NoSQL

Uppstod när behovet att undvika data-duplicering slutade vara lika stort. No-SQL-databaser kan ur ett perspektiv anses göra det enklare för programmerare/systemutvecklare att arbeta med datalagring eftersom strukturen på datan bättre överensstämmer med hur man oftast hanterar data i programmeringskod. Det innebär att det ofta inte blir lika mycket jobb med att hantera mappningen av datan från databasen till källkoden och vice versa.

Terminologi SQL vs MongoDB

SQL	MongoDB
Table	Collection
Row	Document
Colums	Field
Primary Key	ObjectId
Index	Index
Nested table or object	Embedded document
Array	Array
View	View

Några olika typer av NoSQL:

- Key-value
- Column
- Graph
- Document

Collection

- NoSQL's version av tabeller.
- Behöver inte schema.
  - Behöver inte samma fält och/eller struktur.
  - Anpassas efter användarens behov
  - Kombinera objekt om dem ska användas tillsammans.
  - Använd aggregation.
  - Join vid insert men bör inte användas vid select.

Key-Value

Den enklaste typen av NoSQL. En Key-Value databas är i grunden en stor hash-tabell, *kan liknas vid ett lexikon.* Varje datavärde kopplas till en unik nyckel *(Key)* och databasen använder sen nyckeln för att lagra data *(Value)* med hjälp av en lämplig hash-funktion.

De flesta Key-Value databaser har endast stöd för enkla Read-, Update- och Deletekommandon. Detta innebär att för att ändra ett värde *(helt eller delvis)* så måste all befintlig data för värdet skrivas över. I majoriteten av alla implementeringar hanteras läsning och/eller skrivning av ett värde som en atomisk *(separat)* åtgärd. Det innebär att hela kommandot måste slutföras; annars återställs värdet till det ursprungliga värdet. Detta arbetssätt innebär också att om värdet som ska skrivas är stort kan det medföra att åtgärden tar lite längre tid att utföra.

En Key-Value databas kan lagra en uppsättning av data i ett värde, vissa implementeringar har dock gränser för storleken på datan som kan lagras i varje värde.

Key-Value databaser är optimerade för program som utför enkla uppslag med hjälp av en eller nycklar. När det kommer till system som behöver hämta data på flera olika platser med hjälp av nycklar och värden, till exempel koppla ihop data över flera tabeller, är en Key-Value databas ett mindre lämpligt val. Key-Value databaser är inte heller optimerade för situationer där frågor och/eller filtrering efter icke-nyckelvärden är en viktig funktion. Med en relationsdatabas går det till exempel hitta en post med hjälp av en WHERE-sats för att filtrera kolumner som inte är nyckelkolumner. En Key-Value databas däremot har vanligtvis inte den typen av sök- och filtreringsfunktion för värden, och om det finns så kräver det oftast en långsam genomsökning av samtliga värden.

Key (Nyckel)

Varje Key måste vara unik då den används för att identifiera varje individuellt Value. Kan vara, nästan, vad som helst; beroende på eventuella restriktioner från programvaran.

Value (Värde)

Value är datan som lagras, alternativt en referens till den plats där datan lagras. Kan vara allt från en textsträng, lista, ett Key-Value par till ett objekt. Vissa programvaror tillåter att användaren specificerar datatypen för Value.

Phone directory

Key	Value
Paul	(091) 9786453778
Greg	(091) 9686154559
Marco	(091) 9868564334

MAC table

Key	Value
10.94.214.172	3c:22:fb:86:c1:b1
10.94.214.173	00:0a:95:9d:68:16
10.94.214.174	3c:1b:fb:45:c4:b1

Column

En kolumndatabas lagrar data i kolumner och rader, likt ett excelark. En kolumndatabas kan i sin enklaste form liknas vid en relationsdatabas, åtminstone konceptuellt. Den verkliga kraften hos en kolumndatabas ligger i dess avormaliserade metod för att strukturera glesa data, som kommer från den kolumnorienterade metoden för att lagra data.

