COMP207
Database Development

Lecture 25

Beyond Relational Data:
NoSQL Databases (Part 2)

NoSQL Databases

- NoSQL databases:
  - **Distributed**
  - Focus on **availability**, **high performance** & **fault-tolerance**
  - Give up some characteristic features of relational DBMS such as full ACID transactions

Recall the CAP Theorem

Key-Value Stores

- Collection of table, tables = key-value pairs

Key	Value
User 1	{ “name”:“John Smith”, “items”:{ “1”:{“name”:“product 1”, “quantity”:3}, “2”:{“name”:“product 2”, “quantity”:1}, … … }
…	…

Essentially an index!

- Simple access mechanism:
  - **find(k)**: returns value for key **k**
  - **write(k, v)**: inserts value **v** under key **k**
- Fast due to index on key, no further indexes & no transactions

Example Voldemort Session

- Terminal:

```
> bin/voldemort-shell.sh test tcp://localhost:6666
Established connection to test via [tcp://localhost:6666]
> put "hello" "world"
> get "hello"
version(1:2) ts:1511089049077: "world"
> delete "hello"
> get "hello"
null
```

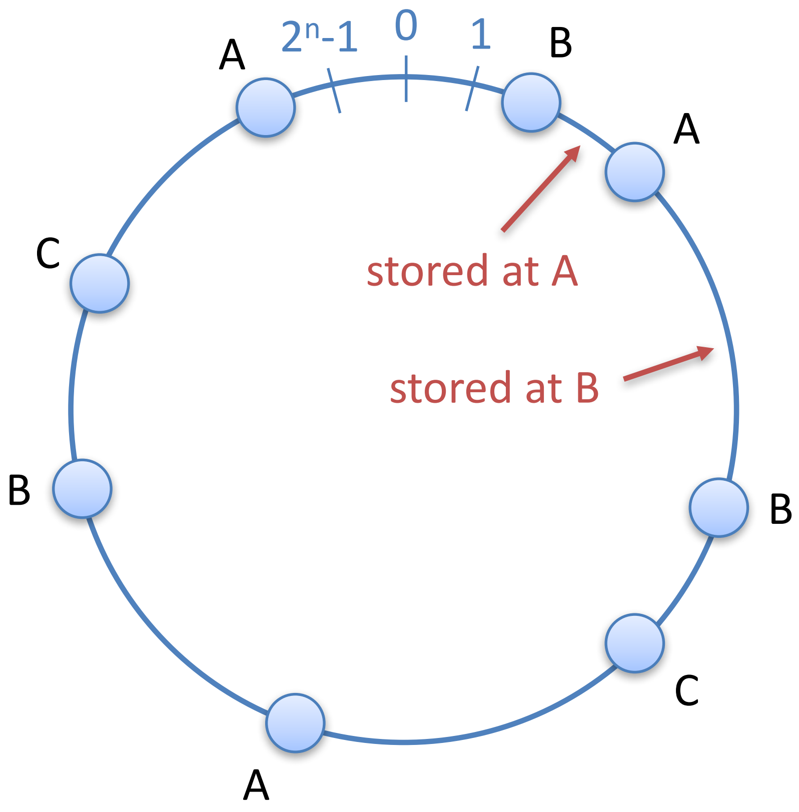
- Java:

```
client = … // connect to server
client.put("hello", "world");
Versioned<String> val = client.get("hello");
System.out.println("Value for 'hello': " + val);
client.delete("hello");
```

More at http://www.project-voldemort.com/voldemort/ (see Quickstart & Design)

DynamoDB & Voldemort Techniques

- Techniques:
  - Distributed hash table
  - Replication
  - Versioning and conflict-resolution using vector clocks



- Access:
  - Reads return local data
  - Writes update local data first, which will then be propagated
- Questions:
  - Why not assign a node to each number?
  - Why reuse nodes?

