Assignment 2 Common Mistakes

Introduction to Databases, 2019 Spring

Datalab, NTHU

Common Mistake 1

- Generate price raise at server side

```
double price = 0.0;

if (s.next()) {
    price = (Double) s.getVal("i_price").asJavaVal();
} else {
    throw new RuntimeException("Cloud not find item record with i_id = " + iid);
}

// TODO Generate random price raise
double price_raise = rvg.fixedDecimalNumber(2, 0.0, 5.0);

// TODO Decide the final price raise
if (price + price_raise >= As2BenchConstants.MAX_PRICE) {
    price = As2BenchConstants.MIN_PRICE;
} else {
    price += price_raise;
}

// Round the price to 2 decimals
BigDecimal bd = new BigDecimal(price);
bd = bd.setScale(2, RoundingMode.HALF_UP);
price = bd.doubleValue();

// TODO Update the price
VanillaDb.newPlanner().executeUpdate(
        "UPDATE item SET i_price = " + price + " WHERE i_id = " + iid, tx);
```

UpdateItemPriceTxnProc.java

UpdateItemPriceParamGen.java

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

    paramList.add(UPDATE_COUNT);
    for (int i = 0; i < UPDATE_COUNT; i++)
        paramList.add(rvg.number(1, As2BenchConstants.NUM_ITEMS));
    for (int i = 0; i < UPDATE_COUNT; i++)
        paramList.add(rvg.fixedDecimalNumber(2, 0.0, 5.0));

    return paramList.toArray();
}
```

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

    updateCount = (Integer) pars[indexCnt++];
    updateItemId = new int[updateCount];
    updateItemPrice = new double[updateCount];
    itemName = new String[updateCount];
    itemPrice = new double[updateCount];

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

UpdateItemPriceProcParamHelper.java

Common Mistake 1

- Generate price raise at server side
- Correct: Generate price raise in client side

Common Mistake 2

- Determine the time interval based on accumulated transaction response time

```
for (TxnResultSet resultSet : resultSets) {
    responseTime = TimeUnit.NANOSECONDS.toMillis(resultSet.getTxnResponseTime());

    if(timeAccumulate + TimeUnit.NANOSECONDS.toMillis(resultSet.getTxnResponseTime()) <= TRIGGER_TIME){
        batch.add(responseTime);
        timeAccumulate += responseTime;
    }else {
        // output result
        time += 5; throughput = batch.size() * (60000/TRIGGER_TIME);
        avg_latency = timeAccumulate / batch.size();
        Collections.sort(batch);
        quartiles =  this.getQuartiles(batch);
        min = batch.get(0); max = batch.get(batch.size()-1);
```

StatisticMgr.java

Common Mistake 2

- Determine the time interval based on accumulated transaction response time
- Correct: Group transactions into buckets based on transaction start or end time

```
private void addTxnLatency(TxnResultSet rs) {
    long elapsedTime = TimeUnit.NANOSECONDS.toMillis(rs.getTxnEndTime() - Benchmarker.BENCH_START_TIME);
    long timeSlotBoundary = (elapsedTime / GRANULARITY) * GRANULARITY / 1000; // in seconds

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

StatisticMgr.java