# Assignment 2 Phase 2 Report

110062229 翁語辰
110081014 程詩柔
110062171 陳彥成

1. UpdatePriceParamGen.java

**TA:** 將itemid,raise包成一個物件，一起放入linklist中，最後再轉換成arrary 回傳

```
paramList.add(WRITE_COUNT);

for (int i = 0; i < WRITE_COUNT; i++) {
    int itemId = rvg.number(min:1, As2BenchConstants.NUM_ITEMS);
    double raise = ((double) rvg.number(min:0, MAX_RAISE)) / 10;

    paramList.add(new UpdateItemPriceTxnParam(itemId, raise));
}

return paramList.toArray();
```

我們**:** 將itemid,raise分別放入linklist中

```
@Override
public Object[] generateParameter() {
    RandomValueGenerator rvg = new RandomValueGenerator();
    ArrayList<Object> paramList = new ArrayList<Object>();

    // Set update count
    paramList.add(TOTAL_UPDATE_COUNT);

    // Generate item IDs and price raises
    for (int i = 0; i < TOTAL_UPDATE_COUNT; i++) {
        // Generate a random item ID to update
        paramList.add(rvg.number(min:1, As2BenchConstants.NUM_ITEMS));

        // Generate a random price raise value between 0.0 and 5.0
        paramList.add(rvg.fixedDecimalNumber(decimal:1, min:0.0, max:5.0));
    }

    return paramList.toArray(new Object[0]);
```

比較**:** TA的方式比較好，打包成一個物件可以使最終在拿出資料來用時不易出錯。

2. 取出random後的itemid和raise

**TA:** 直接在UpdateItemPriceTxnJdbcJob中assign更新值給itemid[]和raises[]

```
@Override
public SutResultSet execute(Connection conn, Object[] pars) throws SQLException {
    // Parse parameters
    int readCount = (Integer) pars[0];
    int[] itemIds = new int[readCount];
    double[] raises = new double[readCount];

    for (int i = 0; i < readCount; i++) {
        itemIds[i] = (Integer) (((UpdateItemPriceTxnParam) pars[i + 1]).itemId);
        raises[i] = (Double) (((UpdateItemPriceTxnParam) pars[i + 1]).raise);
    }
```

我們: 在UpdatePriceProcParamHelper先做資料處理再回傳給JdbcJob, 相關參數如updateCount等等也是從JdbcJob呼叫ParamHelper取得

```
@Override
public void prepareParameters(Object... pars) {
    int indexCnt = 0;

    updateCount = (Integer) pars[indexCnt++];
    updateItemId = new int[updateCount];
    priceRaise = new double[updateCount];

    for (int i = 0; i < updateCount; i++) {
        updateItemId[i] = (Integer) pars[indexCnt++];
        priceRaise[i] = (Double) pars[indexCnt++];
    }
}
```

```
for (int i = 0; i < paramHelper.getUpdateCount(); i++) {
    int iid = paramHelper.getUpdateItemId(i);
    double raise = paramHelper.getPriceRaise(i);
```

比較: TA的方式比較好, 直接在JdbcJob做資料處理比較簡潔, 相較我的寫法, 可以節省很多呼叫function call的時間

3. UpdateItemPriceTxnProc

**TA:** 呼叫newPlanner 來執行query

```
int iid = paramHelper.getItemId(idx);
Plan p = VanillaDb.newPlanner().createQueryPlan("SELECT i_name, i_price FROM item
Scan s = p.open();
```

我們: 呼叫executeQuery(),executeUpdate(), 因為他本身就含newplanner()

```
int iid = paramHelper.getUpdateItemId(idx);
double raise = paramHelper.getPriceRaise(idx);
// Select the current price
Scan s = StoredProcedureHelper.executeQuery(
    "SELECT i_price FROM item WHERE i_id = " + iid,
    tx
);
```

比較: 我們的寫法相較之下比較簡潔, 不過或許TA直接呼叫newplanner()可以節省些許時間

4. getNextTxType() in As2BenchmarkRte

**TA:** 調用RandomValueGenerator()

```
protected As2BenchTransactionType getNextTxType() {
        RandomValueGenerator rvg = new RandomValueGenerator();

        // flag would be 100 if READ_WRITE_TX_RATE is 1.0
        int flag = (int) (As2BenchConstants.READ_WRITE_TX_RATE * precision);

        if (rvg.number(0, precision - 1) < flag) {
                return As2BenchTransactionType.READ_ITEM;
        } else {
                return As2BenchTransactionType.UPDATE_ITEM_PRICE;
        }

}
```

我們: 直接用random()

```
protected As2BenchTransactionType getNextTxType() {        TwPanTsu, 8 months ago
    // return As2BenchTransactionType.UPDATE_PRICE;
    Random random = new Random();
    // System.out.println("random:"+random+", rate:"+VanillaBenchParameters.READ_
    return random.nextDouble() > VanillaBenchParameters.READ_WRITE_TX_RATE ?
    As2BenchTransactionType.READ_ITEM:As2BenchTransactionType.UPDATE_PRICE;
}
```

