國立臺灣海洋大學

資訊工程學系

碩士學位論文

指導教授：張雅惠博士

快速處裡生產流程查詢與控管之研究

The Research on Efficiently Supporting the Control and Querying on Manufacturing Processes

研究生：李韋錫　撰

中華民國 104 年 2 月

快速處裡生產流程查詢與控管之研究

研 究 生：李韋錫　　Student：Wei-Hsi Lee

指導教授：張雅惠　　Advisor：Ya-Hui Chang

國立臺灣海洋大學

資訊工程學系

碩士論文

A Thesis
Submitted to Department of Computer Science and Engineering
College of Electrical Engineering and Computer Scienece
National Taiwan Ocean University
In Partial Fulfillment of the Requirements
for the Degree of
Master of Science
in
Computer Science and Engineering
January 2015
Keelung, Taiwan, Republic of China

中華民國 104 年 2 月

摘要

製造執行系統（Manufacturing Execution Systems，MES）指的是專門使用於工廠製造的電腦系統，主要是提供物品在生產流程當中於各個站台的數據資料。本論文討論如何利用在 NAS（Network Attached Storage）上支援的 MYSQL 和 PHP 實作出具有基本功能的 MES 系統，簡稱 NMES。針對其中的核心功能，也就是生產流程查詢，我們提出兩種作法。第一個方法是利用原始的條碼資料，並根據使用者下達的條件組成合適的 SQL 查詢句，此方法稱為「Baseline 方法」，但是其查詢效率可能會有不佳的情形。因此我們又提出第二種方法。此方法將原始條碼資料轉換成追蹤資料，並根據物品經過的站台流程與時間來建立流程時間樹，再利用此樹設計對應的 SQL 查詢句並篩選錯誤流程的條碼，此方法稱為「FTT(Flow-Time Tree)方法」。最後，透過一系列的實驗以及不同數量的資料集，比較此二方法的查詢效率與中間資料量變化。實驗結果顯示，FTT 方法在查詢效率方面大多優於 Baseline 方法，而在資料量部分，FTT 方法所處理的資料量也比 Baseline 要來的少，使得查詢速度比較快。

Abstract

Manufacturing Execution Systems (MES) are used in factories to provide the information about items during the production process. In this thesis, we study how to use the MYSQL and PHP supported in NAS to build an MES with basic functions. We focus on the core functions of an MES, that is, supporting process queries, and propose two methods. The Baseline method uses the original barcode data transmitted directly from the items to form SQL queries according to the search conditions provided by users. However, this method might be inefficient. We therefore propose another method, called the FTT method. It will convert the original barcode data into tracking data, and build flow-time trees. These trees will be represented by several relations, which are used as bases to form SQL queries and to detect data with incorrect process flow. We have performed a series of experiments based on a variety of datasets to compare amounts of intermediate results and the efficiency of the proposed two methods. Experimental results show that although the FTT method requires pre-built flow-time trees, it performs better than the Baseline method in terms of efficiency and the amount of intermediate results.

誌謝

首先，感謝指導教授張雅惠博士，對於本論文給予許多幫助，且在研究論文期間不時地共同討論，解決論文的諸多疑點，使學生順利的完成論文，同時也讓學生獲益匪淺，在此向敬愛的老師致上衷心謝忱。

除此之外，感謝口試委員許為元博士和劉傳銘博士，細心審稿以及修正論文，並對本論文提供寶貴的建議，使本論文更趨近於嚴謹完善，在此深表謝意。

最後，感謝威強電集團的 Kenny 與 Anderson 所率領的團隊，於建教合作案「NAS 製造執行系統研發(I)」中提供工廠製造管理流程的相關知識與技術指導。感謝學長姊，快速帶領我適應實驗室的環境。感謝實驗室的同學承翰與學弟妹陪我度過多采多姿的研究所生活。感謝國中麻吉們江藥師、許宸、褚哥、杰修、好騙，陪伴我走過 12 年歲月，在學習成長階段的支持與鼓勵。感謝親愛的爸爸、媽媽和弟弟，謝謝你們仔細的照顧我，讓我在生活中有堅強的依靠。特別要感謝 Sabrina 的陪伴，大多時候的假日都陪我待在咖啡廳裡做研究，有妳的鼓勵使我能全力以赴，專心解決問題並完成論文。感謝的人太多太多，在此一併致上謝意，謝謝你們。

目錄

圖目錄

表目錄

第 1 章序論

在此章，我們先敘述本論文的研究動機與目的，同時提出本論文的研究方法與貢獻，並針對相關研究進行討論，最後說明本論文各章節的內容規劃。

1.1 研究動機與目的

製造執行系統（Manufacturing Execution Systems，MES)指的是專門使用於工廠製造的電腦系統，此類系統有以下幾個主要的功用:

- 定義產品的整個生產過程（product life-cycle）
- 顯示目前工廠生產線的運作狀況、訂單的執行和分配，可供決策者用來管理或最佳化生產流程，以提高產品的輸出
- 可以讓管理者即時（real time）控制或記錄產品生產過程的不同要素，譬如數量的變化、分批或併批等

由上可知，一個好的 MES 系統可以協助一個公司或工廠做更有效率的管理。而另一方面，由威強電子公司 QNAP 所生產的 NAS 系統，也越來越受到中小企業組織的歡迎。NAS 原本為基於 UNIX 平台所設計的雲端儲存系統，在該類商品中屬於翹首的地位。但是隨著硬體功能越來越強大，NAS 不僅可以作為儲存之用，其上也可執行具有一般功能的應用系統。QNAP 公司特別提供專用的開發工具包 (QDK)，供系統開發者設計出可由遠端系統 (智慧手機或個人電腦) 上執行的軟體，來存取 NAS 文件或進行各項管理。在 QNAP 的二次開發網站[QNAP]上列舉了許多目前已經開發成功的系統，譬如捷銳行動科技開發的車隊管理系統等等。

目前市面上最主要的 MES 系統，為甲骨文公司（Oracle）所開發[OMES]，雖然功能完整，但是價格昂貴，且許多功能非一般公司所需要。所以，本論文的目的，就是在研究如何在 NAS 上開發具有基本功能的 MES 系統，以下簡稱 NMES，

以供一般中小型的工廠使用。而此系統提供的功能如：條碼資料擷取、生產流程查詢與控管、存貨流程管理、製品流程管理以及製作與列印條碼等。

1.2 研究方法與貢獻

在我們所建立的系統中，生產流程為最重要的一部分。其功能主要是追蹤一個物品或者是相同型號的物品，及物品經過站台的所有資訊，包含站台的讀取數量和讀取物品進、出站的時間。本系統將站台的條碼讀碼器所讀取到的資訊存到資料庫表格中，並且進一步分析資料以達到生產流程查詢的功能。而流程控管是當物品在出貨時，工單會先行規劃一個工單號碼以及該物品出貨的所屬批號和正確流程。當追蹤到一個物品沒有走到正確的流程，我們希望能找出錯誤資訊並記錄起來。

至於實作方面，我們將所有資訊儲存在 NAS 的關聯式資料庫(MYSQL)，在介面上的呈現則採用 PHP 程式語言和 SQL 語法來實作本系統。針對生產流程查詢，我們提出兩個做法，第一個做法是利用使用者下達的限制條件所組成適當的 SQL 語法，稱作「暴力法(Baseline)」，我們透過 SQL 語法的延伸表格(derived relation)找出我們要的中間資料(intermeditate)再藉由反覆查詢、比對資料以達到生產流程功能。不過方法一在比對資料時，可能會有效率不佳的情況。

我們希望能有更精準的限制條件來減少資料比對的次數，在此我們提出了第二個做法，在此方法中我們先將原始資料做前處理轉換成 trace records，接著利用這些 trace records 依據工單所定義的正確生產流程並將這些 trace records 依照起始站台的進、出站時間來分類，對每一類的 trace records 所經過的站台流程進行建樹，並將所有資訊存到資料表格，此方法稱作「FTT 方法(Flow-Time Tree)」。

本論文的貢獻總結如下所示：

1. 我們以 NAS 為開發平台，為中小型工廠提供基本功能的製造執行系統，使中小型工廠能花較少的預算也能有工作運作流程的管理。

2. 我們提出 FTT(Flow Time Tree)方法，以工單資訊的生產流程為建樹基礎，進而篩選出錯誤流程的物品資訊。

3. 最後，經由實驗來比較 Baseline 方法與 FTT 方法，實驗結果顯示，FTT 的中間資料數要比 Baseline 要來的少，也因此效率更加提昇。

1.3 相關研究

首先，我們先討論製造執行系統的相關研究。製造執行系統主要是從工廠接到訂單之後從事生產到產品完成之間，收集製造現場的各種資訊並且提供管理者各種的即時資訊，讓管理者可以隨時監控機台狀況並且有效的追蹤工單的製造流程以及預定的生產狀況，建置製造執行系統，能有效的縮短製造週期、確保產品品質、以及提供產線和管理者之間一個整合的溝通平台。在實際應用上[TCCY13]基於半導體測試，主要是研究著重在晶圓的管理，為了滿足不同客戶的需求和即時處理大規模訂單，所以提出了以 RFID 技術來實現零延遲和零庫存為目標，進而實現自動化過程、監測以及生產狀況，並改善良率。

其次是 RFID data tracking 的問題，在近幾年來已經有相當多的模組和技術在處理 RFID 資料並儲存在資料庫。[LC08]、[LC11]這兩篇論文提出 path encoding scheme 的方式，將每一筆 RFID 資料做整合，之後透過樹狀結構的方式儲存每個物品所經過的站台以及站台的時間資訊。其對樹狀結構的編碼方式分為兩種，其一為每個站台以質數(Prime)做為編碼，經由中國餘式定理(Chinese Remainder Theorem)對質數的關係計算出站台在樹狀結構中每條路徑獨特的值；其二是加入站台和時間的資訊於樹狀結構上，並將節點以 range encoding 方式編碼，以提供

路徑與時間的快速查詢。[LC13]提出的 IPES(improved prime encoding scheme)是為了要解決產品流程可能會有迴圈路徑“cycling path”的問題。在[NW11] [LC11] [TSH14]發現到在這些類別當中可能會有路徑過長而導致編碼溢位(overflow)也就是超過資料庫資料型態的長度限制，所以[NW11]進一步提出的 path encoding scheme 是基於尤拉公式(Euler's formula)對質數與整數經由壓縮計算成一個實數，使得 overflow 的值可以儲存至資料庫。而[LC11]更提出“divide a path”新的方法將過長的路徑做切割，如此一來就能將這些路徑片段儲存在資料庫，而[TSH14]提出了一個新的儲存框架來解決這樣的問題。

然而以上所設計的查詢句為 path-oriented，大多是使用者給定一個物品的流程，進而找出有那些物品會經過這些流程，但是對於流程的控管就沒有辦法用一個很有效的方法來判定一個物品走的這個生產流程是對還是錯，所以我們希望能以流程控管的概念，先將已知的流程建進樹狀結構，然後再快速查詢物品。

最後我們來介紹用分散式系統來處理大規模的 RFID 資料，[CSDS11]設計一個可擴充性的分散式系統來追蹤 RFID 資料和監測模擬，由於 RFID 缺乏容錯，因此[CSDS11]提出在單一架構下，結合容錯的推論和站台位置，來處理複雜的查詢句，藉由分布式的推論與查詢處理，使得效率和延展性都有較佳的效率，而[LXBC14]設計一個新的分散式系統模型“bloom filters”，傳統的分散式系統會使資料會重複出現(data redundancy)導致儲存與查詢不佳的形況，透過 bloom filters 與建立相對應的查詢句處理方案，並支援現有的查詢句與路徑查詢，使得在 RFID 資料能在空間和時間上得到較佳的效率。

1.4 論文架構

本論文其餘各章節的架構如下。第二章敘述本論文的相關系統架構，其中包含各個模組的主要功能與資料介面，藉以對本論文的研究有基礎的認識。第三章介紹資料庫的表格定義以及Baseline方法中之SQL查詢句。第四章介紹FTT方法，

說明其新的資料表示法以及如何從眾多的原始資料，經由樹狀結構轉換產生新的資料表格以及對應的SQL查詢句。第五章中以實驗比較兩種SQL的查詢句效率以及中間資料。最後於第六章提出本論文的結論與未來方向。

第 2 章系統功能介紹

在此章我們先介紹 NMES 的整體系統架構，接著再介紹各個模組的詳細功能。

2.1 系統架構

在本節中首先說明MES系統運作的環境。在工廠中物品在生產線上的狀況，是由每個站台(station)所附的條碼讀碼器（barcode reader）透過網路將條碼封包傳回 NAS。在此系統中，我們假設每個站台可能配備有進站(input)和出站(output)兩個條碼讀碼器，或只有一個條碼讀碼機。而一個工單(workorder)內包含同一個型號(product number)的數批物件(batch)，每個物件有一個唯一的序號(serial number)做代替。

我們架設在 NAS 上的 MES 系統主要包含一個網站伺服器（Apache Server）和資料庫伺服器（MySQL Server）。我們的網站程式會從原始的條碼資料擷取重要資訊存入資料庫伺服器，並提供如存貨管理、流程查詢等功能。在此系統中，我們規劃兩類使用者：(1)管理者（Admin）：具有所有的權限，包含輸入、修改和查詢資料；(2)作業員（Operator）：只具有查詢的權限。此兩類使用者皆是透過其個人電腦上的瀏覽器連到 NAS 上的網站進行相關操作，同時也可透過網頁進行標籤的製作與列印。在語言方面，我們也支援三種語系「繁體中文/簡體中文/English」。整體架構圖大致如圖 2-1 所示。

本系統依其功能共分為五個模組，如表 2-1 所示，而其相關性如圖 2-2 所示。其中模組 A 負責接收條碼封包，並將重要資訊擷取出來存放至資料庫中，而模組 B、C、D 則利用資料庫中的資料回答使用者的查詢，模組 E 則是一個單獨的模組，可提供條碼的製作與列印。在下一節中，我們將會對各模組提供更進一步的說明。

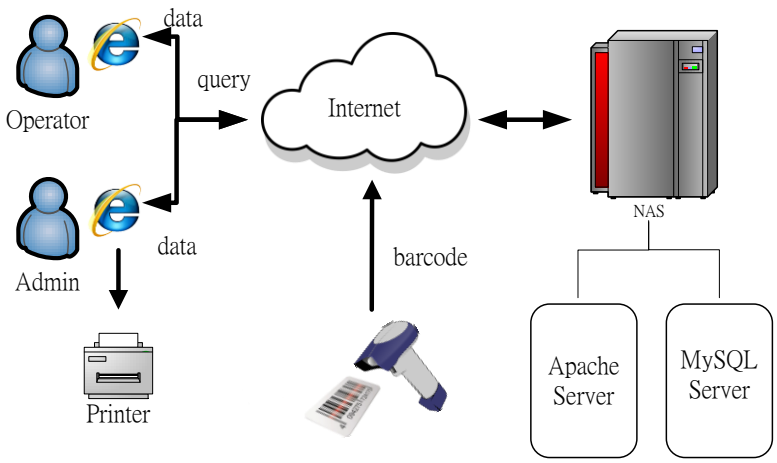


圖 2-1：NMES 運作環境圖

表 2-1：模組列表

模組編號	模組名稱
A	條碼處理模組
B	生產流程查詢模組
C	存貨管理模組
D	製品流程管理模組
E	條碼製作與標籤列印

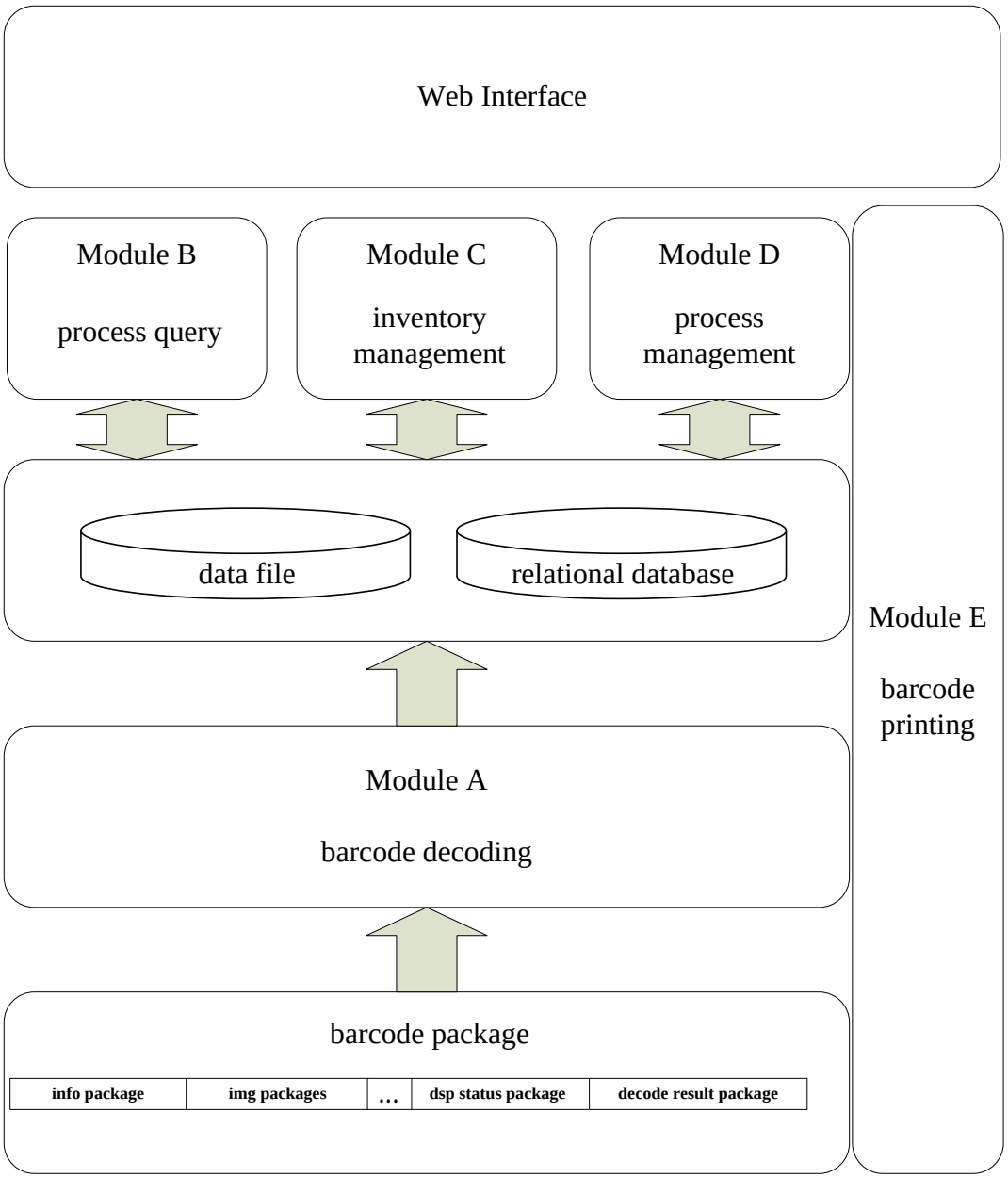


圖 2-2：系統架構圖

2.2 各模組介紹

首先，在模組 A 中，我們建立一個 Server 的服務，去接收條碼讀碼器(reader)所傳回來的封包資料(package)。如下圖 2-3 所示，條碼讀碼器每讀一次條碼，在正常運作的情況下，會傳回以下封包資料：一個 info package、291 個 img packages、一個 dsp status package 以及一個 decode result package，而每個封包的長度皆為 1412bytes。從這些封包中擷取出我們需要的原始資料分別如下：(1)條碼讀碼器

