Database Assignment2 Report

104062203 陳涵宇
104062232 廖子毅
104062234 林士軒

How to Implement the txns by JDBC and stored procedures

我們先觀察整體 readItem 是如何實作，並以此為基底，增加這次所需要的功能：updatePrice 的兩種版本 (JDBC, SP)。

主要部分改動：

- JDBC

  我們修改及增建以下的 class：

  1. 增加一個 As2UpdatePriceJob, As2UpdatePriceParamGen class，來執行 selectSQL 及 updateSQL 。此外，將 selectSQL 執行完的結果，放進 outputMsg 中，用以確認資料是否有誤。
  2. 增建 As2SchemeBuilderJob, As2TestbedLoaderJob 兩個 class，用來產生資料庫中的 table 及 gen 出 NUM_ITEMS 筆資料，匯入資料庫中。

- SP

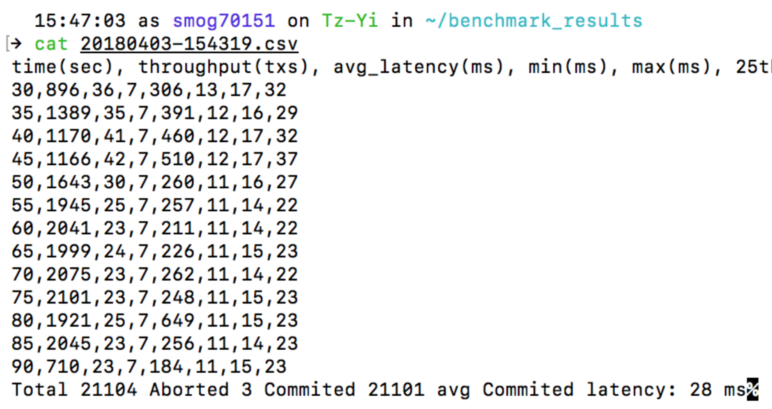
  1. 將原本的 As2RTE sample code 更動成會因為 w/ ratio 來選擇要執行何種程序，並增添相對應的變數。(Read -> Update)
  2. 新建 As2ParaHelper, As2UpdatePriceProc 等相關 class，用以獲取更新時所需的參數，及執行 select ＆ Update SQL。
  3. 修改 As2StoredProcFactory class，使得他可以藉由我們定義的 UPDATE_ITEM 的 pid，進到新建的 class As2UpdatePriceProc。

其餘參數修改：

1. 新增一個 property WRITE_TXN_RATE，它的型別為 int，default 值為 5 (範圍 在 0 ~ 10 對應到 0% ~ 100%)。
2. 新增一個的 txn type：UPDATE_ITEM，用以應對更新的功能。
3. 在 As2Rte 這個 class 中新增一個 int random()，以此判斷要 readItem 或 updatePrice，來完成 w/ ratio 的實作。
4. 將 staticsMgr 做修改，將原先的 3000ms 改為 5000ms 做一次 trace ，此外，將輸出格式從 txt 轉為 csv 檔，並將內容改為 spec 所要求樣式 (e.g. time(sec), throughput(txs), ...)。

