資料流查詢最佳化之研究

蔡秀滿
明新科技大學資工系
pauray@must.edu.tw

林文宗
明新科技大學資工系
wtlin@must.edu.tw

彭雅筠
明新科技大學資管系
168619@yahoo.com.tw

摘要

在許多實際的應用中，例如網路流量分析(network traffic analysis)、網頁點選串流探勘(Web click stream mining)、以及線上即時交易分析(on-line transaction analysis)等，我們所要處理的資料不再是靜態的資料，而是一連串即時且連續的動態資料流(dynamic data stream)。在本論文中，我們針對交易資料流，考慮多資料流(multiple data streams)與關聯表(relations)的“視窗結合”(window join)運算以及“視窗聚合”(window aggregate)運算。我們發現，傳統資料庫中查詢最佳化的方法不適合完全套用在交易資料流的環境中。我們提出三種可能的查詢處理方案，並且探討查詢最佳化的處理方式，以期能夠更有效率地處理交易資料流的查詢。

關鍵字：資料探勘、資料流探勘、查詢處理、查詢最佳化

Abstract

In many applications such as network traffic analysis, Web click stream mining, and on-line transaction analysis, the data we need to process is not static, but a continuous dynamic data stream. In this paper, we focus on transactional data streams, and consider “window join” operation and “window aggregate” operation for multiple data streams and relations. We find that the optimization techniques used in the traditional relational database system can not be directly applied on the data stream processing system. We propose three possible query execution plans, and investigate the techniques of query optimization, in order to efficiently process the query of transactional data streams.

Keywords: data mining, data stream mining, query processing, query optimization

1.前言

在高速資料流的環境中，為了有效處理“連續性的查詢”(continuous query)，通常只允許資料被讀取一次，並且每隔一段時間，必須對查詢做出最新的回應。由於本質上的差異，和傳統關聯式資料庫系統相較起來，資料流查詢處理的研究所面臨的挑戰將更為複雜。

由於資料流是以極快速的方式，大量且持續產生的一連串資料，因此對於資料流的處理必須滿足下列三個要求[14]：(1)資料流中的每一個資料項目最多只能被檢視一次。(2)資料流中新資料不斷的產生，但是處理資料流所能使用的記憶體是有限的。(3)最新資料流的處理結果必須要能夠配合使用者的要求快速產生。由於傳統資料庫管理系統以“集合”概念處理記錄的方式無法直接應用在資料流的環境中，因此隨著資料流應用的興起，將使得資料管理系統面臨一個極大的挑戰。

連續查詢最早出現在 Tapestry [13]系統中，它的查詢語言被稱為 TQL(Tapestry Query Language)，是一種以 SQL 結構化查詢語言為基礎所設計出來的語言，主要是用來處理 append-only 型態的資料庫之查詢。 TQL 查詢在每個時間點會對資料庫的 snapshot 執行一次查詢，並且使用 “set union”來合併這些

one-time queries 的結果。 Tapestry 沒有提供移動視窗(sliding window)的描述功能，而且在查詢的執行上也沒有考慮資料流的資料只被讀取一次的要求。

Arasu et al.[4] 提出了 CQL 語言(Continuous Query Language)，他們使用 SQL 語言的架構以及視窗(window)的概念，將資料流對應到關聯表，以呈現查詢的真實語意。CQL 已經被實作在 Stanford 大學計畫中一個名叫 *STREAM* 的資料流管理系統模型中。CQL 所使用的資料型態包括 streams 和 relations，另外它也提供三類主要的運算，包括從一個 stream 產生一個 relation 的 stream-to-relation 運算、從其它 relations 產生一個 relation 的 relation-to-relation 運算、以及從一個 relation 產生一個 stream 的 relation-to-stream 運算。基本上， CQL 是屬於一般目的的連續查詢語言。

資料流探勘 (data stream mining) [5-6,8,9,11-12]和資料流管理系統(DSMS)是資料庫領域中兩個相關且重要的研究課題。通常在進行資料探勘之前，首先會將資料從資料庫中轉移到 cache，然後再設計一個 procedural program 來對 cache 中的資料作探勘。然而在資料流管理系統中使用這種方式來處理是有困難的，這是因為一般的程式語言無法有效率地處理連續且龐大的資料流資料，除非另外設計特殊的元件來管理 buffers、windows、load shedding 、和 indexing 等問題。Luo et al. [10]提出ESL (Extension of SQL)資料流語言，它是以 SQL 為基礎所發展出來的，可以將資料流探勘所需要用到的相關功能編寫在 ESL 中。這種方式能夠讓資料流探勘直接使用資料流管理系統中的資料，避免了資料複製的問題。