名稱、(2)條碼讀碼器 IP address、(3)條碼讀碼器 mac address、(4)條碼讀碼器解析文字資料、(5)條碼圖檔檔名、(6)NAS 接收到封包的系統時間。其中，前三項皆從 dsp status package 中取得，第四項文字資料從 decode result package 中取得，影像圖檔從 img package 中取得，而最後是封包進 NAS 資料庫的時間，這些資訊將會存到表格 raw_data 中，而表格定義會在下個章節再詳細介紹。

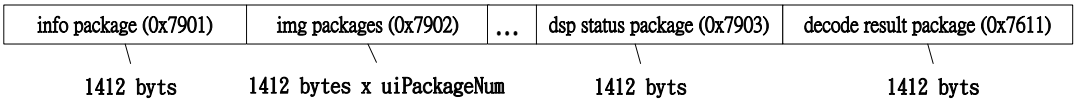


圖 2-3：條碼封包示意圖

在 Server 規劃部分，資料傳輸是透過 TCP Protocol 的方式進行。首先，先開啟 Server 讓系統去 Listen 指定的 Port，當條碼讀碼器連接到 Server IP 和觸發 Port 時即建立連線。Server 在接收到封包資料之後會根據事先所定義的封包標頭檔(usheader)來進行區分是那一類封包，一直接收到 decode result package (0x7611)時就完成傳送一個完整的封包資料。由於每個 package 都有屬於自己的結構，我們會從結構中擷取上述五項我們所要之資訊存在於資料庫表格 raw_data。

至於條碼解析部分，由於目前市面上的條碼有多種型式，其中，針對上述第四項的文字資料，有可能解出來的資料只能對應到「產品序號」，但是另一種條碼資料可能會對應到「產品型號」和「產品序號」，而中間以特殊的分隔符號隔開。舉例來說，圖 2-4 為市售變壓器，其型號(P/N)和序號(S/N)是合併為一張標籤，透過條碼讀碼器讀取之後內容為：「9NA0361608 H3161100696」，中間以空格隔開，目前本系統只針對這種形式來做處理，不過預先設計一個設定檔以便未來擴充功能。該設定檔會在之後進一步解釋。另外，在此模組中我們還會針對以下兩種異常狀況來做處理：(1)條碼器異常但仍送封包；(2)封包經過的站台順序不符合工單定義。完整的程式請參照附錄 A。

圖 2-4：所處裡的條碼格式

接下來，在模組 B 中我們是根據模組 A 所建立的表格 raw_data，以提供生產過程查詢。為了方便後續的說明我們假設工廠裡的產品型號是“FXC”，而這個型號有“S-00 ~ S-04”這五個序號。另外，工單 1 包含物品“S-00 ~ S-02”且其正確的流程為依序經過站台 A(stationA)、站台 B(stationB)、站台 C(stationC)；工單 2 包含物品“S-03 ~ S-04”且其正確流程為經過站台 A(stationA)、站台 B(stationB)、站台 E(stationE)。其中每個條碼讀碼器的名稱(reader_name)用 R 和 input/output 所組成，而條碼讀碼器的 mac adress 用 M 和 input/output 所表示。此模組提供的查詢功能如下：

- [功能一]給定一個條碼(型號或序號)，查詢條碼何時經過某些站台並依照時間先後順序排序：

此功能的操作範例如表 2-2 功能一。假設輸入參數為型號“FXC”，則會輸出對應序號的資訊，如(S-00、stationA、input、2)。若輸入參數為序號“S-00”，則輸出結果(R-AI，stationA，input，2)，代表 S-00 物品於 2 的時間經過 stationA 的入站條碼讀碼器 R-AI。

- [功能二]給定一個序號，查詢條碼經過前、後站台(所有組合和相鄰站台)所需要花的時間：

此功能的操作範例如表 2-2 功能二。假設輸入參數為“S-00”，則輸出結果為該物品經過站台的所有組合的時間如(stationA,stationB,3)、(stationA,stationC,7)、

(stationB,stationC,4)或者是相鄰兩個站台的經過時間差[1]，其輸出結果為(stationA,stationB,3)和(stationB,stationC,4)。

- [功能三]給定一個型號，針對其所對應到的所有序號，查詢以下兩種狀況：

(i)第一筆資料通過前一個站台的時間與後一個站台的時間差。此功能的操作範例如表 2-2 功能三的第一列，假設輸入參數為型號“FXC”，則我們會輸出(stationA,stationB,3)、(stationB, stationC,4)。也就是在 FXC 這個型號當中，S-00 為第一個進站的物品，因此以 S-00 物品來計算進、出站台的時間差。

(ii)使用者給定一個時間區間，在這時間範圍內，所有序號通過前、後站台的時間相減，再取平均值。若是站台前、後的時間數量不符合，以後面站台的時間為主。參照表 2-2 的第三列，假設輸入參數為型號“FXC, 2, 6”，則我們會輸出(stationA, stationB, 3)，也就是站台 A 到站台 B 所有序號的經過站台時間平均，算式為在時間區間內：$\sum$ *（B的進站時間－A的進站時間）/物品總數*

- [功能四]給定一個時間範圍，選擇某些條碼讀碼器並且個別列出條碼讀碼器的讀取數量與文字清單，或者是加總所有條碼讀碼器的讀取數量以及彙整所有文字清單：

此功能的操作範例如表 2-2 的第四列，假設輸入參數為“M-AI, M-BI, M-BO, 2, 5”，則輸出形式分為兩種，其一輸出結果為(M-AI,5),(FXC,S-00~S-04)、(M-BI，5),(FXC，S-00,S-01,S-03,S-04)。由於 M-BO 在時間範圍內沒有物品經過，所以沒有對應的資料輸出。其二輸出結果為(M-AI,M-BI,10),(FXC,S-00~S-04)。

[1]以產業的實際應用，大多採用兩個站台的進站時間加減。

此模組主要是要讓使用者下限制條件，我們的系統會組成適合的 SQL 語法對資料表格進行搜尋，在下個章節中我們會詳細介紹。但是由於原始條碼資料眾多進而查詢效率可能會不佳。我們會在第四章針對這部分進行改良，以提升效率。

表 2-2：各個功能範例說明

[功能一]	輸入參數：{ FXC } 輸出內容：(S-00,stationA,input,2) (S-02,stationA,intput,2) (S-00,stationA,output,3) (S-02,stationA,output,3) (S-00,stationB,input,5) (S-02,stationB,input,7) (S-00,stationB,output,6) (S-02,stationB,output,8) (S-00,stationC,intput,9) (S-02,stationC,intput,10) (S-00,stationC,output,11) (S-02,stationC,output,13) (S-01,stationA,intput,2) (S-03,stationA,intput,2) (S-01,stationA,output,3) (S-03,stationA,output,3) (S-01,stationB,input,5) (S-03,stationB,input,5) (S-01,stationB,output,6) (S-03,stationB,output,6) (S-01,stationC,intput,9) (S-03,stationE,intput,11) (S-01,stationC,output,11) (S-03,stationE,output,14)⋯
	輸入參數：{ S-00 } 輸出內容：(R-AI,stationA,input,2) (R-AO,stationA,output,3) (R-BI,stationB,input,5) (R-BO,stationB,output,6) (R-CI,stationC,output,9) (R-CO,stationC,output,11)
[功能二]	輸入參數：{ S-00 } 輸出內容：(stationA,stationB,3) (stationA,stationC,7) (stationB,stationC,4)

	輸入參數：{ S-00 } 輸出內容：(stationA,stationB,3) (stationB,stationC,4)
[功能三]	輸入參數：{ FXC } 輸出內容：(stationA, stationB, 3) (stationB, stationC, 4)
	輸入參數：{ FXC,2,5 } 輸出內容：(stationA, stationB,3)
[功能四]	輸入參數：{ M-AI, M-BI, M-BO, 2, 5 } 輸出內容：(M-AI,5),(FXC,S-00 ~S-04) (M-BI,5),(FXC,S-00 ~S-04) 輸出內容：(M-AI,M-BI,15),(FXC,S-00 ~S-04)

至於模組 C 主要是建立貨品數量的監測，我們可由使用者選擇某些站台後顯示該站台的條碼讀碼器讀取到的最新條碼所對應其型號，接著進行數量的顯示或計算，我們提供兩種不種不同的形態來計算，一為由系統對同一種型號的條碼自動加總。假設使用者選擇站台 A，當條碼讀碼器第一次讀取到型號為 FXC 的物品時，若接下來也是同一型號的物品，則系統會自動加總，然後記錄起來並顯示於螢幕上。二為針對類似倉庫的站台，我們會利用該站台進站條碼讀碼器所讀取到所有該型號的數量，減去該站台出站條碼讀碼器所讀到該型號的數量。

模組 D 主要是記錄物品經過某些站台狀態。首先使用者先選取某個站台，系統就會顯示該站台目前讀取到的最新條碼序號(若是型號有對應到多個序號就一起顯示)，我們提供更新及查詢的功能，使用者更改條碼的狀態分別為「過站/報廢/反報廢/重工/維修」，或者是把條碼原來的批號改成另一個批號。此外我們可以針對批號進行查詢或者是使用者給定一個條碼序號，查詢此條碼所有經過的站台、以及此條碼的批號資料。

最後在模組 E 中，提供在網頁上產生一維條碼(1D barcode)以及二維條碼(2D barcode)，目前市面上有很多種條碼形式，本系統只提供四種比較常在市面上看到的條碼，分別為 Code128、EAN13、Datamatrix、QR codec，前兩種是一維條碼而後面二種為二維條碼。產生出來的條碼可以讓使用者在網頁上自行定義條碼的大小、位置，排版完之後再列印出來。

第 3 章資料庫設計

此章節中，我們介紹系統內部的資料庫表格設計和對應的模組，接續在敘述模組 B 的查詢句。

3.1 資料庫表格定義

根據各個模組所需的功能，我們規劃系統所需要建立的表格，基本上我們將資料分為前置資料(offline data)、線上資料(online data)、輔助資料(other data)三大類。

前置資料為系統運作前，必須先行手動建立的資料。在表 3-1 中，我們列舉此類表格以及其所定義的屬性，其中以底線標示的為主鍵。首先，表格 reader (條碼讀碼器資訊)主要是記錄條碼讀碼器的 mac address(reader_mac)、站台名稱(station)、生產線(number)、功能(function)以及狀態(status)。其中，function 分為「input/output」，也就是一個站台可能配備有入站和出站兩台條碼讀碼器。status 分為「normal/abnormal」，用來判斷條碼讀碼器是否有異常，若是有異常的條碼讀碼器有傳封包資料，我們將不會把資料加入表格 raw_data。其次，表格 work_order (工單資訊)主要記錄工單號碼(wip_no)、批號(batch_no)、型號(product_no)、對應序號的起(serial_no_start)迄(serial_no_end)以及這批物品所走的正確流程(flow)，我們將利用型號和序號的起迄來找出某個物品是屬於那張工單或那一個批號。表格 separator (分隔設定)主要是記錄分隔種類以及描述，auto_id 作為 primary key，如上個章節模組 A 所討論，目前系統內只有一筆資料，其 type 為「space」，表示型號和序號在同一張標籤並且以空格隔開，若日後要處理其他類型的條碼，可再進行擴充。最後表格 station 主要是給模組 C 存貨管理之用，主要是記錄站台名稱(station)、功能(function)以及型態(type)。其中，function 分為「realtime」和「storagecount」這兩種，realtime 表示此站台計算數量方式是由

系統自動累加同個型號的物品而得到數量，storagecount 則是將物品的進站數量減去出站數量；而 type 欄位主要是將站台分為兩類情況，「IO」表示該站台有進站和出站兩台條碼讀碼器，「I」或「O」表示該站台只有進站條碼讀碼器或是出站條碼讀碼器。

表 3-1：前置資料

reader(條碼讀碼器資訊)		
reader_mac	varchar(100)	記錄條碼讀碼器 mac address
station	varchar(20)	記錄條碼讀碼器是屬於哪個站台
number	int	記錄位於站台的生產線
function	varchar(20)	記錄條碼讀碼器是屬於進站或出站
status	varchar(20)	記錄條碼讀碼器是 normal 或 abnormal
work_order(工單資訊)		
wip_no	varchar(100)	記錄工單號碼
batch_no	varchar(100)	記錄批號
product_no	varchar(100)	記錄物品型號
serial_no_start	varchar(100)	記錄此批物品序號第一筆
serial_no_end	varchar(100)	記錄此批物品序號最後一筆
flow	varchar(100)	記錄此物品所走的正確流程
separator(分隔設定)		
auto_id	int	primary key
type	varchar(20)	記錄分隔種類
description	varchar(100)	描述型號、序號是用哪種方式作區隔
station(站台資訊)		
stationName	varchar(100)	記錄站台名稱
function	varchar(100)	記錄 realtime 或 storagecount
type	varchar(100)	記錄該站台是那類，分 IO、I 或 O

線上資料是透過條碼讀碼器一直傳送我們所要的封包資料。為了提供生產流程查詢以及管理功能，我們把部分的資料分別記錄在不同的表格，而表格的詳細定義則列在表 3-2 中，以下分別說明各個表格。

表格raw_data主要紀錄原始條碼封包資料中，我們所要的有用資料，其內容包含：條碼讀碼器名稱(reader_name)、條碼讀碼器IP(reader_ip)、條碼讀碼器mac address(reader_mac)、型號(product_no)、序號(serial_no)、條碼圖檔名稱(image)、收到條碼的時間(time)以及目前走到的站台(current_flow)，而auto_id作為表格的主鍵。我們會依據此表格和reader表格的資料，找出物品在什麼時間經過哪一些站台，以提供模組B所需之功能。其次，表格productstation中主要供模組D製品流程管理用，我們同時也將接收到的條碼資料，抓取對應的批號等資訊記錄到表格中，主要記錄的內容為型號(product_no)、序號(serial_no)、條碼讀碼器mac address(reader_mac)、批號(batch_no)、狀態(status)、儲位(storage_no)、時間(time)以及使用者(user)，而auto_id作為主鍵。一開始預設物品在站台的狀態為過站，系統有以下五種狀態可以選擇：過站、報廢、反報廢、重工、維修。而表格reader_amount，主要是給模組C存貨管理用，我們透過trigger程式來監測每個讀碼器所讀取條碼的數量以及時間，記錄的內容為型號(product_no)、條碼讀碼器mac address(reader_mac)、讀取到第一筆資料的時間(start_time)、最新一筆資料的時間(end_time)、讀碼器讀取數量(amount)，而型號以及條碼讀碼器mac address做為此表格的主鍵，完整的trigger程式請參照附錄B。最後，表格stock主要是記錄站台讀取條碼的數量，包含型號(product_no)、站台名稱(station)、開始計算時間(start_time)、最後更新時間(end_time)以及計算的數量(amount)為何，其型號和站台的欄位做為此表格的主鍵。

