資料庫設計(NoSQL)

授課講師 郭惠民

教材編寫 郭惠民

緯育 TibaMe

- ◆模組1：NoSQL之興起
- ◆模組2：CAP及BASE定理
- ◆模組3：NoSQL之種類與特性
- ◆模組4：NoSQL塑模原理
- ◆模組5：MongoDB資料庫
- ◆模組6：Redis資料庫

教材編著者＆授課講師介紹

授課講師

郭惠民

isaachmkuo@gmail.com

老師的話

緯育 TibaMe

學習本課程須知

先備知識

任一種關聯式資料庫之架構與SQL指令之操作

學習目標

A. 了解NoSQL之興起與其原理
B. 了解NoSQL之種類與塑模原理
C. 了解MongoDB及Redis之架構與使用

模組1：NoSQL之興起

1-1 關聯式資料庫之特性
1-2 關聯式資料庫面臨的挑戰
1-3 NoSQL之興起與特性

1-1：關聯式資料庫之特性

1.使用SQL標準語言

- 關聯式資料庫使用 SQL 或結構式查詢語言做為主要的通訊介面。美國國家標準協會 (ANSI) 在 1986 年將 SQL 納入標準。所有常見的關聯式資料庫引擎都支援標準 ANSI SQL，而且有些引擎還提供 ANSI SQL 延伸功能，以支援該引擎的專屬功能。
- 您可以利用 SQL 新增、更新、刪除多個橫列的資料、擷取資料子集供交易處理和分析應用程式使用，以及管理資料庫的所有面向。

1-1：關聯式資料庫之特性

2.強調資料完整性 (Data Integrity)

- 資料完整性是指資料整體完備無缺、準確而且一致。關聯式資料庫利用一套限制條件讓資料庫達成資料完整性。其中包括主索引鍵、外部索引鍵、「非空白值」限制條件、「唯一」限制條件、「預設」限制條件、「檢查」限制條件。
- 這些完整性限制條件對表格中的資料實施商業規則，確保資料的準確性和可靠性。除此之外，大部分關聯式資料庫也允許在觸發中嵌入自訂程式碼，當資料庫發生某個動作時便會執行。

1-1：關聯式資料庫之特性

3.提供交易處理 (Transaction)

- 資料庫交易是指透過一連串的操作執行的一或多個 SQL 陳述式，形成單一邏輯工作單位。
- 交易的原則是「全有或全無」，表示整個交易必須做為單一單位完成並寫入資料庫，否則交易的任何個別元件都不應該通過。在關聯式資料庫的術語中，交易的結果包括「認可」(Commit)或「轉返」(Rollback)。處理每一個交易皆採用一致可靠的方式，和其他交易互不相干。

1-1：關聯式資料庫之特性

4.具有ACID的特性

- 所有資料庫交易必須符合 ACID，即不可分割性 (Atomic)、一致性 (Consistent)、獨立性 (Isolated) 和耐用性 (Durable)，以確保資料完整性。
  - 不可分割性要求交易必須整體成功執行，若是交易有一部分操作失敗，整個交易都會失效。
  - 一致性要求做為交易的一部分寫入資料庫的資料，必須遵守所有明定規則以及約束，包括限制條件、級聯、觸發。
  - 獨立性是達成並行控制的重要關鍵，可以確保每一個交易都是獨立的。
  - 持久性要求在一個交易成功完成後，對資料庫所做的變更都是永久性的。

1-2 關聯式資料庫面臨的挑戰

關聯式資料庫面臨的挑戰

- 大數據時代的 3V
  - Volume (數據量)：大量資料的產生、處理、保存
    - 分散儲存
  - Variety (多變性)：資料的形態，包含文字、影音、網頁、串流等半結構性、非結構性的資料
    - NoSQL Database
  - Velocity (時效性)：處理效能是非常關鍵的
    - 平行處理, In-Memory Computing

1-2 關聯式資料庫面臨的挑戰

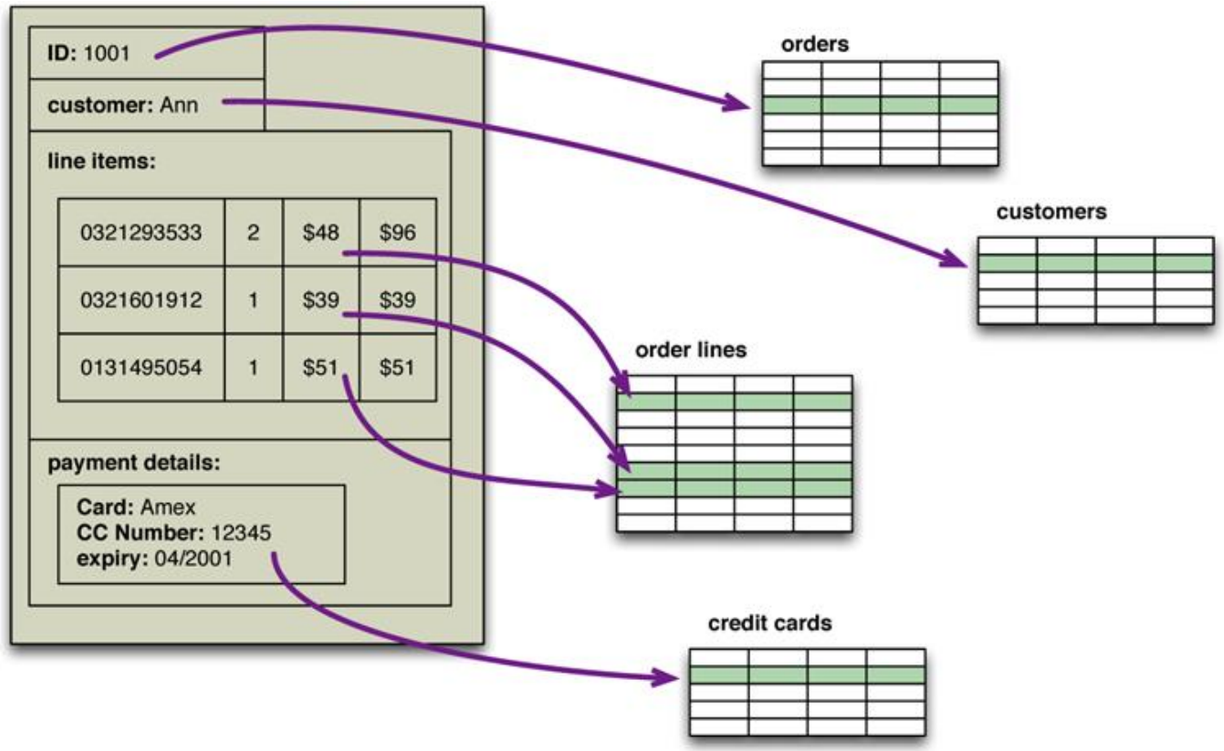
關聯式資料庫面臨的挑戰

- 使用關聯式資料庫的困難
  - 阻抗不匹配 (Impedance Mismatch)
  - 叢集(Clusters)需求

1-2 關聯式資料庫面臨的挑戰

阻抗不匹配

- Impedance mismatch: the difference between the relational model and the in-memory data structures. If you want to use a richer in-memory data structure, you have to translate it to a relational representation to store it on disk.



1-2 關聯式資料庫面臨的挑戰

叢集(Clusters)需求

- Coping with the increase in data and traffic required more computing resources. To handle this kind of increase, you have two choices: up or out.
- Scaling up implies bigger machines, more processors, disk storage, and memory.
- The alternative is to scaling out — use lots of small machines in a cluster. A cluster of small machines can use commodity hardware and ends up being cheaper at these kinds of scales.
- Relational databases are not designed to be run on clusters.
- As the 2000s drew on, two companies produced brief but highly influential papers about their efforts: BigTable from Google and Dynamo from Amazon.

1-3 NoSQL之興起與特性

NoSQL因應而生

- Most people who talk about NoSQL say that it really means “Not Only SQL.”
- NoSQL is an accidental neologism. There is no prescriptive definition.
- It’s better to think of NoSQL as a movement rather than a technology.

1-3 NoSQL之興起與特性

NoSQL之特性

- The common characteristics of NoSQL databases are
  - Not using the relational model
  - Running well on clusters
  - Open-source
  - Built for the 21st century web estates
  - Schemaless
  - The most important result of the rise of NoSQL is Polyglot Persistence.

學習目標：A

模組2：CAP及BASE定理

2-1 CAP定理

2-2 BASE定理

2-3 NoSQL使用時機

2-1 CAP定理

NoSQL CAP 定理

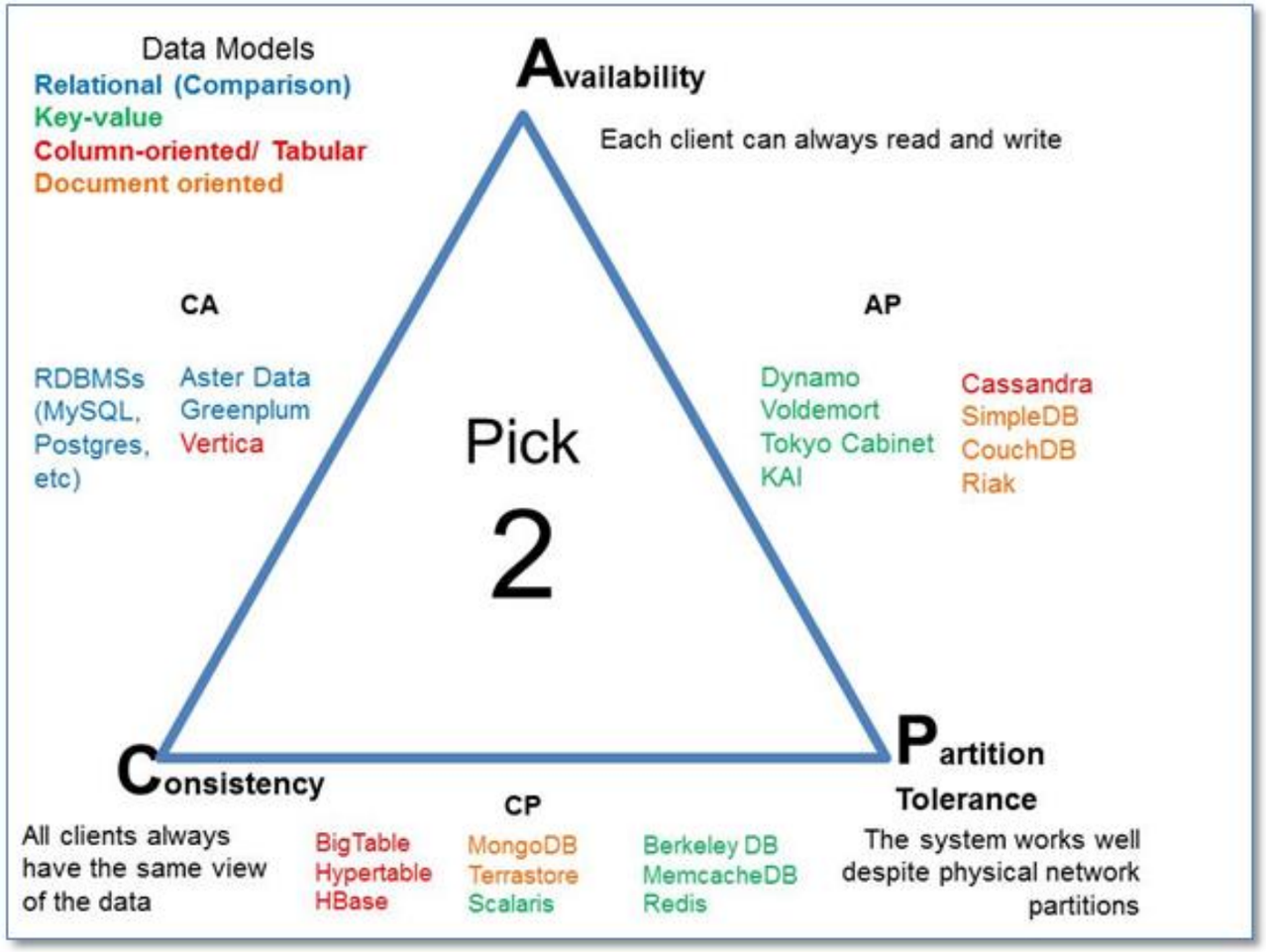
CAP定理, 又被稱作布魯爾定理（Brewer‘s theorem）, 它指出對於一個分散式計算系統來說，不可能同時滿足以下三點:

- Consistency: 一致性 (所有節點在同一時間具有相同的數據) (all nodes see the same data at the same time)
- Availability: 可用性 (保證每個請求不管成功或者失敗都有回應) (a guarantee that every request receives a response about whether it was successful or failed)
- Partition tolerance: 分隔容忍 (系統中任意信息的丟失或失敗不會影響系統的繼續運作) (the system continues to operate despite arbitrary message loss or failure of part of the system)

根據定理，分散式系統只能滿足三項中的兩項而不可能滿足全部三項。

2-1 CAP定理

NoSQL CAP 定理



2-1 CAP定理

NoSQL CAP 定理

- 隨著網際網路應用的快速發展，資料量與日俱增，傳統的ACID資料庫已經不能滿足如此大的海量資料存儲了。
- 這個時候需要設計出好的分散式資料存儲方式。而這些分散式資料存儲方式受到CAP理論的約束，不可能達到高一致性，高可用性，高分區容錯性的完美設計。
- 所以我們在設計的時候要懂得取捨，重點關注對應用需求來說比較重要的，而放棄不重要的，在CAP這三者之間進行取舍，設計出貼合應用的存儲方案。

2-2 BASE定理

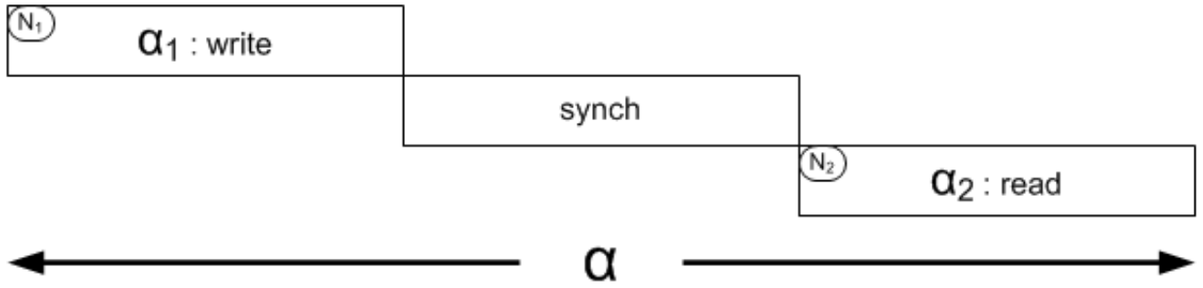
BASE 定理

- BASE(Basically Available, Soft-state Eventually consistent)
- 基本可用(Basically Available) – 意指資料會是可獲得的，但有可能資料庫系統對資料讀取的回應是失敗，或是讀取到的資料是不一致的。
- 軟狀態(Soft state) –意指資料庫系統在尋求一致性時，這段期間資料可能還在變動。
- 最終一致性(Eventual consistency) –意指或快或慢，資料庫系統最終還是會達到一致性。
- The tradeoff between ACID and BASE as a spectrum, not a binary choice.

