Some issues on Conceptual Modeling and NoSQL/Big Data

Tok Wang Ling

National University of Singapore

Database Models

- **File system** - field, record, fixed length record
- Hierarchical Model (IMS) - fixed length record, tree structure
- **Network** Model (IDMS) - field, fixed length record; owner, member, set (circular linked list)
- Relational Model - fixed length, normal forms based on FD & MVD
  - Nested Relation
- Entity-Relationship Approach - for conceptual database design
- **Object-Oriented** (OO) Data Model – object, object ID, class hierarchy, inheritance, method, …
- Object Relational Data Model
- Deductive and Object-Oriented (DOOD) – deductive OO rules
- Semi-structured Data Model (XML data) – hierarchical structure

Some problems in Relational Model

- Use FDs and MVDs, but no concept on relationship
  - ❖ FDs and MVDs are not relationships, they are integrity constraints
- Normal forms are to remove redundancies and to reduce updating anomalies.
- Does redundancy definitely incur updating anomalies?

  Not always!

  - E.g. **supply (S#, Sname, P#, Pname, price)**

    has redundant information on Sname and Pname, but it does not suffer from updating anomalies as we don't change Sname and Pname of suppliers and parts, resp.
  - Concepts: **Strong FD, Strong MVD, Strong relationship**.
  - E.g. In an **invoice** of a **purchase order**, we add the product name, price of product, and total amount of the invoice. These are redundant, but they don't incur updating anomalies we don't change the invoice. Better!

❖ *Adding redundancy for physical database design may not incur updating anomalies and instead may improve performance significantly.*

❖ *Normal form relations may be bad for the performance of some applications. Any theory?*

Some problems in Relational Model

- RDBMS cannot handle multi-valued attributes and composite attributes efficiently.

E.g. Nested relation:

employee (e#, name, sex, dob, hobby*,
qual (degree, university, year)*)

To store employee information, we need **3** normal form relations. To get information of an employee, we need to join the 3 relations, very inefficient and slow.

❖ Normal form relations are not efficient for storing and processing multi-valued attributes; nested relation/OO storage is better.

Some problems in Relational Model

- RDBMS join operation is very expensive.
- SQL is not useful for many applications
- SQL is a declarative language which operates on a set of tuples at a time basis.

  MapReduce is an imperative language which operates a record at a time basis. Good for some applications.
- ACID (Atomicity, Consistency, Isolation, Durability) is to sure consistency of data.

  Overhead for enforcing ACID is very high and may not be necessary for many applications (such as large data volume applications which don't update the data).

NoSQL

NoSQL (not only SQL? no join?) databases are categorized according to the way they store the data.

4 major categories for NoSQL databases:

1) Key-value stores
   - Each data/object is stored, indexed, and accessed using a key. Data can be structured or semi-structured or unstructured.
   - E.g. Tables: Customer and Orders

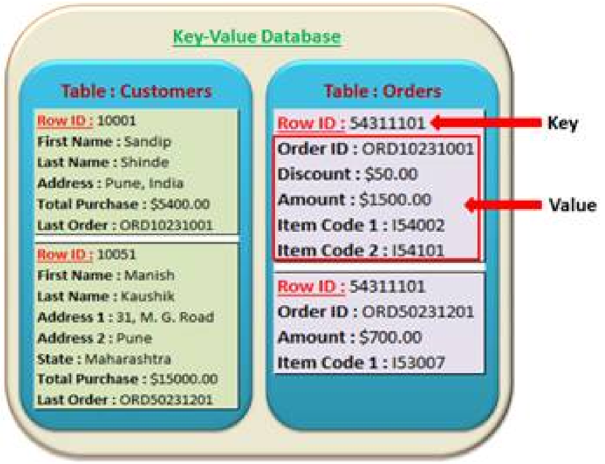

Table : Customers

Row ID : 10001
First Name : Sandip
Last Name : Shinde
Address : Pune, India
Total Purchase : $5400.00
Last Order : ORD10231001

Row ID : 10051
First Name : Manish
Last Name : Kaushik
Address 1 : 31, M. G. Road
Address 2 : Pune
State : Maharashtra
Total Purchase : $15000.00
Last Order : ORD50231201

Table : Orders

Row ID : 54311101
Order ID : ORD10231001
Discount : $50.00
Amount : $1500.00
Item Code 1 : I54002
Item Code 2 : I54101

Row ID : 54311101
Order ID : ORD50231201
Amount : $700.00
Item Code 1 : I53007

Notes. Tag name: value. Can have different tag names for different tuples, semi-structured or unstructured.