表3-2：線上資料

raw_data(條碼原始資訊)		
auto_id	int	primary key
reader_name	varchar(100)	記錄條碼讀碼器的名稱
reader_ip	varchar(100)	記錄條碼讀碼器的 IP
reader_mac	varchar(100)	記錄條碼讀碼器的 mac address
product_no	varchar(100)	記錄物品的型號
serial_no	varchar(100)	記錄物品的序號
image	varchar(20)	記錄條碼讀碼器所拍攝的照片檔名
time	datetime	收到條碼的時間
current_flow	varchar(100)	記錄物品目前走到那個站台
productstation(產品過站資訊)		
auto_id	int	primary key
product_no	varchar(100)	記錄物品的型號
serial_no	varchar(100)	記錄物品的序號
reader_mac	varchar(100)	記錄條碼讀碼器的 mac address
batch_no	varchar(100)	記錄此條碼是屬於哪個批號
status	varchar(20)	記錄條碼在站台的狀態
storage_no	varchar(100)	記錄此物品是放到哪個倉庫
time	varchar(20)	記錄使用者修改表格時間
user	datetime	哪個使用者修改此表格
reader_amount(讀取數量)		
product_no	varchar(100)	記錄物品的型號
reader_mac	varchar(100)	記錄條碼讀碼器的 mac address
start_time	datetime	第一筆資料的時間
end_time	datetime	最新一筆資料的時間
amount	int	記錄數量

stock(存貨管理)		
product_no	varchar(100)	記錄物品的型號
station	varchar(20)	記錄站台名稱
start_time	datetime	記錄數量的第一個時間點
end_time	datetime	記錄數量的最後更新時間點
amount	int	記錄此型號的數量

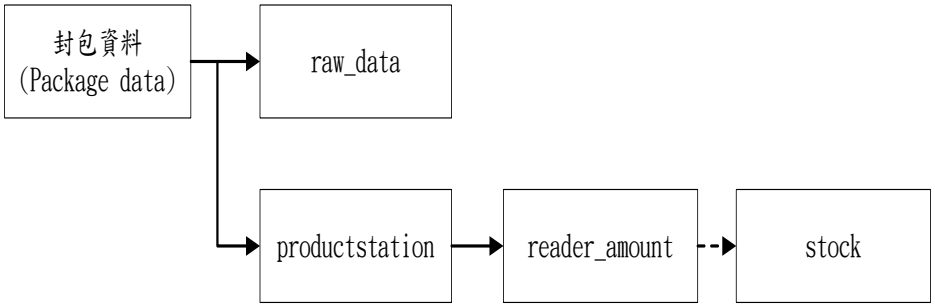


圖 3-1：資料流向(實線表示即時傳送，虛線表示由使用者啟動)

整個資料傳送方式如圖 3-1 所示，條碼讀碼器讀取到物品條碼並且解析條碼內容之後經由 TCP 網路的方式傳送封包資料(Package data)，當模組 A 的 Server 接收到封包資料之後，會即時將封包解析，擷取出我們所需要的欄位值並儲存至表格 raw_data 和 productstation，當表格 productstation 有資料被新增時，資料庫的 trigger 程式就會自動去讀取表格 reader_amount 中 product_no 和 reader_mac 欄位，判斷剛才被新增的資料有沒有出現在此表格，若是 product_no 和 reader_mac 欄位已經被記錄過則更新 end_time 和累加 amount 欄位；若是沒有被記錄則新增一筆資料並將 amount 欄位設定為 1。最後模組 C 的兩種計算數量方式都是透過表格 reader_amount 的欄位資訊，將其計算結果儲存至表格 stock。

最後一類資料，主要是提供一些輔助的功能，如表3-3所示。其中，表格user，主要記錄其內容為帳號(account)、密碼(password)、權限(level)、帳號名稱(name)、

帳號是否停用(valid)，其中level欄位將使用者分為兩類「使用者/管理者」，使用者只能執行系統的查詢功能，而管理者可以對存貨管理、使用者的帳戶修改以及前置資料的建置來做新增或修改，而valid欄位是記錄此帳號是否有被停用。在權限管理部份，我們特別設置一個最高權限(super root)的帳號，可管理所有管理者(admin)以及使用者(member)，而管理者與管理者之間無法修改彼此權限，但是管理者可以去修改所有的使用者。其次是表格log，主要記錄其內容為使用者帳號(user)、時間(time)和操作事項(description)，而user欄位主要是記錄使用者的帳號，其在本系統的任何操作如：登入、登出、修改或新增資料等任何操作，都將會被記錄在description欄位，而使用者帳號和time做為此表格的主鍵。最後，表格new_work_order主要是支援模組D修改批號之功能，將物品由原來的批號修改至新的批號存放至此表格，其內容為工單號碼(wip_no)、批號(batch_no)、型號(product_no)、相同型號的序號(all_serial_no)和時間(time)，而all_serial_no欄位是將使用者選到的序號，依照JSON (Javascript Object Notation)格式將資料存放至此欄位，其工單號碼和批號做為此表格的主鍵。

表 3-3：輔助資料

user(權限管理)		
account	varchar(20)	記錄帳號
password	varchar(20)	記錄密碼
level	varchar(20)	權限分為「管理者」和「使用者」
name	varchar(20)	記錄名稱
valid	boolean	記錄此帳號是否停用
log(錯誤資訊)		
user	varchar(20)	記錄使用者帳號
time	datetime	記錄時間
description	text	記錄使用者的任何操作事項

new_work_order(新的工單資訊)		
wip_no	varchar(100)	記錄此物品原來的工單號碼
batch_no	varchar(100)	記錄新的批號
product_no	varchar(100)	記錄物品型號
all_serial_no	text	紀錄物品的所有序號
time	datetime	更改批號的時間

3.2 模組 B 之查詢句設計

根據上節所敘述之表格定義，我們接著描述如何將模組 B 的功能，以適當的 SQL 查詢句實作出來。注意到，在以下的查詢句中，我們會將使用者輸入的參數以中括號括起來。

[功能一]查詢條碼何時經過那些站台：使用者給定一個條碼(型號或序號)，找出該條碼在什麼時間，經過那些站台並且依照時間先後順序排序。如表 3-4 所示，假設輸入參數為“serial_no”則其對應的 SQL 指令如 Q1-1 所示。系統會去表格 raw_data 中抓出此“serial_no”在何時(time)經過哪些 reader_mac，再透過 reader_mac 欄位去表格 reader 中得到對應的站台資料(station 和 function)，並將所有資料以時間遞增排序。而 Q1-2 和 Q1-1 的差別，只在於將輸入參數改為“product_no”。此功能的範例輸出請參見表 2-2 的第一列。

表 3-4：功能一之查詢設計

Q1-1	L01: **SELECT** reader_name, station, function, time L02: **FROM** raw_data, reader L03: **WHERE** raw_data.serial_no = [serial_no] L04: **AND** raw_data.reader_mac = reader.reader_mac L05: **ORDER BY** time **ASC**

Q1-2	L01: **SELECT** serial_no, station, function, time L02: **FROM** raw_data, reader L03: **WHERE** raw_data.product_no = [product_no] L04: **AND** raw_data.reader_mac = reader.reader_mac L05: **ORDER BY** time **ASC**

[功能二]給定一個條碼序號查詢該物品經過前、後站台所需要花的時間：如表 3-5 所示，假設輸入參數為“serial_no”則其對應的 SQL 指令如 Q2-1，在 L02-L07 定義了一個延伸表格(derived relation)別名為 a,其查詢句和 Q1-1 大致相同，只是在 L06 多了一個限制條件，只取出該站台的入站條碼讀碼器所讀取的相關資訊。而 L08-L11 定義的延伸表格別名為 b，則也是同樣取出對應的站台以及時間資訊。由於此兩個延伸表格都列在同一個 FROM 子句中，會產生所有可能的配對組合，所以我們在 L13 加入兩個限制條件，第一個是所對應的兩個站台不能是同一個，第二是 timediff()必須>0，也就是 b 站台在 a 站台之後。L01 顯示輸出結果為 a.station、b.station、時間差(timediff)。其中 SQL 指令所使用到的 timediff(datetime time2, datetime time1)函數，主要是用以回傳 time2 和 time1 的時間間隔。而 Q2-2 此作法搭配 PHP 和 SQL 語法，首先 L01 宣告一個名稱 match_station 的陣列，主要是存放物品經過的站台序列，由於我們不知道此物品最終會走到哪個站台,所以我們在 L02-L07 先做一次查詢知道此序號會經過那些站台，其限制條件為進站(input)並且以進站的時間點(time)作為排序存放在 $match_station 陣列中，其中 L02 的 Select into 非 SQL 用法而是將符合的值放入陣列，接著在 L10-L16 定義一個延伸表格 a，其限制條件和 Q2-1 差不多，差別在於站台限制在前站($match_station[$i])，符合的資料於 L16 以時間遞增作為排序。而 L17-L23 定義延伸表格 b，而延伸表格 b 的限制條件是將站台限制在後站($match_station[$i+1]),其輸出結果為 a.station、b.station、時間差我們透過 L08 for

迴圈將 match_station 所有站台執行過一遍，如此一來就能將所有可能的組合限制在前、後兩個站台而非所有站台。此功能的範例輸出請參見表 2-2 的第二列。

表 3-5：功能二之查詢設計

Q2-1	L01: **SELECT** a.station, b.station, timediff(b.time,a.time) L02: **FROM** (**SELECT** station, time L03: **FROM** raw_data, reader L04: **WHERE** raw_data.serial_no = [serial_no] L05: **AND** raw_data.reader_mac = reader.reader_mac L06: **AND** reader.function = 'input' L07: **ORDER BY** time **ASC**) **AS** a, L08: (**SELECT** station ,time **FROM** raw_data,reader L09: **WHERE** raw_data.serial_no = [serial_no] L10: **AND** raw_data.reader_mac = reader.reader_mac L11: **AND** reader.function = 'input' L12: **ORDER BY** time **ASC**) **AS** b L13: **WHERE** a.station != b.station **AND** timediff(b.time, a.time) > 0 L14: **ORDER BY** a.station,b.station
Q2-2	L01: $match_station = array() L02: **SELECT** distinct station **INTO** $match_station //符合的值放入陣列 L03: **FROM** reader, raw_data L04: **WHERE** reader.function = 'input' L05: **AND**raw_data.reader_mac = reader.reader_mac L06: **AND** raw_data.serial_no = [serial_no] L07: **ORDER BY** time **ASC** L08: For ($i=0;$i<count($match_station);$i++){ L09: **SELECT** a.station, b.station, timediff(b.time,a.time) L10: **FROM** (**SELECT** station,time L11: **FROM** raw_data,reader L12: **WHERE** raw_data. serial_no = [serial_no] L13: **AND** raw_data.reader_mac = reader.reader_mac

	L14:	**AND** reader.station = [$match_station [$i]]
	L15:	**AND** reader.function = ‘input’
	L16:	**ORDER BY** time **ASC**) **AS a**,
	L17:	(**SELECT** station,time
	L18:	**FROM** raw_data,reader
	L19:	**WHERE** raw_data. serial_no = [serial_no]
	L20:	**AND** raw_data.reader_mac = reader.reader_mac
	L21:	**AND** reader.station = [$match_station [$i+1]]
	L22:	**AND** reader.function = ‘input’
	L23:	**ORDER BY** time **ASC**) **AS B**}

[功能三]給定一個條碼型號，查詢對應物品經過前、後相鄰站台所需要花的時間：由於一種型號可能會對應到多個序號，因此我們再細分為兩種狀況。

第一種狀況輸出通過前一個站台的第一個物品的時間與後一個站台的第一個物品的時間之差。如表 3-6 所示，假設輸入參數為“product_no”，則其對應的 SQL 指令如 Q3-1，此作法與 Q2-2 相同，差別在於 L10-L16 延伸表格別名為 a 的限制條件由序號改為型號，符合的資料於 L16 以時間遞增作為排序並且限定只能抓取一筆資料，如此可以確定只拿到第一筆資料，L17-L22 延伸表格別名為 b 其限制條件與上述相同，其輸出結果為前站(a.station)、後站(b.station)、時間差。

第二種狀況則是給定一個時間區間(start_time,end_time)，將這時間範圍內，該型號所有對應的序號，通過前、後相鄰站台的時間相減，再取平均值。假設輸入參數為“product_no”、“start_time”、“end_time”，則其對應的 SQL 指令如 Q3-2，在 L01-L07 和 Q3-1 用法一樣將型號所經過的站台用$match_station 陣列存放，並在 L10-L16 定義一個延伸表格別名為 a，但不同之處再於 L16 多了一個“between … and”的限制條件，使得在“start_time”和“end_time”中的序號資訊和時間資訊才會取出，而 L17-L24 為另一個延伸表格 b 定義和延伸表格 a 一致，差別在於限制條件站台為後站(match_station[$i+1])。最後在 L25 限制條件

下限定必須是相同的序號，才能將其時間差計算出來，並求得平均時間。其中 SQL 指令所使用到的 SEC_TO_TIME(second)和 TIME_TO_SEC(time)函數，主要是 AVG 函數的需求必須是數值，所以在計算時間之前，先把前、後兩站台的時間間隔轉換成秒數型態，取完平均值之後再將秒數型態轉換成時間型態。此功能的範例輸出請參見表 2-2 的第三列。

表 3-6：功能三之查詢設計

Q3-1	L01: $match_station = array() L02: **SELECT** distinct station **INTO** $match_station //將符合的值放入陣列 L03: **FROM** reader, raw_data L04: **WHERE** reader.function = 'input' L05: **AND** raw_data.reader_mac = reader.reader_mac L06: **AND** raw_data.product_no = [product_no] L07: **ORDER BY** time **ASC** L08: For ($i=0;$i<count($match_station);$i++){ L09: **SELECT** a.station, b.station, timediff(b.time,a.time) L10: **FROM** (**SELECT** station, time L11: **FROM** raw_data, reader L12: **WHERE** raw_data.product_no = [product_no] L13: **AND** raw_data.reader_mac = reader.reader_mac L14: **AND** reader.station = [$match_station[$i]] L15: **AND** reader.function = 'input' L16: **ORDER BY** time **ASC LIMIT** 1) **AS a**, L17: (**SELECT** station, time L18: **FROM** raw_data,reader L18: **WHERE** raw_data.product_no = [product_no] L19: **AND** raw_data.reader_mac = reader.reader_mac L20: **AND** reader.station = [$match_station[$i+1]] L21: **AND** reader.function = 'input' L22: **ORDER BY** time **ASC LIMIT** 1) **AS B**}

Q3-2	L01: $match_station = array() L02: **SELECT** distinct station **INTO** $match_station //將符合的值放入陣列 L03: **FROM** reader, raw_data L04: **WHERE** reader.function = 'input' L05: **AND** raw_data.reader_mac = reader.reader_mac L06: **AND** raw_data.product_no = [product_no] L07: **ORDER BY** time L08: For ($i=0;$i<count($match_station);$i++){ L09: **SELECT** SEC_TO_TIME(AVG(TIME_TO_SEC(timediff(b.time,a.time)))) L10: **FROM** (**SELECT** raw_data.time,raw_data.serial_no L11: **FROM** raw_data,reader L12: **WHERE** raw_data.reader_mac = reader.reader_mac L13: **AND** reader.function = 'input' L14: **AND** raw_data.product_no = [product_no] L15: **AND** reader.station = match_station[$i] **AND** raw_data.time L16: **BETWEEN** [start_time] **AND** [end_time]**) AS A,** L17: (**SELECT** raw_data.time, raw_data.serial_no L18: **FROM** raw_data, reader L19: **WHERE** raw_data.reader_mac = reader.reader_mac L20: **AND** reader.function = 'input' L21: **AND** raw_data.product_no = [product_no] L22: **AND** reader.station = match_station[$i+1] L23: **AND** raw_data.time L24: **BETWEEN** [start_time] **AND** [end_time]) **AS B,** L25: **WHERE** a.serial_no = b.serial_no **AND** timediff(b.time,a.time)>0 }

[功能四]查詢某段時間內，某些條碼讀碼器讀取條碼的總數量和解碼的文字清單：如表 3-7 所示，假設輸入參數為“chosen_reader”、“start_time”、“end_time”則其對應的 SQL 指令如 Q4-1，L01 宣告一行名稱為 chosen_reader

陣列，主要是存放使用者選擇的條碼讀碼器 mac address，而在數量的計算和文字清單我們將其分為兩段查詢句來實作，在 L03-L06 計算讀取數量部分，我們在限制條件中直接對表格 raw_data 去查詢在 start_time 和 end_time 時間範圍內，count 條碼讀碼器所讀取的所有序號並透過 L02 for 迴圈將使用者選到的站台($chosen_reader)逐一執行並顯示其結果，其輸出結果為個別條碼讀碼器數量(count)和文字清單(product_no, serial_no)。第二種狀況，是將所有條碼讀碼器讀取的條碼數量加總($sum)以及彙整所有文字清單(product_no,serial_no)，跟 Q4-1 輸入的參數一模一樣，差別是輸出數量的方式，因此對應的 SQL 指令如 Q4-2，對表格 raw_data 去查詢符合在時間範圍內 start_time 和 end_time 的所有條碼讀碼器，並使用 count 語法將所有條碼讀碼器所讀取到的序號用一個變數$sum 累加，最後顯示結果。在 L08-L12 文字清單在時間範圍內“start_time”、“end_time”藉由 L10 IN 的語法在表格 raw_data 中找出符合所有使用者選擇到的條碼讀碼器 mac address($chosen_reader[0],$chosen_reader[1]…,$chosen_reader[n-1])，查詢的結果用 SQL 語法 distinct 讓序號不重複。此功能的範例輸出請參見表 2-2 的第四列。