2-2 BASE定理

最終一致性

兩個節點數據冗餘，第一個節點先有一個寫操作，第二個節點後有一個讀操作。下面的圖中a是整個過程，要具有一致性的話需要等待a1進行write，然後同步到a2，然後a2再進行write，只有整個事務完成以後，a2才能夠進行read。但是這樣的話使得整個系統的可用性下降，a2一直阻塞在那裏等待a1同步到a2。這個時候如果對一致性要求不高的話，a2可以不等待a1數據對於a2的寫同步，直接讀取，這樣雖然此時的讀寫不具有一致性，但是在後面可以通過非同步的方式使得a1和a2的數據最終一致，達到最終一致性。



2-3 NoSQL使用時機

為什麼要用 NoSQL

- 管理大規模資料：NoSQL 資料庫能輕易處理大量的讀寫週期、眾多用戶，以及數以 petabytes 計的資料。

- 資料庫綱要 (Schema)？不需要：多數 NoSQL 資料庫沒有綱要，所以相當彈性。當涉及到綱要建構之時，它們提供了相當廣泛的選擇空間，能輕易地和物件相對應。這表示再也看不到像是正規化與複雜資料連接 (complex joins) 等字眼。

- 開發者親和性：NoSQL 資料庫對各主要程式語言提供了簡單的 API，因此再也用不著複雜的 ORM 框架。如果特定程式語言沒有 API 可用時，還是可以透過簡單的 RESTful API，使用 XML 以及 JSON 格式經由 HTTP 存取資料。

2-3 NoSQL使用時機

為什麼要用 NoSQL

- 可用性：多數分散式 NoSQL 資料庫都提供簡易的資料複製，單一節點的毀損不會過分地影響資料的可用性。
- 延展性：NoSQL 資料庫不需要專用的高效能伺服器。事實上，它們可以輕易地運行在一般硬體組成的叢集上，只要增加新節點就可輕易向外延展。
- 低延遲：除非執行是 Trillion 等級資料伺服器組成的叢集，否則 NoSQL 應該可以協助你達成絕佳的低延遲性。當然，延遲本身與可成功載入記憶體中的資料量有關。

學習目標：A

模組3：NoSQL之種類與特性

3-1 NoSQL之種類

3-2 鍵值資料庫與列導向資料庫特性

3-3 文檔資料庫與圖形資料庫特性

3-1 NoSQL之種類

NoSQL 種類

- 鍵值資料庫(Key-Value Database)
  - Redis, Amazon DynamoDB, Memcached, Riak KV
- 列導向資料庫(Column-Oriented Database)
  - Cassandra, Hbase, Google Cloud Bigtable
- 文檔資料庫(Document-Oriented Database)
  - MongoDB, Amazon DynamoDB, CouchDB
- 圖形資料庫 (Graph Database)
  - Neo4j, Giraph

3-1 NoSQL之種類

常用資料庫排名

360 systems in ranking, November 2020

Rank			DBMS	Database Model	Score		
Nov 2020	Oct 2020	Nov 2019			Nov 2020	Oct 2020	Nov 2019
1.	1.	1.	Oracle	Relational, Multi-model	1345.00	-23.77	+8.93
2.	2.	2.	MySQL	Relational, Multi-model	1241.64	-14.74	-24.64
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1037.64	-5.48	-44.27
4.	4.	4.	PostgreSQL	Relational, Multi-model	555.06	+12.66	+63.99
5.	5.	5.	MongoDB	Document, Multi-model	453.83	+5.81	+40.64
6.	6.	6.	IBM Db2	Relational, Multi-model	161.62	-0.28	-10.98
7.	↑ 8.	↑ 8.	Redis	Key-value, Multi-model	155.42	+2.14	+10.18
8.	↓ 7.	↓ 7.	Elasticsearch	Search engine, Multi-model	151.55	-2.29	+3.15
9.	9.	↑ 11.	SQLite	Relational	123.31	-2.11	+2.29
10.	10.	10.	Cassandra	Wide column	118.75	-0.35	-4.47
11.	11.	↓ 9.	Microsoft Access	Relational	117.23	-1.02	-12.84
12.	12.	↑ 13.	MariaDB	Relational, Multi-model	92.29	+0.52	+6.72
13.	13.	↓ 12.	Splunk	Search engine	89.71	+0.30	+0.64
14.	14.	↑ 15.	Teradata	Relational, Multi-model	75.60	-0.19	-4.75
15.	15.	↓ 14.	Hive	Relational	70.26	+0.71	-13.96
16.	16.	16.	Amazon DynamoDB	Multi-model	68.89	+0.48	+7.52
17.	17.	↑ 25.	Microsoft Azure SQL Database	Relational, Multi-model	66.99	+2.59	+39.37
18.	18.	↑ 19.	SAP Adaptive Server	Relational	55.39	+0.23	+0.10
19.	19.	↑ 20.	SAP HANA	Relational, Multi-model	53.58	-0.66	-1.53
20.	↑ 21.	↑ 22.	Neo4j	Graph	53.53	+2.20	+3.00

3-2 鍵值資料庫與列導向資料庫特性

鍵值資料庫(Key-Value Database)

編號	姓名	生日	薪水	語言
130412	王小明	1980/01/01	38000	英:讀寫 日:聽說讀寫

key	value
130412	王小明, 1980/01/01, 38000,英:讀寫, 日:聽說讀寫

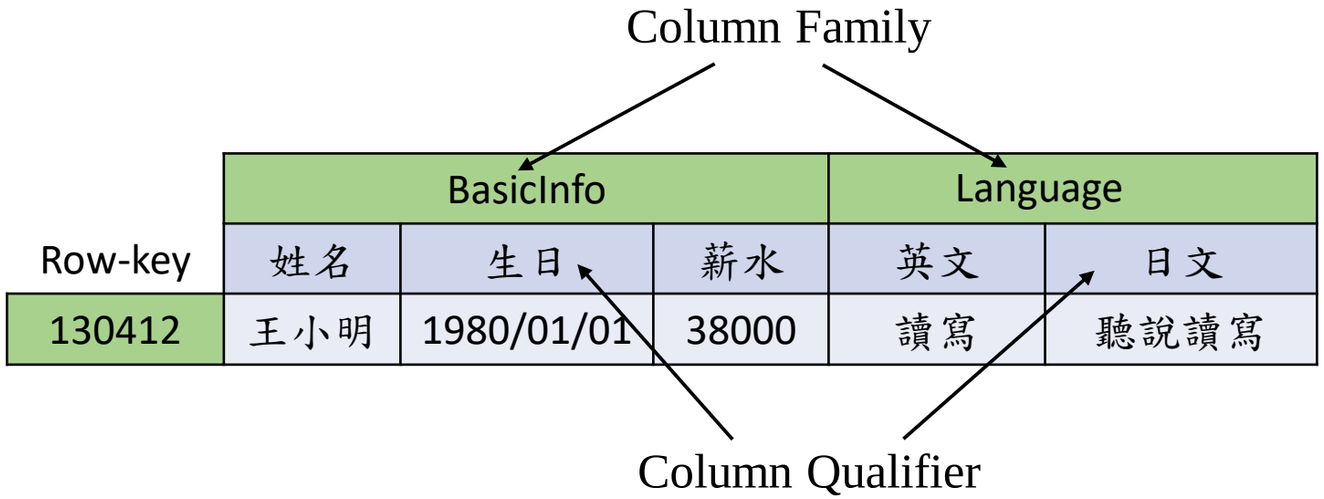
key	value
編號	130412
姓名	王小明
生日	1980/01/01
薪水	38000
語言	英:讀寫, 日:聽說讀寫

key	value
130412-姓名	王小明
130412-生日	1980/01/01
130412-薪水	38000
130412-語言	英:讀寫, 日:聽說讀寫

3-2 鍵值資料庫與列導向資料庫特性

列導向資料庫(Column-Oriented Database)

- 以Hbase為例



(Row key, Column key, Timestamp) → Value

(130212, BasicInfo:姓名) →王小明

3-3 文檔資料庫與圖形資料庫特性

文檔資料庫(Document-Oriented Database)

- 以MongoDB為例

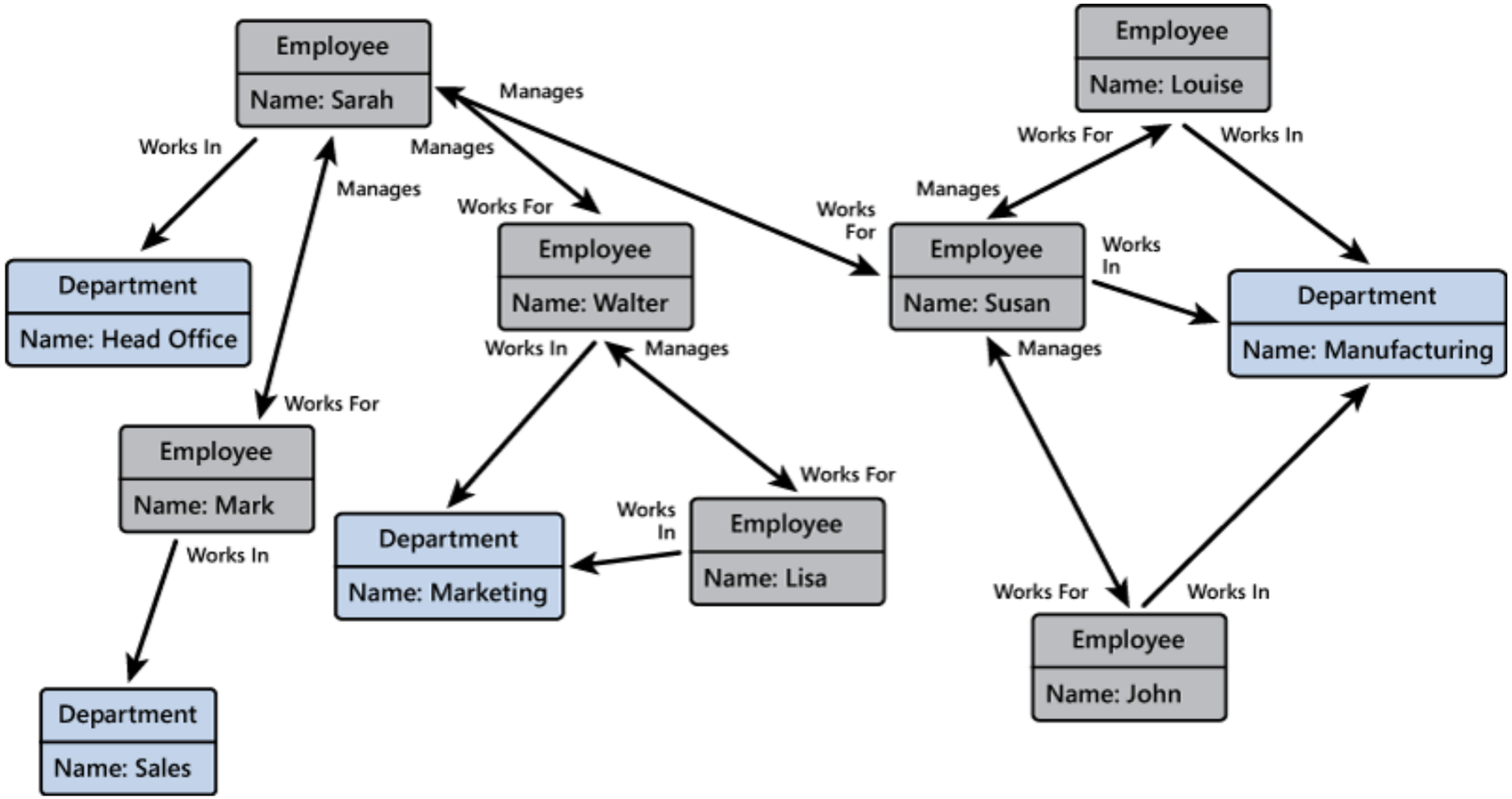
collection: Employee

```
{
  id : “130412”,
 姓名:“王小明”,
 生日: 1980/01/01,
 薪水 : 38000,
 語言 :[{英文:“讀寫”},
        {日文:“聽說讀寫”},
       ],
}
```