Screenshot of csv report

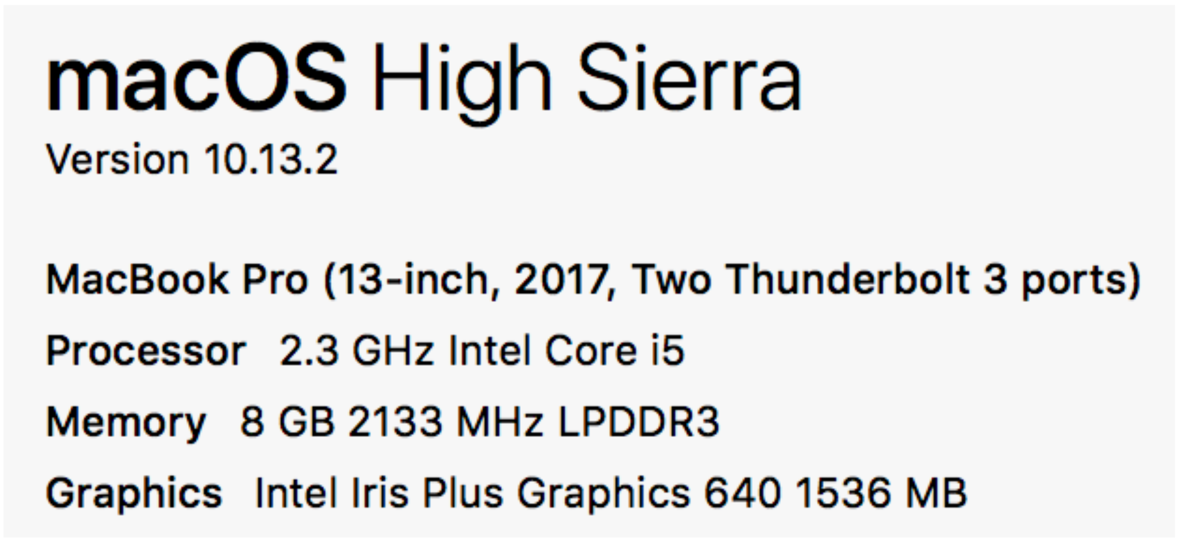
// WRITE_TXN_RATE = 5 (it means 50% write tx rate)



```
  15:47:03 as smog70151 on Tz-Yi in ~/benchmark_results
→ cat 20180403-154319.csv
time(sec), throughput(txs), avg_latency(ms), min(ms), max(ms), 25th_lat(ms), median_lat(ms), 75th_lat(ms)
30,896,36,7,306,13,17,32
35,1389,35,7,391,12,16,29
40,1170,41,7,460,12,17,32
45,1166,42,7,510,12,17,37
50,1643,30,7,260,11,16,27
55,1945,25,7,257,11,14,22
60,2041,23,7,211,11,14,22
65,1999,24,7,226,11,15,23
70,2075,23,7,262,11,14,22
75,2101,23,7,248,11,15,23
80,1921,25,7,649,11,15,23
85,2045,23,7,256,11,14,23
90,710,23,7,184,11,15,23
Total 21104 Aborted 3 Commited 21101 avg Commited latency: 28 ms%
```

Experiments

1. env



macOS High Sierra

Version 10.13.2

MacBook Pro (13-inch, 2017, Two Thunderbolt 3 ports)

Processor 2.3 GHz Intel Core i5

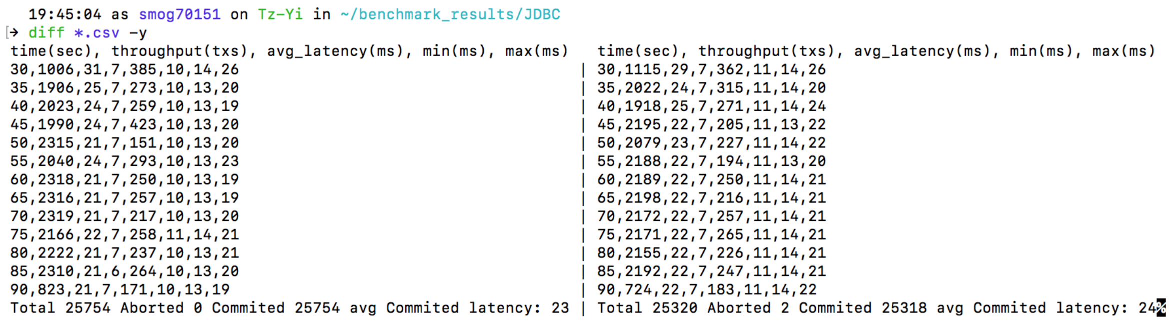
Memory 8 GB 2133 MHz LPDDR3

Graphics Intel Iris Plus Graphics 640 1536 MB

2. performances & analysis

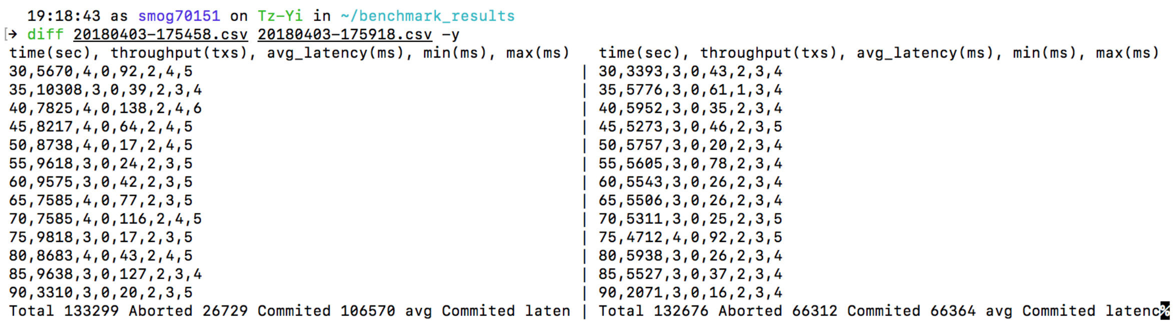
實驗八種情況：在兩種 connection mode 下，分別將 WRITE_TXN_RATE 調為 0 / 2 / 5 / 10 四個數字。

