CHAPTER 24

NoSQL Databases

Credit Note: Slides, content, web links, and end-chapter questions are prepared from Pearson's textbook and other Internet resources.
Additionally, some of the information was included from another Pearson book by Kroenke, Auer, and others of 2019 – 15th edition and also from the Cengage book of Coronel and Morris -13th Edition.

Topics of Discussion

Introduction

- NoSQL: NoSQL means 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
- NOSQL systems focus on the storage of "big data". Typical applications that use NOSQL
  - Social media,
  - Web links,
  - User profiles,
  - Marketing and sales,
  - Posts,
  - Tweets,
  - Road maps and spatial data
  - Email storage

Introduction

Emergence of NOSQL:

- The name was chosen as a Twitter hashtag for coordinating a meeting of developers to discuss ideas about the non-relational database technologies that were being developed by organizations like Google, Amazon, and Facebook to deal with the problems they were encountering as their data sets reached enormous sizes.

- Many companies and organizations are faced with applications that store vast amounts of data.
  - For example, a free e-mail application, such as Google Mail or Yahoo Mail can have millions of users, and each user can have thousands of e-mail messages.
  - Another example, Facebook, with millions of users who submit posts, many with images and videos; then these posts must be displayed on pages of other users using the social media relationships among the users.

Introduction

- There is a need for a storage system that can manage all these e-mails; a structured relational SQL system may not be appropriate because
  - (1) a structured data model such the traditional relational model may be too restrictive; and
  - (2) SQL systems offer too many services (powerful query language, concurrency control, etc.), which this application may not need.
- Some of the organizations that were faced with this challenge decided to develop their own. (It is an ongoing process even today known as *Not Only SQL Movement)*
  - Google developed a NOSQL system known as BigTable, which is used in many of Google's applications that require vast amounts of data storage, such as Gmail, Google Maps, and Web site indexing

Introduction

- Apache Hbase is an open source NOSQL system based on similar concepts. Innovation led to NOSQL systems known as column-based or wide column stores;
- Amazon developed a NOSQL system called DynamoDB that is available through Amazon's cloud services. This innovation led to the category known as key-value data stores or sometimes key-tuple or key-object data stores.
- Facebook developed a NOSQL system called Cassandra, which is now open source and known as Apache Cassandra. This NOSQL system uses concepts from both key-value stores and column-based systems.

Introduction

<u>Characteristics of NOSQL Systems</u>

- NOSQL characteristics related to distributed databases and distributed systems
  1. **<u>Scalability</u>**: In NOSQL systems, horizontal scalability is generally used, where the distributed system is expanded by adding more nodes for data storage and processing as the volume of data grows.
  2. **<u>Availability, replication, and eventual consistency</u>**: Replication improves data availability and can also improve read performance and NOSQL applications do not require serializable consistency, so more relaxed forms of consistency known as eventual consistency are used.
  3. **<u>Replication models</u>**: Two major replication models are used in NOSQL systems: master-slave and master-master replication.

Introduction

4. **Sharding of files**: Files (or collections of data objects) can have many millions of records; these records can be accessed concurrently by thousands of users. So, it is not practical to store the whole file in one node. Sharding (also known as horizontal partitioning), is often employed in NOSQL systems.
5. **High-performance data access**: To find individual records or objects (data items) from among the millions of data records or objects in a file, most NoSQL systems use one of two techniques: 1) hashing. In hashing, a hash function h(K) is applied to the key K, and the location of the object with key K is determined by the value of h(K).
   Or 2) range partitioning on object keys. In range partitioning, the location is determined via a range of key values;

Introduction

- NOSQL characteristics related to <u>data models</u> and query languages:
  6. <u>**Schema not required**</u>: The flexibility of not requiring a schema is achieved in many NOSQL systems by allowing semi-structured, self-describing data.
     There are various languages for describing semi-structured data, such as JSON (JavaScript Object Notation) and XML (Extensible Markup Language).
     As there may not be a schema to specify constraints, any constraints on the data would have to be programmed in the application programs that access the data items.

Introduction

7. **Less powerful query languages**: NOSQL systems typically provide a set of functions and operations as a programming API, so reading and writing the data objects is accomplished by calling the appropriate operations by the programmer.
   In many cases, the operations are called CRUD operations, for Create, Read, Update, and Delete. Some NOSQL systems not have the full power of SQL; only a subset of SQL querying capabilities would be provided.
8. **Versioning**: Some NOSQL systems provide storage of multiple versions of the data items, with the timestamps of when the data version was created.

Introduction

Categories of NOSQL systems:

1. Document-based NOSQL systems:
   These systems store data in the form of documents using well-known formats, such as JSON (JavaScript Object Notation). Documents are accessible via their document id, but can also be accessed rapidly, using other indexes.
2. NOSQL key-value stores:
   These systems have a simple data model based on fast access by the key to the value associated with the key; the value can be a record or an object or a document or even have a more complex data structure.
3. Column-based or wide column NOSQL systems:
   These systems partition a table by column into column families, where each column family is stored in its own files. They also allow versioning of data values.

Introduction

4. Graph-based NOSQL systems:

   Data is represented as graphs, and related nodes can be found by traversing the edges using path expressions.

Additional categories can be added as follows to include some systems that are not easily categorized into the above four categories.

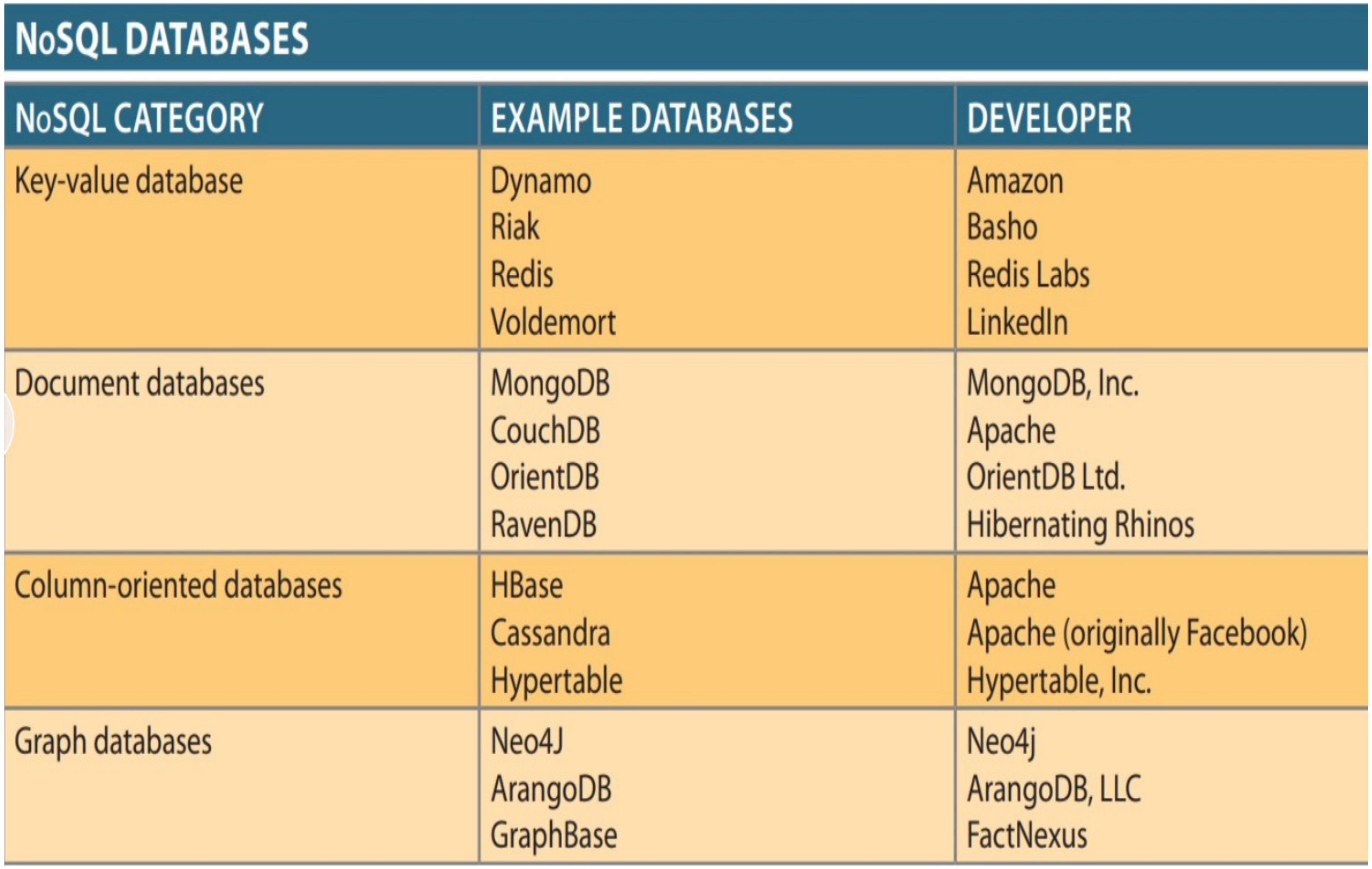
5. Hybrid NOSQL systems:

   These systems have characteristics from two or more of the above four categories.