RDB	MongoDB
Database	Database
Table	Collection
Record	Document
Field	Field

3-3 文檔資料庫與圖形資料庫特性

圖形資料庫(Graph Database)



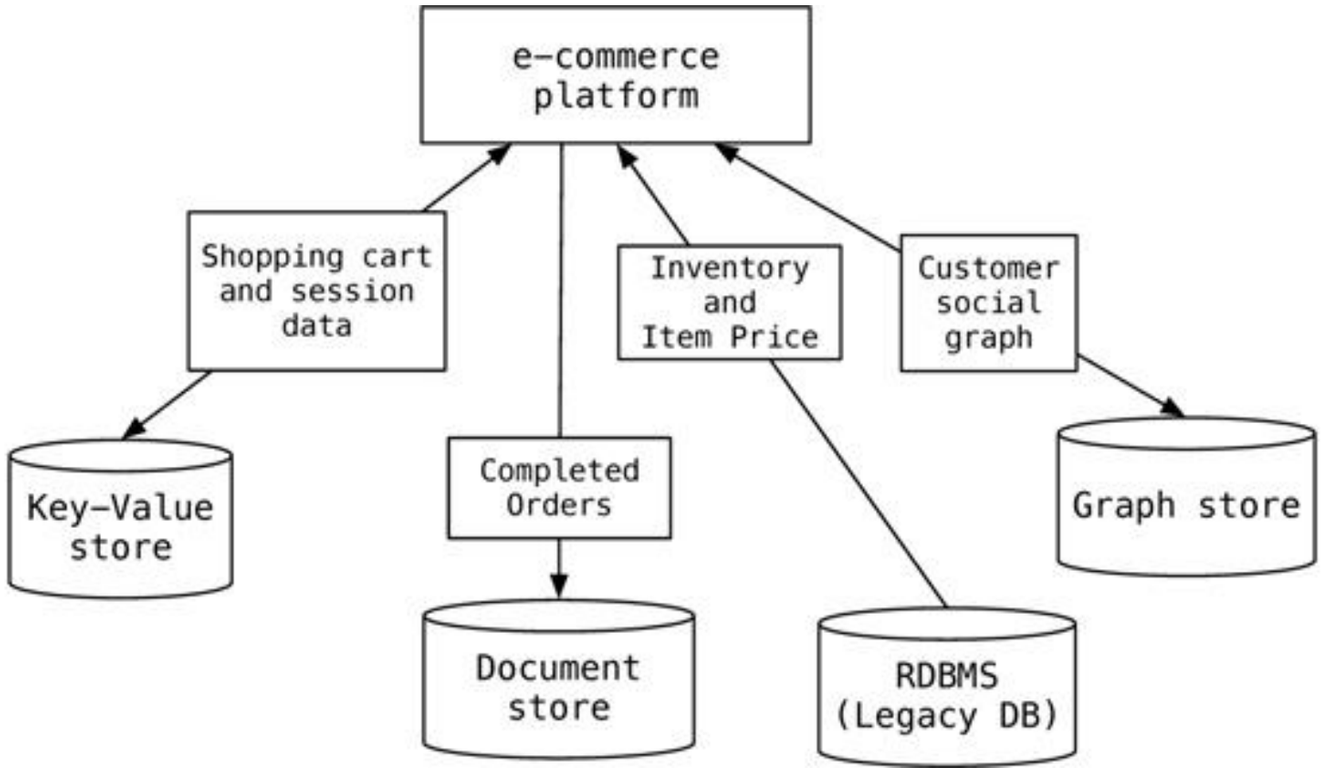
3-3 文檔資料庫與圖形資料庫特性

混合持久性資料庫(Polyglot Persistence)

- Polyglot persistence is about using different data storage technologies to handle varying data storage needs.
- Polyglot persistence can apply across an enterprise or within a single application.
- There is a rich choice of data storage solutions. Initially, the pendulum had shifted from special databases to a single RDBMS database which allows all types of data models to be stored, although with some abstraction. The trend is now shifting back to using the data storage that supports the implementation of solutions natively.

3-3 文檔資料庫與圖形資料庫特性

混合持久性資料庫(Polyglot Persistence)



模組4 : NoSQL塑模原理

4-1 NoSQL 資料庫設計原則
4-2 NoSQL 資料庫設計步驟
4-3 聚集與參考

4-1 NoSQL 資料庫設計原則

NoSQL 相關議題

- 分散式模型 (Distribution Model)
- 一致性(Consistency)

4-1 NoSQL 資料庫設計原則

分散式模型 (Distribution Model)

- Broadly, there are two paths to data distribution: replication and sharding. Replication takes the same data and copies it over multiple nodes. Sharding puts different data on different nodes.
- Replication and sharding are orthogonal techniques. A system may use either or both techniques.
  - 單一伺服器(Single Server)
  - 分片(Sharding)
  - 主從式複製(Master-slave Replication)
  - 對等式複製(Peer-to-peer Replication)
  - 結合分片與複製(Combining Sharding And Replication)

4-1 NoSQL 資料庫設計原則

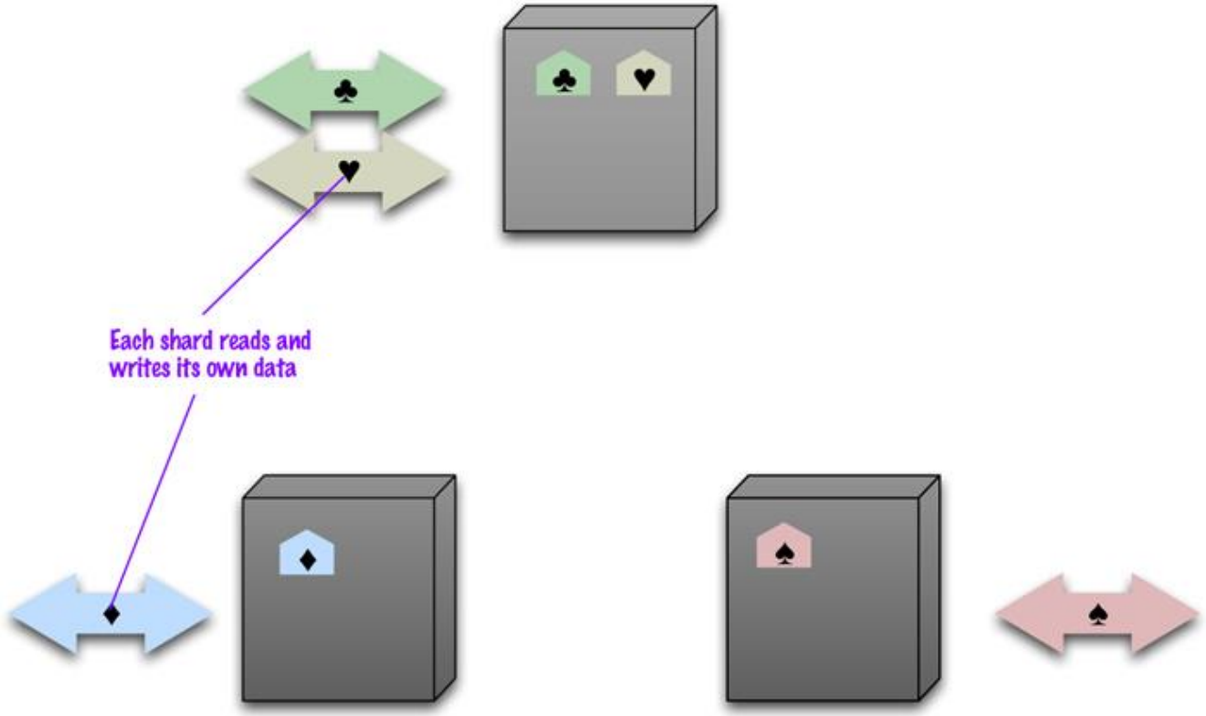
單一伺服器 (Single Server)

- Run the database on a single machine that handles all the reads and writes to the data store.
- We prefer this option because it eliminates all the complexities that the other options introduce; it's easy for operations people to manage and easy for application developers to reason about.
- Although a lot of NoSQL databases are designed around the idea of running on a cluster, it can make sense to use NoSQL with a single-server distribution model if the data model of the NoSQL store is more suited to the application.
- If we can get away without distributing our data, we will always choose a single-server approach.

4-1 NoSQL 資料庫設計原則

分片 (Sharding)

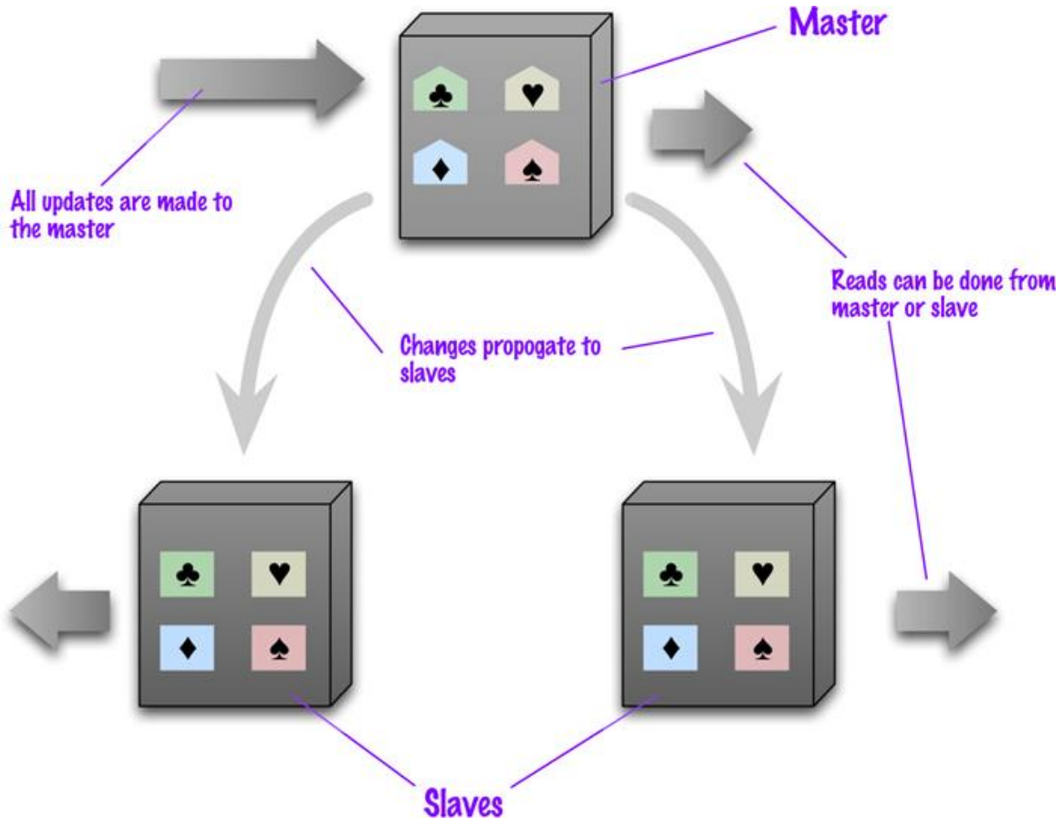
- Sharding — a technique that put different parts of the data onto different servers to support horizontal scalability.



4-1 NoSQL 資料庫設計原則

主從式複製(Master-slave Replication)

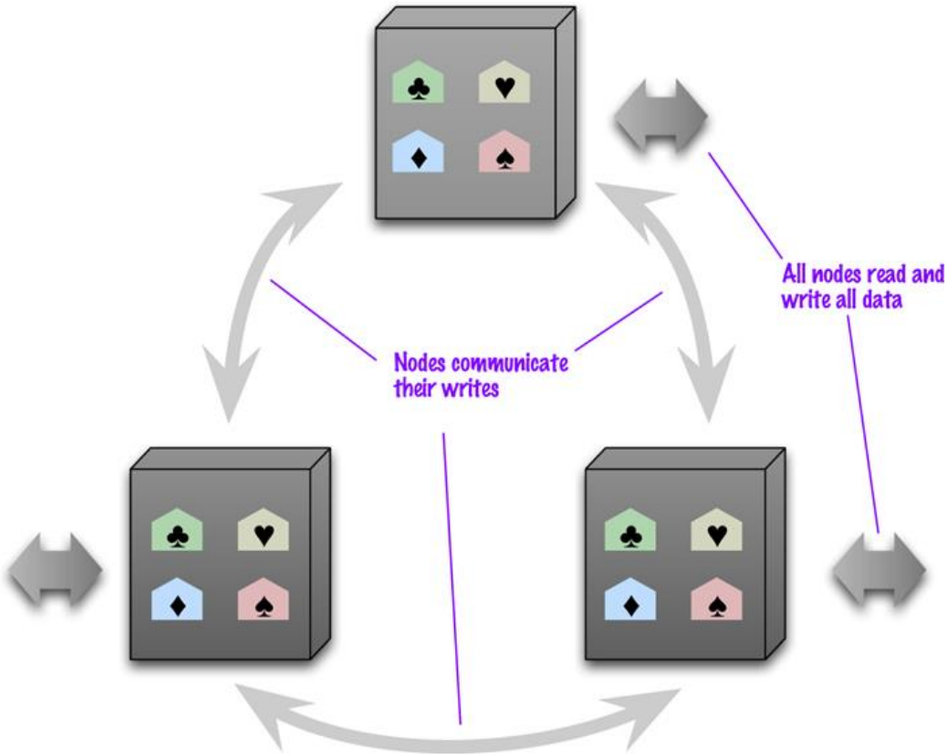
- With master-slave distribution, you replicate data across multiple nodes.