與傳統關聯式資料庫系統一樣，結合(join)運算在資料流管理系統中也是一個非常重要的運算。在許多應用中，我們常常需要結合不同的資料流以計算相關的重要資訊。由於資料流是無限的，而且“結合”運算可能是多對多(many-to-many)的結合型式，因此配合應用的需要與“結合”運算的特性，通常將結合的範圍限制在一個固定大小的視窗中。這種型態的“結合”運算，被稱為“移動視窗結合”(sliding window join)。Das et al.[7] 考慮在資源有限的環境中，對“移動視窗結合”的查詢處理以 approximate query answer 來取代 exact query answer。 為了節省資源，load shedding 的觀念常被使用來將資料流中某些記錄作刪除，以減輕“結合”運算執行時的負擔。

在資料流查詢處理的研究方面，目前相關研究所提出的查詢，主要是針對線上拍賣和流量監控的應用，尚未有一個具體的方法可以在遵守資料流“one-pass”的條件下，有效地處理多個交易資料流的複雜查詢。在本論文中，我們將以論文[2]的“移動視窗模式”(sliding window model)為基礎，並且接續前一個階段的研究成果[3]，考慮多個交易資料流(multiple transactional data streams)與關聯表(relations)的“視窗結合”(window join)運算以及“視窗聚合”(window aggregate)運算。我們發現，傳統資料庫中查詢最佳化的方法不適合完全套用在交易資料流的環境中。因此我們提出三種可能的查詢處理方案，並且探討查詢最佳化的處理方式，以期能夠更有效率地處理交易資料流的查詢。

2. 資料流查詢語言

在本論文中，我們將根據之前所提出的 TSQL(Transactional data Stream Query Language)查詢語言[2]來描述相關的查詢。在論文[2]中，我們以交易資料流為對象，設計了一套適合交易資料流的查詢語言。我們設計的重點包括多資料流與關聯表的“視窗結合”運算以及“視窗聚合”運算。

TSQL 的主要架構包括：

●資料流的設定

OpenStream 資料流名稱

Range 距離現在時間點所要觀察的視窗大小

Slide 連續查詢的時間間隔

Start_time 資料流開始使用的起始時間

End_time 資料流停止讀取的時間

●查詢條件的設定

Select 屬性名稱

From 關聯表名稱或資料流名稱

Where 條件設定

Group by 建立群組的屬性名稱

Having 設定群組的條件

假設交易資料流中的每一筆記錄包含“交易編號” (tid)、“日期” (date)、以及“交易項目集” (<item, quantity>)。另外，假設資料庫中有一個相關的關聯表R(item, location, category, supplier)。下列是一個使用 TSQL 來進行查詢的例子，之後我們也將以這個例子來說明查詢最佳化的處理方式。

範例：查詢過去五天內在DS1和DS2資料流中，交易類別(category)屬於“食品”且區域(location)在“北部”的物品之交易量總和大於或等於10者，列出其 item、supplier、以及交易總量，每隔一天計算一次。假設我們所考慮的交易時間範圍是從 2007/10/28 至2007/11/30為止。以TSQL 描述查詢如下：

OpenStream DS1,DS2
Range 5 days
Slide 1 day
Start_time 2007/10/28 AM 8:00
End_time 2007/11/30 AM 8:00

select item, sum(quantity)
from DS1
group by DS1.item
as ds1(item,quantity)

select item, sum(quantity)
from DS2
group by DS2.item
as ds2(item,quantity)

select ds1.item, ds1.quantity+ds2.quantity
from ds1, ds2
where ds1.item=ds2.item
as ds(item,quantity)

select R.item, R.supplier, quantity
from ds, R
where ds.item=R.item **and** R.location= “北部”**and** R.category = “食品” **and** quantity≧ 10

3. one-pass 演算法

由於本研究所探討的查詢最佳化的方法是以論文[2]的 “one-pass” 演算法為基礎所發展出來的，因此，在這一小節中，我們簡單介紹 “one-pass” 演算法的概念。

假設使用者的查詢視窗分為n個區間，以$w_i(1 \le i \le n)$ 來表示。從圖1可以很清楚地看到，在時間點T_2 的視窗區間中 w_i 所代表的資訊，相當於在時間點T_1 的視窗區間中 w_{i-1} 所代表的資訊。根據連續查詢的定義，在時間點T_2即可將時間點T_1 的視窗區間w_5 所包含的資訊捨去，另外再記錄一個新區間(亦即 T_2的視窗區間w_1)之相關資訊。依此類似下去，即可得到資料流在連續查詢的每一個階段所需要的相關資料，並且執行聚合運算（例如，第2節範例中的 sum(quantity)）。一旦資料流的聚合運算結果得到之後，即可將其以關聯表的形式儲存在資料庫中，我們稱之為 “摘要資訊”，然後再和其它關聯表進行 “結合”(join)運算。

one-pass 演算法之設計主要包含下列兩個步驟：

步驟一：根據資料流查詢的起始時間、Range值、Slide值以及聚合運算的種類，將資料流對應的視窗區間之“摘要資訊”儲存起來。

步驟二：將資料流的“摘要資訊”以關聯表的形式儲存在資料庫中，然後再和其它關聯表進行資料庫的相關運算。

當處理某一個視窗範圍的查詢時，資料流仍持續在進行中。此方法對於接收到的資料流都只需要讀取一次，再根據所對應的視窗區間將摘要資訊儲存起來，為下一階段的連續查詢作準備。

