University of Glasgow

Data Storage and Retrieval

Lecture 10

Beyond The Relational Model

NoSQL

Dr. Graham McDonald

Graham.McDonald@glasgow.ac.uk

Relational Databases: Recap

- Data modelling:
  - Requirements analysis → conceptual design → logical design → schema refinement → physical design/tuning
- Query patterns:
  - **Selection** based on complex conditions, **projection**, **joins**, **aggregation**, derivation of new values, recursive queries, …
- Query languages:
  - SQL (based on relational algebra)
- Examples:

University of Glasgow

Relational Databases: Concurrent Access

- Transactions:
  - Flat sequence of operations (READ, WRITE, COMMIT, ABORT)

- **ACID** properties:
  - Atomicity: partial execution is not allowed (all or nothing)
  - Consistency: transactions turn one valid database state into another
  - Isolation: uncommitted effects are concealed among transactions
  - Durability: effects of committed transactions are permanent

University of Glasgow

Data is Abundant

- **Big Data**
  - **Volume**: terabytes to zettabytes
  - **Variety**: structured to structured and unstructured data
  - **Velocity**: batch processing to streaming data
  - (also Veracity etc.)

- Huge amounts of online activity
  - Population, time, devices, etc.
  - Companies, web applications, etc.

University of Glasgow

Databases: Recent(ish) Trends

- Modern software paradigms
  - Software as a Service (SaaS)
  - Platform as a Service (PaaS)
  - Infrastructure as a Service (IaaS)

Traditional relational databases often do not fit very well with the needs of recent trends.

- Data is changing:
  - Formats are becoming *unknown* or *inconsistent*
  - Updates are often not so frequent (data is just replaced)
  - Used to be linear growth, now exponential
  - Often strong consistency is no longer mission-critical
  - Read requests can vastly outnumber write requests

University of Glasgow

NoSQL: What is it?

- What is NoSQL?
  - The terminology is not used consistently.
  - No SQL?, Not only SQL?, …
- NoSQL is an accidental term with no precise meaning.
  - First used in 1998 for a relational database without SQL
  - By 2009 it had evolved to mean non-relational databases.
- Generally thought of as an alternative when a relational database is a bad fit.

University of Glasgow

Advantages of NoSQL

- Scalability:

NoSQL databases use *horizontal scale-out* to makes it easy to add or reduce capacity quickly and non-disruptively using commodity hardware.

This eliminates the very large costs and complexity of *manual sharding* that is necessary when scaling relational DBMS.

University of Glasgow

Advantages of NoSQL

- Performance:

Enterprises can increase performance with NoSQL databases by simply adding additional commodity resources.

Enables organizations to deliver reliably fast user experiences with a predictable return on investment for the added resources.

University of Glasgow

Advantages of NoSQL

- High Availability:

NoSQL databases are generally designed to ensure the availability of data and avoid many of the complexities that are associated with a typical relational DBMS.

Some “distributed” NoSQL databases deploy a masterless architecture that automatically distributes data equally among multiple resources to ensure the availability of the data during failures.

University of Glasgow

Advantages of NoSQL

- Flexible Data Model:

NoSQL provides application developers with more options to select solutions to match the data types and query options that are the most appropriate to their specific application use case, instead of being constrained by a single database schema.

University of Glasgow

Common Types of NoSQL Databases

- Main types:
  - **Key-value** stores
  - **Document** stores
  - **Wide column** (column family, column oriented, …) stores
  - **Graph** databases
- Other types
  - **Object** databases
  - Native **XML** databases
  - **RDF** stores
  - …

University of Glasgow

Key-Value Stores

- The simplest NoSQL database type, essentially a simple hash table (mapping)

- **Key-value pairs**
  - Key (id, identifier, primary key)
  - Value: binary object, black box for the database system
- Query patterns
  - Create, update or remove value for a given key
  - **Get value** for a given key
- Characteristics
  - Excellent performance, easily scaled, …
  - Not for complex queries or complex data

University of Glasgow

Key-Value Stores

- Suitable use cases
  - Session data, user profiles, user preferences, shopping carts, …
  - i.e. when values are only accessed via keys

- When not to use
  - Relationships among entities
  - Queries requiring **access to the content of the value part**
  - **Set operations** involving multiple key-value pairs

University of Glasgow

Document Stores

- Typically stores *self-describing* documents organised into collections
- Similar to key-value stores, but a value is a single document that stores all data related to a specific key.
  - Documents are usually hierarchical tree structures, e.g. XML, JSON, BSON, etc.
  - Scalar values, maps, lists, sets, nested documents, …
    - Identified by a **unique identifier** (key, …)
- The value part is examinable!

University of Glasgow

Document Stores

- Query patterns:
  - Create, update or remove a document
  - **Retrieve documents according to complex query conditions**
- Suitable use cases
  - Event logging, content management systems, blogs, web
  - i.e. for structured documents with a similar schema
- When not to use
  - **Set operations** involving multiple documents
  - If the document structure changes frequently

amazon DynamoDB

mongoDB. CouchDB relax RethinkDB Couchbase terrastore

University of Glasgow

Wide Column Store

- Stores data in tables with rows and columns similar to the relational model, but:
  - The names and the formats of columns can vary across tables.
- Column family (table)
  - collection of **similar rows** (not necessarily identical)
- Row is a collection of **columns**
  - Should encompass a group of data that is accessed together
  - Associated with a unique **row key**
- A column has a **column name** and **column value** (+metadata)
  - Scalar values, but also **flat sets, lists or maps** may be allowed

University of Glasgow

Wide Column Store

	ecosystem				
	Name	Location		Address	
1	IPX	city	Glasgow	Protocol	https
		Street	Byres Rd	Domain	caf.com
		Number	32		
2	JPM	City	Dundee	Protocol	https
		Street	Main St	Domain	jpm.com
		Number	109		

Row Key

Column Family

Colum

University of Glasgow

Wide Column Store

- Querry patterns
  - Create, update or remove a row within a given column family
  - Select rows according to a row key or simple conditions

- Suitable use cases
  - Event logging, content management systems, blogs, …
  - i.e. for structured flat data with similar schema

- When not to use
  - When ACID transactions are required
  - Complex queries: aggregation (SUM, AVG, …), joining, …

University of Glasgow

Graph Databases

- Graph databases use graph structures to store, map, and query relationships.
  - Directed or undirected graphs.

