國立政治大學資訊科學系

Department of Computer Science
National Chengchi University

碩士論文

Master's Thesis

基於可延伸性表格的多租戶應用程式資料綱要轉換工具

Implementing Customizable Data Schemas for Multi-tenant Applications Using Extension Table Layout

研 究 生：李明憲

指導教授：陳　恭

中華民國一百一年 十一 月

致謝辭

今天能夠順利完成學業，首要感謝的是陳恭老師，謝謝他這兩年多來願意給我許多機會嘗試不同面向的發展，並且給我許多專業上的建議；除此之外，在平日的指導過程中，他也以嚴謹中帶點幽默的態度和我互動，讓我們之間的相處沒有任何壓力。另外，也要感謝口試委員王有禮老師以及廖峻鋒老師在口試當天的提點與指教，讓我最後能夠順利完成論文。

此外，要感謝從小一路培養我到大的父母，在他們無私的關照下，我才能擁有一個好的學習環境，無顧慮地專心在我的研究上面，每個周末的晚餐時刻和父母的閒聊、關心也成了我研究生活中最好的調劑。

再來，感謝實驗室的夥伴一路以來的協助、包容以及鼓勵。首先要感謝于育、文楷學長在我有技術上面的問題時，給我提點以及建議。感謝昱霖在我修課期間一起努力完成多項課程上的專題。感謝子文在我心情煩躁的時候願意陪我聊天解悶；感謝宸暉跟昱宇協助我完成實驗室的大小事情，減輕我許多負擔，感謝佐玄擔任實驗室開心果的角色，讓實驗室充滿歡笑，紓解我的壓力。感謝家宇跟榆澤願意傾聽我任何大小瑣事，讓我在最後的衝刺階段能擁有一個好的傾吐空間。感謝致緯跟苑霖給了我很多協助以及建議，雖然你們不常出現，但總是能夠適時地幫助我解決許多問題。

接下來，感謝我的兩位好室友：柏堯以及建成，謝謝他們在我生活上的照顧，忍耐了我很多不好的生活習慣。

然後，感謝我的大學同窗好友：季儒、翌涵、辰鍵以及耀瑋，大家偶爾的聚會讓我在辛苦的研究生活中，能夠有稍微喘息的機會。

最後，要感謝政大資科所的所有老師、助教以及同學們，在大家的指教以及協助之下，我才能順利地完成學業。

2012 11 27 明憲

基於可延伸性表格的
多租戶應用程式資料綱要轉換工具

摘要

雲端運算環境的興起，軟體即服務(Software as a Service, SaaS)的營運模式也開始受到軟體開發商的注意，其中一項關鍵技術是支援多租戶的軟體開發。在設計多租戶應用程式時有許多需要考量的因素，包含每個租戶各自的客製化設定、資料安全性等等。本論文著重於資料層級的客製化部份，如何讓各個租戶之間可以共用資料庫，但又能提供租戶適度地更改其資料表綱要(schema)以達到客製化的需求。

為了解決資料層級客製化所面臨的問題，一種常見的作法是可延伸性表格(Extension Table Layout)。但是，在關聯式資料庫中實作可延伸性表格，軟體開發人員為了配合可延伸性表格的資料存放方式，所使用的 SQL 語句會變得相當複雜且易錯。因此，本研究實作一個系統工具協助軟體開發人員將 SQL 語句，從一租戶一資料表寫法自動轉換成以可延伸性表格邏輯表達的 SQL 語句，透過執行轉換後的語句來對儲存在可延伸性表格的資料進行增刪修改。

軟體開發人員在本系統工具的協助下，加入租戶識別碼(TenanId)

的開發概念後，即可實際建立一個多租戶應用程式。為了評估本系統工具的效用，我們以其建置了一個多校(租戶)選課系統，並進行多項效能實驗以探討影響本系統工具效能的因素。初步的結果顯示，本系統工具可以協助軟體開發人員，以較低的代價滿足資料層級客製化應用的需求。

國立政治大學 National Chengchi University

Implementing Customizable Data Schemas for Multi-tenant Applications Using Extension Table Layout

Abstract

Software as a service (SaaS) is an emerging service model of cloud computing. One of the key technology in SaaS is supporting multi-tenant applications. There are many considerations in the design of multi-tenant applications, including customized configuration of tenants and data security. In this thesis, we focus on data-level customized configuration, and propose an approach for not only sharing database between tenants but also supporting tenants to modify their table schemas within limits.

Extension Table Layout is one solution for solving the problems in data-level customized configuration for multi-tenant applications. However, SQL statemenst based on the Extension Table Layout are rather complicated and error-prone. Thus, we develop a tool to help software developers that will automatically rewrite the SQL statements from the common Private Table Layout into those from the Extension Table Layout at runtime. In other words, our tool enable software developers to write SQL statements in a multi-tenant application like in a single tenant application.

Indeed, software developers could develop a multi-tenant

application easily by using our tool and the multi-tenant enabler - TenantId. In order to assess the feasibility of our tool, we develop an online multi-tenant course election application. Besides, we disscus the effectiveness of the factors that affecting our tool and work on a number of experiments and performance tests. The preliminary results show that our SQL rewriting tool can help software developers at a lower cost to meet the needs of data-level customized applications to a degree.

目錄

國立政治大學 National Chengchi University

表 目 錄

圖 目 錄

第一章

緒論

1.1 前言

雲端運算的技術和架構是目前十分受到重視的研究議題，同時，也是各個軟體大廠競相努力發展的目標。根據NIST的定義[Mell et al.11]：雲端運算是一種藉由網路存取所配置的共享資源(network、server、storage、application、and service)，以最小的成本做管理工作以及運算資源的配置。

隨著雲端運算環境的興起，軟體即服務(Software as a Service, SaaS)的營運模式也開始受到軟體開發商注意。不同於以往到客戶端安裝(on-premises)，供單一用戶使用的模式，SaaS的主要特色是在資料中心安裝，以多租戶(multi-tenants)方式營運：每個租戶的使用者透過網路使用該應用程式，其運作所需的軟硬體設施由資料中心與應用程式的軟體開發人員負責維運，租戶只需按使用量與服務品質等因素依合約付費。在設計多租戶應用程式架構時有許多需要考量的因素，包含租戶各自的客製化設定、資料安全性等等，要能達到獨立而不會受到其他租戶的影響，這些對SaaS服務而言都非常的重要，否則，一旦一個租戶在使用時發生狀況，其他租戶可能就會連帶的受到影響。由於此種開發模式在軟硬體資源的使用上相當可觀，軟體開發人員不可能為了每一個租戶都提供一台實體的機器，所以，提供具安全性的共用資料架構、客製化組態為多租戶技術所面臨到的問題。

1.2 研究動機

由於中小企業常常基於預算而無法獨立負擔一個私有應用程式開發以及維運，所以，軟體開發商採用多租戶共享資源的模式企圖減少開發以及維運成本，同時，這些減少的成本反應在對租戶的收費以符合中小企業的預算。藉此，軟體開發商渴望長尾效應[Chong et al.06]的發生，透過較低的租用價格去吸引中小企業，擴大消費市場並接觸到更多消費者。

如圖 1.1 所示，採用多租戶共享資源的模式在開發初期由於技術層面大多尚未成熟，因此，可預期到相較於單一租戶的開發模式來說，需要花費較大的成本進行技術的探索以及設計；但相對地，軟體開發人員採用該模式可以在有限的軟硬體環境下，透過多租戶共享軟硬體資源，提供服務給較多數量的租戶；另一方面，藉由多個租戶分攤軟硬體資源費用，例如：軟體授權費用、硬體設備成本等，亦可以減少每個租戶實際所需負擔的花費以解決中小企業基於預算而無法獨立負擔一個私有應用程式開發以及維運的問題。長期看來，可以預期每個租戶實際花費的所有租用成本會相對於獨立負擔一個私有應用程式開發以及維運來得低。

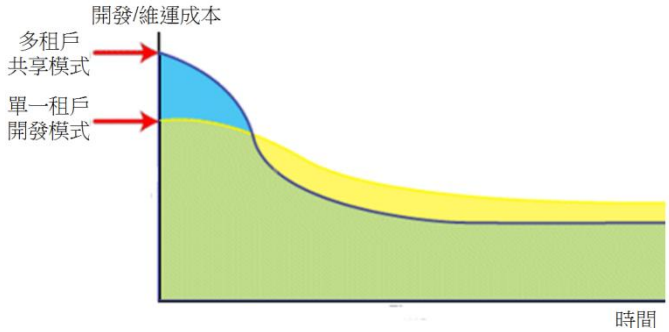

圖 1.1 多租戶共享模式以及單一租戶開發模式隨時間變化之開發、維運成本比較圖

多租戶技術主要提出兩項目標，第一項是在軟硬體資源上達到租戶共享以減少開發成本；第二項是在共享資源的前提下亦可以提供每個租戶以自助的方式適度地客製化以達到擴充性。目前在實作多租戶應用程式之資料層級設計有兩種作法，第一種作法是一租戶一資料庫（如圖 1.2 所示），此種作法在成本上

極為浪費，無法達到多租戶技術所提出的資源共享目標；第二種作法是多租戶共享一資料庫（如圖 1.3 所示），此種作法雖能達到多租戶資源共享，但缺乏擴充性，所有的租戶都必須使用同一種資料表綱要，無法滿足客製化的需求。本研究嘗試在上述第一種和第二種作法中找到一個平衡點，提供一種既可以讓多租戶共享資料庫又可以提供租戶適度地客製化其資料表綱要的作法。

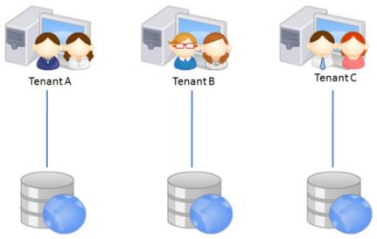

圖 1.2 一租戶一資料庫

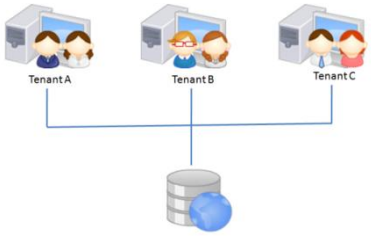

圖 1.3 多租戶共享一資料庫

圖 1.4 為多租戶技術的開發概念示意圖，藉由一個應用程式搭配租戶共享資料庫的模式可以提供服務給多個租戶使用。圖 1.4 中三個租戶透過網路介面設定或是撰寫設定檔的方式對租用的應用程式進行相關設定，例如：設定應用程式的使用介面、適度地客製化資料欄位等動作；軟體開發人員則從應用程式加入每個租戶所獨有的租戶識別碼(TenantId)並迭代到資料層級，所以，不論是應用程式或是資料層級皆可以透過租戶識別碼(TenantId)去辨識租戶以完成該租戶及其使用者所交付的任務。

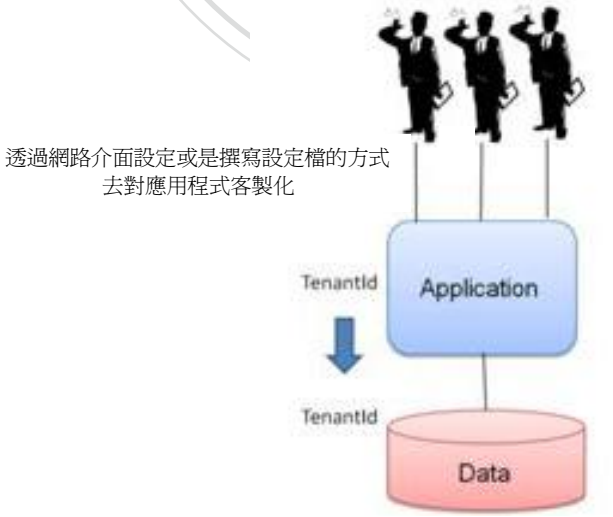

圖 1.4 多租戶技術的開發概念示意圖

以開發一個多校(租戶)選課系統為例，軟體開發人員可能會提供一個課程資料表來存放課程資訊頁面所顯示的資料，其中包含了許多欄位，例如：課程名稱欄位、上課老師名稱欄位等，上述這些欄位都是每個租戶在課程資訊頁面共同需要的資料欄位，稱之為共有欄位；但由於每個租戶在課程資訊頁面的資料會有些不同，這裡，透過租戶建立其獨自擁有的私有欄位，讓每個租戶的課程資訊頁面上除了顯示共有欄位資料以外還有租戶自己獨有的私有欄位(客製化欄位)資料。但在開發過程中，由於軟體開發人員無法預先得知租戶會建立哪些私有欄位，所以，不會單獨針對私有欄位資料進行功能設計，私有欄位資料通常是扮演與共有欄位資料一起顯示在頁面上的角色，例如：當學生進到課程資訊的頁面時，應用程式可能會執行 SELECT 語句，將共有欄位資料以及私有欄位資料一起顯示在頁面，所以，不同租戶的課程資訊頁面，可能因為擁有不同的私有欄位而造成頁面顯示的內容有所不同。

如同上一段所敘述的，站在軟體開發人員的角度，由於無法預測各個租戶的私有欄位(即客製化欄位)，所以，軟體開發人員在開發、維運的階段只曉得共有欄位的存在，因此，只會針對共有欄位資料進行多租戶應用程式的功能設計。倘若能夠提升軟體開發人員統一管理租戶的共有欄位以及其資料的能力，則可以增加開發以及維運上的方便性。

在 2.2.1 小節所介紹的 Private Table Layout 雖符合在多租戶共享資料庫的情況下亦可以提供租戶適度地客製化其資料表綱要的能力(這裡所提到的”租戶適度地客製化其資料表綱要的能力”是指租戶在不影響軟體開發人員所建立的共有欄位前提下，能夠以自助的方式透過設定來新增(建立)有別於其他租戶的私有欄位，亦即客製化欄位，此後，租戶也可以自行去更改這些私有欄位的資料型態或是欄位名稱，甚至在不需要這些私有欄位的時候進行刪除私有欄位的動作)；但是，在眾多考量因素下，相對於 Private Table Layout，2.2.3 小節所描述的 Extension Table Layout 更適合多租戶應用程式之資料層級設計，

其所提出共有欄位、私有欄位的觀念，除了可以滿足租戶適度地客製化其資料表綱要的需求外，亦可以提升軟體開發人員統一管理租戶的共有欄位以及其資料的能力。簡單來說，採用 Extension Table Layout 的概念，租戶以自助的方式建立其私有欄位，滿足租戶客製化欄位的需求；軟體開發人員統一管理租戶的共有欄位以及其資料則可以減少在維運多租戶應用程式時修改既有資料表綱要以及資料所花費的時間成本，例如：軟體開發人員在開發新服務的過程中，倘若需要修改已存在的資料表綱要(schema)，則可以透過共有欄位的修改代替逐一對個別租戶資料表欄位修改的動作，節省其所花費的時間成本。

1.3 研究目的

採用 Extension Table Layout 的概念進行多租戶應用程式之資料層級設計，除了可以達到租戶客製化欄位的目標外，亦可以提供軟體開發人員對租戶的共有欄位以及其資料進行統一管理的能力。但是，軟體開發人員在撰寫 SQL 語句時，為了配合 Extension Table Layout 的資料存放方式，所以，無法沿用一租戶一資料表的寫法來撰寫 SQL 語句。為了解決上述問題，本研究將實作一套系統工具協助軟體開發人員將 SQL 語句從一租戶一資料表寫法轉換成以 Extension Table Layout 邏輯表達的 SQL 語句，透過執行轉換後的語句來對儲存在可延伸性表格的資料進行增刪修改。

本研究的主要目標有：

1. 開發一套系統工具，協助軟體開發人員可以沿用一租戶一資料表的寫法開發應用程式，但透過 SQL 語句的改寫，資料層仍可以是以 Extension Table Layout 方式來儲存資料。
2. 評估以我們發展的工具來開發 Extension Table Layout 多租戶應用程式之資料層級設計的可行性。

在本系統工具的協助下，軟體開發人員以 Extension Table Layout 的概念進行多租戶應用程式之資料層級設計，使用 JDBC 介面進行資料庫的操作，沿用一租戶一資料表的 SQL 語句邏輯對資料進行增刪修改以及查詢等動作；系統工具則改寫 JDBC 的部分 API 去攔截軟體開發人員所撰寫的 SQL 語句，有系統地將 SQL 語句轉換成以 Extension Table Layout 邏輯表達的 SQL 語句並在資料庫上執行。

1.4 研究成果

基於上述的研究動機以及目的，蒐集相關的技術資料，依據 Extension Table Layout 的邏輯歸納出語句轉換步驟，同時，實作系統工具協助軟體開發人員雖以 Extension Table Layout 的概念進行多租戶應用程式之資料層級設計，卻可以沿用一租戶一資料表的 SQL 語句邏輯撰寫 SQL 語句。另外，採用本系統工具協助開發多租戶應用程式展示使用本系統工具所帶來的優點。最後，透過一連串的實驗推斷出影響此系統工具效能的因素並模擬多租戶共用資料庫的環境進行多項測試以評估此作法的可行性。

本研究的主要成果有：

1. 實作系統工具協助軟體開發人員將 SQL 語句從一租戶一資料表寫法轉換成以 Extension Table Layout 邏輯表達的 SQL 語句，透過執行轉換後的語句來完成軟體開發人員原先所預期達到的目的。
2. 採用本系統工具實際開發一個多校(租戶)選課系統並從開發過程中展示採用 Extension Table Layout 提出的共有欄位、私有欄位所帶來的優點(提升軟體開發人員統一管理租戶的共有欄位以及其資料的能力以降低維運所需花費的時間成本、滿足租戶適度地客製化其資料表綱要的需求)。

3. 探討影響本系統工具效能的因素，另外，實際測試本系統工具在效能上所造成的影響並根據其測試結果評估此作法的可行性。

1.5 論文大綱

本論文分成五個章節，第一章為緒論，主要對研究的動機、目的進行解說，同時，簡單地對系統工具的實作設計、研究結果作概略性說明。第二章主要介紹三種多租戶應用程式之資料層級設計，另外，介紹 SQL 語句結構剖析之工具－JSqlParser 的相關技術。第三章則完整地從設計層面到實作層面對本系統工具的核心架構進行詳細地解說。第四章將實際使用本系統工具協助多租戶應用程式之資料層級設計以展示採用本系統工具的優點。另外，進行一連串的實驗，藉著實驗結果推斷影響本系統工具執行時間的因素。最後，進行多項測試去更進一步地了解實際使用本系統工具在效能上會造成哪些影響並根據其結果進行可行性評估。在最後第五章的部分，提出結論並探討未來值得深入研究的發展方向以及相關議題。

國立政治大學 National Chengchi University

第二章

技術背景與相關研究

多租戶(Multi-Tenancy)技術主要希望可以讓多個租戶(這裡的”租戶”泛指每個向軟體開發人員租用服務的客戶)共同使用一個運算環境或是應用程式，同時，提供完善的隔離技術以維護每個租戶的資料安全性。此外，亦要讓不同的租戶以自助的方式透過網路介面進行功能設定以達到客製化的效果。

採用多租戶技術可以創造許多實質上的效益。對軟體開發人員來說，可以透過多租戶共享資源以達到在有限軟硬體設備下，提供服務給最多租戶數量的目的；此外，租戶們透過分攤軟體授權費用以及硬體設備成本的方式，降低其租用應用程式所需負擔的成本。另一方面，在這種共享的架構下，租戶亦可以享受到每次應用程式所更新的服務，增加其使用功能。

本研究著重於資料層級的客製化部份，如何讓各個租戶之間可以共用資料庫，但又能提供租戶適度地更改其資料表綱要(schema)以達到客製化的需求。首先，在 2.1 小節多租戶應用程式之資料層級設計中，依照租戶資料存放的共享程度分成三種類型的資料層級設計架構並逐一詳細探討，最後，歸納出實際在進行多租戶應用程式之資料層級設計時所需考量的因素。為了在多租戶共用資料庫的前提下，達到租戶適度地客製化其資料表綱要的需求，在 2.2 小節中將介紹四種符合上述需求的多租戶應用程式之資料表綱要類型，其中，2.2.3 小節的 Extension Table Layout 提出了共有欄位、私有欄位的觀念，除了滿足租戶客製化欄位的需求外，亦可以提升軟體開發人員對租戶共有欄位以及其資料進行統一管理的能力，藉此可以讓軟體開發人員更容易開發多租戶應用程式。

