To SQL or Not to SQL, that is the question!

Relational Databases
- Relational databases are the dominant form of database and apply to many data management problems.
- Relational databases are not the only way.
  - Other models:
    - Hierarchical model
    - Object-oriented
    - XML
    - Graphs
    - Key-value stores
    - Document models

So why not relational.....
- Database application?
- Data trends?

Database Application example
- 1990 Database to store information about Customers
  - One table
  - Cust(Last-name, First-name, SSN, address, telephone number)
- 2000 Database needs to add email address
- 2005 Database needs to add mobile phone
- So….change schema?
A better design – exploiting relational model

- Cust (Last-name, First-name, SSN, address, telephone number)
- Three table schema:
  - Cust (Last-name, First-name, SSN, Address)
  - Cust_Phone (SSN, number, phone_type)
  - Type_Detail (Type, Description)
    - example: (o, office), (h, house), (c, cell)

- Do the same for address – physical or email
  - now you have a lot of join operations to compute ‘original’ data
- But in which world do people think of such designs ?!
- Suppose I live in such a world……so:

So you think you solved the problem of adding new fields…

- Today you have more types of contact info (more fields/attributes in the table)
  - Web
  - Twitter
  - Github
  - …
- Not everyone may have this contact information
  - Having a column in table leads to too many NULLs
- Relational model may not be the appropriate model for such ‘schema less’ data

What are trends in Data

Data is getting bigger:
“Every 2 days we create as much information as we did up to 2003”
– Eric Schmidt, Google

Facebook generates 4 Petabytes per day!
Data is more connected:

- Text
- HyperText
- RSS
- Blogs
- Tagging
- RDF

Trend 2: Connectedness

- Text Documents
- Hypertext
- Feeds
- Blogs
- UGC
- Wiki
- Tagging
- Folksonomies
- RDF
- Ontologies
- HTML

Data is more Semi-Structured:

- If you tried to collect all the data of every movie ever made, how would you model it?
- Actors, Characters, Locations, Dates, Costs, Ratings, Showings, Ticket Sales, etc.

Database Scaling – big data performance challenges:

- RDBMS are “scaled up” by adding hardware processing power
  - Need more performance, upgrade your machine

- NoSQL is “scaled out” by spreading the load
  - Partitioning (sharding)/replication
  - Need more performance, thrown in more machines and distributed the work
How to store/retrieve ‘schema less’ data? …..JSON (also XML)

- **JavaScript Object Notation (JSON)** is a method for serializing data objects into text form.

- Benefits:
  - Human-readable
  - Supports semi-structured data
  - Supported by many languages (not just JavaScript)

- Often used for data interchange especially with AJAX/REST from web server to client

---

**JSON Example**

**JSON constructs:**
- **Values:** number, strings (double quoted), true, false, null
- **Objects:** enclosed in `{ }` and consist of set of key-value pairs
- **Arrays:** enclosed in `[]` and are lists of values
- **Objects and arrays can be nested.**

**Example:**

```
({"Employee": ["name":"Lee", "SSN": 123, "address": "Main St", "SSN": 123, "tel": "2025550000"],
"name": "Miller", "SSN": 456, "address": "Walnut St", "tel": "2155550000"},
{"name": "Kim", "SSN": 789, "address": "I St", "tel": "2029940000", "cell": "2025551111"})
```

---

**JSON Parsers**

- **JSON parser:** converts JSON file (or string) into program objects (checks syntax)
  - In javascript, can call eval() method on variable containing a JSON string
  - Many prog. Lang. have APIs to allow for creation and manipulation of JSON objects

- **Common use:**
  - JSON data provided from a server (NoSQL or relational) and sent to web client
  - Web client uses javascript to convert JSON into objects and manipulate as required

---

**Return to First example**

- **Customer info in 1990**
  ```
  { "Customer": [ {"lastname": "Lee", "SSN": 123, "address": "Main St", "tel": "2025550000"},
  {"lastname": "Miller", "SSN": 456, "address": "Walnut St", "tel": "2155550000"} ] }
  ```

- **2019 New customer with cell phone:**
  ```
  { "Customer": [ {"lastname": "Lee", "SSN": 123, "address": "Main St", "tel": "2025550000"},
  {"lastname": "Miller", "SSN": 456, "address": "Walnut St", "tel": "2155550000"},
  {"lastname": "Kim", "SSN": 789, "address": "I St", "tel": "2029940000", "cell": "2025551111"} ] }
  ```
Relational Databases Challenges