4-1 NoSQL 資料庫設計原則

對等式複製(Peer-to-peer Replication)

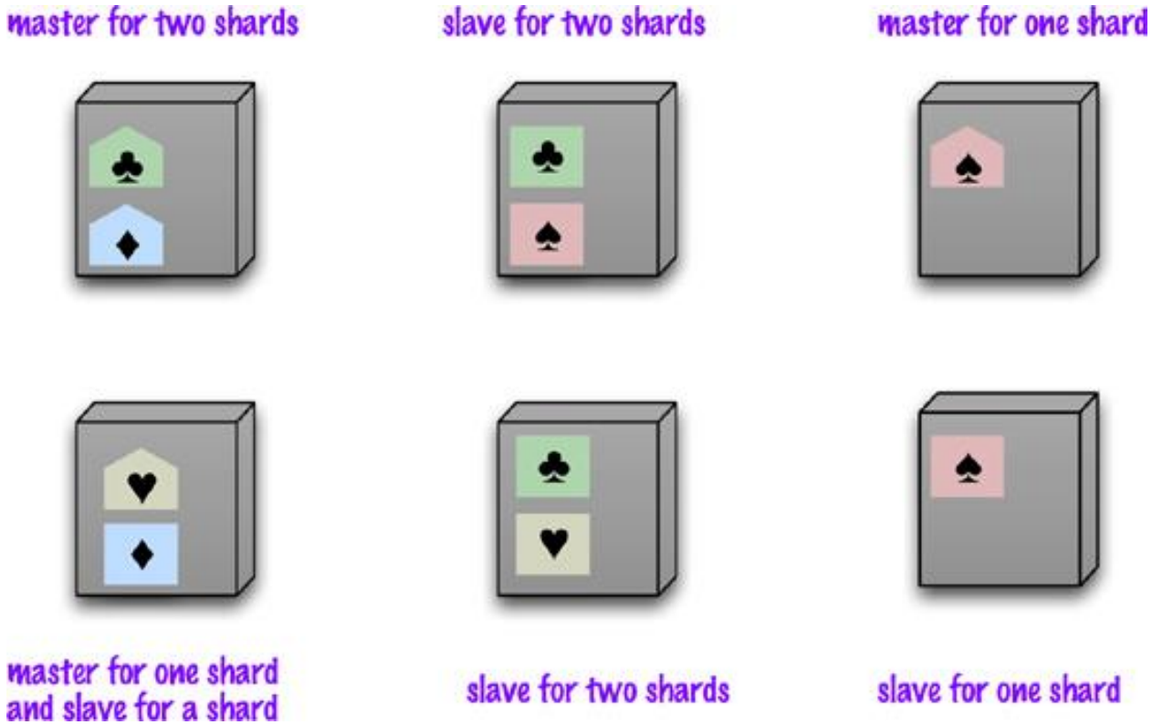
- Peer-to-peer replication has all nodes applying reads and writes to all the data.



4-1 NoSQL 資料庫設計原則

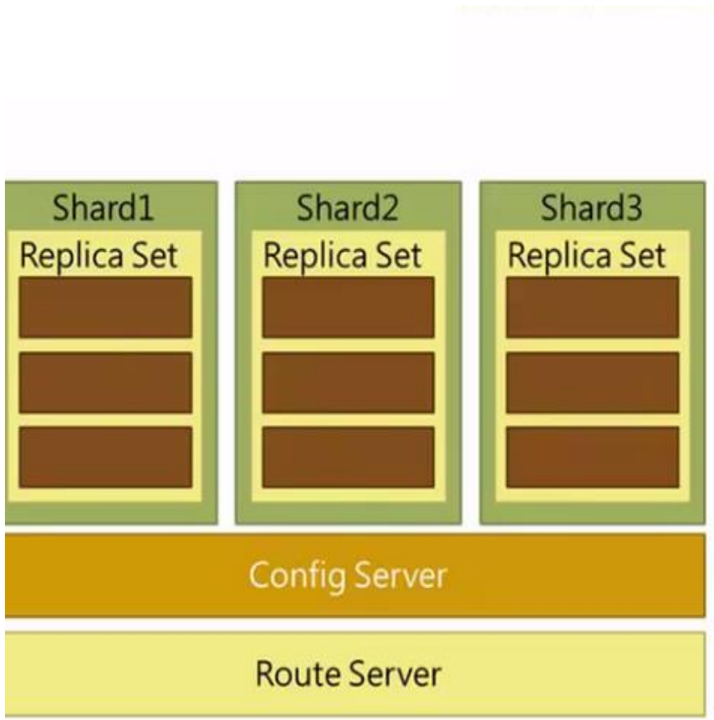
結合分片與主從式複製

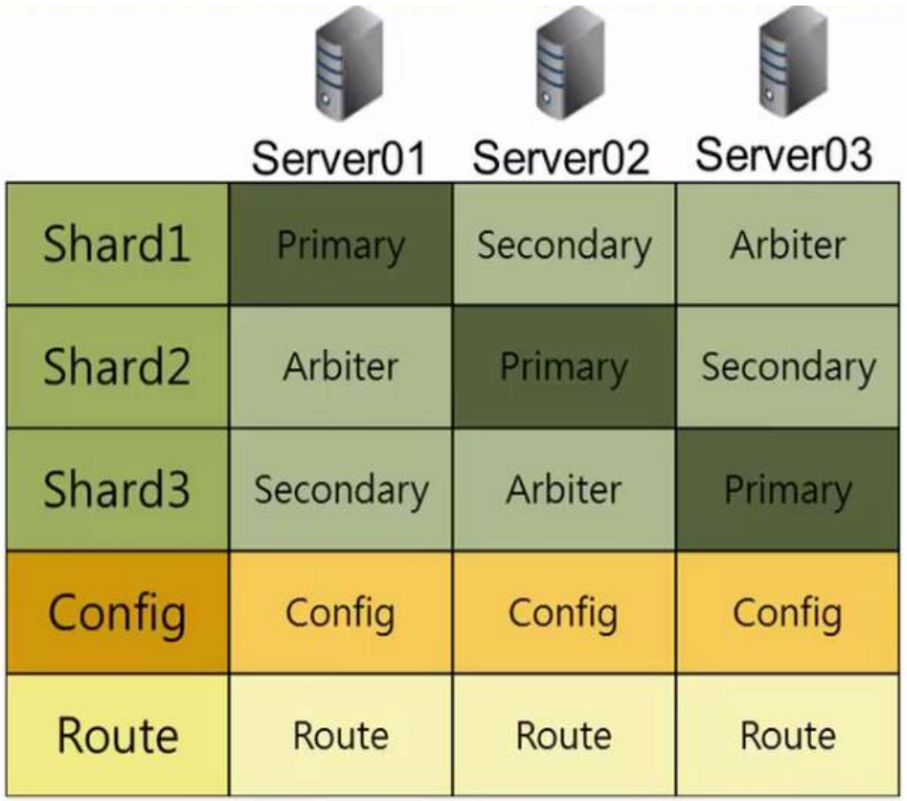
- Using master-slave replication together with sharding



4-1 NoSQL 資料庫設計原則

結合分片與主從式複製範例

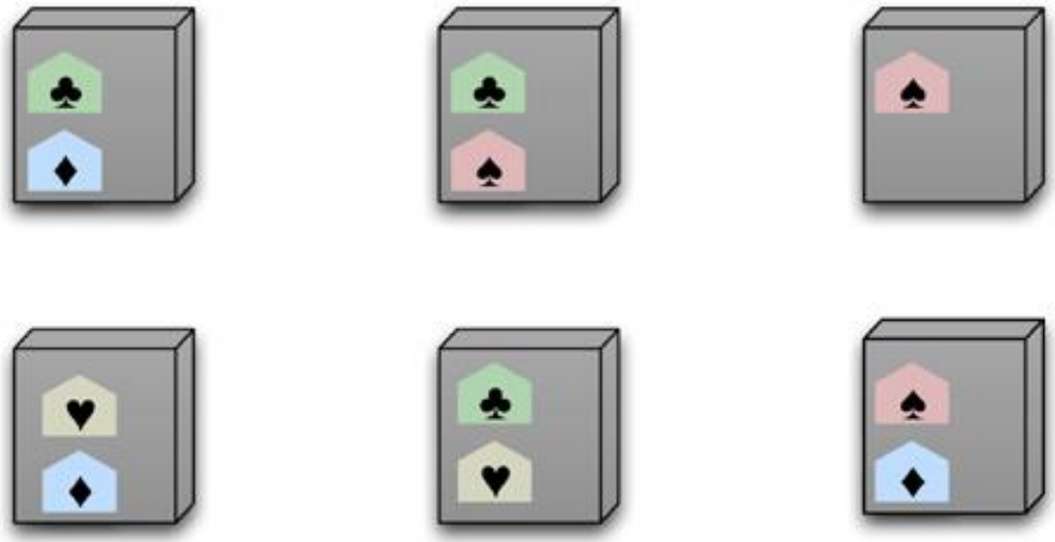




4-1 NoSQL 資料庫設計原則

結合分片與對等式複製

- Using peer-to-peer replication together with sharding



4-1 NoSQL 資料庫設計原則

一致性(Consistency)

- One of the biggest changes from a centralized relational database to a cluster-oriented NoSQL database is in how you think about consistency.
- Relational databases try to exhibit strong consistency by avoiding all the various inconsistencies.
- In NoSQL world, phrases such as “CAP theorem” and “eventual consistency” appear, and as soon as you start building something you have to think about what sort of consistency you need for your system.
- There are various shapes consistency can take.
  - Update Consistency
  - Read Consistency

4-1 NoSQL 資料庫設計原則

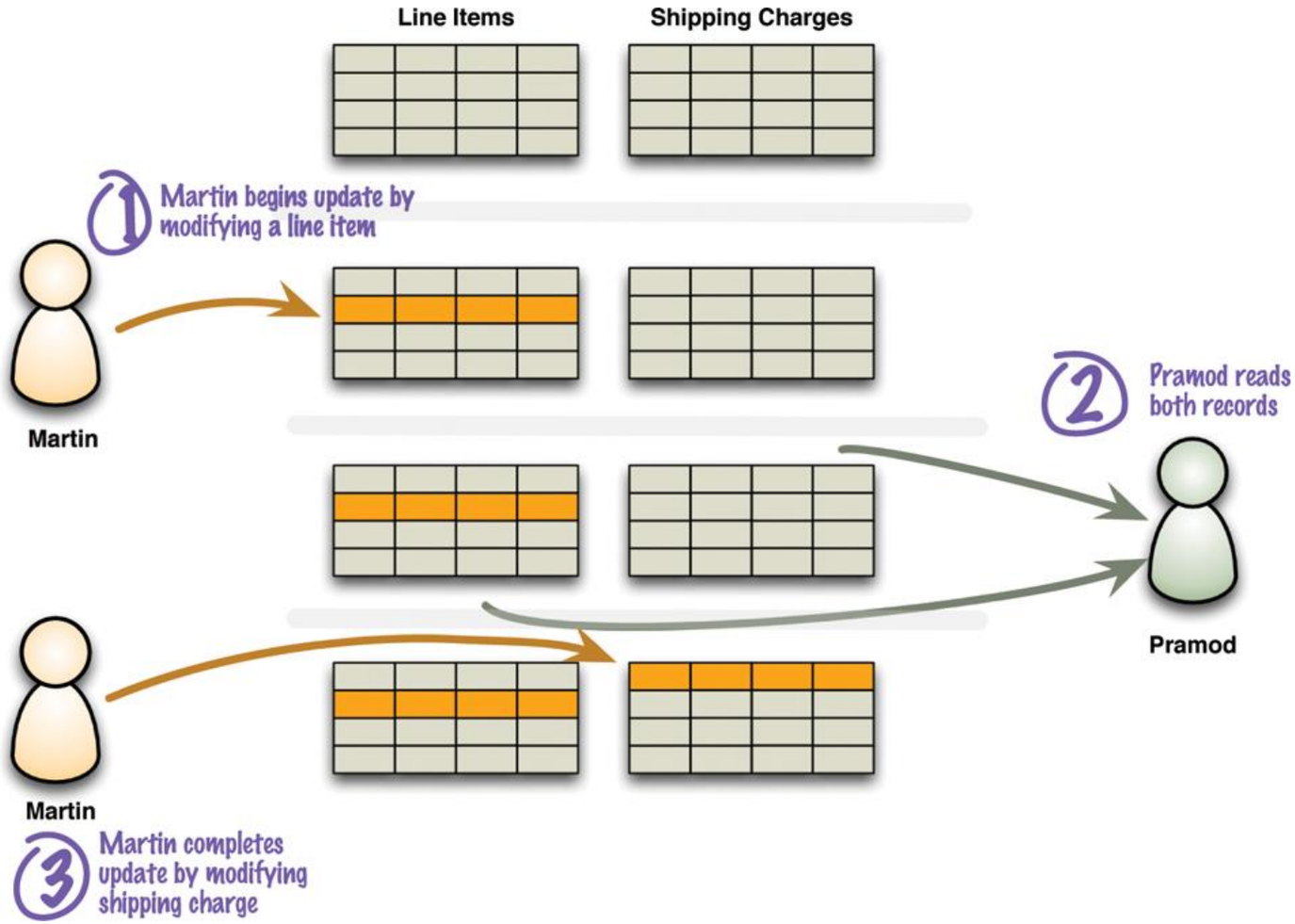
更新一致性(Update Consistency)

- write-write conflict: two people updating the same data item at the same time. (lost update problem)
- A pessimistic approach works by preventing conflicts from occurring; an optimistic approach lets conflicts occur, but detects them and takes action to sort them out.
- For update conflicts, the most common pessimistic approach is to have write locks.
- A common optimistic approach is a conditional update where any client that does an update tests the value just before updating it to see if it's changed since his last read.