Extension Table Layout 的資料存放方式會讓軟體開發人員無法沿用一租

戶一資料表的寫法撰寫 SQL 語句。為了讓軟體開發人員雖以 Extension Table Layout 的概念進行多租戶應用程式之資料層級設計，卻可以沿用一租戶一資料表的 SQL 語句邏輯對資料進行增刪修改，所以，本研究實作一套系統工具將 SQL 語句從一租戶一資料表寫法轉換成以 Extension Table Layout 邏輯表達的 SQL 語句。在進行實際的語句轉換之前，本系統工具必需對 SQL 語句進行結構剖析以獲得語句轉換過程中所需的語句元素，2.3 小節所介紹的 SQL 語句結構剖析工具-JSqlParser 則可以協助系統工具進行語句結構剖析並提供語句元素以利後續的語句轉換動作。

2.1 多租戶應用程式之資料層級設計

進行多租戶應用程式之資料層級設計時，如何提高租戶間資料存放的共享程度是很重要的議題。下列各小節中，將探討三種不同共享程度的多租戶應用程式之資料層級設計(圖 2.1)，並在後續的各個小節中，針對每一種技術在開發上所面臨的問題進行探討並作出經濟效益上的評估，最末節提出在多租戶應用程式之資料層級設計時所需要考量的因素。

2.1.1 多租戶應用程式之資料層級設計分類

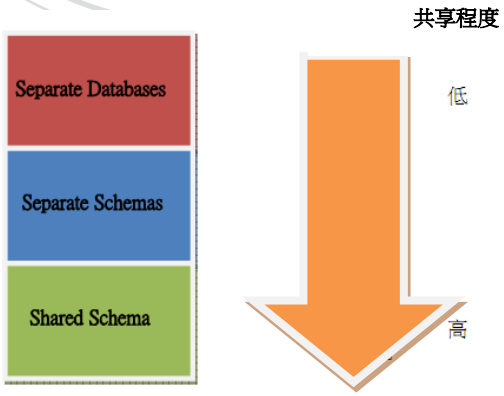

圖 2.1 三種多租戶應用程式之資料層級技術

(Separate Databases、Separate Schemas、 Shared Schema)共享程度示意圖

多租戶應用程式之資料層級設計依租戶資料存放的共享程度，如圖 2.1 所示，可以簡單地分成三類，從租戶資料存放的共享程度由低到高，分別是 ：Separate Databases、Separate Schemas 以及 Shared Schema[Chong et al.06]。採用 Separate Databases 的開發過程較容易且租戶的資料安全性也較高，但是，由於每個租戶都要負擔費用去建置、維護硬體設備，所以，在經濟效益上相對於其它兩種技術來得低；採用 Separate Schemas 以及 Shared Schema 雖在初期的開發過程需要較長的時間且需提供額外的機制去加強租戶資料的隔離性，但是，由於租戶間資源共享程度較高，所以，可以降低每個租戶所需負擔的費用。

2.1.2 Separate Databases

Separate Databases 主要是將每個租戶的資料存在放在各自的獨立資料庫中，以圖 2.2 為例，假定有三間學校(租戶)，分別為 Nccu、Fju 以及 Tku，每間學校皆有三個資料表，分別是 StudentInfo(儲存該校學生資料)、SelectCourse(儲存該校學生選課記錄)以及 CourseInfo(儲存該校課程資料)，每間學校提供一個獨立的資料庫，讓每間學校可以各自將自己的資料存放在各自的獨立資料庫。

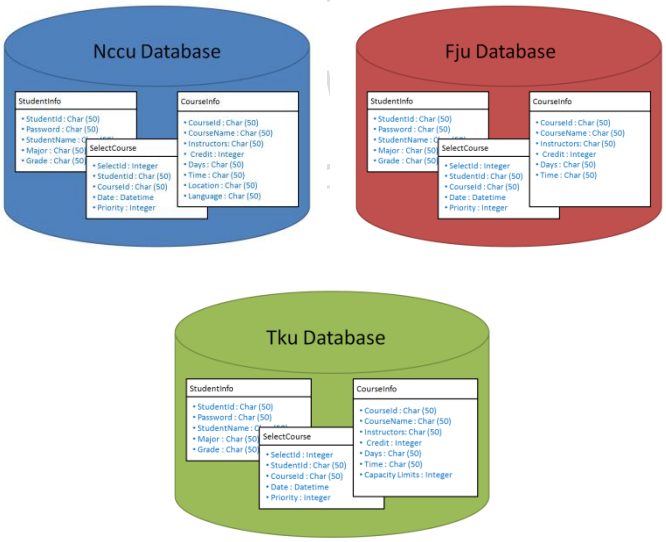

圖 2.2 Separate Databases 示意圖:各個 Tenant 的資料存放在各自獨立的 Database

因為每個租戶各自擁有一個資料庫來存放自己的資料，所以，可以達到很高的資料安全性,資料表綱要(schema)也較容易依照各個租戶的需求來做異動，客製化的程度較高。除此之外，資料一旦損壞或遺失，單一租戶的還原資料(restore data)的程序也比較容易，只需要獲得該租戶的備份資料(back-up data)來進行還原資料(restore data)的動作。但相對地，以經濟效益的層面看來，採用這種方法的租戶(例如:銀行、醫院)主要是為了達到高度的資料安全性(銀行交易資料、醫院病歷資料)以及客製化，所以，需要負擔較高額的費用去提供硬體設備以及往後的設備維護[Chong et al.06]。

由於採用這種方法必需為每個租戶提供一套硬體設備，因此，在有限的硬體設備下只能提供給有限數量的租戶，不符合經濟效益上的期望。在接下來的小節，將提出兩種方法(Separate Schemas、Shared Schema)可以透過技術來突破有限的硬體環境，實現多租戶的理想。

2.1.3 Separate Schemas

Separate Schemas 主要是將每個租戶的資料存放在同一個的資料庫中，同時，每個租戶又各自擁有各自的資料表集合,不同的租戶之間,資料表綱要(schema)也不完全相同。以圖 2.3 為例，假定有三間學校(租戶)，分別為 Nccu、Fju 以及 Tku，每間學校可以根據自己的需求去制訂屬於自己的資料表綱要(schema)以及資料表集合存放在同一個資料庫裡。從圖 2.3 的紅色方框部份可以發現 Nccu 的 CourseInfo 資料表綱要和另外兩間學校的 CourseInfo 資料表綱要有明顯的不同，這就是本研究一直提到的重點：租戶可以擁有客製化欄位的能力。

由於各個租戶都擁有各自的資料表集合，所以，仍然可以根據自己的需求制定資料表綱要(schema)。但還是有一些缺點，一旦有一個租戶的資料表毀損或是資料遺失,還原資料(restore data)的程序就會比採用 Separate Databases

來得複雜，必須利用備份資料(back-up data)對整個資料庫進行還原資料(restore data)的動作才能解決問題[Chong et al.06]。

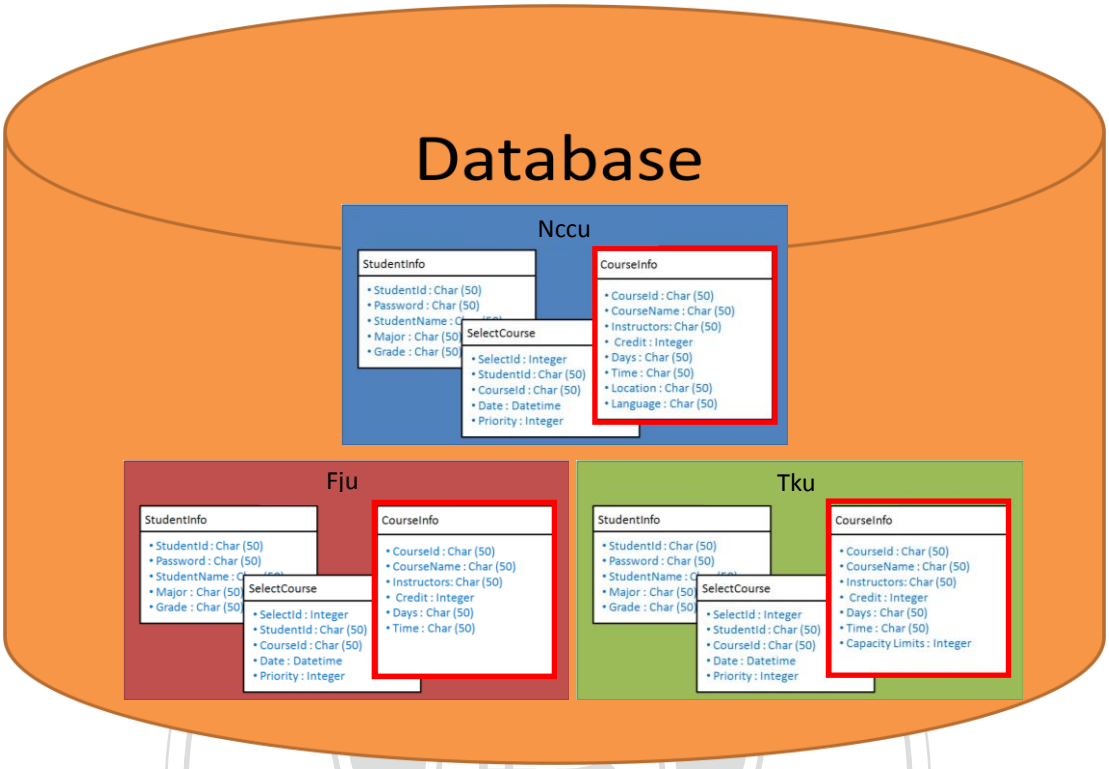

圖 2.3 Separate Schemas 示意圖:各個租戶將各自不同的資料表綱要以及資料表集合存放在同一個資料庫

此外，這種方法將所有租戶的資料都存放在同一個資料庫中，因此，不同租戶間的資料隔離程度降低，同時，資料的安全性也相對降低，所以，”如何設計好的隔離機制以防止租戶的資料遭到攻擊”變成了一個很重要的議題。

在有限的硬體設備下，和採用 Separate Databases 方法相比，這種方法能夠提供服務給較多的租戶，相對地，每個租戶所需要負擔的費用也大幅降低，但有一個重要的前提是租戶要能夠了解並且同意它的資料和其它租戶的資料是放在同一個資料庫中；軟體開發人員也要事先將一些安全上的顧慮告知租戶並且提供相對應的隔離機制來有效地提升資料的安全性。

2.1.4 Shared Schema

這種方法是利用資料庫裡面的單一資料表去存放所有租戶的資料，並將資料依序存放在共用表格欄位（Col1、Col2、Col3、Col4…）中，利用 TenantId 以及 Table 這兩個欄位作為依據去區分不同的租戶的資料(record)。以圖 2.4 為例，假定有三間學校(租戶)，分別為 Nccu、Fju 以及 Tku，其相對應的 TenantId 即以其學校英文代號表示(即 Nccu、Fju 以及 Tku)，利用 Table 欄位(1、2、3…)去區分不同的資料表，例如：同樣 TenantId 為 Nccu，但 Table 欄位不同(1、2)則代表兩個同租戶但存放不同資料的資料表(圖 2.4 中綠色及藍色方框)。共用欄位(Col1、Col2、Col3、Col4…)則依照每個租戶自訂的資料表綱要(schema)去依序存放每一筆紀錄(record)。

TanantId	Table	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8
Nccu	1	Nccu1	軟體工程	陳恭	3	Mon	D56	大仁3301	中文
Nccu	1	Nccu2	等候理論	蔡子傑	3	Fri	123	大仁1103	中文
Fju	1	Fju1	財務報表分析	林昶佑	2	Tue	234	空	空
Fju	1	Fju2	租稅各論	蔡麗雯	3	Wed	234	空	空
Tku	1	Tku1	微積分	陳功宇	3	Wed	234	70	空
Tku	1	Tku2	多媒體系統	郭經華	3	Tue	567	30	空
Nccu	2	99753020	Kevin1234	李明憲	資訊科學系	碩二	空	空	空
Nccu	2	100753025	19871222	林子文	資訊科學系	碩一	空	空	空

圖 2.4 Shared Schema 示意圖:各個租戶將各自的資料表存放在同一個資料庫的單一資料表中

由於所有的租戶資料都存放在同一個資料表中，所以，可支援的租戶數目增加了，因此，和上述兩種方法相比，租戶所需負擔的費用更低。但同時，資料的安全性就降低許多，如果沒有好的隔離機制去區分租戶，很容易會讓租戶的資料遭受攻擊。除此之外，還需要額外提供資料表去紀錄每個租戶所有資料表的詮釋欄位(meta-data)，以協助之後執行資料庫存取動作。另外，還原資料(restore data)的部份，一旦有資料毀損或是遺失，則需要將整個資料庫進行還原資料(restore data)的動作才能解決問題[Chong et al.06]。

相對於前兩種方法，採用 Shared Schema 方法更能充分地利用硬體設備，簡單來說，也就是在有限的硬體設備下，提供給相較於上述所有方法中最多的

租戶數量，但同樣地，前提是租戶必需要能夠同意它的資料和其它租戶的資料是放在同一個資料表中，此外，軟體開發人員也要事先將一些安全上的顧慮告知租戶並且提供相對應的隔離機制來有效地提升租戶間的資料安全性。

2.1.5 多租戶應用程式之資料層級設計考量

上述的三種多租戶應用程式之資料層級設計在經濟效益以及開發層面各有優缺點，所以，必需先了解租戶的需求並且在經濟效益上做適當的評估，在[Chong et al.06]也列出以下三項考量因素：

1. **經濟效益的考量：**依據經濟效益的考量，在一定數量的硬體設備下，採用 Shared Schema 可以提供服務給最多數量的租戶，但是，軟體開發人員必需事先將一些安全上的顧慮告知租戶並且提供相對應的隔離機制來有效地提升資料的安全性。
2. **資料的安全性：**當租戶資料是屬於需要較高的安全性(例如:醫療紀錄、銀行交易資料)的資料，則採用 Separate Databases 會較為適當。但事實上，如果採用 Shared Schema、Separate Schemas，額外搭配一些隔離機制，也是可以提供高度的資料安全性。
3. **租戶的數量以及需求：**租戶的數量、每個租戶的資料量大小都會影響到多租戶應用程式之資料層級設計。如圖 2.5 所示，若租戶的數量很多，則採用 Shared Schema 會比較節省成本；若每個租戶所存放的資料量很大，則採用 Separate Databases 的方式或許比較合適。

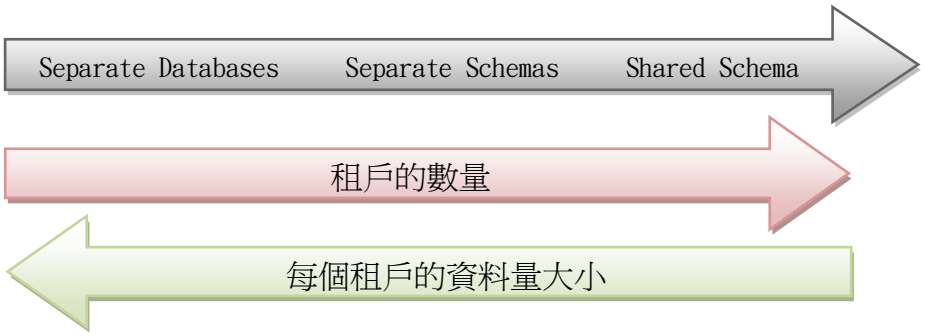

圖 2.5 租戶的數量以及每個租戶的資料量大小影響圖

2.2 多租戶應用程式之資料表綱要類型

目前實作多租戶應用程式之資料層級設計有兩種方式。一租戶一資料庫的方式在成本上極為浪費，無法達到資源共享的理念；多租戶共享一資料庫的方式雖然符合資源共享的理念，但缺乏擴充性，所有的租戶都必需使用同一種資料表綱要(Schema)，無法達到客製化的理想。所以，在接下來的各個小節中，將在上述二種方式中找到一個平衡點，從資料表綱要設計的面向，提供多種既可讓租戶共享資料庫又可滿足租戶適度地客製化欄位需求的資料表綱要類型。

2.2.1 Private Table Layout

假設目前有三間學校(租戶)： Nccu、Fju、Tku，圖 2.6 是每間學校都要使用的一個資料表：CourseInfo。所有學校的 CourseInfo 資料表都有 CourseId、CourseName、Instructors、Credit、Days、Time 這六個共有欄位，但也有各間學校各自擁有的私有欄位(即客製化欄位)以表達客製化欄位的需求。

NccuCourseInfo

CourseId	CourseName	Instructors	Credit	Days	Time	Location	Language
Nccu1	軟體工程	陳恭	3	Mon	D56	大仁3301	中文
Nccu2	等候理論	蔡子傑	3	Fri	123	大仁1103	中文

FjuCourseInfo

CourseId	CourseName	Instructors	Credit	Days	Time
Fju1	財務報表分析	林昶佑	2	Tue	234
Fju2	租稅各論	蔡麗雯	3	Wed	234

TkuCourseInfo

CourseId	CourseName	Instructors	Credit	Days	Time	Capacity Limits
Tku1	微積分	陳功宇	3	Wed	234	70
Tku2	多媒體系統	郭經華	3	Tue	567	30

圖 2.6 基礎範例(Private Table Layout 示意圖)：

三間學校(租戶)各別擁有不同的 CourseInfo 資料表，有共有欄位，

同時，也有各別租戶所需要的私有欄位以表達客製化欄位需求

Private Table Layout [Aulbach et al.08]，此種方法是將不同租戶的資料放置在不同的資料表，這是最常見、最簡單的多租戶應用程式之資料表綱要設計。主要優點是租戶客製化資料表綱要的程度高，缺點是當租戶的數量一增加，其資料表的數量也隨之增加，且無法達到資源共享的理想。後面的章節將以此範例為基礎，介紹其他三種多租戶資料表綱要的設計技巧。

2.2.2 Universal Table Layout

Universal Table Layout [Aulbach et al.08]，此種方法是將所有租戶的資料放置於一個共用資料表中，同時，為了支援租戶間資料欄位的差異，它是採稀疏矩陣的方式來存放資料。具體而言，共用資料表的資料欄位是所有租戶的聯集，簡單以流水號命名(Col1、Clo2、…)，並以新增的 TenantId 欄位與 Table 欄位(圖 2.2 中紅色方框)來區分不同租戶的資料。圖 2.7 以上述的基礎範例來展示這種作法。

CourseInfoUniversal

TanantId	Table	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8
Nccu	1	Nccu1	軟體工程	陳恭	3	Mon	D56	大仁3301	中文
Nccu	1	Nccu2	等候理論	蔡子傑	3	Fri	123	大仁1103	中文
Fju	1	Fju1	財務報表分析	林昶佑	2	Tue	234	空	空
Fju	1	Fju2	租稅各論	蔡麗雯	3	Wed	234	空	空
Tku	1	Tku1	微積分	陳功宇	3	Wed	234	70	空
Tku	1	Tku2	多媒體系統	郭經華	3	Tue	567	30	空

圖 2.7 基礎範例轉換成 Universal Table Layout 示意圖

Universal Table Layout[Aulbach et al.08]的主要優點是無論多少個租戶都只需要一個資料表以滿足多租戶共用資料庫的基本需求。但因為不同租戶間的資料差異，所以，也造成一些缺點：(1)因為不同租戶間的資料差異，部份欄位會有浪費的情況發生，圖 2.7 中綠色方框即為此類欄位。(2)因為不同租戶間的資料差異，部份資料欄位的資料會有異質性型別的情況(例如，Col7 有字串

也有數字)。(3)需要額外的 meta-data 去記錄欄位名稱。

2.2.3 Extension Table Layout

Extension Table Layout[Aulbach et al.08]，此種作法是將租戶共有的欄位提出放置於一個資料表，並以新增的 TenantId 欄位與 Row 欄位(圖 2.8 中紅色區塊)來區分不同租戶的資料。其它個別租戶私有的資料欄位(user column)則分別寫入不同的資料表中。圖 2.8 以上述的基礎範例來展示這種作法，其中欄位 Row 是用來標記表格內資料紀錄(record)的編號。

CourseInfoCommonFields

TenantId	Row	CourseId	CourseName	Instructors	Credit	Days	Time
Nccu	1	Nccu1	軟體工程	陳恭	3	Mon	D56
Nccu	2	Nccu2	等候理論	蔡子傑	3	Fri	123
Fju	1	Fju1	財務報表分析	林昶佑	2	Tue	234
Fju	2	Fju2	租稅各論	蔡麗雯	3	Wed	234
Tku	1	Tku1	微積分	陳功宇	3	Wed	234
Tku	2	Tku2	多媒體系統	郭經華	3	Tue	567