6. Object databases
7. XML databases

Introduction

Some of known NoSQL database software listed as below.



NoSQL DATABASES

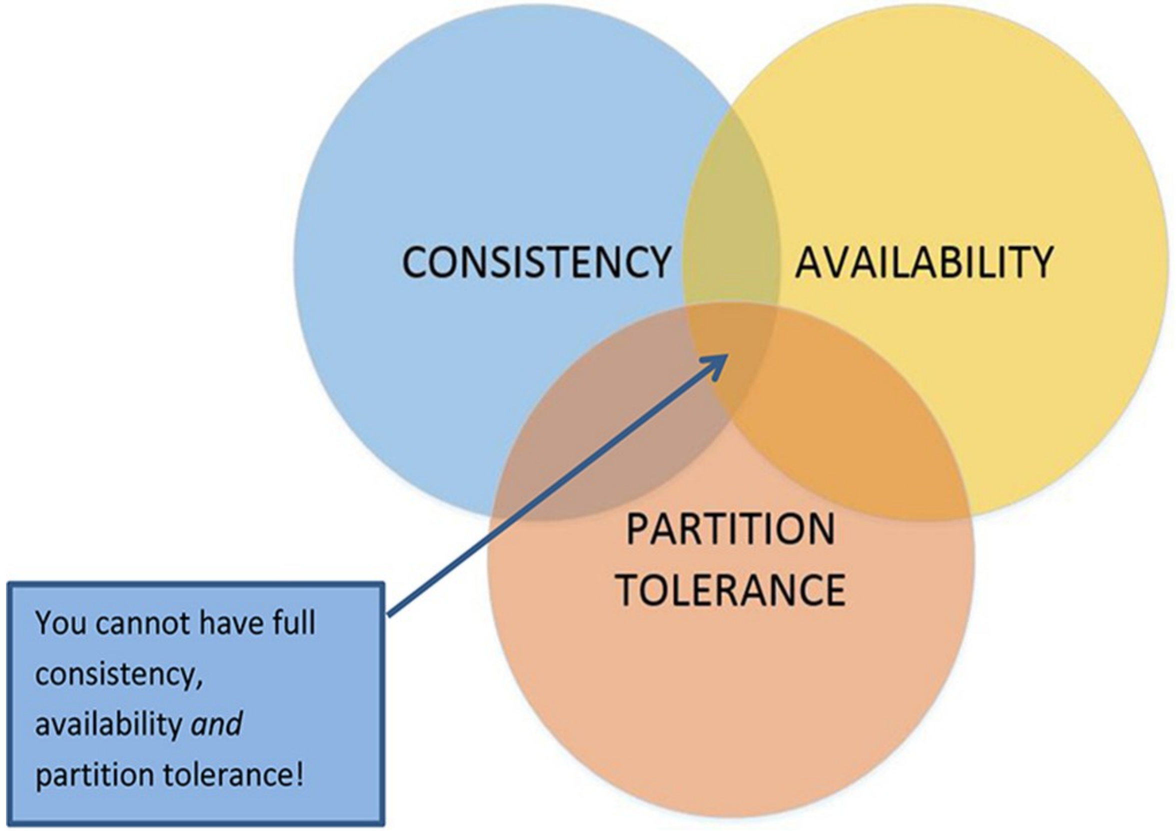
NoSQL CATEGORY	EXAMPLE DATABASES	DEVELOPER
Key-value database	Dynamo Riak Redis Voldemort	Amazon Basho Redis Labs LinkedIn
Document databases	MongoDB CouchDB OrientDB RavenDB	MongoDB, Inc. Apache OrientDB Ltd. Hibernating Rhinos
Column-oriented databases	HBase Cassandra Hypertable	Apache Apache (originally Facebook) Hypertable, Inc.
Graph databases	Neo4J ArangoDB GraphBase	Neo4j ArangoDB, LLC FactNexus

The CAP Theorem

- The CAP theorem (formally proven by Seth Gilbert and Nancy Lynch) defines three properties of distributed database systems:
  - **Consistency** means that all the database replicas see the same data at any given point in time.
  - **Availability** means that every request received by a server will result in a response, as long and the network is available.
  - **Partition tolerance** means that the distributed database can continue to operate even when the cluster is partitioned by network failures into two or more disconnected sections (partitions).

The CAP Theorem

- Various levels of consistency among replicated data items.
- Enforcing serializabilty the strongest form of consistency.
- That High overhead – can reduce read/write operation performance.


CONSISTENCY
AVAILABILITY
PARTITION TOLERANCE
You cannot have full consistency, availability *and* partition tolerance!

The CAP Theorem

- Not possible to guarantee all three simultaneously.
- 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

d) NoSQL Key-Value Stores

- Key-value stores focus on high performance, availability, and scalability
  - Can store structured, unstructured, or semi-structured data
- **Key**: unique identifier associated with a data item
  - Used for fast retrieval
- **Value**: the data item itself. The value can be anything such as text, an XML document, URL, or an image.
  - Can be string or array of bytes
  - Application interprets the structure
  - the database simply stores whatever value is provided for the key
- No query language

NOSQL Key-Value Stores

- Key-Value databases use a simple key and its associated value pairing.
- Each key appears only once in each database.
- A key-value database is extremely simple and allows for easy distribution of the key-value pairs in a network cluster.
- Key-value databases are ideal for large amounts of data that need fast storage and retrieval of simple objects but do not need the full complexity of SQL queries.
- Basic operations available to query a key-value database:
  - get (key): retrieves the value associated with a key
  - set(key, value): creates or updates a key-value pair
  - delete (key): removes a key-value pair

NOSQL Key-Value Stores

- Key-value pairs are typically organized into "<u>buckets</u>".
- A bucket can roughly be thought of as the KV database equivalent of a table.
- A bucket is a logical grouping of keys. Key values must be unique within a bucket, but they can be duplicated across buckets.
- All data operations are based on the bucket plus the key.
- In other words, it is not possible to query the data based on anything in the value component of the key-value pair.
- All queries are performed by specifying the bucket and key.

DynamoDB Overview

- DynamoDB part of Amazon's Web Services/SDK platforms. In DynamoDB, tables, items, and attributes are the core components that you work with.
- A table is a collection of items, and each item is a collection of attributes.
  - DynamoDB uses primary keys to uniquely identify each item in a table and secondary indexes to provide more querying flexibility.
  - You can use DynamoDB Streams to capture data modification events in DynamoDB tables.
  - For example, see the example table called People that you could use to store personal contact information about friends, family, or anyone else of interest.

Detail official documentation – everything on DynamoDB

DynamoDB Overview

- Item consists of attribute-value pairs
  - Attribute values can be single or multi-valued.
  - An item is a group of attributes that is uniquely identifiable among all of the other items.
- *Attribute* – Each item is composed of one or more attributes. An attribute is a fundamental data element, something that does not need to be broken down any further.
  - For example, an item in a People table contains attributes called PersonID, LastName, FirstName, and so on.
  - For a Department table, an item might have attributes such as DepartmentID, Name, Manager, and so on.
  - Attributes in DynamoDB are similar in many ways to fields or columns in other database systems.