4-1 NoSQL 資料庫設計原則

讀取一致性

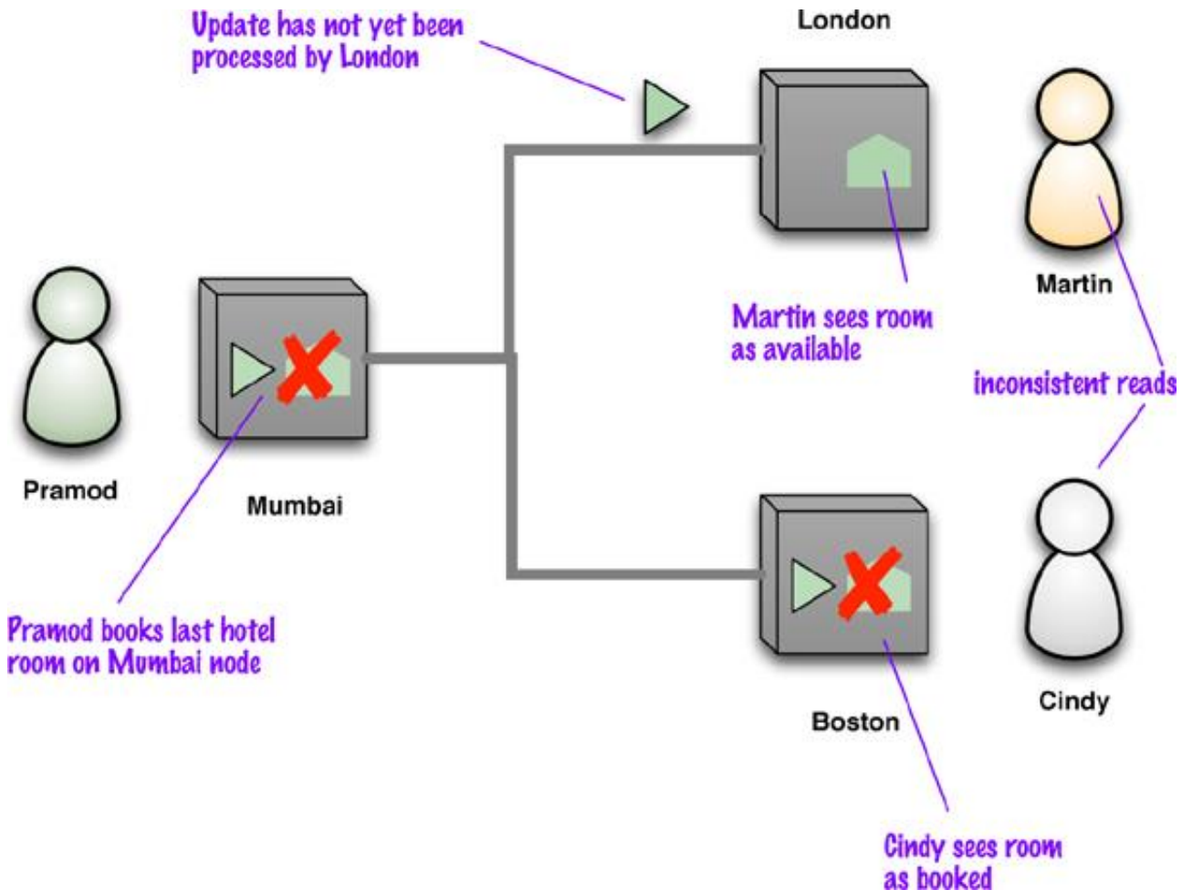
- inconsistent read
  or read-write conflict



4-1 NoSQL 資料庫設計原則

讀取一致性

- An example of replication inconsistency



4-1 NoSQL 資料庫設計原則

放寬一致性(Relaxing Consistency)

- Consistency is a Good Thing—but, sadly, sometimes we have to sacrifice it.
- We often have to tradeoff consistency for something else, we see it as part of the inevitable tradeoffs involved in system design.
- Trading off consistency is a familiar concept even in single-server relational database systems.
- Many systems forgo transactions entirely because the performance impact of transactions is too high.

4-1 NoSQL 資料庫設計原則

法定人數(Quorums)

- When you're trading off consistency or durability, it's not an all or nothing proposition. The more nodes you involve in a request, the higher is the chance of avoiding an inconsistency.
- replication factor (*N*) : the number of replica
- write quorum (*W*): the number of nodes participating in the write
- read quorum (*R)* : the number of nodes you need to contact for a read

  Strict consistency $R + W > N$

  Eventually consistency $R + W <= N$

4-2 NoSQL 資料庫設計步驟

NoSQL 設計概念

- NoSQL無法替代SQL，對於非Big Data，毫無疑問SQL更加好用，我們不應該執著的想著用NoSQL去替代SQL, 而是僅僅將SQL無法處理那部分資料(往往關係性不強)放到NoSQL上。
- NoSQL的設計核心為DDI，(Denormalization、 Duplication and Intelligent Keys)
- 在NoSQL設計中的金科玉律是: *“Design for the questions, not the answers”*。
- NoSQL是一種妥協，任意NoSQL的設計只是正對當前現況的優化, 只是為了回答這個question。

4-2 NoSQL 資料庫設計步驟

NoSQL 設計概念

- NoSQL data modeling often starts from the application-specific queries as opposed to relational modeling:
  - Relational modeling is typically driven by the structure of available data. The main design theme is "What answers do I have?"
  - NoSQL data modeling is typically driven by application-specific access patterns, i.e. the types of queries to be supported. The main design theme is "What questions do I have?"
- NoSQL data modeling often requires a deeper understanding of data structures and algorithms than relational database modeling does.

4-2 NoSQL 資料庫設計步驟

DDI 原則

- Denormalization

  反正規化的目的是在讀取資料時，減少Join的動作以提高效能，寫入資料時可以一次完成(Atomic)。

- Duplication

  資料重複儲存是在反正規化時常會採取的手段，重複儲存資料可以加快存取速度。

- Intelligent Keys

  適當的設計鍵值以利排序、索引等，解此達到讀取資料最佳化的效果。

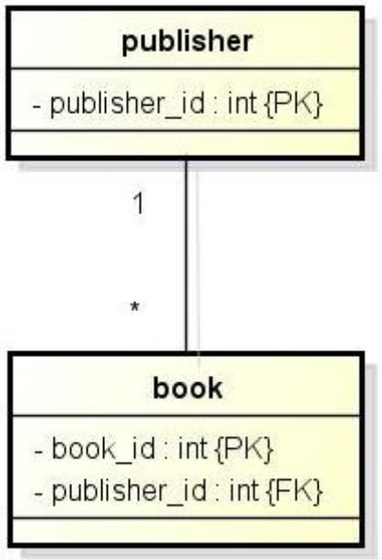
4-2 NoSQL 資料庫設計步驟

設計步驟

1. 需求的搜集與分析
2. 找出存取樣式(Access Patterns)
   - 如何存取資料，包括 Read patterns及Write patterns
3. 概想資料庫設計
   - 產出ER model
4. 資料模型轉換
   - 聚集(Aggrigating)? 參考(Referencing)?
5. 實體資料庫設計

4-3 聚集與參考

聚集與參考



publisher 嵌入 book

```
{
   title: "MongoDB: The Definitive Guide",
   author: [ "Kristina Chodorow", "Mike Dirolf" ],
   published_date: ISODate("2010-09-24"),
   pages: 216,
   language: "English",
   publisher: {
              name: "O'Reilly Media",
              founded: 1980,
              location: "CA"
            }
}

{
   title: "50 Tips and Tricks for MongoDB Developer",
   author: "Kristina Chodorow",
   published_date: ISODate("2011-05-06"),
   pages: 68,
   language: "English",
   publisher: {
              name: "O'Reilly Media",
              founded: 1980,
              location: "CA"
            }
}
```

4-3 聚集與參考

聚集與參考

publisher 參考 book

```
{
    name: "O'Reilly Media",
    founded: 1980,
    location: "CA",
    books: [12346789, 234567890, ...]
}

{
     _id: 123456789,
     title: "MongoDB: The Definitive Guide",
     author: [ "Kristina Chodorow", "Mike Dirolf" ],
     published_date: ISODate("2010-09-24"),
     pages: 216,
     language: "English"
}

{
    _id: 234567890,
    title: "50 Tips and Tricks for MongoDB Developer",
    author: "Kristina Chodorow",
    published_date: ISODate("2011-05-06"),
    pages: 68,
    language: "English"
}
```

book 參考 publisher

```
{
    _id: "oreilly",
    name: "O'Reilly Media",
    founded: 1980,
    location: "CA"
}

{
    _id: 123456789,
    title: "MongoDB: The Definitive Guide",
    author: [ "Kristina Chodorow", "Mike Dirolf" ],
    published_date: ISODate("2010-09-24"),
    pages: 216,
    language: "English",
    publisher_id: "oreilly"
}

{
    _id: 234567890,
    title: "50 Tips and Tricks for MongoDB Developer",
    author: "Kristina Chodorow",
    published_date: ISODate("2011-05-06"),
    pages: 68,
    language: "English",
    publisher_id: "oreilly"
}
```

模組5：MongoDB資料庫

5-1 MongoDB特性與架構
5-2 MongoDB塑模原理
5-3 MongoDB塑模範例

5-1 MongoDB特性與架構

MongoDB 資料庫

- MongoDB 是由C++語言編寫的，是一個基於分散式文檔存儲式的開來源資料庫系統。
- 在高負載的情況下，添加更多的節點，可以保證伺服器性能。
- MongoDB 旨在為WEB應用提供可擴展的高性能資料存儲解決方案。
- MongoDB 將資料存儲為一個文檔，資料結構由鍵值(key=>value)對組成，MongoDB 文檔類似於 JSON 物件，欄位值可以包含其他文檔，陣列及文檔陣列。

5-1 MongoDB特性與架構

Document Store 排名

☐ include secondary database models

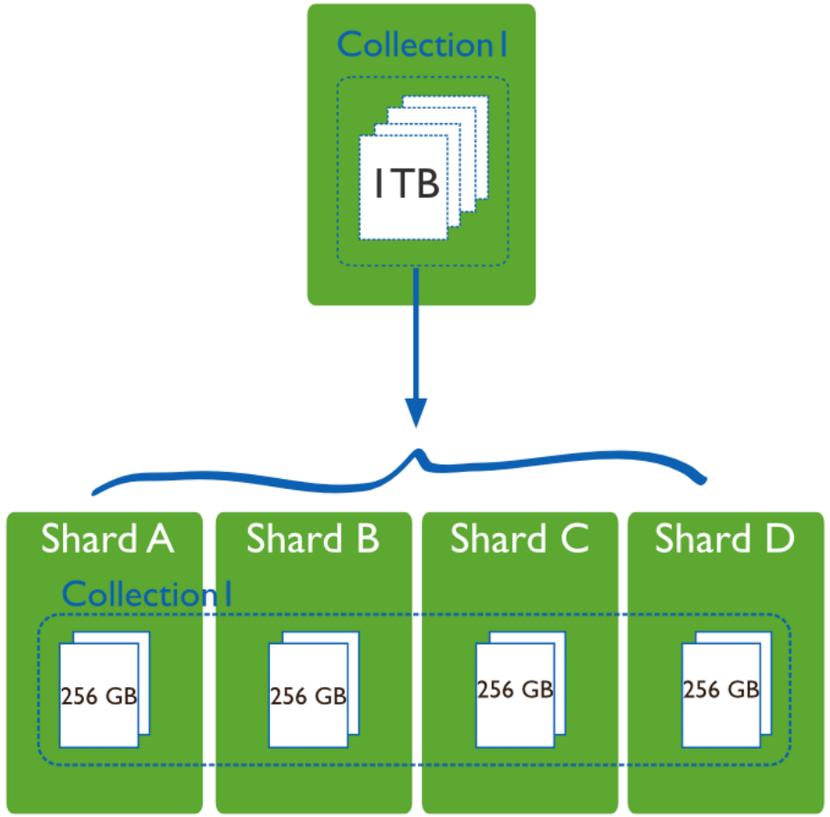
50 systems in ranking, November 2020

Rank			DBMS	Database Model	Score		
Nov 2020	Oct 2020	Nov 2019			Nov 2020	Oct 2020	Nov 2019
1.	1.	1.	MongoDB	Document, Multi-model	453.83	+5.81	+40.64
2.	2.	2.	Amazon DynamoDB	Multi-model	68.89	+0.48	+7.52
3.	3.	↑ 4.	Microsoft Azure Cosmos DB	Multi-model	32.50	+0.49	+0.52
4.	4.	↓ 3.	Couchbase	Document, Multi-model	30.55	+0.22	-1.44
5.	5.	5.	CouchDB	Document	17.25	-0.16	-1.14
6.	6.	↑ 7.	Firebase Realtime Database	Document	16.85	+0.59	+5.26
7.	7.	↓ 6.	MarkLogic	Multi-model	11.10	-0.63	-1.72
8.	8.	8.	Realm	Document	8.95	+0.20	+0.97
9.	9.	9.	Google Cloud Firestore	Document	8.45	-0.16	+2.65
10.	10.	↑ 11.	Google Cloud Datastore	Document	5.51	-0.42	+0.30