表 3-7：功能四之查詢設計

Q4-1	L01: $chosen_reader = array(); L02: For ($i=0;$i<count($chosen_reader);$i++){ L03: **SELECT** count(distinct serial_no) L04: **FROM** raw_data L05: **WHERE** raw_data.reader_mac = [$chosen_reader[$i]] L06: **AND** time **BETWEEN** [start_time] **AND** [end_time]; L07: **SELECT** distinct product,serial_no L08: **FROM** raw_data L09: **WHERE** raw_data.reader_mac = [$chosen_reader [$i]] L10: **AND** time **BETWEEN** [start_time] **AND**[end_time])

Q4-2	L01: $sum = 0 L02: For ($i=0;$i<count($chosen_reader);$i++){ L03: **SELECT** count(distinct serial_no) L04: **FROM** raw_data L05: **WHERE** raw_data.reader_mac = [chosen_reader[$i]] L06: **AND** time **BETWEEN** [start_time] **AND** [end_time] L07: $sum **+=** count(serial_no);} L08: **SELECT** distinct product_no, serial_no L09: **FROM** raw_data L10: **WHERE** raw_data.reader_mac L11: **IN** (‘$chosen_reader [0]’, ‘$chosen_reader [1]’… ‘$chosen_reader [n-1]’) L12: **AND** time **BETWEEN** [start_time] **AND** [end_time]

第 4 章流程時間樹

我們在此章說明如何建立流程時間樹(Flow-Time Tree)，以改善生產流程查詢的處裡效率並進行流程控管，包括如何進行原始資料的轉換與轉換後表格的定義，以及對應的查詢句設計。

4.1 資料表示法

為了方便後續的討論，根據 3-1 節表格 raw_data 所定義的原始資料，在此我們只取部分的欄位來討論，形式為：(reader_name,product_no,serial_no,time)。舉例來說，如圖 4-1 為簡化過後的原始資料(raw_data)，其 S-00 這個物品，在 2 的時間點進入 A 站台，在 3 的時間點出站，在 5 的時間點進入 B 站台，在 6 的時間點出站…。我們需要反覆比對此表格如第三章查詢句 Q1-Q2 所示，才能找到 S-00 所經過站台的時間。

為了減少反覆比對同一個表格所產生出來的大量中間層資料，以加快查詢效率，我們提出轉換原始資料至追蹤資料(trace record)。trace records 結構為 product_no,serial_no：station[start_time,end_time]，product_no 為條碼的型號、serial_no 為條碼的序號、station 是站台名稱、start_time 和 end_time 分別代表進站的時間與出站的時間，透過 trace records 來取代我們的原始資料。舉例來說，我們先將“S-00”的 reader 資訊去表格 reader 找到所屬站台並且合併進站和出站的兩筆資料，轉化成(S-00,A,2,3)，(S-00,B,5,6)，(S-00,C,9,11)，如圖 4-2 所示，這樣我們就可以很清楚的知道 S-00 在什麼時間點經過與離開 A、B、C 站台。

(R-AI,FXC,S-00 ,2)
(R-AO,FXC,S-00 ,3)
(R-BI,FXC,S-00 ,5)
(R-BO,FXC,S-00 ,6)
(R-CI,FXC,S-00 ,9)
(R-CO,FXC,S-00 ,11)
(R-AI,FXC,S-01 ,2)
(R-AO,FXC,S-01 ,3)
(R-BI,FXC,S-01 ,5)
(R-BO,FXC,S-01 ,6)
(R-CI,FXC,S-01 ,9)
(R-CO,FXC,S-01 ,11)
.
.
.

圖 4-1：簡化過後的 raw_data 資料

FXC,S-00： A[2,3],B[5,6],C[9,11]
FXC,S-01： A[2,3],B[5,6],C[9,11]
FXC,S-02： A[2,3],B[7,8],C[10,13]
FXC,S-03： A[2,3],B[5,6],E[11,14]
FXC,S-04： A[2,3],B[5,6],E[11,14]
FDQ,S-05： A[2,3],D[7,8]
SAB,S-06： B[3,4],C[6,7],D[9,12]

圖 4-2：對應圖 4-1 raw_data 的 trace records

為了將所有物品經過站台的時間資訊整合表示，以方便對站台的順序與特定時間範圍做查詢，同時便於做錯誤流程的檢測與控管，因此我們建造一棵有時間順序的流程樹，簡稱流程時間樹(flow-time tree)$T=\{V, E\}$，其中流程時間樹的節點 V 為一筆 trace record 的站台名稱和進站與出站的時間，流程時間樹的邊 E 為一筆 trace record 所依序經過的站台並且是有方向性的邊，舉例來說，trace record “S-00” 會在流程時間樹中產生一條 A[2,3] → B[5,6]→ C[9,11]這樣的路徑，其中 A[2,3]、B[5,6]、C[9,11]代表流程時間樹上的三個節點，其邊從 A 站台走到 B

站台再走到 C 站台。但是在不同的時間經過同個站台路徑的話，流程時間樹就會產生出新的分支，以 trace record“S-02”為例，雖然 S-00 與 S-02 分別都經過站台 A、B、C，但由於進、出站台 B 的時間不同，使得流程時間樹會產生出新的分支，如圖 4-3 左邊那棵樹(T1)。

為了能在流程時間樹上有很好的搜尋效率，我們對每棵流程樹的節點經由 range encoding scheme[LC08]的(start,end)編號方式來表示節點在流程樹上的位置，而編碼方式是根據深度搜尋(Depth first search)來編碼，其編碼為連續正整數。以圖 4-3 為例子，A[2,3]、B[5,6]、C[9,11]這三個節點的編碼分別為(1,14)、(2,7)、(3,4)。range encoding 的特色為若 a 為 b 的祖先，則 a.start < b.start 且 a.end > b.end。從以上例子中若我們要查詢“S-00”條碼經過那些站台，我們由其最後站台 C[9,11]的編碼(3,4)往回指，找出 start<=3 並且 end>=4 的所有節點，則可得到 A[2,3]、B[5,6]、C[9,11]三個節點。

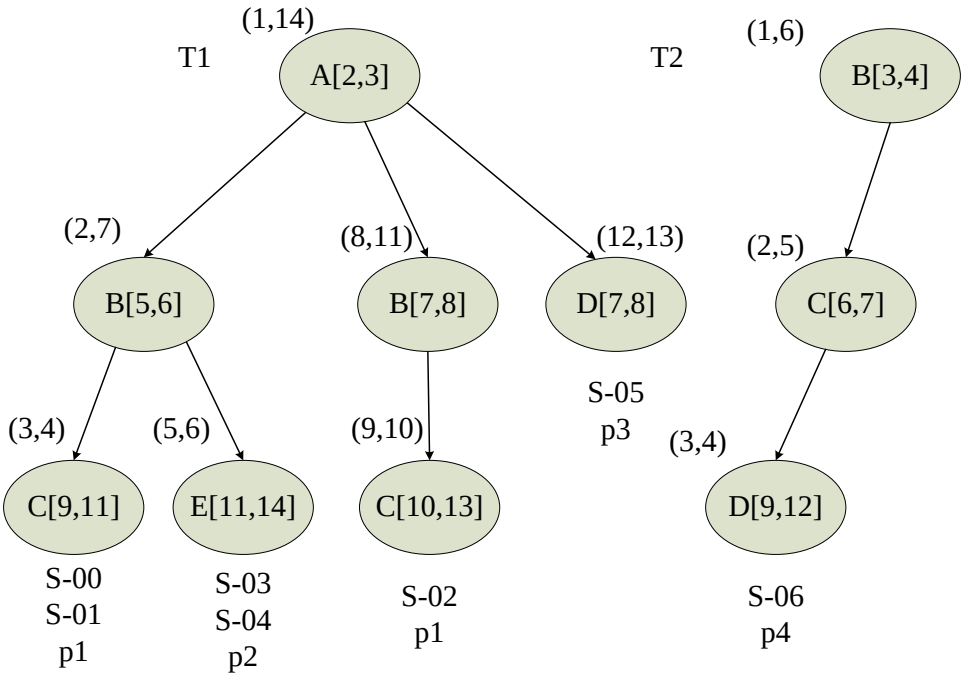


圖 4-3：流程時間樹

在本系統中，我們考慮到每樣物品不一定經過相同的生產流程，以圖 4-2 trace record“S-06”為例，經過的站台時間為 B[3,4]、C[6,7]、D[9,12]與“S-00”和“S-01”的起始站台和時間都不同，所以會另外再建造一棵流程時間樹如圖 4-3 右邊那棵樹(T2)。對生產流程來說，何時進站、何時出站的時間點是很重要的，我們的流程時間樹可將不同流程的物品歸納在同一群，如圖 4-2 trace record “S-05”對應到圖 4-3(T1)，而非自行建立一棵樹，因此可以減少流程時間樹的產生，也就是說一棵流程時間樹可能會包含不只一條生產流程。由於每個節點所經過的站台可由其祖先節點得到，所以此樹狀結構可以稱作是一個 trie。另外，我們是依據 trace records 的起始站台的時間來判別，若起始站台的進、出時間不同，則會另外建一棵樹，否則會合併到既有的流程時間樹當中。我們分別對每一群 trace records 建立流程時間樹，依 Depth First Search 的方式對每個節點編碼，最後會將所有資訊全部存到資料表格裡。

4.2 表格定義

根據流程時間樹將物品的進、出站資訊分類之後，我們會將所有資料存放在表格 flow、item、station_time 裡如表 4-1 所示。表格 flow 主要是紀錄在系統中有幾種不同的生產流程，以及每個流程是建立在那棵樹的對應關係，其內容包含流程 ID(flow_id)和經過那些站台(path)，此表格主鍵為流程 ID。其次是表格 item，對應到所有樹的所有葉節點的詳細物品資訊，主要記錄的內容為序號(serial_no)、型號(product_no)、流程 ID(flow_id)、樹 ID(tree_id)、start、end，而(start,end)編碼是根據此序號在流程時間樹中走到最後一個站台的位置，其主鍵為序號。而表格 station_time 表格對應到所有樹的所有節點資訊，主要記錄的內容為樹的 ID(tree_id)、start、end、站台名稱(station)、進站時間(start_time)、出站時間(end_time)。這裡的(start,end)表示站台在流程樹中的位置，其主鍵為樹 ID、start、end。最後表格 error_flow 主要是記錄物品走的錯誤流程，其內容為序號(serial_no)、型號

(product_no)、物品所屬批號(batch_no)、物品所屬工單號碼(wip_no)和走的流程(flow)，而 flow 欄位主要是記錄此物品在原始資料(raw_data)中，所走的錯誤流程，其型號和序號的欄位做為此表格的主鍵。舉例來說，假設型號“FXC”的“S-00”物品，本來的正確流程為 A → B→ C，若是“S-00”走的 A → D→ E 的話，則會將錯誤的流程資訊記錄在 flow 欄位，其主鍵為序號。

表 4-1：FTT 方法的線上資料表格

flow(路徑資訊)		
flow_id	varchar(100)	記錄流程 ID
path	varchar(100)	記錄物品經過的站台列表
item(物品資訊)		
serial_no	varchar(100)	記錄物品的序號
product_no	varchar(100)	記錄物品的型號
flow_id	varchar(100)	記錄此序號是那一個流程 ID
tree_id	int	紀錄此序號是屬於那一個樹 ID
start	int	記錄此物品所在之葉節點編號
end	int	記錄此物品所在之葉節點編號
station_time(站台時間資訊)		
tree_id	int	記錄樹的 ID
start	int	記錄樹節點編號
end	int	記錄樹節點編號
station	varchar(50)	記錄站台名稱
start_time	datetime	記錄站台的進站時間
end_time	datetime	記錄站台的出站時間

error_flow(錯誤流程資訊)		
serial_no	varchar(100)	記錄型號
product_no	varchar(100)	記錄序號
wip_no	varchar(100)	記錄此物品是屬於那張工單號碼
batch_no	varchar(100)	記錄該序號是屬於那個批號
flow	varchar(100)	記錄該序號所走的流程

表 4-2：資料列數比較

raw_data	station_time
(R-AI,FXC,S-00,2) (R-AO,FXC,S-00,3) (R-BI,FXC,S-00,5) (R-BO,FXC,S-00,6) (R-CI,FXC,S-00,9) (R-CO,FXC,S-00,11) (R-AI,FXC,S-01,2) (R-AO,FXC,S-01,3) (R-BI,FXC,S-01,5) (R-BO,FXC,S-01,6) (R-CI,FXC,S-01,9) (R-CO,FXC,S-01,11)	(t1,1,14,A,2,3) (t1,2,7,B,5,6) (t1,3,4,C,9,11)
	item
	(S-00,FXC,p1,t1,3,4) (S-01,FXC,p1,t1,3,4)

在此我們來比較表格 raw_data 和 station_time 的資料列數。如表 4-2 所示，我們以 S-00 和 S-01 為例，在 raw_data 表格當中，每個物品經過一個站台時就會有兩筆資料被回傳，而 S-00 和 S-01 經過了 A → B→ C 三個站台，在表格 raw_data 就會有 12 筆資料。然而在表格 station_time，其 S-00 和 S-01 經由 trace record 的轉換，在建立時間流程樹時被歸類在相同的葉節點，所以在 station_time

只會記錄該葉節點和其二祖先節點等三筆資料，若是有更多物品走相同路徑又被時間流程樹歸類在相同葉節點時這資料量的差距會更大。所以，station_time 對於 raw_data 來說資料量小很多。

接續之前的例子，我們將圖 4-2 的 trace records(或圖 4-3 的流程時間樹)轉換成 FTT 線上資料表格，範例資料如表 4-3 所示，假設我們要查詢物品“S-00”的經過站台列表，則我們會先去表格 item 中查詢物品 S-00 是建立在 tree_id 為 1 底下而且編碼位置為(3,4)，再根據樹 ID 到對應表格 station_time，得到這棵樹底下每個所屬站台位置(start,end)編碼，最後由 range encoding 的特性，3(物品 start)>= 站台的 start 且 4(物品 end)<=站台的.end，如此一來符合內容站台為 A(1,14)、B(2,7) 和 C(3,4)。

表 4-3：圖 4-2 範例資料表格

flow	
flow_id	path
p1	A,B,C
p2	A,B,E
p3	A,D
p4	B,C,D

item					
serial_no	product_no	flow_id	tree_id	start	end
S-00	FXC	p1	1	3	4
S-01	FXC	p1	1	3	4
S-02	FXC	p1	1	9	10
S-03	FXC	p2	1	5	6
S-04	FXC	p2	1	5	6
S-05	FDQ	p3	1	12	13
S-06	QBC	p4	2	3	4

station_time					
tree_id	start	end	station	start_time	end_time
1	9	10	C	10	13
1	5	6	E	11	14
1	3	4	C	9	11
1	12	13	D	7	8
1	8	11	B	7	8
1	2	7	B	5	6
1	1	14	A	2	3
2	3	4	D	9	12
2	2	5	C	6	7
2	1	6	B	3	4

4.3 資料流向與流程控管

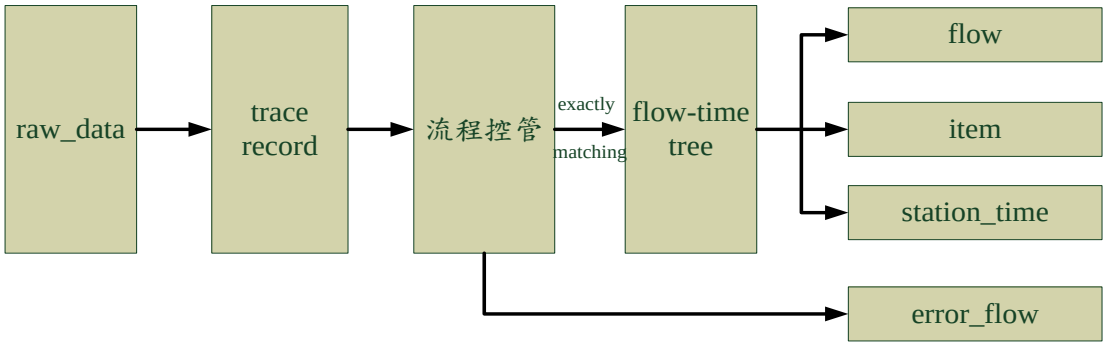


圖 4-4：FTT 資料流向圖

在本節中，我們說明此方法完整的資料流向與轉換。如圖 4-4 所示，當 Server 從各個站台的條碼讀碼器接收到 raw_data 資料，我們會將 raw_data 資料轉換成 trace record，然後透過流程控管(flow control)比對工單資訊以及 trace record 的每筆資料，其中具有正確流程的 trace record 會被建立在時間流程樹(flow time tree)中，並將物品資訊以及站台時間資訊儲存在表格 flow、item、station_time，而錯

誤流程的資訊則儲存在表格 error_flow 內。

在流程控管的部分，工單資訊是事前由使用者定義好每個物品所需要走的正確流程，如圖 4-5 所示，我們透過 exactly matching 的方式對工單資訊(work_order)的 flow 欄位和 trace record 所經過的站台列表整個做比對，舉例來說，假設我們有 w1-w8 這八張工單，而每張工單都依序記錄著批號、型號、序號以及每個物品所必須經過的正確流程。我們在建流程時間樹之前，將工單資訊的所有欄位全部讀取出來，接續透過 trace record 以時間排序的經過站台列表來做比較。舉例來說，trace record S-00 依時間排序所經過的站台為 A、B、C，此時因為跟工單號碼 w1 中的流程正好一樣，所以我們認為其流程都正確，可以進流程時間樹來建樹。若另一方面，如型號為 NNZ，S-30 的這個物品，其對應的工單號碼 w8 的正確流程為站台 B、C、D，但是 S-30 實際走的流程為 B、D，跟我們所定義的工單資訊不符，因此我們就把該物品的資訊以及走的錯誤流程記錄在表格 error_flow。