Document Stores

Document Stores

- Databases that store collections of “documents”
  - Document = semistructured data associated with an object ID

Collection ”restaurants”

ObjectID	Value

1028434	{ “name”: “Morris Park Bake Shop”,
	  “address”: {
	    “coord”: [-73.856077, 40.848447],
	    “street”: “Morris Park Ave”,
	    “zipcode”: “10462” },
	  “grades”: [
	    { “date”: 1393804800000, “grade”: “A”, “score”: 2 },
	    { ”date”: 1299715200000, “grade”: “B”, “score”: 14 }] }

…	{ “name”: “Wendy’S”,
	  “address”: {
	    “coord”: [-73.961704, 40.662942],
	…

…	…

often assigned automatically

Documents represented in JSON

Typical Use Case

- Restaurant information web service
  - Filter collection of restaurants by location, ratings, etc.
  - Data associated with restaurants might vary
  - User requests typically require all data associated with a restaurant, but do not need to join data from different restaurants

Rules out key-value stores

Collection ”restaurants”

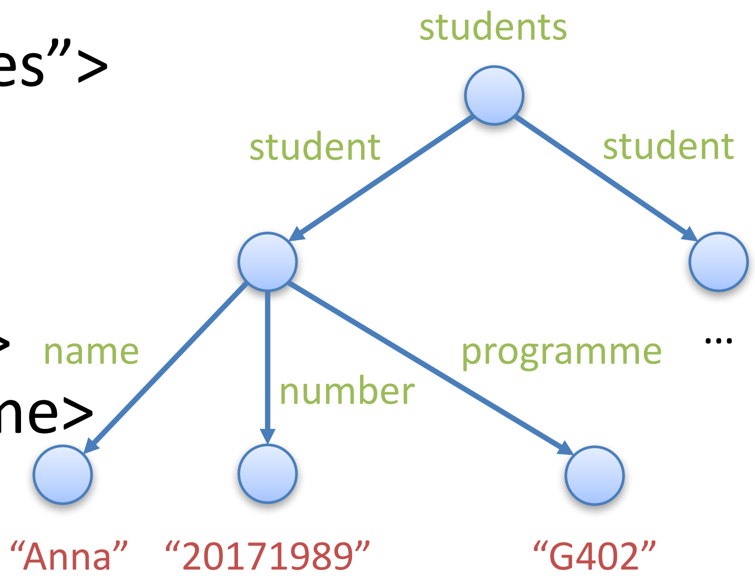
ObjectID	Value

1028434	`{ “name”: “Morris Park Bake Shop”,`
	`  “address”: {`
	`     “coord”: [-73.856077, 40.848447],`
	`     “street”: “Morris Park Ave”,`
	`     “zipcode”: “10462” },`
	`  “grades”: [`
	`     { “date”: 1393804800000, “grade”: “A”, “score”: 2 }, …`

…	`{ “name”: “Wendy’S”, …`

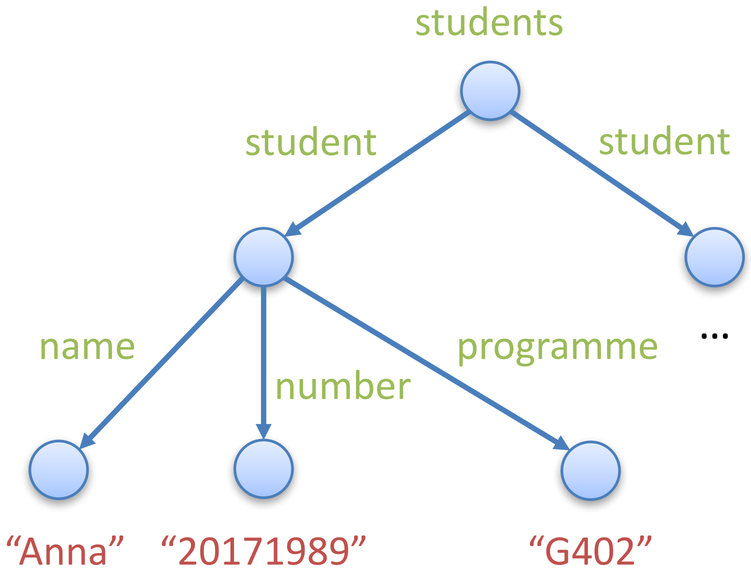
JSON vs XML

```
<?xml version="1.0" standalone="yes">
<students>
  <student>
    <name>Anna</name>
    <number>20171989</number>
    <programme>G402</programme>
  </student>
  <student>
    <name>John</name>
    <number>20174378</number>
    <programme>G702</programme>
  </student>
  …
</students>
```



JSON vs XML

```
{"students":[
        {"name": "Anna",
        "number":20171989,
        "programme":"G402"},
        {"name": "John",
        "number":20174378,
        "programme":"G702"},
        …
]}
```



Case Study: MongoDB

https://www.mongodb.com

- Stores documents in variant of JSON (previous slide)
  - Creating/managing collections

```
db.createCollection("students")
```

  - Insert/update/delete documents

```
db.students.insert(
    {name: "Anna", studentID: 20171989, year: 2})
```