比較: 我們的寫法雖然簡單直接, 但沒有利用utils中的RandomValueGenerator, 在拓展性和維護上就會因為少了RandomValueGenerator提供的功能而受到限制。應該有工具就要好好使用。

5. READ_WRITE_TX_RATE

**TA:** 將這個常數設在As2BenchConstants。

```
public class As2BenchConstants {

        public static final int NUM_ITEMS;
        public static final double READ_WRITE_TX_RATE;

        static {
                NUM_ITEMS = BenchProperties.getLoader().getPrope
```

我們: 設在VanillaBenchParameters。

```
public static final long BENCHMARK_INTERVAL;
public static final int NUM_RTES;
public static final long RTE_SLEEP_TIME;
public static final double READ_WRITE_TX_RATE;
public static final String SERVER_IP;
```

比較: 助教的寫法是將READ_WRITE_TX_RATE, 作為更新操作會用的常數設置的, 符合實際功能, 較為合理, 利於擴展性和系統結構。但我們考慮READ_WRITE_TX_RATE是一個不斷調整測試的常數, 因為主要作業就是測試更新功能, 所以放在VanillaBenchParameters一起設置, 比較便於測試和維護。

6. StatisticMgr

**TA:** 新增一個latencyHistory的TreeMap

```
private TreeMap<Long, ArrayList<Long>> latencyHistory = new TreeMap<Long, ArrayList<Long>>();
```

**TA:** 每個resultset都去利用addTxnLatency把該次txn的respondtime放到正確的ArrayList裡

```
private void addTxnLatency(TxnResultSet rs) {
    long elapsedTime = TimeUnit.NANOSECONDS.toSeconds(rs.getTxnEndTime() - recordStartTime);
    long timeSlotBoundary = (elapsedTime / timelineGranularity) * timelineGranularity ;


    ArrayList<Long> timeSlot = latencyHistory.get(timeSlotBoundary );
    if (timeSlot == null) {
        timeSlot = new ArrayList<Long>();
        latencyHistory.put(timeSlotBoundary, timeSlot);
    }
    timeSlot.add(TimeUnit.NANOSECONDS.toMillis(rs.getTxnResponseTime()));
}
```

**TA:** outputAs2Report的時候就有點像是去iterate through所有latencyHistory中的ArrayList然後利用makeStatString+calcMedian+calcMean來print正確的資訊

```
private void outputAs2Report(String fileName) throws IOException {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(new File(outputDir, fileName + ".csv"))))
        writer.write(
                "time(sec), throughput(txs), avg_latency(ms), min(ms), max(ms), 25th_lat(ms), median_lat(ms),
        writer.newLine();

        for (long timeBound = 0, outCount = 0; outCount < latencyHistory.size(); timeBound += timelineGranula
            List<Long> slot = latencyHistory.get(timeBound);
            if (slot != null) {
                writer.write(makeStatString(timeBound, slot));
                outCount++;
            } else
                writer.write(String.format("%d, 0, NaN, NaN, NaN, NaN, NaN, NaN", timeBound));
            writer.newLine();
        }
    }
}
```

我們**:** 直接iterate through整個resultSets, 利用每個resultSet的txnEndTime來判斷現在這個txn是不是還在同一個timeslot中, 還在的話就把responsetime加入ArrayList中, 不在的話就用目前ArrayList裡的資料去計算前一個timeslot的所有統計數據並print出來然後重置arrayList

```
for (TxnResultSet resultSet : resultSets) {
    currentTime = resultSet.getTxnEndTime();
    if (currentTime - lastReportTime > TimeUnit.MILLISECONDS
            .toNanos(VanillaBenchParameters.CSV_REPORT_INTERVAL)) {
        recordTimes++;
        List<Long> stats = genStatisticFromList(respTimes);
        writer.write(String.format("%d,%d,%d,%d,%d,%d,%d,%d",
                TimeUnit.MILLISECONDS.toSeconds(VanillaBenchParameters.WARM_UP_INTERVAL)
                        + recordTimes * TimeUnit.MILLISECONDS
                                .toSeconds(VanillaBenchParameters.CSV_REPORT_INTERVAL),
                stats.get(0), TimeUnit.NANOSECONDS.toMillis(stats.get(1)),
                TimeUnit.NANOSECONDS.toMillis(stats.get(2)), TimeUnit.NANOSECONDS.toMillis(stats.get(3)),
                TimeUnit.NANOSECONDS.toMillis(stats.get(4)), TimeUnit.NANOSECONDS.toMillis(stats.get(5)),
                TimeUnit.NANOSECONDS.toMillis(stats.get(6))));
        writer.newLine();
        lastReportTime = lastReportTime
                + TimeUnit.MILLISECONDS.toNanos(VanillaBenchParameters.CSV_REPORT_INTERVAL);
    } else
        respTimes.add(resultSet.getTxnResponseTime());
}
```

比較**:** 兩者的實作方式基本上一樣, 硬要說的話TA的會好一點因為少iterate through一次resultSets, TA在outputDetailReport的時候順便去計算每次txn在哪個timeslot中, 我們是另外去iterate一次才去計算