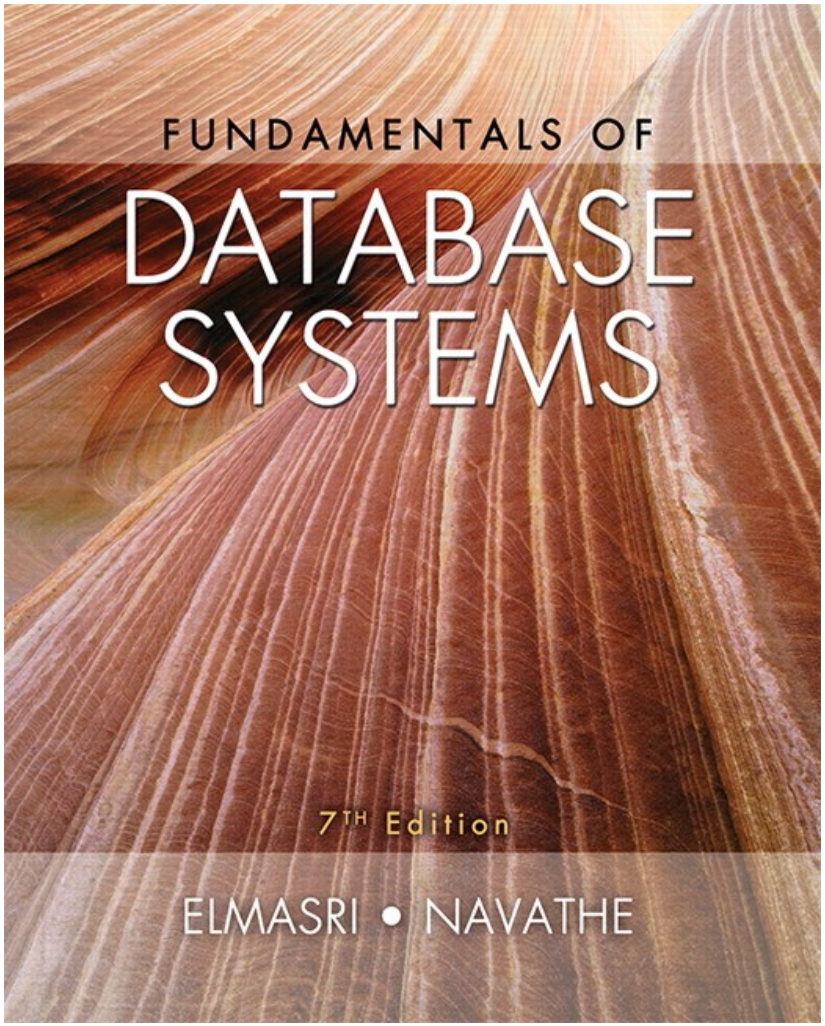

CHAPTER 24

NOSQL Databases and Big Data Storage Systems

Introduction

- NOSQL
  - Not only SQL
- Most NOSQL systems are distributed databases or distributed storage systems
  - Focus on semi-structured data storage, high performance, availability, data replication, and scalability

Introduction (cont'd.)

- NOSQL systems focus on storage of "big data"
- Typical applications that use NOSQL
  - Social media
  - Web links
  - User profiles
  - Marketing and sales
  - Posts and tweets
  - Road maps and spatial data
  - Email

24.1 Introduction to NOSQL Systems

- BigTable
  - Google's proprietary NOSQL system
  - Column-based or wide column store
- DynamoDB (Amazon)
  - Key-value data store
- Cassandra (Facebook)
  - Uses concepts from both key-value store and column-based systems

Introduction to NOSQL Systems (cont'd.)

- MongoDB and CouchDB
  - Document stores
- Neo4J and GraphBase
  - Graph-based NOSQL systems
- OrientDB
  - Combines several concepts
- Database systems classified on the object model
  - Or native XML model

Introduction to NOSQL Systems (cont'd.)

- NOSQL characteristics related to distributed databases and distributed systems
  - Scalability
  - Availability, replication, and eventual consistency
  - Replication models
    - Master-slave
    - Master-master
  - Sharding of files
  - High performance data access

Introduction to NOSQL Systems (cont'd.)