  - Finding documents

More flexible than key-value stores

```
db.students.find({year: 2})

db.students.find({year: {$lt: 3}}).sort({year:1})
```

  - Aggregation, …

Indexes in MongoDB

- Also: indexes on one or more fields of documents (in contrast to key-value DBs)

Collection ”students”

ObjectID	Value
1028434	{ “name”: “Anna”, ”studentID”: “1234”, “year”: 2, “programme”: “G402” }
…	…

- db.students.createIndex({ name: 1 })
- db.students.createIndex({ programme: 1, year: -1 })

MongoDB Techniques

- Sharding (horizontal fragmentation)
  - Collections are split into **horizontal fragments**
  - Based on **shard key**: indexed field, exists in all documents

ObjectID	Value
`1028434`	`{ “name”: “Anna”, ”studentID”: “1234”, “year”: 2 }`
…	…

- Replication
  - Horizontal fragments of collections are replicated
  - Master-slave approach:
    - Primary copy (master) and secondary copies (replicas)
    - Updates: on master, then delegated to replicas
    - Reads: delegated to master (default) or to any replica

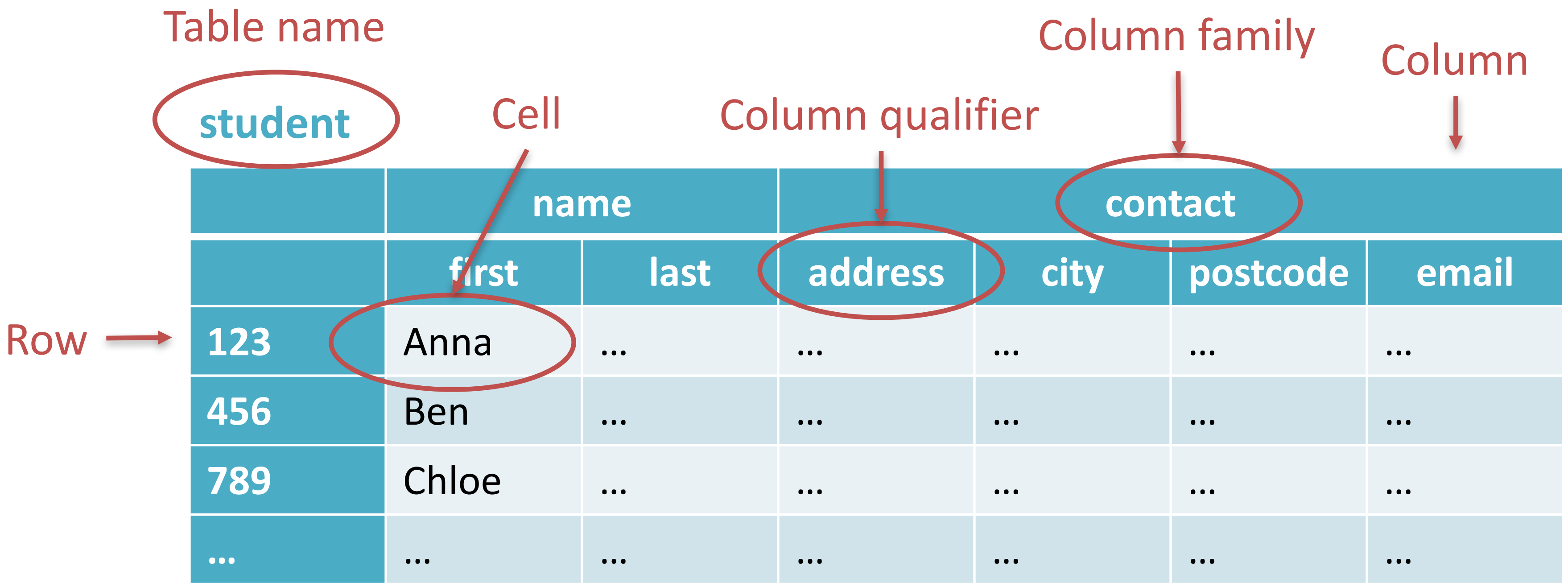
Limited Transaction Support

- ACID for transactions affecting single documents, and the multi-document version is being developed Concurrency control:
  - Multi-granularity locks at global/database/collection level, optimistic scheduling at lower levels
  - If you need two-phase commit, must implement it yourself (or use a distributed relational DBMS), or wait until they implement it
  - Limited isolation:
    - Updates can be “locked” so that other operations see their effect only after completion
    - Doesn’t work with sharding yet
  - See https://docs.mongodb.com/manual/faq/concurrency