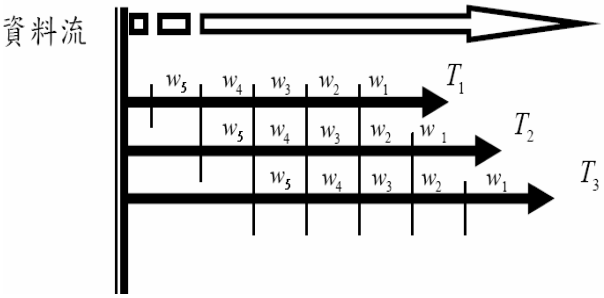

圖1. 資料流移動視窗模式

表1: T1 (2007/11/1)時DS1的資料

視窗	日期	項目集
w_{11}^1	2007/11/1	<C,2>
		<B,2><F,3>
		<A,1><K,2>
w_{12}^1	2007/10/31	<C,1><F,1>
		<A,2><B,3>
w_{13}^1	2007/10/30	<E,2><L,3>
		<B,1><C,1><J,2>
		<K,1><L,1>
w_{14}^1	2007/10/29	<E,2><F,2>
		<C,2><J,1>
		<A,1>
		<F,1><L,2>
w_{15}^1	2007/10/28	<C,1><L,2>
		<J,2>
		<B,3><E,1>

表2: T1 (2007/11/1)時DS2的資料

視窗	日期	項目集
w_{21}^1	2007/11/1	<K,1><G,2>
		<A,1><B,1><C,1>
		<E,2><G,2>
		<A,1><D,2>
w_{22}^1	2007/10/31	<D,2><G,1>
		<D,1><K,1>
		<J,1><K,2>
w_{23}^1	2007/10/30	<A,1><C,2>
		<E,2><J,1>
		<K,1><L,2>
w_{24}^1	2007/10/29	<D,1><E,1>
		<A,2><B,3>
		<G,1><D,1>
w_{25}^1	2007/10/28	<B,2><C,1>
		<G,1><K,1>

表3: T2 (2007/11/2)時DS1的資料

視窗	日期	項目集
w_{11}^2	2007/11/2	<C,1><E,1>
		<A,1><B,2>
		<K,2>
w_{12}^2	2007/11/1	<C,2>
		<B,2><F,3>
		<A,1><K,2>
w_{13}^2	2007/10/31	<C,1><F,1>
		<A,2><B,3>
w_{14}^2	2007/10/30	<E,2><L,3>
		<B,1><C,1><J,2>
		<K,1><L,1>
w_{15}^2	2007/10/29	<E,2><F,2>
		<C,2><J,1>
		<A,1>
		<F,1><L,2>

表4: T2(2007/11/2)時DS2的資料

視窗	日期	項目集
w_{21}^2	2007/11/2	<D,1><G,2><J,3>
		<A,2><K,1>
w_{22}^2	2007/11/1	<K,1><G,2>
		<A,1><B,1><C,1>
		<E,2><G,2>
		<A,1><D,2>
w_{23}^2	2007/10/31	<D,2><G,1>
		<D,1><K,1>
		<J,1><K,2>
w_{24}^2	2007/10/30	<A,1><C,2>
		<E,2><J,1>
		<K,1><L,2>
w_{25}^2	2007/10/29	<D,1><E,1>
		<A,2><B,3>
		<G,1><D,1>

4. 查詢最佳化的處理方式

考慮第2節的範例，對DS1、DS2資料流和關聯表R作查詢，將交易類別(category)屬於“食品”且區域(location)在“北部”的物品之交易量總和大於或等於10 者，列出其item、supplier、以及交易總量，每隔一天計算一次。表1和表2分別是在時間點T1(2007/11/1)時，資料流DS1和DS2的資料。表3和表4分別是在時間點T2(2007/11/2)時，DS1和DS2的資料。以表1為例，2007年11月1日有3筆交易，其中第二筆交易包含項目 “B” 的數量為2，項目 “F” 的數量為3。

我們使用符號w_{ab}^c表示資料流 DSa 在時間點 Tc 時，編號為 b 的視窗區間。根據one-pass 演算法的步驟一，掃描表 1 到表 4 的資料後，即可分別得到表 5 到表 8 的結果。以表 5 為例，在視窗w_{11}^1中 A(1)表示項目“A”的交易數量總和為 1。在時間點 T1，根據表 5 和表 6 即可彙整出時間點 T1 的摘要資訊，如表 9 所示。同樣地，在 T2 時間點，也可以根據表 7 和表 8 彙整出時間點 T2 的摘要資訊，如表 10 所示。