Man kan tänka på en kolumndatabas som en tabell med rader och kolumner, men att kolumnerna är indelade i grupper som kallas *kolumnfamiljer*. Varje kolumnfamilj innehåller en uppsättning kolumner som är logiskt relaterade och som i regel hämtas och/eller ändras som en enhet. Andra data som kan nås separat kan lagras i separata kolumnserier. I en kolumnserie kan nya kolumner läggas till dynamiskt, och en rad kan hantera null-värden. *(dvs. en rad behöver inte ha ett värde för varje kolumn).*

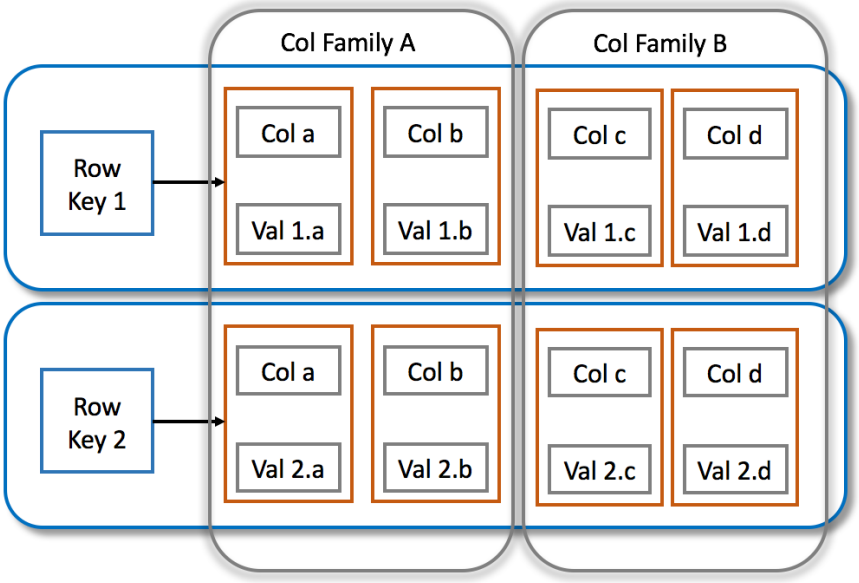
Till skillnad från en Key-Value databas eller en dokumentdatabas lagrar de flesta kolumndatabaser data fysiskt i nyckelordning, istället för att beräkna och tillämpa en hash. Radnyckeln anses vara det primära indexet och möjliggör nyckelbaserad åtkomst via en eller flera nycklar. Vissa implementeringar tillåter skapandet av sekundära index över flertalet kolumner i en kolumnfamilj, med hjälp av dessa sekundära index kan du hämta data baserat kolumnvärde istället för radnyckel.

Vid lagring på disk lagras alla kolumner i en kolumnfamilj tillsammans i en och samma fil, med ett visst antal rader i varje fil. Vid stora datamängder skapar detta en prestandaförmån genom att minska mängden data som behöver läsas från disken när endast ett fåtal kolumner efterfrågas tillsammans åt gången.

Läs- och skrivåtgärder för en rad är vanligtvis atomiska inom en enda kolumnfamilj, vissa implementeringar möjliggör dock atomicitet över hela raden; som sträcker sig över flera kolumnfamiljer.

Kort summering

- Bygger på rader och kolumner.
- Kolumner kan grupperas till “kolumnfamiljer”.
- Kolumner är logiskt relaterade och läses/ändras som en enhet.
- Lagring sker i nyckelordning.
- Rader kan hantera null-värden.
- Lagring på disk delas upp i mindre filer med ett antal rader i varje.