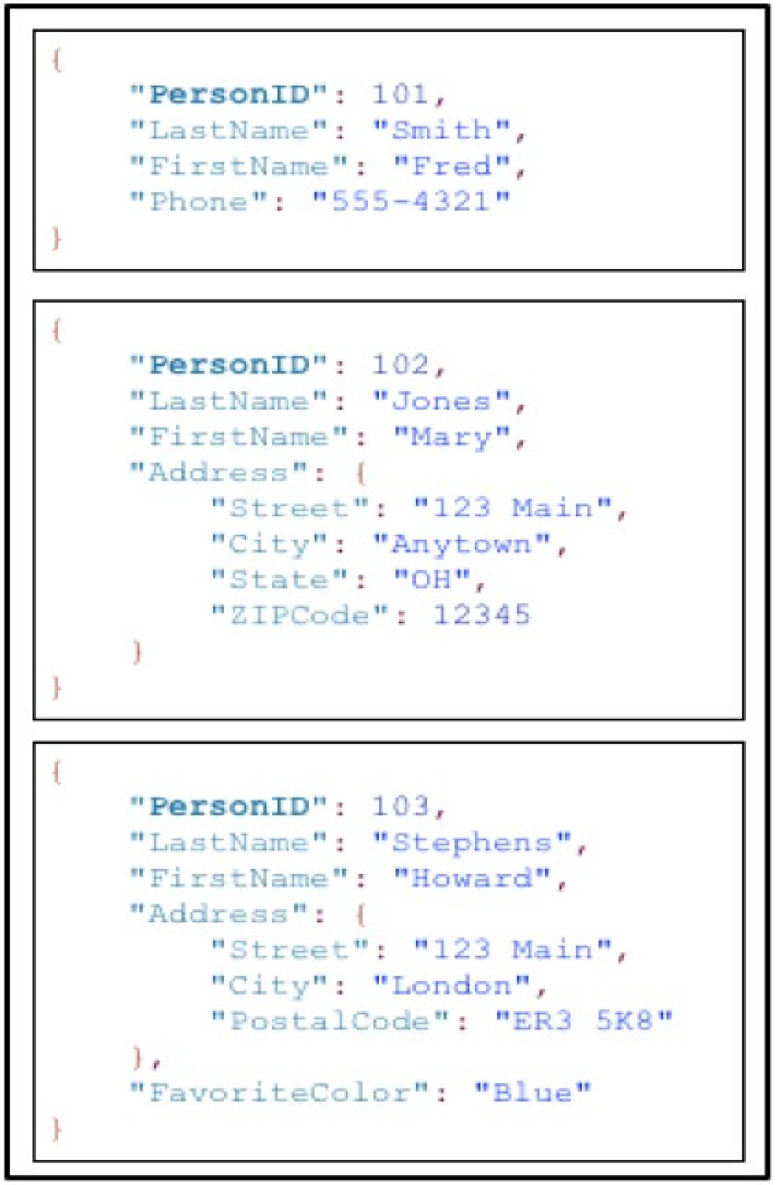
DynamoDB Overview

List

A list type attribute can store an ordered collection of values. Lists are enclosed in square brackets: […]

```
{
“PersonID”: 104,
“LastName”:  “Patel”,
“FirstName”: “Ashok”,
“Address”: {
          “Street”: “1487, Silver leaf dr.”,
          “City”: “Lakeland”,
          “State”: “FL”,
          “ZIPCode”:  33813”,
          },
“FavoriteColor”: [“White”, “Blue”, “Red”]
}
```

People



```
{
    "PersonID": 101,
    "LastName": "Smith",
    "FirstName": "Fred",
    "Phone": "555-4321"
}
```

```
{
    "PersonID": 102,
    "LastName": "Jones",
    "FirstName": "Mary",
    "Address": {
        "Street": "123 Main",
        "City": "Anytown",
        "State": "OH",
        "ZIPCode": 12345
    }
}
```

```
{
    "PersonID": 103,
    "LastName": "Stephens",
    "FirstName": "Howard",
    "Address": {
        "Street": "123 Main",
        "City": "London",
        "PostalCode": "ER3 5K8"
    },
    "FavoriteColor": "Blue"
}
```

DynamoDB Overview

Working with Items:

- DynamoDB provides four operations for basic create, read, update, and delete (CRUD) functionality:
  - PutItem — Create an item.
  - GetItem — Read an item.
  - UpdateItem — Update an item.
  - DeleteItem — Delete an item.
- Each of these operations requires that you specify the primary key of the item that you want to work with.
- For example, to read an item using GetItem, you must specify the partition key and sort key (if applicable) for that item.
- In addition to the four basic CRUD operations, DynamoDB also provides the following:
  - BatchGetItem — Read up to 100 items from one or more tables.
  - BatchWriteItem — Create or delete up to 25 items in one or more tables.

Examples of Other Key-Value Stores

- Oracle key-value store
  - Oracle NoSQL Database
- Redis key-value cache and store
  - Caches data in main memory to improve performance
  - Offers master-slave replication and high availability
  - Offers persistence by backing up cache to disk
- Apache Cassandra
  - Offers features from several NoSQL categories
  - Used by Facebook and others
- Voldemort Key-Value Distributed Data Store
  - Open-source system available through Apache 2.0 open-source licensing rules.
  - Focus is on high performance and horizontal scalability

e) 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
- Document Databases store data according to a document-oriented format of which the two more popular are:
  - XML
  - JSON
- Documents stored in binary JSON (BSON) format.
- Individual documents stored in a collection. (can think of table of relational database).

Document-Based NOSQL Systems and MongoDB

- JSON (JavaScript Object Notation), which is used in several document databases, has some similarities to XML.
- JSON does not include any schema external to the data as XML does.
- A document consists of a set of (field, value) pairs.
- JSON includes arrays which are ordered, indexed list of values.
- To give you a feel for the basic JSON syntax, here is a JSON representation for student data, consisting of two documents (one student per document):

Document-Based NOSQL Systems and MongoDB

```
{
        "id": "student1",
        "lastName": "Smith",
        "firstName": "John",
        "majors": [
              "History",
              "Computer Science"
       ]
},
{
        "id": "student2",

        "minor": "English",
        "majors": [
              "Math"
        ],
        "GPA": 3.2
}
```

Document-Based NOSQL Systems and MongoDB

- The following slide JSON code represents student and advisor information.
- The first document would be stored in a collection of similar "Advisor" documents and the second would be stored in a collection of similar "Student" documents.
- As in the examples, the curly braces indicate a document (a set of (field, value) pairs) and the square brackets indicate an array (an ordered, indexed list of values).
- The student has two majors (each of which is a simple string) and two addresses (each of which is itself a document embedded within the student's addresses field). –known as sub-document. The student's advisorID field contains the identifier for the student's advisor

Document-Based NOSQL Systems and MongoDB

```
{
        "id":  "Advisor1",
        "name":  "Shire, Robert",
        "dept":  "History",
        "yearHired":  1975
}

{
        "id":  555667777,
        "name":  "Tierney, Doris",
        "majors":  ["Music", "Spanish"],
        "addresses:  [
                {
                "street":  "14510 NE 4th Street",
                "city":  "Bellevue",
                "state":  "WA",
                "zip":  "98005"
                },
                {
                "street":  "335 Aloha Street",
                "city":  "Seattle",
                "state":   "WA",
                "zip":  "98109"
                }
                ],
        "advisorID":  "Advisor1"
}
```

Document-Based NOSQL Systems and MongoDB

- Typical commands available in a document database (using a generic syntax) include:
  - **insert (doc, collection)**: put document "doc" into the collection named “collection”
  - **update (collection, doc_specifier, update_action)**: within the collection, update all documents matching the "doc_specifier" (e.g. all students with last name 'Smith'). The "update_action" can alter the values of multiple fields within each document.
  - **delete (collection, doc_specifier)**: removes all documents from collection that match the "doc_specifier"
  - **find (collection, doc_specifier)**: retrieves all documents in a collection matching the "doc_specifier"

Document-Based NOSQL Systems and MongoDB

- 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**. 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

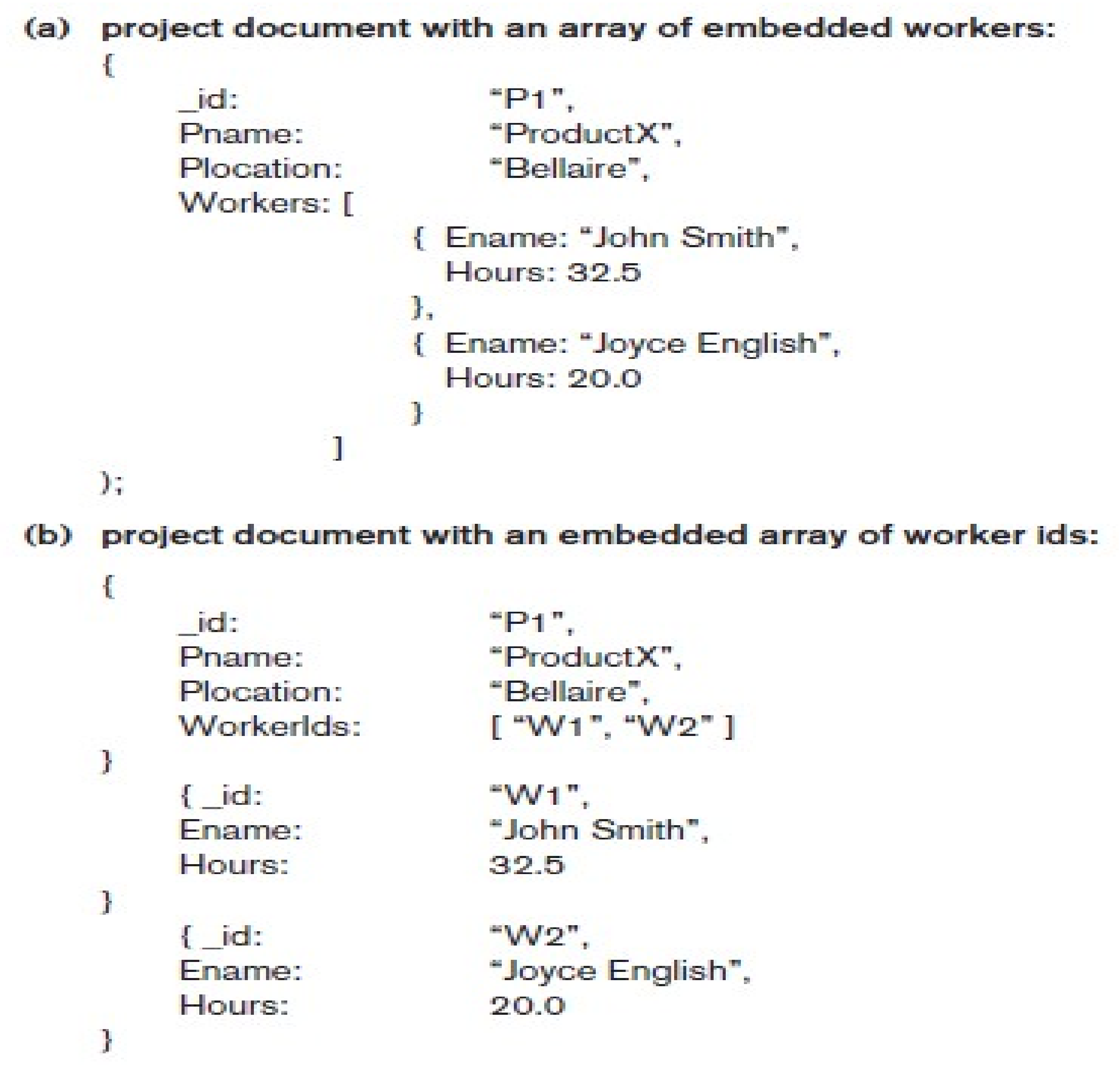
```
db.<collection_name>.insert(<document(s)>)
```