- Some features of relational databases make them "challenging" for certain problems:
  - 1) Fixed schemas – The schemas must be defined ahead of time, changes are difficult, and loss of real-world data is "messy".
    - Solution: Get rid of the schemas! Who wants to do that design work anyways? Will you miss them?
  - 2) Complicated queries – SQL is declarative and powerful but may be overkill.
    - Solution: Simple query mechanisms and do a lot of work in code.
  - 3) Transaction overhead – Not all data and query answers need to be perfect. "Close enough is sometimes good enough".
  - 4) Scalability – Relational databases may not scale sufficiently to handle high data and query loads or this scalability comes with a very high cost.

What is NoSQL?

- Stands for No-SQL or Not Only SQL?
- Class of non-relational data storage systems
  - E.g. BigTable, Dynamo, PNUTS/Sherpa, ..
- Usually do not require a fixed table schema nor do they use the concept of joins
  - Distributed data storage systems
- All NoSQL offerings relax one or more of the ACID properties (will talk about the CAP theorem)

NoSQL

NoSQL databases are useful for several problems not well-suited for relational databases with some typical features:

- **Variable data**: semi-structured, evolving, or has no schema
- **Massive data**: terabytes or petabytes of data from new applications (web analysis, sensors, social graphs)
- **Parallelism**: large data requires architectures to handle massive parallelism, scalability, and reliability
- **Simpler queries**: may not need full SQL expressiveness
- **Relaxed consistency**: more tolerant of errors, delays, or inconsistent results (“eventual consistency”)
- **Easier/cheaper**: less initial cost to get started

NoSQL is not really about SQL but instead developing data management systems that are not relational.

NoSQL – “Not Only SQL”

CAP Theorem

- The CAP Theorem (proposed by Eric Brewer) states that there are three properties of a data system:
  - consistency
  - availability
  - partitions

- and that you can have at most two of the three properties at a time.

- Since scaling out requires partitioning, many NoSQL systems sacrifice consistency for availability.
**Eventual Consistency**

- Updates will eventually propagate through all nodes in the system which will then be consistent.
- For each update, either it is reflected in a working node or the node is no longer used in the system.
- Known as **BASE**:
  - **B**asically **A**vailable
  - **S**oft state
  - **E**ventual consistency
- Different consistency properties than ACID (for relational).

**NoSQL Database Models**

- **Key-value stores** are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or “key”), together with its value. Examples of key-value stores are Dynamo DB, Riak and Berkeley DB.
- **Wide-column stores** such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows.
- **Document databases** pair each key with a complex data structure known as a document, similar to key-value stores except value is a document in some form (e.g. JSON).
- **Graph stores** are used to store information about networks of data, such as social connections. Graph stores include Neo4j and triple stores like Fuseki.
- **Map reduce**: useful for large scale analysis
  - Parallelization technique

**Typical NoSQL API**

- Basic API access:
  - `get(key)` -- Extract the value given a key
  - `put(key, value)` -- Create or update the value given its key
  - `delete(key)` -- Remove the key and its associated value
  - `execute(key, operation, parameters)` -- Invoke an operation to the value (given its key) which is a special data structure (e.g. List, Set, Map ... etc).

**What do you lose with NoSQL systems?**

- joins
- group by
- order by
- ACID transactions
- SQL
- enterprise integration with other relational and SQL-based systems
- JDBC/ODBC APIs
- familiarity and standards compliance
MapReduce

MapReduce was invented by Google and has an open source implementation called Hadoop.

Data is stored in files. Users provide functions:
- reader(file) – converts file data into records
- map(records) – converts records into key-value pairs
- combine(key, list of values) – optional aggregation of pairs after map stage
- reduce(key, list of values) – summary on key values to produce output records
- write(file) – writes records to output file

MapReduce (Hadoop) provides infrastructure for tying everything together and distributing work across machines.

MapReduce Example

Web Data Analysis

- Data file records: URL, timestamp, browser

- Goal: Determine the most popular browser used.

MapReduce Extensions

The key benefit of MapReduce and Hadoop is their scalable performance, not that they do not support SQL. In fact, schemas and declarative SQL have many advantages.