NccuCourseInfo

TenantId	Row	Location	Language
Nccu	1	大仁3301	中文
Nccu	2	大仁1103	中文

TkuCourseInfo

TenantId	Row	Capacity Limits
Tku	1	70
Tku	2	30

FjuCourseInfo

TenantId	Row
Nccu	1
Nccu	2

圖 2.8 基礎範例轉換成 Extension Table Layout 示意圖

Extension Table Layout[Aulbach et al.08]的主要優點是透過共有欄位的集中存放，可以統一管理共有欄位的部分。但也存在著一些缺點：(1)共用資料表中需額外增加一些詮釋欄位(meta-data)來區分不同的租戶，圖 2.8 中紅色方框即代表此資料表的詮釋欄位。(2)讀取資料的時候，需要執行較費時的 JOIN 指令。(3)跟 Private Table Layout 的作法一樣，仍然有資料表數量隨租戶數量增加而增加的問題。

2.2.4 Pivot Table Layout

Pivot Table Layout[Aulbach et al.08]，此種作法是將所有租戶的資料依其資料型別來分別存放在不同的資料表，即同一型別的資料放置於同一個資料表，並以新增的 TenantId 欄位、Table 欄位、Col 欄位以及 Row 欄位來區分不同租戶的資料。其餘的欄位用來放置實際的租戶資料。圖 2.9 以上述的基礎範例來展示這種作法。

PivotInt

TanantId	Table	Col	Row	Int
Nccu	1	1	1	3
Nccu	1	1	2	3
Fju	1	4	1	2
Fju	1	4	2	3
Tku	1	4	1	3
Tku	1	7	1	70
Tku	1	4	2	3
Tku	1	7	2	30

PivotStr

TanantId	Table	Col	Row	Str
Nccu	1	1	1	Nccu1
Nccu	1	2	1	軟體工程
Nccu	1	3	1	陳恭
Nccu	1	5	1	Mon
Nccu	1	6	1	D56
Nccu	1	7	1	大仁3301
Nccu	1	8	1	中文
Nccu	1	1	2	Nccu2
Nccu	1	2	2	等候理論
Nccu	1	3	2	蔡子傑
Nccu	1	5	2	Fri

TanantId	Table	Col	Row	Str
Nccu	1	6	2	123
Nccu	1	7	2	大仁3301
Nccu	1	8	2	中文
Fju	1	1	1	Fju1
Fju	1	2	1	財務報表分析
Fju	1	3	1	林昶佑
Fju	1	5	1	Tue
Fju	1	6	1	234
Fju	1	1	2	Fju2
Fju	1	2	2	租稅各論
Fju	1	3	2	蔡麗雯

圖 2.9 基礎範例轉換成 Pivot Table Layout 示意圖

Pivot Table Layout[Aulbach et al.08]的主要優點是加入了資料型別的處理，同時，也不會有欄位浪費的情況發生。但共用資料表中需額外增加不少的詮釋欄位（meta-data）來區分不同的租戶，圖 2.9 中紅色方框即代表此資料

表的詮釋欄位。

2.3 語句結構剖析工具

在 SQL 語句轉換成以 Extension Table Layout 邏輯表達的 SQL 語句過程中，必需先取得 SQL 語句中的語句元素，以圖 2.10 為例，其 UPDATE 語句是由資料表名稱(CourseInfo)、更新欄位以及其更新欄位數值(Days = 'Tue', Time = 'D56')、WHERE 條件式(CourseName = '軟體工程')這三項語句元素所構成的。本研究將採用 JSqlParser 來協助系統工具進行語句結構剖析的動作。

JSqlParser 可以在 Java 程式中進行 SQL 語句的結構剖析並將其結構內容轉換成一種階層式的 Java 類別，並以 vistor pattern 的方式去操作這些 Java 類別。下圖 2.10 是一個 JSqlParser 對 SQL 語句進行剖析的程式碼範例。

```
1. String sql = "UPDATE CourseInfo SET Days = 'Tue' , Time
   = 'D56' WHERE CourseName = '軟體工程'";
2. CCJSqlParserManager pm = new CCJSqlParserManager();
3.     //進行剖析的動作
4. Update updatestmt = (Update) pm.parse(new
   StringReader(sql));
5.     //取得JSqlParser剖析後的資料
6.     //取得更改的欄位資訊，Ex：[ Days , Time ]
7. List columns = updatestmt.getColumns();
8.     //取得更改欄位其數值的資訊，Ex：[ 'Tue' , 'D56' ]
9. List expressions  = updatestmt.getExpressions();
10.    //取得更改的資料表名稱，Ex：CourseInfo
11.Table table = updatestmt.getTable();
12.    //取得Where條件式的內容，Ex：CourseName = '軟體工程'
13.Expression where = updatestmt.getWhere()
```

圖 2.10 JSqlParser 對 SQL 語句進行剖析的程式碼範例

以圖 2.10 程式碼為例，圖中名為 sql 的 String 物件是即將進行結構剖析的 SQL 語句(line 1)，首先，建立一個名為 pm 的 CCJSqlParserManager 物件(line 2)。接下來，藉由 parse()函式對名為 sql 的 String 物件進行語句

結構剖析，同時，將剖析後獲得的語句元素存放在名為 updatestmt 的 Update 類別物件中(line 4)。最後，透過 getColumn()、getExpressions()、getTable()以及 getWhere()這四個 getter 函式(line 7-13)來取得所需的語句元素，包含更改欄位(例如：[Days,Time])、更改欄位數值(例如：['Tue', 'D56'])、資料表名稱(例如：CourseInfo)以及 Where 條件式的內容(例如：CourseName='軟體工程')，並將取得的資訊儲存在 columns、expressions、table 以及 where 這四個變數中以協助後續的語句轉換動作。

國立政治大學 National Chengchi University

第三章

系統設計

基於多租戶技術的概念，在租戶資源共享的情況下，也要讓租戶以自助的方式去客製化其資源，所以，本研究在租戶共享資料庫的前提下，以提供租戶適度地客製化其資料表綱要的能力納入主要的設計考量，另外，由於採用 Extension Table Layout 的概念進行設計，因此，也可以提升軟體開發人員統一管理租戶的共有欄位以及其資料的能力。本系統工具透過改寫 JDBC 的部分 API 去攔截軟體開發人員所撰寫的 SQL 語句；接下來，將攔截到的 SQL 語句交給 JSqlParser 進行結構剖析以獲得語句元素；再來，重組獲得的語句元素，將 SQL 語句從一租戶一資料表寫法轉換成以 Extension Table Layout 邏輯表達的 SQL 語句；最後，將轉換後的 SQL 語句在資料庫中執行以進行資料表綱要的建立、更改或是資料的增刪修改以及查詢等動作。本章節將針對系統設計的部分進行詳細地解說，包含了系統設計理念、系統設計考量、系統流程以及系統實作方法。

3.1 系統設計理念

現今的多租戶應用程式之資料層級大部份是以 Private Table Layout 的概念去設計，雖然可以提供租戶客製化其資料表綱要的能力，但此種方式容易造成軟體開發人員在多租戶應用程式維運上的不便，倘若開發一個新服務的過程中，需要修改已存在的資料表綱要，則軟體開發人員要逐一針對每個租戶的資料表綱要(schema)進行修改，當租戶的數量或是受影響的資料表綱要(schema)逐漸增加，這無非是一個很費時的工程。相較之下，採用 2.2.3 小節 Extension Table

Layout 的概念更符合多租戶應用程式之資料層級的需求，其所提出的共有欄位、私有欄位概念可以提升軟體開發人員統一管理租戶的共有欄位以及其資料的能力；除此之外，亦可以讓租戶建立私有欄位，也意味著滿足租戶適度地客製化欄位的需求。

但是，當資料以 Extension Table Layout 的資料存放方式儲存，會讓軟體開發人員在撰寫 SQL 語句時，為了配合實際資料存放方式而無法採用一租戶一資料表寫法撰寫 SQL 語句，造成軟體開發人員在開發上的困難。

為了解決上述採用 Extension Table Layout 的概念對軟體開發人員所造成的困難，本研究實作一套系統工具將 SQL 語句從一租戶一資料表寫法轉換成以 Extension Table Layout 邏輯表達的 SQL 語句，讓軟體開發人員可以沿用一租戶一資料表的 SQL 語句邏輯對以 Extension Table Layout 的資料存放方式儲存的資料進行增刪修改。

因此，在應用程式的部分，軟體開發人員透過 JDBC 的 API 以標準的 SQL 語句邏輯撰寫 SQL 語句來表達其所要對資料庫進行的動作，同時，設定 TenantId 讓本系統工具知道其所要執行 SQL 語句的對象(租戶)。透過語句轉換機制的處理，產生以 Extension Table Layout 邏輯表達的 SQL 語句並在資料庫上執行，完成軟體開發人員原先所要達到的目的。(圖 3.1)

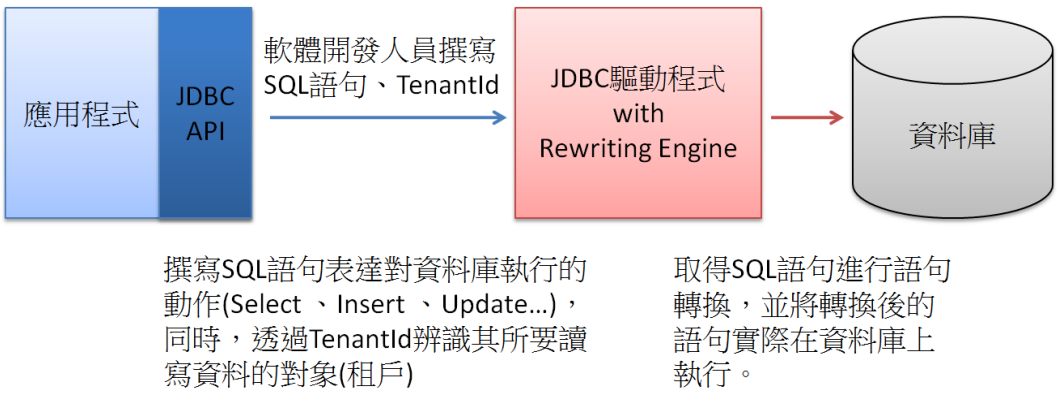

圖 3.1 採用本系統工具的運作模式

3.2 系統設計考量

在進行多租戶應用程式之資料層級設計時需要經過多方面的考量，除了軟體開發人員利用共享資源的概念，在有限的資源設備下，提供服務給更多的租戶之外，站在租戶以及軟體開發人員的角度也有其他的考量要素，對租戶而言，應該要讓租戶能夠適度地客製化其資料表綱要；對軟體開發人員而言，應該要提升軟體開發人員統一管理租戶的共有欄位以及其資料的能力。接下來，會針對這兩點進行細部的探討：

1. **提供租戶適度地客製化其資料表綱要的能力：**

實際開發一個多租戶應用程式，在資料表的部份，除了存在軟體開發人員所建立的共有欄位以外，提供租戶根據其需求在其資料表中新增、修改以及刪除私有欄位(客製化欄位)是一個很重要的需求，畢竟，每個租戶的資料表綱要(schema)不盡相同。採用 Extension Table Layout 其所提出的私有欄位概念讓租戶針對各自的需求新增、修改以及刪除其獨有的私有欄位為本研究所強調的”提供租戶適度地客製化其資料表綱要的能力”。

2. **提升軟體開發人員統一管理租戶的共有欄位以及其資料的能力：**

這裡，先定義甚麼是統一管理租戶的共有欄位以及其資料的能力，如同 1.2 小節所提到的，站在軟體開發人員的角度，由於在開發階段無法預測租戶會建立哪些私有欄位(即客製化欄位)，只曉得共有欄位的存在，所以，只會針對共有欄位進行多租戶應用程式的系統功能設計，因此，如果能提升軟體開發人員統一管理租戶的共有欄位以及其資料的能力，則可以方便軟體開發人員進行開發以及維運。如同上一章節所敘述的，現今的多租戶應用程式之資料層級大部份是以 Private Table Layout 的概念去進行設計，此種開發方式雖然直覺、簡單，但當租戶的數量逐漸增加，資料表的數量也隨之增加，此時，軟體開發人員在多租戶應用程式的維運上需要花費更多的成本。雖然 Extension Table

Layout 無法減少租戶數量增加，資料表數量增加的問題，但其所提出的共有欄位、私有欄位的概念，可以提供一套有效管理方式讓軟體開發人員能夠統一對所有租戶資料表中的共有欄位進行增刪修改，因此，降低軟體開發人員在多租戶應用程式開發以及維運上所需花費的時間成本。

3.3 系統流程

關於系統流程，主要可以分成三個階段(圖 3.2)，首先，透過改寫 JDBC 的部分 API 去攔截軟體開發人員所撰寫的 SQL 語句，接下來，將 SQL 語句經由 JSqlParser 進行結構剖析，再來，從其剖析結果取得語句元素以協助系統工具以 Extension Table Layout 的邏輯進行語句轉換，最後，將轉換後的語句回傳至 JDBC 並在資料庫中執行，影響各別租戶的資料表綱要(schema)或是資料。

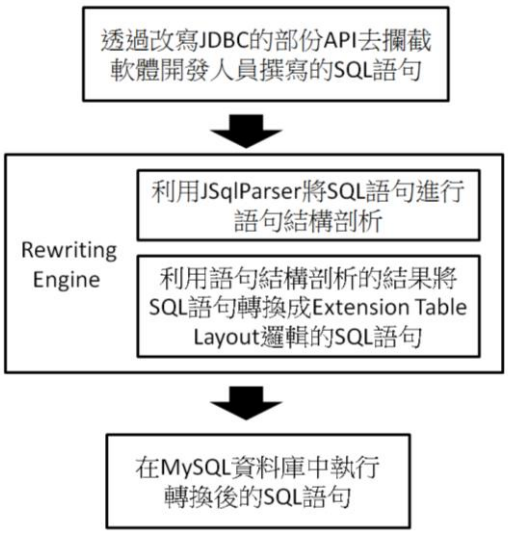

圖 3.2 系統工具進行語句轉換流程圖

第一階段，如圖 3.3 所示，建立一個新類別－Multitenantstatement，此類別去繼承原先在 JDBC 中扮演實際執行 SQL 語句以及回傳查詢結果角色的 Statement 類別(此 Statement 類別已實作 JDBC 中的 Statement 介面)。在 Multitenantstatement 類別中，基於 1.2 小節所提到的多租戶技術的開發概念，

除了既有的類別成員之外，額外增加了一個類別成員－TenantId，透過此類別成員(TenantId)去辨識此 SQL 語句的執行對象(租戶)。

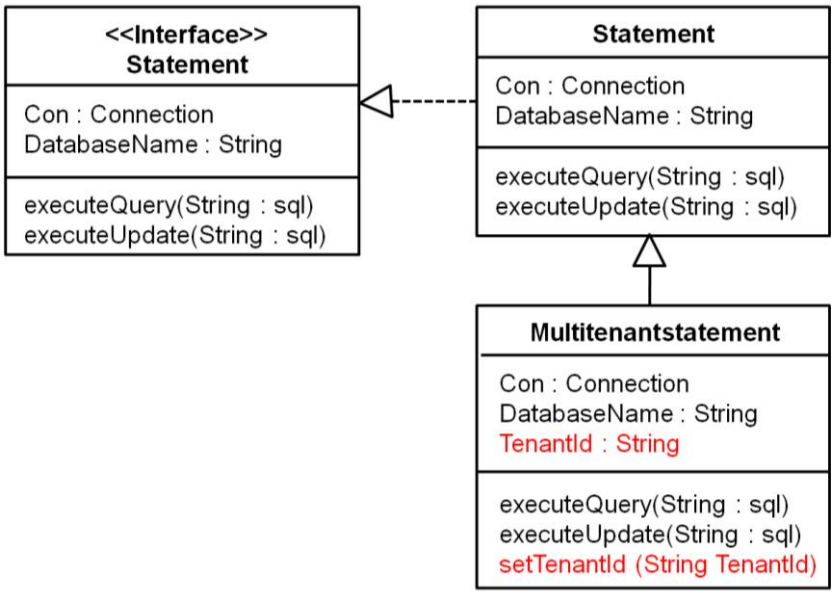

圖 3.3 Multitenantstatement 類別繼承關係圖：

藉由此類別協助系統工具取得租戶的識別碼－TenantId

```
1. Class.forName("com.mysql.jdbc.Driver");
2.    //建立與database的連線
3. Connection con = (Connection)
   DriverManager.getConnection("jdbc:mysql://localhost/Exte
   nsionTable?useUnicode=true&characterEncoding=utf-
   8","root","1234");
4.    //建立新的Multitenantstatement物件
5. Multitenantstatement stmt = new
   Multitenantstatement(con,"ExtensionTable");
6. String sql = "UPDATE CourseInfo SET Days = 'Tue' , Time
   = 'D56' WHERE CourseName = '軟體工程'";
7. stmt.setTenantId('Nccu');
8. stmt.executeUpdate(sql);
```

圖 3.4 系統工具攔截軟體開發人員所撰寫的 SQL 語句程式碼範例

如何攔截到軟體開發人員所撰寫的 SQL 語句。站在軟體開發人員的角度，以圖 3.4 為例，首先，會先建立與資料庫的連線(line 1-3)，緊接著，建立一個新的 Multitenantstatement 物件取代原先 Statement 物件的位置，扮演實際

執行 SQL 語句以及回傳查詢結果的角色(line 5)，接下來，撰寫其所要執行的 SQL 語句(line 6)，最後，利用 setTenantId()設定執行的對象(租戶)，透過 executeUpdate()去執行 SQL 語句。

從圖 3.4 以及圖 3.5 清楚知道系統工具藉由 executeUpdate()攔截到軟體開發人員所撰寫的 SQL 語句(圖 3.4 的 line 8)，接下來，將 SQL 語句利用 If-Else 去判斷其語句型態(圖 3.5 的 line 6)，根據判斷結果將 SQL 語句進行不同型態的語句轉換。

第二階段，將 SQL 語句轉換成以 Extension Table Layout 邏輯表達的 SQL 語句之前，必需先透過 JSqlParser 去進行語句結構剖析，從其剖析結果獲得進行語句轉換時所需的語句元素。最後，將取得的語句元素以 Extension Table Layout 邏輯進行重組。

第三階段，將上一階段轉換後所產生的語句傳回 executeUpdate()並在資料庫中執行，影響個別租戶的資料表綱要(schema)或是資料。

```
1. public java.sql.ResultSet executeUpdate(String sql)
   throws SQLException {
2.     /*由於回傳改寫後的SQL語句不只一句，所以，必需存放在List中，
3.       並利用一個Count變數去記錄改寫後的SQL語句句數*/
4.     int Count = 0
5.     List sqlList = new ArrayList();
6.     if(sql.toUpperCase().startsWith("INSERT"))
7.     {
8.          /*進行INSERT語句改寫
9.            回傳改寫後的SQL語句*/
10.    }
11.    else if(sql.toUpperCase().startsWith("UPDATE"))
12.   {
13.         /*進行UPDATE語句改寫
14.           回傳改寫後的SQL語句*/
15.   }
```

圖 3.5 executeUpdate()的虛擬碼

3.4 系統實作方法

在上一章節說明了系統工具如何利用改寫 JDBC 的部分 API 去攔截軟體開發人員

所撰寫的 SQL 語句，進而將 SQL 語句交給 JSqlParser 進行結構剖析以取得語句轉換時所需的語句元素。緊接著，本章節將詳細地解釋如何依據 Extension Table Layout 邏輯進行各種型態的語句轉換。

3.4.1 Extension Table Layout 概念實作

在 2.2.3 小節中提到 Extension Table Layout 的概念是將所有租戶的共有欄位提出後放置在一個共用資料表中，同時，所有租戶屬於共有欄位的欄位資料也都存放在該共用資料表中。另外，為了滿足租戶客製化欄位的需求，替每個租戶建立其存放私有欄位(即客製化欄位)資料的私有資料表。接下來的部分，將了解如何實現 Extension Table Layout 所提出的共有欄位、私有欄位概念在多租戶應用程式之資料層級設計。

在多租戶應用程式的開發過程中，資料層級的部分，軟體開發人員會依照應用程式的需求設計一套資料表集合，以開發一個多租戶(此範例的租戶定義為學校)選課系統為例，假定目前有 Nccu、Fju 以及 Tku 三間學校(租戶)使用這個多租戶選課系統，根據 Extension Table Layout 所提出的概念，首先，軟體開發人員會透過 CREATE 語句去建立三個存放所有學校共有欄位資料的共用資料表，分別是用來存放所有學校學生共有欄位資料的 StudentInfoCommonFields、所有學校課程共有欄位資料的 CourseInfoCommonFields 以及記錄所有學校學生選課記錄共有欄位資料的 SelectCourseCommonFields。接下來，為了讓每個學校(租戶)可以依據各自需求適度地客製化其資料表綱要(schema)，本系統工具增加了一個名為 CREATE EXTENSION TABLE 的語句去協助軟體開發人員建立存放學校(租戶)其私有欄位(客製化欄位)資料的私有資料表集合，另外，藉由 ALTER 語句讓每個學校(租戶)對其所擁有的私有資料表綱要(schema)進行欄位的增刪修改，此一動作意味著私有欄位的增刪修改，同時，也代表著租戶擁有客製化欄位的

