Written by Dr. Thapanapong Rukkanchanunt

10 NoSQL Database

OUTLINE

- NoSQL Introduction
- Relational-NoSQL Trade-offs
- Firebase Database
  - Model and Queries
- Project: Checkpoint 3

Timeline for Data Model

- Hierarchical (IBM IMS) – 1960s, 1970s
- Network, CODASYL (Beckman, IDS) – 1960s
- Relational – 1970s
- Object-relational (Stonebraker, et al) – 1990s
- OODMBS (Atkinson, et al) – 1990s
- Array Database (MonetDB, SciDB, …) – 1990s
- XML (document-oriented) – 2000s
- NoSQL – 2010s
- NewSQL – 2011

Why NoSQL

- ฐานข้อมูลในปัจจุบันไม่สามารถแก้ไขได้ทุก
- ความต้องการในแอพรุ่นใหม่อาจจะไม่สามารถตอบสนองได้ด้วย
- บริษัทใหญ่ ๆ ที่ให้บริการบนเว็บต่าง (เช่น Google, Facebook, Instagram) พัฒนาฐานข้อมูลโดยเฉพาะเจาะจง เพื่อแก้ปัญหาใหม่ ๆ ที่พบเจอ

The FOUR V's of Big Data

From traffic patterns and music downloads to web history and medical records, data is recorded, stored, and analyzed to enable the technology and services that the world relies on every day. But what exactly is big data, and how can these massive amounts of data be used?

As a leader in the sector, IBM data scientists break big data into four dimensions: **Volume, Velocity, Variety and Veracity**

Depending on the industry and organization, big data encompasses information from multiple internal and external sources such as transactions, social media, enterprise content, sensors and mobile devices. Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.

By 2015
4.4 MILLION IT JOBS
will be created globally to support big data, with 1.9 million in the United States

Volume
SCALE OF DATA

40 ZETTABYTES
[43 TRILLION GIGABYTES]
of data will be created by 2020, an increase of 300 times from 2005

It's estimated that
2.5 QUINTILLION BYTES
[2.5 TRILLION GIGABYTES]
of data are created each day

6 BILLION PEOPLE
have cell phones

Most companies in the U.S. have at least
100 TERABYTES
[100,000 GIGABYTES]
of data stored

Variety
DIFFERENT FORMS OF DATA

As of 2011, the global size of data in healthcare was estimated to be
150 EXABYTES
[161 BILLION GIGABYTES]

By 2014, it's anticipated there will be
420 MILLION WEARABLE, WIRELESS HEALTH MONITORS

4 BILLION+ HOURS OF VIDEO
are watched on YouTube each month

30 BILLION PIECES OF CONTENT
are shared on Facebook every month

400 MILLION TWEETS
are sent per day by about 200 million monthly active users

Velocity
ANALYSIS OF STREAMING DATA

The New York Stock Exchange captures
1 TB OF TRADE INFORMATION
during each trading session

Modern cars have close to
100 SENSORS
that monitor items such as fuel level and tire pressure

By 2016, it is projected there will be
18.9 BILLION NETWORK CONNECTIONS
– almost 2.5 connections per person on earth

Veracity
UNCERTAINTY OF DATA

1 IN 3 BUSINESS LEADERS
don't trust the information they use to make decisions

in one survey were unsure of how much of their data was inaccurate

Poor data quality costs the US economy around
$3.1 TRILLION A YEAR

Sources: McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS

IBM

Two Problems with Big Data

- ปัญหาการวิเคราะห์ข้อมูลขนาดใหญ่
  - ทำอย่างไรให้เราสามารถดึงข้อมูลที่ต้องการได้ ใช้ Modeling, Machine Learning, หรือสถิติ? (Facebook เลือกคนที่เราควรเป็นเพื่อนจากหลายล้านคนที่ใช้ Facebook)
- ปัญหาการเก็บข้อมูลขนาดใหญ่
  - ต้องเก็บข้อมูลแบบไหนถึงจะสามารถเข้าถึงเข้ามูลได้รวดเร็ว (Google Search ใช้เวลาค้นหาไม่ถึงวินาที)
- ปัญหาเหล่านี้ไม่สามารถแก้ไขได้ด้วยฐานข้อมูลแบบเดิม ๆ (เช่น Relational)
  - ไม่มีความยืดหยุ่น
  - ยากในการแบ่งข้อมูล เช่น แบ่งไฟล์ไว้หลาย ๆ เครื่อง