表5: 時間點T1—DS1中各項目數量的總和

視窗	各項目交易的數量總和
w_{11}^1	A(1),B(2),C(2),F(3),K(2)
w_{12}^1	A(2),B(3),C(1),F(1)
w_{13}^1	B(1),C(1),E(2),J(2),K(1),L(4)
w_{14}^1	A(1),C(2),E(2),F(3),J(1),L(2)
w_{15}^1	B(3),C(1),E(1),J(2),L(2)

表6: 時間點T1— DS2中各項目數量的總和

視窗	各項目交易的數量總和
w_{21}^1	A(2),B(1),C(1),D(2),E(2),G(4),K(1)
w_{22}^1	D(3),G(1),J(1),K(3)
w_{23}^1	A(1),C(2),E(2),J(1),K(1),L(2)
w_{24}^1	A(2),B(3),D(2),E(1),G(1)
w_{25}^1	B(2),C(1),G(1),K(1)

表7: 時間點T2—DS1中各項目數量的總和

視窗	各項目交易的數量總和
w_{11}^2	A(1),B(2),C(1),E(1),K(2)
w_{12}^2	A(1),B(2),C(2),F(3),K(2)
w_{13}^2	A(2),B(3),C(1),F(1)
w_{14}^2	B(1),C(1),E(2),J(2),K(1),L(4)
w_{15}^2	A(1),C(2),E(2),F(3),J(1),L(2)

表8: 時間點T2—DS2中各項目數量的總和

視窗	各項目交易的數量總和
w_{21}^2	A(2),D(1),G(2),J(3),K(1)
w_{22}^2	A(2),B(1),C(1),D(2),E(2),G(4),K(1)
w_{23}^2	D(3),G(1),J(1),K(3)
w_{24}^2	A(1),C(2),E(2),J(1),K(1),L(2)
w_{25}^2	A(2),B(3),D(2),E(1),G(1)

表9：時間點T1 —DS1和DS2的摘要資訊

視窗區間數量 / item	DS1 視窗					DS2 視窗				
	w_{11}^1	w_{12}^1	w_{13}^1	w_{14}^1	w_{15}^1	w_{21}^1	w_{22}^1	w_{23}^1	w_{24}^1	w_{25}^1
A	1	2	0	1	0	2	0	1	2	0
B	2	3	1	0	3	1	0	0	3	2
C	2	1	1	2	1	1	0	2	0	1
D	0	0	0	0	0	2	3	0	2	0
E	0	0	2	2	1	2	0	2	1	0
F	3	1	0	3	0	0	0	0	0	0
G	0	0	0	0	0	4	1	0	1	1
J	0	0	2	1	2	0	1	1	0	0
K	2	0	1	0	0	1	3	1	0	1
L	0	0	4	2	2	0	0	2	0	0

表 10：時間點 T2— DS1 和 DS2 的摘要資訊

視窗區間數量 / item	DS1 視窗					DS2 視窗				
	w_{11}^2	w_{12}^2	w_{13}^2	w_{14}^2	w_{15}^2	w_{21}^2	w_{22}^2	w_{23}^2	w_{24}^2	w_{25}^2
A	1	1	2	0	1	2	2	0	1	2
B	2	2	3	1	0	0	1	0	0	3
C	1	2	1	1	2	0	1	0	2	0
D	0	0	0	0	0	1	2	3	0	2
E	1	0	0	2	2	0	2	0	2	1
F	0	3	1	0	3	0	0	0	0	0
G	0	0	0	0	0	2	4	1	0	1
J	0	0	0	2	1	0	0	1	1	0
K	2	2	0	1	0	1	1	3	1	0
L	0	0	0	4	2	0	0	0	2	0

4.1 資料流的前置處理

將T1時間點的DS1和DS2各視窗中的摘要資訊分別轉換成關聯表ds1和ds2。以表9 的項目 “A” 為例，它會對應到關聯表ds1中的五筆記錄，這五筆記錄的內容如表11 所示。對關聯表ds1 執行下列的聚合運算，得到表12的結果，將結果儲存在T1_ds1表格中。

```
select item, sum(quantity)
from ds1
group by ds1.item
as T1_ds1 (item,quantity)
```

