



樹德科技大學資訊工程系碩士班

碩士論文

基於 PHP 與 MySQL 應用程式之 Apache HBase 分散式資料庫與

關聯式資料庫中介橋接機制設計與實作

Bridging Apache HBase and Relational Database Operating

Mechanism Design and Implementation Based on PHP and

MySQL Application

研究生：曾坤福撰

指導教授：張晉源 博士

中華民國 101 年 6 月

基於 PHP 與 MySQL 應用程式之 Apache HBase 分散式資料庫與  
關聯式資料庫中介橋接機制設計與實作

Bridging Apache HBase and Relational Database Operating Mechanism  
Design and Implementation Based on PHP and MySQL Application

研究生：曾坤福  
指導教授：張晉源 博士

樹德科技大學  
資訊工程系碩士班  
碩士論文

A Thesis  
Submitted to  
Institute of Computer Science and Information Engineering  
Shu-Te University  
In Partial Fulfillment of the Requirements  
For the Degree of  
Master In  
Computer Science and Information Engineering

June 2012  
中華民國一零一年六月

樹德科技大學碩士班研究生

指導教授推薦書

本校資訊工程系碩士班曾坤福 君所提之論文基於 PHP 與 MySQL 應用程式之 Apache HBase 分散式資料庫與關聯式資料庫中介橋接機制設計與實作

係由本人指導撰述，同意提付審查。

指導教授：張晉源 副教授

101 年 7 月 30 日

樹德科技大學碩士班研究生

學位考試審定書

100 學年度第 2 學期

資訊工程系碩士班 曾坤福 君所提之論文

題目：基於 PHP 與 MySQL 應用程式之 Apache HBase 分散式資料庫與

關聯式資料庫中介橋接機制設計與實作

Bridging Apache HBase and Relational Database Operating  
Mechanism Design and Implementation Based on PHP and MySQL  
Application

經本學位考試委員會審議，認為符合碩士資格標準。

召集人 吳朝貴 委員 \_\_\_\_\_

委員 林政良 委員 \_\_\_\_\_

委員 \_\_\_\_\_ 委員 \_\_\_\_\_

委員 \_\_\_\_\_ 委員 \_\_\_\_\_

指導教授 張晉源 系所主管 陳怡良

中華民國 101 年 8 月 27 日

# 樹德科技大學資訊工程系碩士班

學生：曾坤福

指導教授：張晉源 博士

## 基於 PHP 與 MySQL 應用程式之 Apache HBase 分散式資料庫與 關聯式資料庫中介橋接機制設計與實作

### 摘要

本研究探討開放原始碼分散式資料庫 Apache HBase，採用欄導向 (Column-oriented) 為主的資料存取方式，搭配底層的 Apache Hadoop 運算平台的 HDFS 進行資料儲存，由於 HBase 的架構不同於 MySQL、SQL Server 等傳統關聯式資料庫架構，因此使用傳統關聯式資料庫的應用程式無法直接地轉換使用 HBase 資料庫，目前 HBase 僅提供 Java 語言的客戶端應用程式介面，其它程式語言無法直接地對 HBase 進行資料存取，雖然目前已有 Facebook 所開發的 Thrift 讓其它程式語言可以對 HBase 進行資料存取，但開發者必須先學習相關的 API，且須要重新撰寫既有之應用程式，對開發者與企業的花費成本與風險相當的高，因此本研究提出以 PHP & MySQL 應用程式為例的資料庫存取方法之中介機制，可維持既有程式碼去存取 HBase 資料庫，使開發者無須大幅更改程式碼或學習相關 API 的方式來解決此問題。

關鍵字：HBase、Hadoop、PHP、MySQL、關聯式資料庫

Department of Computer Science and Information  
Engineering, Shu-Te University

Bridging Apache HBase and Relational Database Operating  
Mechanism Design and Implementation Based on PHP and  
MySQL Application

Student : Kun-Fu Tseng

Advisors : Dr. Chin-Yuan Chang

ABSTRACT

In this study, Apache HBase is an open-source, distributed and column-oriented store modeled is based on Hadoop and HDFS. Due to HBase architecture is different from the traditional MySQL, SQL Server and other relational database architecture, for the reason that we use of traditional relational database application cannot be directly converted to HBase database schema. Now, HBase only provides the client libraries of the Java language like the other programming languages cannot directly access information from HBase. At present, Thrift is a framework that developed by Facebook can access HBase Database, but developers must learn the Thrift's library API usage and need to rewrite existing applications, so that the costs and risks that developers and businesses need to sustain are quite high, therefore, this study proposes a method overloading mechanism to access HBase database based on PHP and MySQL, it can maintain the existing code to access the HBase, so the developer does not significantly change the code or learning the other library and API to solve this problem.

Keyword : HBase, Hadoop, PHP, MySQL, Relational database

## 誌謝

首先要感謝指導教授張晉源老師悉心教誨並指示研究方向，促使本論文能夠順利完成，老師在學術與社會經驗所指點的學習方向讓我獲益良多，各方面均有所成長。

感謝口試委員朝陽科技大學洪朝貴老師、工業技術研究院林政良博士、百忙之餘，撥冗擔任學生的口試委員，感謝各位委員的悉心指教，並提出諸多寶貴建議，使本論文更加完善。

感謝碩士班的祿韋、品豪、鈺霖、昇翰、翊晉、伯諭、書毅、國源、俊宇、國宸、思維、昇瀚、宏勳、瑞峰、昱詠、舜全等學長對我學業、技術上的指點幫助；感謝伯勳、俊宏、穹恩、泰豪、禮綱、則炆、富鍵同學在我低潮時為我加油打氣；感謝實驗室的院昇、家賢、貫銘、政揚、龐董、佳緯、承劭協助大小事；感謝電算中心系統開發組的博修、幸眉、俊卿、偉菁、銘隆、銘泉、凱中；網路應用組的淵鐘、柏宇、佑霖、愷儀、嘉駿、力瑀、哲均；行政諮詢組的仁倫、宏義、幼晴、昕昀、智堯、炎輝、耀文、盈江等長輩/同事們工作上的合作與工讀生涯的照顧。

謝謝視傳系龔蒂菀老師對我美學設計上的指導以及六年來的談心、照顧、謝謝公共事務室慧雯姐特地安排為我慶生；特別感謝視傳畢業的瓊微、培峰、景民四年來的案子合作。

感謝 KKBOX 公司南區分局長畢瑄易經理賞識讓我有機會在業界實習，也謝謝親切和善的高雄同事們瑄易局長、采穎、美瑤、文駿、詠淑、Jason、曼榕、育賢、瑋倫、Milly, Maple、承杭、宇成、紫快以及吟秋等前輩從中山大學文學院至高雄軟體科技園區八個月以來的栽培與指導。

感謝 PIXNET 痞客邦公司的皇韋、瑞男、向榮等前輩賞識，讓尚未畢業的我就不用擔心現今社會，不少學生畢業即失業的憂慮，錄取我為公司的研發替代役。

感謝所有文獻、參考資料以及我使用到的所有開放原始碼自由軟體的作者，各位巨人的貢獻功不可沒：「如果說我看得比別人更遠，那是因為我站在巨人的肩膀上。」——牛頓

最後，謹以此拙作獻給我在背後給予最大支持的家人、朋友、師長以及所有在我求學階段曾經交流過的人，謝謝！

曾坤福 一百零一年八月於高雄・樹德科技大學

# 目錄

中文摘要 .....	i
英文摘要 .....	ii
表目錄 .....	vii
圖目錄 .....	viii
第一章 緒論 .....	1
1.1 前言 .....	1
1.2 研究動機與目的 .....	2
1.3 論文架構 .....	3
第二章、相關技術分析與文獻探討 .....	4
2.1 Google Bigtable .....	4
2.2 HBase 簡介 .....	7
2.2.1 HBase 初探 .....	7
2.2.2 HBase資料處理 (Data manipulation) .....	13
2.3 Thrift 簡介 .....	25
2.3.1 Thrift架構 .....	27
2.3.2 Thrift使用範例：PHP存取HBase .....	30
2.4 SQL 結構化查詢語言簡介 .....	31
2.5 Luís Ferreira 提出之消弭 SQL 與 NoSQL之間的時間隙想法 .....	32
2.6 John Roijackers提出之NoSQL query pattern .....	32
2.7 陳韡提出之Web Service 與SQL-HBase mapping模組方法 .....	34
第三章、系統設計與方法 .....	35
3.1 系統架構 .....	36
3.2 運作流程 .....	42



3.3 從MySQL資料遷移至 HBase .....	48
3.4 SQL指令剖析與Thrift HBase API映射 .....	51
第四章、系統實作與實驗設計 .....	54
4.1 實驗環境規格配置 .....	54
4.2 系統實作 .....	56
4.3 實驗一：SHOW TABLES .....	61
4.4 實驗二：HBaseMyAdmin案例實作 .....	65
第五章、結論與未來研究方向 .....	69
參考文獻 .....	70
附錄A HBase篩選器規格與範例參考表 .....	72
通用篩選器字串語法(General Filter String Syntax) .....	72
複合篩選器與運算子(Compound Filters and Operators) .....	72
優先順序(Order of Evaluation) .....	72
比較運算子與比較器(Compare Operator and Comparators) .....	73
PHP Client端程式使用Filter篩選器範例 .....	74
Filter篩選器字串範例 .....	74
附錄B HBase單獨篩選器語法參考表 .....	76
KeyOnlyFilter .....	76
FirstKeyOnlyFilter .....	76
PrefixFilter .....	76
ColumnPrefixFilter .....	76
MultipleColumnPrefixFilter .....	77
ColumnCountGetFilter .....	77
PageFilter .....	77
ColumnPaginationFilter .....	78

InclusiveStopFilter .....	78
TimeStampsFilter .....	78
RowFilter.....	79
Family Filter.....	79
QualifierFilter.....	79
ValueFilter.....	80
DependentColumnFilter .....	80
SingleColumnValueFilter.....	81
SingleColumnValueExcludeFilter .....	81
ColumnRangeFilter .....	82
附錄C 中介橋接機程式碼之 mmb.php .....	83
附錄D 中介橋接機程式碼之 ThriftHBaseClientWrapper.php .....	94
附錄E 中介橋接機程式碼之 sql-spec.php .....	98
作者簡介 .....	103

## 表目錄

表 1. HBase Put類別建構子摘要.....	15
表 2. HBase Put類別方法摘要.....	15
表 3. HBase Get類別方法摘要 .....	17
表 4. HBase Delete類別方法摘要.....	19
表 5. Thrift與其他傳輸方式的資料內容大小比較 .....	29
表 6. Thrift與其他傳輸方式的CPU資源消耗比較.....	29
表 7. Sqoop匯入資料至HBase之指令參數表.....	49
表 8. SQL與Thrift HBase API對應表 .....	52
表 9. SQL與HBase Filter對應表 .....	53
表 10. 系統實作環境硬體規格.....	54
表 11 系統實作環境軟體與開發工具.....	55

## 圖目錄

圖 1. Bigtable的網頁索引資料儲存範例.....	5
圖 2. HBase 邏輯模型：有序Map .....	21
圖 3. 範例資料表myTable中的info欄位家族於HFile儲存範例資料樣貌.....	22
圖 4. Apache Thrift架構.....	27
圖 5. PHP透過Thrift存取HBase程式碼範例.....	30
圖 6. John Roijackers提出之NoSQL query pattern語法範本.....	33
圖 7. 本研究提出之中介橋接機制MH bridge系統架構圖 .....	37
圖 8. MH bridge 中介橋接機制架構圖 .....	38
圖 9. 攔截mysql_connect系統函式之同名函式.....	40
圖 10. 攔截mysql_field_name系統函式之同名函式 .....	41
圖 11. 本研究之中介橋接機制運作流程 .....	42
圖 12. 對於SELECT陳述式之處理流程圖 .....	44
圖 13. 對於UPDATE與DELETE陳述式之處理流程圖.....	46
圖 14. 使用Sqoop匯入資料至HBase過程 .....	48
圖 15. 在hbase shell環境輸入list檢查是否成功匯入資料表.....	50
圖 16. 於hbase shell觀看匯入後的資料表 .....	50
圖 17. 處理SELECT <select list> FROM <table refemce>句型實作程式碼範例.....	57
圖 18. SELECT <select list> FROM <table reference> WHERE <search condition>句 型之實作程式碼 .....	58
圖 19. getMappingPredicate方法實作程式碼.....	59
圖 20. LikePredicate類別實作程式碼.....	60
圖 21. 實驗一SHOW TABLES程式碼.....	61
圖 22. 實驗一SHOW TABLES 程式執行結果.....	62
圖 23. 實驗一載入中介橋接機制後的SHOW TABLES程式碼.....	63
圖 24. 實驗一載入中介橋接機制後的SHOW TABLES執行結果.....	64
圖 25. HBaseMyAdmin專案首頁 .....	65
圖 26. HBaseMyAdmin觀看資料表欄位畫面 .....	66
圖 27. HBaseMyAdmin瀏覽資料表畫面 .....	66
圖 28. HBaseMyAdmin之SQL查詢介面 .....	67
圖 29. HBaseMyAdmin之建立資料表功能畫面.....	67
圖 30. HBaseMyAdmin之匯入資料表功能畫面.....	68

## 第一章 緒論

### 1.1 前言

近年來隨著網際網路急遽發展普及化與網路頻寬速度的升級，網路服務業者提供的應用服務愈來愈豐富多元，如 Facebook、Flickr、Gmail、Google Maps、Google Docs、LinkedIn、Plurk、Slideshare、Twitter、YouTube 等。由於這些服務不再是企業單向的提供資料，使用者們亦可透過分享回饋方式增加資料，例如發布訊息、給予評價、上傳影音照片、共用文件、分享地理位置等，使得資料量呈現爆炸性的成長，據資料顯示，eBay 資料庫每日增加 50 TB、中華電信每一個月增加 4 TB 資料 [1]，越來越多提供運算或儲存服務的企業使用巨量資料 (Big Data) 來描述所面臨到的嚴峻挑戰。巨量資料一詞是指資料集中儲存的資料相當龐大，超出傳統軟體工具所能處理的資料量，巨量資料的定義逐步上修，截至 2012 年，一個資料集的大小已從數 10 TB 至數百 PB (Petabyte)，目前常見的巨量資料像是伺服器的 Log 記錄、搜尋引擎索引、社群網路資料、使用者行為記錄、網際網路文件、天文地理資料、氣象科學分析、生物基因、醫學記錄等。不僅網路服務或科學研究遭遇巨量資料的挑戰，一般企業電子化後也將儲存愈來愈多的資料，例如統一發票無紙化、政府機關公文電子化、醫療影像及病歷等。

當傳統資料庫無法滿足儲存巨量資料需求時，以分散式運算處理，有著雲端運算(Cloud computing)特色的分散式資料庫會是較好的選擇。1998 年 Carlo Strozzi 提出一個開放原始碼、不提供 SQL 功能的資料庫 [2] 雖然在當時沒有成為主流，直到近年出現許多使用者提供資料的網站，帶動分散式資料庫的需求，為了滿足資料成長需求，傳統的關聯式資料庫如 Oracle、SQL Server 必須借助資料庫叢集技

術方能解決，然而這樣的解決方案須花費高額經費擴充軟硬體。為了解決巨量資料儲存與擴充問題，陸續有人研發提出不同的分散式資料庫並開放原始碼供給社群開發者投入研究持續改進，近年來知名網路公司如 Google 研發的 BigTable 即為一例、Amazon 亦提出 DynamoDB、Yahoo 也投入 Hadoop 研發。

## 1.2 研究動機與目的

由於近代網路應用程式造成儲存資料量快速成長，各種非關聯式資料庫陸續被提出，舊有的應用程式如何節省研發資源成本來達到遷移至非關聯式資料庫成為一個新的議題。典型的資料庫應用程式透過 SQL 指令與資料庫系統存取資料，前面所提的分散式資料庫並不具備 SQL 指令的支援，如此當我們想遷移資料庫至這些分散式資料庫時，應用程式必須大幅度的修改程式碼，並且開發者還須要學習各種分散式資料庫的 API 方可完成。對於一個已上線的網路應用程式，大幅改寫程式碼是風險相當高的挑戰。

因此本研究以常見的基於 PHP 程式語言與 MySQL 資料庫系統之應用程式以及 HBase 分散式資料庫為例，提出一種以 SQL 指令介接轉換 HBase 分散式資料庫 API 的機制，讓現有的應用程式可以保留透過 SQL 存取 MySQL 資料庫系統的程式碼，無須做修改便可存取 HBase 分散式資料庫系統的資料。並且透過實驗顯示出本研究提出之機制的確可達到介接轉換的目的。

### 1.3 論文架構

本研究主要探討 MySQL 關聯式資料庫遷移至 HBase 非關聯式資料庫後，如何避免修改既有的應用程式程式碼，以既有程式的 SQL 指令存取非關聯式資料庫，降低開發人員的負擔與程式大幅修改的風險。

本論文章節介紹如下：

1. 第一章緒論，簡述巨量資料時代的來臨，企業逐漸導入非關聯式資料庫系統和研究動機與目的以及論文的架構。
2. 第二章為文獻探討，探討近年來發展之技術，主要針對 Bigtable、Apache Thrift、SQL 結構化查詢語言這些技術進行更深一步的探討。
3. 第三章為系統設計與方法，包含本研究所提出的中介橋接機制系統架構、運作流程、SQL 指令剖析與 HBase API 映射等。
4. 第四章為實驗設計與驗證，介紹系統實驗環境與相關軟體、開發工具以及本系統操作畫面。
5. 最後第五章為結論與未來研究方向，探討可繼續研究加強的議題。在本論文的最後，附上過程中所引用及參考之文獻資料以及附錄。

## 第二章、相關技術分析與文獻探討

### 2.1 Google Bigtable

Bigtable 是一個分散式的結構化資料儲存系統，用來處理 Petabytes 級別以上的巨量資料，通常分散於數千台伺服器上。Google 的許多應用服務廣泛使用 Bigtable 來儲存資料，包含 Web indexing (搜尋引擎索引)、Google Earth (衛星及航空影像)、Google Maps (地圖與街景資訊)、Google Finance (財經股市金融數據)、Google Blogger (部落格文章)、YouTube 影音以及 Gmail 電子郵件等。這些應用的儲存資料各自有獨特需求，無論是在資料規模上（從網頁索引到衛星影像）以及延遲性差異（從後端的批次資料處理到即時資料服務），Bigtable 仍然成功地提供靈活與高效能的解決方案。Bigtable 設計了與常見資料庫完全不同的介面。它不支援完整的關聯式資料模型，僅提供簡單的資料模型，對 Bigtable 而言，資料是沒有綱要(Schema)的，由使用者自行定義，如此一來成就了 Bigtable 帶給各種產品的儲存需求靈活度。[3]

#### Bigtable 資料模型

Bigtable 由多維度有序的 Map 組成，Map 由索引與值組成(Key/Value)，可想像為點名表(Map)-學號(Key)-姓名(Value)，在點名表上查詢學號可取得姓名。



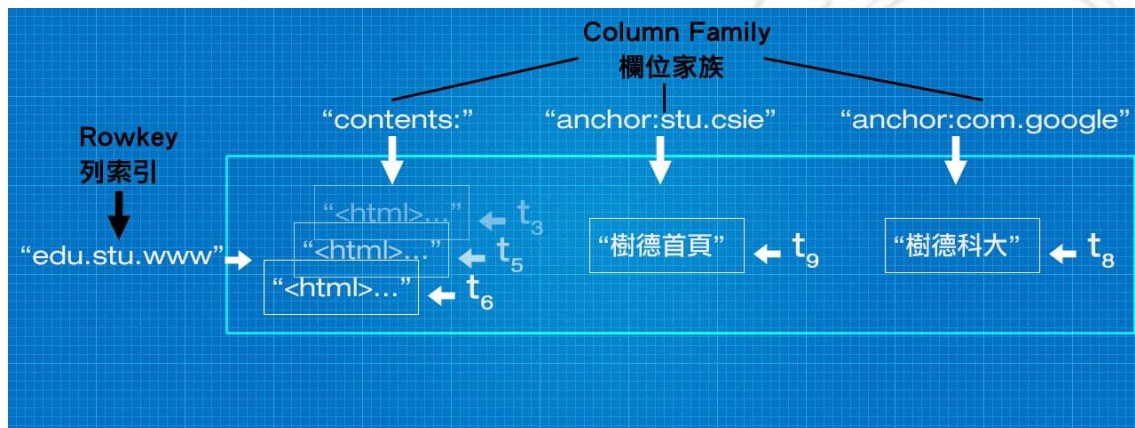


圖 1. Bigtable 的網頁索引資料儲存範例

圖 1 為 Bigtable 的網頁索引資料儲存範例，列索引存放網址，欄位家族 “contents:” 與 “anchor:” 則分別儲存網頁原始碼內容(HTML)及該網頁被哪些網站引用(超連結)，並依照擷取網頁內容的時間作為時間戳記，以儲存多個版本的網頁內容。

Bigtable 的 Map 其索引由以下三個部分組成：

(1) 列索引 (Row key)

例如 “edu.stu.www”，類似關聯式資料庫的主鍵 (Primary Key)，也是 Bigtable 內部排序資料的根據。

(2) 欄索引 (Column key)