以下先以 WRITE_TXN_RATE=2 及 WRITE_TXN_RATE=5 的狀況下比較：



```
  19:45:04 as smog70151 on Tz-Yi in ~/benchmark_results/JDBC
→ diff *.csv -y
time(sec), throughput(txs), avg_latency(ms), min(ms), max(ms)   time(sec), throughput(txs), avg_latency(ms), min(ms), max(ms)
30,1006,31,7,385,10,14,26                                     | 30,1115,29,7,362,11,14,26
35,1906,25,7,273,10,13,20                                     | 35,2022,24,7,315,11,14,20
40,2023,24,7,259,10,13,19                                     | 40,1918,25,7,271,11,14,24
45,1990,24,7,423,10,13,20                                     | 45,2195,22,7,205,11,13,22
50,2315,21,7,151,10,13,20                                     | 50,2079,23,7,227,11,14,22
55,2040,24,7,293,10,13,23                                     | 55,2188,22,7,194,11,13,20
60,2318,21,7,250,10,13,19                                     | 60,2189,22,7,250,11,14,21
65,2316,21,7,257,10,13,19                                     | 65,2198,22,7,216,11,14,21
70,2319,21,7,217,10,13,20                                     | 70,2172,22,7,257,11,14,21
75,2166,22,7,258,11,14,21                                     | 75,2171,22,7,265,11,14,21
80,2222,21,7,237,10,13,21                                     | 80,2155,22,7,226,11,14,21
85,2310,21,6,264,10,13,20                                     | 85,2192,22,7,247,11,14,21
90,823,21,7,171,10,13,19                                      | 90,724,22,7,183,11,14,22
Total 25754 Aborted 0 Commited 25754 avg Commited latency: 23 | Total 25320 Aborted 2 Commited 25318 avg Commited latency: 24%
```

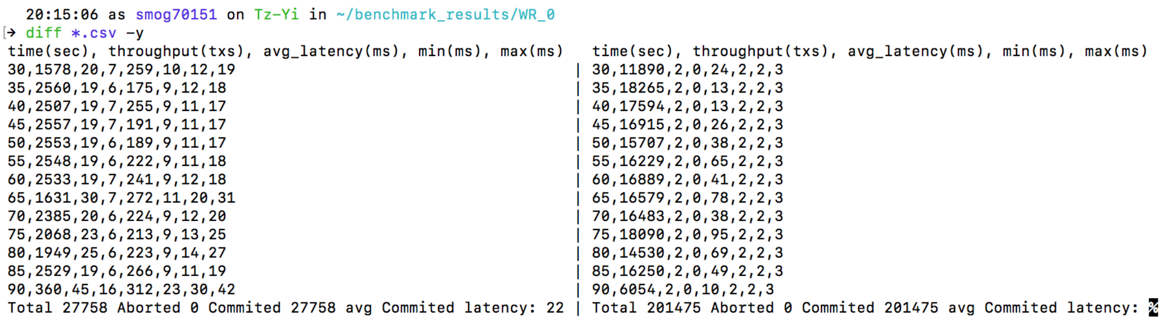
首先上圖為比較 JDBC 在不同的 WRITE_TXN_RATE，我們發現在不同的值下，其實 JDBC 的 throughput 不會差異很大。主因是 JDBC 中的 bottleneck 是在 txn 的 latency。也就是說，讀取與寫入的比例，在 JDBC 其實不會影響很多，主要影響的是每個交易的處理時間。



```
  19:18:43 as smog70151 on Tz-Yi in ~/benchmark_results
[→ diff 20180403-175458.csv 20180403-175918.csv -y
time(sec), throughput(txs), avg_latency(ms), min(ms), max(ms)   time(sec), throughput(txs), avg_latency(ms), min(ms), max(ms)
30,5670,4,0,92,2,4,5                                          | 30,3393,3,0,43,2,3,4
35,10308,3,0,39,2,3,4                                         | 35,5776,3,0,61,1,3,4
40,7825,4,0,138,2,4,6                                         | 40,5952,3,0,35,2,3,4
45,8217,4,0,64,2,4,5                                          | 45,5273,3,0,46,2,3,5
50,8738,4,0,17,2,4,5                                          | 50,5757,3,0,20,2,3,4
55,9618,3,0,24,2,3,5                                          | 55,5605,3,0,78,2,3,4
60,9575,3,0,42,2,3,5                                          | 60,5543,3,0,26,2,3,4
65,7585,4,0,77,2,3,5                                          | 65,5506,3,0,26,2,3,4
70,7585,4,0,116,2,4,5                                         | 70,5311,3,0,25,2,3,5
75,9818,3,0,17,2,3,5                                          | 75,4712,4,0,92,2,3,5
80,8683,4,0,43,2,4,5                                          | 80,5938,3,0,26,2,3,4
85,9638,3,0,127,2,3,4                                         | 85,5527,3,0,37,2,3,4
90,3310,3,0,20,2,3,5                                          | 90,2071,3,0,16,2,3,4
Total 133299 Aborted 26729 Commited 106570 avg Commited laten | Total 132676 Aborted 66312 Commited 66364 avg Commited latenc%
```