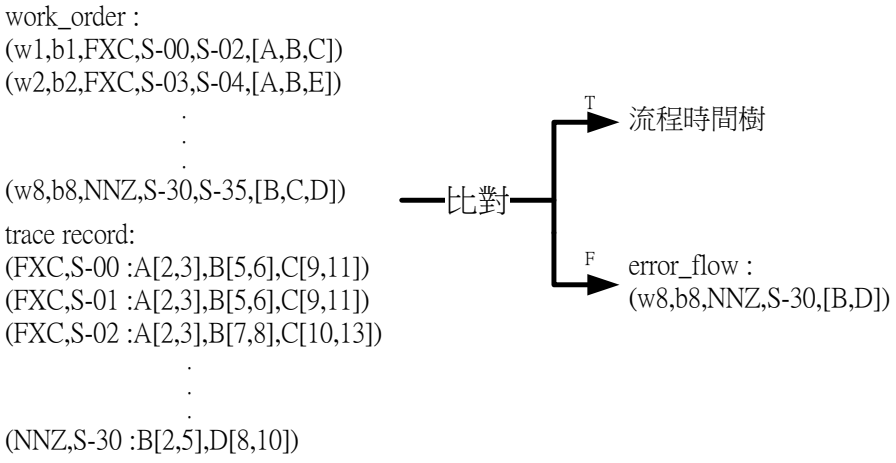


圖 4-5：流程控管比對圖

接下來我們說明如何進行如圖 4-4 的資料轉換，也就是將 trace records 和表格 work_order 當作輸入，而希望輸出表格 flow、item、station_time 和 error_flow。

對應的演算法稱作 FTT，其輸入參數 tr 是一個 trace record 的集合，每一個 trace record 紀錄該物品的型號(product_no)、序號(serial_no)、所經站台流程(flow)和完整站台時間流程(flow-time)，而“path”和“p_t”這兩個陣列是以 Key &Value 的資料型態來儲存資料，path 陣列主要記錄 trace record 的完整流程時間(flow-time)和流程 ID(flow_id)的對應關係、p_t 陣列主要是記錄流程 ID(flow_id)和樹 ID(tree_id)的對應關係，而變數“flow_id”和“tree_id”分別記錄目前的流程 ID 和樹 ID。

完整的 FTT 演算法如圖 4-6 所示，首先我們初始化 path 和 p_t 這兩個陣列先設定為 NULL，而變數 flow_id 和 tree_id 皆設為 1，接著，在 L03-L04 中，我們判斷 trace record 的站台列表(*k*.flow)有沒有和任何一個表格 work_order 的 flow 欄位值所記錄走的流程相同，若皆不同我們就把該物品依照下面欄位(型號、序號、工單號碼、批號、流程)新增至表格 error_flow。若是 trace record 走正確流程，在 L06-L07 我們將完整的流程時間(*k*.flow-time)與目前的流程 ID(flow_id)給記錄起來；流程 ID 與樹 ID 的對應關係也記錄在 p_t 陣列。接下來，在 L08 判斷此 trace record 第一個站台的進、出時間(*k*.flow-time)與之前的 trace record 是否相同並檢查此物品的完整流程有沒有被 path 陣列記錄過，符合限制表示該物品是屬於同一顆樹的不同路徑，則我們將變數 flow_id 加一。不相同者如 L11-L12，表示該物品是屬於不同群，需要另外再建一棵樹，則變數 flow_id 與 tree_id 加一。在 L14 將所有分類好的站台流程與路徑 ID 儲存至表格 flow。接著，L18-L19 我們針對相同樹 ID 的 trace record 建樹，並透過 DFS(Depth First Search)編碼方式將每個節點依序編號。其變數節點 *n* 結構包含 station_name、start、end、start_time 和 end_time，最後 L21-L23 走訪樹的每個節點時，將所有資訊全部存到表格物品資訊(item)和表格站台資訊(station_time)。

演算法名稱：FTT

輸入：tr：a set of trace record,表格 work_order

輸出：表格 flow、item、station_time、error_flow

變數：path：trace record 對應流程 ID 的陣列

key 為 tr.flow-time，value 為 flow_id

p_t：流程 ID 對應的樹 ID 的陣列，key 為 flow_id，value 為 tree_id

tree_id：記錄目前的樹 ID

flow_id：記錄目前的流程 ID

Begin

L01: **Initialize** path ← NULL; p_t ← NULL; tree_id ← 1; flow_id ← 1

L02: **foreach** *k* ε tr **do**

L03: **if** *k*.flow not in the "flow" attribute of the 'work_order' table

L04: store(*k*.product_no,*k*.serial_no,null,null,*k*.flow) into'error_flow' table

L05: **else**

L06: path[*k*.flow-time] ←flow_id

L07: p_t[flow_id] ← tree_id

//起始站台名稱與時間相同但是之後走不同的流程

L08: **if** the first station of *k*.flow-time exists in the key of the path array && *k*.flow-time not in the key of the path array **then**

L09: flow_id++

L10: **else** //起始站台名稱或時間不同

L11: tree_id++

L12: flow_id++

L13: **end foreach**

L14: store each entry of the path array (path.key.flow,path.value) into flow table

L15: **foreach** *j* <= tree_id **do**

L16: *T*←NULL

L17: **foreach** *k* ε tr **do**

L18: **if** p_t[path[*l*.flow]] == j

L19: *T*= CreateTree(*T*,*k*.flow)

L20: assign (start ,end)編號 to node in *T* by DFS

```
        //each node n has name, start_time, end_time,start, end
L21:    while traversing node n in T do
L22:        store(k.serial_no,k.product_no,path[k.flow],k,n.start,n.end)
            into 'item' table
L23:        store(j,n.start,n.end,n.name,n.start_time,n.end_time)
            into 'station_time' table
L24:    end foreach
L25: end foreach
END
```

圖 4-6：FTT 演算法

4.4 查詢句設計

延續上述所介紹，我們將 FTT 的表格拿來實作模組 B 功能一的改善查詢，如表 4-4，假設輸入參數為“serial_no”則其對應的 SQL 指令 Q'1-1 所示，在 L03-L06 的限制條件中，我們將給定的序號經由表格 item 找出對應的樹 ID，再從樹 ID 查詢表格 station_time 所對應到樹 ID 得到該樹中節點的(start,end)編碼，藉由滿足 range encoding 的編碼特性，可以快速尋找其所有祖先，也就是經過的站台和時間。而 Q'1-2 和 Q'1-1 的差別，只在於將輸入參數改為“product_no”。對 Q1-1 比較，原始做法要去龐大的 raw_data 表格做查詢。新的做法雖然必須連接二個表格，但每個表格的資料量都很少，只要透過適當的索引建立，即可加快查詢速度，此部分我們會後續進行實驗比較。

表 4-4：功能一改善後的查詢句

Q'1-1	L01: **SELECT** station, start_time, end_time L02: **FROM** item, station_time L03: **WHERE** item.serial_no = [serial_no] L04: **AND** item.tree_id = station_time.tree_id

	L05: **AND** item.start >= station_time.start L06: **AND** item.end <= station_time.end
Q'1-2	L01: **SELECT** item.serial_no, station, start_time, end_time L02: **FROM** item, station_time L03: **WHERE** item.product_no = [product_no] L04: **AND** item.tree_id = station_time.tree_id L05: **AND** item.start >= station_time.start L06: **AND** item.end <= station_time.end

接續功能二的改善查詢，如表 4-5，假設輸入參數為“serial_no”則其對應的 SQL 指令如 Q'2-1 所示，在 L02 我們將表格 station_time 定義兩個別名分別為 t1 和 t2，這兩個別名都代表此物品所經過的站台，而 L09-L10 限制條件中我們只選取不同的站台做組合並且站台和站台之間的時間差一定要大於 0。Q'2-2 需要限制在相鄰前、後站台，在 L01 先宣告一個名稱為 station_list 的陣列，L02-L05 先行查詢表格 flow 中該物品所走的流程，在限制條件 L11-L12 中將 t1 限定在前站($station_list[$i])，t2 限定在後站($station_list[$i+1])，而 L13 在 t1 的限制條件中因為沒有使用“=”，所以只取到最後一站的前一個站台，如此一來就能達到前、後站台的關係。與 Q2 比較，由於查詢的對象同樣由龐大的 raw_data 表格轉換成較輕小的 station_time，所以效率也會提升。

表 4-5：功能二改善後的查詢句

Q'2-1	L01: **SELECT** t1.station,t2.station,timediff(t2.start_time,t1.start_time) L02: **FROM** item,station_time **AS** t1, station_time **AS** t2 L03: **WHERE** item.serial_no = [serial_no] L05: **AND** t1.tree_id = item.tree_id

	L06: **AND** item.start >= t1.start **AND** t1.end >= item.end L07: **AND** item.start >= t2.start L08: **AND** t2.end >= item.end **AND**t2.tree_id = item.tree_id L09: **AND** t1.station != t2.station L10: **AND** timediff(t2.start_time,t1.start_time) > 0
Q'2-2	L01: $station_list =array() L02: **SELECT**distinct path **into** $station_list L03: **FROM** item,flow L04: **WHERE** item.serial_no = [serial_no] L05: **AND** flow.flow_id = item.flow_id L06: For ($i=0;$i<count($station_list);$i++){ L07: **SELECT** timediff(t2.start_time,t1.start_time) L08: **FROM** item, station_time **AS** t1, station_time **AS** t2 L09: **WHERE** item.serial_no = [serial_no] L10: **AND** item.tree_id = t1.tree_id **AND** item.tree_id = t2.tree_id L11: **AND** t1.station = [$station_list[$i]] L12: **AND** t2.station = [$station_list[$i+1]] L13: **AND** item.start > t1.start **AND** item.end < t1.end L14: **AND** item.end <= t2.end **AND** item.start >= t2.start}

其次是功能三改善如表 4-6,假設輸入參數為“product_no”則其對應的 SQL 指令 Q'3-1 所示，其做法與 Q'2-2 相似，差別在於輸入參數改為型號並且在 L16 以序號遞增為排序，取第一筆資料。而 Q'3-2 則沒有限制資料筆數的條件，而是根據相同型號所對應到的所有序號去計算其通過相鄰前、後站台的平均時間，並且在 L15 限定後站的時間下界，如此一來就能在時間範圍內計算同個型號的平均時間。

表 4-6：功能三改善後的查詢句

Q'3-1	L01: $station_list =array() L02: **SELECT** path **into** $station_list L03: **FROM** item, flow L04: **WHERE** item.product_no = [product_no]

	L05: **AND** flow.flow_id = item.flow_id L06: For ($i=0;$i<count($station_list);$i++){ L07: **SELECT** timediff(t2.start_time, t1.start_time) L08: **FROM** item,station_time **AS** t1, station_time **AS** t2 L09: **WHERE** item.product_no = [product_no] L10: **AND** item.tree_id = t1.tree_id **AND** item.tree_id = t2.tree_id L11: **AND** t1.station = [$station_list[$i]] L12: **AND** t2.station = [$station_list[$i+1]] L13: **AND** item.start > t1.start **AND** item.end < t1.end L14: **AND** item.end <= t2.end **AND** item.start >= t2.start L15: **ORDER BY** item.serial_no **LIMIT** 1}
Q'3-2	L01:$station_list = array() L02: **SELECT** path **into** $station_list L03: **FROM** item, flow L04: **WHERE** item.product_no = [product_no] L05: **AND** flow.flow_id = item.flow_id L06: For ($i=0;$i<count($station_list);$i++){ L07: **SELECT** SEC_TO_TIME(AVG(TIME_TO_SEC(timediff(b.time,a.time)))) L08: **FROM** item, station_time **AS** t1, station_time **AS** t2 L09: **WHERE** item.product_no = [product_no] L10: **AND**item.tree_id = t1.tree_id **AND** item.tree_id = t2.tree_id L11: **AND** t2.station = [$station_list[$i+1]] L12: **AND** t1.station = [$station_list[$i]] L13: **AND** item.start > t1.start **AND** item.end < t1.end L14: **AND** item.end <= t2.end **AND** item.start >= t2.start **AND** L15: t2.start_time **BETWEEN**[start_time] **AND** [end_time]}

最後是模組四改善後的查詢句，如表 4-7，假設輸入參數為"station_inform"、"start_time"、"end_time"則其對應的SQL指令如Q'4-1，L01 先宣告一個名稱為 station_inform 的二維陣列和變數$reader_time。由於 FTT 方法已經將兩個條碼讀碼器的 mac address 合併成站台時間資訊的一筆資料，因

此在時間區間的判定上，我們需要知道使用者選擇的條碼讀碼器是屬於哪個站台，並對應其進站與出站。因此使用者在網頁前端選擇條碼讀碼器的 mac address 然後 POST 到後端時，我們會利用該條碼讀碼器的 mac address 去查詢表格 reader(見表 3-1)以轉換至該條碼讀碼器所屬的站台名稱和作用並存放到$station_inform 陣列，譬如:$station_inform[*n*][0]表示第 *n* 個站台的名稱、$station_inform[*n*][1]表示第 *n* 個站台的作用，至於 L02-L06 則是判斷該讀碼器是屬於進站或出站，並檢查該站台的時間並設定$reader_time 變數，然後我們在 L13 的時間限制條件中，會透過變數$reader_time，來判斷是要檢查進站時間還是出站時間，若是條碼讀碼器是屬於 input 則需要檢查該站台的進站時間(start_time 欄位)有沒有在時間範圍內，若是條碼讀碼器是屬於 output 則檢查站台的出站時間(end_time 欄位)。在 14-20 文字清單(型號、序號)部分則同樣根據 range encoding 編碼特色和時間限制條件即可選出物品型號和序號。而 Q'4-2 和 Q'4-1 類似，差別在於 L14 用一個變數$sum 將每個站台所讀取的數量加總。

表 4-7：功能四改善後的查詢句

Q'4-1	L01: $station_inform = array(array()); $reader_time; L02: For ($i=0;$i<count($station_inform);$i++){ L03: if($station_inform[$i][1] = 'input') L04: $reader_time = 'start_time'; L05: if($station_inform[$i][1] = 'output') L06: $reader_time = 'end_time'; L07: **SELECT** count(serial_no) L08: **FROM** item, station_time L09: **WHERE** station_time.station = $station_inform [$i][0] L10: **AND** item.start >= station_time.start L11: **AND** item.end <= station_time.end L12: **AND** item.tree_id = station_time.tree_id **AND** L13: $reader_time **BETWEEN** [start_time] **AND** [end_time]; L14: **SELECT** product_no, serial_no L15: **FROM** item, station_time L16: **WHERE** station_time.station = $station_inform [$i][0]

	L17: **AND** item.start >= station_time.start L18: **AND** item.end <= station_time.end L19: **AND** item.tree_id = station_time.tree_id **AND** L20: $reader_time **BETWEEN** [start_time] **AND** [end_time]}
Q'4-2	L01: $station_inform =array(array()); $reader_time; $sum; L02: For ($i=0;$i<count($station_inform);$i++){ L03: if($station_inform[$i][1] = 'input') L04: $reader_time = 'start_time'; L05: if($station_inform[$i][1] = 'output') L06: $reader_time = 'end_time'; L07: **SELECT** count(serial_no) L08: **FROM** item, station_time L09: **WHERE** station_time.station = $station_inform [$i][0] L10: **AND** item.start >= station_time.start L11: **AND** item.end <= station_time.end L12: **AND** item.tree_id = station_time.tree_id **AND** L13: $reader_time **BETWEEN** [start_time] **AND** [end_time] L14: $sum += count(serial_no); } L15: **SELECT** product_no, serial_no L16: **FROM** item, station_time L17: **WHERE** station_time.station **IN** ($station_inform[0][0],…$station_inform[*n*][0]) L18: **AND** item.start >= station_time.start L19: **AND** item.end <= station_time.end L20: **AND** item.tree_id = station_time.tree_id L21: **AND** $reader_time **BETWEEN** [start_time] **AND** [end_time]

第 5 章實驗

在本章節中，第 3.2 節中的模組 B 之查詢句設計為方法「Baseline」，而在第四章透過 Flow-Time Tree 方法所組成的 SQL 指令稱為「FTT」。我們進行實驗來比較 Baseline 和 FTT 這兩個方法的時間效率以及中間資料的資料筆數。首先，先說明我們進行的實驗的環境，我們以個人電腦作為實驗環境，其 CPU 為四核心的 Intel i5 3550，其中每個核心的時脈為 3.3GHz 而記憶體為 8GB，所採用的作業系統為 Windows 7 企業版 SP1。

5.1 資料集

由於 bardcode data 沒有眾所皆知的真實資料集，因此我們基於部分工廠本身的真實情況加以延伸產生出人造資料集，其中包含工單號碼、批號、物品的型號和序號以及正確流程。如表 5-1 所示，我們有五張工單，型號分別為「FXC」、「AHD」、「AE」、「CS」和「BP」這五種，序號我們以流水號的方式來區別同個型號的不同物品，其每個工單的正確流程分別為“A,B,C,D”、“A,B,D”、“A,D,E”、“A,E”、“B,D,E”，其中英文字母對應到一個站台。我們假設每張工單在生產物品時其進、出站的時間相同，並且設定物品每一次的移動量為參數 G (G 設定大小為：10、100、1000、10000)，由於每個站台的進、出站各會產生一筆 raw_data，所以產生的資料筆數的計算方式為$\sum_1^5 G * \textit{正確流程的站台個數} * 2$。而 raw_data 的資料筆數分別為<300,3000,30000,300000>，四種不同的資料數量。