欄索引組成的集合為「欄位家族」(Column Family)，儲存於相同欄位家族下的所有資料通常有關聯性，且能壓縮在一起。例如一個名為 Contact 的欄位家族有著聯絡電話、通訊地址、傳真號碼相同性質的欄索引。

欄位家族在使用之前必須建立，爾後方可在欄位家族中任意建立欄索引並儲存資料。根據 Google 設計，Bigtable 的資料表的欄位家族數量最多數百個，產品上線後較少變更，而欄位家族中的欄索引則沒有數量限制。

欄索引的命名方式為「family:qualifier」(欄位家族:限定詞)，欄位家族名稱必須是可被印出的文字，限定詞可以是任意文字。以圖 1 為例，anchor 為其中

一個欄位家族名稱，該欄位家族的每一個欄索引代表一個被引用的超連結，分別有 stu.csie 以及 com.google，依序儲存「樹德首頁」與「樹德科大」，代表 stu.csie 網頁以「樹德首頁」超連結至“edu.stu.www”；com.google 網頁以「樹德科大」超連結至“edu.stu.www”。

(3) 時間戳記 (Timestamp)

如  $t_3$ 、 $t_5$ 、 $t_6$ 、 $t_9$ 、 $t_8$ ，時間戳記為寫入 Bigtable 時自動加入，這裡用來表達擷取網頁的時間。



## 2.2 HBase 簡介

HBase 是一個開放原始碼的分散式資料庫，其具備高可靠性與延展性。HBase 可儲存結構化及半結構化(semi-structured)的資料，例如文字訊息與系統日誌等。HBase 也能儲存非結構化資料，即影音、圖像檔案，結合 Hadoop 分散式檔案系統可達到良好的效能與備份目的。HBase 並非一般人熟悉的關聯式資料庫架構，它不支援 SQL 結構化查詢語言指令、資料表之間的關聯性以及交易(transaction)機制以及字串以外的欄位型態，也不介意我們儲存重複的資料在同一列的其他欄位。HBase 並不要求使用價格昂貴的磁碟陣列儲存設備，或是在單一機器裝配大量記憶體。不過，HBase 更傾向使用者以多部一般等級的硬體設備佈署 HBase 叢集。HBase 是專門設計用來運作於叢集(cluster)環境的資料庫，並非單一機器。在叢集環境的每個節點(node)提供儲存、快取以及運算的用途。這使得 HBase 有著卓越的擴充彈性及容錯性。沒有任何節點是獨一無二的，假設叢集環境某一個節點故障，我們只需換掉它，完全不影響整體運作，此特性讓 HBase 有著良好的延展性(Scalability)，易於擴充資料庫叢集的規模。[4][5][6]

### 2.2.1 HBase 初探

HBase 使用 Java 程式語言撰寫而成，可整合在 Apache Hadoop [7]與 Apache Zookeeper [8]之上，亦可單獨安裝。以下是安裝 HBase 的步驟，適用於 Ubuntu 10.04 Lucid 作業系統 [9]。

將 Cloudera CDH3 套件[10]加入 apt 檔案庫



```
$ sudo apt-get install python-software-properties
$ sudo add-apt-repository "deb http://archive.canonical.com/
lucid partner"
$ wget
http://archive.cloudera.com/one-click-install/lucid/cdh3-re
pository_1.0_all.deb
$ sudo dpkg -i cdh3-repository_1.0_all.deb
```

當檔案庫設定完成後，更新 apt 並安裝 Java SDK 及 HBase Master

```
$ sudo apt-get update
$ sudo apt-get install sun-java6-jdk hadoop-hbase-master
```

由於昇陽公司(Sun)已於 2009 年被甲骨文公司(Oracle)併購 [11]，使用以下指令確保我們安裝的是 Sun Java JDK：

```
$ apt-cache policy sun-java6-jdk hadoop-hbase
sun-java6-jdk:
Installed: 6.26-2lucid1
Candidate: 6.26-2lucid1
Version table:
*** 6.26-2lucid1 0
500 http://archive.canonical.com/ lucid/partner Packages 100
/var/lib/dpkg/status
hadoop-hbase:
Installed: 0.90.4+49.1-1~lucid-cdh3
```

Candidate: 0.90.4+49.1-1~lucid-cdh3

Version table:

\*\*\* 0.90.4+49.1-1~lucid-cdh3 0500

<http://archive.cloudera.com/debian/lucid-cdh3/contrib>

Packages 100 /var/lib/dpkg/status

HBase 安裝完成後，接下來要啟動 HBase，這會一併執行 Zookeeper 監控 HBase 節點群的運作。

```
$ sudo /etc/init.d/hadoop-hbase-master start
```

```
Starting Hadoop HBase master daemon: starting master, logging  
to /usr/lib/hbase/logs/hbase-hbase-master-ubuntu.out
```

```
2011-08-22 23:40:18,709 INFO server.ZooKeeperServer: Server  
environment:zookeeper.version=3.3.3-cdh3u2--1, built on  
07/18/2011 16:48 GMT
```

```
...
```

```
2011-08-22 23:40:18,709 INFO
```

```
org.apache.zookeeper.server.ZooKeeperServer: Server  
environment:os.name=Linux
```

```
...
```

到此我們已成功安裝 HBase 於單機模式(stand-alone)，我們可以在此環境練習熟悉 HBase 的操作以及本地端環境上的程式開發。HBase 也能執行於虛擬分散式模式(pseudo-distributed)，即一台機器上運作多個 Java Processes 來達到模擬分散式叢集環境的模式。待環境備妥後我們便可將 HBase 運作於完整分散式模式

(full-distributed)。

HBase 提供 HBase Shell 讓我們可以在命令列(command line)上與其互動操作。使用以下指令進入 HBase Shell：

```
$ hbase shell
```

```
HBase Shell; enter 'help<RETURN>' for list of supported  
commands.
```

```
Type "exit<RETURN>" to leave the HBase Shell
```

```
Version 0.92.0, r1231986, Mon Jan 16 13:16:35 UTC 2012
```

```
hbase(main):001:0>
```

如沒有出現任何錯誤訊息表示 Java 與 HBase 皆能正常執行，接著使用 list 指令列出資料庫中所有的資料表：

```
hbase(main):001:0> list
```

```
TABLE
```

```
event
```

```
student
```

```
subject
```

```
3 row(s) in 0.4750 seconds
```

### 2.2.1.1 新增表格

在 HBase，新增資料表的指令是 `create`，語法如下：

`create '表格名稱', '表格名稱', '欄位家族名稱'`

HBase 規定資料表必須至少有一個欄位家族方可新增，若下指令時沒有指定欄位家族名稱則會出現 “Table must have at least one column family” 的錯誤訊息。

以下指令(建立一個名為 `MyFirstTable` 的資料表，欄位家族名稱為 `info`。

```
hbase(main):003:0> create 'MyFirstTable', 'info'
```

```
0 row(s) in 1.1560 seconds
```

### 2.2.1.2 觀察表格綱要(Schema)

在新增表格的小節裡，我們可以發現新增 HBase 的資料表並不須要像關聯式資料庫如 MySQL、SQL Server 等指定欄位的資料型態，這便是 HBase 時常被稱作 schema-less 類型的資料庫，簡單的說就是沒有嚴格定義綱要(Schema)的資料模型。


HBase 的 `describe` 指令讓我們取得更多資料表的內部資訊，例如定義了哪些欄位家族、保留多少個版本數、資料表是否為壓縮狀態等等。以下使用 `describe` 指令列出 `MyFirstTable` 的內部資訊：

```
hbase(main):005:0> describe 'MyFirstTable'
```

```
DESCRIPTION
```

```
ENABLED
```

```
{NAME => 'MyFirstTable', FAMILIES => [{NAME => 'info', BLOOMFILTER => 'NONE', REPLICATION_SCOPE => '0', VERSIONS => '3', true
```



```
COMPRESSION => 'NONE', MIN_VERSIONS => '0', TTL => '2147483647',  
BLOCKSIZE => '65536', IN_MEMORY => 'false', BLOCKCACHE =  
> 'true']}]}
```

1 row(s) in 0.0410 seconds

### 2.2.1.3 建立連線

前面兩個小節介紹 HBase Shell 環境來實際練習指令，不過 HBase Shell 並不能讓我們開放應用程式，要讓應用程式存取 HBase，必須透過 API 應用程式介面來達成，由於 HBase 以 Java 程式語言開發而成，自然提供 Java 語言原生 API，以下 Java 程式碼將建立一個連線至 MyFirstTable：

```
HTableInterface usersTable = new HTable("MyFirstTable");
```

HTable 建構子將會讀取預設的環境配置資訊來載入 HBase，我們也可以透過以下程式碼傳入自訂的 HBaseConfiguration 環境配置物件予 HTable 建構子：

```
Configuration myConf = HBaseConfiguration.create();  
HTableInterface myTable = new HTable(myConf, "users");
```

我們也能在建立 HBaseConfiguration 類別實例 myConf 後，透過 set 方法動態修改環境參數的值，如下：

```
myConf.set(“參數名稱”， “參數值” );
```





#### 2.2.1.4 連線管理

建立資料表的類別實例須要一個網路連線，效能花費較為高昂，因此我們可以使用連線池(Pool)的方式來取得資料表的實例會是較好的實踐方式。

```
HTablePool pool = new HTablePool();  
HTableInterface myTable = pool.getTable("MyFirstTable");  
...  
// 使用完畢後關閉表格  
myTable.close();
```

#### 2.2.2 HBase 資料處理 (Data manipulation)

HBase 的資料表中，每一列都有著唯一的識別字，就像 Google Bigtable 一樣，有列索引(row key)的存在，在資料表中與資料的互動便由列索引(row key)開始。假如我們要建置一個會員系統，會員帳號具有唯一特性，就可以作為列索引。HBase 有五個主要的指令用來與其互動：get、put、delete、scan 以及 increment，分別用來取得、寫入、刪除、掃描與遞增，以下將逐一介紹各個指令。

##### 2.2.2.1 新增資料

以下 Java 程式碼用來新增一位會員，並且加入其姓名、email、密碼等資訊：

```
Put p = new Put(Bytes.toBytes("afutseng"));  
  
// A. 新增姓名  
p.add(Bytes.toBytes("info"),
```



```
Bytes.toBytes("name"),  
Bytes.toBytes("曾坤福"));  
  
// B. email  
p.add(Bytes.toBytes("info"),  
Bytes.toBytes("email"),  
Bytes.toBytes("s99639102@stu.edu.tw"));  
  
// C. 密碼  
p.add(Bytes.toBytes("info"),  
Bytes.toBytes("password"),  
Bytes.toBytes("weakPassword"));
```

上述程式碼使用 Put 類別建立一個列索引“afutseng”，接著呼叫其 add 方法加入資料，add 方法傳入之第一個參數為欄位家族名稱(Column Family)，第二個參數為欄位限定詞(qualifier)，第三個參數是存入的資料，最後儲存的樣子如以下的條列項目表達：

```
//A. 在 “info:name” 儲存"曾坤福"  
//B. 在 “info:email” 儲存 "s99639102@stu.edu.tw"  
//C. 在 “info:password” 儲存"P@55w0rd"
```

完成 Put 實例後，尚須呼叫 HTableInterface 實例 myTable 的 put 方法將 p 加入以及叫用連線池 pool 的 putTable 方法將 myTable 傳入方可把資料寫入 HBase。程式碼如下：

```

HTableInterface myTable = pool.getTable("myTable");

Put p = new Put(Bytes.toBytes("afutseng"));

p.add(...);

myTable.put(p);

pool.putTable(myTable);

```

表 1、表 2 列出本章節提及的Put建構子與方法說明，摘錄自HBase 官方API文件 [12]

表 1. HBase Put類別建構子摘要

方法名稱及參數	說明
Put()	可事後指定列索引的建構子
Put(byte[] row)	建立一個指定列索引的Put實例
Put(byte[] row, long ts)	建立一個指定列索引並給予時間戳記的Put實例，

表 2. HBase Put 類別方法摘要

回傳型別	方法名稱、參數與說明
Put	<b>add(byte[] family, byte[] qualifier, byte[] value)</b> 新增指定的欄位與值至呼叫的Put實例
Put	<b>add(byte[] family, byte[] qualifier, long ts, byte[] value)</b> 新增指定的欄位與值以及時間戳記作為版本至呼叫的Put實例
Put	<b>add(KeyValue kv)</b> 新增指定的鍵值對(KeyValue)至呼叫的Put實例



### 2.2.2.2 修改資料

在 HBase 中，修改資料也由 put 類別完成，直接寫入新的資料即可。以下範例修改會員帳號 afutseng 的 info:password 欄位：

```
Put p = new Put(Bytes.toBytes("afutseng"));
p.add(Bytes.toBytes("info"),
Bytes.toBytes("password"),
Bytes.toBytes("MoreSecureP@55w0rd"));
myTable.put(p);
pool.putTable(myTable);
```

### 2.2.2.3 讀取資料

透過 Get 類別與資料表的 get 方法可以從 HBase 讀取資料，例如：

```
Get g = new Get(Bytes.toBytes("afutseng"));
Result r = myTable.get(g);
```

資料表會傳回一個 Result 實例，包含這個列索引的所有欄位家族的所有欄位。由於取回的資料可能遠大於我們所需要的，因此我們可以限制傳回的資料量，根據表 3 [13]，使用 addFamily()方法取得特定欄位家族或是使用 addColumn()方法取得特定欄位，addFamily()方法會傳回所有欄位(Column)，使用上須注意；以下程式碼可取回指定的會員帳號的密碼欄位。

```

Get g = new Get(Bytes.toBytes("afutseng"));

g.addColumn(

Bytes.toBytes("info"),

Bytes.toBytes("password"));

Result r = myTable.get(g);

```

表 3. HBase Get 類別方法摘要

回傳型別	方法名稱、參數與說明
Get	<b>addColumn</b> (byte[] family, byte[] qualifier) 取得指定的欄索引（欄位家族:限定詞）
Get	<b>addFamily</b> (byte[] family) 取得指定的欄位家族其所有欄位
Map<byte[],NavigableSet<byte[]>>	<b>getFamilyMap</b> () 取得所有欄位家族Map
byte[]	<b>getRow</b> () 取回指定的列

要從 Result 實例取出我們撰寫程式時常見的字串，可以透過 `getValue()` 方法：

```

byte[] b = r.getValue(
Bytes.toBytes("info"),
Bytes.toBytes("email"));

String email = Bytes.toString(b);

```

上述程式碼中的 email 字串變數即為 "s99639102@stu.edu.tw" 字串。

#### 2.2.2.4 刪除資料

Delete 類別用來刪除 HBase 資料表中的資料，簡單地傳入列索引即可刪除該列：

```
Delete d = new Delete(Bytes.toBytes("afutseng"));  
myTable.delete(d);
```

我們也可以刪除特定的欄位，例如刪除 email 欄位：

```
Delete d = new Delete(Bytes.toBytes("afutseng"));  
d.deleteColumns(  
Bytes.toBytes("info"),  
Bytes.toBytes("email"));  
myTable.delete(d);
```

根據表 4 內容，其中 deleteColumns()方法將會刪除該欄位的所有版本資料，若只想移除該欄位最後一個版本的資料，則可以使用 deleteColumn()方法。[14]

表 4. HBase Delete 類別方法摘要

回傳型別	方法名稱、參數與說明
Delete	<b>deleteColumn</b> (byte[] family, byte[] qualifier) 刪除指定欄位(欄位家族:限定詞)的最後一個版本資料
Delete	<b>deleteColumn</b> (byte[] family, byte[] qualifier, long timestamp) 刪除指定欄位與指定版本(時間戳記)的資料
Delete	<b>deleteColumns</b> (byte[] family, byte[] qualifier) 刪除指定欄位的資料 (所有版本)
Delete	<b>deleteColumns</b> (byte[] family, byte[] qualifier, long timestamp) 刪除指定欄位的資料 (包含指定版本及更舊版本)
Delete	<b>deleteFamily</b> (byte[] family) 刪除指定的欄位家族及指定版本所有欄位資料
Delete	<b>deleteFamily</b> (byte[] family, long timestamp) 刪除指定的欄位家族所有欄位資料 (包含指定版本及更舊版本)

### 2.2.3 HBase 資料模型 (Data Model)

HBase 的資料模型與普遍熟悉的關聯式資料庫有所不同，關聯式資料庫的資料表有著嚴格的要求，欄位與資料型態，符合這些要求的資料被稱作結構化資料。HBase 被設計用來專門處理沒有這些嚴格要求的半結構化與非結構化資料，例如：每筆資料列可以有不同數量的欄位數量；以下將介紹 HBase 資料的邏輯模型與實體模型。

#### 2.2.3.1 邏輯模型：Sorted Map

在程式設計語言中，Map 或 Dictionary 是 Key/Value 的實作，指定一個索引鍵查詢 Map 中的資料，好比以英文單字找到字典中該單字，得到它的詳細解說；想像 HBase 是一個無上限、蜂巢狀並可區別資料版本的 Map 結構。HBase 使用「列索引、欄位家族、欄位限定詞、時間戳記」作為 key 來儲存資料，若用 Java 程式語言的 Map 容器來描述的話，可讀作：

`Map<RowKey, Map<ColumnFamily, Map<ColumnQualifier, Map<Version, Data>>>>>`



圖 2 介紹 HBase 的邏輯模型，即有序的 Map，HBase 將資料組織化為蜂巢狀的 map，每個 map 都根據 map 的索引鍵(key)來做字典排序，HBase 的欄位限定詞(qualifier)也是 map，qualifier 名稱為 key，value 由一或多個以時間戳記作為 key 的 map 組成，較新的版本排在舊版本前面，因此在本案例中 Rowkey 為 afutseng 的三個欄位順序便為 email、name、password 三個 key 的字典排序；新版本的密碼 “P@55w0rd” 在舊版本 “weakPassword” 之前。

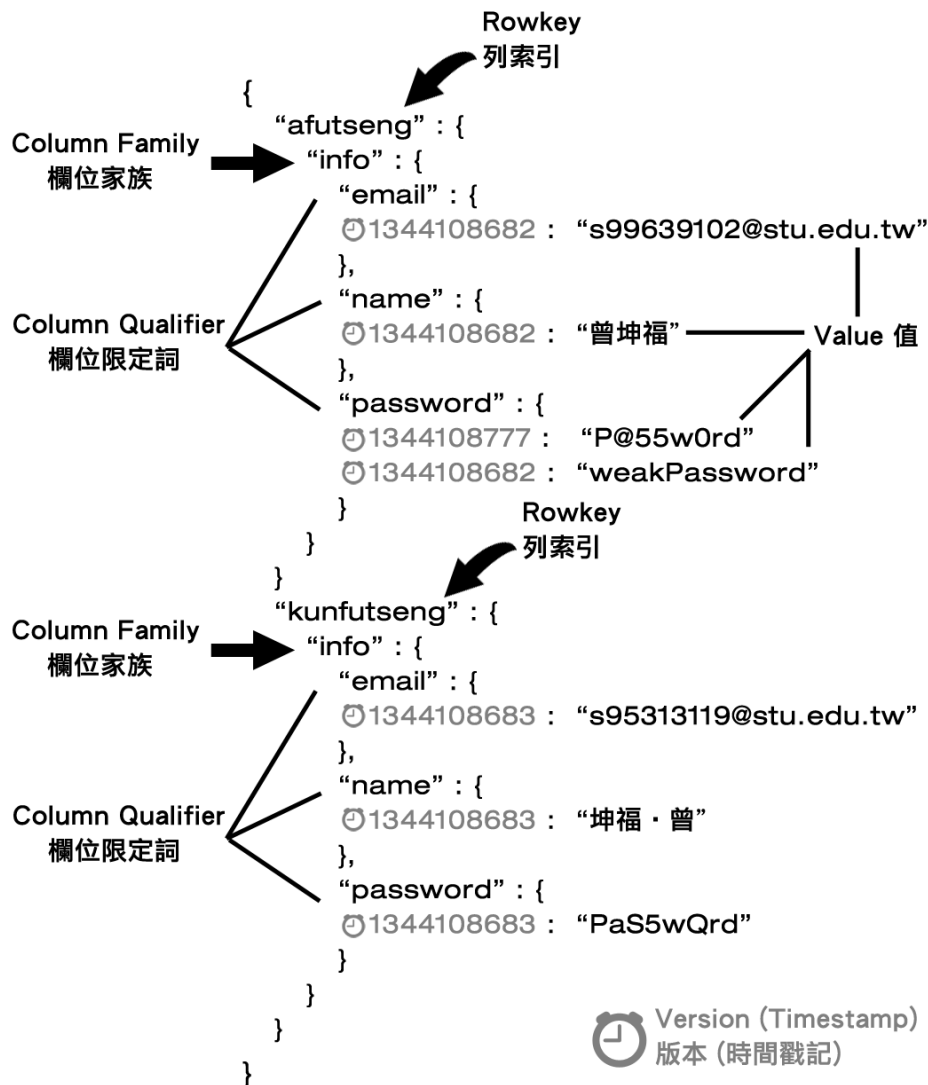


圖 2. HBase 邏輯模型：有序 Map

### 2.2.3.2 實體模型：Column-oriented

如同關聯式資料庫，HBase 的資料表也有列(row)與欄(column)，HBase 資料表將多個欄位群組成一個欄位家族(Column Family)。每一個欄位家族在實體儲存上有各自的 HFile。這種實體隔離允許每一個欄位家族可被個別儲存及管理。HBase 資料表中的一筆資料記錄在 HFile 內以多個 key-value pairs 儲存，HFile 本身為二進制格式。圖 3 說明前一小節的範例資料儲存在 HFile 的樣貌，每一個欄位限定詞有著自己的紀錄：列索引、限定詞、時間戳記、值。假如列索引“afutseng”在其它的欄位家族及欄位限定詞也有儲存資料，由於欄位家族有各自的 HFile，HBase 在讀取資料並不需要讀取所有的 HFile，因為實務上會指定讀取特定的欄位家族與欄位限定詞。