Storage Problem

- ปัญหา: คลังสินค้ามีของหลายประเภท แต่ละประเภทมีจำนวนคุณสมบัติต่างกัน
- เราต้องการ Schema ที่ยืดหยุ่นได้
- ถ้าจะแก้ปัญหานี้ด้วย Relational
  - สร้างตารางสำหรับสินค้าแต่ละประเภท
  - ใส่ทุกอย่างในตารางเดียว
  - ใช้ Inheritance
  - ใช้ Entity–Attribute–Value
  - ใส่ทุกอย่างใน BLOB (Binary Large Object)

First Solution: Table per Product

```
CREATE TABLE 'product_audio_album'
( 'sku' char(8) NOT NULL, ...
'artist' varchar(255) DEFAULT NULL,
'genre_0' varchar(255) DEFAULT NULL,
'genre_1' varchar(255) DEFAULT NULL,
...
PRIMARY KEY('sku')) ...
```

```
CREATE TABLE 'product_film'
( 'sku' char(8) NOT NULL, ...
'title' varchar(255) DEFAULT NULL,
'rating' char(8) DEFAULT NULL, ...
PRIMARY KEY('sku')) ...
```

Second Solution: Table for all

```
CREATE TABLE 'product'
( 'sku' char(8) NOT NULL, ...
'artist' varchar(255) DEFAULT NULL,
'genre_0' varchar(255) DEFAULT NULL,
'genre_1' varchar(255) DEFAULT NULL, ...
'title' varchar(255) DEFAULT NULL,
'rating' char(8) DEFAULT NULL, ...
PRIMARY KEY('sku'))
```

Third Solution: Inheritance

```
CREATE TABLE 'product'
( 'sku' char(8) NOT NULL,
'title' varchar(255) DEFAULT NULL,
'description' varchar(255) DEFAULT NULL,
'price', ...
PRIMARY KEY('sku'))
```

```
CREATE TABLE 'product_audio_album'
( 'sku' char(8) NOT NULL, ...
'artist' varchar(255) DEFAULT NULL,
'genre_0' varchar(255) DEFAULT NULL,
'genre_1' varchar(255) DEFAULT NULL, ...
PRIMARY KEY('sku'),
FOREIGN KEY('sku') REFERENCES
'product'('sku')) ....
```

Fourth Solution: Entity–Attribute–Value

Entity	Attribute	Value
sku_00e8da9b	Type	Audio Album
sku_00e8da9b	Title	A Love Supreme
sku_00e8da9b	…	…
sku_00e8da9b	Artist	John Coltrane
sku_00e8da9b	Genre	Jazz
sku_00e8da9b	Genre	General

NoSQL Solution

- ใช้แนวคิด Key-Value

```
{ sku: "00e8da9b",
  type: "Audio Album",
  title: "A Love Supreme",
  description: "by John Coltrane",
  shipping: { weight: 6,
    dimensions: { width: 10, height: 10, depth: 1 } },
  pricing: { list: 1200, retail: 1100, savings: 100},
  details: { title: "A Love Supreme [Original Recording]",
    artist: "John Coltrane",
    genre: [ "Jazz", "General" ]}
}
```

Analysis Problem

- จากตัวอย่างก่อนหน้านี้เราได้เห็นการแก้ปัญหาการเก็บข้อมูลที่มีความหลากหลาย แต่การแก้ปัญหาดังกล่าวสามารถช่วยแก้ไขปัญหาเชิงวิเคราะห์ได้หรือไม่ กล่าวคือเราจะแก้ปัญหาข้อมูลจำนวนมากอย่างไร
- Relational Database ใช้วิธีการขยาย Server ให้รองรับปริมาณข้อมูลที่มากขึ้น
  - การแยกข้อมูลไว้หลาย Server ทำให้เกิดความซับซ้อน
- NoSQL สามารถแยกข้อมูลไว้หลาย Server ได้ (Cloud Service)
  - ใช้ Map-Reduce ในการคำนวณข้าม Server