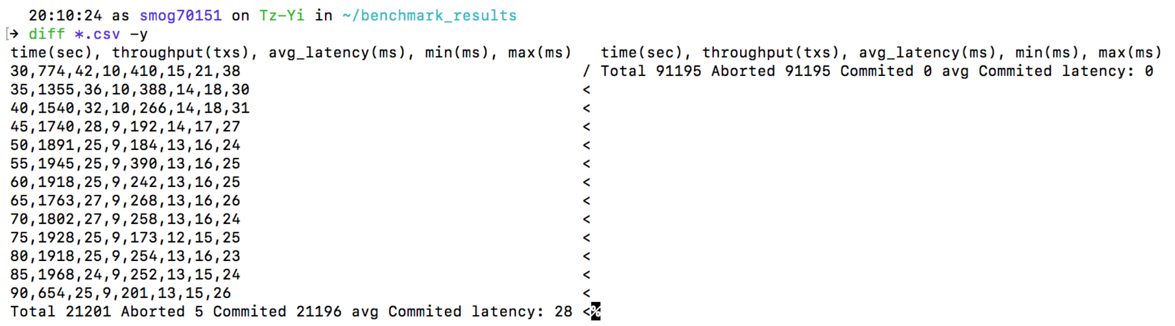
其次，比較 SP 在不同的 WRITE_TXN_RATE 下的 throughput ， 我們發現在 WRITE_TXN_RATE 有變動時，throughput 也會有明顯的變動。這樣的結果顯示，WRITE_TXN_RATE 與 throughput 應有相關性，而原因則是因為在 SP 中，我們可以直接執行 SQL，也因此，我們的 throughput 會因單位時間可寫入率的變化有相對應的改變。

下面比較 WRITE_TXN_RATE=0 的極端狀況：(左為 JDBC | 右為 SP)



```
  20:15:06 as smog70151 on Tz-Yi in ~/benchmark_results/WR_0
[→ diff *.csv -y
time(sec), throughput(txs), avg_latency(ms), min(ms), max(ms)   time(sec), throughput(txs), avg_latency(ms), min(ms), max(ms)
30,1578,20,7,259,10,12,19                                     | 30,11890,2,0,24,2,2,3
35,2560,19,6,175,9,12,18                                      | 35,18265,2,0,13,2,2,3
40,2507,19,7,255,9,11,17                                      | 40,17594,2,0,13,2,2,3
45,2557,19,7,191,9,11,17                                      | 45,16915,2,0,26,2,2,3
50,2553,19,6,189,9,11,17                                      | 50,15707,2,0,38,2,2,3
55,2548,19,6,222,9,11,18                                      | 55,16229,2,0,65,2,2,3
60,2533,19,7,241,9,12,18                                      | 60,16889,2,0,41,2,2,3
65,1631,30,7,272,11,20,31                                     | 65,16579,2,0,78,2,2,3
70,2385,20,6,224,9,12,20                                      | 70,16483,2,0,38,2,2,3
75,2068,23,6,213,9,13,25                                      | 75,18090,2,0,95,2,2,3
80,1949,25,6,223,9,14,27                                      | 80,14530,2,0,69,2,2,3
85,2529,19,6,266,9,11,19                                      | 85,16250,2,0,49,2,2,3
90,360,45,16,312,23,30,42                                     | 90,6054,2,0,10,2,2,3
Total 27758 Aborted 0 Commited 27758 avg Commited latency: 22 | Total 201475 Aborted 0 Commited 201475 avg Commited latency: %
```

