Introduction to NoSQL

Kevin Swingler

What is NoSQL?

- Term coined in 2009 by Johan Oskarrson who needed a short hashtag for a meeting he was arranging
- Not a good name for a number of reasons
  - Says what it is not, not what it is
  - Some NoSQL systems use SQL!
  - Best used as a label for new ways of organising databases

Why NoSQL?

- The relational database has a lot of advantages
- They are mature and work well and many people have the skills to run them
- Transactions and integrity mean they can be relied upon to be correct
- Pretty much every programming language supports them

Yes, but Why NoSQL?

- Object Relational Impedance Mismatch
  - We like to program in object oriented languages
  - The objects don't match well with the structure of the RDB
- Shared application databases
  - One database may serve many applications, but all have to use the relational model to access data at the table level

So?

- New models such as Service Oriented Architectures share data in a format that is divorced from the database
- XML or JSON, for example
- There are many advantages to this, so rather than use RDBMS for everything, developers can choose the right DB for the right application

And ...

- Traditional RDBMSs can be made to run over clusters, but weren't really designed for it
- Problems with integrity arise if the DB is spread across a cluster, and licence costs can be prohibitive across many machines
- If you need to run across a cluster, which is cheap and desirable for big systems, you need a different DBMS

And...

- Many collections of data do not fit into a relational schema
- Constraints of normal form can be too restrictive
- Different records might need to contain different fields, or different numbers of fields
- A field might need to hold several values
- Availability and low latency might be more important than absolute integrity

Finally ...

- With agile methods in programming becoming popular, it makes sense to have a database that is easy to alter during development
- You can add or remove fields from a RDB, but the process can be difficult
- NoSQL databases are designed to be more flexible in what is stored

New Databases

- Some new ways of organising databases have emerged that challenge the strict schemata, normal forms and ACID transactions and integrity of RDBMSs
- These have become collectively known as NoSQL databases

Examples

- We will study three types of NoSQL database on this course, and a commercial example of each:
  - MongoDB is a Document Database
  - Cassandra is a Column Database
  - Neo4J is a Graph Database
- These are all available in free versions for anyone to download and use

Some Data

- Imagine we decide to build a product review database:

Product: iPhone 5
Price: £500
Camera: Fine
Screen: Very good
Accessories: Headphone, Case, ...

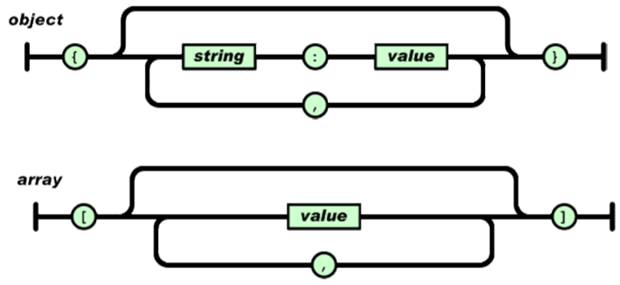
Product: iPhone 5
Price: £500
Camera: Excellent
Screen: Poor in sunshine
Operating system: Easy to use

Features

- Different products have different fields
- Some fields have several values
- Products are related to other products as accessories
- A new product may have new qualities (fields) not yet in the database
- The same product may appear many times with different values and fields

JSON

- In case you haven't seen it, here is a quick introduction to JavaScript Object Notation



Example Review in JSON

```
{
Id:847543,
Name:iPhone5,
Features:[GPS,Retina Display,Siri],
Reviews:[
{Reviewer:458743,Date:12.4.1013,Speed:Slow},
{Reviewer:636534,Date:2.5.1013,Camera:Great},
]
}
```

Choosing a DB

- With data exchanged by XML or JSON and a new set of DBs to choose from, developers can select the right DB for the task, rather than always use a RDB.