- Document deletion using remove operation

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

Example of simple documents in MongoDB (a) Denormalized document design with embedded subdocuments

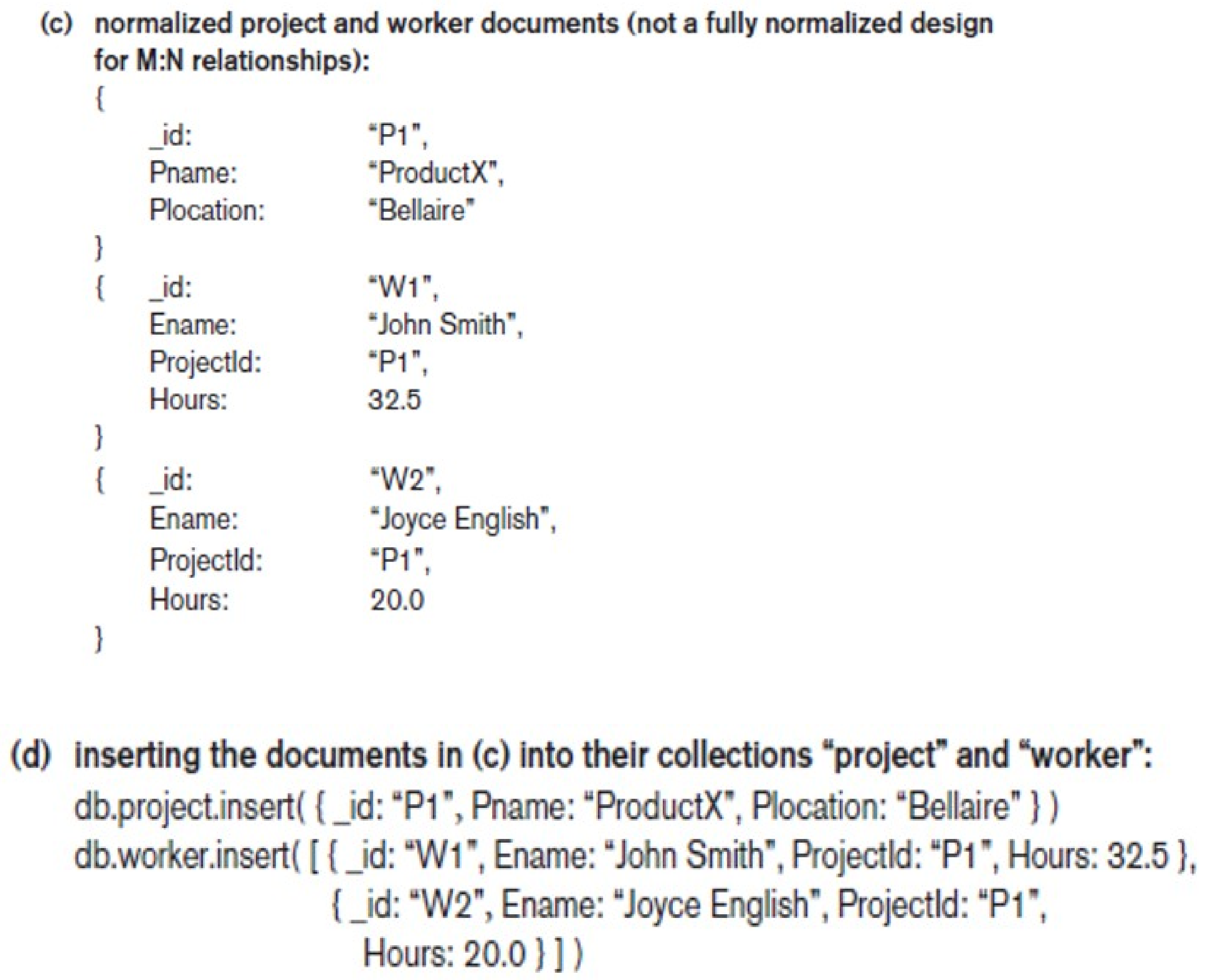
(b) Embedded array of document references



```
(a)  project document with an array of embedded workers:
     {
           _id:                    "P1",
           Pname:                  "ProductX",
           Plocation:              "Bellaire",
           Workers: [
                                {  Ename: "John Smith",
                                   Hours: 32.5
                                },
                                {  Ename: "Joyce English",
                                   Hours: 20.0
                                }
                        ]
     );
(b)  project document with an embedded array of worker ids:
     {
           _id:                    "P1",
           Pname:                  "ProductX",
           Plocation:              "Bellaire",
           WorkerIds:              [ "W1", "W2" ]
     }
           { _id:                  "W1",
           Ename:                  "John Smith",
           Hours:                  32.5
     }
           { _id:                  "W2",
           Ename:                  "Joyce English",
           Hours:                  20.0
     }
```

Example of simple documents in MongoDB (c) Normalized documents.

(d) Inserting the documents in Figure (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
  - 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

Detail official documentation – everything on MongoDB

MongoDB Distributed Systems Characteristics

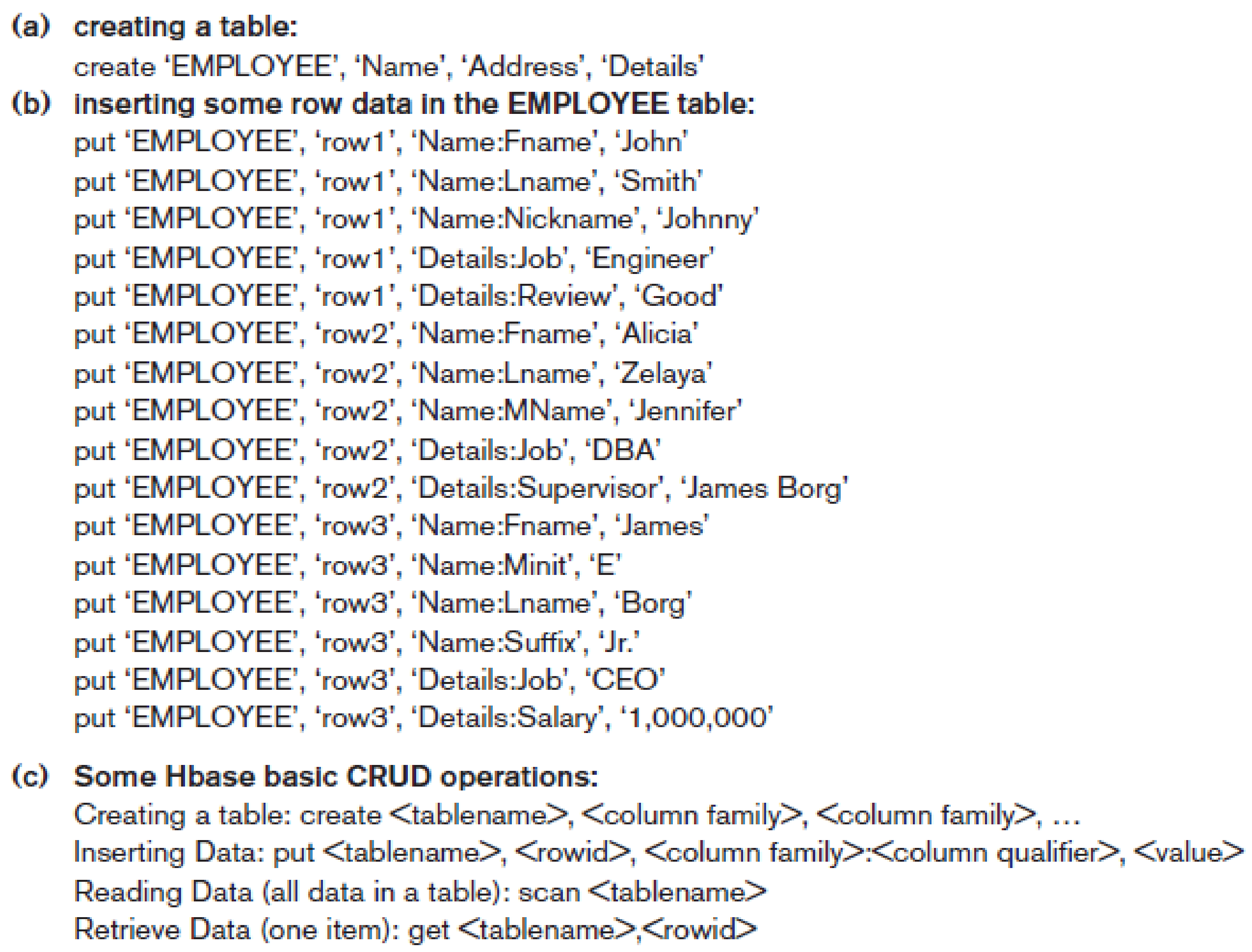
- Sharding in MongoDB
  - Horizontal partitioning divides the documents into disjoint partitions (shards)
  - Allows adding more nodes as needed
  - Shards stored on different nodes to achieve load balancing
  - Partitioning field (shard key) must exist in every document in the collection
    - Must 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

f) Column-Based / Wide Column NoSQL Systems

- BigTable: Google’s distributed storage system for big data
  - Used in Gmail and Uses Google File System for data storage and distribution
- Apache Hbase a similar, open-source system
  - Uses Hadoop Distributed File System (HDFS) for data storage
  - Can also use Amazon’s Simple Storage System (S3)
- Hbase Data Model and Versioning
  - Data organization concepts
    - Namespaces, Tables, Column families, Column qualifiers, Columns, Rows, Data cells
  - Data is self-describing
- HBase stores multiple versions of data items
  - Timestamp associated with each version