**“afutseng”, “info”, “name”, 1344108682, “曾坤福”**  
**“afutseng”, “info”, “email”, 1344108682, “s99639102@stu.edu.tw”**  
**“afutseng”, “info”, “password”, 1344108777, “P@55w0rd”**  
**“afutseng”, “info”, “password”, 1344108682, “weakPassword”**

圖 3. 範例資料表 myTable 中的 info 欄位家族於 HFile 儲存範例資料樣貌

## 2.2.4 HBase 表格掃描 (Table Scan)

Scan 類別用來讀取資料表的某部分範圍的資料也可以結合篩選器(Filter)過濾符合條件的資料，由於 HBase 資料記錄在內部是依照列索引字典排序，因此透過 Scan 取得列索引 A 到列索引 B 的範圍資料是相當快速的。

### 2.2.4.1 全表掃描

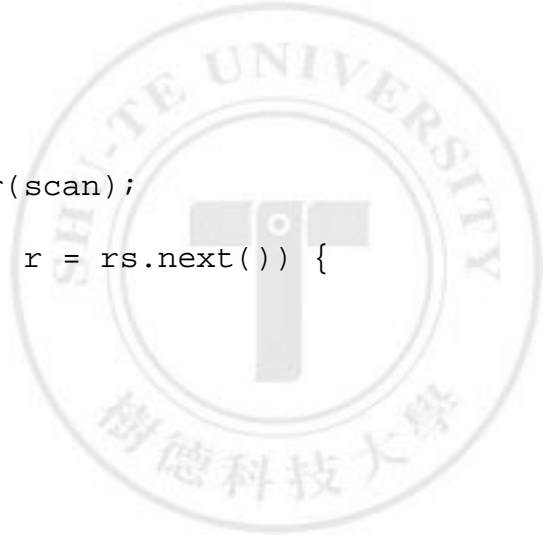
在沒有指定起始與終止列索引以及篩選器情形下，Scan 會取回全部的資料列索引，以下程式碼可取得 myTable 資料表的欄位家族 info 的所有欄位限定詞的全部資料列

```
Scan scan = new Scan();
scan.addFamily(Bytes.toBytes("info"));
ResultScanner rs = myTable.getScanner(scan);
for (Result r = rs.next(); r != null; r = rs.next()) {
    // process result...
}
rs.close();
```

### 2.2.4.2 鍵值(Row Key)掃描

Scan 類別建構子可讓我們指定列索引鍵值(RowKey)以取得指定的資料列，例如以下程式碼將取回 “afutseng” 資料列：

```
Scan scan = new Scan(Bytes.toBytes("afutseng"));
```



```
ResultScanner rs = myTable.getScanner(scan);
for (Result r = rs.next(); r != null; r = rs.next()) {
    // process result...
}
rs.close();
```

### 2.2.4.3 範圍掃描

Scan類別的 `setStartRow` 以及 `setStopRow` 兩個方法可以指定起始列索引與終止列索引以取得某部分範圍的資料列。

```
Scan scan = new Scan();
scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"));
;
scan.setStartRow( Bytes.toBytes("row1"));
scan.setStopRow( Bytes.toBytes("row9"));
ResultScanner rs = myTable.getScanner(scan);
for (Result r = rs.next(); r != null; r = rs.next()) {
    // process result...
}
rs.close();
```

上述範例程式碼作用為建立一個 Scan 實例，設定起始列索引為“row1”，終止列索引為“row9”，挑選欄位家族 info 的 name 欄位限定詞，終止列索引是排除在外的，假設資料表有九筆資料紀錄，列索引依序為 row1 至 row9，那麼

本例將取回 row1 至 row8。

#### 2.2.4.4 篩選器(Filter)掃描

有時候我們需要取得符合特定條件的資料，好比姓名為王小明的使用者、瀏覽人次超過 100 的網址文章、計算研討會報名表勾選素食的參與者等等。對於關聯式資料庫，SQL 結構化查詢語言的 WHERE 條件式可以滿足以上的需求；HBase 則提供篩選器(Filter)結合 Scan 掃描取出我們想要的特定資料，這也可以避免很多不必要的資料傳輸，節省網路頻寬與記憶體。舉例而言，我們可以使用 ValueFilter 與 RegexStringComparator 來達成找出姓名欄位 name 有“曾”的資料列：

```
Scan scan = new Scan();
scan.addColumn(Bytes.toBytes("info"), Bytes.toBytes("name"))
;
Filter f = new ValueFilter(
    CompareOp.EQUAL,
    new RegexStringComparator(".*曾.*")
);
scan.setFilter(f);
```

### 2.3 Thrift 簡介

Apache Thrift [15][16]源於知名社群網站 Facebook，2007 年 Facebook 貢獻 Thrift 原始碼給 Apache 基金會，使其成為 Apache 基金會下的開放原始碼專案。[17] Facebook 創造 Thrift 是為了解決其系統中各個子系統大量資料的傳輸通訊以及各

系統之間的程式語言及環境皆有所差異因而需要跨平台的特性。Thrift 支援多種程式語言，例如：C#、C++、Cocoa、Erlang、Haskell、Java、Ocaml、Perl、PHP、Python、Ruby、Smalltalk。Thrift 適用於大型資料交換的通用工具，對於大型系統中的內部資料傳輸相對於 XML 與 JSON 來說，Thrift 無論在效能及資料傳輸大小上有明顯的優勢。

Thrift 是用來開發通用、可延展的服務的軟體框架，使用它所提供的介面描述語言 (IDL) 可以讓我們定義服務，無需考慮底層要用哪種程式語言實作，只要使用 Thrift code generation 來幫助我們產生各種語言所需要的檔案，就可以使用多種程式語言來達成一或多種服務的開發，因此我們可以在不同環境下用其適合的語言來達成需求。

Thrift 看似與 RESTful Web Service 相仿，以 HTTP 協定及 XML 介面為例，只要環境、程式語言能處理 HTTP Request/Response 以及解析 XML，同樣可以用不同的程式語言來達成，為何不使用 HTTP+XML 而要使用 Thrift 技術我們可用一個情境來看：現在有一查詢介面發送 HTTP Request 要求伺服器傳回一萬筆資料，伺服器便會去資料庫撈取資料，並將查詢結果在伺服器轉換成一份包含一萬筆的 XML 或 JSON 格式的文件，再經過伺服器的轉送傳回，這中間將花費不少網路頻寬來傳送這份查詢結果文件。而 Thrift 提供最佳化的序列化工具，無論是透過何種協定傳送資料 (TCP、HTTP 等)、傳回何種類型的資料格式 (XML、JSON、Binary)，都有 API 可以使用，在時間與空間使用上都有不錯的效能表現，可節省使用頻寬，來提升服務彼此之間溝通傳輸的效率。

### 2.3.1 Thrift 架構

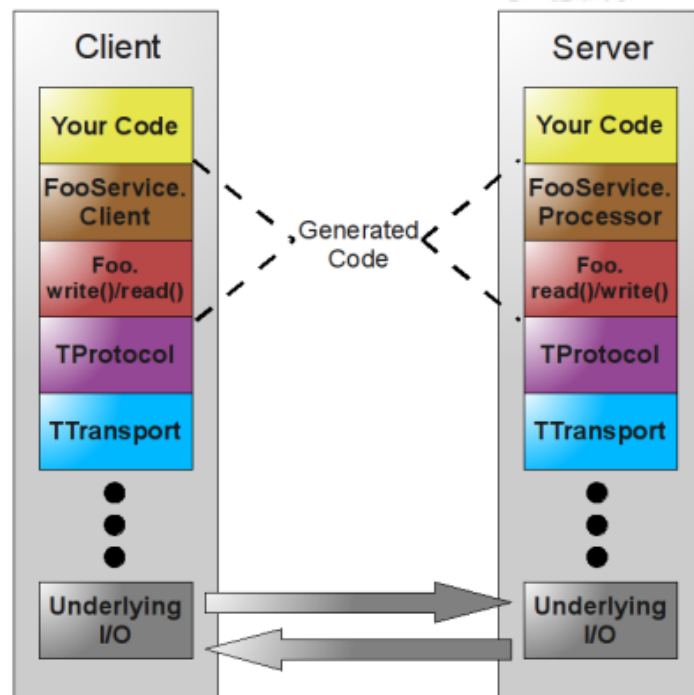


圖 4. Apache Thrift 架構

如圖 4. [18] 所示，上面三層 (Your Code、FooService、Foo. Write) 是透過 Thrift 產生的程式碼。圖中的棕色部份 FooService.Client 和 FooService.Processor 主要是產生終端和處理器程式碼，圖中的紅色部份 Foo.write( )/read( )和 Foo.read( )/write( )表示被發送的資料結構 (除了內建的形態之外) 也會產生程式碼的結果。protocol 和 transport 是 Thrift runtime 函式庫的一部分，因此使用者可以定義服務，而且自由的改變 protocol 和 transport 而不用重新產生程式碼；Client 端在 Thrift 描述檔案中描述自己的服務，這些服務經過編譯後會生成相對應語言的程式碼檔案。

Thrift 提供了以下的資料型別(Types)供開發者定義其服務：





Base Types：基本類型

Struct：結構體類型

Container：容器類型，即 List、Set、Map

Exception：例外類型

Service：定義物件的介面

通訊協定：

Thrift 可以讓我們選擇 Client 端與 Server 端之間的傳輸通訊協定類型，整體上分為文字 (text) 和二進位 (binary) 傳輸協定，為節省頻寬提升效率，一般多使用二進為類型作為傳輸協定，有時會根據實際需求使用文字類型的協定，Thrift 提供的通訊協定如下：

TBinaryProtocol：二進位編碼格式進行資料傳輸。

TCompactProtocol：使用 Variable-Length Quantity (VLQ) 編碼進行資料壓縮。

TJSONProtocol：使用 JSON 格式的資料編碼進行傳輸。

TDebugProtocol：在開發的過程中幫助開發人員測試用的，以純文字的形式以方便閱讀。

### **Apache Thrift 與其他傳輸方式的比較**

XML 與 JSON 相比體積太大，但是 XML 廣泛被使用，複雜度亦不高。

JSON 體積較小，但相較於 XML 與各種語言的支援度較差一點。

Thrift 體積非常小，使用起來比較麻煩，不如前兩者方便，但是對於資料傳輸量大、多種程式語言環境，使用 Thrift 還是值得的。

假設需要傳輸相同的資料內容，從 1. 傳輸內容所產生的大小 2. 傳輸過程中 Server 端和 Client 端所產生的資源消耗，這兩個項目進行比較。使用 Thrift 和其他方式的



所產生的內容大小比較結果如下：

表 5. Thrift 與其他傳輸方式的資料內容大小比較

方法	大小	與 TCompactProtocol 比較
Thrift - TCompactProtocol	278	N/A
Thrift - TBinaryProtocol	460	65.47%
Protocol Buffers	250	-10.07%
RMI (using Object Serialization for estimate)	905	225.54%
RESTful – JSON	559	101.08%
RESTful - XML	836	200.72%

在表 5 中我們可看出，體積最大的是 RMI，其次是 XML，使用 Thrift 的 TCompactProtocol 協定和 Google 的 Protocol Buffers 相差不大。

使用 Thrift 中的協定和其他方式的所產生的 CPU 資源消耗比較結果如表 6：

表 6. Thrift 與其他傳輸方式的 CPU 資源消耗比較

方法	Server CPU %	平均 Client CPU %
RESTful - XML	12.00	80.75
RESTful – JSON	20.00	75.00
RMI (using Object Serialization for estimate)	16.00	46.50
Protocol Buffers	30.00	37.75
TBinaryProtocol	33.00	21.00
Thrift - TCompactProtocol	30.00	22.50

### 2.3.2 Thrift 使用範例：PHP 存取 HBase

```
1 <?php
2
3 //1. 引入 Thrift 函式庫
4 $GLOBALS['THRIFT_ROOT'] = '/var/www/hbase/thrift';
5 require_once( $GLOBALS['THRIFT_ROOT'].'/Thrift.php' );
6 require_once( $GLOBALS['THRIFT_ROOT'].'/transport/TSocket.php' );
7 require_once( $GLOBALS['THRIFT_ROOT'].'/transport/TBufferedTransport.php' );
8 require_once( $GLOBALS['THRIFT_ROOT'].'/protocol/TBinaryProtocol.php' );
9 require_once( $GLOBALS['THRIFT_ROOT'].'/packages/Hbase/Hbase.php' );
10
11 //2. 初始化 Thrift 連線
12 $socket = new TSocket( 'ThriftServerIP', 9090 );
13 $socket->setSendTimeout( 10000 ); // Ten seconds (too long for production, but)
14 $socket->setRecvTimeout( 20000 ); // Twenty seconds
15 $transport = new TBufferedTransport( $socket );
16 $protocol = new TBinaryProtocol( $transport );
17 $client = new HbaseClient( $protocol );
18 $transport->open();
19
20 //3. 取得HBase內所有資料表
21 $tables = $client->getTableNames();
22 foreach ( $tables as $name ) {
23     echo $name . "\n";
24 }
25
26 //4. 刪除資料表
27 $name = "DEMO_TABLE_NAME";
28 if ( $client->isTableEnabled( $name ) ) {
29     $client->disableTable( $name );
30     $client->deleteTable( $name );
31 }
32
33 //5. 新增資料表
34 $columns = array(
35     new ColumnDescriptor( array(
36         'name' => 'family:',
37         'maxVersions' => 3
38     ) ),
39     new ColumnDescriptor( array(
40         'name' => 'family2:'
41     ) )
42 );
43
44 $t = "NEW_TABLE_NAME";
45 try {
46     $client->createTable( $t, $columns );
47 } catch ( AlreadyExists $e ) {
48     echo $e->message;
49 }
50
51 //6. 取得一個資料列
52 $table_name = "TableName";
53 $row_name = "RowKey";
54 $arr = $client->getRow( $table_name, $row_name );
55 foreach ( $arr as $k=>$TRowResult ) {
56     printTRowResult( $TRowResult );
57 }
```

圖 5. PHP 透過 Thrift 存取 HBase 程式碼範例

## 2.4 SQL 結構化查詢語言簡介

SQL (Structure Query Language)結構化查詢語言[19][20]是用於關聯式資料庫的標準資料查詢語言，其中包含的 DML(Data Manipulation Language)資料處理語言是 SQL 語言中負責存取資料庫資料的指令有四種指令:SELECT、INSERT、UPDATE、DELETE，分別代表查詢、新增、更新與刪除，與刪除，這些指令是典型的資料庫應用程式必定會使用到的指令語法。除了 INSERT 新增指令以外，其它指令都可以配合 WHERE 指令來篩選過濾特定的資料。

1. SELECT 是查詢資料的指令，以下是一個找出條件符合職業為工程師、男性、碩士畢業的範例：

```
SELECT * FROM member WHERE  job = 'engineer' AND gender = 'male'
AND education = 'Master'
```

2. INSERT 是將資料新增至資料庫的指令，以下是新增一個學生資料的範例：

```
INSERT INTO student (stu_id, name, dept) VALUES ('s99639102', '曾坤福', '資
工系')
```

3. UPDATE 是根據特定的篩選條件，將符合的資料記錄更新指定的欄位與資料值，下例為更新 score 欄位小於 60 的記錄，使其 score 欄位資料更新為 60：

```
UPDATE student_score SET score = 60 WHERE score < 60
```

4. DELETE 是根據特定的篩選條件，刪除符合的資料記錄，下例為刪除在學狀態為退學的學生資料：

```
DELETE FROM student WHERE status = '退學'
```

## 2.5 Luís Ferreira 提出之消弭 SQL 與 NoSQL 之間的間隙想法

Luís Ferreira 認為大部分的資料庫應用程式程式碼往往依賴所使用的特定關聯式資料庫系統，這造成當我們要從一個關聯式資料庫系統遷移(migrate)至 NoSQL 資料庫系統時，既有的程式碼會無法使用的問題，Luís Ferreira 的目標是試著去打造一個 SQL 程式碼與語法解譯器之間的中間層，以提供像是 Cassandra 這樣的 NoSQL 資料庫可以執行 SQL 指令，並且儘可能的縮減額外的效能負擔。不過這個目標並沒有被作者嘗試，作者認為在分散式環境之中使用 SQL 處理資料是新穎的方式；為了確保這個作法是可行的，作者使用 Apache DerbyDB 與 Apache Cassandra 驗證其想法，提出以關聯式資料庫作為 SQL 解譯器，NoSQL 資料庫則作為儲存資料用途。實際的系統實作應採取更好的方法，必須有更大的深度以及深思熟慮的設計。[21]

### 分析與討論：

Luís Ferreira 提出的構想指出關聯式資料庫系統與 NoSQL 資料庫的差異，並進一步指出若要進行資料遷移，現存之應用程式的程式碼將會無法使用，與本論文的研究動機不謀而合，可惜的是其提出的獨立運作之中間層，在實作上必須處理分散式架構及相關細節，不符合我們的需求，本研究希望中間機制是依附於應用程式本身，達到高可攜性以及負載平衡的效益。

## 2.6 John Roijackers 提出之 NoSQL query pattern

John Roijackers 提出一個 NoSQL 查詢範本，基於標準 SQL 加強的一套語法(如圖 6)，

提供參數傳遞的方式指定查詢語法中的變動值，並增加 NoSQL (...) 這樣的子句達到可同時查詢 SQL 與 NoSQL 資料庫的資料功能。[22]

```
1 SELECT
2   p.t AS type,
3   p.p AS price
4 FROM
5   NoSQL(
6     type: ?t, versions: (
7       1: (
8         price: ?p,
9         copies: ?c
10      )
11    )
12  ) AS p
13 WHERE
14   p.p < %4\$d
```

Listing A.3: Product query template for  $\mathcal{F}_i$  and  $Q_3$

```
1 SELECT
2   p.t AS type,
3   p.p AS price,
4   p.b AS booking_deadline
5 FROM
6   NoSQL(
7     type: ?t, location: ?l, versions: (
8       1: (
9         price: ?p,
10        copies: ?c,
11        booking_deadline: ?b
12      )
13    )
14  ) AS p
15 WHERE
16   p.p < %4\$d
```

Listing A.4: Product query template for  $\mathcal{F}_i$  and  $Q_4$

圖 6. John Roijackers 提出之 NoSQL query pattern 語法範本

分析與討論：

John Roijackers 想要一個通用的新式 SQL 語法，達成同時存取傳統關聯式資料庫以及各種 NoSQL 非關聯式資料庫，其提出的 NoSQL 查詢範本語法設想的十分嚴謹，其研究列舉出 32 種因應不同情形之語法範本，可惜的新式的查詢語法加入許多字彙，開發者必須額外學習此語法範本，既有的應用程式也必須全部修改為該語法來實作存取資料庫的程式，這是較為不利的一點，並不符合我們的需求。

## 2.7 陳韡提出之 Web Service 與 SQL-HBase mapping 模組方法

### Web Service 部分分析與討論

陳韡提出之 Web Service 係以 Java 程式撰寫，透過 ServerSocket 類別指定固定 port 號建立起 Socket 偵聽 Client 端程式傳遞之 SQL 指令再轉送至 sqlToHbase 元件做 SQL 語法翻譯與執行。[23]

透過 ServerSocket 類別指定固定 port 號建立起 Socket 偵聽將會限制同一時間連線的 Client 只能有一個，其他的 Client 連線請求必須等到第一個 Client 終止連線後才能收到回應，由於網站程式會在同一時間被大量使用者連線，因此這樣的 Client/Server 架構設計顯然是較為不佳的。

### SQL 指令剖析部分分析與討論：

根據其研究之附錄 5 程式碼，用來剖析拆解 SQL 指令的 preprocess 方法，無法處理語法有多個斷行的 SQL 指令。INSERT 指令部分僅能處理一次新增一筆資料，無法處理一次新增多筆資料的情形，例如以下 SQL：

```
INSERT INTO student (stu_id, name)
VALUES
('S1', '張三'),
('S2', '李四'),
('S3', '王五');
```

由於其研究提出的 SQL-HBase mapping 模組需在 Client 端另外建立該元件之物件實例，仍然需要進行較多的程式碼修改作業，因此不符合我們的需求。

### 第三章、系統設計與方法

由於網際網路的迅速發展，大型網站儲存的資料量暴增，拍賣網站每天數以兆計的瀏覽商品記錄、社群平台的文字訊息以及社群遊戲的使用者互動記錄等；這些企業為了解決資料庫大量存取的問題，紛紛改用非關聯式資料庫(NoSQL)技術如 HBase 資料庫以提昇擴充性及效能，當企業欲導入此類技術時，從典型的基於關聯式資料庫（例如 MySQL）的 PHP 應用程式如遷移至 HBase 時可能遭遇這些問題：