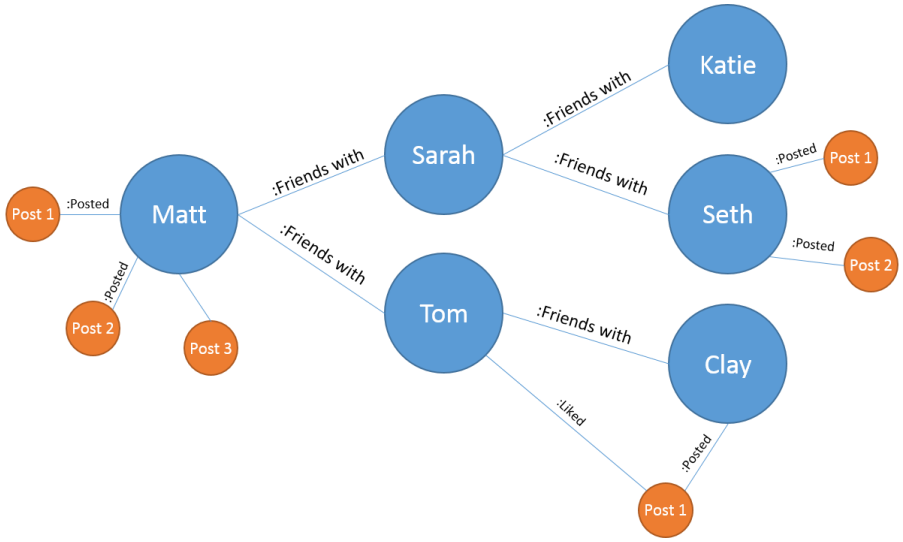
- Collections of
  - Nodes (vertices) for real-world enties, and
  - Relationships (edges) connecting the nodes
  - Both can have additional associated properties

- **Non-transactional** = small number of very large graphs
- **Transactional** = large number of small graphs

University of Glasgow

Graph Databases

- Query patterns
  - Create, update or remove a node / relationship in a graph
  - **Graph algorithms** (shortest paths, spanning trees, …)
  - General **graph traversals**
  - **Sub-graph** queries or **super-graph** queries
  - Similarity based queries (approximate matching)



University of Glasgow

Graph Databases

- Suitable use cases:
  - Social networks, routing, dispatch, and location-based services, recommendation systems, ..
- When not to use
  - When extensive batch operations are required
    - If multiple nodes / relationships are to be affected
  - When only **very** large graphs to be stored
    - Graph distribution can be difficult or impossible

University of Glasgow

RDF Stores

- RDF (Resource Description Framework) is a mark-up language.
  - Used for describing ontologies and resources on the Web
  - Written in XML.
- RDFtriples
  - Components: <**subject**>, <**predicate**>, <**object**>
    - Each usually stored as globally unique URIs
  - Each triple represents a statement about a real-world entity
  - Triples can be viewed as **graphs**
    - Vertices for subjects and objects
    - Edges directly correspond to individual statements
- Query language
  - **SPARQL**: *SPARQL Protocol and RDF Query Language*

University of Glasgow

RDF Stores

BASE Properties and Horizontal Scaling

University of Glasgow

BASE Principles

- Recall ACID for relational DBMS desired properties of transactions:
  - Atomicity, Consistency, Isolation, and Durability

- NoSQL databases typically conform to BASE properties
  - Basically Available,
  - Soft state,
  - Eventually consistent

University of Glasgow

BASE Principles

- By giving up ACID constraints NoSQL databases can achieve much higher performance and scalability

- NoSQL databases differ in how much they give up
  - e.g. most of the systems call themselves "eventually consistent", meaning that updates are eventually propagated to all nodes
  - but many of them provide mechanisms for some degree of consistency, such as multi-version concurrency control (MVCC)

University of Glasgow

Cap Theorem

- A system can have only two out of three of the following properties:
  - Consistency,
  - Availability
  - Partition tolerance
- CAP theorem is a basic principle used when deciding on a NoSQL database
  - NoSQL databases generally give up consistency
  - However, the trade offs are complex

University of Glasgow

Horizontal Scalability

- The ability to distribute both the data and the load of simple operations over many servers
  - With no RAM or disk shared among the servers
  - Referred to as Shared-Nothing Horizontal Scaling

- Vertical scaling is where a database system utilizes many cores and/or CPUs that share RAM and disks
  - Some NoSQL databases provide both vertical and horizontal scalability

University of Glasgow

Horizontal vs. Vertical Scaling

- Effective use of multiple cores (vertical scaling) is important but the number of cores that can share memory is limited
- Horizontal scaling is generally less expensive
  - Can use off-the-shelf commodity servers
- Horizontal scaling requires breaking a sequential piece of logic into smaller pieces so that they can be executed in parallel across multiple machines
  - Can cause more processing costs since you have to pass copies of the data.