類似的作法，將T1時間點的DS2、T2時間點的DS1和DS2各視窗中的摘要資訊轉換後執行聚合運算，得到表13到表15的結果。

表11：T1 時間點項目 “A” 在關聯表ds1中所對應的5筆記錄

視窗區間編號	item	quantity
1	A	1
2	A	2
3	A	0
4	A	1
5	A	0

表12：T1時間點對關聯表ds1執行聚合運算的結果— T1_ds1

item	quantity
A	4
B	9
C	7
D	0
E	5
F	7
G	0
J	5
K	3
L	8

表13：T1時間點對關聯表ds2執行聚合運算的結果—T1_ds2

item	quantity
A	5
B	6
C	4
D	7
E	5
F	0
G	7
J	2
K	6
L	2

表14：T2時間點對關聯表ds1執行聚合運算的結果—T2_ds1

item	quantity
A	5
B	8
C	7
D	0
E	5
F	7
G	0
J	3
K	5
L	6

表15：T2時間點對關聯表ds2執行聚合運算的結果—T2_ds2

item	quantity
A	7
B	4
C	3
D	8
E	5
F	0
G	8
J	5
K	6
L	2

4.2 查詢處理的程序

資料流的前置處理完成後，接著我們考慮交易資料流查詢處理最佳化的方法。我們考慮使用兩種不同的處理方式，第一種處理方式又分為『執行方案 A』、『執行方案 B』，第二種處理方式又稱之為『執行方式 C』。

（一）第一種處理方式：

執行步驟如下：

step1：先將同一個時間點的 2 個資料流關聯表進行合併。

step2：將 step1 的結果跟關聯表 R 進行相關運算，並考慮查詢的最佳化處理。

在 step1，將時間點 T1 的關聯表 T1_ds1(表 12)和關聯表 T1_ds2(表 13)執行結合(join)運算，如圖 2 所示，將結果儲存在 T1_ds_q 表格中，如表 16 所示。類似的作法，將時間點 T2 的關聯表 T2_ds1(表 14)和關聯表 T2_ds2(表 15)執行結合(join)運算，結果儲存在 T2_ds_q 表格中，如表 17 所示。

在 step 2，將表 16 的結果(T1_ds_q)和關聯表 R（表 18）執行後續的相關運算，找出交易量總和大於或等於 10 且交易類別屬於“食品”以及位於“北部”的物品，列出其 item、supplier、以及交易總量。

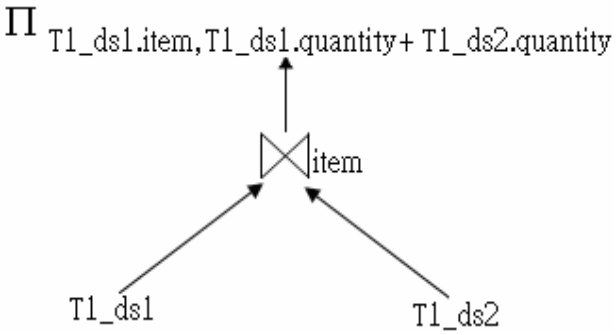

圖 2：第一種處理方式之 step1(在時間點 T1)

表 16：T1_ds_q 表格

item	quantity
A	9
B	15
C	11
D	7
E	10
F	7
G	7
J	7
K	9
L	10

表 17：T2_ds_q 表格

item	quantity
A	12
B	12
C	10
D	8
E	10
F	7
G	8
J	8
K	11
L	8

表18：資料庫中的關聯表 R

item	location	category	supplier
A	北部	食品	統一
B	中部	文具	Hi-tec
C	南部	服飾	小豬兒
D	北部	食品	光泉
E	北部	食品	光泉
F	南部	服飾	小豬兒
G	北部	文具	Hi-tec
H	中部	食品	統一
I	南部	服飾	小豬兒
J	北部	食品	光泉
K	北部	食品	統一
L	南部	文具	Hi-tec
M	北部	食品	統一
N	南部	服飾	小豬兒
O	南部	食品	統一
P	北部	文具	Hi-tec
Q	南部	文具	Hi-tec
R	北部	食品	光泉
S	北部	食品	統一
T	中部	服飾	小豬兒

假設在時間點T1和時間點T2的資料流中，所有可能出現的 item 都已記錄在摘要資訊中，因此在表16和表17中出現的 item 的集合是相同的。我們可以考慮下列兩種不同的執行方案:

『執行方案A』

依據傳統查詢最佳化的方法來處理，先執行 selection 再執行 join ，執行方案A如圖 3 所示。在時間點T1，將表16和表18依執行方案A來處理，得到表19和表20的中間結果以及表21的最後結果。在時間點T2，我們可以使用在T1時的中間結果R1(表19)，以圖4的執行方案來處理，得到表22的中間結果和表23的最後結果。

