1. **簡介關聯式/非關聯式資料庫**
2. **關聯式資料庫（Relational Database Management System, RDBMS）**：

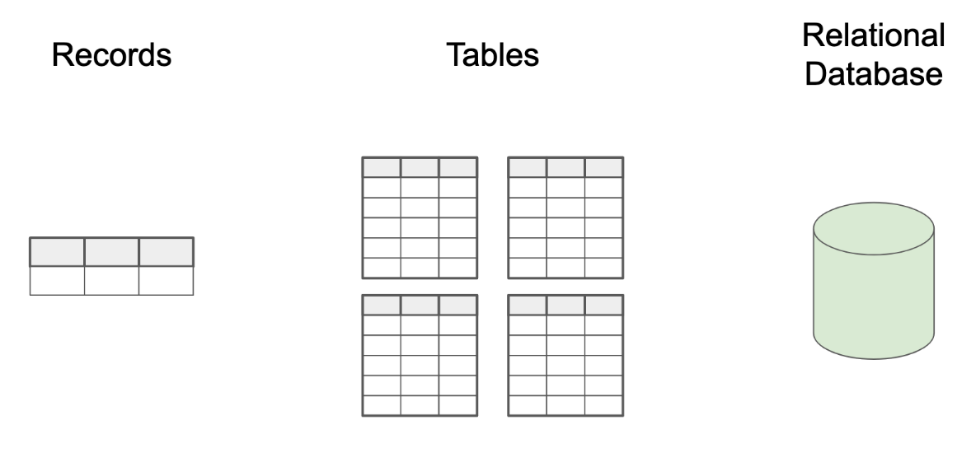
**起源：**1969年， Edgar F. Codd 第一次提出關聯式資料模型的概念。隔年在 ACM 上發表了 A Relational Model of Data for Large Shared Data Banks 論文才引起大家的注意，這份論文也是關聯式資料庫發展的起點。

**資料模型：**將資料透過行與列的二維表格呈現。不同於過往的階層與網狀資料庫使用低階指標連接資料，關聯式資料庫是使用「資料值」(Data Value)建立關聯。

**特點：**

**(1)資料是以一個或是多個資料表 (table) 的方式存放。**

在關聯式資料庫裡，每一筆資料都是在 table 中的一個 record，然後再把不同的 table 集合起來，成為一個關聯式資料庫。

****

**(2)資料之間有明確的關聯。**

關聯式資料庫一般都用來儲存結構化的資料，而資料之間大多會有清楚關聯。

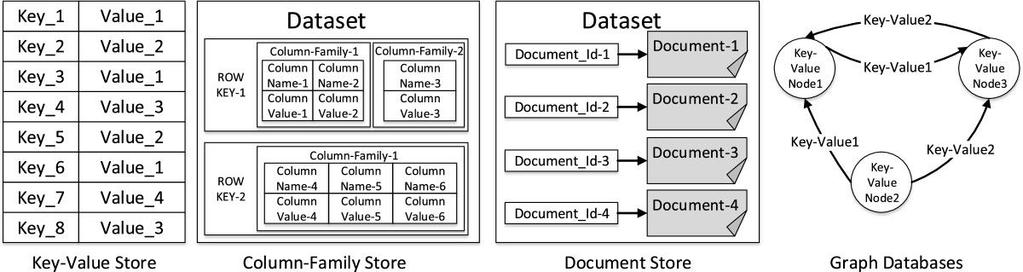
**(3)關聯式資料庫以 SQL 語言操作**

SQL (Structured Query Language 結構化查詢語言) 是一種專門用來管理與查詢關聯式資料庫的程式語言。透過 SQL，我們 能在關聯式資料庫裡新增、查詢、更新和刪除資料，同時也能建立和修改資料庫模式。

**2)非關聯式資料庫NoSQL，Not Only SQL**

**起源：**隨著硬體設備電腦、行動裝置日益精進、互聯網普及和網路應用程式的流量大幅成長，對於社群媒體，每分每秒需要處理的資料量可謂是爆炸增長。以使用者角度出發，從過去對資料庫要求「能夠無誤地同步處理結構性資料」，到現階段需求為「處理高速且大量產生的資料，但不需要即時同步，也不需絕對的零錯誤。」為了呼應新需求，NoSQL 資料庫隨之興起。