能力。

除此之外，由於 Extension Table Layout 會將資料表欄位拆成共有欄位、私有欄位，共有欄位的資料存放在共用資料表中；私有欄位的資料則存放在各租戶各自擁有的私有資料表中。為了協助系統工具進行後續的語句轉換，所以，必須額外記錄各個共用資料表、私有資料表的欄位名稱資訊(不包含詮釋欄位名稱)在 Columns_Metadata Table 中。

圖 3.6 是 Columns_Metadata Table 的欄位資訊存放格式。Columns_Metadata Table 主要記錄了實際在資料庫中，每個資料表所存在的欄位名稱，但不包含詮釋欄位名稱(TenantId、Row)。

```
1  CourseInfoCommonFields, CourseId, CourseName, Instructors, Credit, Days, Time
2  StudentInfoCommonFields, StudentId, StudentName, Password, Major, Grade
3  SelectCourseCommonFields, SelectId, StudentId, CourseId, SelectDate, Priority
4  NccuCourseInfo, Location , Language
5  NccuStudentInfo
6  NccuSelectCourse
7  FjuCourseInfo
8  FjuStudentInfo
9  FjuSelectCourse
10 TkuCourseInfo, CapacityLimits
11 TkuStudentInfo
12 TkuSelectCourse
13
```

圖 3.6 Columns_Metadata Table 的欄位資訊存放格式：每一列以實際資料表名稱開頭，以逗點分隔逐一記錄其所存在的欄位名稱(不包含詮釋欄位：TenantId、Row)

3.4.2 CREATE 語句的轉換、CREATE EXTENSION TABLE 的設計與實作

為了實現共有欄位的概念，軟體開發人員可以撰寫 CREATE 語句建立存放所有租戶共有欄位資料的共用資料表；另外，CREATE EXTENSION TABLE 語句則是協助軟體開發人員建立每個租戶各自擁有的私有資料表集合。

在進行共用資料表建立之前，必需先對共用資料表、私有資料表的命名方式作規定以方便系統工具能夠透過資料表名稱來分辨該資料表屬於共用資料表

或是私有資料表。本系統工具中統一共用資料表名稱必需以”CommonFields”這個關鍵字作命名的結尾。假定建立一個存放學校課程資訊共有欄位資料的共用資料表，可以取課程資訊的英譯”CourseInfo”加上”CommonFields”關鍵字，將此共用資料表命名為”CourseInfoCommonFields”；相對地，私有資料表的命名方式則是加上 TenantId 以供系統工具識別。(表 3.1)

表 3.1 共用資料表與私有資料表比較圖

資料表種類	共用資料表	私有資料表
定義	提供給每個租戶存放共有欄位的資料表	提供給每個租戶存放私有欄位的資料表
特色	讓軟體開發人員可以透過共用資料表的修改來統一影響各租戶的共有欄位，減少逐一修改各租戶資料表的機會以降低開發和維運上的時間成本。	滿足租戶客製化資料表綱要的需求。透過私有資料表綱要的修改達到租戶新增、刪除客製化(私有)欄位或是更改客製化(私有)欄位名稱、資料型態等動作的需求。
資料表命名方式	資料表名稱以 CommonFields 結尾	資料表名稱以 TenantId 開頭
例子	StudentInfo**CommonFields**、SelectCourse**CommonFields**、CourseInfo**CommonFields**…	**Nccu**StudentInfo、**Nccu**SelectCourse、**Nccu**CourseInfo…

A. CREATE 語句的轉換

這一小節，主要探討如何進行 CREATE 語句的轉換，共有三個步驟：

1. 透過改寫 JDBC 的部分 API 去攔截軟體開發人員所撰寫的 SQL 語句。
2. 接下來，將攔截到的 SQL 語句藉由 JSqlParser 進行語句結構剖析的動作，從其結果獲得轉換語句所需要的語句元素。
3. 以 Extension Table Layout 的邏輯進行語句轉換。

以圖 3.7 為例，經由改寫 JDBC 的部分 API 去攔截軟體開發人員所撰寫的 SQL 語句後(line 7)，透過 JSqlPaeser 進行語句結構剖析以獲得語句元素，，例如：資料表名稱(此為 CourseInfoCommonFields)以及欄位宣告的部份(此為

CourseId Char(50) ,CourseName Char(50)…)。最後，如圖 3.8 紅字部分，在原 SQL 語句中欄位宣告的部分加入了詮釋欄位(TenantId、Row)的宣告，同時，將詮釋欄位設置欄位限制為 Not Null；另外，增加一個 Composite Primary Keys 以確保(TenantId ， Row)的組合會是唯一。

```
1. Class.forName("com.mysql.jdbc.Driver");
2.    //建立與database的連線
3. Connection con = (Connection)
   DriverManager.getConnection("jdbc:mysql://localhost/Exte
   nsionTable?useUnicode=true&characterEncoding=utf-
   8","root","1234");
4.    //建立新的Multitenantstatement
5. Statement stmt = new
   Multitenantstatement(con,"ExtensionTable");
6. String sql =
   "CREATE TABLE CourseInfoCommonFields
     (CourseId Char (50) , CourseName Char(50) ,
      Instructors Char (50) , Credit Integer ,
      Days Char(50) , Times Char (50))";
7. stmt.executeUpdate(sql);
```

圖 3.7 CREATE 語句範例

```
CREATE TABLE CourseInfoCommonFields
( TenantId Char (50) not null,
  Row Integer not null,
  CourseId Char (50) ,
  CourseName Char(50) ,
  Instructors Char (50) ,
  Days Char(50) ,
  Times Char (50)
  Credit Integer )
  Primary key (TenantId, Row))
```

圖 3.8 CREATE 語句轉換後的結果

隨著共用資料表的建立，系統工具也會自動更新 Columns Meta data Table，新增一筆記錄共用資料表的欄位資訊在 Columns_Metadata Table(圖 3.9)。

```
1  CourseInfoCommonFields, CourseId, CourseName, Instructors, Credit, Days, Time
```

圖 3.9 執行 CREATE 語句後，Columns_Metadata Table 的更新結果

一般來說，欄位宣告有四種類型的 Integrity Constraints[21]，除了最基本的 Domain Constraints 以外(亦即欄位資料型態限制)，有時也會結合其他三種條件限制：Entity Integrity、Referential Integrity 以及 User Defined Integrity。

由於本系統工具在設計上部份採取共用資料表的架構，所以，在處理 Integrity Constraints 的時候會發生一些問題，必須將 TenantId 這個詮釋欄位納入考量。以圖 3.10 的 Uniqueness Check 處理為例，假定要將 CourseId 這個共有欄位設定為 Unique，則需要透過多欄位的唯一限制來協助處理(如圖 3.10 方框所示)。目前本系統工具尚未支援所有類型的 Integrity Constraints，其他有關於 Integrity Constraints 的討論會在第五章做出進一步地探討。

```
CREATE TABLE CourseInfoCommonFields
( TenantId  Char(50) ,
  Row Integer ,
  CourseId Char(50) Unique
              ⋮

  Primary key (TenantId, Row))
```

```
CREATE TABLE CourseInfoCommonFields
( TenantId  Char(50) ,
  Row Integer ,
  CourseId Char(50) ,
  Constraint U_CourseId Unique ( TenantId, CourseId ) ,
              ⋮

  Primary key (TenantId, Row))
```

圖 3.10 處理 Integrity Constraints 的範例

B. CREATE EXTENSION TABLE 語句的設計概念及實作

CREATE EXTENSION TABLE 語句的設計概念主要是為了協助軟體開發人員在新增租戶的時候，可以撰寫該類型的語句建立新租戶所獨自擁有的私有資料表

集合。所以，CREATE EXTENSION TABLE 語句的實作主要有兩個步驟：

1. 查詢 Columns_Metadata Table 找出所有的共用資料表名稱並產生私有資料表名稱。
2. 為新加入的租戶建立其獨自擁有的私有資料表集合。

```
1. Class.forName("com.mysql.jdbc.Driver");
2.    //建立與database的連線
3. Connection con = (Connection)
4. DriverManager.getConnection("jdbc:mysql://localhost/Exte
   nsionTable?useUnicode=true&characterEncoding=utf
   8","root","1234");
4.    //建立新的Multitenantstatement
5. Statement stmt = new
   Multitenantstatement(con,"ExtensionTable");
6. String sql =
   "CREATE EXTENSION TABLE Nccu";
7. stmt.executeUpdate(sql);
```

圖 3.11 CREATE EXTENSION TABLE 語句範例

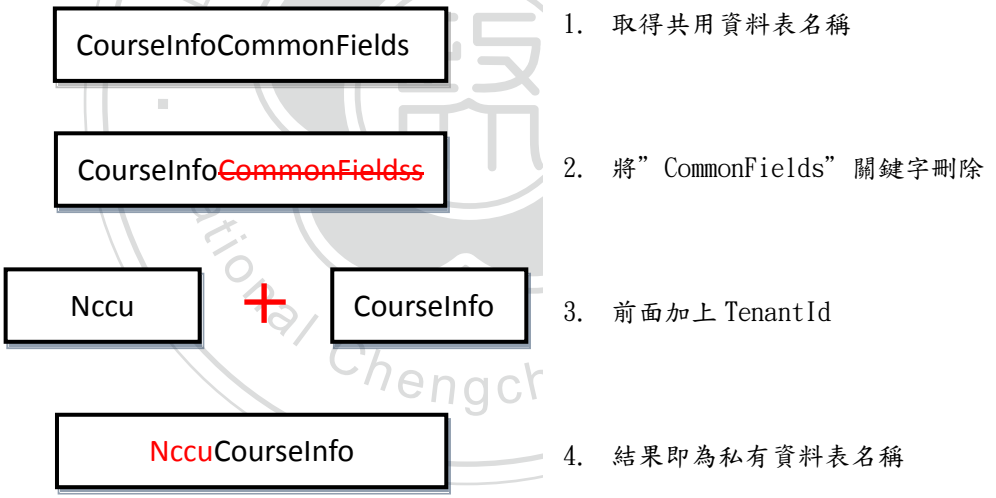

圖 3.12 產生私有資料表名稱的流程

以圖 3.11 的 CREATE EXTENSION TABLE 語句為例(line 6)，先利用共用資料表名稱的關鍵字"CommonFields"在 Columns_Metadata Table 中搜尋出所有已存在的共用資料表名稱。接下來，為新加入的租戶建立其獨自擁有的私有資料表合(圖 3.12)，首先，將所搜尋到的共用資料表名稱中的關字"CommonFields"刪除並在資料表名稱前面加上 TenantId(此為 Nccu)以作為識別，組合成私有資料表名稱；最後，產生 CREATE 語句(圖 3.13)，在欄位宣告的部分加入了詮釋欄

位(TenantId、Row)的宣告，同時，將詮釋欄位設置欄位限制為 Not Null；另外，增加一個 Composite Primary Keys 以確保(TenantId , Row)的組合會是唯一。

```
CREATE TABLE NccuCourseInfo
( TenantId  Char(50) ,
  Row Integer ,
  Primary key (TenantId, Row))
```

圖 3.13 實作 CREATE EXTENSION TABLE 語句概念所產生的 CREATE 語句

同時，在 Columns_Metadata Table 進行更新的動作，新增一筆記錄(圖 3.14)該私有資料表(此為 NccuCourseInfo)的欄位資訊在 Columns_Metadata Table。

```
4  NccuCourseInfo
```

圖 3.14 執行 CREATE EXTENSION TABLE 語句後，Columns_Metadata Table 的更新結果

```
CREATE TABLE NccuCourseInfo
( TenantId  Char(50) ,
  Row Integer ,
  Primary key (TenantId, Row))

CREATE TABLE NccuStudentInfo
( TenantId  Char(50) ,
  Row Integer ,
  Primary key (TenantId, Row))

CREATE TABLE NccuSelectCourse
( TenantId  Char(50) ,
  Row Integer ,
  Primary key (TenantId, Row))
```

圖 3.15 多個共用資料表存在，實作 CREATE EXTENSION TABLE 語句所產生的 CREATE 語句

```
1  CourseInfoCommonFields, CourseId, CourseName, Instructors, Credit, Days, Time
2  StudentInfoCommonFields, StudentId, StudentName, Password, Major, Grade
3  SelectCourseCommonFields, SelectId, StudentId, CourseId, SelectDate, Priority
4  NccuCourseInfo
5  NccuStudentInfo
6  NccuSelectCourse
```

圖 3.16 多共用資料表存在時，執行 CREATE EXTENSION TABLE 語句後，Columns_Metadata Table 的更新結果

事實上 CREATE EXTENSION TABLE 語句概念是由多個 CREATE 語句所實作而

成。倘若資料庫中存在一個以上的共用資料表，則仿造上述的作法，逐一為新租戶建立每一個私有資料表。因此，實作 CREATE EXTENSION TABLE 語句所產生的 CREATE 語句數量會等同於共用資料表的數量。假定目前已存在三個共用資料表，分別是 CourseInfoCommonFileds、StudentInfoCommonFields 以及 Select-CourseCommonFields，則實作 CREATE EXTENSION TABLE 語句所產生的 CREATE 語句數量會是三句(圖 3.15)。另外，Columns_Metadata Table 也會隨之進行更新的動作，其所增加的欄位資訊部分如圖 3.16 的紅色方框部份。

3.4.3 ALTER 語句的轉換

上一小節藉著建立共用資料表去實現共有欄位的概念，接下來的小節將以建立好的私有資料表透過 ALTER 語句的撰寫來滿足租戶客製化欄位的需求。本系統工具中，主要支援了四種常見的 ALTER 語句類型，這些 ALTER 語句分別在執行之後對資料表綱要產生不同影響，依照其影響結果可以分成四種類型：

類型 1. 增加一個資料表欄位

在”CourseInfo”資料表中增加一個資料表欄位 Location，其欄位型態為 char，長度為 10。

```
ALTER TABLE CourseInfo ADD Location Char(10)
```

類型 2. 改變資料表欄位的名稱

在” CourseInfo”資料表中的資料表欄位 Location 的名稱變更成 Classroom。

```
ALTER TABLE CourseInfo CHANGE Location Classroom Char(10)
```

類型 3. 改變資料表欄位的資料型態

在” CourseInfo”資料表中將資料表欄位 Location 的資料型態從 char(10)變更成 char(30)。

```
ALTER TABLE CourseInfo MODIFY Location Char(30)
```

類型 4. 刪去一個資料表欄位

刪除” CourseInfo”資料表中的資料表欄位 Location。

```
ALTER TABLE CourseInfo DROP Location
```

介紹完四種 ALTER 語句的表達方式，可以知道執行 ALTER 語句對私有資料表影響可以滿足租戶建立客製化欄位或是對其客製化欄位進行變更、刪除等動作。假定租戶要建立客製化欄位可以利用上述介紹的類型 1.去完成；若要更改其客製化欄位名稱或是資料型態，則分別透過類型 2 以及類型 3 去完成；若執行類型 4.則可達到刪除客製化欄位的效果。接下來，探討本系統工具如何進行 ALTER 語句轉換，主要有三個步驟：

1. 透過改寫 JDBC 的部分 API 去攔截軟體開發人員所撰寫的 SQL 語句，此外，軟體開發人員在 `setTenantId()`設定其 TenantId 以協助系統進行語句轉換。在此，透過 TenantId 能夠清楚得知其 SQL 語句所要執行的對象(租戶)。
2. 將攔截到的語句藉由 JSqlParser 進行語句結構剖析的動作，從其結果獲得轉換語句所需要的語句元素。
3. 以 Extension Table Layout 的邏輯進行語句轉換。

這個部份將以圖 3.17 的 ALTER 語句(line 6)為範例來進行詳細的轉換過程說明。首先，透過改寫 JDBC 的部分 API 去攔截軟體開發人員所撰寫的 SQL 語句，利用 `setTenantId()`取得 TenantId(此為 Nccu)以協助系統進行語句轉換(line 7)。接下來，JSqlPaeser 分析語句結構取得所需的語句元素，例如：資料表名稱(此為 CourseInfo)、新建立的欄位名稱(此為 Location)以及新建立的欄位型態(此為 Char(10))。

最後，進入依據 Extension Table Layout 邏輯進行語句轉換的階段，先判斷資料表名稱來了解執行對象是共用資料表或是私有資料表，此例因為資料表名稱結尾沒有關鍵字”CommonFields”，所以，可以知道執行對象是屬於私有資料表，實際上，也就是租戶進行新增客製化欄位的動作(反之，若其執行對象為共有資料表，則代表軟體開發人員對共用資料表進行新增欄位的動作)，接下來，本系統工具把 TenantId(此為 Nccu)以及資料表名稱(CourseInfo)組合起來形成轉換所需的私有資料表名稱（此為 NccuCourseInfo)，其餘的語句元素不需要做額外的改變。轉換後的結果如同 3.18 所示。

```
1. Class.forName("com.mysql.jdbc.Driver");
2.    //建立與database的連線
3. Connection con = (Connection)
   DriverManager.getConnection("jdbc:mysql://localhost/Exte
   nsionTable?useUnicode=true&characterEncoding=utf-
   8","root","1234");
4.    //建立新的Multitenantstatement
5. Statement stmt = new
   Multitenantstatement(con,"ExtensionTable");
6. String sql =
   "ALTER TABLE CourseInfo ADD Location Char(10)";
7. stmt.setTenantId("Nccu");
8. stmt.executeUpdate(sql);
```

圖 3.17 ALTER 語句範例

```
ALTER TABLE NccuCourseInfo ADD Location Char(50)
```

圖 3.18 ALTER 語句轉換後的結果

最後，同樣在 Columns_Metadata Table 新增”Location”欄位名稱到原本記錄 NccuCourseInfo 資料表欄位資訊的該筆記錄(圖 3.18)。

```
4  NccuCourseInfo, Location
```

圖 3.19 執行 ALTER 語句後， Columns_Metadata Table 的更新結果

總結 3.4.2 以及 3.4.3 兩小節，在其所提到的 CREATE、CREATE EXTENSION TABLE 以及 ALTER 語句相互配合之下，順利在多租戶應用程式之資料層級實現

Extension Table Layout 的共有欄位、私有欄位概念。

3.4.4 INSERT、UPDATE、DELETE 語句的轉換

INSERT、UPDATE、DELETE 語句主要是協助軟體開發人員進行資料的新增、更改以及刪除等動作。由於 Extension Table Layout 的概念將欄位分成共有欄位、私有欄位並且分別存放在共用資料表、私有資料表，所以，在進行這三種類型語句的語句轉換過程中，必要時會同時新增、更改共用資料表以及私有資料表上的資料，亦或是同時刪除存在於共用資料表以及私有資料表中的資料。因此，轉換過程中系統工具會自動產生兩個語句分別對存在於共用資料表以及私有資料表的資料進行增刪修改的動作以達到原 SQL 語句的目的。

A. INSERT 語句的轉換

在轉換之前，先了解 INSERT 語句的語句結構，從圖 3.20(line 6)清楚得知 INSERT 語句是由資料表名稱、新增資料欄位名稱與新增資料數值這三項語句元素所組成。轉換 INSERT 語句有四個主要步驟，歸納如下：

1. 透過改寫 JDBC 的部分 API 去攔截軟體開發人員所撰寫的 SQL 語句，此外，軟體開發人員在 `setTenantId()` 設定其 TenantId 以協助系統進行語句轉換。在此，透過 TenantId 能夠清楚得知其 SQL 語句所要執行的對象(租戶)。
2. 將攔截到的語句藉由 JSqlParser 進行語句結構剖析的動作，從其結果獲得轉換語句所需要的語句元素。
3. 產生獨一無二的詮釋欄位組合以及查詢新增資料欄位、新增資料數值所在的資料表並依其查詢結果進行分派。
4. 以 Extension Table Layout 的邏輯去進行語句轉換。

```
1.Class.forName("com.mysql.jdbc.Driver");
2.    //建立與database的連線
3.Connection con = (Connection)
  DriverManager.getConnection("jdbc:mysql://localhost/Exte
  nsionTable?useUnicode=true&characterEncoding=utf
  8","root","1234");
4.    //建立新的Multitenantstatement
5. Statement stmt = new
   Multitenantstatement(con,"ExtensionTable");
6. String sql =
    "INSERT INTO CourseInfo
            (CourseId, CourseName, Instrutors, Credit,
             Days, Time, Location)
     VALUES ('Nccu3', '編譯器設計', '陳恭', 3,
             'Wed', '234', '大仁3301')";
7. stmt.setTenantId("Nccu");
8. stmt.executeUpdate(sql);
```