1. NoSQL 資料庫與傳統關聯式資料庫設計觀念不同，必須訓練工程技術人員接受新觀念與方法，一旦遷移至 NoSQL 資料庫，應用程式與資料庫都必須修改，要說服資深技術人員重新學習是一個難題。
2. NoSQL 資料庫提供的 API 應用程式介面不夠多元，以 HBase 而言僅提供 Java 原生 API 支援，其它程式語言必須再另外尋覓像是 Thrift 函式庫整合方能使用。Thrift 也有其自訂的 API 應用程式介面，要使用其 API 來存取 HBase 必須要改寫程式碼，此外 Thrift 的安裝佈署也是一個需要考慮的問題。
3. 由於 NoSQL 資料庫不支援結構化查詢語言(SQL)存取資料，對於既有的關聯式資料庫應用程式而言，存取資料庫部分的原始程式碼必須重新撰寫，對目前已上線運作的應用程式產品的風險影響是比較高的。



### 3.1 系統架構

有鑑於上述數項問題，本研究使用 PHP 程式語言以及 Apache Thrift Client，提出一個能使用基於 SQL 指令語法的 PHP 應用程式程式碼之中介橋接機制 MH bridge，以期達成保留 SQL 指令程式碼的應用程式存取 HBase 分散式資料庫。本論文具有下列特色：

- 在程式語言的選擇上，採用與應用程式相同的 PHP 語言實作本研究提出之機制，因此 Web Server 網站伺服器不用安裝 Java 等其它語言以減少網站伺服器環境需要多種程式語言的相依性。本機制相依於 Web Server 上亦可容易達成水平擴充(horizontal scalability)特性。
- 透過 PHP 的 namespace 命名空間特性的使用，可對同名函式重載(Overloading)，進而重新定義存取 MySQL 資料庫的原生系列函式 mysql\_\* 的行為，以達成不用移除既有應用程式程式碼之中的存取 mysql 相關函式。
- 使用 SQL 語法剖析器分解應用程式存取 MySQL 資料庫的 SQL 指令，MH bridge 機制便會依據 SQL 的種類是新增、查詢、修改、刪除來決定呼叫對應的 Thrift Client API 與 HBase 資料庫存取資料。
- 本研究的實作程式碼使用 Git 版本控制系統管理程式碼，由於軟體開發通常無法在極短的時間內完成，必須經歷漫長的撰寫、除錯，再到完成的後續維護與功能特色增減，然而在軟體開發過程中難免會發生像是程式碼或檔案的誤刪、錯誤的操作造成舊版本程式碼覆蓋新的版本等等，在多人團隊共同合作開發時，狀況變得更加複雜。透過版本控制系統的使用讓本研究的程式碼管理品質穩定許多。
- 本研究使用 Github 服務平台放置實作程式之 Git 版本檔案庫，如此可提昇專案的能見度，讓對本論文相關議題有興趣的開發者容易取得本研究的實作程



式碼。

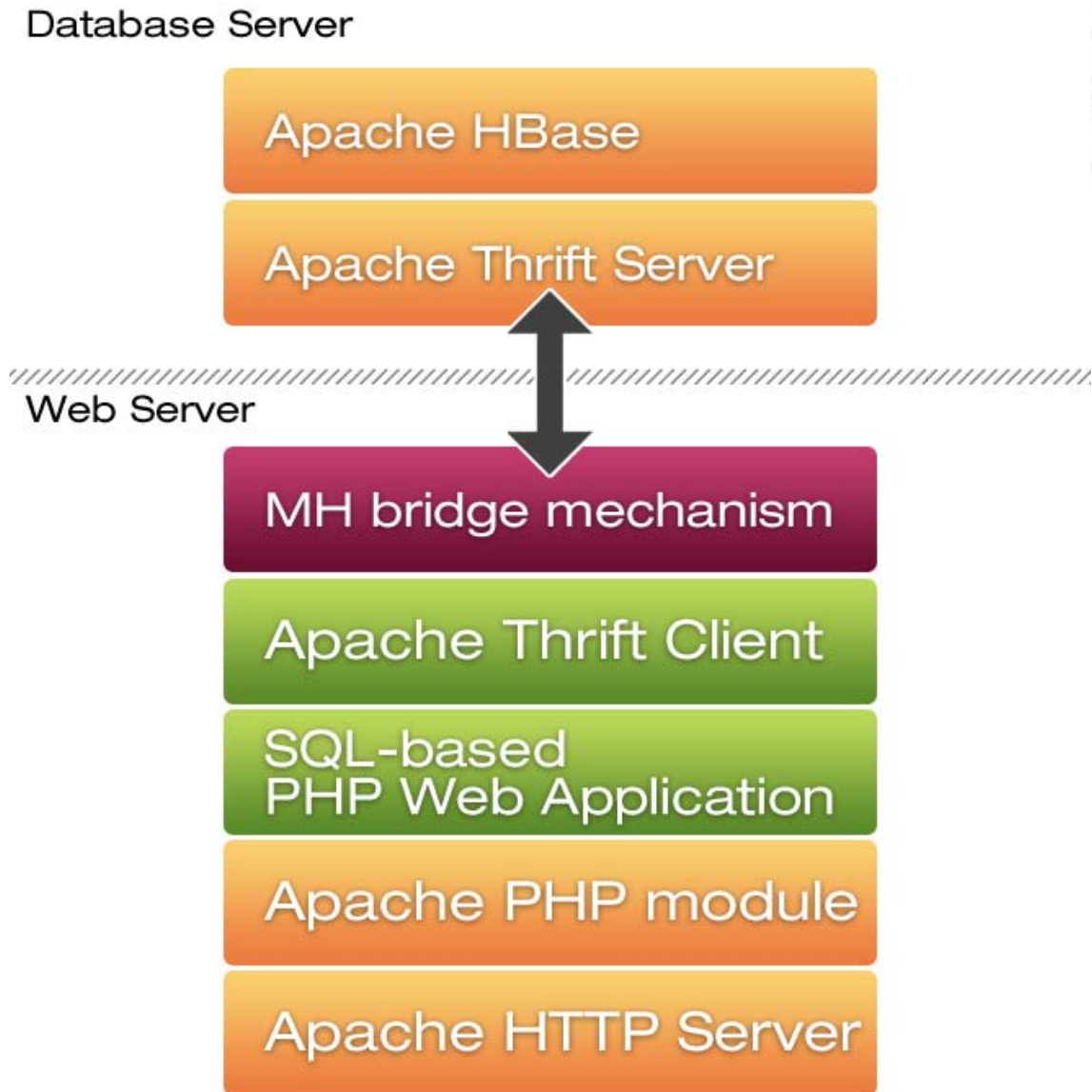


圖 7. 本研究提出之中介橋接機制 MH bridge 系統架構圖

為達成上述特色，本論文提出之中介橋接機制系統架構如圖 7 所示，包含兩大部分：

1. 資料庫伺服器端：我們佈署 Apache Thrift Server 以供給網站伺服器端的 Apache Thrift Client 連線傳輸資料以及要遷移的目標：Apache HBase 分散式資料庫。

2. 網站伺服器端：我們採用最普及的 Apache HTTP Server 作為網站伺服器，配合 PHP 模組的啟用，可將 PHP 程式執行於 Apache HTTP Server。

SQL-based PHP Web Application 為基於關聯式資料庫的應用程式，應用程式使用 SQL 對 MySQL 資料庫新增、查詢、修改、刪除資料等等。我們在應用程式程式碼載入本論文提出之中介橋接機制 MH bridge 程式，藉由 `mysql_query` 等函式的重載，SQL 指令將會被 MH bridge 機制攔截，分析語法後呼叫對應功能的 Apache Thrift Client API，進而達到不用大幅修改現有應用程式程式碼之目的。




圖 8. MH bridge 中介橋接機制架構圖

圖 8 為本論文提出之 MH bridge 中介橋接機制系統架構圖。我們將透過此架構圖詳細說明每個元件的功能。

### 攔截函式

為重載 PHP 語言提供的 mysql 系列函式，我們需要一系列的同名函式搭配命名空間特性來攔截預設的 mysql 系列函式，例如 圖 9、圖 10





```

namespace MHBridge;

/**
 * mysql_connect
 *
 * @param string $result Optional
 * @param string $field_offset Optional
 * @param string $password Optional
 * @param string $new_link Optional
 * @param string $client_flags Optional
 * @return resource
 */
function mysql_connect(
    $server = "localhost",
    $username = NULL,
    $password = NULL,
    $new_link = false,
    $client_flags = 0) {

    $params = array(
        "server"      => $server,
        "username"    => $username,
        "password"    => $password,
        "new_link"    => $new_link,
        "client_flags" => $client_flags
    );

    return MMB::connect($params);
}

```

圖 9. 攔截 mysql\_connect 系統函式之同名函式



```
namespace MHBridge;

/**
 * mysql_connect
 *
 * @param resource $result
 * @param int $field_offset
 * @return string
 */
function mysql_field_name(&$result, $field_offset) {
    $row = $result[0];
    $offset = 0;

    foreach ($row as $col_name => $v) {
        if ($field_offset == $offset) {
            return $col_name;
        }
        $offset++;
    }
}
```

圖 10. 攔截 mysql\_field\_name 系統函式之同名函式

### 3.2 運作流程

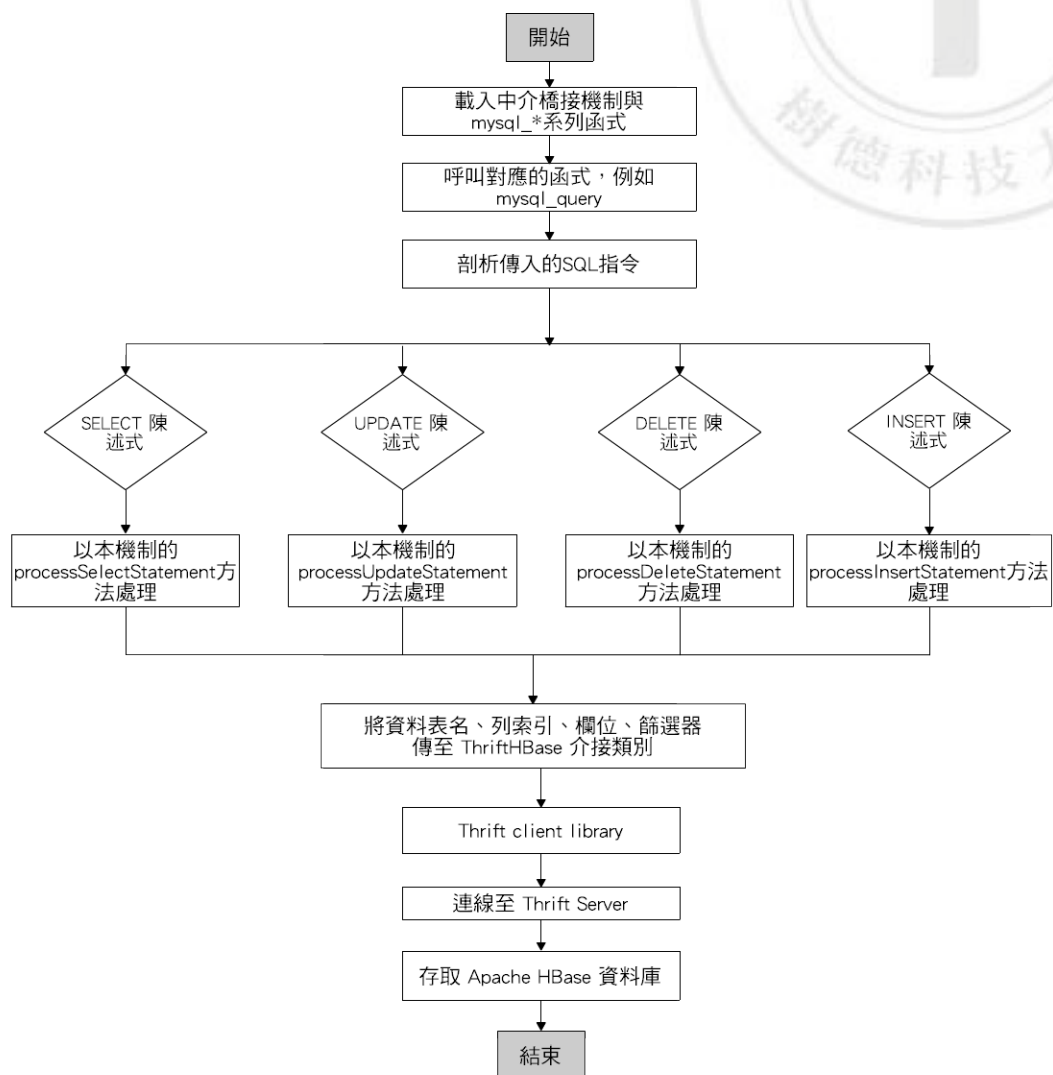


圖 11. 本研究之中介橋接機制運作流程

圖 11 為本研究之中介橋接機制運作流程，以下說明流程與其細節。

1. 載入中介橋接機制與本機至提供之 mysql\_\* 系列函式
2. 呼叫對應的函式，例如原有的應用程式呼叫 mysql\_query 函式時，本機制提供之 mysql\_query 函式會被叫用，取而代之內建的 mysql\_query 函式將不會作用，因此不會連到 MySQL 資料庫。
3. mysql\_query 函式接收一個字串參數，其為 SQL 指令語法，我們需要對這個 SQL 指令進行語法剖析，以便判斷出是何種指令陳述句型，資料處理的 SQL 分為四種：SELECT 查詢陳述式、UPDATE 更新陳述式、DELETE 刪除陳述式、INSERT 新增陳述式。以下將針對每種陳述式逐一介紹運作流程。

### SELECT 查詢陳述式

對於 SELECT 陳述式之處理，如圖 12. 對於 SELECT 陳述式之處理流程圖，我們需要剖析取出的部份有「欄位清單」、「資料表名稱」、「WHERE 條件陳述式組」、「LIMIT 數量限制」。

以下為一個 SELECT 陳述式語法範例：

```
SELECT * FROM student WHERE age = 24 LIMIT 50
```

上述範例中，「\*」為欄位清單，表示要取出全部欄位；「student」為資料表名稱、「WHERE age =24」為條件陳述式，表示篩選出 age 欄位資料為 24 的紀錄；「LIMIT 50」為數量限制，表示最多取出 50 筆記錄。

若 SELECT 陳述式之中有指定欄位清單，則套用 MultipleColumnPrefixFilter。

若 SELECT 陳述式之中有 WHERE 條件陳述式，則逐一取出每一組條件式，條件式是由欄位、運算子、值三者構成，接著根據欄位名稱若為主鍵名稱則套用

RowFilter，反之皆套用 SingleColumnValueFilter。

若 SELECT 陳述式之中 LIMIT 數量限制語法，則加入 PageFilter 對應。

下一步將資料表名、列索引、欄位組、篩選器交由本機制中的 ThriftHBase 介接類別後，便由 Thrift client library 之 scannerOpenWithScan 方法連線至 Apache Thrift Server，最後取回 Apache HBase 資料庫中的資料。

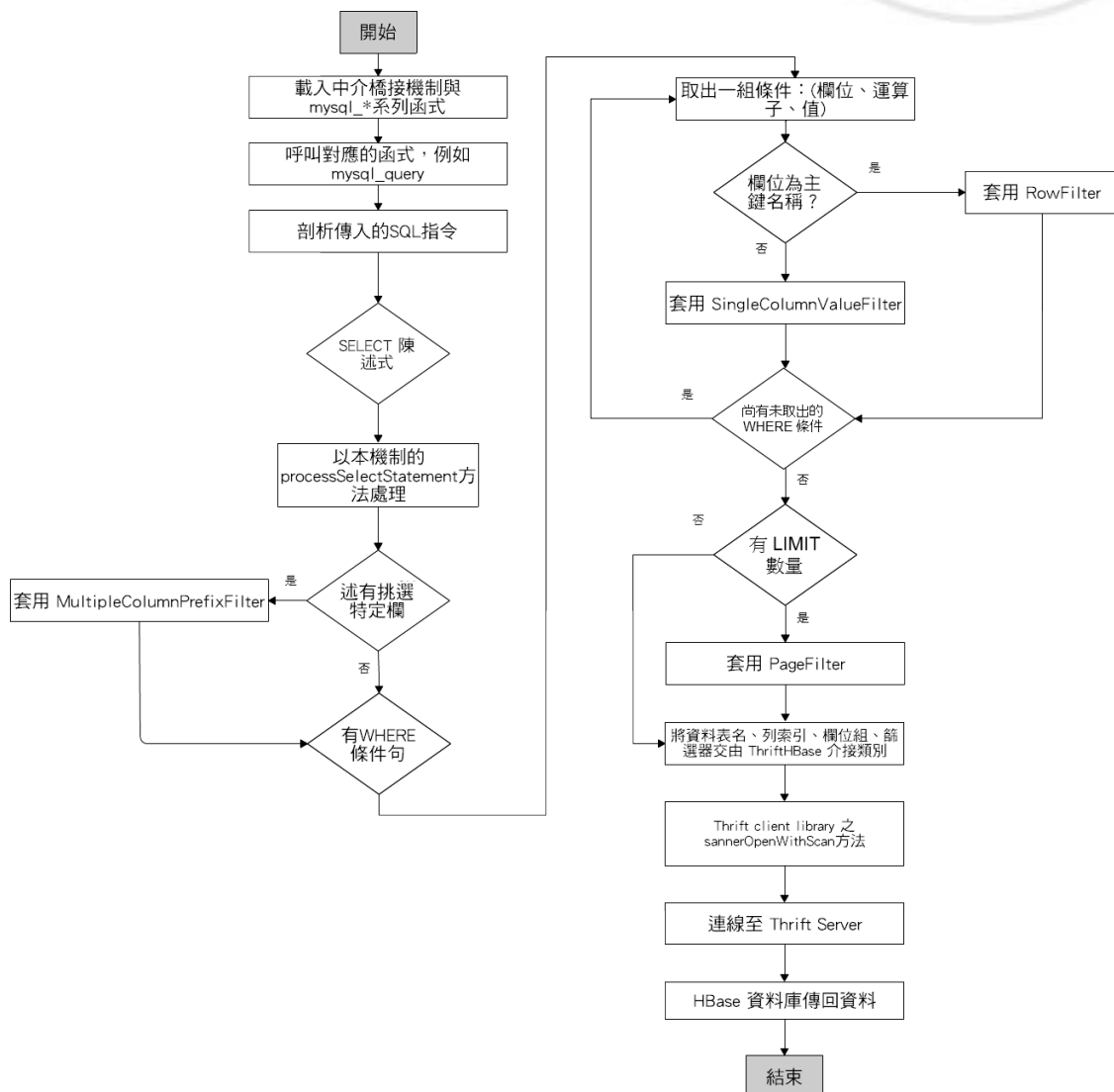


圖 12. 對於 SELECT 陳述式之處理流程圖

## UPDATE 更新陳述式



對於 UPDATE 陳述式之處理，如圖 13，我們需要剖析取出的部份有「要更新的欄位與值清單」、「資料表名稱」、「WHERE 條件陳述式組」、「LIMIT 數量限制」。

以下為一個 UPDATE 陳述式語法範例：

```
UPDATE score SET gpa = 60 WHERE age < 60 LIMIT 10
```

上述範例中，「gpa = 60」要更新的欄位與值清單，表示要更新 gpa 欄位為 60；「score」為資料表名稱、「WHERE age < 60」為條件陳述式，表示篩選出 age 欄位資料小於 60 的紀錄；「LIMIT 10」為數量限制，表示最多異動 10 筆記錄。

對於 UPDATE 陳述式，本研究的作法為將 SQL 指令的 UPDATE 代換為 SELECT，使其成為一個 SELECT 查詢陳述式，接著交由本機制之 ThriftHBase 介接類別的 insert 方法處理，insert 方法會叫用 Apache Thrift client library 的 mutateRow 方法。將欲更新的資料記錄列索引與新的資料值傳至 Thrift Server 端，最後更新 HBase 資料庫中的資料。