- NOSQL characteristics related to data models and query languages
  - Schema not required
  - Less powerful query languages
  - Versioning

Introduction to NOSQL Systems (cont'd.)

- Categories of NOSQL systems
  - Document-based NOSQL systems
  - NOSQL key-value stores
  - Column-based or wide column NOSQL systems
  - Graph-based NOSQL systems
  - Hybrid NOSQL systems
  - Object databases
  - XML databases

24.2 The CAP Theorem

- Various levels of consistency among replicated data items
  - Enforcing serializabilty the strongest form of consistency
    - High overhead – can reduce read/write operation performance
- CAP theorem
  - Consistency, availability, and partition tolerance
  - Not possible to guarantee all three simultaneously
    - In distributed system with data replication

The CAP Theorem (cont'd.)

- Designer can choose two of three to guarantee
  - Weaker consistency level is often acceptable in NOSQL distributed data store
  - Guaranteeing availability and partition tolerance more important
  - Eventual consistency often adopted

24.3 Document-Based NOSQL Systems and MongoDB

- Document stores
  - Collections of similar documents
- Individual documents resemble complex objects or XML documents
  - Documents are self-describing
  - Can have different data elements
- Documents can be specified in various formats
  - XML
  - JSON

MongoDB Data Model

- Documents stored in binary JSON (BSON) format
- Individual documents stored in a collection
- Example command
  - First parameter specifies name of the collection
  - Collection options include limits on size and number of documents

```
db.createCollection("project", { capped : true, size : 1310720, max : 500 } )
```

- Each document in collection has unique ObjectID field called _id

MongoDB Data Model (cont'd.)

- A collection does not have a schema
  - Structure of the data fields in documents chosen based on how documents will be accessed
  - User can choose normalized or denormalized design
- Document creation using insert operation

db.<collection_name>.insert(<document(s)>)

- Document deletion using remove operation

db.<collection_name>.remove(<condition>)