和比較 WRITE_TXN_RATE=10 的極端狀況：(左為 JDBC | 右為 SP)



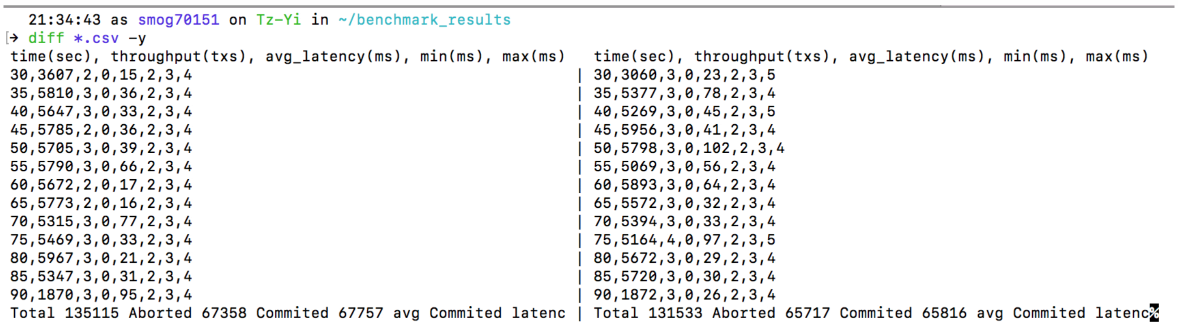
```
  20:10:24 as smog70151 on Tz-Yi in ~/benchmark_results
[→ diff *.csv -y
time(sec), throughput(txs), avg_latency(ms), min(ms), max(ms)   time(sec), throughput(txs), avg_latency(ms), min(ms), max(ms)
30,774,42,10,410,15,21,38                                     / Total 91195 Aborted 91195 Commited 0 avg Commited latency: 0
35,1355,36,10,388,14,18,30                                    <
40,1540,32,10,266,14,18,31                                    <
45,1740,28,9,192,14,17,27                                     <
50,1891,25,9,184,13,16,24                                     <
55,1945,25,9,390,13,16,25                                     <
60,1918,25,9,242,13,16,25                                     <
65,1763,27,9,268,13,16,26                                     <
70,1802,27,9,258,13,16,24                                     <
75,1928,25,9,173,12,15,25                                     <
80,1918,25,9,254,13,16,23                                     <
85,1968,24,9,252,13,15,24                                     <
90,654,25,9,201,13,15,26                                      <
Total 21201 Aborted 5 Commited 21196 avg Commited latency: 28 <%
```

我們可以從上面極端的 Write ratio 證實，JDBC 的 throughput bottleneck 的確不會卡在 w/ ratio，而是在 latency 上。而 SP 的 bottleneck 的確是 w/ ratio。

