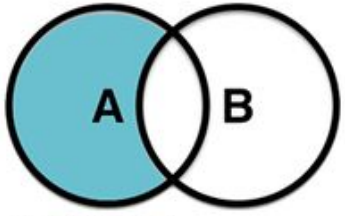

```
SELECT <fields list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



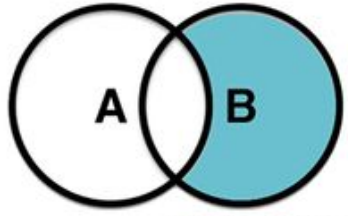

```
SELECT <fields list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



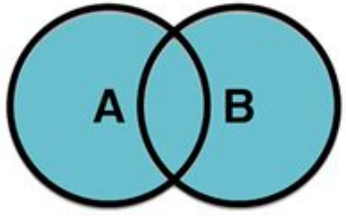

```
SELECT <fields list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



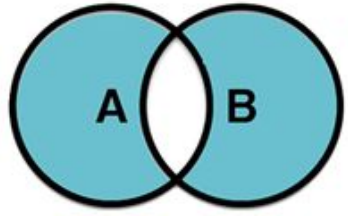

```
SELECT <fields list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <fields list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <fields list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <fields list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL OR B.Key IS NULL
```

SQL: Summary

- W3Schools is an excellent resource to learn more about SQL [5]
- Formatting SQL queries is helpful in debugging query errors [6]
- Indexing columns will speed up queries and joins dramatically
- Advantages
  - Excellent choice at storing relational data, tabular data
  - Provides powerful and flexible querying language
- Disadvantages
  - Does designed with distributed architectures in mind, care must be taken in adjusting schema
  - Strict schema

[5] https://www.w3schools.com/sql/
[6] http://sqllint.com/

MongoDB: Overview

- Terminology change:
  - 'rows' in SQL databases are called 'documents' in MongoDB
  - 'tables' in SQL databases are 'collections' in MongoDB
- All documents are stored in Binary JSON format (BSON)
- MongoDB offers a highly flexible document format
  - Documents are key-value pairs (like `dicts`)
  - Values may themselves be `dicts`
- MongoDBs are schema-less:
  - documents from the same collection do not need to adhere to the same key/value pairs

---

`https://www.tutorialspoint.com/mongodb_terminal_online.php`

Example document:

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

Example document:

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

Example query:

```
# returns documents where the
# ‘firstName‘ field is "John"
db["collection"].find({
    "firstName": "John"
})

# returns documents with age
# fields that are greater than 20
db["collection"].find({
    "age": {
        "$gt": 20
    }
})
```

MongoDB Demo

Demo items

- Create collections
- Add documents
- Query documents
- Return specific key/value pairs
- Exists, distinct, sort

General observations

- No native querying language
- Queries are constructed with expression

Command	Description
$regex	Match by any PCRE-compliant regular expression string (or just use the // delimiters as shown earlier)
$ne	Not equal to
$lt	Less than
$lte	Less than or equal to
$gt	Greater than
$gte	Greater than or equal to
$exists	Check for the existence of a field
$all	Match all elements in an array
$in	Match any elements in an array
$nin	Does not match any elements in an array
$elemMatch	Match all fields in an array of nested documents
$or	or
$nor	Not or
$size	Match array of given size
$mod	Modulus
$type	Match if field is a given datatype
$not	Negate the given operator check

MongoDB: Summary

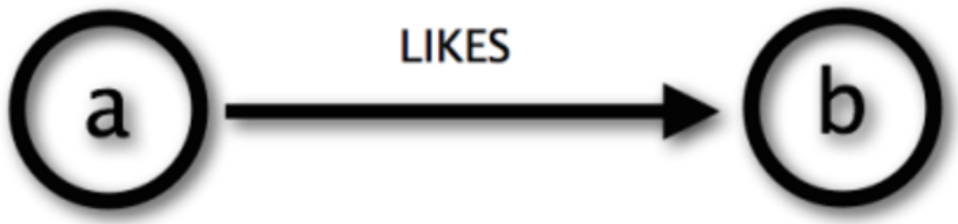
- Documentation on MongoDB website is excellent
- Advantages
  - Copes with huge data and requests
  - Highly flexible storage format
  - Easy to use
  - Complex querying capabilities
- Disadvantages
  - Schema-less documents
  - Susceptible to typo error
  - Cannot joins between collections easily
  - Scaling to multiple clusters not trivial

Neo4j: Overview

- Graph database
  - **SQL**: relational database
  - **MongoDB**: Document-based
  - **Neo4J**: relationships
- Queries performed with the Cypher language[7]
- Core ingredients of Neo4j: nodes and edges
- Properties can be attributed to both nodes and edges
- Excellent built-in administrative/visualisation zone
  - Instantaneous visual feedback assists with understanding the querying language

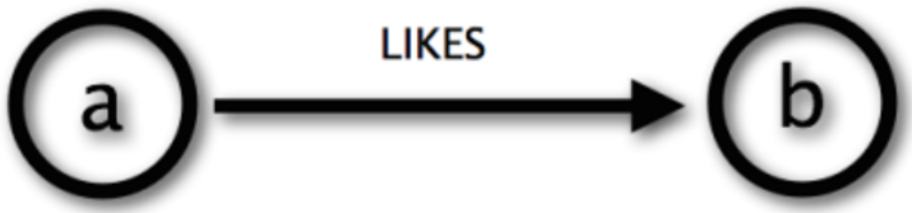
[7] `https://neo4j.com/developer/guide-sql-to-cypher/`
`https://neo4j.com`

Cypher using relationship 'likes'



https://neo4j.com/developer/cypher-query-language/

Cypher using relationship 'likes'



Cypher