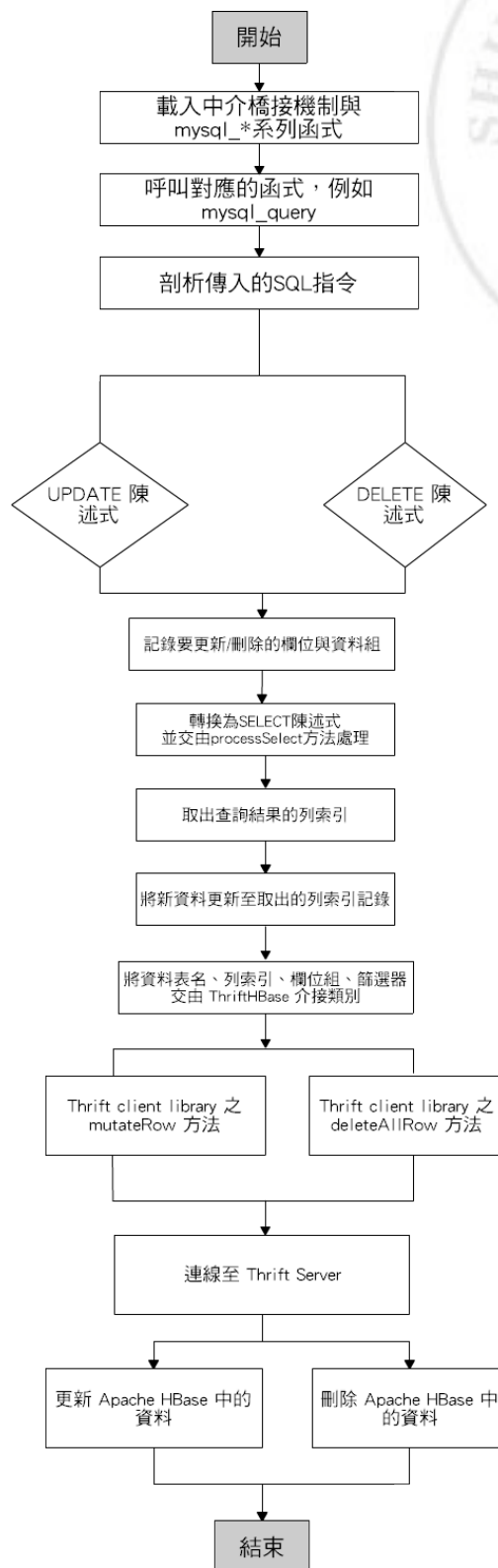


圖 13. 對於 UPDATE 與 DELETE 陳述式之處理流程圖



## DELETE 刪除陳述式

至於 DELETE 陳述式，如圖 13，我們需要剖析取出的部份有「資料表名稱」、「WHERE 條件陳述式組」、「LIMIT 數量限制」。

以下為一個 DELETE 陳述式語法範例：

```
DELETE FROM student WHERE status = '退學' LIMIT 100
```

上述範例中，「WHERE status = '退學」為條件陳述式，表示篩選出 status 欄位資料為“退學”字串的紀錄；「LIMIT 100」為數量限制，表示最多刪除 100 筆記錄。對於 DELETE 陳述式，本研究的作法為將 SQL 指令的 DELETE 代換為 SELECT，使其成為一個 SELECT 查詢陳述式，接著交由本機制之 ThriftHBase 介接類別的 deleteAllRow 方法處理，deleteAllRow 方法會叫用 Apache Thrift client library 的 deleteAllRow 方法。將欲刪除的資料記錄列索引傳至 Thrift Server 端，最後刪除 HBase 資料庫中的資料。

## INSERT 新增陳述式

最後的 INSERT 陳述式，我們需要剖析取出的部份有「欄位清單」、「資料表名稱」、「值清單」。

以下為一個 INSERT 陳述式語法範例：

```
INSERT INTO student (id, first_name, last_name, age, email)  
VALUES ('s99639102', '坤福', '曾', 24, 's99639102@stu.edu.tw')
```

上述範例中，「student」為資料表名稱；「id, first\_name, last\_name, age, email」為欄位清單，表示要新增這五個欄位；「's99639102', '坤福', '曾', 24, 's99639102@stu.edu.tw'」為資料值的清單，表示五個欄位分別對應的值。

對於 INSERT 陳述式，本研究的作法為記錄要新增的欄位與資料組後，交由本機制之 ThriftHBase 介接類別的 insert 方法處理，insert 方法會叫用 Apache Thrift client library 的 mutateRow 方法。將資料傳至 Thrift Server 端，最後新增資料至 HBase。

### 3.3 從 MySQL 資料遷移至 HBase

我們使用 Apache Sqoop 來做 MySQL 資料遷移至 HBase 的工作，輸入指令範例如下，參數用途詳見表 7 [24]，匯入過程如圖 14：

```
sqoop import --connect jdbc:mysql://MySQL 資料庫位置/資料庫名稱 --table 匯入資料表名稱 --hbase-table 匯入資料表名稱 --hbase-create-table --column-family mysql --username root -P
```

```
12/08/22 22:11:50 WARN client.ZooKeeperSaslClient: SecurityException: java.lang.SecurityException: Unable to locate a login configuration occurred when trying to find JAAS configuration.
12/08/22 22:11:50 INFO client.ZooKeeperSaslClient: Client will not SASL-authenticate because the default JAAS configuration section 'Client' could not be found. If you are not using SASL, you may ignore this. On the other hand, if you expected SASL to work, please fix your JAAS configuration.
12/08/22 22:11:50 INFO zookeeper.ClientCnxn: Socket connection established to localhost/127.0.0.1:2181, initiating session
12/08/22 22:11:50 INFO zookeeper.ClientCnxn: Session establishment complete on server localhost/127.0.0.1:2181, sessionId = 0x1394dc4e8dd0012, negotiated timeout = 40000
12/08/22 22:11:50 WARN conf.Configuration: hadoop.native.lib is deprecated. Instead, use io.native.lib.available
12/08/22 22:11:50 INFO zookeeper.ZooKeeper: Initiating client connection, connectString=localhost:2181 sessionTimeout=180000 watcher=CatalogTracker-on-org.apache.hadoop.hbase.client.HConnectionManager$HConnectionImplementation@4f62198b
12/08/22 22:11:50 INFO zookeeper.ClientCnxn: Opening socket connection to server /127.0.0.1:2181
12/08/22 22:11:50 WARN client.ZooKeeperSaslClient: SecurityException: java.lang.SecurityException: Unable to locate a login configuration occurred when trying to find JAAS configuration.
12/08/22 22:11:50 INFO client.ZooKeeperSaslClient: Client will not SASL-authenticate because the default JAAS configuration section 'Client' could not be found. If you are not using SASL, you may ignore this. On the other hand, if you expected SASL to work, please fix your JAAS configuration.
12/08/22 22:11:50 INFO zookeeper.ClientCnxn: Socket connection established to localhost/127.0.0.1:2181, initiating session
12/08/22 22:11:50 INFO zookeeper.RecoverableZooKeeper: The identifier of this process is 10502@carglecloud0
12/08/22 22:11:50 INFO zookeeper.ClientCnxn: Session establishment complete on server localhost/127.0.0.1:2181, sessionId = 0x1394dc4e8dd0013, negotiated timeout = 40000
12/08/22 22:11:50 INFO zookeeper.ZooKeeper: Session: 0x1394dc4e8dd0013 closed
12/08/22 22:11:50 INFO mapreduce.HBaseImportJob: Creating missing HBase table wp_term_relationships
12/08/22 22:11:50 INFO zookeeper.ClientCnxn: EventThread shut down
12/08/22 22:11:54 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN('object_id'), MAX('object_id') FROM 'wp_term_relationships'
12/08/22 22:11:54 INFO mapreduce.JobSubmitter: number of splits:4
12/08/22 22:11:54 WARN conf.Configuration: mapred.job.classpath.files is deprecated. Instead, use mapreduce.job.classpath.files
12/08/22 22:11:54 WARN conf.Configuration: mapred.cache.files is deprecated. Instead, use mapreduce.job.cache.files
12/08/22 22:11:54 WARN conf.Configuration: mapred.job.name is deprecated. Instead, use mapreduce.job.name
12/08/22 22:11:54 WARN conf.Configuration: mapred.cache.files.timestamps is deprecated. Instead, use mapreduce.job.cache.files.timestamps
12/08/22 22:11:54 WARN conf.Configuration: mapred.working.dir is deprecated. Instead, use mapreduce.job.working.dir
12/08/22 22:11:54 INFO mapred.ResourceMgrDelegate: Submitted application application_1345629503770_0001 to ResourceManager at /0.0.0.0:8032
12/08/22 22:11:54 INFO mapreduce.Job: The url to track the job: http://carglecloud0:8088/proxy/application_1345629503770_0001/
12/08/22 22:11:54 INFO mapreduce.Job: Running job: job_1345629503770_0001
12/08/22 22:12:01 INFO mapreduce.Job: Job job_1345629503770_0001 running in uber mode : false
12/08/22 22:12:01 INFO mapreduce.Job: map 0% reduce 0%
```

圖 14. 使用 Sqoop 匯入資料至 HBase 過程

表 7. Sqoop 匯入資料至 HBase 之指令參數表

參數	功能
<b>--connect</b>	JDBC 連線字串
<b>--column-family &lt;family&gt;</b>	要把匯入的全部欄位放入哪個欄位家族名稱
<b>--hbase-create-table</b>	將自動建立不存在的資料表名稱
<b>--hbase-row-key &lt;col&gt;</b>	指定要用哪個欄位作為 row key, 預設值為來源資料表之主鍵欄位
<b>--hbase-table &lt;table-name&gt;</b>	指定匯入 HBase 後的資料表名稱
<b>--table</b>	匯入指定的 MySQL 資料表名稱
<b>--username</b>	用來連線的資料庫使用者
<b>-P</b>	將從命令列等待使用者輸入密碼
<b>--password</b>	用來連線的資料庫使用者密碼

我們以匯入 MySQL 資料表 wp\_links 為例，匯入完成後，可從 **hbase shell** 進入 HBase 命令列互動操作介面以 list 指令檢查是否匯入成功，如圖 15：

```
$ hbase shell
```

```
hbase(main):001:0> list
```

```
afu at carglecloud0 in ~/public_html/import-hbase-sqoop
$ hbase shell
12/08/24 03:19:48 WARN conf.Configuration: hadoop.native.lib is deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.92.1-cdh4.0.1, rUnknown, Thu Jun 28 18:13:01 PDT 2012

hbase(main):001:0> list
TABLE
tbl1
tbl2
tbl3
tbl4
tbl5
test
test2
test_student
user browser log
wp_links
wp_term_relationships
11 row(s) in 0.5020 seconds
```

圖 15. 在 hbase shell 環境輸入 list 檢查是否成功匯入資料表

於 hbase shell 觀看匯入的資料，如圖 16.：

hbase(main):001:0> scan 'wp\_links', {FILTER => "( PageFilter (1))"}

```
afu at carglecloud0 in ~
$ hbase shell
12/08/24 03:57:14 WARN conf.Configuration: hadoop.native.lib is deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.92.1-cdh4.0.1, rUnknown, Thu Jun 28 18:13:01 PDT 2012

hbase(main):001:0> scan 'wp_links', {FILTER => "( PageFilter (1))"}
ROW COLUMN+CELL
1 column=mysql:link_description, timestamp=1345749552720, value=
1 column=mysql:link_image, timestamp=1345749552720, value=
1 column=mysql:link_name, timestamp=1345749552720, value=Documentation
1 column=mysql:link_notes, timestamp=1345749552720, value=
1 column=mysql:link_owner, timestamp=1345749552720, value=1
1 column=mysql:link_rating, timestamp=1345749552720, value=0
1 column=mysql:link_rel, timestamp=1345749552720, value=
1 column=mysql:link_rss, timestamp=1345749552720, value=
1 column=mysql:link_target, timestamp=1345749552720, value=
1 column=mysql:link_url, timestamp=1345749552720, value=http://codex.wordpress.org/
1 column=mysql:link_visible, timestamp=1345749552720, value=Y
1 row(s) in 0.5310 seconds
```

圖 16. 於 hbase shell 觀看匯入後的資料表



### 3.4 SQL 指令剖析與 Thrift HBase API 映射

為了將 SQL 指令對應至功能相呼應的 API，我們首先必須從應用程式發送的 SQL 指令分析出是屬於何種類型的 SQL，再進一步去擷取每種類型的 SQL 所隱含的欄位、資料、條件等，以下以 SELECT 為例說明 SQL 的分析過程。

#### SELECT

我們使用以下的範例 SQL 說明 SELECT：

```
SELECT address, school FROM director
```

```
WHERE address LIKE '%高雄%' AND school LIKE '%科技大學' LIMIT 200
```

此 SQL 的目的是從 director 資料表挑選出符合 address 欄位含有高雄字串、school 欄位以科技大學結尾的資料記錄，最多取 200 筆，因此我們需要擷取以下五項資訊：

1. address, school
2. director
3. address LIKE '%高雄%' AND school LIKE '%科技大學'
4. LIMIT 200
5. director

接著我們根據表 8. 選出我們要使用的對應 API 為「scannerOpenWithScan 結合 Filter 匹配符合條件的紀錄」，由於 HBase 的 Filter 語法與 WHERE 條件子句完全不同，因此我們再根據表 9. 對每一個條件句對應適當的 Filter：

address LIKE '%高雄%' 是指 address 欄位包含「高雄」的任意字串，對應的 HBase Filter 為

```
SingleColumnValueFilter ('mysql', 'address', =, 'regexstring:高雄', true, false)
```

school LIKE '%科技大學'是指 school 欄位以「科技大學」為字尾的字串，對應的 HBase Filter 為

```
SingleColumnValueFilter ('mysql', 'school', =, 'regexstring:科技大學$', true, false)
```

兩個子條件句之間的關係為 AND，在 HBase Filter 部分直接使用 AND 串連上述兩個 Filter，這裡用 Filter1 與 Filter2 代表以節省篇幅：

```
Filter1 AND Filter2
```

挑選特定欄位 address 及 school 對應的 HBase Filter 為：

```
MultipleColumnPrefixFilter ('address','school')
```

限制數量的 LIMIT 子句，對應的 HBase Filter 為：

#### PageFilter (200)

最後我們把用到的 Filter 彼此間用 AND 串連，成為最終的 Filter 篩選器字串：

```
MultipleColumnPrefixFilter ('address','school') AND SingleColumnValueFilter ('mysql',  
'address', '=', 'regexstring:高雄', true, false) AND SingleColumnValueFilter ('mysql',  
'school', '=', 'regexstring:科技大學$', true, false) AND PageFilter (200)
```

建立帶有 Filter 的 Scan 實例：

```
$params["filterString"] = 篩選器字串;  
$scan = new \TScan($params);
```

建立 Scanner 實例：

```
$scanner = scannerOpenWithScan("director", $scan);
```

取回 Scanner 掃描到的資料紀錄

```
while ($row = scannerGet($scanner)) {  
    // .....  
}
```

表 8. SQL 與 Thrift HBase API 對應表

SQL	Thrift HBase API
<b>SELECT</b>	scannerOpenWithScan 結合 Filter 匹配符合條件的紀錄 scannerGet 取回 scanner 掃描到的資料
<b>UPDATE</b>	1. scannerOpenWithScan 結合 Filter 匹配符合條件的紀錄 2. scannerGet 取回 scanner 掃描到的資料 3. 將需要寫入的多個欄位及欄位值建立成多個 Mutation 實例 4. 將 row key 與 Mutation 實例給予 mutateRow 方法更新資料
<b>DELETE</b>	1. scannerOpenWithScan 結合 Filter 匹配符合條件的紀錄 2. scannerGet 取回 scanner 掃描到的資料 3. deleteAllRow 刪除匹配到的紀錄
<b>INSERT</b>	1. 將需要寫入的多個欄位建立成多個 Mutation 實例 2. 將 row key 與 Mutation 實例給予 mutateRow 方法新增資料



表 9. SQL 與 HBase Filter 對應表

SQL	對應之 HBase 篩選器 Filter
<b>SELECT [...]</b> FROM (挑選特定欄位)	MultipleColumnPrefixFilter ('欄位 1','欄位 2' ...)
<b>WHERE</b> 主鍵欄位 [運算子] 值	RowFilter (運算子, 'binary:值')
<b>WHERE</b> 一般欄位 [運算子] '%關鍵字%'	SingleColumnValueFilter ('欄位家族', '欄位', '=', 'regexstring:關鍵字', true, false)
<b>WHERE</b> 一般欄位 [運算子] '%關鍵字'	SingleColumnValueFilter ('欄位家族', '欄位', '=', 'regexstring:關鍵字\$', true, false)
<b>WHERE</b> 一般欄位 [運算子] '關鍵字%'	SingleColumnValueFilter ('欄位家族', '欄位', '=', 'regexstring:^關鍵字', true, false)
<b>LIMIT</b> 數量	PageFilter (數量)

## 第四章、系統實作與實驗設計

### 4.1 實驗環境規格配置

本實驗架構一台雙核心 PC 電腦，實體機器硬體規格如表 10。作業系統採用 Ubuntu 12.04 LTS Server 64-bit，Ubuntu 是一個開放原始碼的自由軟體作業系統，使用 Ubuntu 的好處是安裝容易、操作簡便、易於管理。

表 10. 系統實作環境硬體規格

名稱	規格
作業系統	Ubuntu 12.04 LTS Server 64-bit
CPU	Intel(R) Core(TM) i5 CPU 760 2.80GHz
記憶體	4 GB
硬碟容量	1 TB

本研究的實作環境軟體與開發工具皆採用開放原始碼之自由軟體，由於本研究探討的應用程式為網頁應用程式，因此需要網站伺服器方可運作，我們選用歷史悠久穩定的 Apache HTTP Server，程式語言採用最新版本的 PHP 5.4 系列，5.4 版本的 PHP 安全性與效能皆比 5.2、5.3 版本優越。Apache Hadoop 系列軟體我們架設 Apache Hadoop 分散式運算檔案系統平台、Apache HBase 分散式資料庫、Apache Sqoop 用來將關聯式資料庫之資料轉換遷移至 HBase，Apache ZooKeeper 提供分散式應用程式協調服務，可稱作是 Hadoop 動物園家族的管理員。Apache Thrift 是一個異質程式語言資料交換之軟體框架，可讓不同的程式語言互相溝通，例如以 PHP 應用程式存取 HBase (Java 語言)。Twitter Bootstrap 是一套新穎的網頁前端設計框

架，可快速打造具有不錯質感的網頁介面。Vim 是歷久彌新的文字編輯器，幾乎可以拿來撰寫開發任何程式語言，其強大的操作模式讓熟練的程式設計師提高生產力。Git 是一個分散式版本控制系統，用來管理本研究實作的程式碼。

表 11 系統實作環境軟體與開發工具

名稱	功能	版本
<b>Apache HTTP Server</b>	網站伺服器	2.2.22
<b>Apache Hadoop</b>	分散式運算平台	2.0.0+91 (CDH4.0.1)
<b>Apache HBase</b>	分散式資料庫	0.92.1+67 (CDH4.0.1)
<b>Apache Sqoop</b>	將關聯式資料庫之資料轉換 遷移至 HBase	1.4.1+28 (CDH4.0.1)
<b>Apache Thrift</b>	異質程式語言資料交換之軟體 體框架	0.8.0
<b>Apache ZooKeeper</b>	提供分散式應用程式協調服務	3.4.3+15 (CDH4.0.1)
<b>Git</b>	分散式版本控制系統，管理 本研究實作的程式碼	1.7.9
<b>PHP</b>	程式語言，開發本研究實作 之程式	5.4.4
<b>MySQL</b>	資料庫伺服器，資料遷移的 來源	Ver 14.14 Distrib 5.5.24
<b>Twitter Bootstrap</b>	網頁前端設計框架	2.1.0
<b>Vim</b>	歷久彌新的文字編輯器，用 來開發程式	7.3

## 4.2 系統實作

SQL 指令語法剖析部分以 PHP 的 `preg_match_all` 函式配合正規表達式(Regular Expression)分析萃取 SQL 指令中我們所需要的資料，以下以本機制中的 `processSelectStatement` 方法為例，說明如何處理 SELECT 系列的 SQL 指令。

首先介紹最簡單的 SELECT 句型，沒有 WHERE 條件與 LIMIT 子句：

語法：SELECT <select list> FROM <table reference>

對於這個句型，使用以下的正規表達式範本處理：

```
“/SELECT ([\w,\*\`s]+) FROM\s+([\w`]+)/i”
```

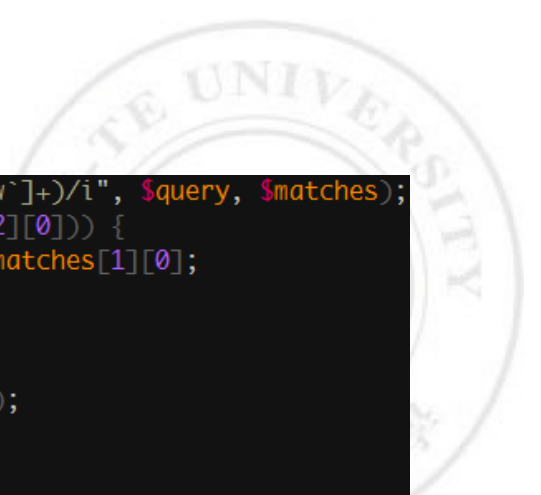
SELECT：語法以 SELECT 開頭，接著是<select list>欄位清單

`([\w,\*\`s]+)`：

欄位清單格式為 欄位 1, 欄位 2, 欄位 N，MySQL 的欄位名稱可以使用 ` 符號夾住，因此先列出欄位清單可能出現的字元規則為：英文字元、數字、底線、空白字元、星號、` 符號，得出 `[\w,\*\`s]`，欄位清單為兩個字元以上組合成的字串，因此加上 + 號表示有多個該規則的字元：`[\w,\*\`s]+`，最後以一對圓括號夾住以便讓 PHP 的 `preg_match_all` 函式取回獨立的資料結果：`([\w,\*\`s]+)`。FROM 與資料表名稱之間可能有多個空白字元，得出 `FROM\s+`