Relational Data Models

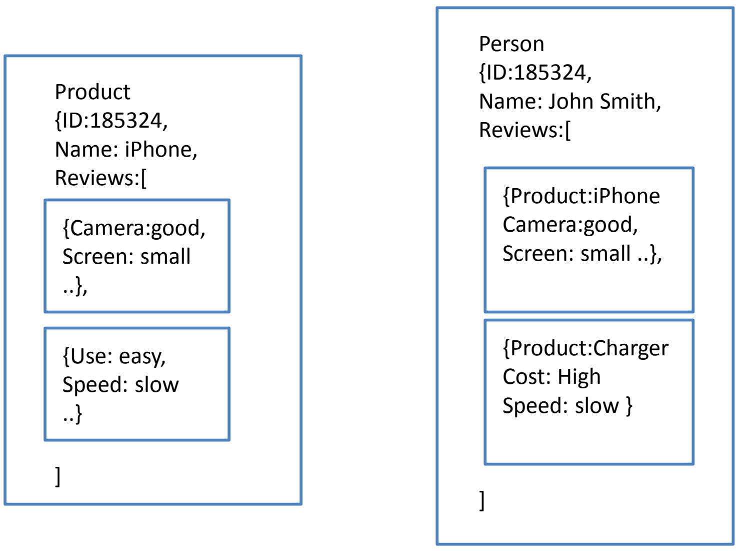
- The relational data model, used by relational databases is the most common model for databases
- Tables contain fields, which correspond to variables (Name) and rows, which correspond to entities (Bill Smith)
- A field on one table (UserID) can relate to a row in another table ...

Aggregate Data Models

- There are a number of alternatives to the relational model
  - Documents
  - Column Families
  - Graph
- Treat units of data in different ways, a unit might be a person, or a document, or a product review, etc.
- Sometimes called 'aggregates'

Aggregate Design

- There is a new freedom, away from ER models
- But with more choice come more decisions...
- For our product review database, potential units to aggregate are:
  - Users – keep all a person's reviews together
  - Products – keep all the reviews of a product together



What are Common Queries?

- Which design you choose depends on a few things, for example, what is the most common query?
  – List all reviews from a given customer
  – List all reviews of a given product
  – Find all products that are 'fast'

NoSQL ACID

- Atomic, Consistent, Isolated and Durable: Relational databases key strength and requirement for transactions
- NOT a cornerstone of NoSQL databases, though available in some of them
- Aggregate design can help impose atomicity – an aggregate is updated in a single transaction
- More on this later ...

Distribution

- What can you do when the server running your database can’t cope with the volume of traffic imposed on it?
- Buy a bigger, faster machine – gets expensive and there is a limit, also a single point of failure
- Buy lots of commodity machines (any old cheap and replaceable machine) and run a cluster

Distributed Aggregates

- With a relational database, distribution can be problematic as a join might end up being made across multiple nodes on a cluster
- With an aggregate model, all the data about a unit of interest (say, a customer) is kept together and distribution separates different customers on different machines

Two Methods of Distribution

- Sharding
  - Each server has a small bit of a very big database on it
  - Good if the database is large and can be naturally shared out
- Replication
  - Each shard is copied onto (two) other machines for resilience (and a little latency gain)
- There will be a lecture on this later ...

Consistency

- If you replicate a database across different servers, you can read from any one of them with confidence until you have to write
- Then, you can either write to all the copies, which takes time, or to some of them, which leaves the entire set of them inconsistent
- What happens if the next read is from a copy that was not updated?

Consistency

- Relational databases take consistency seriously, sometimes at quite a cost
- Part of the NoSQL approach is that consistency may be relaxed in return for speed, or availability or partition tolerance
- There is the idea of eventual consistency
- A distributed database can be allowed to diverge for a while as long as things are put right in the end

Eventual Consistency

- If you pay some money into your bank account, you don’t want some cash machines to know its there and others not to
- But if you post something on Facebook, you might put up with the fact that a friend in the UK can see it for a moment while an American friend, looking at a slightly older replication cannot.
- He’ll see it eventually

CAP Theorem

- CAP theorem states that among Consistency, Availability, and Partition tolerance, you can only ever guarantee two.
- As soon as you partition a database, you have to accept that data might either:
  - Become inconsistent – one replication holds different data from another
  - Become unavailable – to keep consistency, we must propagate updates, which takes time

Map-Reduce

- Distributed databases lend themselves to a Map-Reduce programming model
- Map tasks read data from aggregates one row at a time, so can be done in parallel
- We will look in more detail at Map Reduce on a database in a future lecture.

Summary

- NoSQL removes a number of the key restrictions (or strengths!) of the relational database model to allow:
  - Flexible data models
  - Relaxed consistency constraints
  - Distributed databases on clusters
  - Map Reduce across clusters