Type of NoSQL

- Key-Value

- Column-Oriented

- Document

Firebase

- Graph

Row-Oriented vs Column-Oriented

Row-oriented: rows stored sequentially in a file

Key	Fname	Lname	State	Zip	Phone	Age	Sales
1	Bugs	Bunny	NY	11217	(123) 938-3235	34	100
2	Yosemite	Sam	CA	95389	(234) 375-6572	52	500
3	Daffy	Duck	NY	10013	(345) 227-1810	35	200
4	Elmer	Fudd	CA	04578	(456) 882-7323	43	10
5	Witch	Hazel	CA	01970	(567) 744-0991	57	250

Column-oriented: each column is stored in a separate file
Each column for a given row is at the same offset.

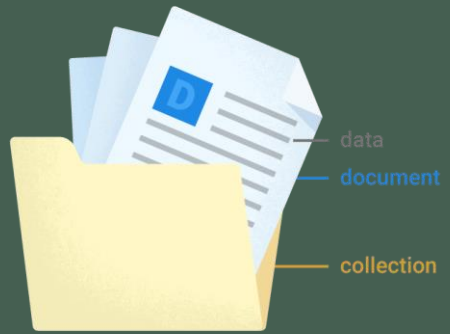
Key	Fname	Lname	State	Zip	Phone	Age	Sales
1	Bugs	Bunny	NY	11217	(123) 938-3235	34	100
2	Yosemite	Sam	CA	95389	(234) 375-6572	52	500
3	Daffy	Duck	NY	10013	(345) 227-1810	35	200
4	Elmer	Fudd	CA	04578	(456) 882-7323	43	10
5	Witch	Hazel	CA	01970	(567) 744-0991	57	250

Relational–NoSQL Trade–offs

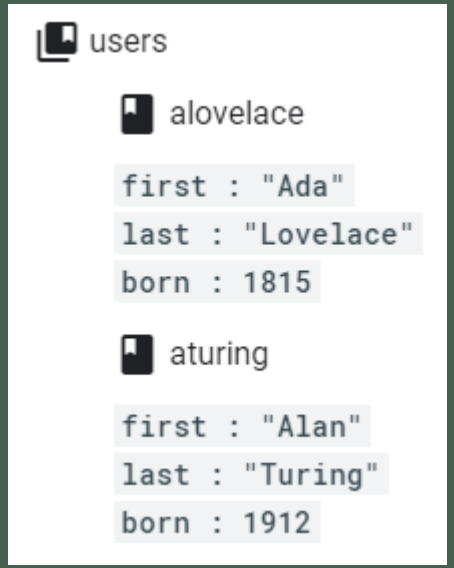
- ข้อดีและข้อเสียระหว่าง Relational Database และ NoSQL Database
  - Schema -> Relational Database มี Schema ทำให้ประมวลเร็ว (รู้ว่าอะไรอยู่ที่ไหน) -> NoSQL ไม่มี Schema ทำให้รองรับข้อมูลได้หลากหลาย แต่ต้องเสียเวลา Parse ข้อมูลตอนแรก
  - Replication, Data Partition -> Replication ทำให้ Query ประมวลเร็วขึ้น แต่อัพเดตข้อมูลช้าลง
  - Level of Abstraction -> NoSQL ใช้ High–level Query ที่ต้อง Parse ข้อมูล จึงต้องคำนึงถึง Optimization ในส่วนนี้ด้วย
  - Consistency -> Relational รองรับการอัพเดตแบบ Concurrent ได้ดีกว่า

