Download Salesforce Tables in a Flash

Gregory Smith
Capstorm, LLC

Conventional wisdom says that the fastest way to extract data from Salesforce is to use the Bulk API. This study puts this wisdom to the test by comparing Bulk versus SOAP versus REST under various levels of parallelism. The winner may surprise you.

Table of Contents

The Problem....1
The Approach....2
The Most Important Measurement....3
Experiments....4
  Unbalanced 1.8 Million Records....4
  Balanced 4.9 Million Records....5
  Blow BULK API Performance Out of the Water....5
Conclusions....6
Appendices....7
  Software & Hardware....7
  PK Chunking....7
  Raw 1.8 Million Data Set....7
  Raw 4.9 Million Data Set....8
  How Can I Reproduce These Results?....9

The Problem

Our basic problem was how to bring all data from a set of Salesforce tables and store it in a local Microsoft SQL/Server database. Why? Our business analysts wanted to slice and dice the data quickly with the analytic tools they already know.

Of course, BULK with PK chunking can deliver data fairly quickly unless you need:

- flexibility with the SOQL retrieving the data.
- all tables.

We needed flexibility, all tables, and wanted to achieve performance comparable to PK chunking.

To achieve the desired results we looked at two data sets.

The first data set had 12 tables, 1,788,195 records, and was designed to show how performance varies when the size of tables to extract varies. The tables included and the number of records in each is listed in the the following table.

Table	# Records
Account	100,000
AccountContactRole	9,929

Table	# Records
AccountShare	100,000
Case	562,026
CaseHistory	562,026
Contact	157,927
Lead	17,863
Opportunity	69,403
OpportunityHistory	69,403
OpportunityShare	138,806
Solution	406
SolutionHistory	406

The second data set had 12 tables and 4,905,440 records where each table contained similar sized rows. The actual table names are proprietary but their sizes are as follows:

Table	# Records
Table 1	583,180
Table 2	417,626
Table 3	376,584
Table 4	304,252
Table 5	255,202
Table 6	608,692
Table 7	410,018
Table 8	390,338
Table 9	205,009
Table 10	647,538
Table 11	384,527
Table 12	322,514

The Approach

We ran the same experiments using five Salesforce protocol variations to discover the fastest approach to copy Salesforce tables to SQL/Server. The protocol variations used included:

- REST
- SOAP using the built in JDK based transport layer.

- SOAP using a transport layer based on an Apache/Commons http client pool.
- BULK using the same query parameters as REST and SOAP.
- BULK where all records were read in a single batch.

For each protocol variation we backed up 1, 2, 3, ... up to 10 tables concurrently. We stopped at 10 because the BULK API limits the number of concurrent jobs to 10. However, a later section in this document demonstrates how to use more than 10 concurrent table backups and achieve much better performance than in the base experiments.

Since we wanted all records, even if the tables were changing during our copy, the logical query we ran for each table was like:

- SELECT * FROM Account WHERE systemModStamp>=lastReadTimeStamp ORDER BY systemModStamp

In addition we wanted the ability to resume a query if the Salesforce connection failed (since this happens in practice).

Note: We did not test the BULK API with PK chunking because:

- Many other people have documented its performance.
- It is not applicable to general SOQL queries or all tables.

The Most Important Measurement

For each experiment the only time related value we respected was the amount of time we had to wait for a process to complete. For example, if Salesforce reported that a BULK API job took 2.5 minutes, but we had to wait 20 minutes for the results, then the job took 20 minutes. Our time is what is important.