Column-Based / Wide Column NOSQL Systems

  - Each row in a table has a unique row key
  - Table associated with one or more column families
- Column qualifiers can be dynamically specified as new table rows are created and inserted
- Namespace is collection of tables, Cell holds a basic data item
- Provides only low-level CRUD (create, read, update, delete) operations
- Application programs implement more complex operations
  - Create: Creates a new table and specifies one or more column families associated with the table
  - Put : Inserts new data or new versions of existing data items
  - Get : Retrieves data associated with a single row
  - Scan : Retrieves all the rows



(a) creating a table:

```
create 'EMPLOYEE', 'Name', 'Address', 'Details'
```

(b) inserting some row data in the EMPLOYEE table:

```
put 'EMPLOYEE', 'row1', 'Name:Fname', 'John'
put 'EMPLOYEE', 'row1', 'Name:Lname', 'Smith'
put 'EMPLOYEE', 'row1', 'Name:Nickname', 'Johnny'
put 'EMPLOYEE', 'row1', 'Details:Job', 'Engineer'
put 'EMPLOYEE', 'row1', 'Details:Review', 'Good'
put 'EMPLOYEE', 'row2', 'Name:Fname', 'Alicia'
put 'EMPLOYEE', 'row2', 'Name:Lname', 'Zelaya'
put 'EMPLOYEE', 'row2', 'Name:MName', 'Jennifer'
put 'EMPLOYEE', 'row2', 'Details:Job', 'DBA'
put 'EMPLOYEE', 'row2', 'Details:Supervisor', 'James Borg'
put 'EMPLOYEE', 'row3', 'Name:Fname', 'James'
put 'EMPLOYEE', 'row3', 'Name:Minit', 'E'
put 'EMPLOYEE', 'row3', 'Name:Lname', 'Borg'
put 'EMPLOYEE', 'row3', 'Name:Suffix', 'Jr.'
put 'EMPLOYEE', 'row3', 'Details:Job', 'CEO'
put 'EMPLOYEE', 'row3', 'Details:Salary', '1,000,000'
```

(c) Some Hbase basic CRUD operations:

```
Creating a table: create <tablename>, <column family>, <column family>, …
Inserting Data: put <tablename>, <rowid>, <column family>:<column qualifier>, <value>
Reading Data (all data in a table): scan <tablename>
Retrieve Data (one item): get <tablename>,<rowid>
```

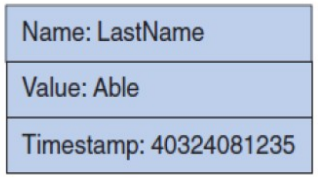
Examples in Hbase (a) Creating a table called EMPLOYEE with three column families: Name, Address, and Details (b) Inserting some in the EMPLOYEE table; different rows can have different self-describing column qualifiers (Fname, Lname, Nickname, Mname, Minit, Suffix, … for column family Name; Job, Review, Supervisor, Salary for column family Details). (c) Some CRUD operations of Hbase

Hbase Storage and Distributed System Concepts

- Each Hbase table divided into several regions
  - Each region holds a range of the row keys in the table
  - Row keys must be lexicographically ordered
  - Each region has several stores
    - Column families are assigned to stores
- Regions assigned to region servers for storage
  - Master server responsible for monitoring the region servers
- Hbase uses Apache Zookeeper and HDFS

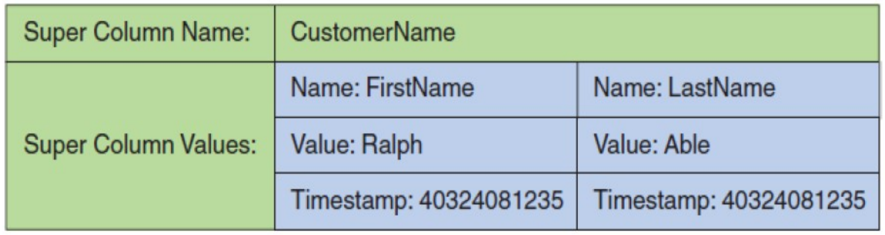
Column-Based / Wide Column NOSQL Systems

- A generalized column family database storage system is shown in Fig. The smallest unit of storage is called a **column.**
- A column consists of three elements: the *column name*, the *column value* or datum, and a *timestamp* to record when the value was stored in the column.
- This is shown in Fig-(a) by the LastName column, which stores the LastName value 'Able'.



Name: LastName
Value: Able
Timestamp: 40324081235

(a) A Column



Super Column Name:	CustomerName	
Super Column Values:	Name: FirstName	Name: LastName
	Value: Ralph	Value: Able
	Timestamp: 40324081235	Timestamp: 40324081235

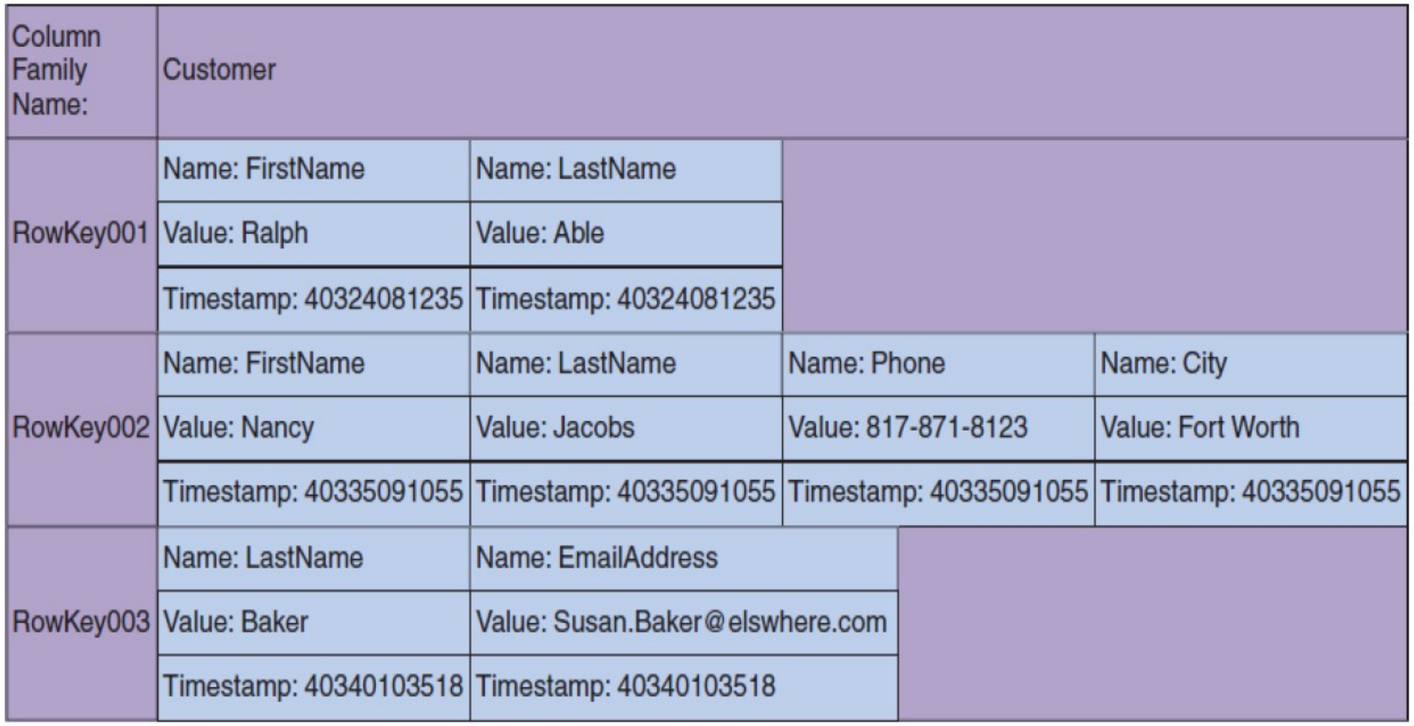
(b) A Super Column

Column-Based / Wide Column NOSQL Systems

- Columns can be grouped into sets referred to as **super columns.**
- This is shown in Fig-(b) by the CustomerName super column, which consists of a FirstName column and a LastName column and which stores the CustomerName value ‘Ralph Able’.
- Columns and super columns are grouped to create **column families**, which are the column family database storage equivalent of RDBMS tables, as they are typically stored together.
- This is illustrated in Fig-(c) by the Customer column family, which consists of three rows of data on customers.
- Note that the first row has no Phone or City columns, while the third row not has FirstName, Phone, or City columns, but also contains an EmailAddress column that does not exist in the other rows.