```
(a) -[:LIKES]-> (b)
```

https://neo4j.com/developer/cypher-query-language/

Notation:

- `(n1:NCat1)-[e12:ECat]->(n2:NCat2)`
  - `n1`, `e12` and `n2` are available elsewhere in the query (*e.g.* for making connections)

More explicitly:

- ```
  (node_variable_1:NodeCategory1)-
      [edge_variable:EdgeCategory]->
          (node_variable_2:NodeCategory2)
  ```
  ```

Neo4j Demo

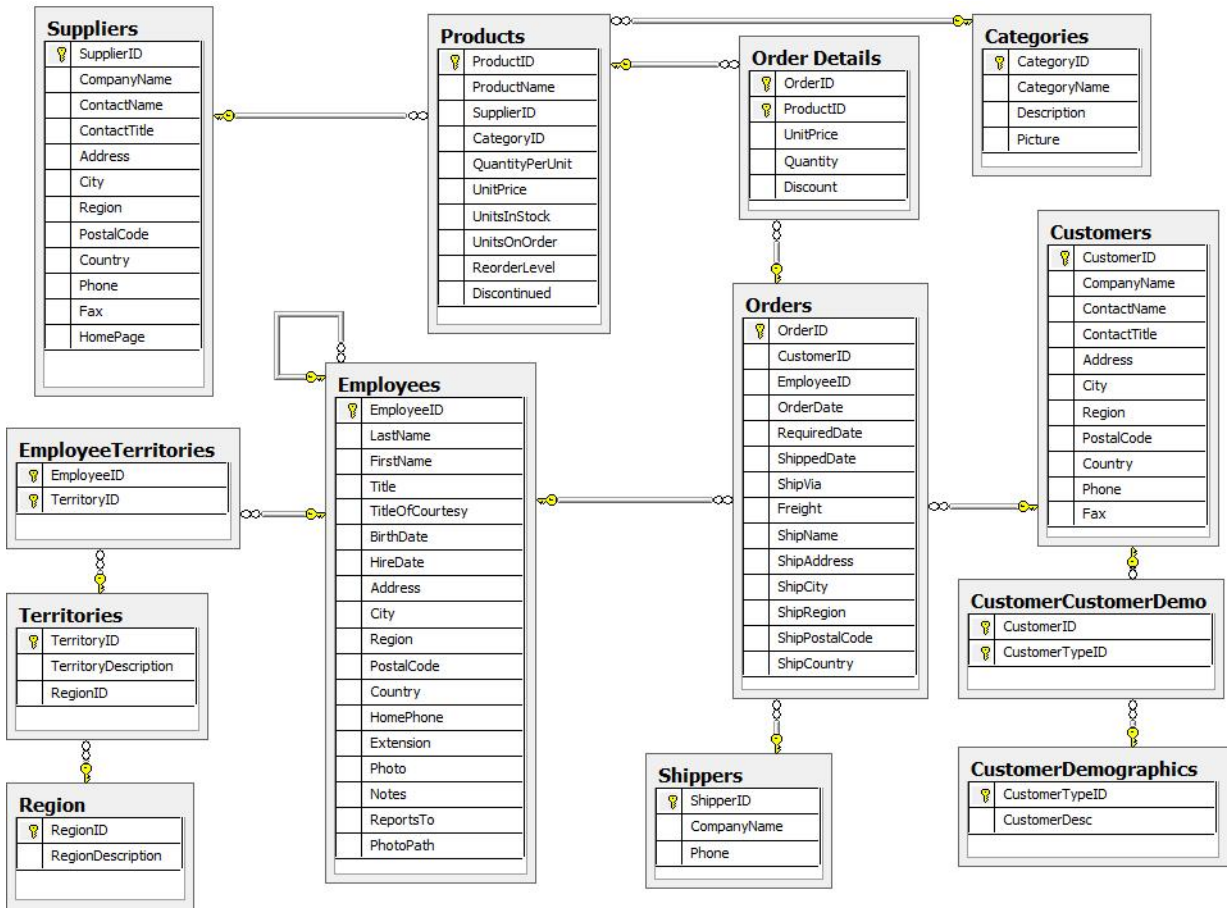
Demo items

- Create nodes
- Create edges
- Make queries