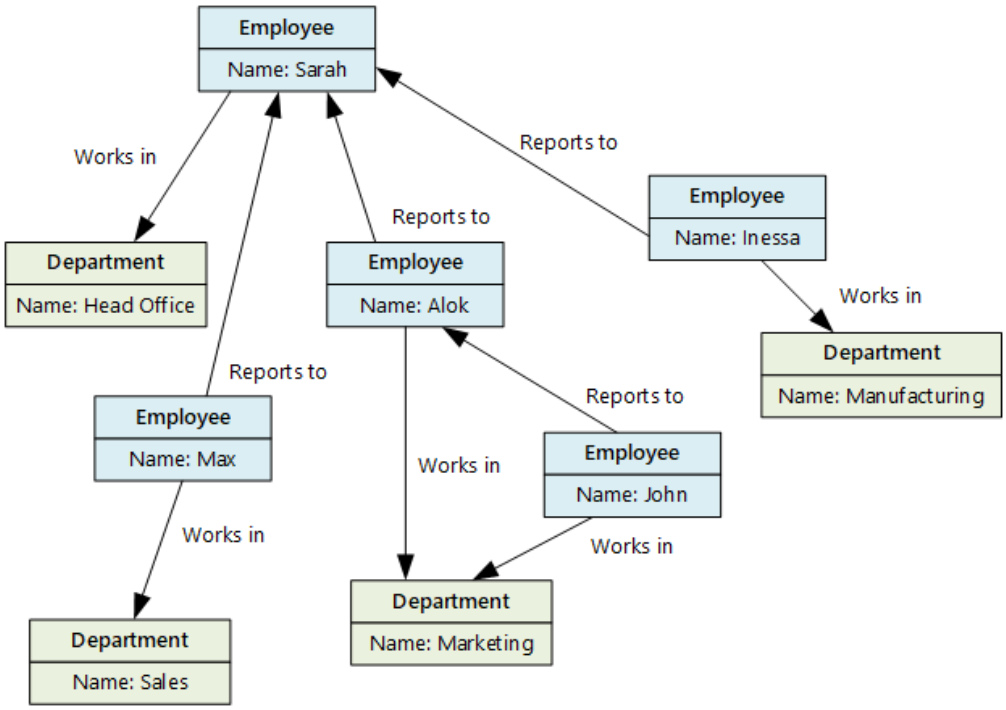


Graph

En grafdatabas hanterar två typer av information, ***noder*** och ***kanter***. Noder representerar *entiteter* medans kanter anger *relationerna mellan entiteterna*. Både noder och kanter kan ha egenskaper som innehåller information om den specifika noden eller kanten, vilket liknar kolumnerna i en tabell. Kanter kan även ha en riktning som visar typen av relation.

Huvudsyftet med en grafdatabas är att applikationer så effektivt som möjligt ska kunna köra queries som passerar nätverket, bestående av noder och kanter, och analysera relationerna mellan entiteterna.

Diagrammet nedan visar en grafdatabas för personaldata inom en organisation. Medarbetare och avdelningar är noder *(entiteter)*, och kanterna visar på rapporteringsvägar samt vid vilken avdelning medarbetarna arbetar. I diagrammet visar pilarna vid kanterna riktningen för varje relation.



Strukturen ovan gör det enkelt att köra queries som "Hitta alla medarbetare som rapporterar direkt eller indirekt till Sarah" samt "Vem arbetar på samma avdelning som John?" I stora grafer med många noder och kanter möjliggör det utförandet av komplexa analyser; snabbt och effektivt.

Kort summering

- Grafdatabas hanterar två typer av information.
- Noder: Entiteter
- Kanter: Relation mellan entiteter.
- Effektivt för komplexa analyser.

Document

En dokumentdatabas består av en uppsättning namngivna fält med tillhörande datavärde, detta lagras i det som kallas *dokument*. Denna typ av databas lagrar som regel data i form av JSON-dokument. Värden som lagras kan vara komplexa och innehålla allt ifrån ett tal till ett sammansatt element; till exempel en lista eller samling.

Fälten i ett dokument kan kodas på en mängd olika sätt, till exempel: XML, YAML, JSON, BSON eller till och med som oformaterad text. Dessa fält i dokumentet blir synliga för systemet som hanterar databasen och gör det möjligt för en applikation att fråga efter, och filtrera, data med hjälp av värdena i varje fält.

En av de stora fördelarna med en dokumentdatabas är att den inte kräver att alla dokument ska samma struktur. Istället för att använda ett databasschema för att designa databasen (vilket man gör med relationsdatabaser) används dokument för att beskriva vilken data som ska lagras i dokumentdatabasen. Enkelt förklarat kan man likna dokument i dokumentdatabaser vid objekt i programmeringens värld. Detta leder till en enorm flexibilitet vid skapandet av dokument och möjliggör lagring av olika data i dokument, som svar på till exempel ändringar i affärskraven.

Ett dokument innehåller vanligtvis all data för en entitet, till exempel information om en kund, en order eller en kombination av båda. Man kan i alltså ett dokument samla information som annars skulle vara utspridd över flera tabeller i en relationsdatabas.