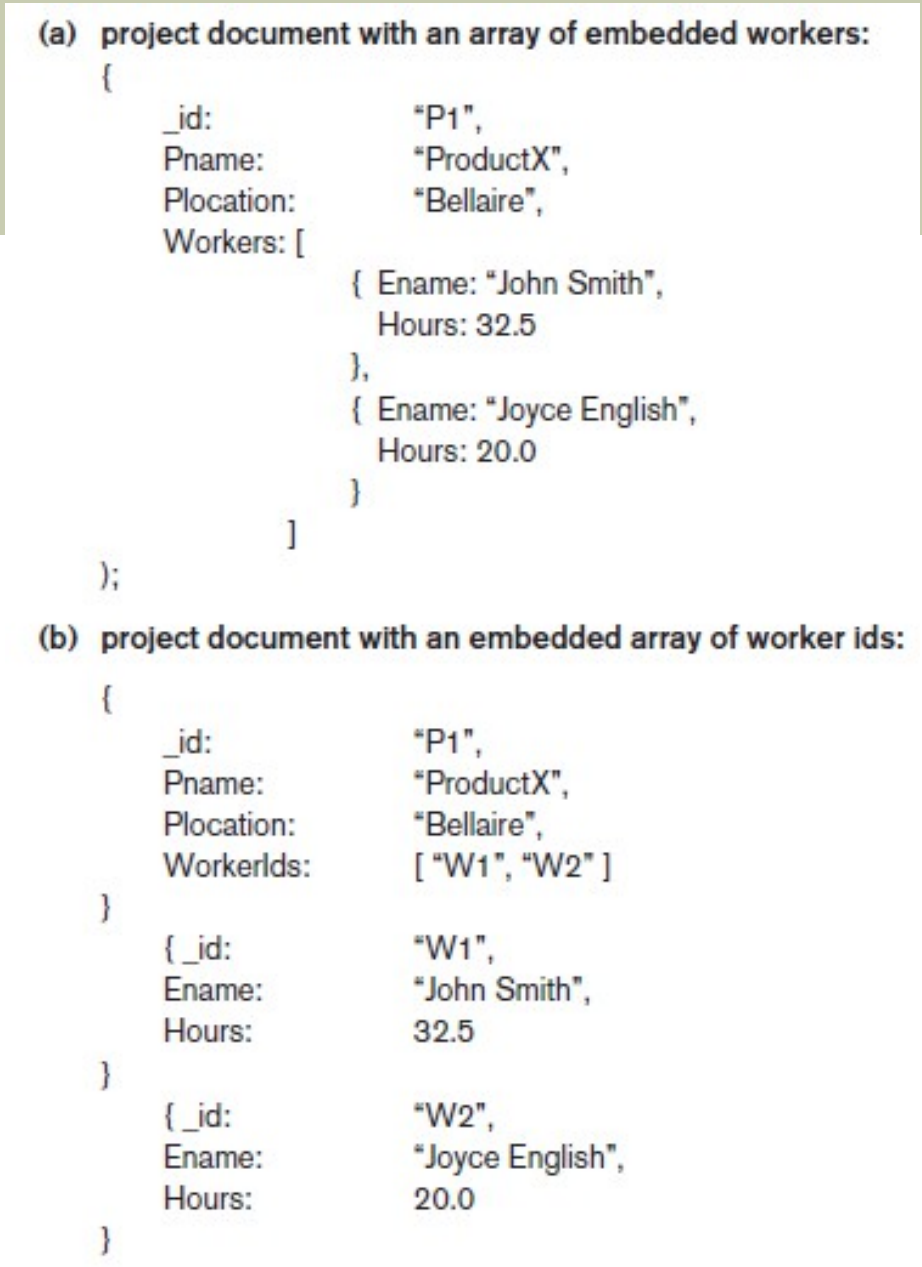

Figure 24.1 (continues) Example of simple documents in MongoDB (a) Denormalized document design with embedded subdocuments (b) Embedded array of document references

Figure 24.1 (cont'd.)
Example of simple documents in MongoDB

(c) Normalized documents

(d) Inserting the documents in Figure 24.1(c) into their collections

(c) normalized project and worker documents (not a fully normalized design for M:N relationships):

```
{
    _id:        "P1",
    Pname:      "ProductX",
    Plocation:  "Bellaire"
}
{   _id:        "W1",
    Ename:      "John Smith",
    ProjectId:  "P1",
    Hours:      32.5
}
{   _id:        "W2",
    Ename:      "Joyce English",
    ProjectId:  "P1",
    Hours:      20.0
}
```

(d) inserting the documents in (c) into their collections "project" and "worker":

```
db.project.insert( { _id: "P1", Pname: "ProductX", Plocation: "Bellaire" } )
db.worker.insert( [ { _id: "W1", Ename: "John Smith", ProjectId: "P1", Hours: 32.5 },
                    { _id: "W2", Ename: "Joyce English", ProjectId: "P1",
                      Hours: 20.0 } ] )
```

MongoDB Distributed Systems Characteristics

- Two-phase commit method
  - Used to ensure atomicity and consistency of multidocument transactions
- Replication in MongoDB
  - Concept of replica set to create multiple copies on different nodes
  - Variation of master-slave approach
  - Primary copy, secondary copy, and arbiter
    - Arbiter participates in elections to select new primary if needed

MongoDB Distributed Systems Characteristics (cont'd.)

- Replication in MongoDB (cont'd.)
  - All write operations applied to the primary copy and propagated to the secondaries
  - User can choose read preference
    - Read requests can be processed at any replica
- Sharding in MongoDB
  - Horizontal partitioning divides the documents into disjoint partitions (shards)
  - Allows adding more nodes as needed
  - Shards stored on different nodes to achieve load balancing

MongoDB Distributed Systems Characteristics (cont'd.)

- Sharding in MongoDB (cont'd.)
  - Partitioning field (shard key) must exist in every document in the collection
    - Must have an index
  - Range partitioning
    - Creates chunks by specifying a range of key values
    - Works best with range queries
  - Hash partitioning
    - Partitioning based on the hash values of each shard key