Extensions to Hadoop combine the massive parallel processing with familiar SQL features:
- Hive – an SQL-like language variant
- Pig – similar to relational operators

Data manipulations expressed in these languages are then converted into a MapReduce workflow automatically.

Key-Value Stores

Key-value stores store and retrieve data using keys. The data values are arbitrary. Designed for “web sized” data sets.

Operations:
- insert(key, value)
- fetch(key)
- update(key)
- delete(key)

Benefits: high-scalability, availability, and performance
Limitations: single record transactions, eventual consistency, simple query interface
Systems: Cassandra, Amazon Dynamo, Google BigTable, HBase
Key Value Stores: Pros and Cons

- **Pros:**
  - Simple data model
  - Scalable

- **Cons:**
  - Poor for complex data

Document Stores

**Document stores** are similar to key-value stores but the value stored as a structured document (e.g., JSON, XML).

- Can store and query documents by key as well as retrieve and filter documents by their properties.
- Benefits: high-scalability, availability, and performance
- Limitations: same as key-value stores, may cause redundancy and more code to manipulate documents

Systems: CouchDB, SimpleDB, ... MongoDB

---

Document Store

- The central concept is the notion of a "document" which corresponds to a row in RDBMS.
- A document comes in some standard formats like JSON (BSON).
- Documents are addressed in the database via a unique key that represents that document.
- The database offers an API or query language that retrieves documents based on their contents.
- Documents are schema-free, i.e., different documents can have structures and schema that differ from one another. (An RDBMS requires that each row contain the same columns.)

---

MongoDB to documents (JSON):

```
{
    _id: ObjectId("51156a1e056d6f966f268f81"),
    type: "Article",
    author: "Derick Rethans",
    title: "Introduction to Document Databases with MongoDB",
    date: ISODate("2013-04-24T16:26:31.911Z"),
    body: "This arti..."
}
```

```
{
    _id: ObjectId("51156a1e056d6f966f268f82"),
    type: "Book",
    author: "Derick Rethans",
    title: "phparchitect’s Guide to Date and Time Programming with PHP",
    isbn: "978-0-9738621-5-7"
}
```
Document Databases: Pros and Cons

- **Pros:**
  - Simple, powerful data model
  - Scalable
- **Cons**
  - Poor for interconnected data
  - Query model limited to keys and indexes
  - Map reduce for larger queries

Column Family

- Most Based on **BigTable:** Google's Distributed Storage System for Structured Data
- **Data Model:**
  - A big table, with column families
  - Every row can have its own schema
  - Helps capture more "messy" data
  - Map Reduce for querying/processing
- **Examples:**
  - HBase, HyperTable, Cassandra

Column Family: Pros and Cons

- **Pros:**
  - Supports Simi-Structured Data
  - Naturally Indexed (columns)
  - Scalable
- **Cons**
  - Poor for interconnected data

What is a Graph Database?

- A database with an explicit graph structure
- Each node knows its adjacent nodes
- As the number of nodes increases, the cost of a local step (or hop) remains the same
- Plus an Index for lookups
Graph Databases

Graph databases model data as nodes (with properties) and edges (with labels).

Systems: Neo4j, FlockDB

Should I be using NoSQL Databases?

- NoSQL Data storage systems makes sense for applications that need to deal with very very large semi-structured data
  - Log Analysis
  - Social Networking Feeds
- For organizational databases, which are not that large and have low update/query rates
  - regular relational databases are the correct solution for such applications

Star Wars Social Network

Star Wars Social Network ‘subgraph’ with only Episode 1 characters: query!!
Graph Databases: Pros and Cons

- **Pros:**
  - Powerful data model, as general as RDBMS
  - Connected data locally indexed
  - Easy to query

- **Cons**
  - Sharding (lots of people working on this)
    - Scales UP reasonably well
  - Requires rewiring your brain

Next….MongoDB: A NoSQL DB

- April 1: Guest Lecture on Mongo DB
  - Katherine Walker, Class of 2017, MongoDB
- April 8, 10, 15 (and maybe 17 if needed)
  - Introduction to MongoDB
  - Inclass queries and homework
- April (17) 22-24: Security and Privacy in Databases
- April 29–April 30:
  - Data Analytics – Data Mining overview
  - Graph Databases
- May 1: Exam 2
- May 3: Project phase 2 questions