5-1 MongoDB特性與架構

MongoDB 架構

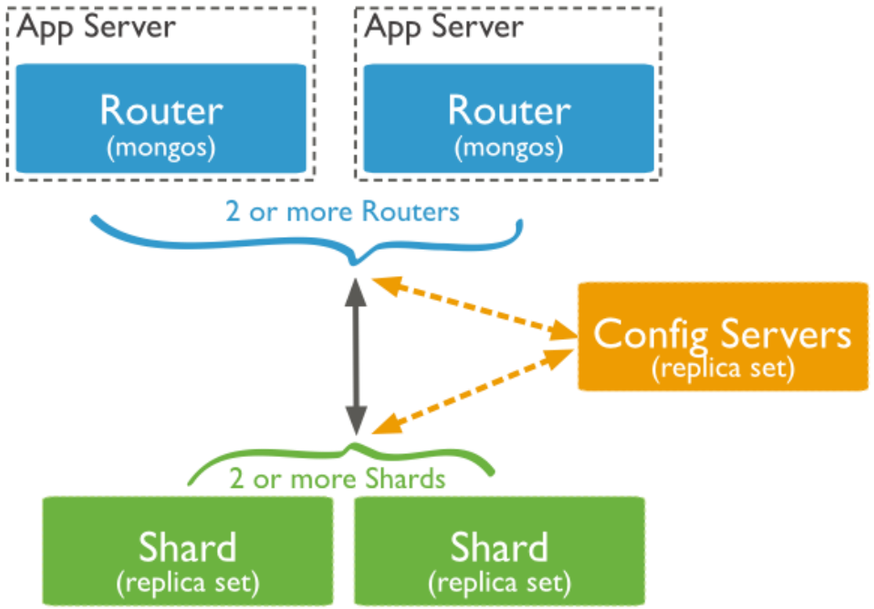
Sharding, or horizontal scaling, divides the data set and distributes the data over multiple servers, or shards.



5-1 MongoDB特性與架構

MongoDB 架構

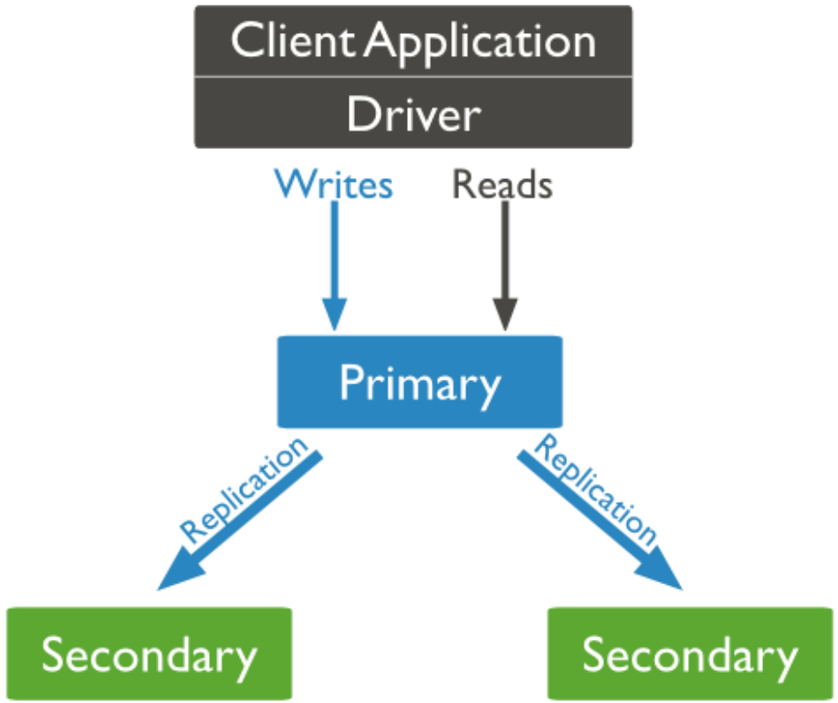
- Shards store the data. Each shard is a replica set.
- Query Routers interface with client applications and direct operations to the appropriate shard or shards.
- Config servers store the cluster's metadata.



5-1 MongoDB特性與架構

MongoDB 架構

- Primary receives all write operations.
- Secondaries replicate operations from the primary to maintain an identical data set.
- Arbiters select a primary if the current primary is unavailable.



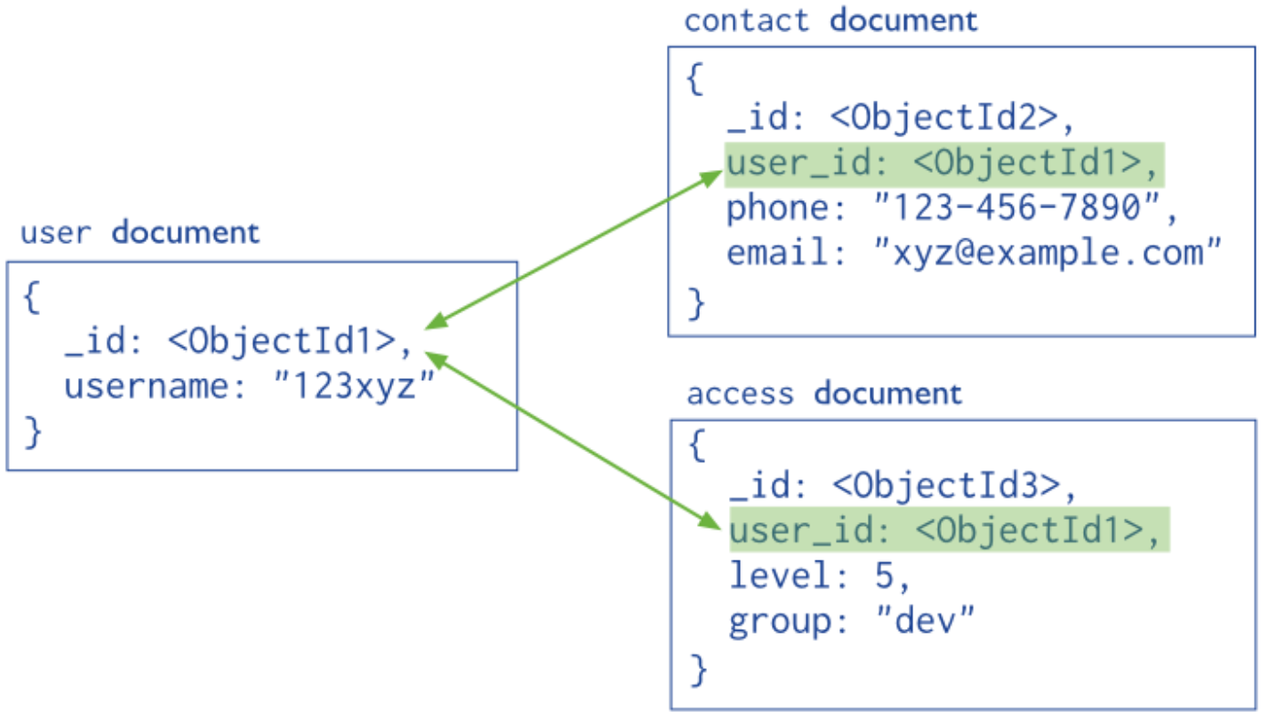
5-2 MongoDB塑模原理

MongoDB資料塑模

- MongoDB資料塑模程序，於建立概想模型時，和RDB是相似的。兩者主要的差別在於將概想模型轉換成實體模型的方法。
- MongoDB中要塑模關係(relationships)時，有兩種方式 嵌入(embedding) 資料或是參考(referencing) documents.
- 嵌入(Embedding) 是將兩個以上的實體利用階層將其合併成一個實體。
- 參考(Referencing) 則是建立類似RDBMS 中的外鍵(foreign key)，用以由一個實體(即collection) 指向另一個實體 (即collection)。

5-2 MongoDB塑模原理

參考(Referencing)



5-2 MongoDB塑模原理

嵌入(Embedding)

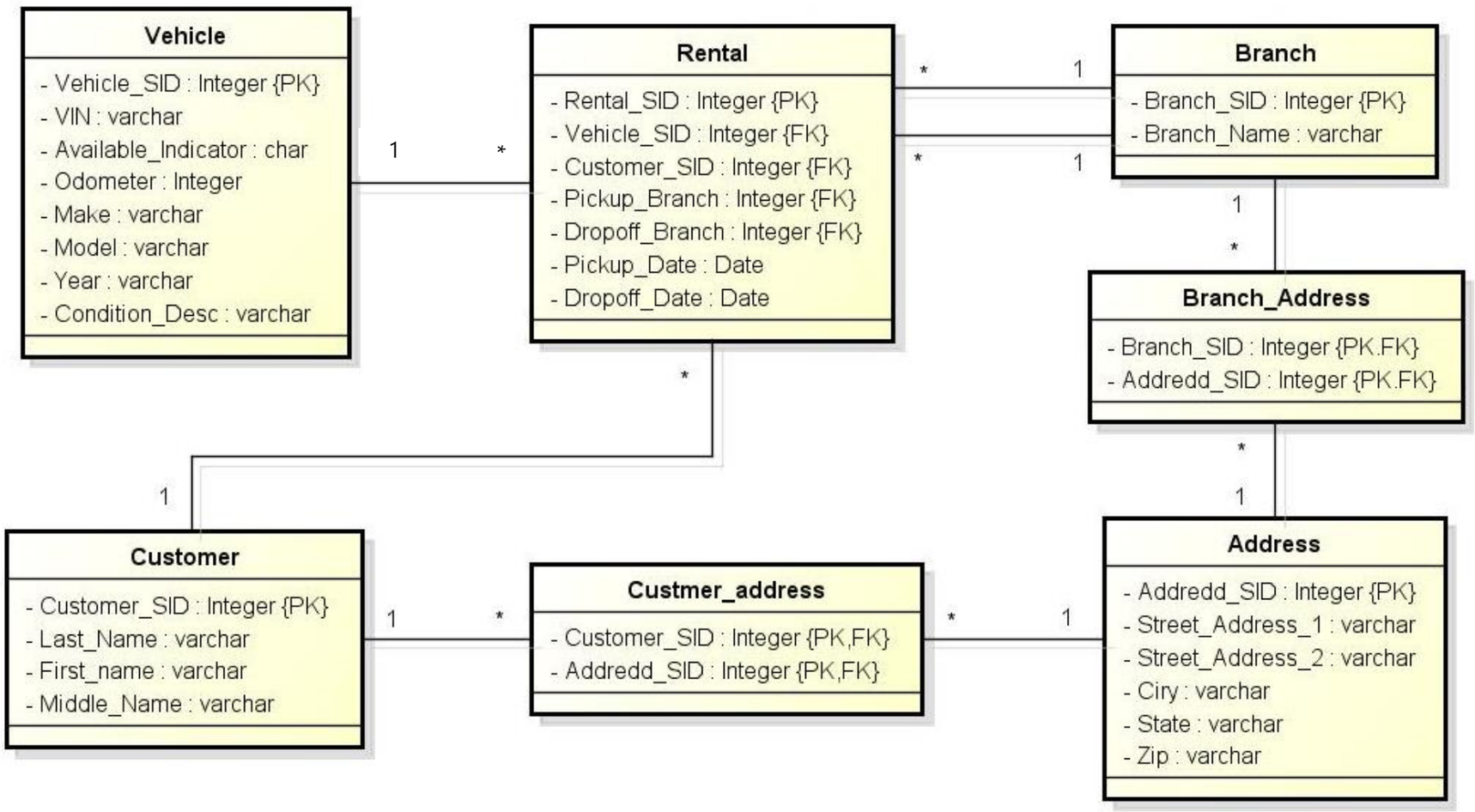
5-2 MongoDB塑模原理

五個 Heuristics

1. 若有些來自於不同實體的資料，常常被同時存取，則包含這些資料的實體可嵌入成同一文件(document)。
2. 若一個實體是依附(dependent)另一個實體，則將其嵌入至所依附之實體中。
3. 若兩個實體間存在一對一的關係，則將一個實體嵌入另一個實體中。
4. 若多個實體有共同頻率的volatility (inserts, updates and deletes) 則可以彼此嵌入在一起。
5. 若有些實體並非關鍵實體，但和關鍵實體間有關係，則應使用”參考”而非“嵌入”的方法。