也就是說，若我們想要增進效能的話，JDBC 的部分需要針對 latency，SP 則為 w/ ratio。

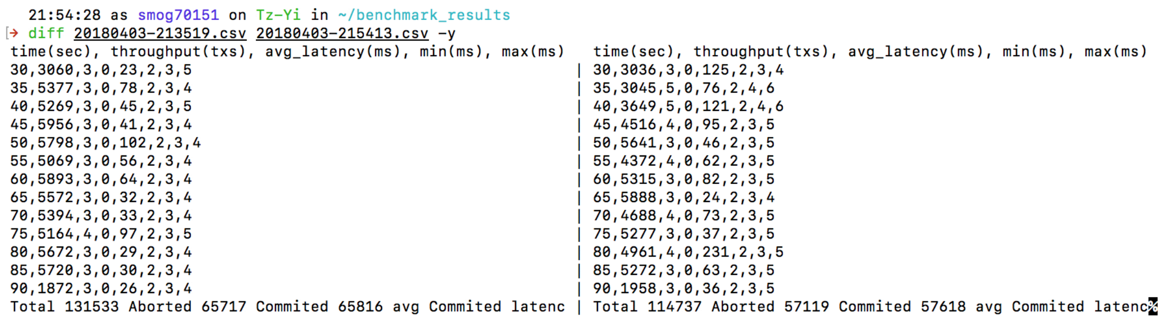
實驗：調整 NUM_ITEMS 下 throughput 狀況。

比較 NUM_ITEMS=1,000 和 NUM_ITEMS=100,000：



```
  21:34:43 as smog70151 on Tz-Yi in ~/benchmark_results
|→ diff *.csv -y
time(sec), throughput(txs), avg_latency(ms), min(ms), max(ms)   time(sec), throughput(txs), avg_latency(ms), min(ms), max(ms)
30,3607,2,0,15,2,3,4                                          | 30,3060,3,0,23,2,3,5
35,5810,3,0,36,2,3,4                                          | 35,5377,3,0,78,2,3,4
40,5647,3,0,33,2,3,4                                          | 40,5269,3,0,45,2,3,5
45,5785,2,0,36,2,3,4                                          | 45,5956,3,0,41,2,3,4
50,5705,3,0,39,2,3,4                                          | 50,5798,3,0,102,2,3,4
55,5790,3,0,66,2,3,4                                          | 55,5069,3,0,56,2,3,4
60,5672,2,0,17,2,3,4                                          | 60,5893,3,0,64,2,3,4
65,5773,2,0,16,2,3,4                                          | 65,5572,3,0,32,2,3,4
70,5315,3,0,77,2,3,4                                          | 70,5394,3,0,33,2,3,4
75,5469,3,0,33,2,3,4                                          | 75,5164,4,0,97,2,3,5
80,5967,3,0,21,2,3,4                                          | 80,5672,3,0,29,2,3,4
85,5347,3,0,31,2,3,4                                          | 85,5720,3,0,30,2,3,4
90,1870,3,0,95,2,3,4                                          | 90,1872,3,0,26,2,3,4
Total 135115 Aborted 67358 Commited 67757 avg Commited latenc | Total 131533 Aborted 65717 Commited 65816 avg Commited latenc%
```

比較 NUM_ITEMS=100,000 和 NUM_ITEMS=1,000,000：



```
  21:54:28 as smog70151 on Tz-Yi in ~/benchmark_results
|→ diff 20180403-213519.csv 20180403-215413.csv -y
time(sec), throughput(txs), avg_latency(ms), min(ms), max(ms)   time(sec), throughput(txs), avg_latency(ms), min(ms), max(ms)
30,3060,3,0,23,2,3,5                                          | 30,3036,3,0,125,2,3,4
35,5377,3,0,78,2,3,4                                          | 35,3045,5,0,76,2,4,6
40,5269,3,0,45,2,3,5                                          | 40,3649,5,0,121,2,4,6
45,5956,3,0,41,2,3,4                                          | 45,4516,4,0,95,2,3,5
50,5798,3,0,102,2,3,4                                         | 50,5641,3,0,46,2,3,5
55,5069,3,0,56,2,3,4                                          | 55,4372,4,0,62,2,3,5
60,5893,3,0,64,2,3,4                                          | 60,5315,3,0,82,2,3,5
65,5572,3,0,32,2,3,4                                          | 65,5888,3,0,24,2,3,4
70,5394,3,0,33,2,3,4                                          | 70,4688,4,0,73,2,3,5
75,5164,4,0,97,2,3,5                                          | 75,5277,3,0,37,2,3,5
80,5672,3,0,29,2,3,4                                          | 80,4961,4,0,231,2,3,5
85,5720,3,0,30,2,3,4                                          | 85,5272,3,0,63,2,3,5
90,1872,3,0,26,2,3,4                                          | 90,1958,3,0,36,2,3,5
Total 131533 Aborted 65717 Commited 65816 avg Commited latenc | Total 114737 Aborted 57119 Commited 57618 avg Commited latenc%
```

由實驗來看，NUM_ITEMS 其實不太影響 throughput。推測原因有二：

> 1. NUM_ITEMS 數量與 throughput 無正相關。
> 2. 在 scale 很大 (e.g. 實驗測資：1,000,000)的時候，vanillaDB 有採取別種演算法，來增加效率。

Anything worth to Mentioned

`executeQuery(sql);` 和 `executeUpdate(sql);`，一開始在寫 JDBC 的時候，不小心搞混了這兩個 function，後來查詢及試驗了之後的結果，才發現，`executeQuery(sql);` 是使用在不會影響到 table 的時候 (e.g. SELECT)，而 `executeUpdate(sql);` 則是使用在更改、更新 table 的時候 (e.g. CREATE, INSERT, UPDATE, DELETE, ...)。