NoSQL

2) Wide-Column stores (Column-oriented stores)

- Contain extendable columns of closely related data,
- E.g. Bigtable
  - Bigtable is a spare, distributed, persistent multi-dimensional sorted map.
  - The map is indexed by a row key, column key, and a timestamp; each value is the map is an un-interpreted array of bytes.

    (row-key, column-key, time) -> string
  - The row key value is a reversed url.
  - Bigtable maintains data in lexicographic order by row key. So webpages in the same domain are grouped together into contiguous rows.
  - Column keys are grouped into sets called column families, which form the basic unit of access control.
  - A column key is named using the syntax: family:qualifier

3) Document Stores

- Similar to XML data: semi-structured data

NoSQL

4) Graph Database

- **Nodes** and **edges** (objects and relationships). Nodes and/or edges are typed an/or labelled? Directed or undirected edges? Edges are weighted?
- **Binary relationships** between pairs of nodes. Where to store the binary relationships' attributes? On the edges?
- Problems in handling **n-ary relationship** types. How?
- If we store a data graph using a *relation*

  ***edge (from-node,* to-node)**

  to traverse from one node to another node (e.g. shortest path problem) will be very slow, needs many joins.

  So, RDBMS is bad for storing and processing graph data.
- Graph data can be stored as **in-memory database** using pointers/node IDs for efficient node traversal.
- Write programs to solve specific problems
- **Problems.** How to express user ad hoc queries? How to present query answers for users to understand? E.g. Steiner tree.

SQL vs NoSQL

- has a fixed schema, many related relations and fixed length
  vs does not require a schema or uses a semi-structure model
- declarative standard query language
  vs different imperative programming languages
- write queries using SQL
  vs write programs (e.g. MapReduce programs with API's, etc.) for queries
- Query optimizer of RDBMS
  vs optimization done by programmers
- return accurate/precise query answer
  vs return "an opinion" or "best guess" – similar to data mining and IR.

SQL vs NoSQL

- allow **updates** (rewrite)

  vs just have new data, no or seldom updates (delete or change).
- Enforce **ACID** for consistency

  vs emphasis on speed performance, use eventual consistent
- Use **join** operator

  vs avoid or no join operator (e.g. use redundant data) to speed up processing
- For many different various database applications

  - mission critical transaction systems

  vs each design for some specific applications

  - search engines, web-based systems, real-time, cloud
- DBMS

  vs **data stores?**

Question: When do we use SQL or NoSQL?

Depend on applications.

Criteria: **Fixed schema? Transactions? ACID required? Precise answers? Ad hoc user queries? Frequent updates?** horizontal and/or vertical partition? **Very large volume data? Complex algorithms needed to solve the problems and queries (e.g. data mining techniques)?**

Use of existing database techniques for NoSQL conceptual modeling and Questions

- Materialized view and introduce redundant data for faster processing. Theory behind.
- Horizontal and vertical partitioning of data for different applications in physical database designing. Can the data be horizontally and/or vertically partitioned? What overhead will incur? Theory behind.

  If data can be **partitioned horizontally** and processed in parallel, and merge the results of the partitions, then Hadoop with MapReduce is a good solution.

  - E.g. To **sort a big volume of string data**, we can partition the string data on the first character, the sort individual partition using a O(n log n) method.

    If the **complexity** of the sort method is **$f(n) = c*n*\log n$** where n is the data size and c is a constant, each partition needs **$c*n/k*\log(n/k) < f(n)/k$** , so much faster completion time, where k is the number of parallel nodes. However, the total computation operations is about the same.
  - For **$O(n^2)$** problems, the speed up can be up to **K^2** times.
- Class hierarchy and inheritance for NoSQL data?
- DOOD rules for NoSQL data?

Use of existing database techniques for NoSQL conceptual modeling and Questions

- Data and Schema integration. Entity resolution (object identification) is not enough we need to consider relationship resolution (relationship identification). Theory behind.
  - We need to handle local/global FD/key, key vs OID of object class.
  - Need to consider n-ary relationship types and their attributes.
  - Temporal aspect is important for data integration like in data warehouse.
- Q: How to use ORA-semantics for NoSQL data in order to improve the quality/accuracy of the answers, such as in Data/schema integration, keyword query search?

Thank you!