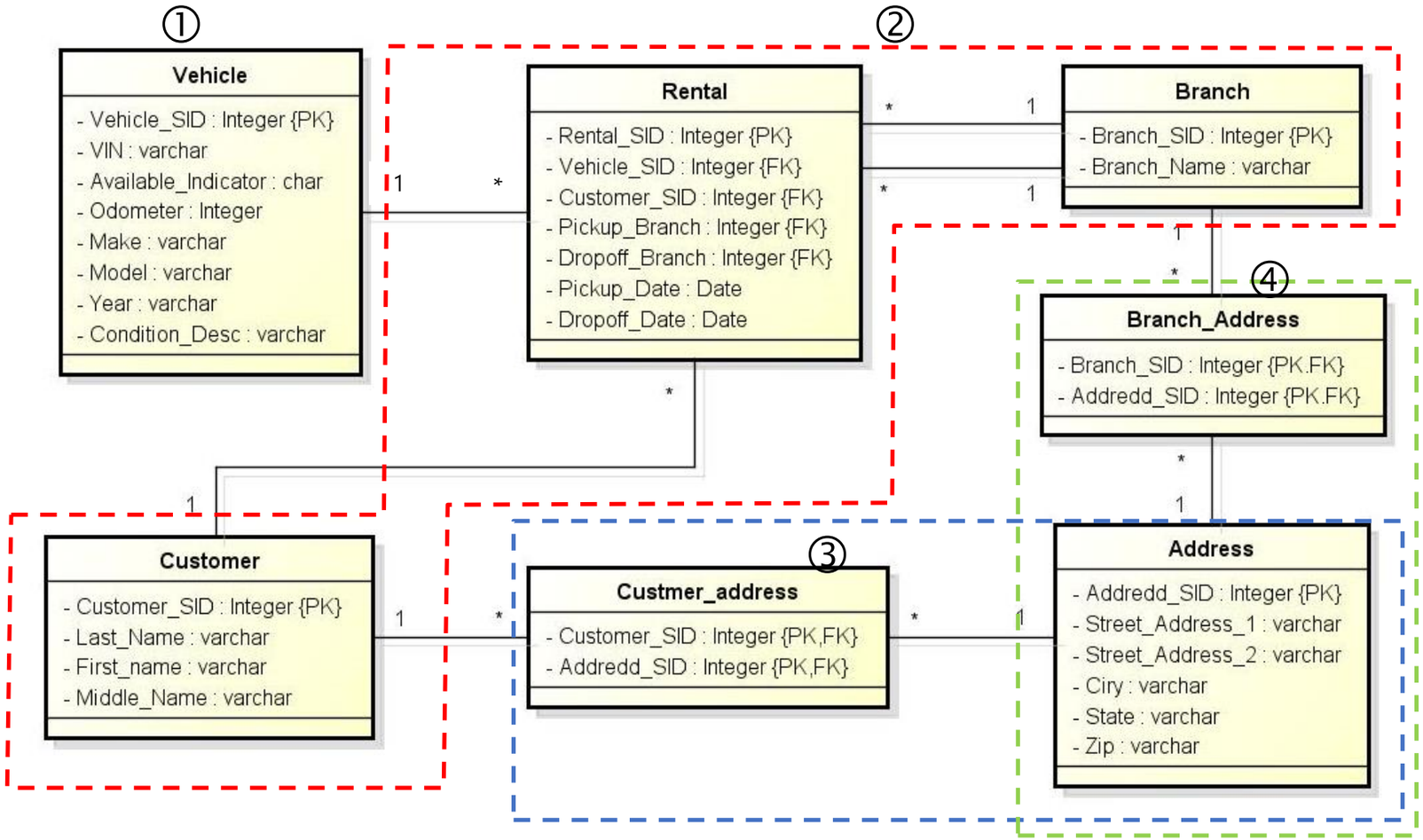
5-3 MongoDB塑模範例

範例



5-3 MongoDB塑模範例

範例 – Grouping



5-3 MongoDB塑模範例

範例 – PDM

1.Vehicle:
{Vehicle_SID, VIN, Available_Indicator, Odometer, Make, Model, Year,
Condtion_Desc}

2. Customer Rental:
{Customer_SID, LastName, First_Name, Middle_Name,
Rental: [{Rental_SID, Pickup_Date, P_Brach_ID, P_Branch_Name,
Drop_Date, D_Branch_SID, D_Branch_Name , Vehicle_SID}]
}

3. Customer Address
{Customer_SID, Cust_Addr: [{Street_Address_1, Street_Address_2, City,
State, Zip}]
}

4. Branch Address
{Branch_SID, Brch_Addr: [{Street_Address_1, Street_Address_2, City,
State, Zip}]
}

5-3 MongoDB塑模範例

範例 – Vehicle

Vehicle:

{ Vehicle_SID: “2134567”,

VIN: “1234ZC33456XYZ2”,

Available_Indicator : “Y”,

Odometer : “3507”,

Make : “Toyota”,

Model : “Camry”,

Year : “2014”,

Condition_Desc : “Scratches to paint on the left back door and scratches on the rear bumper”

}

5-3 MongoDB塑模範例

範例 – Customer Rental

Customer Rental:

{CustomerSID: “987765”,
LastName: “Smith”,
FirstName: “John”,
MiddleName: “Ricardo”,
Rentals: [
{ Rental_SID: “2300453”,
Pickup_Date: ISODate(“2014-07-13”),
Branch_SID: “3374”,
Branch_Name: “Aurora”
Dropoff_Date: ISODate(“2014-07-19”),
Branch_SID: “3370”,
Branch_Name: “Broadway”,
Vehicle_SID: “2134567”},

{ Rental_SID: “2307111”,
Pickup_Date: ISODate(“2014-08-25”),
Branch_SID: “3374”,
Branch_Name: “Aurora”
Dropoff_Date: ISODate(“2014-09-02”),
Branch_SID: “3374”,
Branch_Name: “Aurora”,
Vehicle_SID: “2134977”},
] }

模組6：Redis資料庫

6-1 Redis特性與架構
6-2 Redis資料型別
6-3 Redis使用與範例

6-1 Redis特性與架構

Redis 特性

Redis is an open source, advanced key-value store and an apt solution for building high performance, scalable web applications.

Redis has three main peculiarities that sets it apart.

- Redis holds its database entirely in the memory, using the disk only for persistence.
- Redis has a relatively rich set of data types when compared to many key-value data stores.
- Redis can replicate data to any number of slaves.

6-1 Redis特性與架構

Key-value Store 排名

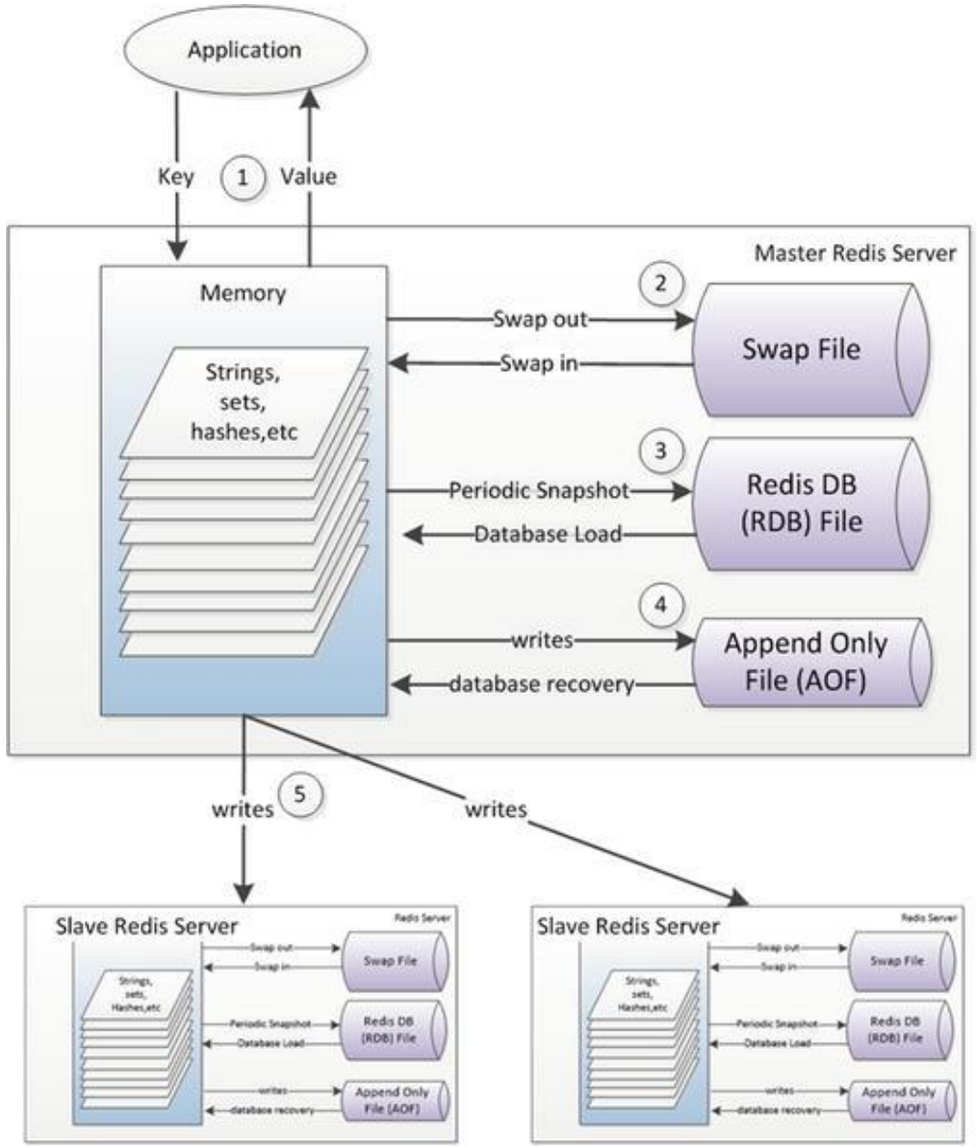
☐ include secondary database models

64 systems in ranking, November 2020

Rank Nov 2020	Rank Oct 2020	Rank Nov 2019	DBMS	Database Model	Score Nov 2020	Score Oct 2020	Score Nov 2019
1.	1.	1.	Redis	Key-value, Multi-model	155.42	+2.14	+10.18
2.	2.	2.	Amazon DynamoDB	Multi-model	68.89	+0.48	+7.52
3.	3.	3.	Microsoft Azure Cosmos DB	Multi-model	32.50	+0.49	+0.52
4.	4.	4.	Memcached	Key-value	25.75	-0.35	+0.63
5.	5.	5.	Hazelcast	Key-value, Multi-model	9.77	+0.05	+1.95
6.	6.	6.	etcd	Key-value	8.82	+0.01	+1.67
7.	7.	↑ 8.	Ehcache	Key-value	7.61	+0.14	+1.45
8.	8.	↓ 7.	Aerospike	Key-value, Multi-model	6.32	-0.68	+0.13
9.	9.	9.	Riak KV	Key-value	5.38	-0.36	-0.28
10.	10.	↑ 11.	ArangoDB	Multi-model	5.37	-0.18	+0.36

6-1 Redis特性與架構

Redis 架構



6-1 Redis特性與架構

Redis 架構說明

① Redis through primary key lookups that return “values”—strings, sets of strings, hashes of strings, and so on.

② The key values will almost always be in memory, though it is possible to configure Redis with a virtual memory system, in which case key values may have to be swapped in or out.

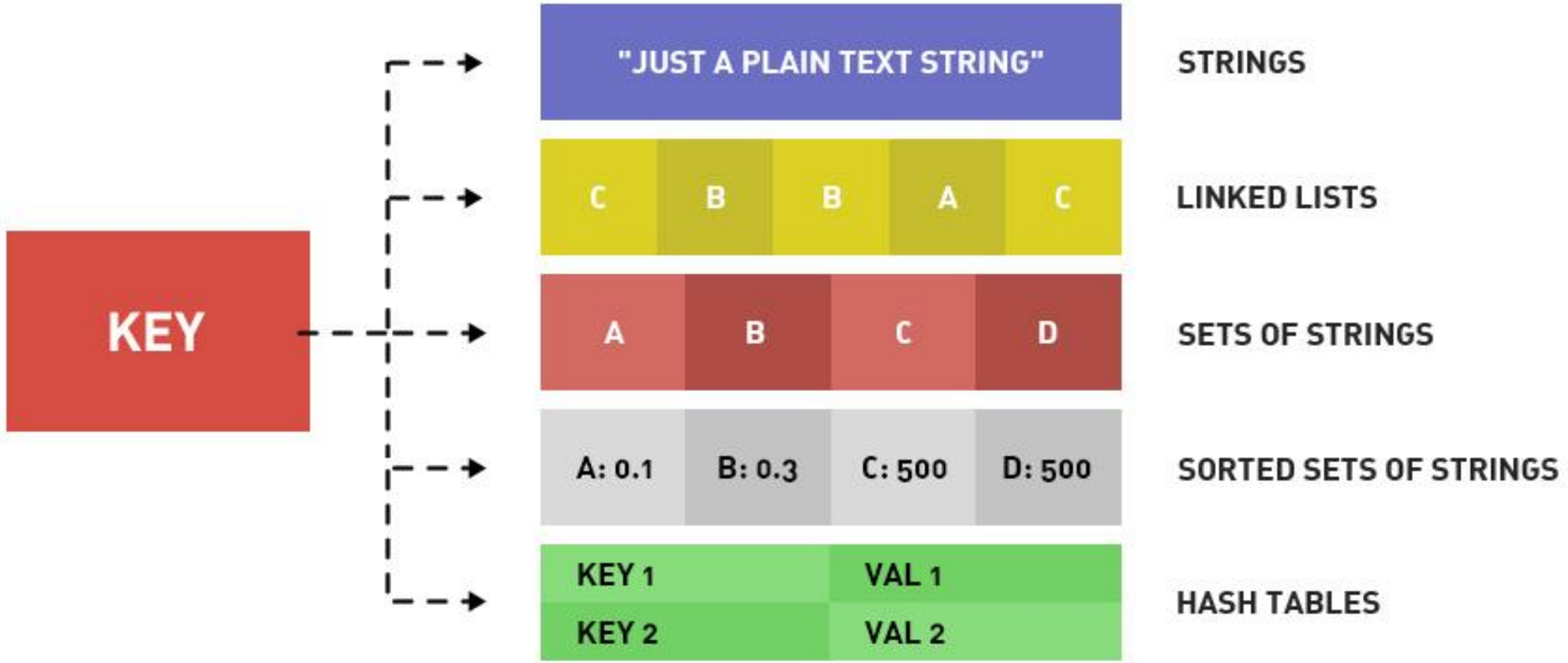
③ Periodically, Redis may dump a copy of the entire memory space to disk.