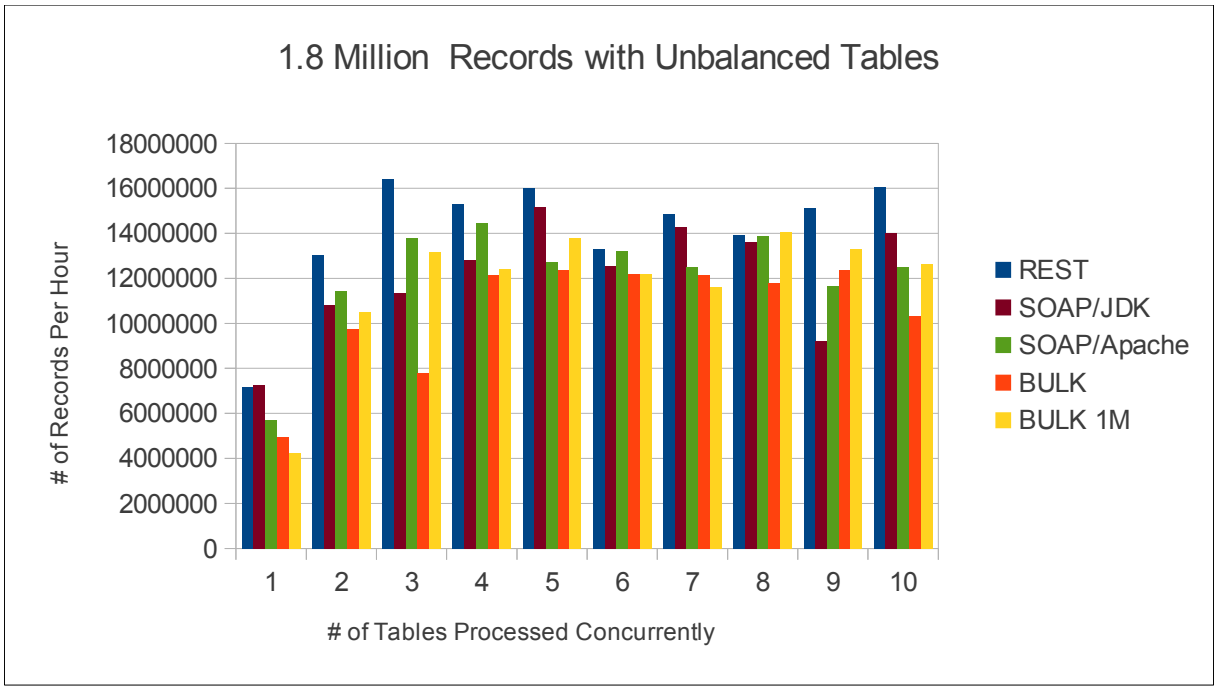
- The only “amount of time” measurement that matters is the “amount of time” we have to wait.

Experiments

Unbalanced 1.8 Million Records

The unbalanced data set contained 12 tables where the distribution of data among the tables was not even. The results for the various protocols is summarized in the following graph where:

- The X axis indicates the number of tables processed concurrently.
- The Y axis indicates the total number of records copied per hour.



It is interesting to note that the BULK API became less attractive as the number of concurrent table backups increased due of the amount of wall time required. Note: "Wall Time" is the amount of time a human has to wait for a process to finish. In retrospect and in light of the next experiment I am guessing that the reason is simple:

- Lower priority BULK SELECT jobs take more wall time than REST or SOAP actions.

This make a lot of sense because:

- A SOQL query will take a similar amount of time to run independent of protocol.
- The communication costs of fetching records 2,000 at time with SOAP (or REST) is not much higher than fetching a complete set of records via BULK.

Not surprisingly, we also learned quite a bit about Salesforce performance.

- The time of day makes a huge difference. For example, in the data we saw that 2 BULK threads ran 25% faster than 3 BULK threads. The difference was the time of day and the corresponding load on Salesforce servers.

- Be wary of performance statistics if they are based on anything but wall time. A BULK job that runs in 3 minutes but takes 60 minutes of wall time is a 60 minute job to the person running it.