表 5-1：人造資料集的物品工單

工單號碼	批號	型號	序號(起)	序號(迄)	正確流程	G	總數
W1-10	B1-10	FXC	8A00529-00000	8A00529-00009	A,B,C,D	10	10*4*2=80
W2-10	B2-10	AHD	8A00528-00000	8A00528-00009	A,B,D	10	10*3*2=60
W3-10	B3-10	AE	8A00527-00000	8A00527-00009	A,D,E	10	10*3*2=60
W4-10	B4-10	CS	8A00526-00000	8A00526-00009	A,E	10	10*2*2=40
W5-10	B5-10	BP	8A00525-00000	8A00525-00009	B,D,E	10	10*3*2=60
工單號碼	批號	型號	序號(起)	序號(迄)	正確流程	G	總數
W1-100	B1-100	FXC	8A00529-00000	8A00529-00099	A,B,C,D	100	100*4*2=800
W2-100	B2-100	AHD	8A00528-00000	8A00528-00099	A,B,D	100	100*3*2=600
W3-100	B3-100	AE	8A00527-00000	8A00527-00099	A,D,E	100	100*3*2=600
W4-100	B4-100	CS	8A00526-00000	8A00526-00099	A,E	100	100*2*2=400
W5-100	B5-100	BP	8A00525-00000	8A00525-00099	B,D,E	100	100*3*2=600
工單號碼	批號	型號	序號(起)	序號(迄)	正確流程	G	總數
W1-1000	B1-1000	FXC	8A00529-00000	8A00529-00999	A,B,C,D	1000	1000*4*2=8000
W2-1000	B2-1000	AHD	8A00528-00000	8A00528-00999	A,B,D	1000	1000*3*2=6000
W3-1000	B3-1000	AE	8A00527-00000	8A00527-00999	A,D,E	1000	1000*3*2=6000
W4-1000	B4-1000	CS	8A00526-00000	8A00526-00999	A,E	1000	1000*2*2=4000
W5-1000	B5-1000	BP	8A00525-00000	8A00525-00999	B,D,E	1000	1000*3*2=6000
工單號碼	批號	型號	序號(起)	序號(迄)	正確流程	G	總數
W1-10000	B1-10000	FXC	8A00529-00000	8A00529-09999	A,B,C,D	10000	10000*4*2=80000
W2-10000	B2-10000	AHD	8A00528-00000	8A00528-09999	A,B,D	10000	10000*3*2=60000
W3-10000	B3-10000	AE	8A00527-00000	8A00527-09999	A,D,E	10000	10000*3*2=60000
W4-10000	B4-10000	CS	8A00526-00000	8A00526-09999	A,E	10000	10000*2*2=40000
W5-10000	B5-10000	BP	8A00525-00000	8A00525-09999	B,D,E	10000	10000*3*2=60000

我們在表 5-2 中列出我們在實驗所使用的查詢句，我們以分號做為輸出與限制條件的區隔，而中括弧為時間限制式。其中 S/N 和 P/N 代表物品的序號和型

號，條碼讀碼器的 mac address 用 M 和所屬站台的 input 或 output 表示。Q1-Q2 主要是追蹤物品經過各個站台的詳細資訊，對應到 3.2 節功能一。Q3-Q4 主要是針對單一物品經過各個站台的時間差，對應到 3.2 節功能二。Q5-Q7 主要是計算同個型號的物品經過各站台的時間差與平均時間，Q5 查詢句是針對該型號第一個進入站台的物品，以該物品進入到各個站台的時間點來做時間平均，而 Q6 和 Q7 查詢句主要是觀察時間限制的執行效率，由於模組 B 的功能三必須給定時間區間，所以在 Q6 設計上，將時間區間設定在該物品必須走完整個流程，也就是從 A 的進站時間到 D 站台的出站時間；Q7 則設定在 C 站台的出站時間，也就是只走到一半。Q8-Q11 主要是計算條碼讀碼器所讀取的物品數量和讀取物品清單，其中 Q8 與 Q10 以符號「|」表示計算“各個”條碼器所讀取的數量，Q9 與 Q11 以符號「+」表示計算“所有”條碼器所讀取的數量，對應到 3.2 節功能四。我們的測量執行時間的方式為：執行查詢句五次，並取平均值。

表 5-2：Query Set

Query Number	Query	Query Number	Query
Query1	<站台名稱, 進出站台時間; S/N=“8A00529-00000”>	Query7	<前站 L1, 後站 L2, AVG(L1,L2)); P/N=“FXC”, [A.start_time, C.end_time]>
Query2	<S/N, 站台名稱, 進出站台時間; P/N=“FXC”>	Query8	<count(),讀取清單; MAI|MAO|MBI|MBO>
Query3	<前站 L1, 所有後站 L2,Timediff(L2,L1)); S/N=“8A00529-00000”>	Query9	<count(), 讀取清單; MAI+MAO+MBI+MBO>
Query4	<前站 L1, 相鄰後站 L2, Timediff(L2,L1)); S/N=“8A00529-00000”>	Query10	< count( ), 讀取清單; MAI|MAO|MBI|MBO, [A.start_time, B.start_time]>
Query5	<前站 L1, 後站 L2, Timediff(L2,L1)); P/N=“FXC”,limit 1>	Query11	<count(),讀取清單; MAI+MAO+MBI+MBO, [A.start_time, B.start_time]>
Query6	<前站 L1, 後站 L2, AVG(L1,L2)); P/N=“FXC”, [A.start_time, D.end_time]>		

5.2 不同查詢句的效率評估與中間資料分析

首先我們先用一個比較小的資料集來測量 Baseline SQL 與 FTT SQL 的執行效率，這次實驗用的資料筆數為 300 筆。參考圖 5-1 及表 5-3，FTT SQL 的執行效率大多都是比 Baseline SQL 還要好，除了 Q8 和 Q10 這兩個查詢句之外，其他查詢句的執行時間都在 3 毫秒以內，主要是因為 FTT 的方法所比對的資料量較少。接下來，我們將探討根據不同大小的資料數量來仔細觀察各個查詢句的執行時間，並以最大的兩個資料集來分析資料量的變化，我們使用 MySQL 分析工具來幫助我們實驗測量。所有的詳細數據資料會放在附錄 C。

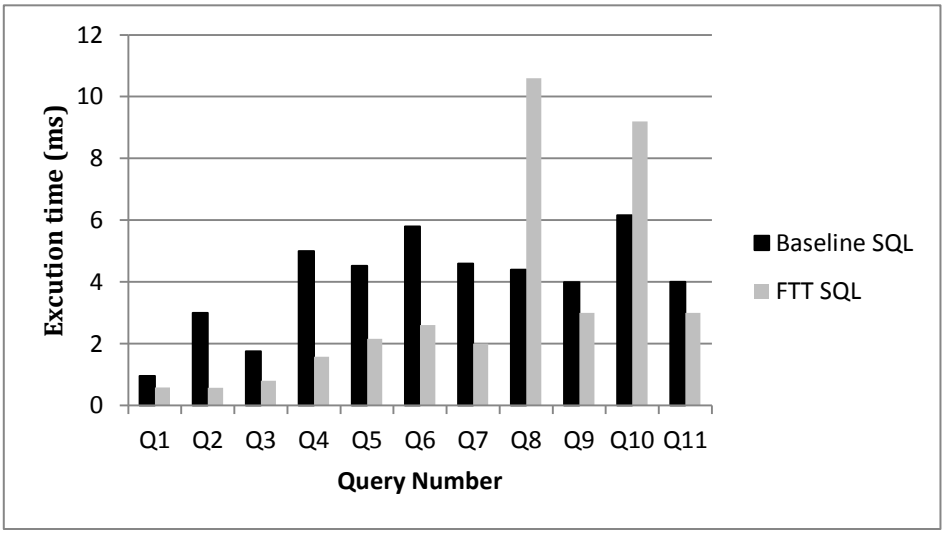
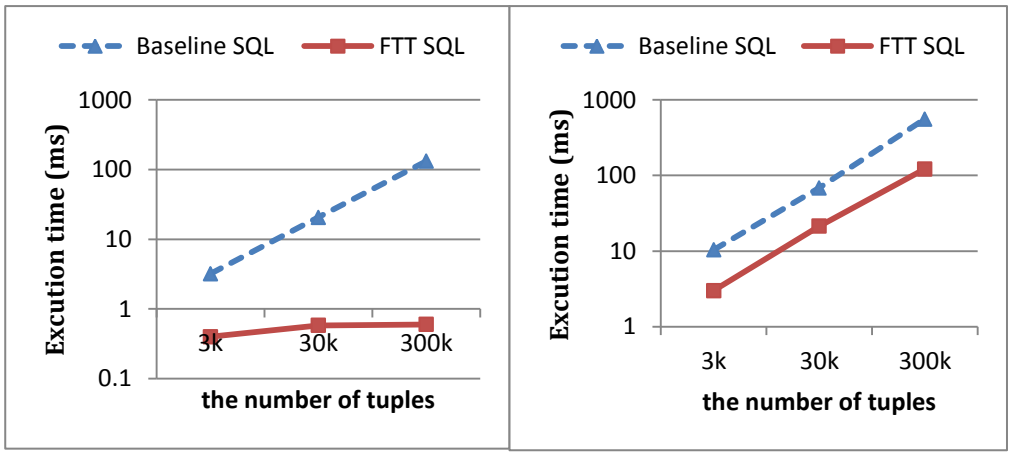


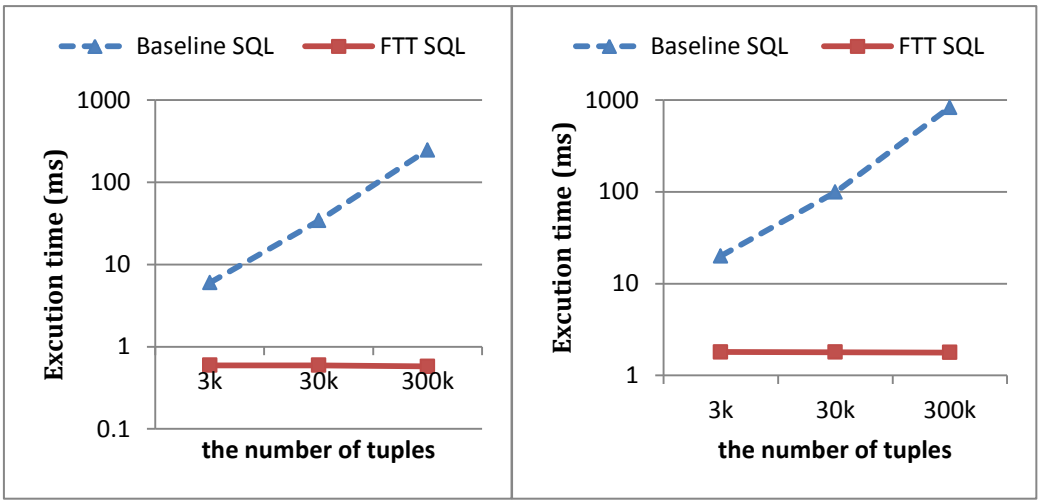
圖 5-1：Excution time for 11 queries

表 5-3：Q1-Q11 執行時間(ms)

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
Baseline SQL	0.96	3.000021	1.76	5	4.52	5.798	4.6	4.4	3.999	6.16	3.9999
FTT SQL	0.58	0.578	0.798	1.58	2.16	2.6	2	10.6	3	9.2	3



(a) Q1 (b) Q2



(c) Q3 (d) Q4

圖 5-2：Excution time for queries Q1-Q4.

參考圖 5-2，這次的實驗主要是針對追蹤物品所經過的各站台資訊的查詢句，在兩個方法中，FTT 的方法在 Q1 Q3 Q4 在執行時間上都不超過 0.01 秒即可完成，即使資料數量高達 10^6 仍能保持穩定效率，主要是因為此三個查詢句的限制條件剛好為表格的主鍵使得中間資料較少。如圖 5-3 查詢句 Q1 的分析，在(a)圖中我們依序列出對應的 SQL 查詢句，300k 資料集的處理過程，和 30k 資料集的處理過程。圖中顯示利用 Baseline 在查詢物品時，需要掃描整個 raw_data 表格才能找到其物品，再 join 表格 reader 來顯示站台資訊，(b)圖顯示 FTT 能很快速找到物

品，是因為物品序號為表格 item 的主鍵可快速找到第一筆，並透過(tree_id,start,end)欄位來查詢表格 station_time，而這三個欄位剛好也是表格 station_time 的主鍵，所以可以快速地取出對應的 4 筆資料。相較之下，FTT 不會因為數量增加而影響效能。也就是 Baseline 方法會受到 raw_data 大小的影響，時間複雜度為 O(n)，而 FTT 不隨資料量大小所影響，時間複雜度為 O(1)。

至於 Q2 在查詢同個型號的所有物品當中，FTT 也會線性成長，但其執行速度比 Baseline SQL 的執行速度快將近 4.5 倍。如圖 5-4(a)所示 Baseline 依然是掃描所有資料表格，所以仍為 O(n)，根據圖 5-4 (b)雖然 FTT 也是需要掃描整個表格 item，所以應為 O(n)，但由於 item 表格儲存經過整合過的 trace records，相較於 raw_data 的中間資料明顯少的多，就效率而言 FTT 比 Baseline 來的好。

SQL:

Select reader_name,station,function,time

From raw_data,reader

Where serial_no = '8A00529-00000' **and** raw_data.reader_mac = reader.reader_mac

Order by time **ASC**

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	raw_data	ALL	*NULL*	*NULL*	*NULL*	*NULL*	300000	Using where; Using temporary; Using filesort
1	SIMPLE	reader	eq_ref	PRIMARY	PRIMARY	302	result.raw_data.reader_mac	1	*NULL*

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	raw_data	ALL	*NULL*	*NULL*	*NULL*	*NULL*	30000	Using where; Using filesort
1	SIMPLE	reader	eq_ref	PRIMARY	PRIMARY	302	result.raw_data.reader_mac	1	*NULL*

(a) Baseline SQL

SQL:

Select station,start_time,end_time

From item,station_time

Where item.serial_no = '8A00529-00000' **and** item.tree_id = station_time.tree_id**and**

item.start >= station_time.start **and** station_time.end >= item.end

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	item	const	PRIMARY	PRIMARY	302	const	1	*NULL*
1	SIMPLE	station_time	range	PRIMARY	PRIMARY	8	*NULL*	4	Using where

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	item	const	PRIMARY	PRIMARY	302	const	1	*NULL*
1	SIMPLE	station_time	range	PRIMARY	PRIMARY	8	*NULL*	4	Using where

(b) FTT SQL

圖 5-3：Q1 的中間資料量

SQL:

Select reader_name,station,function,time

From raw_data,reader

Where product_no = 'FXC' **and** raw_data.reader_mac = reader.reader_mac

Order by time,serial_no **ASC**

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	raw_data	ALL	*NULL*	*NULL*	*NULL*	*NULL*	300000	Using where; Using filesort
1	SIMPLE	reader	eq_ref	PRIMARY	PRIMARY	302	result.raw_data.reader_mac	1	*NULL*

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	raw_data	ALL	*NULL*	*NULL*	*NULL*	*NULL*	30000	Using where; Using filesort
1	SIMPLE	reader	eq_ref	PRIMARY	PRIMARY	302	result.raw_data.reader_mac	1	*NULL*

(a) Baseline SQL

SQL:

Select station,start_time,end_time

From item,station_time

Where item.product_no = 'FXC' **and** item.tree_id = station_time.tree_id **and**

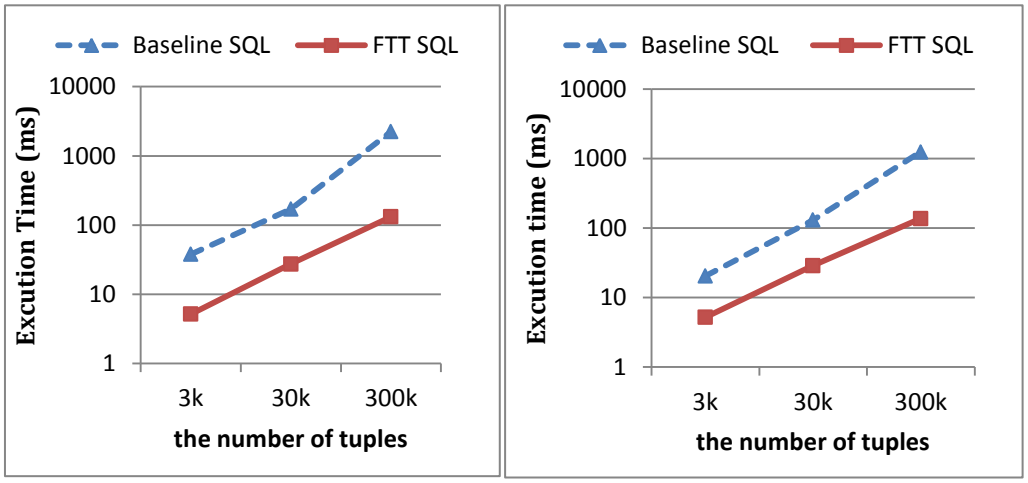
item.start >= station_time.start **and** station_time.end >= item.end

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	item	ALL	*NULL*	*NULL*	*NULL*	*NULL*	50000	Using where
1	SIMPLE	station_time	ref	PRIMARY	PRIMARY	4	result.item.tree_id	1	Using where

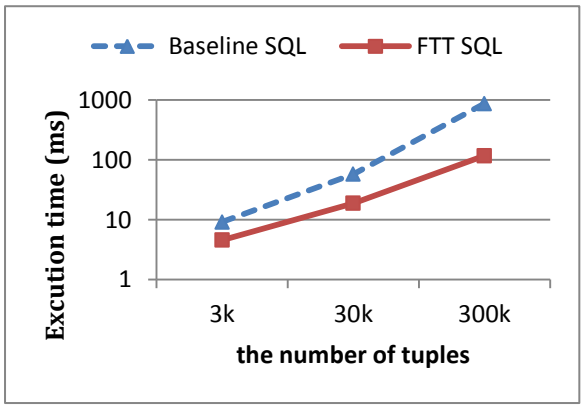
id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	item	ALL	*NULL*	*NULL*	*NULL*	*NULL*	5000	Using where
1	SIMPLE	station_time	ref	PRIMARY	PRIMARY	4	result.item.tree_id	1	Using where

(b) FTT SQL

圖 5-4：Q2 的中間資料量



(a) Q5 (b) Q6



(c) Q7

圖 5-5：Excution time for queries Q5-Q7.