圖 3.20 INSERT 語句範例

這個部份將以圖 3.20 的 INSERT 語句為例說明如何進行 INSERT 語句轉換。首先,藉由改寫 JDBC 的部分 API 攔截去軟體開發人員所撰寫的 SQL 語句,另外,透過 setTenantId()取得 TenantId(此為 Nccu)以協助系統進行語句轉換(line 7)。再來,JSqlParser 取得其他語句元素,例如:資料表名稱(此為 CourseInfo)、新增資料欄位名稱(此為 CourseId,CourseName,Instrutors…)以及新增資料數值(此為'Nccu3','編譯器設計','陳恭'…)。接下來,本系統工具會自動產生 SELECT 語句去計算詮釋欄位 Row 的最大值(圖 3.21),將 Row 的最大值加一之後和 TenantId 組成一個獨一無二的詮釋欄位組合(TenantId、Row)。

緊接著,在 Columns_Metadata Table 中查詢新增資料欄位名稱所在的資料表並進行新增資料欄位名稱的分派,以 CourseId 欄位為例,由於其屬於共有欄位,存在於共用資料表的欄位資訊中,所以,分派在共用資料表 INSERT 語句;反之,Location 欄位存在於私有資料表的欄位資訊中,所以,分派在私有資料表 INSERT 語句,仿造上述的方式,新增資料數值也是進行相同的分派過程。

最後,進入依據 Extension Table Layout 邏輯進行語句轉換的階段,本系

統工具會自動產生兩個 INSERT 語句(圖 3.22)：共用資料表 INSERT 語句、私有資料表 INSERT 語句，分別新增共有、私有欄位資料到共用、私有資料表中，其語句中的資料表名稱分別是共用資料表名稱(此為 CourseInfoCommonFileds)、私有資料表名稱(此為 NccuCourseInfo)，並根據上一步驟新增資料欄位名稱以及新增資料數值分派的結果，加上詮釋欄位組合成新增資料欄位名稱、新增資料數值的部分。

NccuCourseInfo

TenantId	Row	Location	Language
Nccu	1	大仁3301	中文
Nccu	2	大仁1103	中文

SELECT Max(Row)
FROM NccuCourseInfo

➡ Max of Row值：2

➡ 新增的Row值：2 + 1 = 3

➡ 產生獨一無二的詮釋欄位組合(TenantId , Row)=('Nccu' , 3)

圖 3.21 系統工具自動產生計算詮釋欄位 Row 最大值的 SELECT 語句

- **負責新增資料到共用資料表的 INSERT 語句：**

```
INSERT INTO CourseInfoCommonFields
(TenantId, Row, CourseId, CourseName, Instrutors, Credit, Days, Time)
VALUES ('Nccu', 3 ,' Nccu3', '編譯器設計', '陳恭', 3, 'Wed', '234')
```

- **負責新增資料到私有資料表的 INSERT 語句：**

```
INSERT INTO NccuCourseInfo
(TenantId, Row, Location)
VALUES ('Nccu', 3, '大仁3301')
```

圖 3.22 INSERT 語句轉換後的結果

除此之外，在進行 INSERT 語句轉換時還有一個重點，假定 SQL 語句如圖 3.23 的 line 6 所示，只有新增共有欄位的資料，在進行轉換的時候，本系統工具是否只要自動建立一個 INSERT 語句負責新增共有欄位資料到共用資料表中?這個

答案是否定的，為了維持共用資料表和私有資料表的資料存在一致性，即使在進行新增資料時沒有影響到私有欄位，本系統工具仍需產生一個 INSERT 語句負責新增該筆資料的詮釋欄位部份到私有資料表中。(圖 3.24)

```
1. Class.forName("com.mysql.jdbc.Driver");
2.    //建立與database的連線
3. Connection con = (Connection)
   DriverManager.getConnection("jdbc:mysql://localhost/Exte
    nsionTable?useUnicode=true&characterEncoding=utf
    8","root","1234");
4.    //建立新的Multitenantstatement
5. Statement stmt = new
   Multitenantstatement(con,"ExtensionTable");
6. String sql =
    "INSERT INTO CourseInfo
            (CourseId, CourseName, Instrutors, Credit,
             Days, Time)
     VALUES ('Nccu3', '編譯器設計', '陳恭', 3,
             'Wed', '234')";
7. stmt.setTenantId("Nccu");
8. stmt.executeUpdate(sql);
```

圖 3.23 INSERT 語句(僅新增共有欄位資料)範例

- **負責新增資料到共用資料表的 INSERT 語句：**

```
INSERT INTO CourseInfoCommonFields
(TenantId, Row, CourseId, CourseName, Instrutors, Credit, Days, Time)
VALUES ('Nccu', 3, 'Nccu3', '編譯器設計', '陳恭', 3, 'Wed', '234')
```

- **負責新增資料到私有資料表的 INSERT 語句：**

```
INSERT INTO NccuCourseInfo
(TenantId, Row)
VALUES ('Nccu', 3)
```

圖 3.24 INSERT 語句(僅新增共有欄位資料)轉換後的結果

B. UPDATE 語句的轉換

在轉換之前，先對 UPDATE 語句的語句結構進行了解，從圖 3.25(line 6)清楚得知 UPDATE 語句是由資料表名稱、更新資料欄位名稱、更新資料數值以及 WHERE 條件式這幾項語句元素所組合而成。由於 UPDATE 語句的語句結構比較複

雜，所以，轉換 UPDATE 語句有五個主要步驟，歸納如下：

1. 透過改寫 JDBC 的部分 API 去攔截軟體開發人員所撰寫的 SQL 語句，此外，軟體開發人員在 setTenantId()設定其 TenantId 以協助系統進行語句轉換。在此，透過 TenantId 能夠清楚得知其 SQL 語句所要執行的對象(租戶)。
2. 將攔截到的語句藉由 JSqlParser 進行語句結構剖析的動作，從其結果獲得轉換語句所需要的語句元素。
3. 找出符合 WHERE 條件式的資料集合，同時，記錄該集合中每筆資料的詮釋欄位組合。
4. 查詢更新資料欄位所在的資料表並依其結果進行分派。
5. 以 Extension Table Layout 的邏輯進行語句轉換。

```
1.Class.forName("com.mysql.jdbc.Driver");
2.    //建立與database的連線
3.Connection con = (Connection)
  DriverManager.getConnection("jdbc:mysql://localhost/Exte
  nsionTable?useUnicode=true&characterEncoding=utf
  8","root","1234");
4.    //建立新的Multitenantstatement
5. Statement stmt = new
   Multitenantstatement(con,"ExtensionTable");
6. String sql =
    "UPDATE CourseInfo
     SET Credit = 3, Location = '大仁1106'
     WHERE Credit >= 3";
7. stmt.setTenantId("Nccu");
8. stmt.executeUpdate(sql);
```

圖 3.25 UPDATE 語句範例

以圖 3.25 的 UPDATE 語句(line 6)為例詳細說明如何進行 UPDATE 語句的轉換。首先，透過改寫 JDBC 的部分 API 去攔截軟體開發人員所撰寫的 SQL 語句，利用 setTenantId()取得 TenantId(此為 Nccu)以協助系統進行語句轉換(line 7)。再來，藉由 JSqlParser 剖析語句結構來取得其他語句元素，例如：資料表名稱(此為 CourseInfo)、更新資料欄位名稱(此為 Credit,Location)、更新資料數值(此為 3, '大仁 1106')以及 WHERE 條件式(此為 Credit >= 3)。

接下來，本系統工具會自動產生 SELECT with JOIN 語句找出符合 WHERE 條件式的資料集合(圖 3.26)，該 SELECT with JOIN 語句的 WHERE 條件式除了原先的 WHERE 條件式之外還需要增加 TenantId 這個條件(此為 TenantId = 'Nccu')來進行重組，查詢後找出符合條件式的資料集合，同時，記錄集合中每筆資料的詮釋欄位組合(此為(TenantId , Row) ={('Nccu' ,1),('Nccu' ,2)})。

緊接著，利用 Columns_Metadata Table 去查詢更新資料欄位名稱所在的資料表並進行更新資料欄位名稱的分派，以 Credit 欄位為例，由於其屬於共有欄位，存在於共用資料表的欄位資訊中，所以，分派在共用資料表 UPDATE 語句；反之，Location 欄位存在於私有資料表的欄位資訊中，所以，分派在私有資料表 UPDATE 語句，仿造上述的方式，更新資料數值也是進行相同的分派過程。

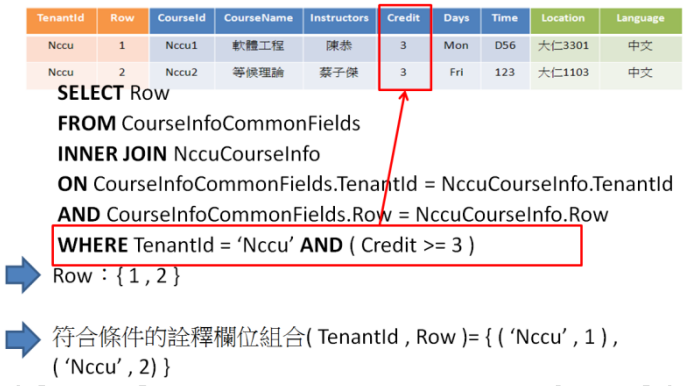

圖 3.26 UPDATE 語句轉換過程中，系統工具自動產生的 SELECT with JOIN 語句：找出符合 WHERE 條件式的資料集合，同時，記錄集合中每筆資料的詮釋欄位組合。

最後，進入依據 Extension Table Layout 邏輯進行語句轉換的階段，對上一步驟集合中的每筆資料來說，本系統工具會各自產生兩個 UPDATE 語句，分別是更新共有欄位資料到共用資料表的共用資料表 UPDATE 語句、更新私有欄位資料到私有資料表的私有資料表 UPDATE 語句。因此，其語句中的資料表名稱分別是共用資料表名稱(此為 CourseInfoCommonFileds)、私有資料表名稱(此為 NccuCourseInfo)，並根據上一步驟更新資料欄位名稱以及更新資料數值分派的結果，形成更新資料欄位名稱、更新資料數值的部分，WHERE 條件式則是集合中

每筆資料的詮釋欄位組合。轉換後的結果如圖 3.27 所示。

- **對詮釋欄位('Nccu' ,1)所產生的共用、私有資料表 UPDATE 語句：**

```
UPDATE CourseInfoCommonFields
SET Credit = 3
WHERE TenantId = 'Nccu' AND Row = 1
```

```
UPDATE NccuCourseInfo
SET Location = '大仁1106'
WHERE TenantId = 'Nccu' AND Row = 1
```

- **對詮釋欄位('Nccu' ,2)所產生的共用、私有資料表 UPDATE 語句：**

```
UPDATE CourseInfoCommonFields
SET Credit = 3
WHERE TenantId = 'Nccu' AND Row = 2
```

```
UPDATE NccuCourseInfo
SET Location = '大仁1106'
WHERE TenantId = 'Nccu' AND Row = 2
```

圖 3.27 UPDATE 語句轉換後的結果

```
1. Class.forName("com.mysql.jdbc.Driver");
2.    //建立與database的連線
3. Connection con = (Connection)
   DriverManager.getConnection("jdbc:mysql://localhost/Exte
   nsionTable?useUnicode=true&characterEncoding=utf
   8","root","1234");
4.    //建立新的Multitenantstatement
5. Statement stmt = new
   Multitenantstatement(con,"ExtensionTable");
6. String sql =
    "UPDATE CourseInfo
     SET Location = '大仁1106'
     WHERE Credit >= 3";
7. stmt.setTenantId("Nccu");
8. stmt.executeUpdate(sql);
```

圖 3.28 UPDATE 語句(僅更新私有欄位資料)範例

相對於 INSERT 語句轉換，假定 UPDATE 語句如圖 3.28 的 line 6 所示，只有更新私有欄位(或共有欄位)的資料，在進行語句轉換的時候，本系統工具只

需自動建立一個 UPDATE 語句負責更新私有欄位(或共有欄位)資料到私有資料表(或共用資料表)即可。(圖 3.29)

- **對詮釋欄位('Nccu' ,1)所產生的私有資料表 UPDATE 語句：**

```
UPDATE NccuCourseInfo
SET Location = '大仁1106'
WHERE TenantId = 'Nccu' AND Row = 1
```

- **對詮釋欄位('Nccu' ,2)所產生的私有資料表 UPDATE 語句：**

```
UPDATE NccuCourseInfo
SET Location = '大仁1106'
WHERE TenantId = 'Nccu' AND Row = 2
```

圖 3.29 UPDATE 語句(僅更新私有欄位資料)轉換後的結果

C. DELETE 語句的轉換

```
1. Class.forName("com.mysql.jdbc.Driver");
2.    //建立與database的連線
3. Connection con = (Connection)
   DriverManager.getConnection("jdbc:mysql://localhost/Ext
   ensionTable?useUnicode=true&characterEncoding=utf-
   8","root","1234");
4.    //建立新的Multitenantstatement
5. Statement stmt = new
   Multitenantstatement(con,"ExtensionTable");
6. String sql =
   "DELETE FROM CourseInfo
     WHERE Credit >= 3"
7. stmt.setTenantId("Nccu");
8. stmt.executeUpdate(sql);
```

圖 3.30 DELETE 語句範例

從圖 3.30 的 DELETE 語句(line 6)得知此種類型語句的語句結構是由資料表名稱以及 WHERE 條件式這兩項語句元素組合而成。轉換 DELETE 語句主要有四個步驟，歸納如下：

1. 透過改寫 JDBC 的部分 API 去攔截軟體開發人員所撰寫的 SQL 語句，此外，軟體開發人員在 setTenantId()設定其 TenantId 以協助系統進行語句轉

換。在此，透過 TenantId 能夠清楚得知其 SQL 語句所要執行的對象(租戶)。

2. 將攔截到的語句藉由 JSqlParser 進行語句結構剖析的動作，從其結果獲得轉換語句所需要的語句元素。

3. 找出符合 WHERE 條件式的資料集合，同時，記錄該集合中每筆資料的詮釋欄位組合。

4. 以 Extension Table Layout 的邏輯進行語句轉換。

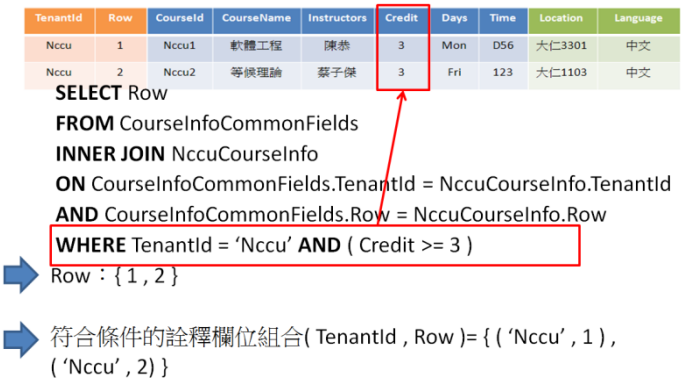

圖 3.31 DELETE 語句轉換過程中，系統工具自動產生的 SELECT with JOIN 語句：找出符合 WHERE 條件式的資料集合，同時，記錄集合中每筆資料的詮釋欄位組合。

以圖 3.30 中的 DELETE 語句(line 6)進行轉換為例。首先，藉由改寫 JDBC 的部分 API 去攔截軟體開發人員所撰寫的 SQL 語句，由 `setTenantId()`取得 TenantId(此為 Nccu)以協助系統進行語句轉換(line 7)。再來，透過 JSqlParser 剖析語句結構來取得其他語句元素，例如：資料表名稱(此為 CourseInfo)以及 WHERE 條件式(此為 Credit >= 3)。接下來，本系統工具會自動產生 SELECT with JOIN 語句找出符合 WHERE 條件式的資料集合(圖 3.31)，但是，該 SELECT with JOIN 語句的 WHERE 條件式除了原先的 WHERE 條件式之外還需要加上 TenantId 這個條件(此為 TenantId = 'Nccu')來進行重組，查詢後找出符合條件式的資料集合，同時，記錄集合中每筆資料的詮釋欄位組合(此為(TenantId , Row) = {('Nccu' , 1), ('Nccu' , 2)})。

最後，依照 Extension Table Layout 邏輯進行語句轉換，對於上一步驟集

合中的每筆資料來說，本系統工具會各自產生兩個 DELETE 語句：共用資料表 DELETE 語句、私有資料表 DELETE 語句，分別刪除在共用、私有資料表中的資料，其語句中的資料表名稱分別是共用資料表名稱(此為 CourseInfoCommonFileds)、私有資料表名稱(此為 NccuCourseInfo)，WHERE 條件式則是集合中每筆資料的詮釋欄位組合。圖 3.32 為轉換後的結果。

- **對詮釋欄位('Nccu' ,1)所產生的共用、私有資料表 DELETE 語句：**

```
DELETE FROM CourseInfoCommonFields
WHERE TenantId = 'Nccu' AND Row = 1
```

```
DELETE FROM NccuCourseInfo
WHERE TenantId = 'Nccu' AND Row = 1
```

- **對詮釋欄位('Nccu' ,2)所產生的共用、私有資料表 DELETE 語句：**

```
DELETE FROM CourseInfoCommonFields
WHERE TenantId = 'Nccu' AND Row = 2
```

```
DELETE FROM NccuCourseInfo
WHERE TenantId = 'Nccu' AND Row = 2
```

圖 3.32 DELETE 語句轉換後的結果

3.4.5 SELECT 語句的轉換

SELECT 語句主要是讓軟體開發人員對現有的資料進行查詢、統計等動作。其可搭配 WHERE 條件式、JOIN 指令等增加查詢的多樣性。

在轉換之前，先了解 SELECT 語句的結構，從圖 3.33(line 6)清楚得知最基本的 SELECT 語句是由回傳資料欄位、資料表名稱這兩項語句元素所組合而成；此外，可以選擇性地加上 WHERE 條件式以縮小查詢的範圍。轉換 SELECT 語句主要有三個步驟，歸納如下：

1. 透過改寫 JDBC 的部分 API 去攔截軟體開發人員所撰寫的 SQL 語句，此外，

軟體開發人員在 setTenantId()設定其 TenantId 以協助系統進行語句轉換。在此，透過 TenantId 能夠清楚得知其 SQL 語句所要執行的對象(租戶)。

2. 將攔截到的語句藉由 JSqlParser 進行語句結構剖析的動作，從其結果獲得轉換語句所需要的語句元素，同時，判斷回傳資料欄位是否為”*”關鍵字。

3. 以 Extension Table Layout 的邏輯進行語句轉換。

```
1. Class.forName("com.mysql.jdbc.Driver");
2.    //建立與database的連線
3. Connection con = (Connection)
   DriverManager.getConnection("jdbc:mysql://localhost/Exte
   nsionTable?useUnicode=true&characterEncoding=utf-
   8","root","1234");
4.    //建立新的Multitenantstatement
5. Statement stmt = new
   Multitenantstatement(con,"ExtensionTable");
6. String sql =
   "SELECT *
     FROM CourseInfo
     WHERE Instructors LIKE '%陳恭%'";
7. stmt.setTenantId("Nccu");
8. stmt.executeUpdate(sql);
```

圖 3.33 SELECT 語句範例

在這個部分以圖 3.33 為例詳細說明 SELECT 語句的轉換細節。首先，透過改寫 JDBC 的部分 API 去攔截到軟體開發人員所撰寫的 SQL 語句，同時，也可以取得 TenantId(此為 Nccu)以協助系統工具進行語句轉換(line 7)。再來，透過 JSqlParser 來取得其他語句元素，例如：回傳資料欄位(此為”*”關鍵字)、資料表名稱(此為 CourseInfo)以及 WHERE 條件式(此為 Instructors LIKE “%陳恭%”)。接下來，根據剖析結果判斷回傳資料欄位是否為”*”關鍵字(此為 True)，系統工具發現回傳資料欄位存在”*”關鍵字，所以，必需先查詢 Columns_Metadata Table 找出相對應記錄共用資料表(此為 CourseInfoCommonFields)、私有資料表(NccuCourseInfo)的欄位資訊部份，如圖 3.34 以及圖 3.36 的紅字