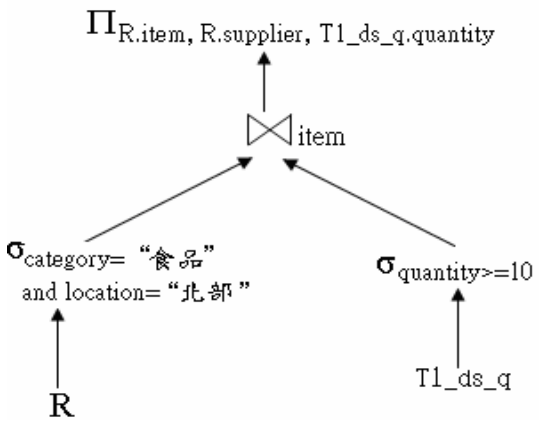

圖 3：執行方案 A（在時間點 T1）

表 19：R1 = σ category= "食品" and location="北部" R

item	location	category	supplier
A	北部	食品	統一
D	北部	食品	光泉
E	北部	食品	光泉
J	北部	食品	光泉
K	北部	食品	統一
M	北部	食品	統一
R	北部	食品	光泉
S	北部	食品	統一

表 20：T1_ds_q1 = $\sigma_{\text{quantity}\geq 10}$ T1_ds_q

item	quantity
B	15
C	11
E	10
L	10

表 21：A_T1_ds = $\Pi_{\text{R.item, R.supplier, T1\_ds\_q1.quantity}}$(R $\bowtie_{\text{item}}$ T1_ds_q1)

item	supplier	quantity
E	光泉	10

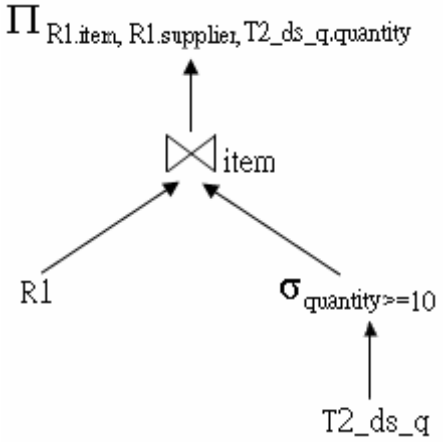

圖 4：執行方案 A（在時間點 T2）

表 22：T2_ds_q1 = $\sigma_{\text{quantity}\geq 10}$ T2_ds_q

item	quantity
A	12
B	12
C	10
E	10
K	11

表 23：A_T2_ds = $\Pi_{\text{R1.item, R1.supplier, T2\_ds\_q1.quantity}}$ (R1 $\bowtie_{\text{item}}$ T2_ds_q1)

item	supplier	quantity
A	統一	12
E	光泉	10
K	統一	11

『執行方案B』

在時間點T1，我們將“quantity≧10”的條件計算延至“join”之後執行，如圖5所示。表24為執行 join 之後的結果。為了對後續在時間T2的查詢作準備，我們從表24選取 R.item 和 R.supplier 兩個欄位，儲存成關聯表R2，如表25所示。接下來對表24執行 selection 和 projection ，即可得到表26的中間結果和表27的最後結果。在時間點T2執行查詢時，我們可以使用關聯表R2來減少執行的時間，如圖6的執行方案。之後的連續查詢也可以用相同的方式來完成。對T2_ds_q (表17)，執行“quantity ≧10”的條件選取後，得到表22的結果。對表22和關聯表R2執行 join 和 projection 之後，即可得到表28的最後結果。

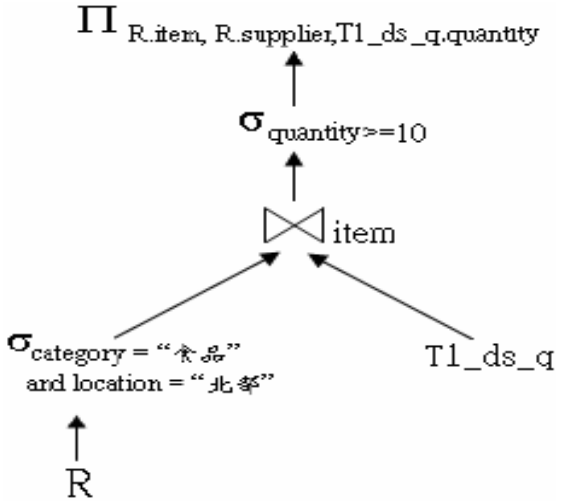

圖 5：執行方案 B（在時間點 T1）

表 24：T1_ds_temp1 = $\sigma_{\text{category=“食品” and location=“北部”}}$ (R $\bowtie_{\text{item}}$ T1_ds_q)

item	location	category	supplier	T1_ds_q.item	quantity
A	北部	食品	統一	A	9
D	北部	食品	光泉	D	7
E	北部	食品	光泉	E	10
J	北部	食品	光泉	J	7
K	北部	食品	統一	K	9