接下來的實驗我們主要是測量 SQL 指令中 AVG()函式和 Timediff()函式並且比對在兩個方法中的執行效率。參考圖 5-5，Q5 主要是計算前、後兩站台的時間差(Timediff())，而 Q6 和 Q7 除了計算時間差之外還額外計算平均時間(AVG())。由於 Q5、Q6、Q7 兩個方法的效率表現雷同，所以我們以 Q5 為代表進行分析。比對兩個方法的執行效率，Baseline SQL 使用兩個延伸表格來實作前、後兩站台的時間差，如圖 5-6(a)首先延伸表格 derived2(也就是別名表格 a)掃描整個表格 raw_data 資料，其資料筆數為 300000 筆，其次延伸表格 derived3(也就是別名表格 b)也相同，最後，兩個 derived 表格的結果利用 serial_no 連接。比較兩個資料集的中間資料量，可得到其時間複雜度為 O(*n*)。對於圖 5-6 (b)，一樣是掃描掃描整個表格，其時間複雜度為 O(*n*)，但是 FTT SQL 可以很快地找到此物品在這個站台的進、出站時間，是因為我們將原始資料轉換成追蹤資料時，已經將資料給整合起來，經由流程時間樹將資料分類，使得要處裡的資料是大幅降低，如整合完的物品資料為 baseline 中 raw_data 表格的 1/6，而時間站台資訊(station_time)分別為 t1 和 t2，其中資料數量各為 11 筆，而且 BNL(block nested loop)在作交集時不一定會將全部資料都拿來 join，因此兩者相比 FTT SQL 的執行時間比 Baseline SQL 快將近一個 order。

SQL:

```
Select SEC_TO_TIME(AVG(TIME_TO_SEC(timediff(b.time,a.time))))
From (Select raw_data.time,raw_data.serial_no From raw_data,reader
        Where raw_data.reader_mac = reader.reader_mac and reader.function = 'input'
        and raw_data.product_no = 'FXC' and reader.station = '前站'
        and raw_data.time between 'A.start_time' and 'D.end_time' ) as a,
        (Select raw_data.time,raw_data.serial_no From raw_data,reader
        Where raw_data.reader_mac = reader.reader_mac and reader.function = 'input'
        and raw_data.product_no = 'FXC' and reader.station = '後站'
        and raw_data.time between 'A.start_time' and 'D.end_time' ) as b
Where a.serial_no = b.serial_no and timediff(b.time,a.time) > 0
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	*NULL*	*NULL*	*NULL*	*NULL*	300000	*NULL*
1	PRIMARY	<derived3>	ref	<auto_key0>	<auto_key0>	302	a.serial_no	10	Using where
3	DERIVED	raw_data	ALL	*NULL*	*NULL*	*NULL*	*NULL*	300000	Using where
3	DERIVED	reader	eq_ref	PRIMARY	PRIMARY	302	result.raw_data.reader_mac	1	Using where
2	DERIVED	raw_data	ALL	*NULL*	*NULL*	*NULL*	*NULL*	300000	Using where
2	DERIVED	reader	eq_ref	PRIMARY	PRIMARY	302	result.raw_data.reader_mac	1	Using where

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	*NULL*	*NULL*	*NULL*	*NULL*	30000	*NULL*
1	PRIMARY	<derived3>	ref	<auto_key0>	<auto_key0>	302	a.serial_no	10	Using where
3	DERIVED	raw_data	ALL	*NULL*	*NULL*	*NULL*	*NULL*	30000	Using where
3	DERIVED	reader	eq_ref	PRIMARY	PRIMARY	302	result.raw_data.reader_mac	1	Using where
2	DERIVED	raw_data	ALL	*NULL*	*NULL*	*NULL*	*NULL*	30000	Using where
2	DERIVED	reader	eq_ref	PRIMARY	PRIMARY	302	result.raw_data.reader_mac	1	Using where

(a) Baseline SQL

SQL:

Select SEC_TO_TIME(AVG(TIME_TO_SEC(timediff(t2.start_time,t1.start_time))))

From item,station_time **as** t1,station_time **as** t2

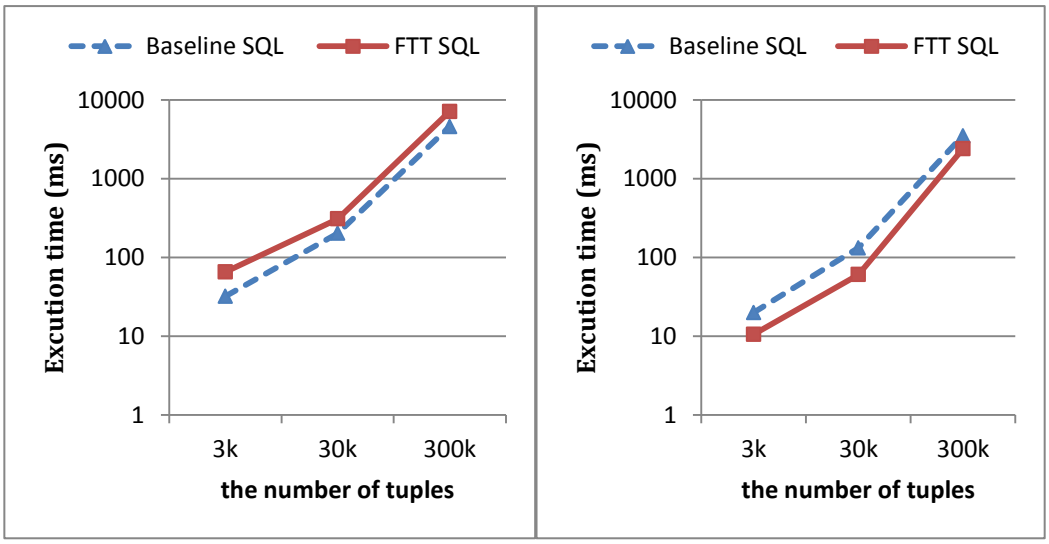
Where item.product_no = 'FXC '**and** item.tree_id = t1.tree_id **and** item.tree_id = t2.tree_id **and** t1.station = '前站' **and** t2.station = '後站' **and** item.start > t1.start **and** item.end < t1.end **and** item.start >= t2.start **and** item.end <= t2.end **and** t2.start_time **between** 'A.start_time' **and** 'D.end_time'

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	item	ALL	*NULL*	*NULL*	*NULL*	*NULL*	50000	Using where
1	SIMPLE	t1	ALL	PRIMARY	*NULL*	*NULL*	*NULL*	11	Using where; Using join buffer (Block Nested Loop)
1	SIMPLE	t2	ALL	PRIMARY	*NULL*	*NULL*	*NULL*	11	Using where; Using join buffer (Block Nested Loop)

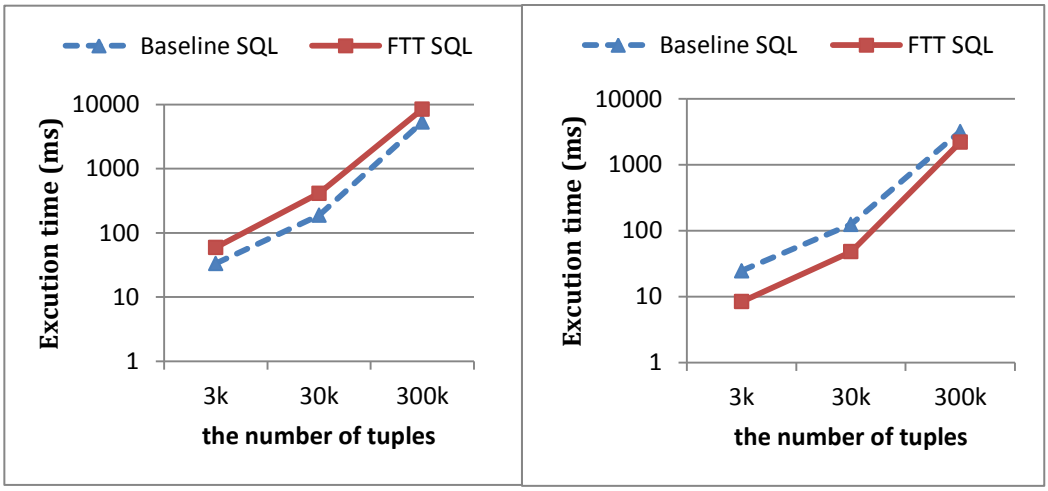
id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	item	ALL	*NULL*	*NULL*	*NULL*	*NULL*	5000	Using where
1	SIMPLE	t1	ALL	PRIMARY	*NULL*	*NULL*	*NULL*	11	Using where; Using join buffer (Block Nested Loop)
1	SIMPLE	t2	ALL	PRIMARY	*NULL*	*NULL*	*NULL*	11	Using where; Using join buffer (Block Nested Loop)

(b) FTT SQL

圖 5-6：Q6 的中間資料量



(a) Q8 (b) Q9



(c) Q10 (d) Q11

圖 5-7：Excution time for queries Q8-Q11.

SQL:

Loop M[MAI,MAO,MBI,MBO**]**

[Select count(serial_no)

From raw_data

Where reader_mac = ‘M[i]’;]

End loop

Select distinct product_no,serial_no

From raw_data

Where reader_mac **IN** (’MAI’,’MAO’,’MBI’,’MBO’)

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	raw_data	ALL	*NULL*	*NULL*	*NULL*	*NULL*	300000	Using where; Using temporary

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	raw_data	ALL	*NULL*	*NULL*	*NULL*	*NULL*	30000	Using where; Using temporary

(a)Baseline SQL

SQL:

Loop station[A,B**]**

[Select count(serial_no)

From item,station_time

Where station_time = ‘**station[i]**’ **and** item.tree_id = station_time.tree_id **and**

item.start >= station_time.start **and** station_time.end >= item.end **]**

end loop

Select distinct product_no, serial_no

From item, station_time

Where station_time.station **IN** (‘A’,’B’) **and** item.tree_id = station_time.tree_id **and**

item.start >= station_time.start **and** station_time.end >= item.end

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	item	ALL	*NULL*	*NULL*	*NULL*	*NULL*	50000	Using temporary
1	SIMPLE	station_time	ref	PRIMARY	PRIMARY	4	result.item.tree_id	1	Using where; Distinct

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	item	ALL	*NULL*	*NULL*	*NULL*	*NULL*	5000	Using temporary
1	SIMPLE	station_time	ref	PRIMARY	PRIMARY	4	result.item.tree_id	1	Using where; Distinct

(b)FTT SQL

圖 5-8：Q9 的中間資料量

最後的實驗我們主要是測量 SQL 指令中 count()函式以及顯示條碼讀碼器所讀取的物品清單。參考圖 5-7(a) (c)，Q8 和 Q10 的執行效率 Baseline 比 FTT 的方法要來的好，主要是因為 Baseline SQL 在查詢原始資料(raw_data)中就已經有條碼讀碼器的 mac address 欄位，可直接計算出該條碼讀碼器所讀取的數量以及該物品資訊。而 FTT SQL 在轉換成追蹤資料(trace record)時，就已經將條碼讀碼器的mac address轉換成站台名稱,所以當要計算條碼讀碼器所讀取物品的數量時，FTT SQL 要反過來從前端得知該條碼讀碼器是屬於哪個站台，當資料送到後端時需要判斷該條碼器是屬於進站還出站，再根據表格 station_time 中的站台名稱去比對所有物品資訊是屬於哪個站台,處裡上較為麻煩,因此執行時間比Baseline SQL 要來的久。

參考圖 5-7(b) (d)，在 Q9 和 Q11 的執行效率方面還是 FTT 方法比較好一點點，其執行步驟如圖 5-8 所示，在彙整物品資料時 Baseline 方法需要掃描整個 raw_data 表格再 distinct 重複的資料，其時間複雜度為 O(n)，而 FTT 方法也是需要掃描整個資料集但是資料量比 Baseline SQL 少,因此FTT 在效率上會快一些。

第 6 章結論與未來方向

在本論文中，我們在雲端儲存系統 NAS 上，藉由 PHP 和 MYSQL 建構出一個具有基本功能的製造執行系統，我們特別針對模組 B 生產流程查詢，提供兩種方法。第一個 Baseline 方法，是依照使用者下達的限制條件組成適當的 SQL 語法，查詢由原始條碼資料所組成的表格 raw_data，此方法會有查詢效率不佳的問題。為了解決上述情況，我們進一步提出了 FTT 方法。在此方法中，我們將表格 raw_data 轉換成 trace record 的形式，透過 exactly matching 的方式將工單資訊(work_order)的 flow 欄位和 trace record 所經過的站台列表整個做比對，並以比對正確的 trace record 來建樹，然後寫出對應的 SQL 語法進行生產流程查詢。

我們進行實驗來評估這兩種方法，分析各個查詢句在不同數量時的執行效率和產生的中間資料量。結論在效率部分，FTT 方法大多比 Baseline 方法來的好，在 Q1 Q3 Q4 查詢句不會因為資料數量增加而影響執行效率，FTT Q2 查詢句執行效率上比 Baseline 方法快 4.5 倍。而 Q5-Q7 查詢句在 Baseline 方法使用兩個延伸表格去 join，導致中間資料量比對次數比 FTT 方法多，因此 FTT 效率仍然比較好。最後在 Q8 和 Q10 查詢句 FTT 執行效率比較差是因為在彙整物品資料時，FTT 方法需要兩個表格去 join 查詢，最後用迴圈去執行各個條碼器所讀取的數量並判斷哪個物品是屬於哪個站台，除了判斷變多之外，中間資料量查詢次數反而比 Baseline 方法多。

最後，關於本論文未來的研究方向，由於 FTT 方法是經由前處理方式比對錯誤資訊來偵測錯誤流程，希望能改善此方法，而達到即時且快速地偵測到經過錯誤流程的物品。

參考文獻

[OMES] Oracle MES User Guide, https://docs.oracle.com/cd/B34956_01/current/acrobat/120gmomesug.pdf.

[QNAP] QNAP二次開發平台, http://www.qnap.com/dev/cht/.

[LC08] Chun-Hee Lee, Chin-Wan Chung, “Efficient Storage Scheme and Query Processing for Supply Chain Management Using RFID”, Proceedings of the SIGMOD conference, 2008.

[CSDS11] Zhao Cao, Charles Sutton, YanleiDiao, Prashant Shenoy, “Distributed inference and query processing for RFID tracking and monitoring”, Proceedings of the VLDB conference, 2011.

[LC11] Chun-Hee Lee, Chin-Wan Chung, “RFID Data Processing in Supply Chain Management Using a Path Encoding Scheme”, IEEE on Transactions on Knowledge and Data Engineering, Vol. 23, No. 25, pp. 742-758, 2011.

[NW11] Wilfred Ng, “Developing RFID Database Models for Analysing Moving Tags in Supply Chain Management”, Proceedings of the ER conference, 2011.

[LC13] Yao Lu, Lijun ChenAn, “Improved RFID Data Encoding Scheme for Cycling Path Problem”, Proceedings of the ICSESS conference, 2013.

[TCCY13] Yung-Shun Tsai, Ruey-Shun Chen, Yeh-Cheng Chen, Chun-Ping Yeh, “An RFID-based Manufacture Process Control and Supply Chain Management in the Semiconductor Industry”, International Journal of Technology Management, Vol.12, No. 1/2, pp. 85-105, 2013.

[LXBC14] Jia Liu, Bin Xiao, Kai Bu, Ijun Chen, “Efficient Distributed Query

Processing in Large RFID-enabled Supply Chains", Proceedings of the INFOCOM conference, 2014.

[TSH14] Tatsuo Tsuji, Ryota Shimono, Ken Higuchi, "Design and Evaluation of a Storage Framework for Supply Chain Management", Proceedings of the DEXA conference, 2014.

附錄 A　模組 A 的條碼處理程式

以下大致介紹此 PHP 程式。L01-L08 的函式主要是將字串轉換成 binary 的形式，以回傳封包給條碼讀碼器。而 L09-L28 處理封包格式 0x7901，主要存放影像的相關資訊如長、寬、高等，L29-L37 為封包格式 0x7902，主要存放影像資料如點陣圖。L38-L59 處理封包格式 0x7903，主要擷取條碼讀碼器的相關設定如(IP,mac_address,reader_name)。L60 處理封包格式 0x7611，主要回傳解碼的條碼內容，我們在 L65-L66 得到條碼內容，並依據空白切割出型號、序號，並分別放入$tag_array[0]、$tag_array[1]陣列中，在 L72 判斷此條碼器是否為異常，當讀碼器為 normal 時，在 L74-L93 查詢此物品有沒有被更改批號，若是沒被更改，就根據工單去查詢此物品的工單號碼和批號。L94-L104 查詢此物品之前所走的流程，L105-109 判斷條碼讀碼器為 output 作用時，直接對 raw_data 表格和 productstation 表格存入資料，L110-L118 則判斷條碼讀碼器為 input 作用時，必須和 work_order 的正確流程做前序比對，比對正確時，才能將資料存入 raw_data 和 productstation 表格。L120-L136 主要是判斷為 raw_data 為空時，第一筆資料就直接和工單資訊做前序比對，正確比對才能將資料加入 raw_data 和 productstation 表格，比對到錯誤流程則將物品資訊存入 error_flow 表格中。L137-L139 將影像資訊存入本機端電腦，最後在 L145-L147 將我們所用到的變數全部初始化。