Other Types of NoSQL Databases

Column Stores

- Google BigTable, Apache Hbase, …



student

	name		contact			
	first	last	address	city	postcode	email
123	Anna	…	…	…	…	…
456	Ben	…	…	…	…	…
789	Chloe	…	…	…	…	…
…	…	…	…	…	…	…

- Table names/column families fixed, column qualifiers vary
- Columns referenced as <column family>:<column qualifier>

Case Study: Hbase

https://hbase.apache.org/

- Uses Hadoop Distributed File System
  - also mentioned in MapReduce lecture
  - Creating table

```
create ‘STUDENT’,’Name’,’Grade’,’Programme’
```

  - Inserting rows

```
put ‘STUDENT’,’row1’,’Name:Fname’,’Anna’
put ‘STUDENT’,’row1’,’Grade:COMP207’,’100’
```

  - Finding documents

```
get ‘STUDENT’,’row1’
scan ‘STUDENT’
```

For going through the full table

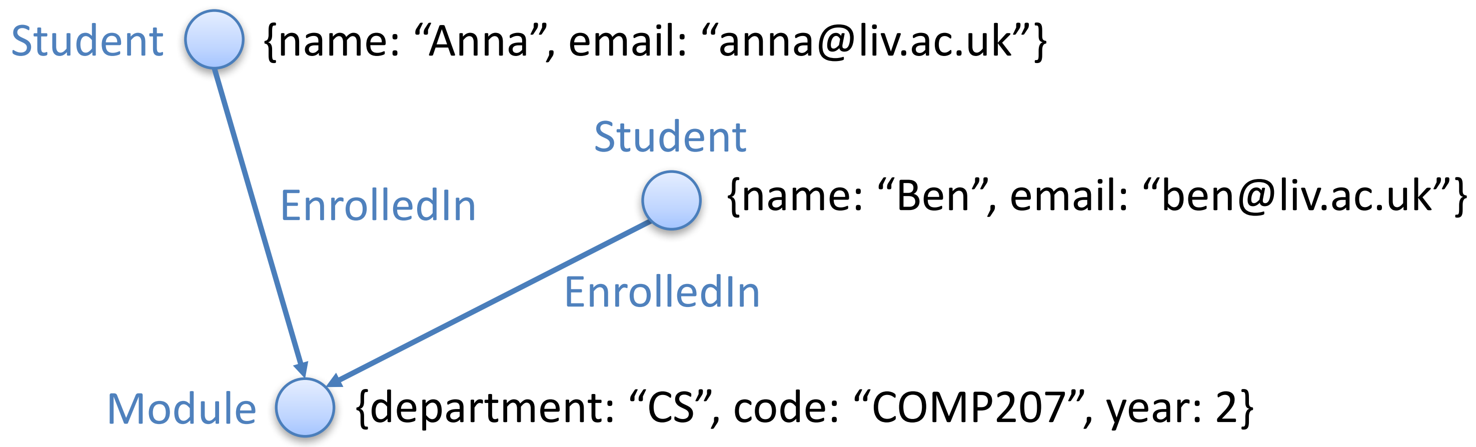
Hbase Techniques

- Uses two levels of fragmentation:
  - Top level: rows are divided into regions (i.e., ranges of rows)
  - Bottom level: regions store different column families in different nodes (i.e., computers)
- No transaction support
- Each item has a time stamp and one can access past versions of the database (if setup)

```
> get ’STUDENT’, ‘row1’
COLUMN          CELL
 name:first     timestamp=1511139129384, value=Anna
 name:last      timestamp=1511139140333, value=…
```

Graph Databases

- Store data as a graph:


Student
{name: “Anna”, email: “anna@liv.ac.uk”}
Student
{name: “Ben”, email: “ben@liv.ac.uk”}
EnrolledIn
EnrolledIn
Module
{department: “CS”, code: “COMP207”, year: 2}

- Many systems: e.g., Neo4j
  - Data is accessed using SQL-like path query language
  - Indexes, …

Summary

- NoSQL databases are the newest members to the data management family
  - Simpler/different data models and query language
  - Weaken ACID properties
- Main use cases:
  - Applications where speed, availability, scalability is crucial, and ACID properties can be weakened
  - Applications where the different way of accessing data (cell-wise as in a column store, as a graph, …) is crucial
- Not a replacement for relational databases