資料表名稱可能出現的字元規則為：英文字元、數字、底線、` 符號，得出 `([\w`]+)`。

實作程式碼輪廓如圖 17，詳細程式碼請參閱附錄：



```
preg_match_all("/SELECT ([\w,\\*`\\s]+) FROM\s+([\w`]+)/i", $query, $matches);
if (! empty($matches[1][0]) && ! empty($matches[2][0])) {
    $column = ($matches[1][0] == "**") ? null : $matches[1][0];
    $column = str_replace("`", "", $column);
    $column = str_replace(" ", "", $column);

    $table = str_replace("`", "", $matches[2][0]);
    $rowkey = "**";

    $filter = "";

    // SQL 有挑選特定欄位就加入 MultipleColumnPrefixFilter
    if ($column != "**" && ! empty($column)) {
        // Convert Col1,Col2 => 'Col1','Col2'
        $column = "'" . str_replace(",", "','", $column) . "'";
        $filter .= "MultipleColumnPrefixFilter ($column) AND ";
    }
}
```

圖 17. 處理 SELECT <select list> FROM <table refemce> 句型實作程式碼範例

第二部分說明含有 WHERE 條件句的句型，語法：

**SELECT <select list> FROM <table reference> WHERE <search condition>**

SELECT...FROM 部分在第一部分已說明，此處從 WHERE 條件句詳述，對於此句型，使用以下正規表達式範本處理：

**"/SELECT ([\w,\\\*`\\s]+) FROM\s+([\w`]+)\s+(WHERE ([\w\W,\\s<=>\"\\-:]+)\*)/i"**

由於 WHERE 條件可能由多個敘述組合而成，本機制透過 parseMultiPredicate 方法解析出一至多個敘述，一個敘述由欄位名稱、運算子、資料值組成，若有多個敘述則彼此之間會有 AND 或 OR 邏輯運算子；欄位名稱由英文字元等組成，後方有一或多個空白字元，以 `([\w`]+)\s+` 處理；運算子包含 <、=、>、LIKE，後方有一或多個空白字元，以 `([<=>LIKE]+)\s+` 處理；資料值可能包含英文字元、數字、底線、單引號、百分比號、以及中文字，後方有零或多個空白字元，以

`([\x{4e00}-\x{9fa5}A-Za-z0-9_'\%]+)\s*` 處理，其中 `\x{4e00}-\x{9fa5}` 表示 PHP 中

UTF-8 編碼下的中文字範圍。最後的 AND 或 OR 邏輯運算子以 `([ANDORandor]+)?`

應對，其中的?符號代表前方字元可有可無。ParseMultiPredicate 方法實作程式碼如圖 18。

```
public static function parseMultiPredicate($expression) {
    preg_match_all("/^s*([W]+)\s+([<=>LIKE])\s+([\x{4e00}-\x{9fa5}A-Za-z0-9_'\%]+\s*([ANDORandor]+)?\s+/ui", $expression, $matches);

    if (isset($matches[0][0])) {
        $predicate_max_cnt = count($matches[1]);
        $idx = 0;
        $predicates = array();

        // 開始蒐集 predicate，最多 $predicate_max_cnt 句
        for ($idx = 0; $idx < $predicate_max_cnt; ++$idx) {
            $predicates[$idx] = "";

            // v1 . op . v2
            for ($i = 1; $i <= 3; ++$i) {
                if (!isset($matches[$i][$idx])) {
                    break;
                }
                $predicates[$idx] .= $matches[$i][$idx] . " ";
            }

            // 砍掉不具備兩個運算元與一個運算子的predicate
            if ($i != 4) {
                unset($predicates[$idx]);
            }
        }
    }

    self::$predicates = $predicates;
    // collect operators if has any operators
    if (isset($matches[4])) {
        $matches[4] = array_map("strtoupper", $matches[4]);
        self::$predicates_operators = array_filter($matches[4]);
    }

    return $predicates;
}
```

圖 18. SELECT <select list> FROM <table reference> WHERE <search condition>句型之實作程式碼

前面提到一個敘述由欄位名稱、運算子、資料值組成，其中的運算子可能包含<、=、>、LIKE，因此我們以 getMappingPredicate 方法來處理單一敘述句種的各種情形：LIKE 句型、NOT LIKE 句型、一般比對(大於、等於、小於)句型，分別使用這三種正規表達式判別："/ NOT LIKE /i"、"/ LIKE /i"、"/ [<|=|>]+ /i"，如圖 19，再給予 LikePredicate 或是 ComparisonPredicate 類別處理對應之句型。

最後以 LikePredicate 為例說明對 LIKE 句型的處理細節：LIKE 子句可使用%百分比符號表示該位置允許任何字元，稱為模糊搜尋，在 HBase 中對於特定字元開頭與結尾的文字比對有不同的符號表示。特定字元開頭之 LIKE 語法例如 LIKE '國

立%’，HBase 部分以「^國立」做代換；特定字元結尾之 LIKE 語法例如 LIKE ‘科技大學%’，HBase 部分以「科技大學\$」做代換；前後自由的模糊比對如 LIKE ‘%高雄%’則以「高雄」代換。實作程式碼如圖 20。

此外，由於 NoSQL 資料庫本身並不適合 JOIN 查詢，因此本研究並無實作 JOIN 子句。

```
public static function getMappingPredicate($expression) {
    preg_match_all("/^s*([\\w\\_]+)s*([<|=|>]+)s*([\\x{4e00}-\\x{9fa5}A-Za-z0-9_.'%']+)s*([AND|OR|Random|+]?s+/ui", $expression, $matches);
    for ($i = 1, $cnt = count($matches); $i < $cnt; ++$i) {
        if (isset($matches[$i][0])) {
            $tokens[] = $matches[$i][0];
        }
    }

    $expression = preg_replace("/^s+/", " ", $expression);
    if (preg_match("/ NOT LIKE /i", $expression)) {
        list($qualifier, $not, $op, $value) = explode(" ", $expression);
        return new LikePredicate($qualifier, "$not $op", str_replace("'", "", $value));
    }
    elseif (preg_match("/ LIKE /i", $expression)) {
        list($qualifier, $op, $value) = explode(" ", $expression);
        return new LikePredicate($qualifier, $op, str_replace("'", "", $value));
    }
    elseif (preg_match("/ [ <|=|>+ ]/i", $expression)) {
        list($qualifier, $op, $value) = $tokens;
        return new ComparisonPredicate($qualifier, $op, str_replace("'", "", $value));
    }
    else {
        return false;
    }
}
```

圖 19. getMappingPredicate 方法實作程式碼



```

class LikePredicate extends Predicate {
    private $match_value;
    private $like_str;
    private $pattern;

    public function __construct($match_value, $like_str, $pattern) {
        $this->match_value = $match_value;
        $this->like_str = $like_str;
        $this->pattern = $pattern;
    }

    public function toFilterArguments() {
        $value = $this->pattern;

        // %value%
        if (preg_match("/^%([\w\W]+)%$/", $value)) {
            $value = str_replace("%", "", $value);
        }
        // value%
        elseif (preg_match("/%$/", $value)) {
            $value = str_replace("%", "", $value);
            // $comparator = "substring";
            $value = "^" . $value;
        }
        // %value
        elseif (preg_match("/^%/", $value)) {
            $value = str_replace("%", "", $value);
            $value .= "$";
        }
        else {
            $value = "^" . $value . "$";
        }

        $so = new \stdClass;
        $so->qualifier = $this->match_value;
        $so->compareOperator = ($this->like_str == "NOT LIKE") ? "!=" : "=";
        $so->comparatorType = "regexstring";
        $so->comparatorValue = $value;
        $so->comparator = "{$so->comparatorType}:{$value}";

        return $so;
    }
}

```

圖 20. LikePredicate 類別實作程式碼



### 4.3 實驗一：SHOW TABLES

SHOW TABLES 為 MySQL 資料庫列出所有資料表的指令，本實驗以一個基於 MySQL 資料庫的 PHP 應用程式為例，分別展示預設執行結果以及載入本論文提出之中介橋接機制後，無須修改現有程式碼中的存取 MySQL 系列函式方法來達成列出 HBase 所有資料表的功能。圖 21 為修改前的程式碼，列出所有 MySQL 資料庫 afu 中的資料表名稱，執行結果如圖 22。載入中介橋接機制後的應用程式碼如圖 23，執行結果如圖 24。實驗結果顯示載入中介橋接機制後，無須修改原本的 SQL 程式碼以及 mysql\_系列函式，便可列出 HBase 資料庫所有資料表。


```
<?php

define("DB_HOST", "MySQL主機位置");
define("DB_USER", "資料庫使用者");
define("DB_PASSWORD", "MySQL資料庫密碼");
define("DB_NAME", "資料庫名稱");

$conn = mysql_connect(DB_HOST, DB_USER, DB_PASSWORD);
$db = mysql_select_db(DB_NAME);
mysql_query("SET NAMES 'utf8'");


$rs = mysql_query("SHOW TABLES");
while ($row = mysql_fetch_array($rs)) {
    print_r($row);
}
```

圖 21. 實驗一 SHOW TABLES 程式碼



```
1 Array
2 (
3     [0] => testafu
4     [Tables_in_afu] => testafu
5 )
6 Array
7 (
8     [0] => testafu2
9     [Tables_in_afu] => testafu2
10 )
11 Array
12 (
13     [0] => user_browser_log
14     [Tables_in_afu] => user_browser_log
15 )
16 Array
17 (
18     [0] => wp_commentmeta
19     [Tables_in_afu] => wp_commentmeta
20 )
21 Array
22 (
23     [0] => wp_comments
24     [Tables_in_afu] => wp_comments
25 )
26 Array
27 (
28     [0] => wp_links
29     [Tables_in_afu] => wp_links
30 )
31 Array
32 (
33     [0] => wp_options
34     [Tables_in_afu] => wp_options
35 )
36 Array
37 (
38     [0] => wp_postmeta
39     [Tables_in_afu] => wp_postmeta
40 )
41 Array
42 (
43     [0] => wp_posts
44     [Tables_in_afu] => wp_posts
45 )
46 Array
```

圖 22. 實驗一 SHOW TABLES 程式執行結果




```
<?php
namespace MySQLMigrationBridge;
require_once __DIR__."/../bootstrap.php";

define("DB_HOST", "MySQL主機位置");
define("DB_USER", "資料庫使用者");
define("DB_PASSWORD", "MySQL資料庫密碼");
define("DB_NAME", "資料庫名稱");

$conn = mysql_connect(DB_HOST, DB_USER, DB_PASSWORD);
$db = mysql_select_db(DB_NAME);
mysql_query("SET NAMES 'utf8'");

$rs = mysql_query("SHOW TABLES");
while ($row = mysql_fetch_array($rs)) {
    print_r($row);
}
```

圖 23. 實驗一載入中介橋接機制後的 SHOW TABLES 程式碼



```

1 Array
2 (
3     [0] => director
4     [Tables_in_afu] => director
5 )
6 Array
7 (
8     [0] => tbl1
9     [Tables_in_afu] => tbl1
10 )
11 Array
12 (
13     [0] => tbl2
14     [Tables_in_afu] => tbl2
15 )
16 Array
17 (
18     [0] => tbl5
19     [Tables_in_afu] => tbl5
20 )
21 Array
22 (
23     [0] => test
24     [Tables_in_afu] => test
25 )
26 Array
27 (
28     [0] => test2
29     [Tables_in_afu] => test2
30 )
31 Array
32 (
33     [0] => test_student
34     [Tables_in_afu] => test_student
35 )
36 Array
37 (
38     [0] => user_browser_log
39     [Tables_in_afu] => user_browser_log
40 )
41 Array
42 (
43     [0] => wp_links
44     [Tables_in_afu] => wp_links
45 )
46 Array
47 (
48     [0] => wp_term_relationships
49     [Tables_in_afu] => wp_term_relationships
50 )
51

```

圖 24. 實驗一載入中介橋接機制後的 SHOW TABLES 執行結果

#### 4.4 實驗二：HBaseMyAdmin 案例實作

phpMyAdmin[25]是PHP界中相當知名且廣受大眾使用的自由軟體專案，實驗二實作一個能基於SQL語法指令操作HBase資料庫之網頁應用程式雛形：

HBaseMyAdmin，如圖 25，完整程式碼為開放原始碼，可於<https://github.com/afutseng/MySQL-HBase-SQLBridge>取得。

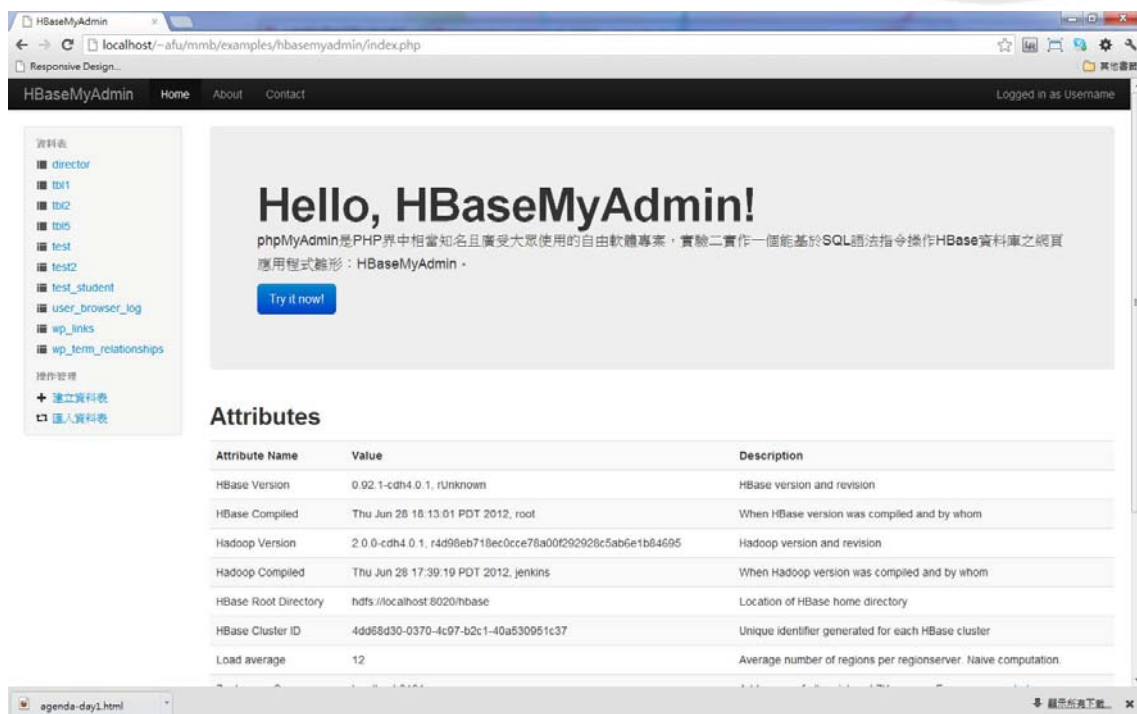


圖 25. HBaseMyAdmin 專案首頁

HBaseMyAdmin 具有以下功能特色：

1. 列出 HBase 資料庫中之資料表 (圖 26)
2. 可選擇特定資料表觀看所有的欄位 (圖 26)
3. 快速瀏覽資料表的資料記錄(圖 27)
4. 提供一個輸入 SQL 指令語法的介面供使用者存取特定條件之資料(圖 28)
5. 建立資料表 (圖 29)
6. 從指定的 MySQL 資料庫伺服器匯入資料表至 HBase 資料庫 (圖 30)