部分所示，取得兩資料表的所有欄位名稱之後，將之組合起來取代”*”關鍵字形成下一階段轉換後所產生之 SELECT 語句的回傳資料欄位。

最後，依據 Extension Table Layout 邏輯進行語句轉換，為了查詢出共用資料表以及私有資料表的所有欄位資料，所以，進行 INNER JOIN 將共用資料表以及私有資料表作鏈結並將鏈結後的資料表進行別名(Alias)的動作以達到利用鏈結後的資料表來進行實際查詢的目的，如圖 3.35 的紅色方框所示。

```
1  CourseInfoCommonFields, CourseId, CourseName, Instructors, Credit, Days, Time
2  StudentInfoCommonFields, StudentId, StudentName, Password, Major, Grade
3  SelectCourseCommonFields, SelectId, StudentId, CourseId, SelectDate, Priority
4  NccuCourseInfo, Location , Language
5  NccuStudentInfo
6  NccuSelectCourse
```

CourseId, CourseName, Instructors, Credit, Days, Time + Location, Language

圖 3.34 回傳資料欄位 “*” 關鍵字的轉換過程：

查詢 Columns_Metadata Table 找出相對應記錄共用資料表(此為 CourseInfoCommonFields)、私有資料表(NccuCourseInfo)的欄位資訊部份，

將兩資料表的欄位資訊進行組合取代”*”成為轉換後語句的回傳資料欄位。

```
SELECT *
FROM
( SELECT *
  FROM CourseInfoCommonFields
  INNER JOIN NccuCourseInfo
  ON CourseInfoCommonFields.TenantId = NccuCourseInfo.TenantId
  AND CourseInfoCommonFields.Row = NccuCourseInfo.Row
) AS NccuCourseInfo
WHERE Instructors LIKE '%陳恭%'
```

圖 3.35 建立實際查詢的資料表示意圖：進行 INNER JOIN 將共用資料表以及私有資料表作鏈結並將鏈結後的資料表進行別名(Alias)的動作以達到利用鏈結後的資料表進行實際查詢的目的。

SELECT CourseId, CourseName, Instructors, Credit, Days, Time, Location, Language

FROM

TenantId	Row	CourseId	CourseName	Instructors	Credit	Days	Time	Location	Language
Nccu	1	Nccu1	軟體工程	陳恭	3	Mon	D56	大仁3301	中文
Nccu	2	Nccu2	等候理論	蔡子傑	3	Fri	123	大仁1103	中文

WHERE Instructors **LIKE** '%陳恭%'

圖 3.36 取代”*”關鍵字示意圖。

第四章

系統實作與評估

此章節針對系統工具進行實作展示與效能評估。在展示的部份，以採用本系統工具實際協助開發一個多校(租戶)選課系統為範例，展示本系統工具的優點。在效能評估的部份，透過一連串的實驗並根據結果推斷影響本系統工具執行時間的因素。最後，分別在(1)Private Table Layout、(2) Extension Table Layout、(3) Extension Table Layout 的詮釋欄位建立索引(Index) 這三種測試環境中進行多項測試並根據其結果評估本系統工具的可行性。

4.1 系統實作展示

採用本系統工具實際協助開發一個多校(租戶)選課系統為例來展現其所帶來的兩項優點：(1)提供租戶適度地客製化其資料表綱要的能力、(2)提升軟體開發人員統一管理租戶的共有欄位以及其資料的能力。

4.1.1 多租戶應用程式範例情境描述

在這個多校(租戶)選課系統的範例中，仿造一般選課系統的使用情境，將設定三個使用角色，分別是：選課系統維運管理者(軟體開發人員)、選課系統承租學校(租戶)以及使用選課系統進行選課的學生(租戶的學生)。以下將針對這三個角色的操作行為作簡單地介紹：

1. **選課系統維運管理者：**意指軟體開發人員，負責選課系統的開發以及維運。
2. **選課系統承租學校：**意指租戶，同時，也是軟體開發人員的客戶。這裡的

學校數量不只一個，可能有兩間以上的學校。每間學校可以新增、更改其課程資訊或是客製化其部份資料表欄位以及查詢部分統計資訊。

3. **使用選課系統進行選課的學生**：意指租戶的學生，也是進行選課的學生。選課系統提供帳戶管理讓學生進行註冊、登入、查詢課程、在選課清單上加入或移除課程以及排列選課志願順序等動作。

4.1.2 多租戶應用程式範例系統設計與實作

在資料層級設計方面，選課系統維運管理者根據一般選課系統的需求，擬訂共用資料表的資料表綱要。以下為該系統的三個共用資料表說明以及資料表綱要展示(schema)(圖 4.1)：

(1) StudentInfoCommonFields：此資料表用來儲存所有學校的學生基本資料(僅含共有欄位部份)。

(2) SelectCourseCommonFields：此資料表用來儲存所有學校其每位學生的選課記錄(僅含共有欄位部份)。

(3) CourseInfoCommonFields：此資料表用來儲存所有學校的課程資料(僅含共有欄位部份)。

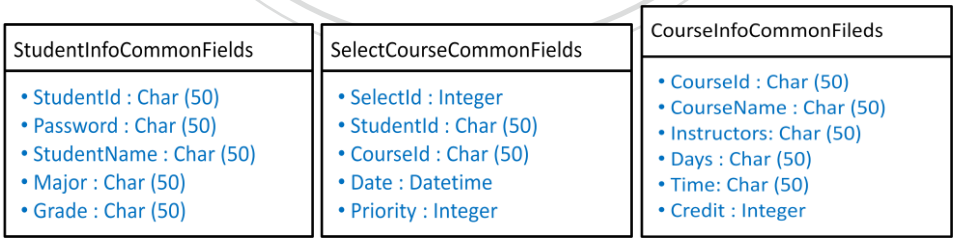

圖 4.1 多校(租戶)選課系統的範例之共用資料表(StudentInfoCommonFields、SelectCourseCommonFields 以及 CourseInfoCommonFields)的資料表綱要設計。

選課系統維運管理者必須撰寫 CREATE 語句在資料庫中建立這三個共有資料表，同時，對共有欄位設定一些限制(圖 4.2)。

```
CREATE TABLE StudentInfoCommonFields
( StudentId Char(50) not null,
  StudentName Char(50) not null,
  Password Char(50) not null,
  Major Char(50) not null,
  Grade Char(50) not null )
```

```
CREATE TABLE SelectCourseCommonFields
( SelectId Integer not null,
  StudentId Char(50) not null,
  CourseId Char(50) not null,
  SelectDate datetime not null,
  Priority Integer not null )
```

```
CREATE TABLE CourseInfoCommonFields
( CourseId Char(50) not null,
  CourseName Char(50) not null,
  Instructors Char(50) not null,
  Credit Integer not null,
  Days Char(50) not null,
  Time Char(50) not null)
```

圖 4.2 CREATE 語句範例展示－建立共用資料表

完成建立共有資料表的動作後，在 CREATE EXTENSION TABLE 語句的協助下，針對租戶(此為 Nccu)建立其使用的私有資料表集合。同時，Columns_Metadata Table 也會隨著 CREATE、CREATE EXTENSION TABLE 語句的執行，更新其欄位資訊(圖 4.3)。

```
1  CourseInfoCommonFields, CourseId, CourseName, Instructors, Credit, Days, Time
2  StudentInfoCommonFields, StudentId, StudentName, Password, Major, Grade
3  SelectCourseCommonFields, SelectId, StudentId, CourseId, SelectDate, Priority
4  NccuCourseInfo
5  NccuStudentInfo
6  NccuSelectCourse
```

圖 4.3 CREATE、CREATE EXTENSION TABLE 語句執行後，Columns_Metadata Table 的更新結果：

CREATE 語句執行後，產生了共用資料表欄位資訊(即新增了 1~3 行的部分)；

CREATE EXTENSION TABLE 語句執行後，產生了私有資料表欄位資訊(即新增了 4~6 行的部分)。

在應用程式設計方面，藉著對資料庫進行新增資料(例如：學校新增課程資料、學生將課程加入選課清單時，將選課記錄新增到資料庫)、更改資料(例如:

學校更改課程資料)、更改資料表綱要(例如：學校自行新增 CourseInfo 資料表的客製化欄位、對其客製化欄位進行更名或刪除的動作)、刪除資料(例如:學生從選課清單中刪除一門課程)亦或是查詢資料(例如：學生查詢選課清單)的動作來協助完成選課系統的基本功能。上述的動作皆可依據一租戶一資料表的寫法來撰寫 SQL 語句，再透過本系統工具依 Extension Table Layout 邏輯進行語句轉換並在資料庫中執行產生影響。

假定應用程式會逐一分配給每間學校(租戶)一個獨一無二的租戶辨別碼，即 TenantId。此 TenantId 透過 session 的方式從應用程式迭代到資料層級，所以，系統工具在執行這些 SQL 語句時可藉由 TenantId 了解其執行對象。接下來的例子，以 Nccu 該間學校(租戶)的角度示範如何藉由 SQL 語句的表達去實作多校(租戶)選課系統的各種功能。

1. **INSERT 語句：**

通常在新增資料的時候會撰寫 INSERT 語句來表達，以學校新增課程資料為例，先在網頁上提供一個 form 表單(圖 4.4)，表單欄位內容為該校 (即 Nccu) 的 CourseInfo 資料表的欄位(除了 CourseId 欄位，此欄位是經網站內部處理所產生的一個獨一無二課程代碼)。

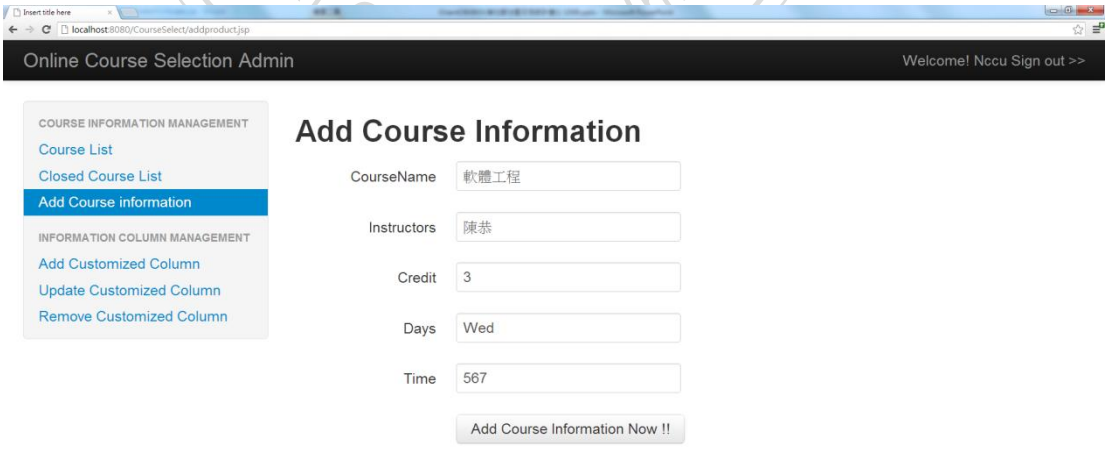

圖 4.4 學校新增課程資料的展示。

在網站內部處理的部份，為了產生一個獨一無二的課程號碼(即 CourseId)，先撰寫 SELECT 語句，找出該校 CourseInfo 資料表中的課程資料筆數，然後，

以課程總數遞增的方式建立該新增課程的 CourseId。

```
SELECT Count(CourseId) FROM CourseInfo
```

緊接著，網站內部經由網頁表單取得其所需欄位資料後，組合產生的 INSERT 語句如下：

```
INSERT INTO CourseInfo (CourseId,CourseName,Instrutors,Credit,Days,Time )
VALUES ('Nccu14','軟體工程','陳恭',3,'Wed','567')
```

最後，可以清楚地從頁面看到其新增結果(圖 4.5)：

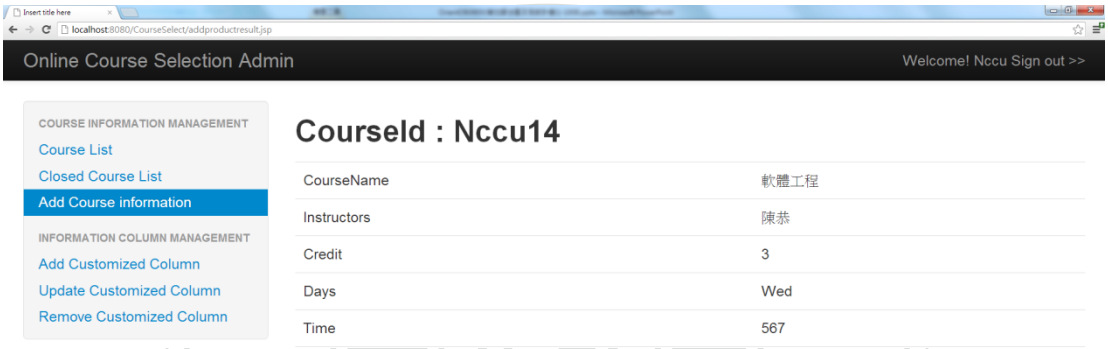

圖 4.5 學校新增課程資料的結果展示。

2. UPDATE 語句：

一般來說，透過 UPDATE 語句可以協助更改資料的動作，以學校(即 Nccu)更改課程資料為例，首先，先讓學校選擇其欲更改資料的課程(圖 4.6)：

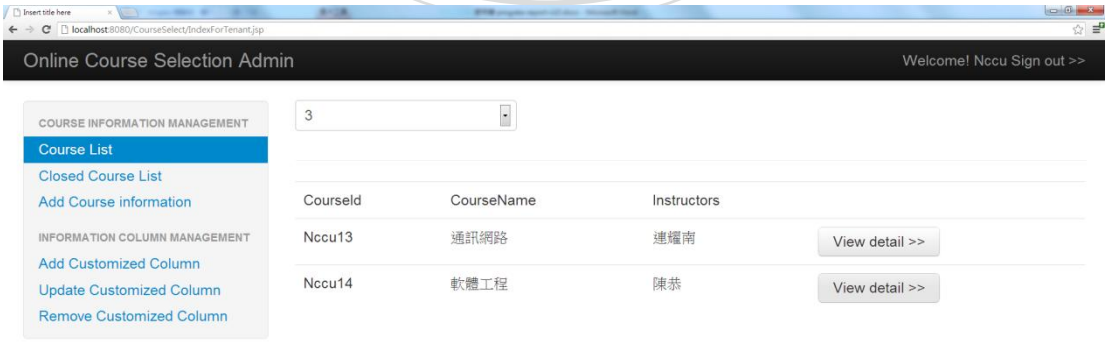

圖 4.6 學校點選欲修改課程的展示。

待選擇以後，則會出現該課程的原始資料，學校可以透過欄位去更改課程

資料，假設將 CourseId 為 Nccu14 的 Days 欄位從 Wed 更改成 Thu(圖 4.7)：

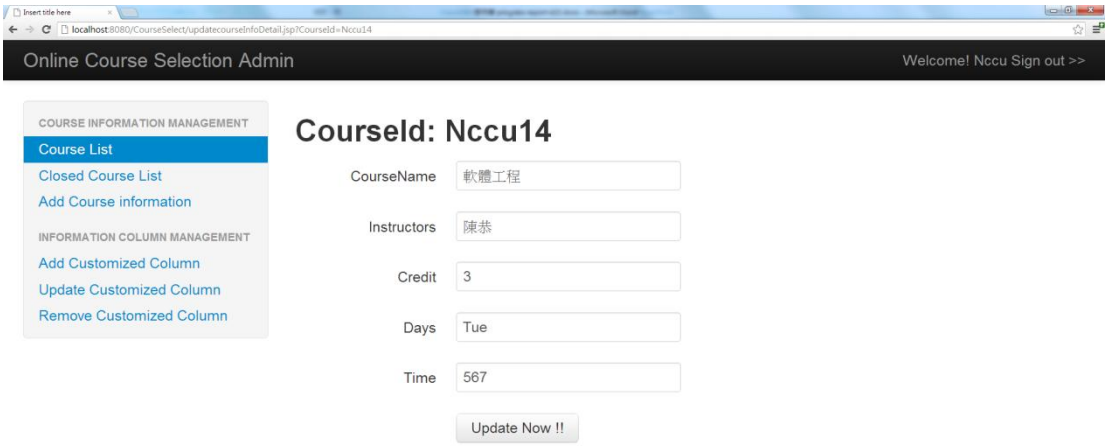

圖 4.7 學校進行課程資料修改的展示。

此時，藉由 UPDATE 語句去進行更改課程資料的動作。網站內部經由網頁表單取得其所需欄位資料後，組合產生的 UPDATE 語句如下：

```
UPDATE CourseInfo SET Days = 'Thu' WHERE CourseId = 'Nccu14'
```

更改課程資料後的結果顯示在圖 4.8：

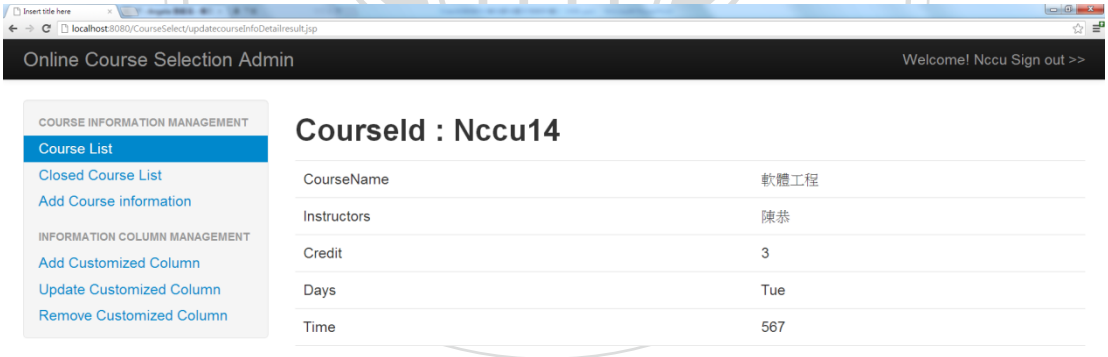

圖 4.8 學校修改課程資料的結果展示。

3. ALTER 語句：

學校可以透過網頁新增 CourseInfo 資料表的客製化欄位(即滿足租戶適度地客製化其資料表綱要的需求)。除此之外，亦可以對客製化欄位進行更名、刪除。以上這些對客製化欄位影響的動作皆利用 ALTER 語句即可完成。

以圖 4.9 為例，該學校(即 Nccu)將新增一個客製化欄位，其欄位名稱為

Location，型態為 char，長度為 50，以用來表示該課程的上課地點。

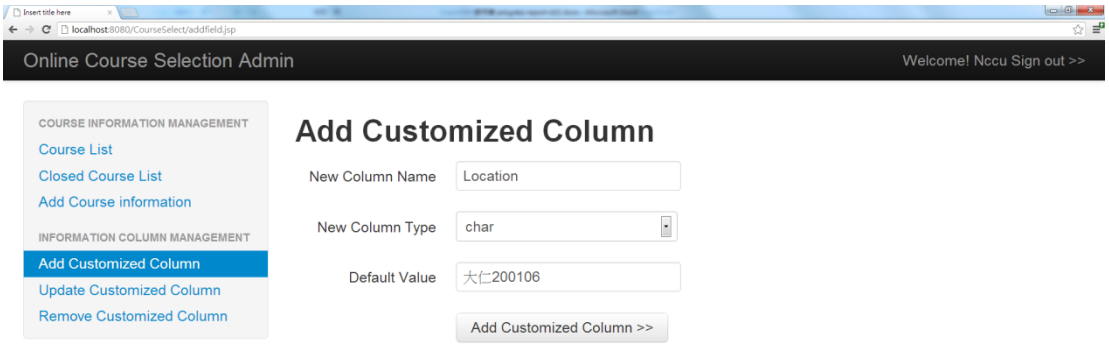

圖 4.9 學校新增客製化欄位的展示。

學校在輸入完該客製化欄位(即 Location)的資料型態等資訊後，網站內部經由網頁表單取得其所需欄位資料後，組合產生以下的 ALTER 語句去完成新增客製化欄位的動作：

```
ALTER TABLE CourseInfo ADD Location Char(50)
```

ATTER 語句執行以後，課程資料的顯示頁面增加了一個 Location 欄位(圖 4.10)：

圖 4.10 學校新增客製化欄位的結果展示。

另外，也可以對客製化欄位進行更名以及刪除的動作，以上述新增的客製化欄位(即 Location)為例，如果要將其名稱更名為 Classroom(圖 4.11)，則可以撰寫圖 4.13 中的 ATTER 語句一；若是撰寫圖 4.13 的 ATTER 語句二，則執行刪除該客製化欄位的動作(圖 4.12)，其執行後的結果如圖 4.14。