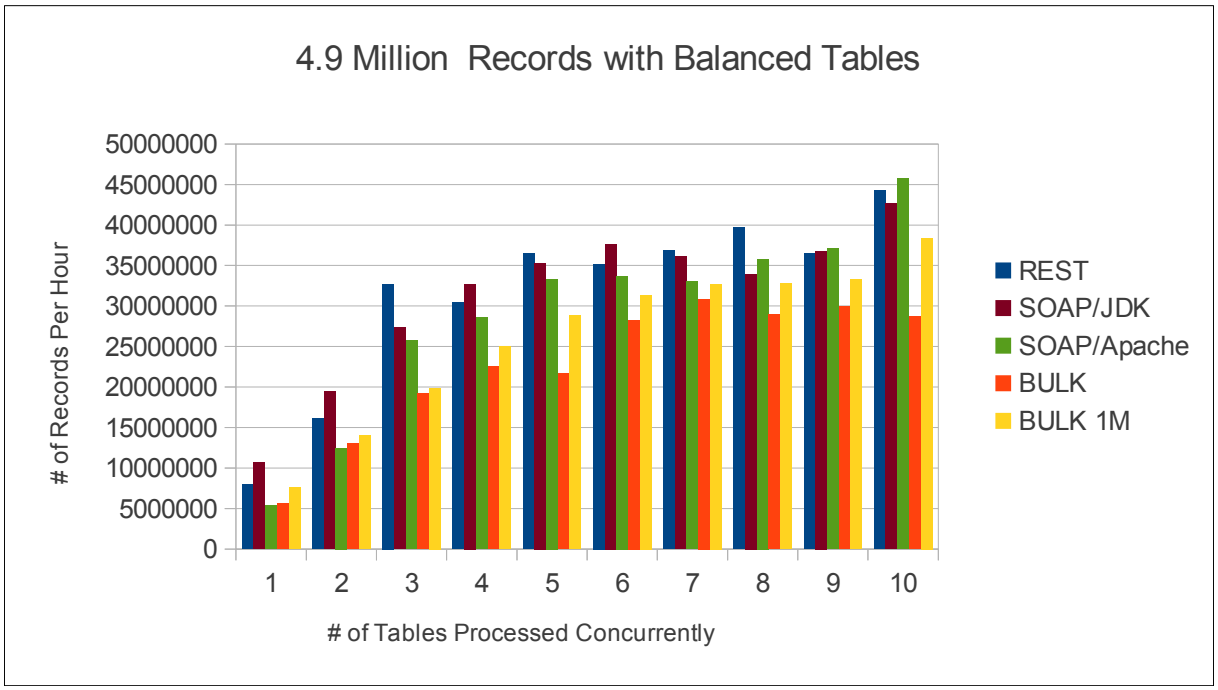
Balanced 4.9 Million Records

The balanced table experiment contained 12 tables where the record size of each table is roughly the same. See the table in the first section of this document for the exact record counts.

That maximum performance achieved was almost 48 millions records per hour and the winning protocol was.....SOAP using an Apache Http client pool.

The results for all protocols is summarized in the following graph where:

- The X axis indicates the number of tables processed concurrently.
- The Y axis indicates the total number of records copied per hour.



Though SOAP/Apache edged out the other approaches, margin between SOAP/JDK and REST are close enough to simply be a time of day factor.

It interesting to see that the BULK API was never faster once 3 or more threads were used.

Blow BULK API Performance Out of the Water

Even though SOAP and REST performed somewhat better than BULK with enough parallelism, there is an approach that will often blow BULK totally out of the water.

The BULK API has two major performance limitations:

- BULK queries run on a slow priority thread than REST and SOAP queries.
- Only 10 BULK jobs can run concurrently.

With SOAP or REST copy performance can be made to skyrocket by backing up more than 10 tables

concurrently. How can this be done?

- Start “N” Salesforce sessions, each with a different Salesforce user. If you use the same Salesforce user for each session you may find Salesforce killing connections randomly because of governor limits.
- Configure each Salesforce session to back up 10 tables concurrently (more than 10 tables per session runs into governor limits). Each session must backup a different set of 10 tables.

Using this approach our performance limiting factor was the raw speed of our internet connection.

Conclusions

When the objective is to minimize the amount of time you have to wait for a Salesforce data extraction and flexibility is needed, parallel extraction using SOAP or REST often performs better than the BULK API.