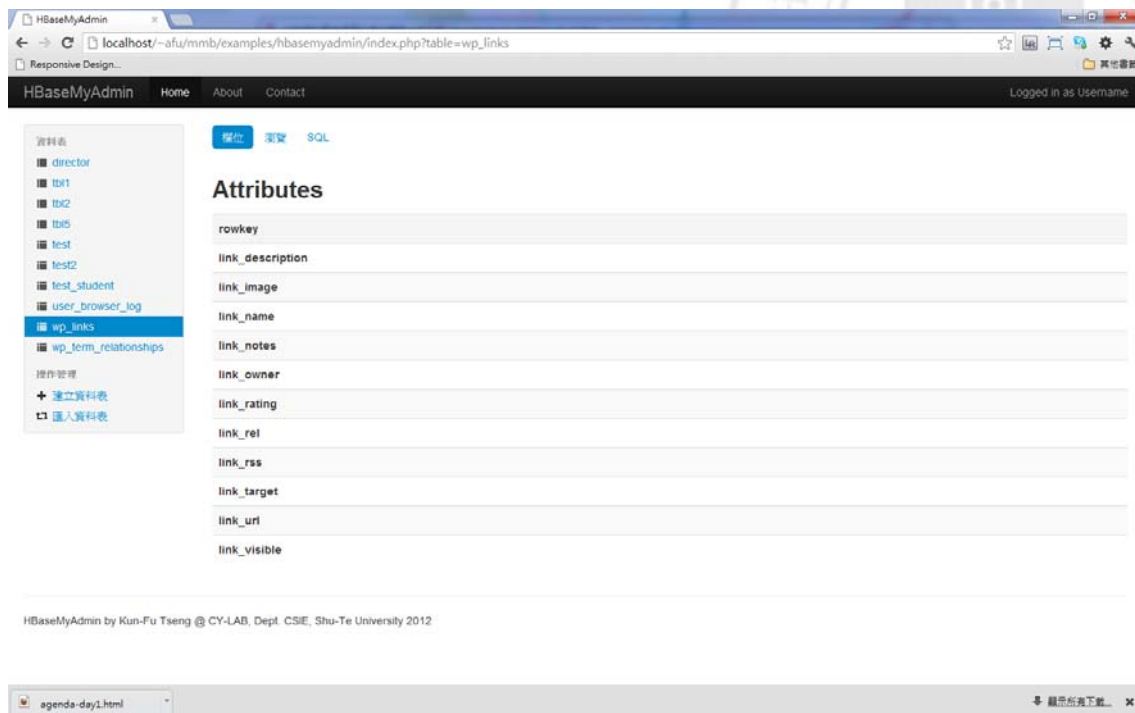


圖 26. HBaseMyAdmin 觀看資料表欄位畫面

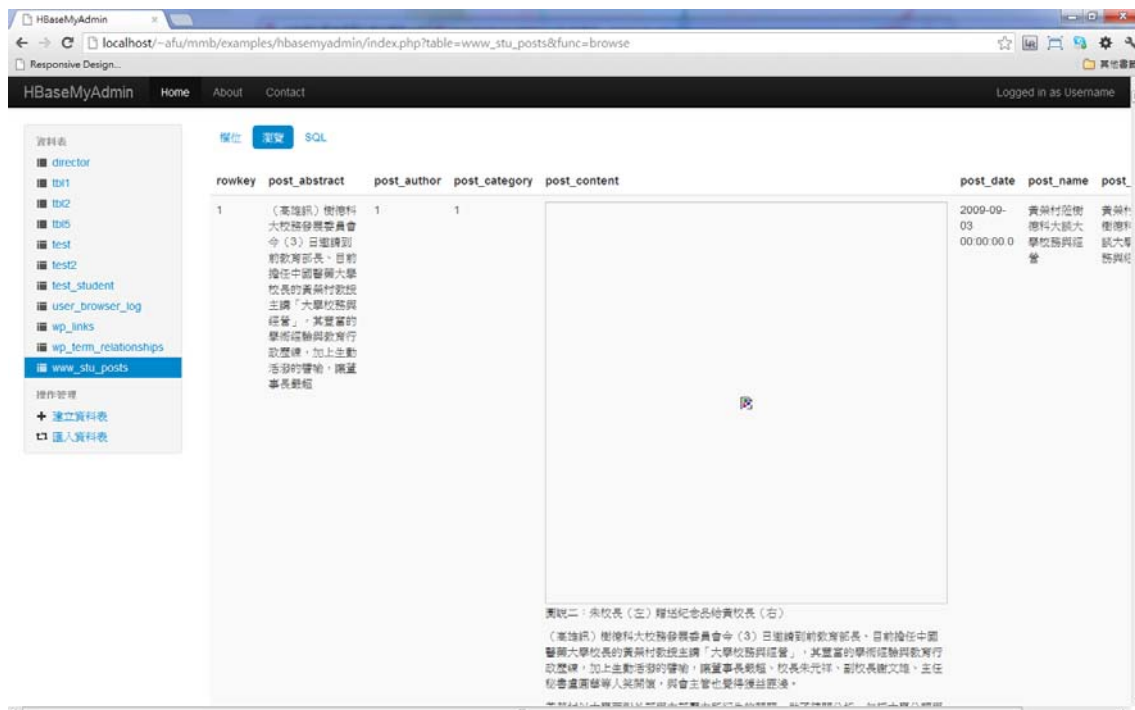


圖 27. HBaseMyAdmin 瀏覽資料表畫面

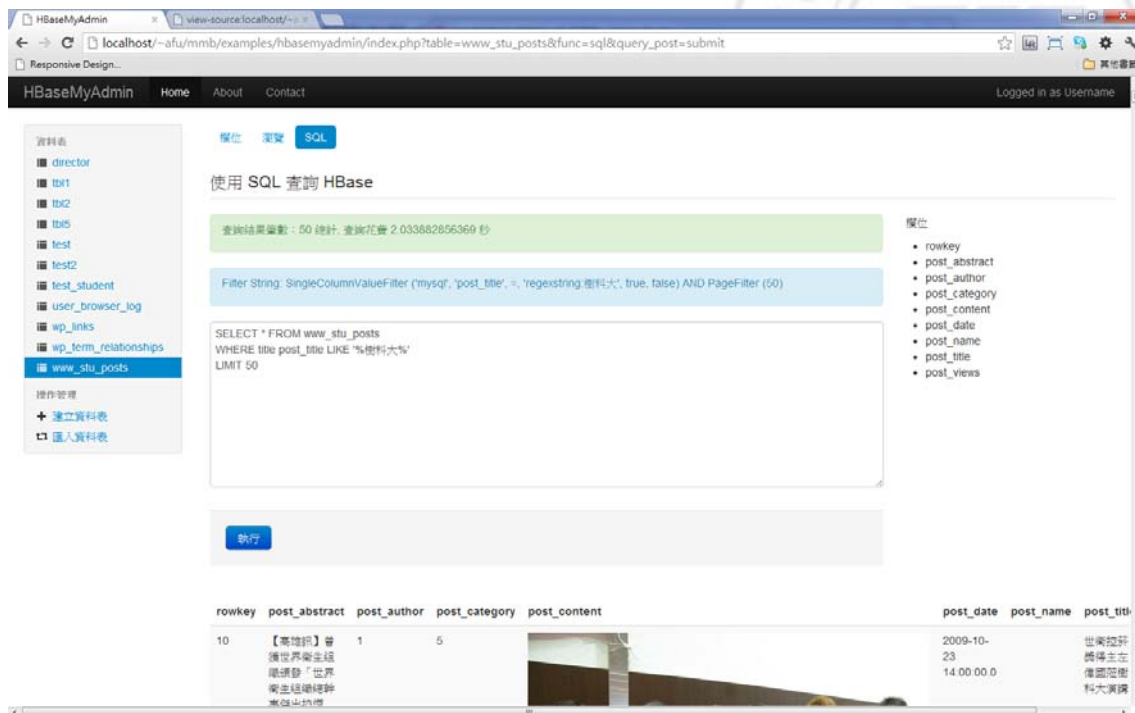
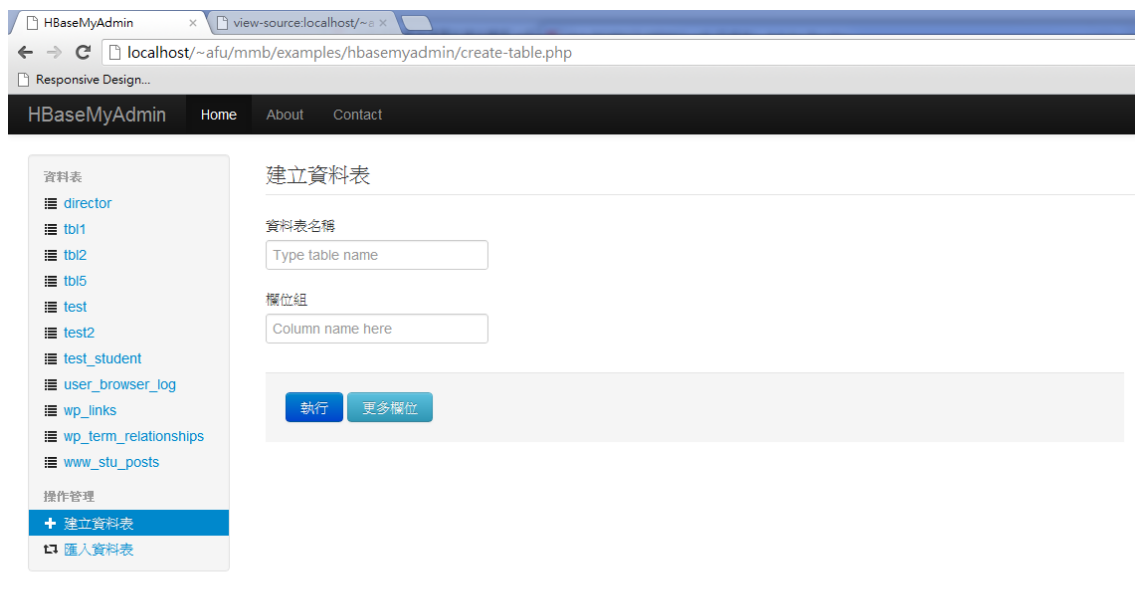


圖 28. HBaseMyAdmin 之 SQL 查詢介面



HBaseMyAdmin by Kun-Fu Tseng @ CY-LAB, Dept. CSIE, Shu-Te University 2012

圖 29. HBaseMyAdmin 之建立資料表功能畫面

The screenshot shows a web browser window with the URL `localhost/~afu/mmb/examples/hbasemyadmin/import-table.php`. The page has a dark header with 'HBaseMyAdmin' and navigation links 'Home', 'About', and 'Contact'. A left sidebar lists various database tables and a 'Import Table' button. The main content area features a light blue banner with a warning about Sqoop and MySQL JDBC driver requirements. Below this, there are tabs for '權位', '瀏覽', and 'SQL'. The 'Import Table to HBase' form contains several input fields: 'MySQL 主機位置' (hostname/IP), 'MySQL 資料庫名稱' (test), 'MySQL 資料表名稱' (table name), 'HBase 資料表名稱' (HBase table name), 'HBase Column Family' (mysql), 'MySQL 使用者帳號' (root), and 'MySQL 使用者密碼' (password). A '匯入' (Import) button is at the bottom of the form.

資料表

- director
- tbl1
- tbl2
- tbl5
- test
- test2
- test\_student
- user\_browser\_log
- wp\_links
- wp\_term\_relationships
- www\_stu\_posts

操作管理

- + 建立資料表
- 匯入資料表

使用前請確認環境有 Sqoop 1.4.1+ 版本，以及MySQL JDBC driver jar 檔於/usr/lib/sqoop/lib

權位 瀏覽 SQL

### 匯入資料表至 HBase

MySQL 主機位置

MySQL 資料庫名稱

MySQL 資料表名稱

HBase 資料表名稱

HBase Column Family

MySQL 使用者帳號

MySQL 使用者密碼

HBaseMyAdmin by Kun-Fu Tseng @ CY-LAB, Dept. CSIE, Shu-Te University 2012

圖 30. HBaseMyAdmin 之匯入資料表功能畫面



## 第五章、 結論與未來研究方向

本研究提出的中介機制解決傳統關聯式資料庫架構之應用程式遷移至如 HBase 分散式資料庫後，開發者必須學習 HBase 資料庫觀念與客戶端函式庫存取用法，應用程式亦須大幅修改方可順利轉移，此機制攔截現有的存取 MySQL 資料庫程式碼，如此可接收應用程式呼叫 MySQL 函式傳送的參數以進行處理，例如接收到 SQL 指令時，剖析 SQL 語法後針對不同類型的 SQL 指令(目前僅支援 SELECT、INSERT、UPDATE、DELETE)發出相對應的請求至 Thrift Server 以存取 HBase 資料庫傳輸資料，再將資料回傳應用程式端，如此一來應用程式便無須大幅修改存取資料庫部分的程式碼，即可順利遷移至 HBase 分散式資料庫。

未來研究方向：

1. 增加對 PHP PDO(PHP Data Object)類別的支援，讓使用 PDO 撰寫的程式也能使用本機制。
2. 增加其它 NoSQL 資料庫的支援，例如 Apache Cassandra、Apache CouchDB、MongoDB 等。
3. 以 C 語言將本論文之中介橋接機制撰寫為 PHP Extension，提昇程式的效能

## 參考文獻

- [1] 吳其勳, “克服Big Data挑戰的曙光”, iThome電腦報, <http://www.ithome.com.tw/itadm/article.php?c=73976>, 2012 年 5 月 29 日
- [2] Lith, Adam and Jakob Mattson, "Investigating storage solutions for large data" Department of Computer Science and Engineering Chalmers University of technology June, 2010, pp. 15.
- [3] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber, "Bigtable: A Distributed Storage System for Structured Data" in OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November, 2006.
- [4] Apache HBase, <http://hbase.apache.org/>
- [5] O'Reilly Media, "HBase: The Definitive Guide", August 2011
- [6] Manning, "HBase in Action", January 2012
- [7] Apache Hadoop, <http://hadoop.apache.org/>
- [8] Apache ZooKeeper™, <http://zookeeper.apache.org/>
- [9] Ubuntu LTS (Lucid Lynx), <http://releases.ubuntu.com/lucid/>
- [10] Cloudera CDH3, <https://ccp.cloudera.com/display/CDHDOC/CDH3+Installation>
- [11] CNET News, "Oracle buys Sun, becomes hardware company", [http://news.cnet.com/8301-30685\\_3-20000019-264.html](http://news.cnet.com/8301-30685_3-20000019-264.html), January 2010
- [12] Put (HBase 0.92.2 API), <http://people.apache.org/~stack/hbase-0.92.2-candidate-0/hbase-0.92.2/docs/apidocs/org/apache/hadoop/hbase/client/Put.html>
- [13] Get (HBase 0.92.2 API), <http://people.apache.org/~stack/hbase-0.92.2-candidate-0/hbase-0.92.2/docs/apidocs/org/apache/hadoop/hbase/client/Get.html>
- [14] Delete (HBase 0.92.2 API), <http://people.apache.org/~stack/hbase-0.92.2-candidate-0/hbase-0.92.2/docs/apidocs/org/apache/hadoop/hbase/client/Delete.html>
- [15] Apache Thrift, <http://thrift.apache.org/>
- [16] Mark Slee, Aditya Agarwal and Marc Kwiatkowski, "Thrift: Scalable Cross-Language Services Implementation", Facebook, April 2007
- [17] Frederic Lardinois, "Open Source: Facebook Is Now an Apache Software

- Foundation Gold  
Sponsor", [http://www.readwriteweb.com/archives/facebook\\_apache\\_foundation\\_o  
pen\\_source\\_sponsor.php](http://www.readwriteweb.com/archives/facebook_apache_foundation_open_source_sponsor.php), January 2010
- [18] Andrew Prunicki, "Introduction The Apache Thrift API's client/server architecture Source", <http://jnb.ociweb.com/jnb/jnbJun2009.html>, 2009
- [19] Chamberlin, Donald D and Boyce, Raymond F, "SEQUEL: A Structured English Query Language", Proceedings of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control (Association for Computing Machinery), 1974
- [20] SQL Tutorial – SQL Query Reference, <http://www.1keydata.com/sql/sql.html>
- [21] Luís Ferreira, "Bridging the gap between SQL and NoSQL", Universidade do Minho, May 2011
- [22] John Roijackers, "Bridging SQL and NoSQL", Master thesis , Eindhoven University of Technology, Department of Mathematics and Computer Science, pp.25-32, 2012,
- [23] 陳韡， “從格網到雲端運算的資料轉移” ，碩士論文，逢甲大學資訊工程所，2012
- [24] Sqoop User Guide  
(v1.3.0-cdh3u5), <http://archive.cloudera.com/cdh/3/sqoop/SqoopUserGuide.html>
- [25] phpMyAdmin, <http://www.phpmyadmin.net/>
- [26] Anirudh Todi, " Exposing HBase Filters to the Thrift API ", University of California, Berkeley, <https://issues.apache.org/jira/browse/HBASE-4176>, August 2011
- [27] php-sql-parser, <http://code.google.com/p/php-sql-parser/>
- [28] 曾坤福， “HBase Thrift 0.5.0 + PHP 5 安裝設定” ， <http://blog.kfchph.com/20-hbase-thrift-050-php5/>

## 附錄 A HBase 篩選器規格與範例參考表

### 通用篩選器字串語法(General Filter String Syntax)

一個簡易的篩選器表達的方式為 “FilterName (參數 1, 參數 2, ..., 參數 N)” ，必須指定篩選器名稱後面括號內的參數組，以逗點分隔。

如果參數為一個字串，字串前後必須加上單引號。

如果參數為布林值、整數、或比較運算子 “<” 、 “>” “=” 等其中之一，不可加上單引號。

篩選器名稱必須是一個 ASCII 字元所組成的單字，允許空格、單引號以及圓括號  
篩選器的參數可以包含任何 ASCII 字元。若裡面有單引號則必須將單引號加上反斜線 “\” 跳脫。

### 複合篩選器與運算子(Compound Filters and Operators)

目前支援四種運算子：AND、OR、WHILE、SKIP。(必須為大寫字)

- AND：關鍵字必須同時符合前後兩個篩選器。
- OR：關鍵字必須符合前後兩個篩選器的任何一個。
- SKIP：針對特定的列索引，若任何關鍵字不符合篩選器，該列索引將被忽略。
- WHILE：針對特定的列索引開始，持續篩選出符合關鍵字的列，直到該列索引不符合關鍵字。
- 複合篩選器：使用這些運算子互相結合組成，例如：  
“(Filter1 AND Filter2) OR (Filter3 AND Filter4)”

### 優先順序(Order of Evaluation)

括號擁有最高的優先順序，SKIP 與 WHILE 運算子為次優先層級，AND 運算子第三優先，OR 運算子優先順序最低。

例如：“Filter1 AND Filter2 OR Filter3” 等同於 “(Filter1 AND Filter2) OR Filter3” ；

“Filter1 AND SKIP Filter2 OR Filter3” 等同於 “(Filter1 AND (SKIP Filter2)) OR Filter3”

## 比較運算子與比較器(Compare Operator and Comparators)

以下為比較運算子：

1. LESS (<)
2. LESS\_OR\_EQUAL (<=)
3. EQUAL (=)
4. NOT\_EQUAL (!=)
5. GREATER\_OR\_EQUAL (>=)
6. GREATER (>)
7. NO\_OP (no operation)

Client 端程式應使用符號(<, <=, =, !=, >, >=)來表達比較運算子。

以下為比較器：

1. **BinaryComparator** – 依字典順序比較指定的字串，語法寫作 “binary”。
2. **BinaryPrefixComparator** – 依字典順序比較指定的字串，不過只比對前綴字，語法寫作 “binaryprefix”。
3. **RegexStringComparator** – 使用正規表達式比對指定的字串，語法寫作 “regexstring”。
4. **SubStringComparator** – 尋找有無指定的子字串，不分大小寫，語法寫作 “substring”。

範例 1：>, 'binary:abc' 將會匹配出任何字典順序大於 “abc” 的字串。

範例 2：=, 'binaryprefix:abc' 將會匹配出任何字典順序以 “abc” 開頭的字串。

範例 3：`!=, 'regexstring:ab*yz'` 將會匹配出以 “ab” 開頭、 “yz” 結尾的字串。

範例 4：`=, 'substring:abc123'` 將會匹配出包含 “abc123” 子字串的字串。

## PHP Client 端程式使用 Filter 篩選器範例

```
<?php
$_SERVER['PHP_ROOT'] = realpath(dirname(__FILE__).'../');
require_once $_SERVER['PHP_ROOT'].'/flib/__flib.php';
flib_init(FLIB_CONTEXT_SCRIPT);
require "storage/hbase";
$Hbase = new HBase('THRIFT_SERVER_IP', THRIFT_PORT); $Hbase->open();
$client = $Hbase->getClient();
$result = $client->scannerOpenWithFilterString('table_name', "(PrefixFilter ('row2')
AND (QualifierFilter (>=, 'binary:xyz')) AND (TimestampsFilter ( 123, 456)))");
$to_print = $client->scannerGetList($result,1);
while ($to_print) {
    print_r($to_print);
    $to_print = $client->scannerGetList($result,1);
}

$client->scannerClose($result); ?>
```

## Filter 篩選器字串範例


`“PrefixFilter (‘Row’) AND PageFilter (1) AND FirstKeyOnlyFilter ()”`

將回傳所有符合以下條件的資料記錄：

列索引(Rowkey)前三個字元必須為 “Row”

只取回一筆記錄

只取回第一個欄位



`“(RowFilter (=, ‘binary:Row 1’) AND TimeStampsFilter (74689, 89734)) OR  
ColumnRangeFilter (‘abc’, true, ‘xyz’, false))”`

將回傳所有符合以下條件的資料記錄：

列索引(Rowkey)必須為 “Row 1”

時間戳記必須為 74689 或 89734

或者符合以下條件：

欄位限定詞依字典順序符合  $\geq abc$  以及  $< xyz$

`“SKIP ValueFilter (0)”`

如果該列的值不為 0 則跳過該列。

## 附錄 B HBase 單獨篩選器語法參考表

### KeyOnlyFilter

描述: 這個篩選器無須任何參數，只回傳每一列的每個欄位限定詞（無 value）

語法: KeyOnlyFilter ()

範例: "KeyOnlyFilter ()"

### FirstKeyOnlyFilter

描述: 這個篩選器無須任何參數，只回傳每一列的第一個欄位限定詞（有 value）

語法: FirstKeyOnlyFilter ()

範例: "FirstKeyOnlyFilter ()"

### PrefixFilter

描述: 這個篩選器需要一個字串參數，將回傳列索引開頭符合指定字串的全部欄位資料

語法: PrefixFilter (<row\_prefix>)

範例: "PrefixFilter ('Row1')"

### ColumnPrefixFilter

描述: 這個篩選器需要一個字串參數，回傳欄位限定詞開頭符合指定字串的欄位

語法: ColumnPrefixFilter (<column\_prefix>)



範例: "ColumnPrefixFilter ('Col')"

## MultipleColumnPrefixFilter

描述: 同 ColumnPrefixFilter，差異在於可傳入多個欄位前綴字參數。

語法: MultipleColumnPrefixFilter (<'column\_prefix'>, <'column\_prefix'>, ..., <'column\_prefix'>)

範例: "MultipleColumnPrefixFilter ('Col1', 'Col2')"

## ColumnCountGetFilter

描述: 這個篩選器需要一個數值參數 limit，僅回傳第一列的前 limit 個欄位資料

語法: ColumnCountGetFilter (<'limit'>)

範例: "ColumnCountGetFilter (4)"

## PageFilter

描述: 這個篩選器需要一個數值參數 page\_size，將回傳 page\_size 列記錄的全部欄位資料

語法: PageFilter (<'page\_size'>)

範例: "PageFilter (2)"

## ColumnPaginationFilter

描述: 這個篩選器需要兩個數值參數 limit 與 offset，將回傳全部列的 limit 個欄位 (從指定的 offset 開始算起)

語法: ColumnPaginationFilter ('<limit>', '<offset>')

範例: "ColumnPaginationFilter (3, 5)"

## InclusiveStopFilter

描述: 這個篩選器需要一個字串參數 stop\_row\_key，將從第一列開始回傳，直到列索引符合指定的 stop\_row\_key 為止。

語法: InclusiveStopFilter ('<stop\_row\_key>')

範例: "InclusiveStopFilter ('Row2')"

## TimeStampsFilter

描述: 這個篩選器可指定多個時間戳記，將回傳符合任一指定的時間戳記記錄。

語法: TimeStampsFilter (<timestamp>, <timestamp>, ... ,<timestamp>)

範例: "TimeStampsFilter (5985489, 48895495, 58489845945)"

## RowFilter

描述: 這個篩選器需要運算子參數 `compareOp` 以及字串參數 `row_comparator`, 將回傳列索引符合條件的的全部列。

語法: `RowFilter (<compareOp>, '<row_comparator>')`

範例: `"RowFilter (<=, 'binary:xyz')"`

## Family Filter

描述: 這個篩選器需要運算子參數 `compareOp` 以及字串參數 `family_comparator`, 將回傳欄位家族符合條件的全部列。

語法: `FamilyFilter (<compareOp>, '<family_comparator>')`

範例: `"FamilyFilter (>=, 'binaryprefix:FamilyB')"`

## QualifierFilter

描述: 這個篩選器需要運算子參數 `compareOp` 以及字串參數 `qualifier_comparator`, 將回傳欄位限定詞符合條件的全部列。

`filterIfColumnMissing` 參數設為 `true` 表示若指定的欄位不存在時不取回其它欄位資料, 預設為 `false`。

`latest_version_boolean` 參數設為 `false` 表示要取回以前的版本, 預設為 `true`, 僅取回最新版本。

語法: `QualifierFilter (<compareOp>, '<qualifier_comparator>')`

範例: `"QualifierFilter (=, 'substring:Column1')"`

## ValueFilter

描述: 這個篩選器需要運算子參數 `compareOp` 以及字串參數 `value_comparator`，將回傳資料內容符合條件的紀錄。

語法: `ValueFilter (<compareOp>, '<value_comparator>')`

範例: `"ValueFilter (!=, 'binary:Value')"`

## DependentColumnFilter

描述: 這個篩選器需要字串參數 `family` 以及字串參數 `qualifier`，可將指定的欄位家族及欄位限定詞的欄位排除，取回不含該欄位以外的資料，`boolean` 參數設為 `true` 表示要排除；`compare operator` 及 `value comparator` 可進一步限制回傳的資料符合條件。

語法:

`DependentColumnFilter ('<family>', '<qualifier>', <boolean>, <compare operator>, '<value comparator>')`