Overview of Firestore

- Firestore Database เป็นตัวอย่างฐานข้อมูลแบบ Document-oriented NoSQL
- Firestore Database เป็นส่วนหนึ่งของบริการ Firebase และ Google Cloud Platform
- ข้อมูลที่ดึงมาจาก Firestore จะถูก Sync อยู่ตลอดเวลา



Firestore Data Model



- ใน Firestore เราเก็บข้อมูลในรูปแบบของ Document
- แต่ละ Document จะต้องอยู่ใน Collection ใด Collection หนึ่ง
- แต่ละ Document จะประกอบไปด้วย Key และ Value
  - Value สามารถเป็น Nested Object หรือ List ได้
- แต่ละ Document สามารถระบุเป็นเอกลักษณ์ได้โดยตำแหน่งที่อยู่

```
var alovelaceDocumentRef = db.collection('users').doc('alovelace');
```

- หรือจะระบุเป็น Collection ก็ได้

```
var usersCollectionRef = db.collection('users');
```

Basic Data Types

- Array
- Boolean
- Bytes (up to 1MB)
- Date and Time (microseconds)
- Floating-point Number (64-bit)
- Geographical point (Lat, Long)
- Integer (64-bit, signed)
- Map {a: "aaa", b: "baz"}
- Null
- Reference (path)
- Text String (up to 1MB)

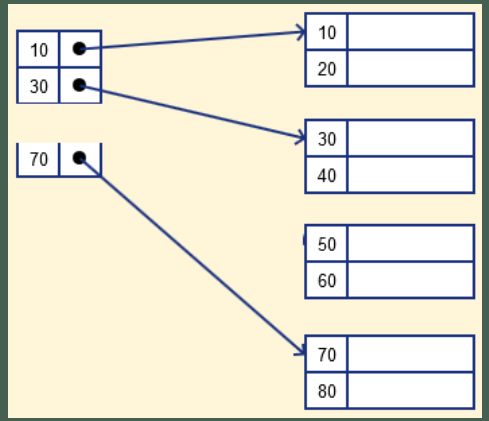
Automatic Indexing

- Firestore จะสร้าง Index ให้กับทุก Field ของ Document อัตโนมัติ

```
var citiesRef = db.collection("cities");

citiesRef.doc("SF").set({
    name: "San Francisco", state: "CA", country: "USA",
    capital: false, population: 860000,
    regions: ["west_coast", "norcal"] });
citiesRef.doc("LA").set({
    name: "Los Angeles", state: "CA", country: "USA",
    capital: false, population: 3900000,
    regions: ["west_coast", "socal"] });
citiesRef.doc("DC").set({
    name: "Washington, D.C.", state: null, country: "USA",
    capital: true, population: 680000,
    regions: ["east_coast"] });
citiesRef.doc("TOK").set({
    name: "Tokyo", state: null, country: "Japan",
    capital: true, population: 9000000,
    regions: ["kanto", "honshu"] });
citiesRef.doc("BJ").set({
    name: "Beijing", state: null, country: "China",
    capital: true, population: 21500000,
    regions: ["jingjinji", "hebei"] });
```

Collection	Field indexed
cities	↑ name
cities	↑ state
cities	↑ country
cities	↑ capital
cities	↑ population
cities	↓ name
cities	↓ state
cities	↓ country
cities	↓ capital
cities	↓ population
cities	`array-contains` regions



Core Firestore Operation

- หลักการ CRUD: Create, Read, Update, Delete
- เพิ่มข้อมูลไปยัง Database โดยการเรียกใช้ฟังก์ชัน set จาก Reference ของ Document

```
var docData = {
    stringExample: "Hello world!",
    booleanExample: true,
    numberExample: 3.14159265,
    dateExample: firebase.firestore.Timestamp.fromDate(new Date("December 10, 1815")),
    arrayExample: [5, true, "hello"],
    nullExample: null,
    objectExample: {
        a: 5,
        b: {
            nested: "foo"
        }
    }
};
db.collection("data").doc("one").set(docData).then(function() {
    console.log("Document successfully written!");
});
```