Column-Based / Wide Column NOSQL Systems

- All the column families are contained in a keyspace, which provides the set of RowKey values that can be used in the data store. RowKey values from the keyspace are shown being used in Fig-C to identify each row in a **column family**

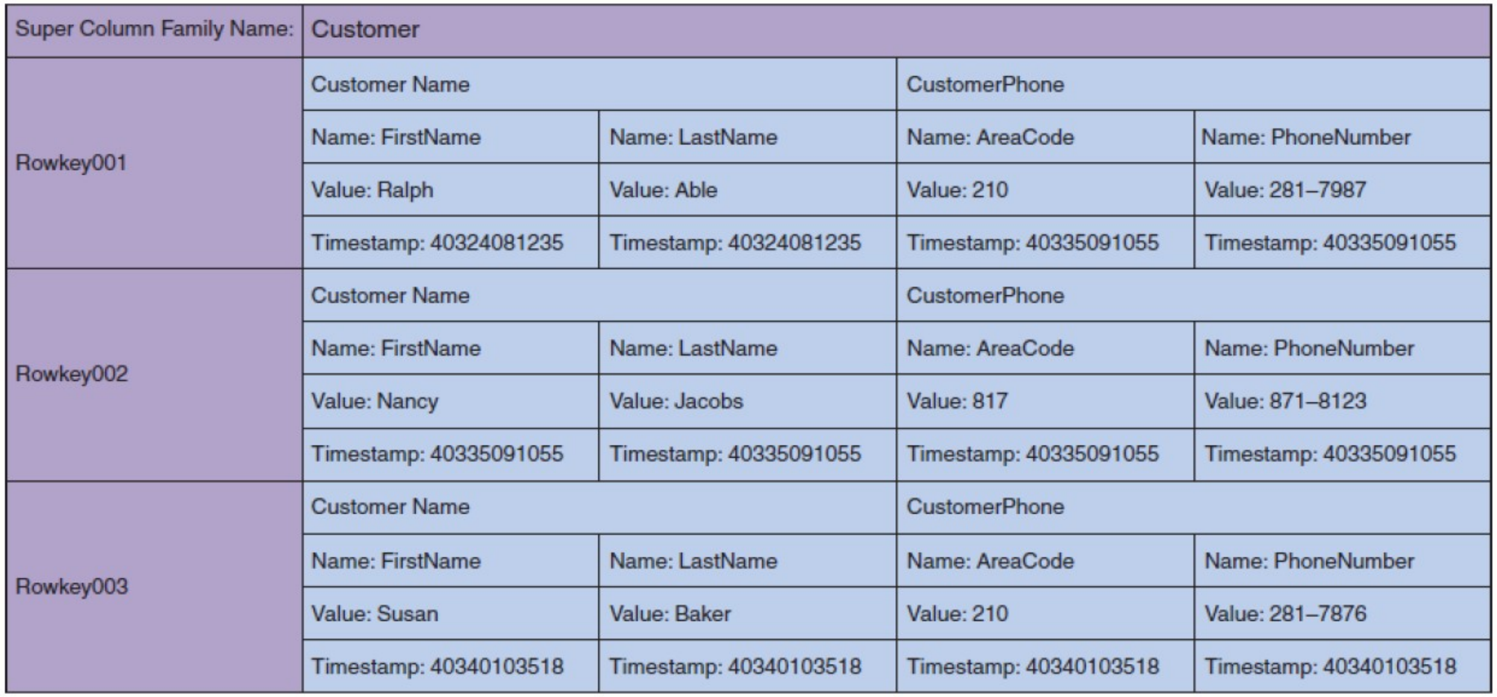


Column Family Name:	Customer			
RowKey001	Name: FirstName	Name: LastName		
	Value: Ralph	Value: Able		
	Timestamp: 40324081235	Timestamp: 40324081235		
RowKey002	Name: FirstName	Name: LastName	Name: Phone	Name: City
	Value: Nancy	Value: Jacobs	Value: 817-871-8123	Value: Fort Worth
	Timestamp: 40335091055	Timestamp: 40335091055	Timestamp: 40335091055	Timestamp: 40335091055
RowKey003	Name: LastName	Name: EmailAddress		
	Value: Baker	Value: Susan.Baker@elswhere.com		
	Timestamp: 40340103518	Timestamp: 40340103518		

(c) A Column Family

Column-Based / Wide Column NOSQL Systems

- As shown in Fig-(d), a **super column family** is similar to a column family, but uses super columns (or a combination of columns and super columns) instead of columns. A super column is a named collection of related columns.
- The Cassandra DBMS supports super columns.



Super Column Family Name:	Customer			
Rowkey001	Customer Name		CustomerPhone	
	Name: FirstName	Name: LastName	Name: AreaCode	Name: PhoneNumber
	Value: Ralph	Value: Able	Value: 210	Value: 281–7987
	Timestamp: 40324081235	Timestamp: 40324081235	Timestamp: 40335091055	Timestamp: 40335091055
Rowkey002	Customer Name		CustomerPhone	
	Name: FirstName	Name: LastName	Name: AreaCode	Name: PhoneNumber
	Value: Nancy	Value: Jacobs	Value: 817	Value: 871–8123
	Timestamp: 40335091055	Timestamp: 40335091055	Timestamp: 40335091055	Timestamp: 40335091055
Rowkey003	Customer Name		CustomerPhone	
	Name: FirstName	Name: LastName	Name: AreaCode	Name: PhoneNumber
	Value: Susan	Value: Baker	Value: 210	Value: 281–7876
	Timestamp: 40340103518	Timestamp: 40340103518	Timestamp: 40340103518	Timestamp: 40340103518

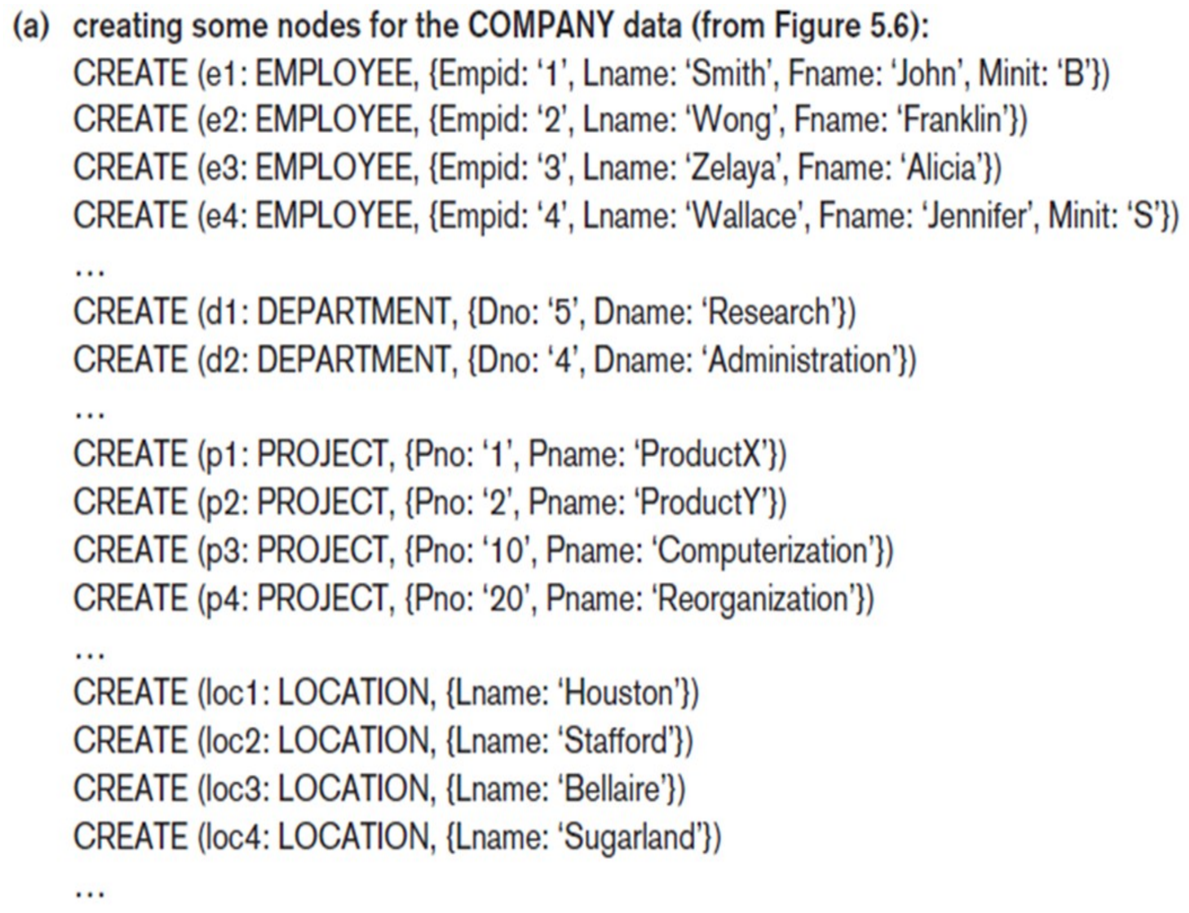
(d) A Super Column Family

g) NoSQL Graph Databases and Neo4j

- Graph databases
  - Data represented as a graph
  - Collection of vertices (nodes) and edges
  - Possible to store data associated with both individual nodes and individual edges
- Neo4j
  - Open-source system, Uses concepts of nodes and relationships
  - Nodes can have labels, Zero, one, or several
  - Both nodes and relationships can have properties
  - Each relationship has a start node, end node, and a relationship type
  - Properties specified using a map pattern