Det är även möjligt att hämta dokument med hjälp av en dokumentnyckel. Dokumentnyckeln är en unik identifierare för dokumentet, som ofta är hash-formaterat, och som hjälper till att fördela data jämnt. Vissa dokumentdatabaser skapar dokumentnyckeln automatiskt medans andra tillåter dig att ange ett attribut för dokumentet som ska användas som nyckel. Det är även möjligt att köra queries mot ett dokument; baserat på värdet för ett eller flera fält.

Det finns även dokumentdatabaser som stöder indexering för att underlätta snabb sökning efter ett dokument; baserat på ett eller flera indexerade fält.

Flertalet dokumentdatabaser har stöd för uppdateringar på plats, detta gör det möjligt att ändra värdena i specifika fält i ett dokument utan att hela dokumentet behöver skrivas om. Vanligtvis är läs- och skrivåtgärder över flera fält i ett dokument atomiska åtgärder.

Kort summering:

- Data lagras i flexibla “dokument”.
- .json liknande format.
- Innehållet i varje dokument kan variera beroende på behovet.
- Datastrukturen kan förändras.
- Flexibelt och skalbart.

Key	Document

1001	<pre>{
	  "CustomerID": 99,
	  "OrderItems": [
	  { "ProductID": 2010,
	  "Quantity": 2,
	  "Cost": 520
	  },
	  { "ProductID": 4365,
	  "Quantity": 1,
	  "Cost": 18
	  }],
	  "OrderDate": "04/01/2017"
	}</pre>

1002	<pre>{
	  "CustomerID": 220,
	  "OrderItems": [
	  { "ProductID": 1285,
	  "Quantity": 1,
	  "Cost": 120
	  }],
	  "OrderDate": "05/08/2017"
	}</pre>

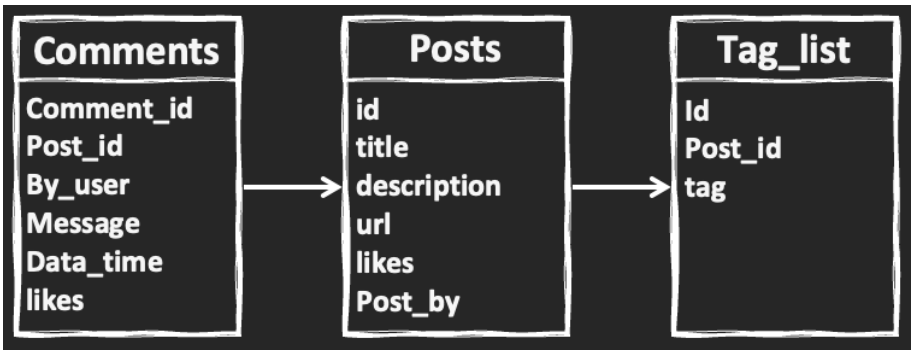
Exempel - Blogg

För att illustrera hur det kan se ut i en relationsdatabas respektive en icke-relationsdatabas kommer vi använda en blogg som exempel. I detta fall blir det en dokumentdatabas som får representera No-SQL.

Kravställning:

- Varje inlägg ska ha title, description och URL.
- Varje inlägg kan ha en eller flera taggar.
- Varje inlägg ska innehålla användarnamn samt totalt antal likes.
- Varje inlägg kan ha kommentarer, 0-*.
    - Dessa ska innehålla:
        - Användarnamn
        - Kommentar
        - Tidpunkt
        - Likes

Relationsdatabas (T.ex. MySQL)



3 tabeller

- **Kommentarer**
  - Comment_id
    - Kommentarens unika id
  - Post_id
    - Id för det inlägg som kommentaren tillhör
  - By_user
    - Vilken användare som skrev kommentaren
  - Message
    - Den text som kommentaren består av
  - Data_time
    - Tidpunkt som kommentaren skrevs
  - likes
    - Antal likes som kommentaren har fått
- **Posts**
  - id
    - Inläggets unika id
  - title
    - Inläggets titel
  - description
    - Beskrivning av inlägget
  - url
    - Länk till inlägget
  - likes
    - Antal likes som inlägget har fått
  - Post_by
    - Vem som postade inlägget
- **Taggar**
  - id
    - Taggens unika id
  - Post_id
    - Id för det inlägg som taggen tillhör
  - tag
    - Den text som taggen består av