Reference

Call

Add or Set

- ในการใช้ฟังก์ชัน Set เราจะต้องระบุ ID ของ Document แต่ถ้า ID ไม่มีความหมายเราสามารถใช้ฟังก์ชัน Add แทนได้ โดย Firestore จะสร้าง ID ให้อัตโนมัติ
- แต่ถ้าต้องการสร้าง Reference เราสามารถไม่ระบุ ID ใน doc ได้

```
db.collection("cities").add({
    name: "Tokyo",
    country: "Japan"
})
.then(function(docRef) {
    console.log("Document written with ID: ", docRef.id);
})
.catch(function(error) {
    console.error("Error adding document: ", error);
});
```

```
// Add a new document with a generated id.
var newCityRef = db.collection("cities").doc();

// later...
newCityRef.set(data);
```

Update

- ถ้าต้องการอัพเดตข้อมูลเฉพาะ Document ใช้ฟังก์ชัน update
- ใช้ฟังก์ชัน Timestamp ถ้าต้องการระบุเวลาที่ Server อัพเดตข้อมูล

```
var washingtonRef = db.collection("cities").doc("DC");

// Set the "capital" field of the city 'DC'
return washingtonRef.update({
    capital: true
})
.then(function() {
    console.log("Document successfully updated!");
})
.catch(function(error) {
    // The document probably doesn't exist.
    console.error("Error updating document: ", error);
});
```

```
var docRef = db.collection('objects').doc('some-id');

// Update the timestamp field with the value from the server
var updateTimestamp = docRef.update({
    timestamp: firebase.firestore.FieldValue.serverTimestamp()
});
```

Nested Update

- ถ้าต้องการอัพเดตข้อมูลใน Nested Object เราสามารถเข้าถึงได้ด้วยการใช้ dot notation

```
// Create an initial document to update.
var frankDocRef = db.collection("users").doc("frank");
frankDocRef.set({
    name: "Frank",
    favorites: { food: "Pizza", color: "Blue", subject: "recess" },
    age: 12
});

// To update age and favorite color:
db.collection("users").doc("frank").update({
    "age": 13,
    "favorites.color": "Red"
})
.then(function() {
    console.log("Document successfully updated!");
});
```

Nested Object Warning

- หากเราไม่ใช่ dot notation เราจะแทนที่ข้อมูลทั้งหมดใน Nested Object

```
db.collection("users").doc("frank").set({
  name: "Frank",
  favorites: {
    food: "Pizza",
    color: "Blue",
    subject: "Recess"
  },
  age: 12
}).then(function() {
  console.log("Frank created");
});
```

```
db.collection("users").doc("frank").update({
  favorites: {
    food: "Ice Cream"
  }
}).then(function() {
  console.log("Frank food updated");
});
```

```
/users
    /frank
        {
            name: "Frank",
            favorites: {
                food: "Ice Cream",
            },
            age: 12
        }
 */
```

Update Array Element

- หากข้อมูลใน Document เป็นลักษณะ Array เราสามารถเพิ่มหรือลดสมาชิกได้ด้วยฟังก์ชัน arrayUnion และ arrayRemove

```
var washingtonRef = db.collection("cities").doc("DC");

// Atomically add a new region to the "regions" array field.
washingtonRef.update({
    regions: firebase.firestore.FieldValue.arrayUnion("greater_virginia")
});

// Atomically remove a region from the "regions" array field.
washingtonRef.update({
    regions: firebase.firestore.FieldValue.arrayRemove("east_coast")
});
```

Increment / Decrement

- เราสามารถอัพเดตข้อมูลที่เป็น Numeric ด้วยการเพิ่มหรือลดค่า โดยการใช้ฟังก์ชัน increment
- หาก Document ไม่มี Field ที่กำหนด จะสร้าง Field ใหม่แล้วให้ค่าตามที่กำหนด
- หาก Document มี Field แต่ไม่ใช่ Numeric จะแทนค่า Field ด้วยค่าที่กำหนด

```
var washingtonRef = db.collection('cities').doc('DC');

// Atomically increment the population of the city by 50.
washingtonRef.update({
    population: firebase.firestore.FieldValue.increment(50)
});
```