`DependentColumnFilter ('<family>', '<qualifier>', <boolean>)`

`DependentColumnFilter ('<family>', '<qualifier>')`

範例: `"DependentColumnFilter ('conf', 'blacklist', false, >=, 'zebra')"`

## SingleColumnValueFilter

描述: 指定字串參數 family、字串參數 qualifier、字串參數 compare operator 以及字串參數 comparator，回傳指定的欄位家族與欄位限定詞符合條件的全部列。

語法: SingleColumnValueFilter '<family>', '<qualifier>', <compare operator>, '<comparator>', <filterIfColumnMissing\_boolean>, <latest\_version\_boolean>)

語法: SingleColumnValueFilter ('<family>', '<qualifier>', <compare operator>, '<comparator>')

範例: "SingleColumnValueFilter ('FamilyA', 'Column1', <=, 'abc', true, false)"

範例: "SingleColumnValueFilter (FamilyA', 'Column1', <=, 'abc')"

## SingleColumnValueExcludeFilter

描述: SingleColumnValueFilter 的反義篩選器，排除指定的欄位。

語法: SingleColumnValueExcludeFilter (<family>, <qualifier>, <compare operators>, <comparator>, <latest\_version\_boolean>, <filterIfColumnMissing\_boolean>)

語法: SingleColumnValueExcludeFilter (<family>, <qualifier>, <compare operator> <comparator>)

範例: "SingleColumnValueExcludeFilter ('FamilyA', 'Column1', '<=, 'abc', 'false', 'true')"

範例: "SingleColumnValueExcludeFilter ('FamilyA', 'Column1', '<=, 'abc')"

## ColumnRangeFilter

描述: 指定字串參數 minColumn、布林值 minColumnInclusive\_bool、字串參數 maxColumn、布林值 maxColumnInclusive\_bool，回傳指定的欄位範圍 minColumn 至 maxColumn(依字典排序)，兩個布林值參數決定是否包含該欄位，true 為包含；false 為排除。

語法: ColumnRangeFilter (<'minColumn'>, <minColumnInclusive\_bool>, <'maxColumn'>, <maxColumnInclusive\_bool>)

範例: "ColumnRangeFilter ('abc', true, 'xyz', false)"

## 附錄 C 中介橋接機程式碼之 mmb.php

```
<?php
namespace MySQLMigrationBridge;

define("LN", "<br>\n");
define("DS", DIRECTORY_SEPARATOR);
function debugln($str) {
    echo $str;
}
function debugdump($str) {
    var_dump($str);
}
require "SQLBridgeException.php";
require "HbaseStargate.php";
require "ThriftHBaseClientWrapper.php";
require "thrift/thrift-loader.php";
require 'php-sql-parser.php';
require 'sql-spec.php';

class MMB {
    protected static $config;
    protected static $hbase;
    protected static $parser;
    protected static $select_db;

    private static $_debug_msgs = array();
    private static $fetch_conunt = 0;
    private static $_last_filter = null;

    public function __construct() {}
    public static function connect($params = array()) {
        self::$config = $params;
```

```

// Initialize Thrift connection
$socket = new \TSocket( $params["server"], 9090 );
$socket->setSendTimeout( 3000 );
$socket->setRecvTimeout( 10000 );
$transport = new \TBufferedTransport( $socket );
$protocol = new \TBinaryProtocol( $transport );
$client = new \HbaseClient( $protocol );
$transport->open();

self::$hbase = new ThriftHBaseClientWrapper($client);

}

public static function query($query, $params = array()) {
    $default_comparator = "binary";

    if (preg_match("/^\s*SHOW TABLES$/i", $query)) {
        return MMB::showTables();
    }

    $parser = new \PHPSQLParser();
    $parsed = $parser->parse($query);
    $parsed["SQL"] = $query;

    self::debug(print_r($parsed, true));

    if (isset($parsed["SELECT"])) {
        return self::processSelectStatement($parsed);
    }
    else if (isset($parsed["INSERT"], $parsed["VALUES"])) {
        return self::processInsertStatement($parsed);
    }
    else if (isset($parsed["UPDATE"])) {
        print_r($parsed);
        return self::processUpdateStatement($parsed);
    }
}

```



```

    }
    else if (isset($parsed["DELETE"])) {
        return self::processDeleteStatement($parsed);
    }

    if ($params["link_identifier"]) {
        return \mysql_query($query, $params["link_identifier"]);
    }
    return array("1");
}

protected static function processSelectStatement($parsed = array()) {
    $query = $parsed["SQL"];

    // SELECT <select list> FROM <table reference>
    preg_match_all("/SELECT ([\w,\\*\`s]+) FROM\s+([\w,`]+)/i", $query, $matches);
    if (! empty($matches[1][0]) && ! empty($matches[2][0])) {
        $column = ($matches[1][0] == "*") ? null : $matches[1][0];
        $column = str_replace("`", "", $column);
        $column = str_replace(" ", "", $column);

        $table = str_replace("`", "", $matches[2][0]);
        $rowkey = "*";

        $filter = "";
        // SQL 有挑選特定欄位就加入 MultipleColumnPrefixFilter
        if ($column != "*" && ! empty($column)) {

            // Convert Col1,Col2 => 'Col1','Col2'
            $column = "" . str_replace(",", "','", $column) . "";
            $filter .= "MultipleColumnPrefixFilter ($column) AND ";
        }
    }
}

```

```

// SELECT <select list> FROM <table reference> WHERE <search condition>
preg_match_all("/SELECT ([\w,\*\`s]+) FROM\s+([\w,`]+)\s+(WHERE
([\w\W,\`s<=>\"-:]+)*)/i", $query, $matches);
if (! empty($matches[3])) {
    self::debug(print_r($matches, true));

    $condition_str = trim($matches[4][0]);
    //debugdump($condition_str);
    Predicate::parseMultiPredicate($condition_str);

    $predicates = Predicate::getPredicates();
    $operators = Predicate::getOperators();

    foreach ($predicates as $predicate_str) {
        $predicate = Predicate::getMappingPredicate($predicate_str);//new
\Predicate($condition_str);
        if (! $predicate) {
            throw new SQLBridgeException('查詢子句語意不明! (WHERE clause is
ambiguity.)');
        }
        $args = $predicate->toFilterArguments();

        $qualifier = $args->qualifier;
        $op = $args->compareOperator;
        $comparator = $args->comparatorType;
        $value = $args->comparatorValue;

        $comparator = (! empty($comparator)) ? $comparator : $default_comparator ;
        $scf = self::$hbase->getFirstColumnFamilyName($stable);

        if (! preg_match("/^rowkey?id$/i", $qualifier)) {
            // SQL WHERE 條件子句以 SingleColumnValueFilter 處理

```

```

        $filter := "SingleColumnValueFilter ('$cf', '{$qualifier}', {$op},
'{$comparator}:{ $value}', true, false)";
    } else {
        $filter := "RowFilter ($op, '{$comparator}:{ $value}')";
    }

    $filter := " " . current($operators) . " ";
    next($operators);
}
}

// LIMIT
preg_match_all("/LIMIT\s*([\d]+)/i", $query, $matches);
if (! empty($matches[1][0])) {
    $limit = $matches[1][0];
    if (! empty($filter)) {
        $filter := " AND ";
    }
    $filter := "PageFilter ($limit)";
}
if (preg_match("/^\s*SELECT /i", $query)) {
    return MMB::select($table, $rowkey, $column, $filter);
}
}

protected static function processInsertStatement($parsed = array()) {
    $record_count = count($parsed["VALUES"]);

    foreach ($parsed["VALUES"] as $parse_values) {
        $columns = array();
        $values = array();

        $size = count($parsed["INSERT"]["columns"]);
        for ($i = 0; $i < $size; ++$i) {
            $column = $parsed["INSERT"]["columns"][$i];

```

```

        $col_name = str_replace("`", "", $column["base_expr"]);
        $columns[] = $col_name;

        $value = $parse_values["data"][$i]["base_expr"];
        $value = str_replace("'", "", $value);
        $value = str_replace("NOW", date("Y-m-d H:i:s"), $value);
        $values[] = $value;
    }

    $table = explode(".", $parsed["INSERT"]["table"]);
    $table = str_replace("`", "", $table[count($table) - 1]);
    $rowkey = time();
    self::$hbase->insert($table, $rowkey, $columns, $values);

    // 0.2s
    usleep(200000);
}
return $record_count;
}

/**
 * Process update sql statement
 * @param array $parsed
 * @return array
 */
protected static function processUpdateStatement($parsed = array()) {
    // Extract columns & values array
    $columns = array();
    $values = array();
    $size = count($parsed["SET"]);
    for ($i = 0; $i < $size; ++$i) {
        $expression = $parsed["SET"][$i]["sub_tree"];
        $col_name = str_replace("`", "", $expression[0]["base_expr"]);
        $columns[] = $col_name;
    }

```

```

        $value = $expression[2]["base_expr"];
        $value = str_replace("", "", $value);
        $value = str_replace("NOW", date("Y-m-d H:i:s"), $value);
        $values[] = $value;
    }

    // Covert UPDATE statement to SELECT statement
    $sql = preg_replace("/UPDATE /i", "SELECT * FROM ", $parsed["SQL"]);
    $sql = preg_replace("/SET ([\w\W,\s<=>\"'\-:]* ) WHERE/i", "WHERE/i", $sql);
    $result = self::processSelectStatement(array("SQL" => $sql));

    $table = explode(".", $parsed["UPDATE"][0]["table"]);
    $table = str_replace("", "", $table[count($table) - 1]);
    foreach ($result as $record) {
        self::$hbase->insert($table, $record["rowkey"], $columns, $values);
    }

    return array(true);
}

/**
 * Process delete sql statement
 * @param array $parsed
 * @return array
 */
protected static function processDeleteStatement($parsed = array()) {
    // Covert to SELECT statement
    $sql = preg_replace("/DELETE /i", "SELECT * ", $parsed["SQL"]);
    $result = self::processSelectStatement(array("SQL" => $sql));

    $table = str_replace("", "", $parsed["DELETE"]["TABLES"][0]);

    //print_r($result);
    // Deleting every match condition's record
    foreach ($result as $record) {

```

```

        self::$hbase->deleteAllRow($table, $record["rowkey"]);
    }
    return array(true);
}

/**
 *
 * @param resource $result
 * @return array
 */
public static function fetchArray(&$result) {
    if (self::$fetch_conunt < count($result)) {
        self::$fetch_conunt++;
        $ret = current($result);
        $key = key($result);

        next($result);
        return $ret;
    }
    return false;
}

public static function fetchAssoc(&$result) {
    if (self::$fetch_conunt < count($result)) {
        self::$fetch_conunt++;
        $ret = current($result);
        $key = key($result);

        next($result);

        $ret_assoc = array();
        foreach ($ret as $k => $v) {
            if (intval($k) === $k) {
                $ret_assoc[] = $v;
            }
        }
    }
}

```

```

    }
}

    return $ret_assoc;
}

return false;
}

public static function select($table, $row, $column = null, $filter = null) {
    self::$fetch_conunt = 0;
    self::$_last_filter = "$filter";
    return self::$hbase->select($table, $row, $column, $filter);
}

public static function showTables() {
    self::$fetch_conunt = 0;
    return self::$hbase->showTables(self::$select_db);
}

public static function createTable($tableName, $columnfamily_names) {
    return self::$hbase->createTable($tableName, $columnfamily_names);
}

public static function debug($msg) {
    array_push(self::$_debug_msgs, $msg);
}

public static function getDebugMessages() {
    return self::$_debug_msgs;
}

public static function select_db($database) {
    self::$select_db = $database;
    return $database;
}

```

```

    }

    public static function getLastFilter() {
        return self::$_last_filter;
    }
} // end of class


// === mysql_* functions overloading ===
function mysql_connect($server = "localhost", $username = NULL, $password =
NULL, $new_link = false, $client_flags = 0) {
    $params = array(
        "server"      => $server,
        "username"    => $username,
        "password"    => $password,
        "new_link"    => $new_link,
        "client_flags" => $client_flags
    );
    return MMB::connect($params);
}

function mysql_fetch_array(&$result, $result_type = MYSQL_BOTH) {
    return MMB::fetchArray($result, $result_type);
}

function mysql_fetch_assoc(&$result) {

    return MMB::fetchAssoc($result);
}

function mysql_query($query, $link_identifier = false) {
    try {
        return MMB::query($query, array("link_identifier" => $link_identifier));
    } catch (SQLBridgeException $e) {
        throw new SQLBridgeException($e->getMessage());
    }
}

```



```

    }
}

function mysql_field_name(&$result, $field_offset) {
    $row = $result[0];
    $offset = 0;
    foreach ($row as $col_name => $v) {
        if ($field_offset == $offset) {
            return $col_name;
        }
        $offset++;
    }
}

function mysql_num_fields(&$result) {
    if (is_array($result) && array_key_exists(0, $result)) {
        return count(array_keys($result[0]));
    }
    return FALSE;
}

function mysql_num_rows(&$result) {
    if (is_array($result)) {
        return count($result);
    }
    return FALSE;
}

function mysql_select_db($database_name, $link_identifier = NULL) {
    MMB::select_db($database_name);
    return TRUE;
}

```

## 附錄 D 中介橋接機程式碼之 ThriftHBaseClientWrapper.php

```
<?php
namespace MySQLMigrationBridge;

class ThriftHBaseClientWrapper {
    protected $_hbaseclient;

    public function __construct(\HbaseClient $hc) {
        $this->_hbaseclient = $hc;
    }

    public function query($query, $params = array()) {

    }

    /**
     *
     * @param resource $result
     * @return array
     */
    public function fetchArray(&$result) {

    }

    public function select($table, $row, $column = null, $filter = "") {
        $tscan_params = array();
        if (! empty($filter)) {
            MMB::debug( " filter = $filter ## <br>");
            $tscan_params["filterString"] = $filter;
        }
        $tscan = new \TScan($tscan_params);
        $scanner = $this->_hbaseclient->scannerOpenWithScan("$table", $tscan);
    }
}
```

```

$res = array();
$idx = 0;
while ($row = $this->_hbaseclient->scannerGet($scanner)) {
    $row = $row[0];
    $res[$idx]["rowkey"] = $row->row;

    foreach ($row->columns as $qualifier => $cell) {
        $column_name = preg_replace("/^\[w\]+:/", "", $qualifier);
        $res[$idx][$column_name] = $cell->value;
    }

    $idx++;
}

return $res;
}

public function insert($table, $rowkey, $columns = array(), $values = array()) {
    $cf = $this->getFirstColumnFamilyName($table);
    $size = count($columns);
    $mutations = array();
    for ($i = 0; $i < $size; ++$i) {
        $column = $columns[$i];
        $value = $values[$i];
        $row = new \Mutation( array(
            "column" => "$cf:$column",
            "value" => "$value"
        ));

        $mutations[] = $row;
    }

    try {
        $this->_hbaseclient->mutateRow($table, $rowkey, $mutations);
    } catch (Exception $e) {

```

```

        return 0;
    }
    return array(1);
}

public function showTables($mysql_db_name) {
    $result = array();
    foreach($this->_hbaseclient->getTableNames() as $name) {
        array_push($result, array(0 => $name, "Tables_in_{ $mysql_db_name}" =>
$name));
    }

    return $result;
}

public function getFirstColumnFamilyName($tableName) {
    $columnfamilies = $this->_hbaseclient->getColumnDescriptors($tableName);
    foreach ( $columnfamilies as $cf ) {
        return str_replace(":", "", $cf->name);
    }
}

public function getColumnDescriptors($tableName) {
    return $this->_hbaseclient->getColumnDescriptors($tableName);
}

public function deleteAllRow($tableName, $rowkey) {
    return $this->_hbaseclient->deleteAllRow($tableName, $rowkey);
}

public function createTable($tableName, $columnfamily_names) {
    $columnfamilies = array();
    if (! is_array($columnfamily_names) && is_string($columnfamily_names)) {
        $columnfamily_names = array($columnfamily_names);
    }
}

```

```

foreach ($columnfamily_names as $cfname) {
    $columnfamilies[] = new \ColumnDescriptor(array(
        "name" => "$cfname:",
        "maxVersions" => 3,
        "compression" => "NONE",
        "inMemory" => 0,
        "bloomFilterType" => "NONE",
        "bloomFilterVectorSize" => 0,
        "bloomFilterNbHashes" => 0,
        "blockCacheEnabled" => 0,
        "timeToLive" => -1
    ));
}

return $this->_hbaseclient->createTable($tableName, $columnfamilies);
}
} // end of class

```

## 附錄 E 中介橋接機程式碼之 sql-spec.php

```
<?php
namespace MySQLMigrationBridge;

class PredicateException extends \Exception
{
}

class Predicate {
    protected static $predicates;
    protected static $predicates_operators;

    public static function parseMultiPredicate($expression) {

        //preg_match_all("/s*([\w`]+)\s+([<=>LIKE]+)\s+([\w\W_@.!\$%\&'`]+)\s*([ANDOR]
        R]+)?\s+/i", $expression, $matches);

        preg_match_all("/s*([\w`]+)\s+([<=>LIKE]+)\s+([\x{4e00}-\x{9fa5}A-Za-z0-9_'\%]+)\s*([ANDORandor]
        +)?\s+/ui", $expression, $matches);

        if (isset($matches[0][0])) {
            $predicate_max_cnt = count($matches[1]);
            $idx = 0;
            $predicates = array();

            // 開始蒐集 predicate, 最多 $predicate_max_cnt 句
            for ($idx = 0; $idx < $predicate_max_cnt; ++$idx) {
                $predicates[$idx] = "";

                // v1 . op . v2
                for ($i = 1; $i <= 3; ++$i) {
                    if (!isset($matches[$i][$idx])) {
                        break;
                    }
                }
            }
        }
    }
}
```

```

    }
    $predicates[$idx] .= $matches[$i][$idx] . " ";
}

// 砍掉不具備兩個運算元與一個運算子的 predicate
if ($i != 4) {
    unset($predicates[$idx]);
}
}

}

self::$predicates = $predicates;
// collect operators if has any operators
if (isset($matches[4])) {
    $matches[4] = array_map("strtoupper", $matches[4]);
    self::$predicates_operators = array_filter($matches[4]);
}

return $predicates;
}

public static function getPredicates() {
    return self::$predicates;
}

public static function getOperators() {
    return self::$predicates_operators;
}

public static function getMappingPredicate($expression) {

preg_match_all("/s*([w`]+)s*([<|=|>]+)s*([x{4e00}-x{9fa5}A-Za-z0-9_`%`]+)s*([
ANDORandor]+)?\s+/ui", $expression, $matches);

```

```

for ($i = 1, $cnt = count($matches); $i < $cnt; ++$i) {
    if (isset($matches[$i][0])) {
        $tokens[] = $matches[$i][0];
    }
}

$expression = preg_replace("/\s+/", " ", $expression);
if (preg_match("/ NOT LIKE /i", $expression)) {
    list($qualifier, $not, $op, $value) = explode(" ", $expression);
    return new LikePredicate($qualifier, "$not $op", str_replace("'", "", $value));
}
elseif (preg_match("/ LIKE /i", $expression)) {
    list($qualifier, $op, $value) = explode(" ", $expression);
    return new LikePredicate($qualifier, $op, str_replace("'", "", $value));
}
elseif (preg_match("/ [<|=|>]+ /i", $expression)) {
    list($qualifier, $op, $value) = $tokens;
    return new ComparisonPredicate($qualifier, $op, str_replace("'", "", $value));
}
else {
    return false;
}
}

/**
 *
 */
<comparison predicate> ::=
    <row value constructor> <comp op>
    <row value constructor>

*/

class ComparisonPredicate extends Predicate {
    private $row_value_constructor1;
    private $comp_op;

```



```

private $row_value_constructor2;

public function __construct($rvc1, $op, $rvc2) {
    $this->row_value_constructor1 = $rvc1;
    $this->comp_op = $op;
    $this->row_value_constructor2 = $rvc2;
}

public function toFilterArguments() {
    $o = new \stdClass;
    $o->qualifier = $this->row_value_constructor1;
    $o->compareOperator = $this->comp_op;
    $o->comparatorType = "binary";
    $o->comparatorValue = $this->row_value_constructor2;
    $o->comparator = "binary:{$this->row_value_constructor2}";
    return $o;
}
}

class LikePredicate extends Predicate {
    //
    private $match_value;
    private $like_str;
    private $pattern;

    public function __construct($match_value, $like_str, $pattern) {
        $this->match_value = $match_value;
        $this->like_str = $like_str;
        $this->pattern = $pattern;
    }

    public function toFilterArguments() {

        $value = $this->pattern;
        // %value%

```

```

if (preg_match("/^%([\w\W]+)%$/", $value)) {
    $value = str_replace("% ", "", $value);
}
// value%
elseif (preg_match("/%$/", $value)) {
    $value = str_replace("% ", "", $value);
    //$comparator = "substring";
    $value = "^" . $value;
}
// %value
elseif (preg_match("/^%/ ", $value)) {
    $value = str_replace("% ", "", $value);
    //$comparator = "substring";
    $value .= "$";
}
else {
    $value = "^".$value."$";
}

$o = new \stdClass;
$o->qualifier = $this->match_value;
$o->compareOperator = ($this->like_str == "NOT LIKE") ? "!=" : "=";
$o->comparatorType = "regexstring";
$o->comparatorValue = $value;
$o->comparator = "{$o->comparatorType}:{ $value}";

return $o;
}
}

```