NOSQL Graph Databases and Neo4j

- Creating nodes in Neo4j
  - CREATE command



(a) **creating some nodes for the COMPANY data (from Figure 5.6):**

```
CREATE (e1: EMPLOYEE, {Empid: ‘1’, Lname: ‘Smith’, Fname: ‘John’, Minit: ‘B’})
CREATE (e2: EMPLOYEE, {Empid: ‘2’, Lname: ‘Wong’, Fname: ‘Franklin’})
CREATE (e3: EMPLOYEE, {Empid: ‘3’, Lname: ‘Zelaya’, Fname: ‘Alicia’})
CREATE (e4: EMPLOYEE, {Empid: ‘4’, Lname: ‘Wallace’, Fname: ‘Jennifer’, Minit: ‘S’})
…
CREATE (d1: DEPARTMENT, {Dno: ‘5’, Dname: ‘Research’})
CREATE (d2: DEPARTMENT, {Dno: ‘4’, Dname: ‘Administration’})
…
CREATE (p1: PROJECT, {Pno: ‘1’, Pname: ‘ProductX’})
CREATE (p2: PROJECT, {Pno: ‘2’, Pname: ‘ProductY’})
CREATE (p3: PROJECT, {Pno: ‘10’, Pname: ‘Computerization’})
CREATE (p4: PROJECT, {Pno: ‘20’, Pname: ‘Reorganization’})
…
CREATE (loc1: LOCATION, {Lname: ‘Houston’})
CREATE (loc2: LOCATION, {Lname: ‘Stafford’})
CREATE (loc3: LOCATION, {Lname: ‘Bellaire’})
CREATE (loc4: LOCATION, {Lname: ‘Sugarland’})
…
```

Figure 24.4 Examples in Neo4j using the Cypher language (a) Creating some nodes

NOSQL Graph Databases and Neo4j

- Part of high-level declarative query language **Cypher**
- Node label can be specified when node is created
- Properties are enclosed in curly brackets

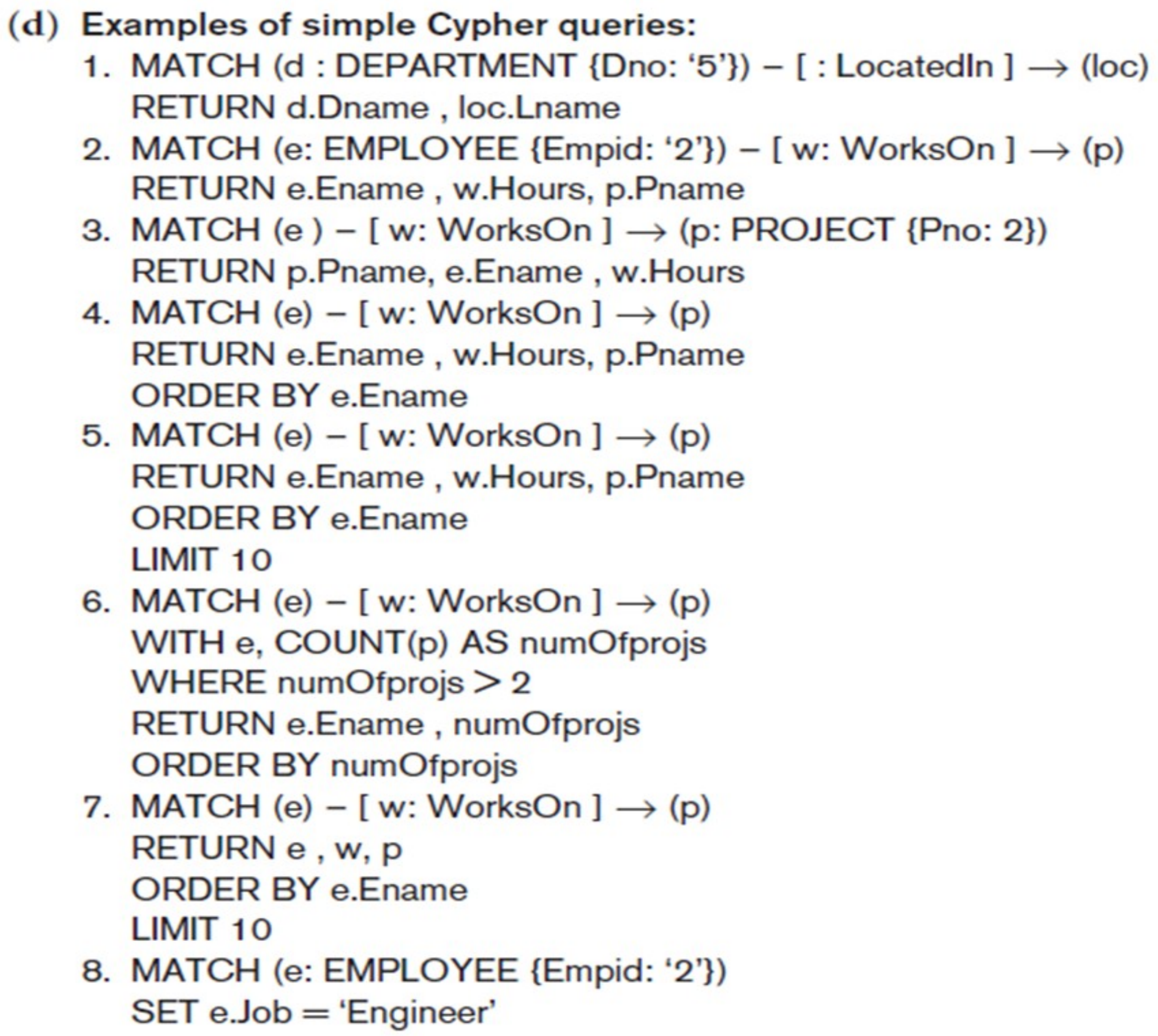
Figure 24.4 (b) Examples in Neo4j using the Cypher language (b) Creating some relationships

(b) creating some relationships for the COMPANY data (from Figure 5.6):

```
CREATE (e1) – [ : WorksFor ] –> (d1)
CREATE (e3) – [ : WorksFor ] –> (d2)
…
CREATE (d1) – [ : Manager ] –> (e2)
CREATE (d2) – [ : Manager ] –> (e4)
…
CREATE (d1) – [ : LocatedIn ] –> (loc1)
CREATE (d1) – [ : LocatedIn ] –> (loc3)
CREATE (d1) – [ : LocatedIn ] –> (loc4)
CREATE (d2) – [ : LocatedIn ] –> (loc2)
…
CREATE (e1) – [ : WorksOn, {Hours: '32.5'} ] –> (p1)
CREATE (e1) – [ : WorksOn, {Hours: '7.5'} ] –> (p2)
CREATE (e2) – [ : WorksOn, {Hours: '10.0'} ] –> (p1)
CREATE (e2) – [ : WorksOn, {Hours: 10.0} ] –> (p2)
CREATE (e2) – [ : WorksOn, {Hours: '10.0'} ] –> (p3)
CREATE (e2) – [ : WorksOn, {Hours: 10.0} ] –> (p4)
…
```

NOSQL Graph Databases and Neo4j

Figure 24.4 (cont'd.) Examples in Neo4j using the Cypher language (d) Examples of Cypher queries