**資料模型：**將收集後的資料，作為儲存的最小單位，再將資料分散到多個節點；比起資料的關聯，NoSQL 更著重資料所代表的人（如使用者）與物（如一篇分享在社交平台上的文章）「狀態」變動，如文章被分享、按讚等。

****

**非關聯式資料庫的四大類型：**

**(1)鍵值資料庫：**最常見的 NoSQL 資料庫儲存類型，資料由key-value的方式儲存，鍵值資料庫具備高度可分割性。

**(2)Column資料庫：**Column資料庫將數據儲存在Column中，一個Column儲存成經常被一起查詢的相關數據。這種資料庫通常用來應對分散式儲存海量數據。

**(3)文件資料庫：**儲存、索引並管理以文件為主的資料或者類似的半結構化資料，像使用 JSON資料結構的應用。

**(4)圖形資料庫：**圖形結構。專門用於處理具有高度相互關聯關係的資料，比較適合社群網路、模式識別、推薦系統等。

**特點：**

**(1)可擴展性：**支援橫向擴展，不同於關聯式資料庫。

**(2)靈活的資料模型：**關聯式資料庫是由代數理論發展，因此關聯式資料庫模型有著嚴格的規範，如：在使用前須明確定義欄

位、資料型態等，且須遵從許多限制條件，無法動態更改架構，而非關聯式資料庫則沒有太多限制。

**(3)與雲端運算緊密融合：**可支援橫向擴展，可結合雲端運算底層基礎架構。

**2. 詳細介紹任一非關聯式資料庫**

**1) 起源：**

Apache Cassandra 基於 Amazon Dynamo 的分散式設計和 Google Big Table 的數據模型，由 Facebook 創建。

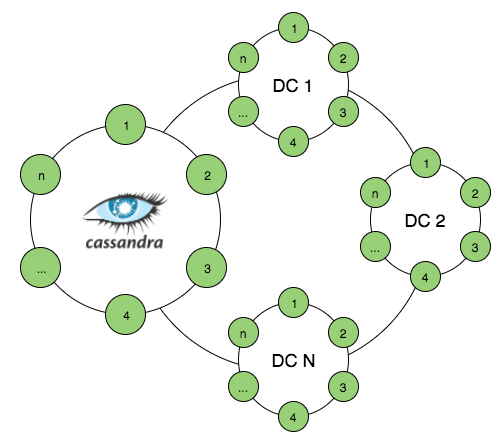
2007 年 Facebook 為了解決訊息收件箱問題（ Inbox Search problem）而開始設計 Cassandra 。當時 Facebook 遇到傳統方法難以解決的超大數據量儲存可擴展性問題。團隊需要處理大量的消息副本、消息的反向索引等不同形式的數據，需要處理很多隨機讀和隨機寫操作。

2008年 Facebook工程師Avinash Lakshman，Karthik Ranganathan 和Prashant Malik，發布Cassandra的代碼，當時僅作為Google Code的一個項目，還未形成社區。直至2010年，Lakshman 和 Malik 在 ACM 首次發表了 Cassandra 的論文 《Cassandra: a decentralized structured storage system》，向全世界介紹了 Cassandra。

**2) 名稱由來：**

在希臘神話裡，Cassandra 是特洛伊戰爭中的先知，商標有個先知眼。

1. **資料模型**



1. **特點**：
2. **分散性和去中心化（Distributed and Decentralized）**

主要特點是由一堆資料庫節點共同構成一個分散式網絡服務，例如一個寫操作，會被複製到其他節點上去，而對讀操作，也會被路由到某個節點上讀取。對於一個Cassandra群集來說，擴展性能是比較簡單的事情，只需在群集裡面添加節點就可以了。Cassandra 是分散式，這意味著它可以運行在多台機器上，並呈現給用戶一個一致的整體。Cassandra 的強大之處在於它的很多設計和實現讓系統不僅可以在多個節點上運行，更為多機部署進行了優化，甚至一個 Cassandra 集群可以運行在分散於世界各地的數據中心上。你可以放心地將數據寫到集群的任意一台機器上，Cassandra 都會收到數據。

對於很多儲存系統（比如 MySQL），一旦你開始擴展它，就需要把某些節點設為主節點，其他則作為從節點。但 Cassandra 是無中心的，也就是說每個節點都是一樣的。與主從結構相反，Cassandra 的協議是 P2P 的，並使用 gossip 來維護存活或死亡節點的列表。去中心化這一事實意味著 Cassandra 不會存在單點失效。Cassandra 集群中的所有節點的功能都完全一樣， 所以不存在一個特殊的主機作為主節點來承擔協調任務。有時這被叫做伺服器對稱（server symmetry）。

1. **Column結構、靈活的模式（Flexible Schema）**

Cassandra是一個混合型的非關聯資料庫，類似於Google的Big Table。主要功能比 Dynamo（分散式的Key-Value儲存系統）更豐富。Cassandra 常被視為一種Column資料庫，它的數據結構不是關聯式的， Cassandra 的數據儲存結構基本可以視為是一個多維稀疏哈希表。在關聯式資料庫中，得先設計一個完整的數據模型， 然後考慮查詢方式。而在 Cassandra 裡， 首先思考如何查詢數據，然後提供這些數據就可以了。