- Join queries
- Query through graph

General observations

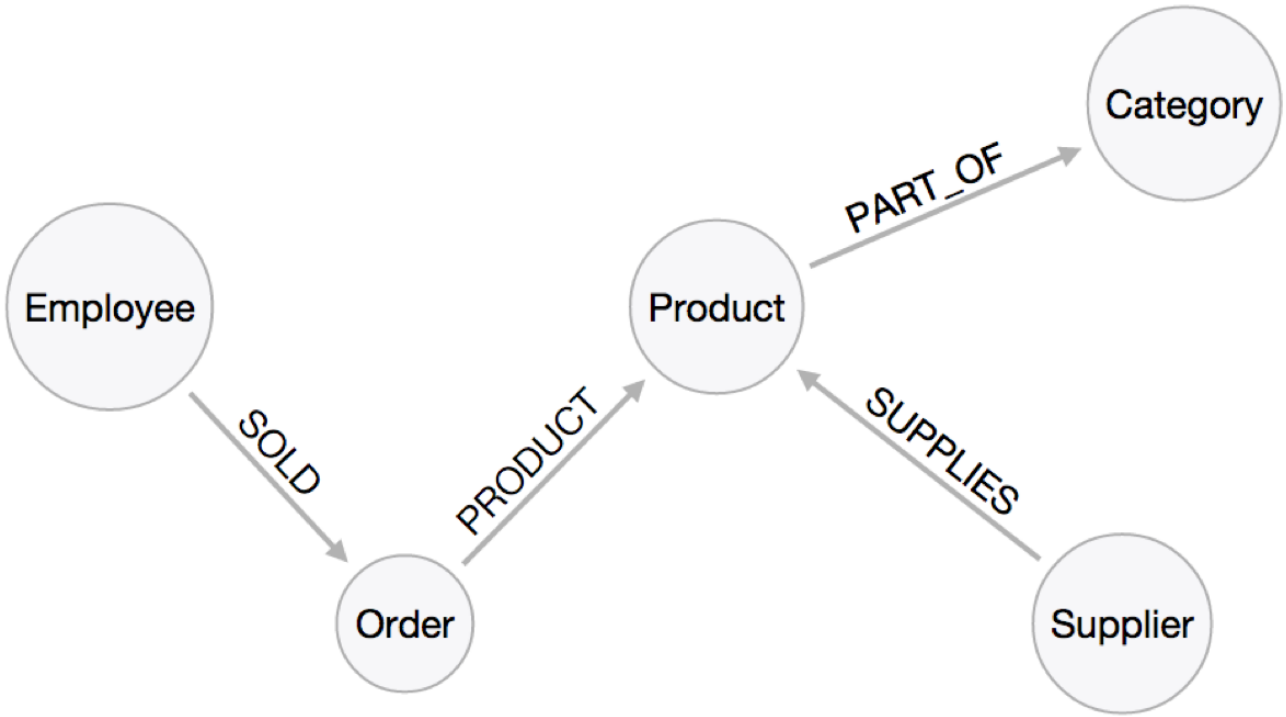
- Queries walk the graph

Why Neo4j (I)



https://neo4j.com/developer/guide-sql-to-cypher/

Why Neo4j (II)



https://neo4j.com/developer/guide-sql-to-cypher/

Neo4j Summary

Advantages

- Great choice for unstructured data
- No schema
- No relational constraints
- In-built graph functionality
- Simplifies complex relational querying
- Traversal is constant time (unlike joins)