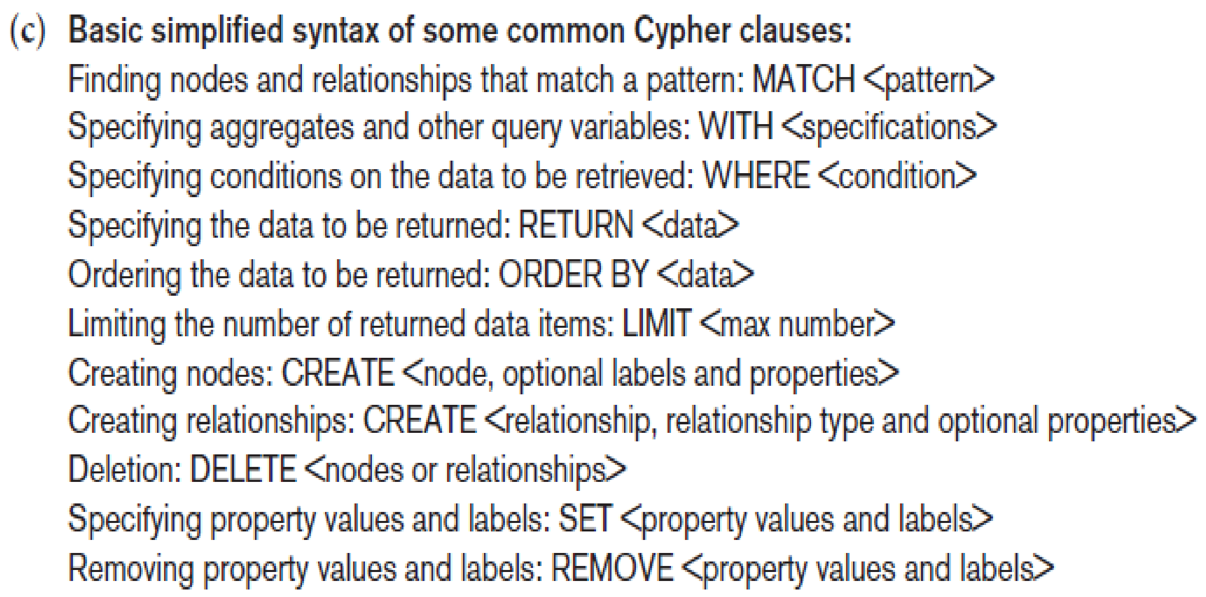


(d) Examples of simple Cypher queries:

```
1. MATCH (d : DEPARTMENT {Dno: '5'}) – [ : LocatedIn ] ⟶ (loc)
   RETURN d.Dname , loc.Lname
2. MATCH (e: EMPLOYEE {Empid: '2'}) – [ w: WorksOn ] ⟶ (p)
   RETURN e.Ename , w.Hours, p.Pname
3. MATCH (e ) – [ w: WorksOn ] ⟶ (p: PROJECT {Pno: 2})
   RETURN p.Pname, e.Ename , w.Hours
4. MATCH (e) – [ w: WorksOn ] ⟶ (p)
   RETURN e.Ename , w.Hours, p.Pname
   ORDER BY e.Ename
5. MATCH (e) – [ w: WorksOn ] ⟶ (p)
   RETURN e.Ename , w.Hours, p.Pname
   ORDER BY e.Ename
   LIMIT 10
6. MATCH (e) – [ w: WorksOn ] ⟶ (p)
   WITH e, COUNT(p) AS numOfprojs
   WHERE numOfprojs > 2
   RETURN e.Ename , numOfprojs
   ORDER BY numOfprojs
7. MATCH (e) – [ w: WorksOn ] ⟶ (p)
   RETURN e , w, p
   ORDER BY e.Ename
   LIMIT 10
8. MATCH (e: EMPLOYEE {Empid: '2'})
   SET e.Job = 'Engineer'
```

NOSQL Graph Databases and Neo4j

- Path
  - Traversal of part of the graph
  - Typically used as part of a query to specify a pattern
- Schema optional in Neo4j. Indexing and node identifiers
  - Users can create for the collection of nodes that have a particular label
  - One or more properties can be indexed
- Cypher query made up of clauses. Result from one clause can be the input to the next clause in the query



(c) **Basic simplified syntax of some common Cypher clauses:**

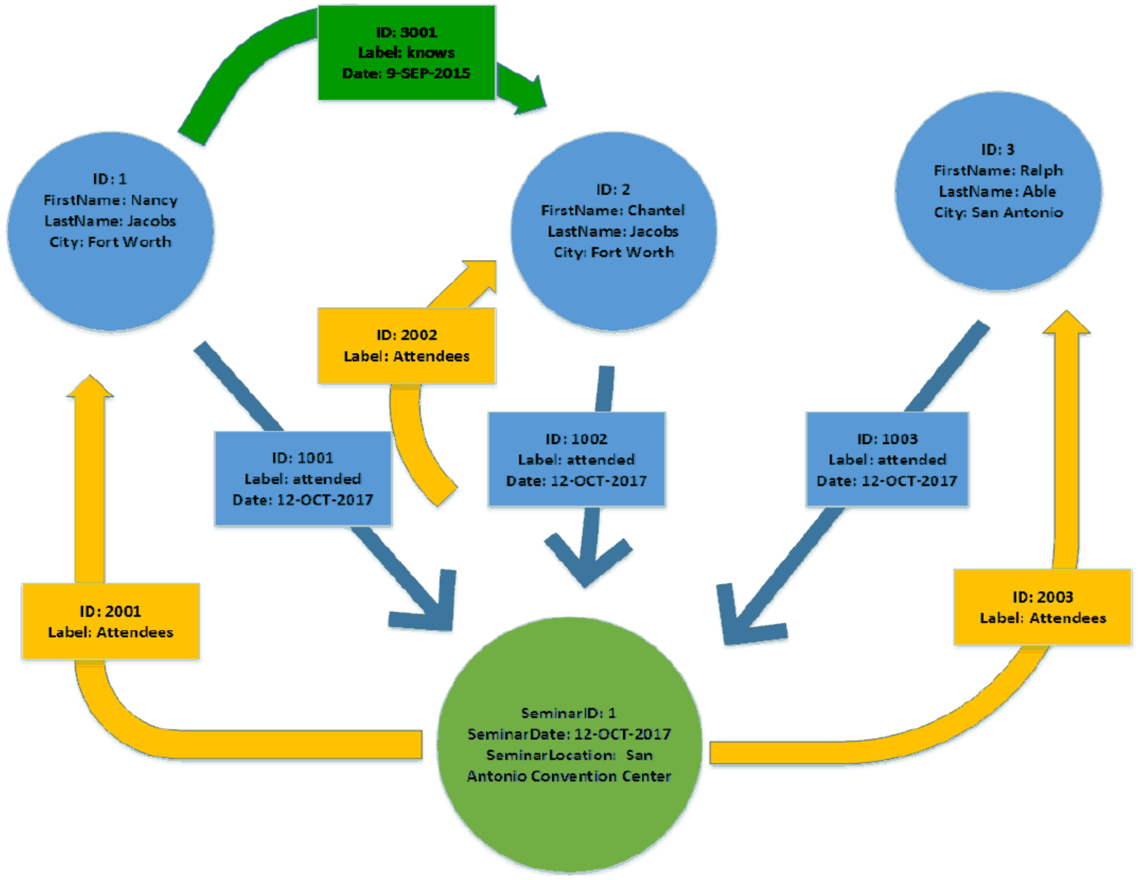
Finding nodes and relationships that match a pattern: MATCH <pattern>
Specifying aggregates and other query variables: WITH <specifications>
Specifying conditions on the data to be retrieved: WHERE <condition>
Specifying the data to be returned: RETURN <data>
Ordering the data to be returned: ORDER BY <data>
Limiting the number of returned data items: LIMIT <max number>
Creating nodes: CREATE <node, optional labels and properties>
Creating relationships: CREATE <relationship, relationship type and optional properties>
Deletion: DELETE <nodes or relationships>
Specifying property values and labels: SET <property values and labels>
Removing property values and labels: REMOVE <property values and labels>

Figure 24.4 Examples in Neo4j using the Cypher language (c) Basic syntax of Cypher queries

NOSQL Graph Databases and Neo4j

- Ex: Figure shows a partial graph database based on part of the SEMINAR DATABASE. SEMINAR, CUSTOMER, and SEMINAR_CUSTOMER and the data about what customer attended which seminar are here.

- The graph database adds edges labeled *Attendees* for a group named Attendees. Similarly, the edge with ID 3001 and labeled *knows* adds the relationships between customers.



NOSQL Graph Databases and Neo4j

- Enterprise edition v/s community edition
  - Enterprise edition supports caching, clustering of data, and locking.
  - Both editions support the Neo4j graph data model and storage system, as well as the Cypher graph query language, and several other interfaces, including a high-performance native API, language drivers for several popular programming languages, such as Java, Python, PHP, and the REST (Representational State Transfer) API.
- Graph visualization interface
  - Subset of nodes and edges in a database graph can be displayed as a graph
  - Used to visualize query results
- Master-slave replication
- Caching
- Logical logs

Chapter end questions

- 24.1. For which types of applications were NOSQL systems developed?
- 24.2. What are the main categories of NOSQL systems? List a few of the NOSQL systems in each category.
- 24.3. What are the main characteristics of NOSQL systems?
- 24.5. What is the CAP theorem? Which of the three properties are most important in NOSQL systems?
- 24.7. What are the data modeling concepts used in MongoDB?
- 24.8. Discuss how replication and sharding are done in MongoDB.

Chapter end questions

- 24.9. Discuss the data modeling concepts in DynamoDB and Data types of DynamoDB.
- 24.11. What are the data modeling concepts used in column-based NOSQL systems and Hbase?
- 24.12. What are the main CRUD operations in Hbase? Discuss the storage and distributed system methods used in Hbase.
- 24.14. What are the data modeling concepts used in the graph-oriented NOSQL system Neo4j?
- 24.15. What is the query language for Neo4j?
- 24.16. Discuss the interfaces and distributed systems characteristics of Neo4j.