Cassandra 的早期版本支持無模式（schema-free）數據模型，可以動態定義新的列。無模式資料庫（如 Big Table 和 MongoDB）在訪問大量數據時具有高度可擴展性和高性能的優勢。無模式資料庫的主要缺點是難以確定數據的含義和格式，這限制了執行複雜查詢的能力。

為了解決這些問題，Cassandra 引入了 Cassandra Query Language（CQL），它提供了一種通過類似於結構化查詢語言（SQL）的語法來定義模式。CQL 集合（比如 list、set、尤其是 map）提供了在無結構化的格式裡面添加內容的能力，從而能擴展現有的模式。CQL 還提供了改變列的類型的能力，以支持 JSON 格式的文本的存儲。

1. **彈性可擴展（Elastic Scalability）**

可擴展性是指系統架構可以讓系統提供更多的服務而不降低使用性能的特性。僅僅通過給現有的機器增加硬體的容量、內存進行垂直擴展，是最簡單的達到可擴展性的手段。彈性可擴展是指水平擴展的特性，意味著集群可以不間斷的情況下，擴展或縮減服務的規模。這樣你就不需要重新啟動進程，不必修改應用的查詢，也無需自己手工重新均衡數據分布。在 Cassandra 裡，只要加入新的計算機，Cassandra 就會自動地發現它並讓它開始工作。

1. **高可用和容錯（High Availability and Fault Tolerance）**

從一般架構的角度來看，系統的可用性是由滿足請求的能力來量度的。但計算機可能會有各種各樣的故障，從硬體器件故障到

網絡中斷都有可能。所以它們一般都有硬體冗餘，並在發生故障事件的情況下會自動響應並進行熱切換。對一個需要高可用的

系統，它必須由多台聯網的計算機構成，並且運行於其上的軟體也必須能夠在集群條件下工作，有設備能夠識別節點故障，並

將發生故障的中端的功能在剩餘系統上進行恢復。Cassandra 就是高可用的。你可以在不中斷系統的情況下替換故障節點，

還可以把數據分布到多個數據中心裡，從而提供更好的本地訪問性能，並且在某一數據中心發生火災、洪水等不可抗災難的時

候防止系統徹底癱瘓。

1. **可調節的一致性（Tunable Consistency）**

CAP 定律表明，對於任意給定的系統，只能在一致性（Consistency）、可用性（Availability）以及分區容錯性（Partition Tolerance）之間選擇兩個。所以 Cassandra 在設計的時候也不得不考慮這些問題，因為分區容錯性這個是每個分散式系統必須考慮的，所以只能在一致性和可用性之間做選擇，而 Cassandra 的應用場景更多的是為了滿足可用性，所以只能犧牲一致性了。但是根據 BASE 理論，我們其實可以通過犧牲強一致性獲得可用性。Cassandra 提供了可調節的一致性，允許我們選定需要的一致性水平與可用性水平，在二者間找到平衡點。因為客戶端可以控制在更新到達多少個副本之前，必須阻塞系統。這是通過設置副本因子（replication factor）來調節與之相對的一致性級別。

通過副本因子（replication factor），你可以決定準備犧牲多少性能來換取一致性。副本因子是你要求更新在集群中傳播到的節點數。客戶端每次操作還必須設置一個一致性級別（consistency level）參數，這個參數決定了多少個副本寫入成功才可以認定寫操作是成功的，或者讀取過程中讀到多少個副本正確就可以認定是讀成功的。這裡 Cassandra 把決定一致性程度的權利留給了客戶自己。所以，如果需要的話，你可以設定一致性級別和副本因子相等，從而達到一個較高的一致性水平，不過這樣就必須付出同步阻塞操作的代價，只有所有節點都被更新完成才能成功返回一次更新。而實際上，Cassandra 一般都不會這麼來用，原因顯而易見，（這樣就喪失了可用性目標，影響性能，而且這不是你選擇 Cassandra 的初衷。）如果一個客戶端設置一致性級別低於副本因子的話，即使有節點當機了，仍然可以寫成功。總體來說，Cassandra可以通過調節一致性水平達到 AP。