Disadvantages

- No self-references
- Replication only supports full graphs (not subgraphs)

Summary

- Not all databases are suitable for all datasets
- Discussed three different options in detail
- Provided overview on where these database options are appropriate

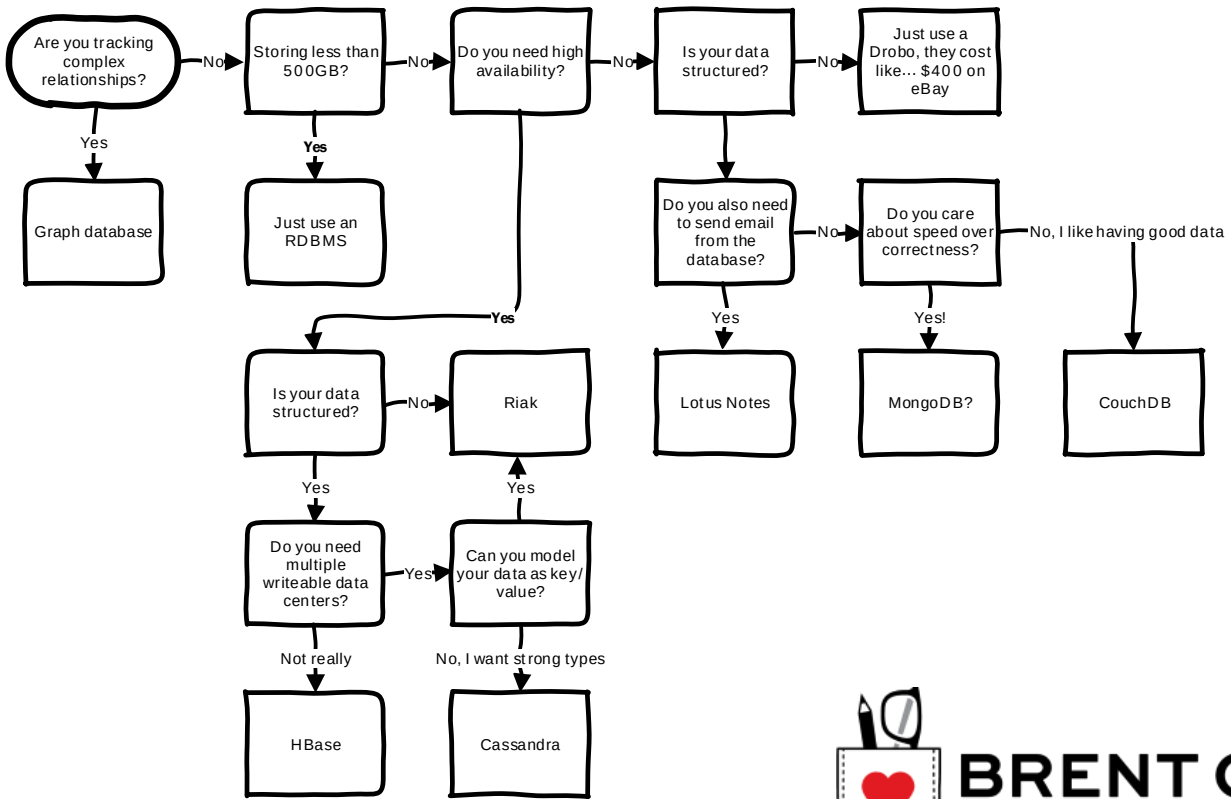
Summary

	Main Differentiator	Weaknesses
MongoDB	Easily query *Big Data*	Embed-ability
CouchDB	Durable and embeddable clusters	Query-ability
Riak	Highly available	Query-ability
Redis	Very, very fast	Complex data
PostgreSQL	Best of OSS RDBMS model	Distributed availability
Neo4j	Flexible graph	BLOBs or terabyte scale
HBase	Very large-scale, Hadoop infrastructure	Flexible growth, query-ability

- Many other useful tables regarding database capabilities are available in the book '*Seven Databases in Seven Weeks*'[8] (relational, query-ability, MapReduce, versioning, replication, security etc)

[8] https://www.amazon.co.uk/Seven-Databases-Weeks-Modern-Movement/dp/1934356921

Which database should I use?



BRENT OZAR UNLIMITED

http://www.brentozar.com/wp-content/uploads/2020/07/database-decision-diagram.pdf