Delete

- ลบ Document จากฐานข้อมูลด้วยฟังก์ชัน delete

```
db.collection("cities").doc("DC").delete()
```

- ลบ Field ด้วย FieldValue.delete()

```
var cityRef = db.collection('cities').doc('BJ');
var removeCapital = cityRef.update({
    capital: firebase.firestore.FieldValue.delete()
});
```

- เราไม่สามารถลบ Collection จากฝั่ง Client ได้

Get a Document

- ดึงค่า Document เอกสารเดียวจากฐานข้อมูลด้วย get

```
var docRef = db.collection("cities").doc("SF");

docRef.get().then(function(doc) {
    if (doc.exists) {
        console.log("Document data:", doc.data());
    } else {
        // doc.data() will be undefined in this case
        console.log("No such document!");
    }
}).catch(function(error) {
    console.log("Error getting document:", error);
});
```

Get all Documents in Collection

- เราสามารถดึงข้อมูลทั้ง Collection ได้ แต่ผลลัพธ์ที่ได้จะมี Meta-data ติดมาด้วย จำเป็นต้อง Map ถ้าจะนำมาใช้ร่วมกับ *ngFor

```
db.collection("cities").get().then(function(querySnapshot) {
    querySnapshot.forEach(function(doc) {
        // doc.data() is never undefined for query doc snapshots
        console.log(doc.id, " => ", doc.data());
    });
});
```

SnapshotChanges

- ใช้ snapshotChanges ในการดึงข้อมูล
- snapshotChanges ได้ข้อมูล Realtime
- Get ได้ข้อมูลครั้งเดียว

```
getTweets() {
  return this.firestore.collection('tweets').snapshotChanges();
}

getTweets2() {
  return this.firestore.collection('tweets').get();
}
```

```
this.fServ.getTweets().subscribe(val => {
  this.tweets = val.map( e => {
    return {
      id: e.payload.doc.id,
      ...e.payload.doc.data()
    } as Tweet
  })
});

this.fServ.getTweets2().subscribe(val => {
  this.tweets = val.docs.map( e => {
    return {
      id: e.id,
      ...e.data()
    } as Tweet
  })
});
```

ValueChanges

- หากเราไม่สนใจ ID เราสามารถใช้ ValueChanges ได้ (ได้ข้อมูล Realtime เหมือนกับ snapshotChanges)

```
getTweets3() {
  return this.firestore.collection('tweets').valueChanges();
}
```

```
this.fServ.getTweets3().subscribe(val => {
  this.tweets = val.map( e => {
    return {
      id: 0,
      ...e
    } as Tweet
  })
});
```

Filtering Data

- หากเราต้องการข้อมูลตามเงื่อนไข เราสามารถเพิ่ม where ได้

```
db.collection("cities").where("capital", "==", true)
    .get()
    .then(function(querySnapshot) {
        querySnapshot.forEach(function(doc) {
            // doc.data() is never undefined for query doc snapshots
            console.log(doc.id, " => ", doc.data());
        });
    })
    .catch(function(error) {
        console.log("Error getting documents: ", error);
    });
```

Chaining Where

- ถ้าต้องการเพิ่มเงื่อนไขในรูปแบบ AND เราสามารถเพิ่ม where ต่อกันได้
- ถ้าใช้ Range Filter จะใช้ได้แค่ Field เดียว

```
citiesRef.where("state", "==", "CO").where("name", "==", "Denver");
citiesRef.where("state", "==", "CA").where("population", "<", 1000000);
```

```
citiesRef.where("state", ">=", "CA").where("population", ">", 100000);
```

Project : Checkpoint 3

- ส่ง Project เบื้องต้น
  - มีข้อมูลในฐานข้อมูล
  - มีการเชื่อมต่อฐานข้อมูลมายังแอพ
  - มี Basic Feature อย่างน้อย 2 อย่างที่เขียนไว้
- เดดไลน์ 27 มีนาคม 2563
- ตั้งชื่อ Repository เป็น รหัสนักศึกษาproject เช่น 123456789project