1. **應用場景：**
2. **大規模部署**

Cassandra 的特點，高可用、可調一致性、Gossip協議、彈性擴展等，這些特性在單節點時是無法顯現的，但卻可以在數十個節點上發揮。考慮資料的讀寫比例，Cassandra 是為優異的寫吞吐量而特別優化的。許多早期使用 Cassandra 的產品都用於存儲用戶狀態更新、社交網絡、建議、評價以及應用統計等。這些都是 Cassandra 很好的應用場景，這些應用大都是寫多於讀的，並且更新可能隨時發生並伴有突發的峰值。事實上，支撐應用負載需要很高的多客戶線程並發寫性能，這正是Cassandra 的主要特性。

1. **地區分布**

Cassandra 直接支持多地分布的數據儲存，Cassandra 可以很容易配置成將數據分布到多個數據中心的儲存方式。如果有一個全球部署的應用，那麼讓數據貼近用戶會獲得不錯的性能收益，Cassandra 正適合這種應用場合。

1. **草創時期**

如果正在「初創階段」，業務會不斷改進，Cassandra 這種靈活的模式的數據模型可能較為適合。這讓資料庫能更快地跟上業務改進的步伐。

**3. 請比較關聯式/非關聯式資料庫優缺點**

(1) 關聯式資料庫

優點：以關聯代數理論作為基礎，有嚴格的標準，支援交易ACID四特性，借助索引機制可以實現高效的查詢，技術成熟，有專業公司的技術支援。

缺點：可擴展性比較差(尤其是橫向擴展性)，無法較好地支援海量資料儲存，資料模型過於死板、無法較好地支援WEB2.0應用，交易機制影響了系統的整體性能等

(2) 非關聯式資料庫

優點：可以支援超大規模資料儲存，靈活的資料模型可以很好地支援WEB2.0應用，具有強大的橫向擴展能力等。

缺點：缺乏數學理論基礎，複雜查詢性能不高，大都不能實現交易強一致性，很難實現資料完整性，企業關鍵任務不能採用。技術上不成熟，缺乏專業團隊的技術支援，維護較困難。

|  |  |  |
| --- | --- | --- |
| 比較標準 | RDBMS | NoSQL |
| 資料庫原理 | 完全支持 | 部分支援 |
| 有關聯代數理論作為基礎 | 沒有統一的理論基礎 |
| 資料規模 | 大 | 超大 |
| 很難實現橫向擴展，縱向擴展的空間也有限，性能會隨著規模的增大而降低 | 可以很容易透過添加更多設備來支援更大的資料 |
| 資料庫模式 | 固定 | 靈活 |
| 需定義資料庫模式，嚴格遵守資料定義和相關限制條件 | 不存在資料庫模式，可以自由靈活定義並儲存不同類型的資料 |
| 查詢效率 | 快 | 可以實現高效的簡單查詢，但是不具備高度結構化查詢特性，複雜查詢的性能不盡人意 |
| 借助於索引機制可以實現快速查詢(包括紀錄查詢和範圍查詢) | 沒有以複雜查詢為主的索引，雖可用MapReduce來加速查詢，但複雜查詢性能仍不如RDBMS |
| 一致性 | 強 | 弱 |
| 嚴格遵守ACID模型，可保證交易強一致性 | 很多放鬆對交易ACID四特性的要求，而遵守BASE模型，只能保證最終一致性 |
| 資料完整性 | 容易實現 | 很難實現 |
| 很容易實現完整性限制，比如透過主鍵或非空限制實現個體完整性，透過主鍵、外來鍵來實現參考完整性，透過限制或觸發器來實現用戶自訂完整性 | 無法實現 |
| 擴展性 | 一般 | 好 |
| 很難實現橫向擴展，縱向擴展的空間也有限 | 在設計之初就充分考慮橫向擴展的需求，可以很容易透過添加設備實現擴展 |
| 可用性 | 好 | 很好 |
| 以保證資料一致性為優先目標，其次才是最佳化系統性能。當資料規模增大，為保證嚴格的一致性，只能提供相對較弱的可用性 | 大多數都能提供較高的可用性 |
| 標準化 | 是 | 否 |
| 已經標準化(ANSI SQL) | 還沒有業界標準，不同NoSQL有自己的查詢語言，很難規範應用程式介面。Stone Braker認為：缺乏統一查詢語言，將會拖慢其發展 |
| 技術支援 | 高 | 低 |
| 經幾十年發展，已非常成熟，Oracle等大型廠商都可以提供很好的技術支援 | 在技術支持方面仍然處於起步階段，還不成熟，缺乏有力的技術支援 |
| 可維護性 | 複雜 | 複雜 |
| 需要專門的資料庫管理員維護 | 資料庫雖然沒有DBMS複雜，卻也難以維護 |

參考資料：聯合大學，陳士杰教授

http://debussy.im.nuu.edu.tw/sjchen/BigData/NoSQL.pdf