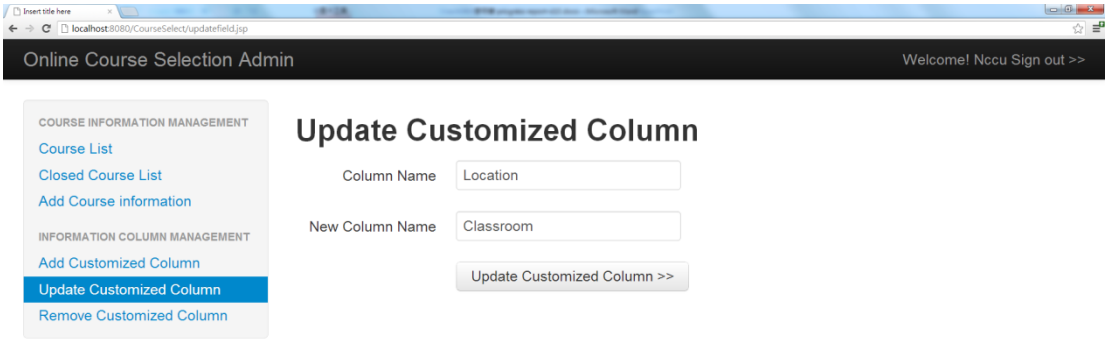

圖 4.11 客製化欄位更名的展示。

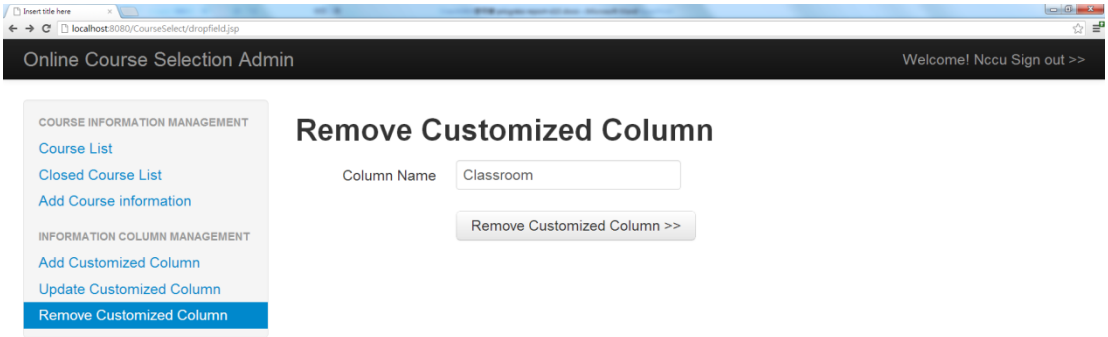

圖 4.12 刪除客製化欄位的展示。

ALTER 語句一:

```
ALTER TABLE CourseInfo CHANGE Location Classroom Char(50)
```

ALTER 語句二:

```
ALTER TABLE CourseInfo DROP Location
```

圖 4.13 客製化欄位更名以及刪除客製化欄位的 ALTER 語句展示。

圖 4.14 客製化欄位更名的結果展示。

4. DELETE 語句

DELETE 語句主要是對過時或是錯誤的資料進行刪除的動作。以學生從選課清單中刪除一門課程為例，如圖 4.15 所示學生可以透過頁面勾取其欲刪除的課程。

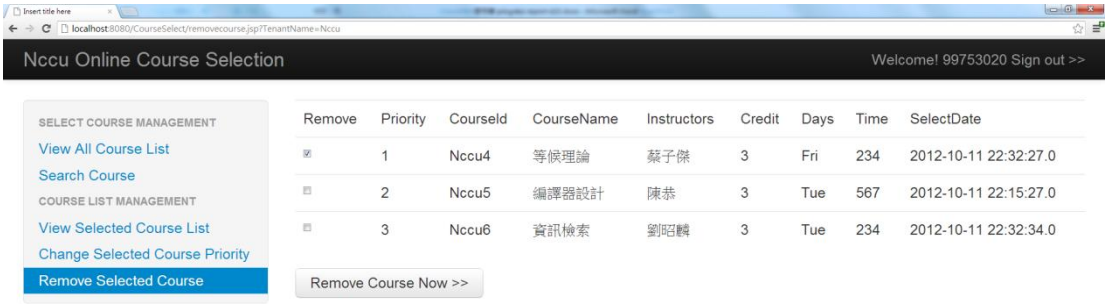

圖 4.15 學生從選課清單刪除課程的展示。

網站內部取得學生欲刪除課程的 CourseId 之後，組合產生下列的 DELETE 語句去進行刪除的動作：

```
DELETE SelectCourse WHERE SelectId = 1
```

學生可以透過選課清單的查詢得知DELETE動作是否有成功執行(圖 4.16)。

圖 4.16 學生從選課清單刪除課程的結果展示。

5. SELECT 語句搭配 WHERE 條件式和 JOIN 指令：

一對一、多對一、多對多的資料關係在一個應用程式中是很常見的，這裡以學生進行查詢選課清單為範例，透過 JOIN 指令去鏈結兩個存在於多校(租戶)選課系統的資料表並進行查詢動作。以下是 StudentId 為 99753020 的該位同學進行查詢選課清單時所需的 SELECT 語句以及查詢結果(圖 4.17):

```
SELECT SelectCourse.Priority , CourseInfo.CourseId,CourseInfo.CourseName ,
        CourseInfo.Instructors ,CourseInfo.Credit ,CourseInfo.Days,
        CourseInfo.Time, SelectCourse.SelectDate
FROM SelectCourse
INNER JOIN CourseInfo
ON SelectCourse.CourseId = CourseInfo.CourseId
WHERE SelectCourse.StudentId = '99753020'
ORDER BY SelectCourse.Priority ASC
```

圖 4.17 學生查詢選課清單的 SELECT 語句與結果。

上述提到的 ALTER 語句可以滿足租戶適度地客製化其資料表綱要的需求，除此之外，以 Extension Table Layout 的概念去設計多租戶應用程式之資料層級的另一項優點是可以提升軟體開發人員方便統一管理租戶的共有欄位以及其資料的能力。從下面的解說可以很清楚了解這項優點。

假定多校(租戶)選課系統要進行系統更新，同時，提供一項新服務給各個學校(租戶)，此時，軟體開發人員可能必須修改既有的資料表以協助該項新服務的開發。假定軟體開發人員要提供一項新服務讓學校可以將不屬於該學期的課程資料關閉，防止顯示在學生選課的資訊頁面，此時，軟體開發人員可能需要新增一個共有欄位(即 IsClosed)在所有學校的 CourseInfo 資料表中，透過學校對此共有欄位的設定讓應用程式判斷是否將該筆課程資料顯示在學生選課的資訊頁面中，若該筆課程資料的 IsClosed 欄位數值為 1，則代表不屬於該學期的課程資料，所以，不顯示該筆課程資料在學生選課的資訊頁面中，反之，若 IsClosed 欄位數值為 0，則表示其屬於該學期的課程資料，則該筆課程資料會顯示在學生選課的資訊頁面中以供學生選擇。

倘若採用 Private Table Layout 的概念去進行該多校(租戶)選課系統的資

料層級設計，以上述情形而言，軟體開發人員為了開發該項新功能必需逐一為每個租戶產生 ALTER 語句去新增 IsClosed 這個共有欄位到其 CourseInfo 資料表中，假定該系統目前有三個租戶(Nccu、Fju 以及 Tku)，則必須產生三個不同的 ALTER 語句(圖 4.18)去完成新增共有欄位的動作。若當租戶數量或是影響的欄位數量增加，相對地會增加時間成本；同樣的情況下，若改採用 Extension Table Layout 的概念去進行該多校(租戶)選課系統的資料層級設計，軟體開發人員只需要產生一個 ALTER 語句(圖 4.19)即可完成新增共有欄位的動作。

```
ALTER TABLE NccuCourseInfo ADD IsClosed Integer
ALTER TABLE FjuCourseInfo ADD IsClosed Integer
ALTER TABLE TkuCourseInfo ADD IsClosed Integer
```

圖 4.18 依 Private Table Layout 的概念設計其資料層級，
進行新增共有欄位的 ALTER 語句範例。

```
ALTER TABLE CourseInfoCommonFields ADD IsClosed Integer
```

圖 4.19 依 Extension Table Layout 的概念設計其資料層級，
進行新增共有欄位的 ALTER 語句範例。

從上述的這些實際例子看來，可以發現軟體開發人員在採用本系統工具的協助下，以 Extension Table Layout 的概念進行多租戶應用程式之資料層級設計，除了可以藉由租戶以自助的方式去更新其私有資料表以滿足租戶適度地客製化其資料表綱要的需求外，亦可以方便軟體開發人員統一管理租戶的共有欄位及其資料，藉此降低開發上的困難。

4.2 系統效能之影響因素

本系統工具為了協助軟體開發人員能夠沿用一租戶一資料表的 SQL 語句邏輯對

以 Extension Table Layout 的資料存放方式儲存的資料進行增刪修改，所以，將 SQL 語句依照其語句類型分別轉換成以 Extension Table Layout 邏輯表達的 SQL 語句並在資料庫中執行。

由於本系統工具在部份設計上採取共用資料表的架構，所以，初步推斷租戶的數量會對本系統進行語句轉換的時間造成影響。接下來，將進行一連串實驗分別模擬三種不同租戶數量的實驗環境，分別存在著 10、50、100 個租戶，並沿續 4.1.2 小節所展示的多校(租戶)選課系統之資料表集合設計，每個租戶的資料表集合中包含了三個資料表(即 4.1.2 小節所提到的 CourseInfo、Select-Course 以及 StudentInfo 資料表)，每個資料表中至少有 5 個共有欄位存在，其中，CourseInfo 以及 StudentInfo 資料表中各自存放了 500 筆課程資料以及學生資料，相對地，SelectCourse 資料表的資料筆數比較多，存放了 5000 筆的選課記錄在 SelectCourse 資料表中。此外，為了展現出租戶擁有客製化欄位的能力，部份租戶的資料表除了共有欄位以外，隨機產生 1~3 個私有欄位(即客製化欄位)，讓實驗的環境更接近事實。

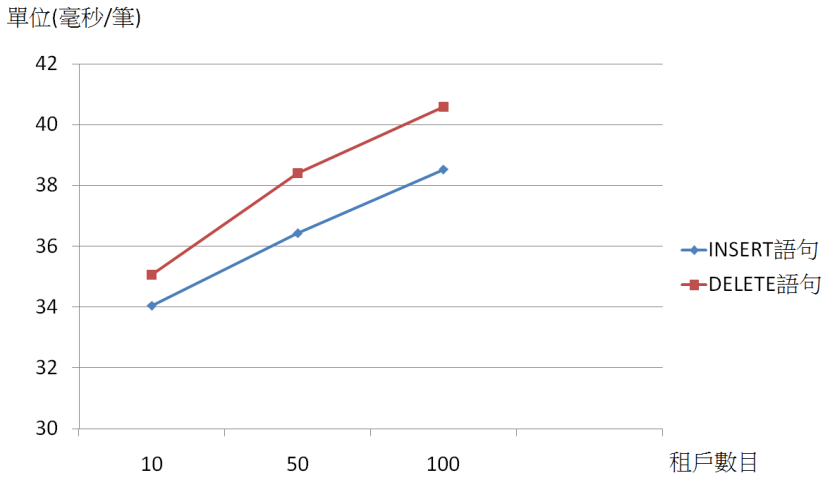

圖 4.20 租戶的數量對 INSERT、DELETE 語句造成的影響。

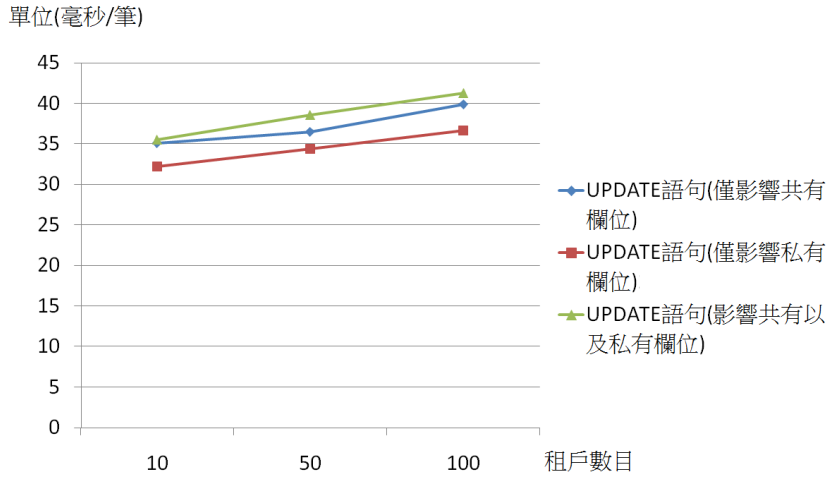

圖 4.21 租戶的數量對 UPDATE 語句造成的影響。

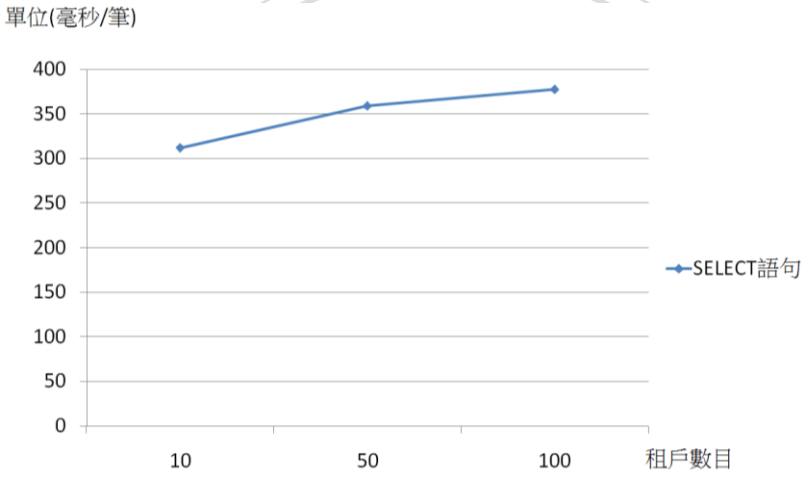

圖 4.22 租戶的數量對 SELECT 語句造成的影響。

在三種實驗環境中，分別進行六個實驗項目，分別是：(1)進行 1000 筆隨機租戶的 INSERT 語句、(2)進行 1000 筆隨機租戶的 DELETE 語句、(3)進行 1000 筆隨機租戶的 UPDATE 語句(僅影響共有欄位)、(4)進行 1000 筆隨機租戶的 UPDATE 語句(僅影響私有欄位)、(5)進行 1000 筆隨機租戶的 UPDATE 語句(影響共有以及私有欄位)、(6)進行 5000 筆資料的 SELECT 語句，實驗結果如圖 4.20、圖 4.21 以及圖 4.22 所示，觀察曲線圖後可以明顯發現當租戶的數量增加，六個實驗項目的執行時間也隨之明顯增加，所以，可以推斷租戶的數量會影響本系統工具在進行語句轉換的執行時間。

下一章節同樣延續 4.1.2 小節所採用的多校(租戶)選課系統之資料表集合

為範例，分別在(1)Private Table Layout、(2) Extension Table Layout、(3) Extension Table Layout 的詮釋欄位建立索引(Index) 這三種測試環境中進行多項測試，藉此更進一步地了解實際採用本系統工具的語句轉換協助會在效能上造成多大的影響。

4.3 系統效能測試

本系統工具主要是協助軟體開發人員以 Extension Table Layout 的概念進行多租戶應用程式之資料層級開發而設計，因此，這一小節將透過實際測試去了解採用本系統工具的語句轉換協助會在效能上造成多大的影響。

4.3.1 效能測試環境、測試項目、測試對象

為了模擬資料庫中有多租戶存在的情形，在進行效能測試之前，建立 10 個租戶資料表集合，沿續 4.1.2 小節所展示的多校(租戶)選課系統之資料表集合設計，每個租戶的資料表集合中包含了三個資料表(即 4.1.2 小節所提到的 CourseInfo、SelectCourse 以及 StudentInfo 資料表)，每個資料表中至少有 5 個共有欄位存在，其中，CourseInfo 以及 StudentInfo 資料表中各自存放了 500 筆課程資料以及 5000 筆學生資料，相對地，SelectCourse 資料表的資料筆數最多，存放了 50000 筆的選課記錄在 SelectCourse 資料表中。此外，為了展現出租戶擁有客製化欄位的能力，部份租戶的 CourseInfo 資料表除了共有欄位外，隨機產生 1~3 個私有欄位(即客製化欄位)，讓效能測試的環境更接近事實。

在測試項目的部分，針對較常見的 SQL 語句制訂七個測試項目，分別是：(1)進行 1000 筆隨機租戶的 INSERT 語句、(2)進行 1000 筆隨機租戶的 DELETE 語句、(3)進行 1000 筆隨機租戶的 UPDATE 語句(僅影響共有欄位)、(4)進行 1000 筆隨機租戶的 UPDATE 語句(僅影響私有欄位)、(5)進行 1000 筆隨機租戶的 UPDATE

語句(影響共有以及私有欄位)、(6)進行 50000 筆資料的 SELECT 語句、(7)進行 50000 X 500 筆資料的 SELECT with JOIN 語句。在七個測試項目中，以 UPDATE 語句測試較為特別，由於 UPDATE 語句在進行語句轉換時，會根據其影響的欄位而產生不同數量的轉換語句，以測試項目三(四)為例，根據 3.4.4 小節的轉換步驟可以知道，若僅影響共有(私有)欄位，在經本系統工具的語句轉換後只會產生一個 UPDATE 語句；反之，若同時影響到共有欄位以及私有欄位，透過本系統工具的語句轉換後會產生兩個 UPDATE 語句，所以，針對這個特性，將 UPDATE 語句的測試分成僅影響共有欄位、僅影響私有欄位以及影響共有以及私有欄位三種類型，即測試項目三、四、五。對於每一個測試項目來說，分別進行 12 次的測試，最後，將 12 筆的測試結果屏除其最高值跟最低值後所得到的 10 筆數據進行平均計算，得到效能測試的最終結果。

在測試對象的部分，分成一個控制組、兩個實驗組，控制組主要是模擬 Private Table Layout 的概念進行多租戶應用程式之資料層級設計，所有資料皆以 Private Table Layout 的資料存放方式來儲存；在實驗組方面，其一是將資料改成以 Extension Table Layout 的資料存放方式來儲存，透過執行本系統工具轉換後的語句對資料進行增刪修改，另外一個則是將資料改成以 Extension Table Layout 的資料存放方式來儲存後，並在其詮釋欄位(即 TenantId、Row)建立索引(Index)，同樣地透過執行本系統工具轉換後的語句對資料進行增刪修改，預期可以加快搜尋的速度。

最後，在測試環境建立後，將七個測試項目逐一在三個測試對象上分別執行，對 12 筆測試結果進行計算後得到效能測試的最終結果。接下來，根據測試結果，對每一個實驗組和控制組進行比較，進一步地對其效能做出評估。

4.3.2 Extension Table Layout 效能測試

根據表 4.1 所示，以 Extension Table Layout 的資料存放方式來儲存資料並且

沒有在其詮釋欄位建立索引(Index)的情況下，讓軟體開發人員以一租戶一資料表的寫法撰寫 SQL 語句，再執行本系統工具進行轉換後的語句之測試結果，在七個測試項目中，表現皆落後給 Private Table Layout。其中又以測試項目六以及測試項目七在時間上的影響最為嚴重，判斷可能是因為進行 SELECT 語句轉換過程中，必須先額外執行 INNER JOIN 指令將共用資料表以及私有資料表鏈結後才對鏈結後所產生的資料表進行實際的查詢動作，所以，造成執行時間增加最多。

表 4.1 Private Table Layout 和 Extension Table Layout 測試數據表。

測試項目	Private Table Layout	Extension Table Layout
INSERT 語句	23.01	36.11
DELETE 語句	23.87	38.13
UPDATE 語句(僅影響共有欄位)	23.80	36.88
UPDATE 語句(僅影響私有欄位)	23.54	36.33
UPDATE 語句(影響共有以及私有欄位)	24.82	40.57
SELECT 語句	671.2	922.1
SELECT with JOIN 語句	815.0	1603.8

單位(毫秒/筆)