④ Redis can be configured to write changes to an append-only journal file either at short intervals or after every operation.

⑤ Redis may replicate the state of the master database asynchronously to slave Redis servers.

6-2 Redis資料型別

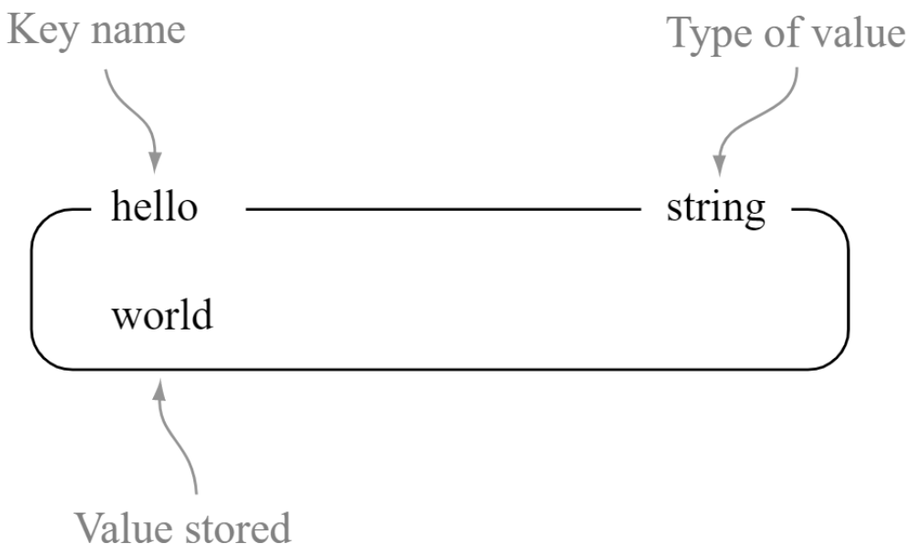
Redis 資料型別



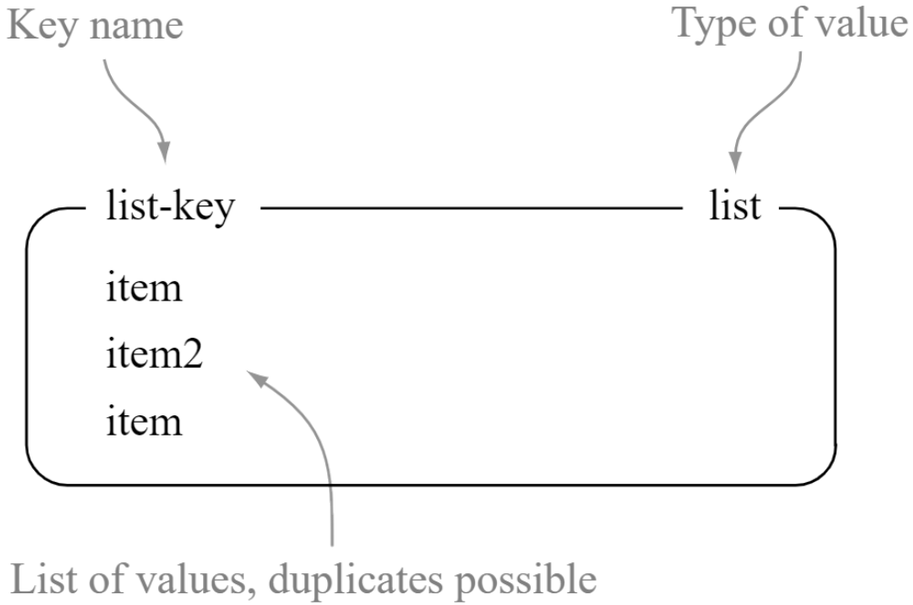
6-2 Redis資料型別

Redis 資料型別

- Strings



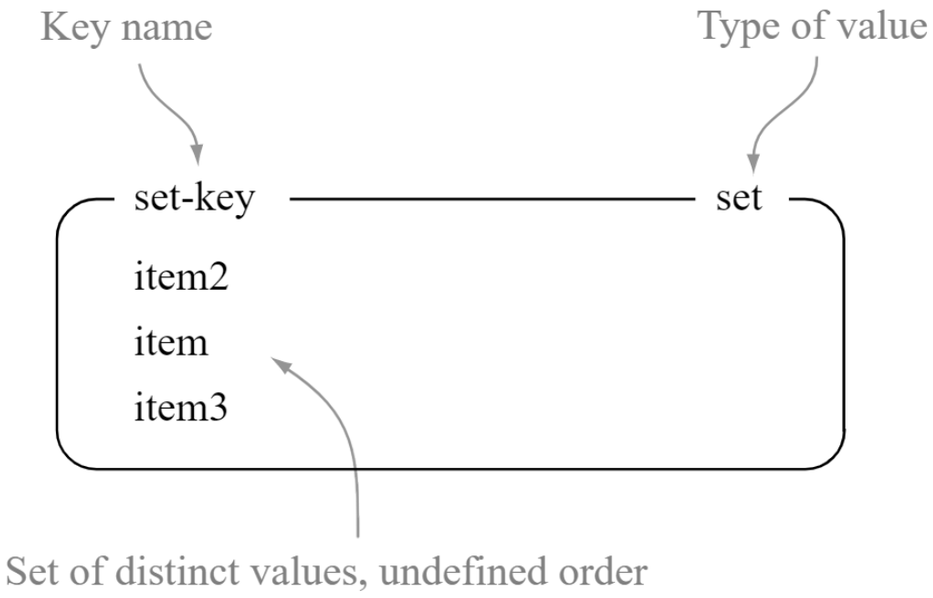
- Lists



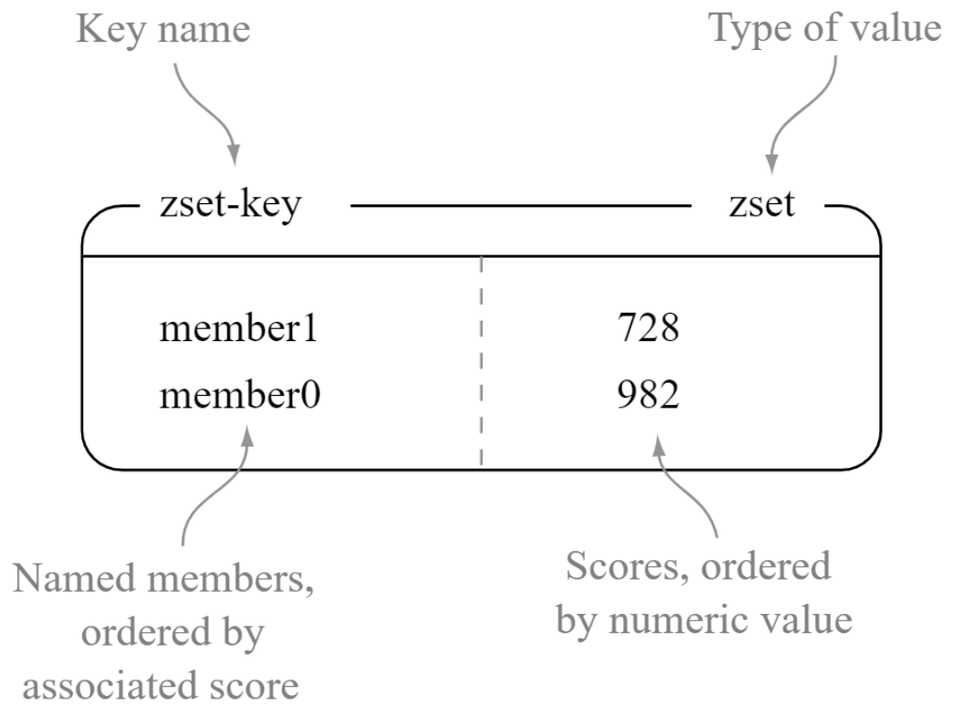
6-2 Redis資料型別

Redis 資料型別

- Sets



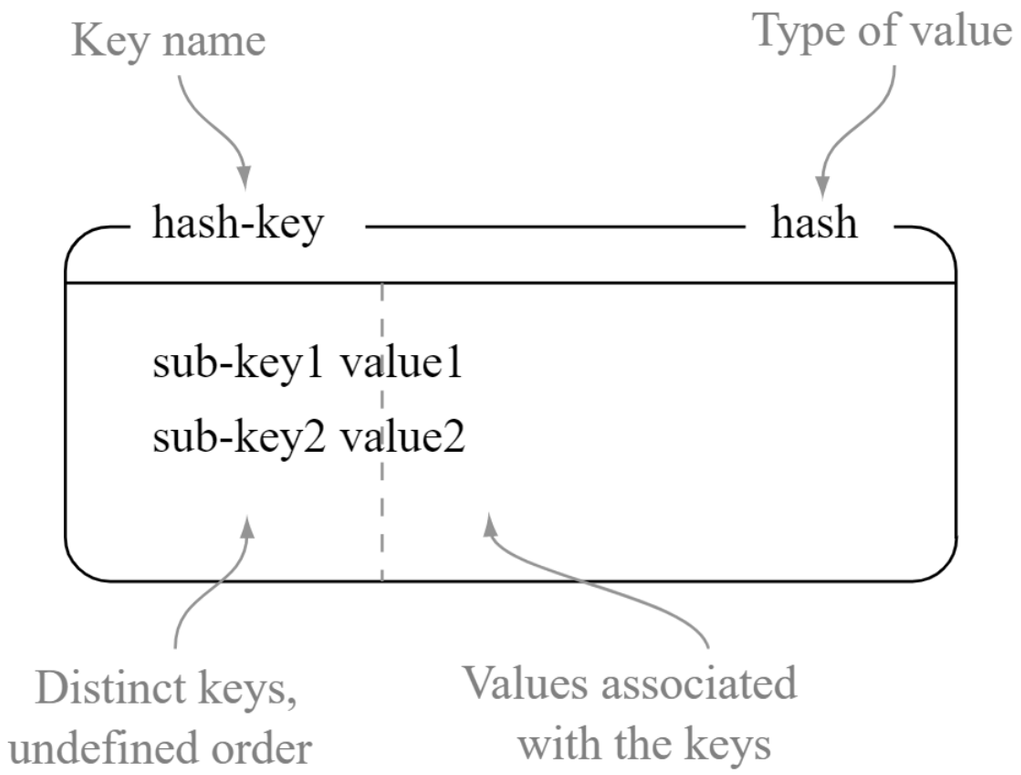
- Sorted Sets



6-2 Redis資料型別

Redis 資料型別

- Hashes



6-3 Redis使用與範例

Redis資料型別之應用－String

- Redis String是最常用的一種數據類型，普通的key/value存儲都可以歸為此類，value其實不僅是String，也可以是數字。

  例:想知道什麼時候封鎖一個IP位址(訪問超過幾次)。INCRBY命令讓這些變得很容易，通過atomic遞增保持計數。

  例:可用於計數器，比如統計在線人数，文章收藏数等。

6-3 Redis使用與範例

Redis資料型別之應用

- Redis List可以將程式中任何陣列直接儲存置資料中。
- Redis Hash 是一組String類型的field和value的映射表，特別適用於儲存有結構的物件。
- Redis Set 對外提供的功能與 List 類似是一個列表的功能，特殊之處在於 Set 是可以自動排除重複的，當你需要存儲一個列表數據，又不希望出現重複數據時，Set 是一個很好的選擇，並且 Set 提供了判斷某個成員是否在一個 Set集合內的指令，這個也是 List 所不能提供的。
- Redis Sorted Set 的使用場景與 Set 類似，區別是 Set 不是自動有序的，而 Sorted Set 可以通過用戶額外提供一個優先級(score)的參數來為成員排序，並且是插入有序的，即自動排序。