表 25：關聯表 R2

item	supplier
A	統一
D	光泉
E	光泉
J	光泉
K	統一

表 26：T1_ds_temp2 = $\sigma_{\text{quantity}\geq 10}$ T1_ds_temp1

item	location	category	supplier	quantity
E	北	食品	光泉	10

表 27：B_T1_ds = $\Pi_{\text{item, supplier, quantity}}$ (T1_ds_temp2)

item	supplier	quantity
E	光泉	10

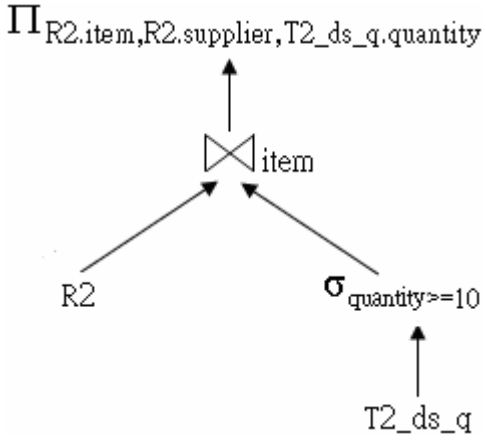

圖 6：執行方案 B（在時間點 T2）

表 28：B_T2_ds = $\Pi_{\text{R2.item, R2.supplier, T2\_ds\_q1.quantity}}$ (R2 $\bowtie_{\text{item}}$ T2_ds_q1)

item	supplier	quantity
A	統一	12
E	光泉	10
K	統一	11

（二）第二種處理方式：

『執行方式 C』

執行步驟如下：

step 1：先將各時間點的資料流關聯表各別跟關聯表 R 進行結合(join)運算。

step 2：再將 step1 的結果執行聚合運算，並且考慮查詢的最佳化處理。

在 step1，將時間點 T1 的關聯表 T1_ds1 (表 12)與關聯表 R (表 18)執行相關運算，如圖 7 所示。$\sigma_{\text{category=“食品” and location=“北部”}}$ R 的執行結果為 R1(如表 19 所示)。將 R1 和 T1_ds1 執行 join 運算，得到表 29 的中間結果，最後再對 T1_ds1_tmep1 執行 projection 運算，得到表 30 的最後結果。類似的作法，將時間點 T1 的關聯表 T1_ds2(表 13)、時間點 T2 的關聯表 T2_ds1(表 14)和關聯表 T2_ds2(表 15)執行類似的運算，結果分別儲存在 T1_ds2_r、T2_ds1_r 和 T2_ds2_r 表格中，(表 31 到表 33)。

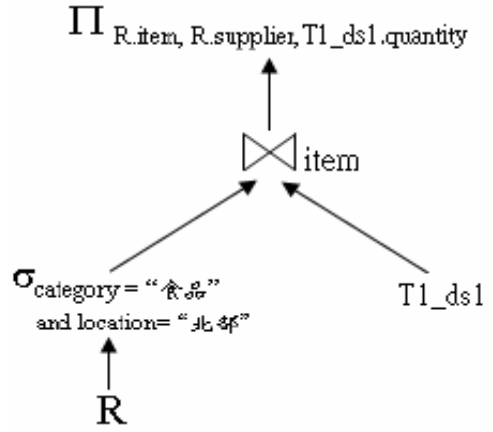

圖 7：執行方案 C 之 step1（在時間點 T1）

表 29：T1_ds1_temp1 = R1 $\bowtie_{\text{item}}$ T1_ds1

item	location	category	supplier	T1_ds1.item	T1_ds1.quantity
A	北部	食品	統一	A	4
D	北部	食品	光泉	D	0
E	北部	食品	光泉	E	5
J	北部	食品	光泉	J	5
K	北部	食品	統一	K	3

表 30：T1_ds1_r = Π item, supplier, quantity (T1_ds1_temp1)

item	supplier	quantity
A	統一	4
D	光泉	0
E	光泉	5
J	光泉	5
K	統一	3

表 31：T1_ ds2_r

item	supplier	quantity
A	統一	5
D	光泉	7
E	光泉	5
J	光泉	2
K	統一	6

表 32：T2_ds1_r

item	supplier	quantity
A	統一	5
D	光泉	0
E	光泉	5
J	光泉	3
K	統一	5

表 33：T2_ ds2_r

item	supplier	quantity
A	統一	7
D	光泉	8
E	光泉	5
J	光泉	5
K	統一	6

在 step 2，根據 step 1 的結果（表 30 到表 33）執行後續的運算，如圖 8 所示，即可找出交易量總和大於或等於 10、交易類別屬於“食品”以及位於“北部”的物品。將時間點 T1 的關聯表 T1_ds1_r(表 30)和 T1_ds2_r(表 31)執行相關運算，得到表 34 的結果。類似的作法，將時間點 T2 的關聯表 T2_ds1_r(表 32)和關聯表 T2_ds2_r(表 33)執行相關運算，得到表 35 的結果。