```
L01: function conver2bin($string) {
L02:            $msg_array = explode(":", $string);
L03:            $binmsg = "";
L04:            for ($i = 0; $i <count($msg_array); $i++) {
L05:            $binmsg .= pack("H*", $msg_array[$i]);
L06:            }
L07:            return $binmsg;
L08:}//convert sting to binary
L09: if ($uscommandtype == '0179') {//0x7901封包
```

```
L10: $usDataIndex = bin2hex(substr($buffer, 4, 2));
L11: $usDataSize = bin2hex(substr($buffer, 6, 2));
L12: $uiVirtual = bin2hex(substr($buffer, 8, 4));
L13: $uiImageID = bin2hex(substr($buffer, 12, 4));
L14: $uiImageFormat = hexdec(bin2hex(substr($buffer, 16, 4)));
L15: $uiImageFormat = hexdec($uiImageFormat[6] . $uiImageFormat[7] .
                             $uiImageFormat[4] . $uiImageFormat[5] .
                             $uiImageFormat[2] .$uiImageFormat[3] .
                             $uiImageFormat[0] . $uiImageFormat[1]);
L16: $uiImageWidth = bin2hex(substr($buffer, 20, 4));
L17: $uiImageWidth = hexdec($uiImageWidth[6] . $uiImageWidth[7] .
                            $uiImageWidth[4] .$uiImageWidth[5] .
                            $uiImageWidth[2] .$uiImageWidth[3] .
                            $uiImageWidth[0] . $uiImageWidth[1]);
L18: $uiImageHeight = bin2hex(substr($buffer, 24, 4));
L19: $uiImageHeight = hexdec($uiImageHeight[6] . $uiImageHeight[7] .
                            $uiImageHeight[4] . $uiImageHeight[5] .
                            $uiImageHeight[2] .$uiImageHeight[3] .
                            $uiImageHeight[0] . $uiImageHeight[1]);
L20: $uiImageSize = $uiImageWidth * $uiImageHeight;
L21: $uiPackageNumber = bin2hex(substr($buffer, 32, 4));
L22: $uiPackageNumber = hexdec($uiPackageNumber[6] . $uiPackageNumber[7] .
                               $uiPackageNumber[4] . $uiPackageNumber[5] .
                               $uiPackageNumber[2] .$uiPackageNumber[3] .
                               $uiPackageNumber[0] . $uiPackageNumber[1]);
L23: $uiDeviceName1 = bin2hex(substr($buffer, 36, 4));
L24: $uiDeviceName1 = $uiDeviceName1[6] . $uiDeviceName1[7] .
                              $uiDeviceName1[4] .$uiDeviceName1[5] .
                              $uiDeviceName1[2] . $uiDeviceName1[3] .
                              $uiDeviceName1[0] . $uiDeviceName1[1];
L25: $uiDeviceName2 = bin2hex(substr($buffer, 40, 4));
L26: $uiDeviceName2 = $uiDeviceName2[6] . $uiDeviceName2[7] .
                              $uiDeviceName2[4] .$uiDeviceName2[5] .
                              $uiDeviceName2[2] . $uiDeviceName2[3] .
                              $uiDeviceName2[0] . $uiDeviceName2[1];
L27: $resend="cc:52:12:09:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00";
L28: socket_write($client, conver2bin($resend), strlen(conver2bin($resend)));}
L29: if ($uscommandtype == '0279') {
```

```
L30: $usDataIndex = bin2hex(substr($buffer, 4, 2));
L31: $usDataIndex = hexdec($usDataIndex[2] . $usDataIndex[3] .
                           $usDataIndex[0] .$usDataIndex[1]);
L32: $usDataSize = bin2hex(substr($buffer, 6, 2));
L33: $uiVirtual = bin2hex(substr($buffer, 8, 4));
L34: $img_data[$usDataIndex] = substr($buffer, 12);
L35: $count_pack++;
L36: $resend = "cc:52:14:09:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00";
L37: socket_write($client, conver2bin($resend), strlen(conver2bin($resend)));}
L38: if ($uscommandtype == '0379') {
L39: $usDataIndex = bin2hex(substr($buffer, 4, 2));
L40: $usDataSize = bin2hex(substr($buffer, 6, 2));
L41: $uiVirtual = bin2hex(substr($buffer, 8, 4));
L42: $device_name1 = substr($buffer, 20, 4); //device_name1
L43: $device_name2 = substr($buffer, 24, 4); //device_name2
L44: $devicename = $device_name1[2] . $device_name1[3] . $device_name1[0] .
                   $device_name1[1] . $device_name2[2] . $device_name2[3] .
                   $device_name2[0] . $device_name2[1];
L45: $ethernet_local_ip = bin2hex(substr($buffer, 192, 4));
L46: $ethernetIP = hexdec($ethernet_local_ip[0] . $ethernet_local_ip[1]) . "." .
                   hexdec($ethernet_local_ip[2] . $ethernet_local_ip[3]) . "." .
                   hexdec($ethernet_local_ip[4] . $ethernet_local_ip[5]) . "." .
                   hexdec($ethernet_local_ip[6] . $ethernet_local_ip[7]);//IP
L47: $ethernet_gateway_ip = bin2hex(substr($buffer, 192, 4));
L48: $ethernet_localmask_ip = bin2hex(substr($buffer, 196, 4));
L49: $ethernet_mac_address1 = bin2hex(substr($buffer, 200, 4));
L50: $ethernet_mac_address2 = bin2hex(substr($buffer, 204, 4));
L51: $ethernet_mac = $ethernet_mac_address1[0] .
                     $ethernet_mac_address1[1] . ":" .$ethernet_mac_address1[2] .
                     $ethernet_mac_address1[3] . ":" .$ethernet_mac_address1[4] .
                     $ethernet_mac_address1[5] . ":" .$ethernet_mac_address1[6] .
                     $ethernet_mac_address1[7] . ":" .$ethernet_mac_address2[0] .
                     $ethernet_mac_address2[1] . ":" .$ethernet_mac_address2[2] .
                     $ethernet_mac_address2[3];//mac_address
L52: if ($uiPackageNumber != $count_pack) {
L53: $count_pack = 0;
L54: unset($img_data);
L55: $resend = "cc:52:11:09:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00";
```

```
L56: socket_write($client, conver2bin($resend), strlen(conver2bin($resend)));}
L57: else {
L58: $resend = "cc:52:10:09:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00";
L59: socket_write($client, conver2bin($resend), strlen(conver2bin($resend)));}
L60: if ($uscommandtype == '1176'){
L61: $station = '';$flow = '';$batch_no = '';$wip_no = '';$i = 0;
L62: $usDataIndex = bin2hex(substr($buffer, 2, 2));
L63: $usDataSize = bin2hex(substr($buffer, 6, 2));
L64: $uiVirtual = bin2hex(substr($buffer, 8, 4));
L65: $ucDecodeResult = substr($buffer, 84);
L66: $tag_array = explode(" ", $ucDecodeResult);
L67: $query="select status from reader where reader_mac='".$ethernet_mac."';";
L68: $result=mysql_query($query);
L69: $row=mysql_fetch_array($result);
L70: $reader_status=$row['status'];
L71: echo $reader_status;
L72: if($reader_status=="normal"){
L73: if(isset($tag_array[0]) && isset($tag_array[1]) && $tag_array[0]!=''){
L74: $query = "select batch_no,all_serial_no,wip_no from new_work_order where
L75: product_no='$tag_array[0]' order by time DESC;";
L76: $result = mysql_query($query);
L77: while ($row = mysql_fetch_array($result)) {
L78: $find_new_batch[$i] = array('batch_no' => $row['batch_no'], 'wip_no' =>
L79: $row['wip_no'], 'all_serial_no' => json_decode($row['all_serial_no'], TRUE));
L80: $i++;}
L81: for ($j = 0; $j < count($find_new_batch); $j++) {
L82: for($k = 0; $k<count($find_new_batch[$j]['all_serial_no']
     ['new_batch_serial']); $k++) {
L83: if ($tag_array[1] ==
          $find_new_batch[$j]['all_serial_no']['new_batch_serial'][$k]){
L84: $batch_no = $find_new_batch[$j]['batch_no'];
L85: $wip_no = $find_new_batch[$j]['wip_no'];
L86: break 2;}}}
L87: $query = "select wip_no,batch_no,flow from work_order where
                  product_no='$tag_array[0]' and '$tag_array[1]' between
                  serial_no_start and serial_no_end;";
L88: $result = mysql_query($query);
L89: $row_work_order = mysql_fetch_array($result);
```

```
L90: if(isset($row_work_order) && $row_work_order!=""){
L91: $flow = $row_work_order['flow'];//工單上面的正確流程
L92: if ($batch_no == ''){$batch_no = $row_work_order['batch_no'];}
L93: if ($wip_no == ''){$wip_no = $row_work_order['wip_no'];}
L94: $query_raw_data = "select flow from raw_data,reader where
                        raw_data.reader_mac=reader.reader_mac
                        and product_no='$tag_array[0]' and
                        serial_no='$tag_array[1]' order by time DESC limit 1;";
L95: $result_raw_data = mysql_query($query_raw_data);
L96: $row_raw_data = mysql_fetch_array($result_raw_data);
L97: if(isset($row_raw_data['flow'])){
L98: $old_flow=$row_raw_data['flow'];//物品目前走到的流程
L99: $query_station = "select station,function from reader where
                      reader_mac='".$ethernet_mac."';";
L100: $result_station = mysql_query($query_station);
L101: $row_station = mysql_fetch_array($result_station);
L102: if(isset($row_station)){
L103: $station = $row_station['station'];//目前讀碼器所屬站台
L104: $new_function=$row_station['function'];//目前讀碼器是進站還出站
L105: if($new_function =='output'){
L106: $query = "INSERT INTO raw_data(auto_id,reader_name,reader_ip
                ,reader_mac,roduct_no,serial_no,image,time,flow)VALUES(null,
'$devicename','$ethernetIP','$ethernet_mac','$tag_array[0]','$tag_array[1]','$tag_array[
1].bmp',NOW(),'$old_flow');";
L107:mysql_query($query);
L108:$query = "INSERT INTO productstation(auto_id,product_no,serial_no,
reader_mac,batch_no,storage_no,time,user)VALUES(null,'$tag_array[0]',
'$tag_array[1]','$ethernet_mac','$batch_no','',NOW(),'');";
L109: mysql_query($query);}
L110: if($new_function =='input'){
L111: $current_flow=$old_flow.$station.',';
L112: if(substr_compare($flow,$current_flow,0,strlen($current_flow))==0 ||
      substr_compare($flow,$current_flow,0,strlen($current_flow))==-1){
L113: $query = "INSERT into raw_data(auto_id,reader_name,reader_ip,reader_mac
,product_no,serial_no,image,time,flow)VALUES(null,'$devicename','$ethernetIP'
,'$ethernet_mac','$tag_array[0]','$tag_array[1]','$tag_array[1].bmp',NOW()
,'$current_flow');";
L114: mysql_query($query);
```

```
L115: $query = "INSERT into productstation(auto_id,product_no,serial_no
,reader_mac,batch_no,status,storage_no,time,user)VALUES(null,'$tag_array[0]'
,'$tag_array[1]','$ethernet_mac','$batch_no',",",NOW(),");";
mysql_query($query);}
L116: if(substr_compare($flow,$current_flow,0,strlen($current_flow))<-1){
L117: $query = "INSERT INTO error_flow(product_no,serial_no,wip_no,batch_no
,time,flow)VALUES('$tag_array[0]','$tag_array[1]','$wip_no','$batch_no'
,NOW(),'$current_flow');";
L118: mysql_query($query);}}}
L119: else{echo "can't not find this station_name";}}
L120: else{echo "not found in the raw_data\n";
L121: $query_station = "select station from reader where
                reader_mac='".$ethernet_mac."';";
L122: $result_station = mysql_query($query_station);
L123: $row_station = mysql_fetch_array($result_station);
L124: if(isset($row_station)){
L125: $first_station = $row_station['station'].",";
L126: if(substr_compare($flow,$first_station,0,strlen($first_station))==0){
L127: $query = "INSERT into raw_data(auto_id,reader_name,reader_ip,reader_mac
,product_no,serial_no,image,time,flow)VALUES (null,'$devicename','$ethernetIP'
,'$ethernet_mac','$tag_array[0]','$tag_array[1]','$tag_array[1].bmp',NOW()
,'$first_station');";
L128: mysql_query($query);
L129: $query = "INSERT INTO productstation(auto_id,product_no,serial_no
,reader_mac,batch_no,time,user)VALUES (null,'$tag_array[0]','$tag_array[1]'
,'$ethernet_mac','$batch_no',NOW(),");";
L130: mysql_query($query);}
L131: else{
L132:$query = "INSERT into error_flow(product_no,serial_no,wip_no,batch_no
,time,flow)VALUES ('$tag_array[0]','$tag_array[1]','$wip_no','$batch_no'
,NOW(),'$first_station');";
mysql_query($query);}}
L133: else{echo "can't not find this station_name";}}}
L134: else{
L135: $query = "INSERT INTO error_flow(product_no,serial_no,wip_no,
batch_no,time,flow)VALUES ('$tag_array[0]','$tag_array[1]','$wip_no','$batch_no'
,NOW(),");";
L136: mysql_query($query);}}}
```

```
$fp = fopen($tag_array[1] . ".bmp", 'w');
ksort($img_data);
foreach ($img_data as $key => $value) {fwrite($fp, $value);}
fclose($fp);
unset($img_data);
$count_pack = 0;
$resend = "cc:52:13:09:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00";
socket_write($client, conver2bin($resend), strlen(conver2bin($resend)));

//flush varienty
$device_name = "";$uiImageWidth = 0;$uiImageHeight = 0;
$uiPackageNumber = 0;$count_pack = 0;$flow=";$first_station=";
$current_flow=";$old_flow=";$station=";$query=";}}
```

附錄 B　Trigger 觸發程序

```
Create Trigger ‘add_amount’ After Insert On ‘productstation’;
For Each Row If ((SELECTproduct_no
                    From reader_amount
                    Where reader_amount.product_no = NEW.product_no
                        AND reader_amount.reader_mac = NEW.reader_mac
                        Limit 1) = NEW.product_no)
Then
    Update reader_amount set amount = amount+1,time = NEW.time
    Where product_no = NEW.product_no AND
        reader_amount.reader_mac = NEW.reader_mac;
Else
    Insert into reader_amount (product_no,reader_mac,start_time,time,amount)
    Values (NEW.product_no,NEW.reader_mac,NOW(),NOW(),1);
End If
```

附錄 C　實驗數據

Q1	0.3k	3k	30k	300k
Baseline SQL	0.96	3.198	20.6	132.2
FTT SQL	0.58	0.4	0.58	0.6
Q2	0.3k	3k	30k	300k
Baseline SQL	3.000	10.4	68.4	555.8
FTT SQL	0.578	3	21.4	122
Q3	0.3k	3k	30k	300k
Baseline SQL	1.76	5.996	34.2	247
FTT SQL	0.798	0.594	0.594	0.576
Q4	0.3k	3k	30k	300k
Baseline SQL	5	20	99.2	830
FTT SQL	1.58	1.796	1.794	1.778
Q5	0.3k	3k	30k	300k
Baseline SQL	4.52	38	170.6	2241.584
FTT SQL	2.16	5.192	27.4	132
Q6	0.3k	3k	30k	300k
Baseline SQL	5.798	20.4	131	1251.2
FTT SQL	2.6	5.198	28.8	137.6
Q7	0.3k	3k	30k	300k
Baseline SQL	4.6	9.16	57.8	870.6
FTT SQL	2	4.56	18.8	117.6
Q8	0.3k	3k	30k	300k
Baseline SQL	4.4	32.2	202.6	4604.133
FTT SQL	10.6	65.4	308.4	7142.66
Q9	0.3k	3k	30k	300k
Baseline SQL	3.999	20.0002	132.4	3509.542
FTT SQL	3	10.58	60.4	2396.872
Q10	0.3k	3k	30k	300k
Baseline SQL	6.16	33.4	188	5348.816
FTT SQL	9.2	59.2	412.6	8478.18
Q11	0.3k	3k	30k	300k
Baseline SQL	3.999	24.6	124.6	3201.876
FTT SQL	3	8.38	47.8	2200.26

表中之時間單位皆為 ms。