4.3.3 Extension Table Layout 詮釋欄位建立 Index 之效能測試

表 4.2 的第四欄數據是以 Extension Table Layout 的資料存放方式來儲存資料並且在其詮釋欄位建立索引(Index)的情況下，讓軟體開發人員以一租戶一資料表的寫法撰寫 SQL 語句，再執行本系統工具進行轉換後的語句之測試結果可以發現在所有的測試項目都比沒有建立索引(Index)來得好，除此之外，在多個測試項目中，平均執行時間也和採用 Private Table Layout 的測試結果不相上下。

從比較圖(即圖 4.23、4.24 以及圖 4.25)看來，事實上，採用 Extension Table

Layout 的方式並且在其詮釋欄位建立索引(Index)進行多租戶應用程式之資料層級設計並不會對執行時間造成太大的影響。因此，若透過本系統工具協助之下，可以讓軟體開發人員更容易利用 Extension Table Layout 的概念去進行多租戶應用程式之資料層級設計，同時，亦可以享受到採用 Extension Table Layout 的概念進行多租戶應用程式之資料層級設計的好處，除了讓租戶適度地擁有客製化欄位的能力外，亦可以提升軟體開發人員統一管理租戶的共有欄位以及其資料的能力，降低開發、維運多租戶應用程式的複雜度。

表 4.2 Private Table Layout、Extension Table Layout 和 Extension Table Layout with index of metadata 測試數據表。

測試項目	Private Table Layout	Extension Table Layout	Extension Table Layout with index of metadata
INSERT 語句	23.01	36.11	35.65
DELETE 語句	23.87	38.13	37.88
UPDATE 語句(僅影響共有欄位)	23.80	36.88	36.33
UPDATE 語句(僅影響私有欄位)	23.54	36.33	35.91
UPDATE 語句(影響共有以及私有欄位)	24.82	40.57	40.02
SELECT 語句	671.2	922.1	831.3
SELECT with JOIN 語句	815.0	1603.8	1123.7

單位(毫秒/筆)

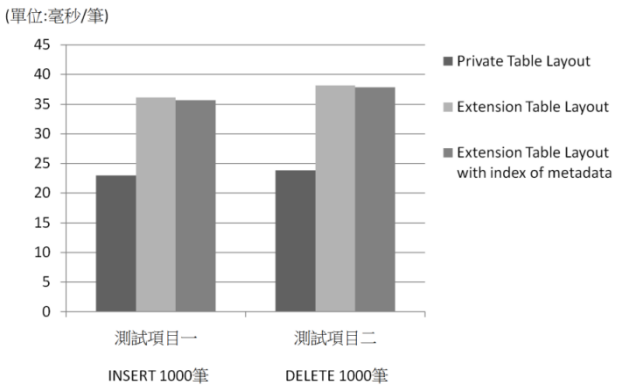

圖 4.23 Private Table Layout、Extension Table Layout 和 Extension Table Layout with index of metadata 在測試項目一與測試項目二的比較圖。

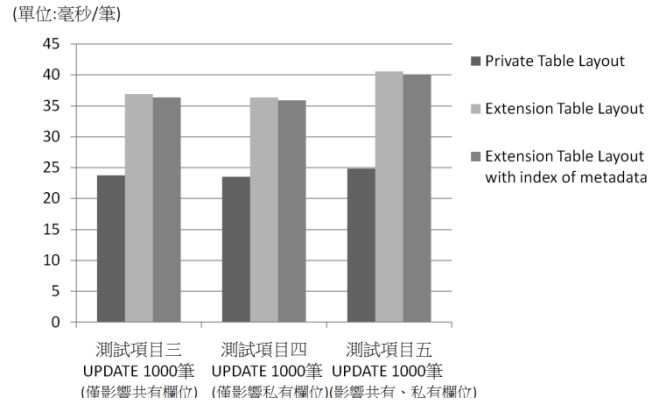

圖 4.24 Private Table Layout、Extension Table Layout 和

Extension Table Layout with index of metadata 在測試項目三、四以及測試項目五的比較圖

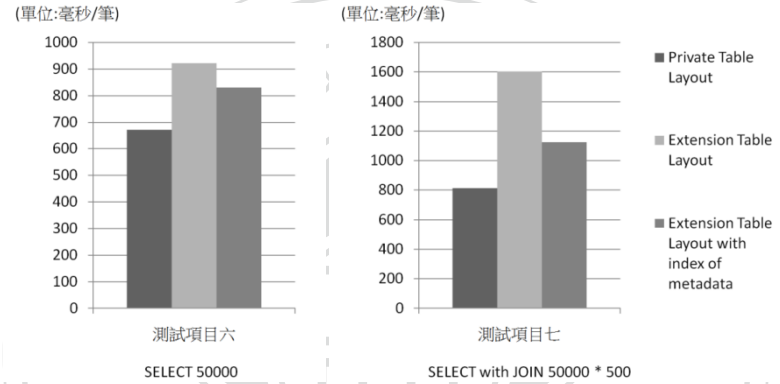

圖 4.25 Private Table Layout、Extension Table Layout 和

Extension Table Layout with index of metadata 在測試項目六與測試項目七的比較圖

National Chengchi University

第五章

結論與未來研究方向

5.1 結論

多租戶技術是未來值得期待的一項發展。對於軟體開發人員來說，透過租戶共享軟硬體設備，除了減少開發成本，同時，對往後應用程式的維運也比較容易；以租戶的角度而言，除了減少租用應用程式的費用，也可以一併享受應用程式每次更新時所提供的新服務，另外，亦可以透過其客製化應用程式的特點來增加使用上的彈性。因此，多租戶技術無論是對軟體開發人員或是租戶來說都可以創造許多實值上的效益。

為了達到多租戶技術所提出的租戶資源共享概念，本研究將多租戶應用程式之資料層級以多租戶共用資料庫的架構來設計，同時，改以 Extension Table Layout 的資料存放方式來儲存資料，以達到既可以滿足租戶適度地客製化其資料表綱要，又可以提升軟體開發人員統一管理租戶的共有欄位以及其資料的雙重目標。

以 Extension Table Layout 的資料存放方式儲存資料雖較適合多租戶應用程式，但對軟體開發人員來說，其 SQL 語句的表達邏輯為了配合實際資料的存放方式，所以，會和一租戶一資料表的寫法有些不同。為了解決這個問題，本研究實作一套系統工具進行語句轉換協助軟體開發人員雖以 Extension Table Layout 的概念去進行多租戶應用程式之資料層級設計，卻可以沿用一租戶一資料表的寫法進行 SQL 語句的表達。轉換過程中，藉由改寫 JDBC 的部分 API 去攔截軟體開發人員所撰寫的 SQL 語句，緊接著，利用 JSqlParser 進行原 SQL 語句

的結構剖析以取得後續轉換時所需的語句元素，最後，重組獲得的語句元素，將 SQL 語句轉換成以 Extension Table Layout 邏輯表達的 SQL 語句並在資料庫中執行。

在實際測試的部分，以採用本系統工具協助開發多校(租戶)選課系統為例來進行功能展示。在效能的部分，由於本系統工具在設計上部分採取共用資料表的架構，從實驗數據結果看來，可以知道租戶的數量會影響本系統工具在進行語句轉換的執行時間。最後，實際進行多項效能測試和 Private Table Layout 比較，從結果顯示，以 Extension Table Layout 的概念並且在其詮釋欄位建立索引(Index)的方式去設計多租戶應用程式之資料層級，軟體開發人員沿用一租戶一資料表寫法來撰寫 SQL 語句，再藉由本系統工具進行語句轉換的這種開發模式在各類型語句的執行時間上並不會造成太大的影響，並且可以在租戶共享資料庫的理念下，既滿足租戶客製化欄位的需求又可以提升軟體開發人員統一管理租戶的共有欄位以及其資料的能力。

5.2 未來研究方向

多租戶技術在資料層級的部份，透過資料存放方式的改變，確實可以創造許多好處給租戶或是軟體開發人員，但還是有一些技術上的困難點，包括加強租戶間的資料隔離性、資料自動轉移等。最後，提出幾個未來值得研究的議題：

1. 本研究暫以租戶識別碼(TenantId)作為資料隔離的主要機制，在資料安全性的考量下，從租戶的心靈層面看來仍有些疑慮，但若能在應用程式層級提供更完善的機制去防止惡意的用戶利用 TenantId 來偽裝自己的身分竊取其他用戶的資料，相信一定能夠提高資料的安全性。
2. 期望未來能夠在極少的更動之下，開發一套系統工具協助資料進行系統化地轉移，改以 Extension Table Layout 的資料存放方式來儲存資料。接下

來，透過本系統工具進行語句轉換，讓軟體開發人員能夠沿用一租戶一資料表的寫法來撰寫 SQL 語句。

3. 由於本研究在部份設計上採取共用資料表的架構，從 4.2 小節所進行的實驗推斷租戶的數量會影響本系統工具進行語句轉換的執行時間，因此，當租戶的數量大幅增加以後，要如何維持資料庫存取、查詢的效能在一定的水準也是未來在實際使用本系統工具時需要解決的問題。

4. 如何在共用資料表的架構下，妥善處理 Integrity Constraints 是未來待解決的問題。如同 3.4.2 小節所提到的，目前本系統工具在實作 Integrity Constraints 的處理，主要是透過資料庫所提供的多欄位限制加上 TenantId 的概念來完成，其他目前尚未支援的部份或許之後也可以仿造上述的方式來解決。

國立政治大學 National Chengchi University

參考文獻

[1] [Chong et al.06] F. Chong, G. Carraro, and R. Wolter, "Multi-Tenant DataArchitecture," http://msdn.microsoft.com/en-us/library/aa479086.aspx

[2] [Mell et al.11] P. Mell, and T. Grance, "Recommendations of the National Institute of Standards and Technology," in The NIST Definition of Cloud Computing, 2011.

[3] [Force.com] The Design of the Force.com Multitenant Internet Application Development Platform

[4] [Aulbach et al.09] Stefan Aulbach, D. Jacobs, A. Kemper, M. Seibold, "A Comparison of Flexible Schemas for Software as a Service," in SIGMOD'09, June 29–July 2, 2009.

[5] [Aulbach et al.08] Stefan Aulbach, T. Grust, D. Jacobs, A. Kemper, J. Rittinger, "Multi-Tenant Databases for Software as a Service: Schema-Mapping Techniques," in SIGMOD'08, June 9–12, 2008.

[6] [Foping et al. 07] Franclin S. Foping, I. M. Dokas, J. Feehan, S. Imran, "A New Hybrid Schema-Sharing Technique for Multitenant Applications".

[7] [Univ et al. 10] Zhejiang Univ., Hangzhou, "Transforming relational database into HBase: A case study," in IEEE'10, July 16–18, 2010.

[8] [Hisashi et al. 10] SHIMAMURA Hisashi, S. Kenji, K. Takayuki, N.Shoji, "Realization of the High-density SaaS Infrastructure with a Fine-grained Multitenant Framework".

[9] [Ramasubramanian Thiyagarajan et al. 10] Ramasubramanian Thiyagarajan, S. Kuppusamy, "Enabling Multi-Tenancy in Web Applications," in devx.com, June 10, 2010.

[10] [Scott Chate 10] Scott Chate, "Convert your web application to a multi-tenant

SaaS solution," in ibm.com, Dec 14, 2010.

[11] [Martin Grund et al. 08] Martin Grund, M. Schapranow, J. Krueger, J. Schaffner, A. Bog, "Shared Table Access Pattern Analysis for Multi-Tenant Applications," in IEEE'08, 2008.

[12] [Alfons Kemper] Alfons Kemper, "Database Technology for SaaS(Software as a Service)".

[13] 吳定威,「支援多租戶應用程式的 SQL 語句轉換機制」,國立政治大學資訊科學系碩士學位論文,台北,民國 101 年 1 月。

[14] JSqlParser http://jsqlparser.sourceforge.net/

[15] SQL Statement tutorial http://www.w3schools.com/sql/default.asp

[16] Eclipse http://www.eclipse.org/

[17] JSP tutorial http://www.jsptut.com/

[18] Apache Tomcat http://tomcat.apache.org/

[19] MySQL http://www.mysql.com/

[20] Databse Design-Design pattern: many-to-many http://www.tomjewett.com/dbdesign/dbdesign.php?page=manymany.php

[21] Data Integrity http://msdn.microsoft.com/en-us/library/aa933058(v=sql.80).aspx

[22] Bootstrap http://twitter.github.com/bootstrap/

[23] XAMPP http://www.apachefriends.org/en/xampp.html

[24] XAMPP tutorial http://www.mediawiki.org/wiki/Manual:Running_MediaWiki_on_XAMPP

[25] JDBC http://www.oracle.com/technetwork/java/javase/jdbc/index.html

[26] W3C http://www.w3.org/

[27] The Long Tail http://www.longtail.com/the_long_tail/about.html

[28] UML http://www.uml.org/

附錄一

[[ ]] **::** SQL statement using Private Table Layout→ SQL statement using Extension Table Layout .

//Notations introduction

- **Bold** fonts is a string in the SQL statement, for example, **CREATE TABLE**
- *Italic* fonts is a element in the SQL statement, for example, *TableName* is the table name of the statement , *ColumnNames* is a list of impacted column names in the statement ; *ColumnValues* is a list of impacted column values in the statement .
- *string1* $\bigoplus$ *string2* : return a new string of the concatenation *string1* and *string2* .
- *string* $\ominus$ *substring* : delete a substring (*substring*) from the end of the string (*string*).
- *GetAllCN()* : scan Column_Metadata Table to return a list of all common table names .
- $\mathcal{M}$*coln*(*ColumnNames*) : return two lists : *CommonColumnNames* , *PrivateColumnNames* .

  $CommonColumnNames = \{ col \mid col$ is a common table column name , col in $ColumnNames. \}$

  $PrivateColumnNames = \{ col \mid col$ is a private table column name , col in $ColumnNames. \}$
- $\mathcal{M}$*colv*(*ColumnValues*) : return two list : *CommonColumnValues* , *PrivateColumnValues*

  $CommonColumnValues = \{ val \mid val$ is a common table column value , val in $ColumnValues. \}$

  $PrivateColumnValues = \{ val \mid val$ is a private table column value , val in $ColumnValues. \}$
- $\mathcal{M}$*ue*(*UpdateExpressions*) : return two lists : *CommonUpdateExpressions* , *PrivateUpdateExpressions* .

$$CommonUpdateExpressions = \{\ col = val \mid col = val \text{ in UpdateExpressions , } col \text{ is a common table column name.}\ \}$$

$$PrivateUpdateExpressions = \{\ col = val \mid col = val \text{ in UpdateExpressions , } col \text{ is a private table column name.}\ \}$$

- *CtoP*(*TableName , TenantId*) : $TenantId \oplus TableName \ominus$ **CommonFields** , return a private table name that converted from a given common table name (*TableName*) and tenant id (*TenantId*) .
- *Mcn*(*TableName*) : $TableName \oplus$ **CommonFields** , return a common table name.
- *Mpn*(*TableName,TenantId*) : $TenantId \oplus TableName$, return a private table name.
- *GetMR*(*TableName*) : get the maximum row in a given table (*TableName*).
- *GenS*(*Statement*) : generate a statement from a given statement(*Statement*)

國立政治大學 National Chengchi University

//Rules for CREATE statement, for example,

CREATE TABLE *TableName* **(** *ColumnDefinitions* **)**

1. [[(**CREATE TABLE** *TableName* **(** *ColumnDefinitions* **)**)]]

= *GenS(* **CREATE TABLE** *TableName*

(TenantId Char (50) not null , Row Integer not null ,

ColumnDefinitions ,

Primary key (TenantId , Row)))

//Rules for CREATE EXTENSION TABLE statement, for example,

CREATE EXTENSION TABLE *TenantId*

2. [[(**CREATE EXTENSION TABLE** *TenantId*)]]

= FOR *CommonTableName* IN *GetAllCN()*

DO

GenS(**CREATE TABLE** *CtoP(CommonTableName , tenantId*)

(TenantId Char (50) not null ,

Row Integer not null ,

Primary key (TenantId , Row)))

END

//Rules for ALTER statement, for example,

ALTER TABLE *TableName AlterType AlterExpression*

3. [[(**ALTER TABLE** *TableName AlterType AlterExpression, TenantId*)]]

= *GenS* (**ALTER** *Mpn*(*TableName , TenantId*) *AlterType AlterExpression*)

4. [[(**ALTER TABLE** *TableName AlterType AlterExpression*)]]

= *GenS* (**ALTER** *TableName AlterType AlterExpression*)

//Rules for INSERT statement, for example,

INSERT INTO *TableName* **(** *ColumnNames* **)**

VALUES (*ColumnValues* **)**

5. [[(**INSERT INTO** *TableName* **(** *ColumnNames* **) VALUES (** *ColumnValues* **)** ,

TenantId)]]

= (*CommonColumnNames , PrivateColumnNames*) ← $\mathcal{M}$*col*$_n$(*ColumnNames*) ;

(*CommonColumnValues , PrivateColumnValues*) ← $\mathcal{M}$*col*$_v$(*ColumnValues*) ;

GenS (**INSERT INTO** $\mathcal{M}$*cn*(*TableName*)

(TenantId , Row , *CommonColumnNames* **)**

VALUES

(*tenantid* **,** *GetMR*($\mathcal{M}$*pn*(*TableName,TenantId*)) , *CommonColumnValues* **)**) ;

GenS (**INSERT INTO** $\mathcal{M}$*pn*(*TableName , TenantId*)

(TenantId , Row , *PrivateColumnNames* **)**

VALUES

(*TenantId* **,** *GetMR*($\mathcal{M}$*pn*(*TableName , TenantId*)) , *PrivateColumnValues* **)**)

國立政治大學 National Chengchi University

//Rules for UPDATE statement, for example,

UPDATE *TableName*

SET *UpdateExpressions*

WHERE *Conditions* , where *UpdateEexpressions* is like $ColumnName_1$ = $ColumnValue_1$ **,** **..,** $ColumnName_n$ = $ColumnValue_n$

6. [[(**UPDATE** *TableName* **SET** *UpdateExpressions* **WHERE** *Conditions*, *TenantId*)]]

= (*CommonUpdateExpressions , PrivateUpdateExpressions*) ← *Mue*(*UpdateExpressions*) ;

FOR *row*

IN (**SELECT Row**

FROM

(*Mcn*(*TableName*)

INNER JOIN *Mpn*(*TableName , TenantId*)

ON *Mcn*(*TableName*)**.TenantId =** *Mpn*(*TableName , TenantId*)**.TenantId**

AND *Mcn*(*TableName*)**.Row =** *Mpn*(*TableName , TenantId*)**.Row)**

WHERE TenantId = *TenantId* **AND (***Conditions* **)**)

DO

GenS(**UPDATE** *Mcn*(*TableName*)

SET *CommonUpdateExpressions*

WHERE TenantId = *TenantId* **AND Row =** *Row*) ;

GenS(**UPDATE** *Mpn*(*TableName,TenantId*)

SET *PrivateUpdateExpressions*

WHERE TenantId = *TenantId* **AND Row =** *Row*)

END

國立政治大學 National Chengchi University

//Rules for DELETE statement, for example,

DELETE FROM *TableName*

WHERE *Conditions*

7. [[(**DELETE FROM** *TableNme* **WHERE** *Conditions*, *TenantId*)]]

= FOR *Row*

IN (**SELECT Row**

FROM

(*Mcn*(*TableName*)

INNER JOIN *Mpn*(*TableName , TenantId*)

ON *Mcn*(*TableName*)**.TenantId =** *Mpn*(*TableName , TenantId*)**.TenantId**

AND *Mcn*(*TableName*)**.Row =** *Mpn*(*TableName , TenantId*)**.Row)**

WHERE TenantId = *TenantId* **AND (***Conditions* **)**)

DO

GenS(**DELETE** *Mcn*(*TableName*)

WHERE TenantId = *TenantId* **AND Row =** *Row*) ;

GenS(**DELETE** *Mpn*(*TableName , TenantId*)

WHERE TenantId = *TenantId* **AND Row =** *Row*)

END

//Rules for SELECT statement, for example,

SELECT *ColumnNames*

FROM *TableName*

WHERE *Conditions*

8. [[(**SELECT** *ColumnNames* **FROM** *TableName* **WHERE** *Conditions*, *TenantId*)]]

= *GenS(* **SELECT** *ColumnNames*

FROM

(SELECT *

FROM *Mcn*(*TableName*)

INNER JOIN *Mpn*(*TableName , TenantId*)

ON *Mcn*(*TableName*)**.TenantId =** *Mpn*(*TableName , TenantId*)**.TenantId**

AND *Mcn*(*TableName*)**.Row =** *Mpn*(*TableName , TenantId*)**.Row)**

AS *TableName*

WHERE *Conditions*)

國立政治大學 National Chengchi University