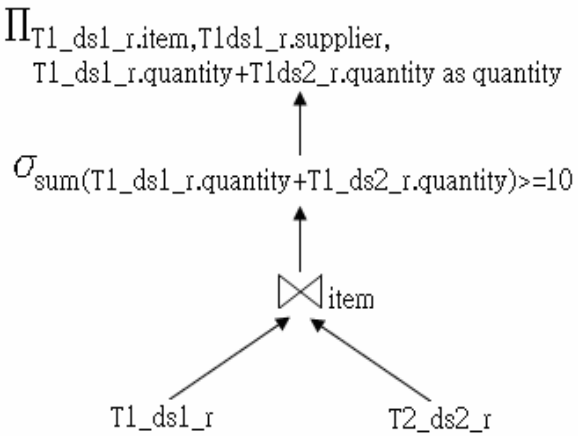

圖 8：執行方案 C 之 step2（在時間點 T1）

表 34：C_T1_ds

item	supplier	quantity
E	光泉	10

表 35：C_T2_ds

item	supplier	quantity
A	統一	12
E	光泉	10
K	統一	11

在上述的兩種處理方式中，我們可以發現執行方案C比執行方案A和執行方案B更為複雜繁瑣且執行的 join 次數也比較多，因此第一種處理方式較第二種處理方式為佳。而執行方案B中R2的資料量比執行方案A中R1的資料量少，因此執行方案B會比執行方案A更有效率。所以，以執行效率的高低來比較，執行方案B＞執行方案A＞執行方案C。

5. 結論

由於傳統處理資料流的“結合”(join)運算時，是先將原始資料流的資料全部轉換成關聯表後，再利用資料庫的 join 運算做結合，因此需要耗費較多的記憶體，而且也會造成重複讀取資料的情況。在本論文的研究結果中發現，傳統資料庫的查詢最佳化的方法不適合完全套用在交易資料流的環境中。目前我們研究查詢最佳化的處理方式已經得到一些重要的成果，未來我們將持續研究，提出一套完整的資料流查詢最佳化的解決方案，讓資料流查詢系統的架構更為完整。

誌謝

本論文之研究獲得國科會專題研究計畫NSC 95-2221-E-159-017 的補助。

參考文獻

[1] 蔡秀滿,陳耀民,“在加權移動視窗模式中快速探勘資料流之研究”,***第十二屆資訊管理暨實務研討會***，台灣，2006。

[2] 蔡秀滿,彭雅筠,“移動視窗模式中資料流查詢處理之研究”,***電子商務與數位生活研討會***，2007。

[3] 蔡秀滿,彭雅筠,“在加權移動視窗模式中的資料流查詢最佳化”，***第十八屆國際資訊管理學術研討會***，台灣，2007。

[4] A. Arasu, S. Babu, and J. Widom, “CQL: A Language for Continuous Queries over Streams and Relations”, ***Proceedings of the International Workshop on Database Programming Languages***, pp.1-11, 2003.

[5] J.H. Chang and W.S. Lee, “Finding Recent Frequent Itemsets Adaptively over Online Data Streams”, ***Proceedings of ACM SIGKDD***, pp. 487-492, 2003.

[6] G. Cormode, F. Korn, S. Muthukrishnan and D. Srivastava, “Finding Hierarchical Heavy Hitters in Data Streams”, ***Proceedings of the VLDB Conference***, 2003.

[7] A. Das, J. Gehrke and M. Riedewald, ”Semantic Approximation of Data Stream Joins”, ***IEEE Transactions on Knowledge and Data Engineering***, 17 (1), pp. 44-59, 2005.

[8] P. Domingos and G. Hulten, “Mining High-Speed Data Streams”, ***Proceedings of ACM SIGKDD***, pp. 71-80, 2000.

[9] C. Jin, W. Qian, C. Sha, J.X. Yu and A. Zhou, “Dynamically Maintaining Frequent Items Over a Data Stream”, ***Proceedings of the Information and Knowledge Management (CIKM)***, 2003.

[10] C. Luo, H. Thakkar, H. Wang, and C. Zaniolo, “A Native Extension of SQL for Mining Data Streams”, ***Proceedings of ACM SIGMOD***, pp. 873-875, 2005.

[11] A. Manjhi, V. Shkapenyuk and K. Dhamdhere. “Finding (Recently) Frequent Items in Distributed Data Streams”, ***Proceedings of the International Conference on Data Engineering***, pp. 767-778, 2005.

[12] G. Manku and R. Motwani, “Approximate Frequency Counts over Data Streams”, ***Proceedings of the VLDB Conference***, pp. 346-357, 2002.

[13] D. Terry, D. Goldberg, D. Nichols, and B. Oki, “Continuous Queries over Append-Only Databases”, ***Proceedings of ACM SIGMOD***, pp. 321-330, 1992.

[14] Y. Zhu and D. Shasha, “StartStream: Statistical Monitoring of Thousands of Data Streams in Real Time”, ***Proceedings of the VLDB Conference***, pp. 358-369, 2002.