Dokumentdatabas (T.ex. MongoDB)

```
  {
  "_id": "POST_ID",                    // Inläggets unika id
  "title": "TITLE_OF_POST",            // Inläggets titel
  "description": "POST_DESCRIPTION",   // Beskrivning av inlägget
  "by": "POST_BY",                     // Vem som postade inlägget
  "url": "URL_OF_POST",                // Länk till inlägget
  "tags": "[TAG1, TAG2, TAG3]",        // Taggar som hör till inlägget
  "likes": "TOTAL_LIKES",              // Antal likes som inlägget har fått
  "comments": [                        // En samling av eventuella kommentarer
    {
      "user": "COMMENT_BY",            // Vilken användare som skrev kommentaren
      "message": "TEXT",               // Den text som kommentaren består av
      "dateCreated": "DATE_TIME",      // Tidpunkt som kommentaren skrevs
      "like": "LIKES"                  // Antal likes som kommentaren har fått
    },
    {
      "user": "COMMENT_BY",            // Vilken användare som skrev kommentaren
      "message": "TEXT",               // Den text som kommentaren består av
      "dateCreated": "DATE_TIME",      // Tidpunkt som kommentaren skrevs
      "like": "LIKES"                  // Antal likes som kommentaren har fått
    }
  ]
}
```

En Collection med ett dokument istället för 3 tabeller.

- **Post**
  - All data lagras i samma dokument.
  - Allt hör till samma inlägg och behöver inte användas på flera ställen.
  - En kommentar hör till exempel till ett specifikt inlägg.
  - Inget annat dokument behöver innehålla den specifika kommentaren.

Exempel - Kommandon MongoDB

Visa alla databaser
```
show dbs
```

Visa aktuell databas
```
db
```

Skapa eller byt databas
```
use [databasnamn]
```

Radera databas
```
[databasnamn].dropDatabase()
```

Skapa collection
```
[databasnamn].createCollection([collectionnamn])
```

Visa collections
```
show collections
```

Skapa dokument
```
[databasnamn].[collectionnamn].insert({
  title: 'No-SQL Demo',
  category: 'Education',
  tags: ['NoSQL', 'Database'],
  user: {
    name: 'Eva Hagner',
    status: 'Teacher'
  },
  students: 100,
  date: Date()
})
```

Skapa flera dokument
```
[databasnamn].[collectionnamn].insertMany([
  {
    title: 'Document One',
    category: 'EdUcation.',
    views: 23,
    date: Date()
  },
  {
    title: 'DocumentTwo',
    category: 'Edumacation',
    views: 2,
    date: Date()
  },
  {
    title: 'Document Three',
    category: 'Education',
    views: 56,
    date: Date()
  }
])
```

Uppdatera dokument

```
[databasnamn].[collectionnamn].update({ title: 'DocumentTwo' },
{
  title: 'Document Two',
  category: 'Education'
  date: Date()
},
{
  upsert: true // Uppdatera, eller skapa om dokumentet inte existerar.
})
```

Uppdatera specifikt fält

```
[databasnamn].[collectionnamn].update({ title: 'Document One' },
{
  $set: {
    category: 'Education'
  }
})
```

Döpa om fält

```
[databasnamn].[collectionnamn].update({ title: 'No-SQL Demo' },
{
  $rename: {
    user: 'codic'
  }
})
```

Radera dokument

```
[databasnamn].[collectionnamn].remove({ title: 'Document Three'})
```

Hämta alla dokument

```
[databasnamn].[collectionnamn].find()
```

Hämta alla dokument - med formattering

```
[databasnamn].[collectionnamn].find().pretty()
```

Hitta dokument

```
[databasnamn].[collectionnamn].find({category: 'Education'})
```

Räkna dokument

```
[databasnamn].[collectionnamn].find().count()
```

Större & mindre än

```
[databasnamn].[collectionnamn].find({ views: { $gt: 30 } })   // Större än
[databasnamn].[collectionnamn].find({ views: { $gte: 30 } })  // Större än eller lika med
[databasnamn].[collectionnamn].find({ views: { $lt: 10 } })   // Mindre än
[databasnamn].[collectionnamn].find({ views: { $lte: 10 } })  // Mindre än eller lika med
```