There were several other lessons learned.

- Waiting for a query to finish on a lower priory BULK queue often takes more wall time than running the same query on a high priority SOAP or REST queue.
- The time of day can make a meaningful difference in the amount of time you have to wait. In North America early in the morning often provides the best performance.
- A SOQL query takes the same amount of time for any API protocol. What differs is the cost of communication and the priority at which Salesforce executes the query.

Finally, these experiments were performed on my farm where the internet connection is modest – 20 down/4 up. I chose this environment so the results would be even better in almost any commercial setting.

Appendices

Software & Hardware

For each backup experiment we used Capstorm's CopyStorm product and stored the data in Microsoft's SQL/Server 2017. The compute environment was modest:

- Commodity quad-core AMD processors with solid state disks.
- CopyStorm ran on a Ubuntu 16.04 environment.
- SQL/Server ran on a Window 10 environment.
- Maximum internet speed was 20 down and 4 up (a lot of this work was done on my farm).

PK Chunking

- https://developer.salesforce.com/docs/atlas.en-us.api_asynch.meta/api_asynch/async_api_headers_enable_pk_chunking.htm

Raw 1.8 Million Data Set

The following table contains the raw data recorded for each 1.8 million record experiment. The columns are:

Column Name	Description
Protocol	The Salesforce protocol used.
# of Threads	The number of tables concurrently processed.
# Records Copied	The total number of records copied from all tables.
Elapsed Minutes	Number of minutes in wall time.
Elapsed Seconds	Number of seconds in wall time
Total Seconds	Total number of seconds for the experiment.
Records Per Second	Number of records copied per second.
Records Per Minute	Number of records copied per minute.
Records Per Hour	Number of records copied per hour.
Secs Minus Slowest	Number seconds required above the time taken for the slowest table in the backup. Example: If the slowest table took 390 seconds and the total backup took 399 seconds then the “Secs Minus Slowest” is 9 seconds.
Avg Table	Average number of seconds per table.
Slowest Table	Number of seconds to backup the slowest table.

Protocol	# of Threads	# Records Copied	Elapsed Minutes	Elapse Seconds	Total Seconds	Records Per Second	Records Per Minute	Records Per Hour	Secs Minus Slowest	Avg Table	Slowest Table
REST	1	1,788,195	14	58	898	1,991	119,479	7,168,710	558	166	340
REST	2	1,788,195	8	14	494	3,620	217,190	13,031,381	124	302	370
REST	3	1,788,195	6	32	392	4,562	273,703	16,422,199	7	380	385
REST	4	1,788,195	7	1	421	4,247	254,850	15,290,979	10	354	411
REST	5	1,788,195	6	42	402	4,448	266,895	16,013,687	4	371	398
REST	6	1,788,195	8	4	484	3,695	221,677	13,300,624	7	308	477
REST	7	1,788,195	7	13	433	4,130	247,787	14,867,210	4	344	429
REST	8	1,788,195	7	42	462	3,871	232,233	13,933,987	5	323	457
REST	9	1,788,195	7	6	426	4,198	251,858	15,111,507	6	350	420
REST	10	1,788,195	6	41	401	4,459	267,560	16,053,621	5	372	396
SOAP/JDK	1	1,788,195	14	44	884	2,023	121,371	7,282,242	523	169	361
SOAP/JDK	2	1,788,195	9	54	594	3,010	180,626	10,837,545	69	251	525
SOAP/JDK	3	1,788,195	9	28	568	3,148	188,894	11,333,630	10	262	558
SOAP/JDK	4	1,788,195	8	23	503	3,555	213,304	12,798,215	5	296	498
SOAP/JDK	5	1,788,195	7	5	425	4,208	252,451	15,147,064	3	351	422
SOAP/JDK	6	1,788,195	8	33	513	3,486	209,146	12,548,737	4	290	509
SOAP/JDK	7	1,788,195	7	31	451	3,965	237,897	14,273,840	3	330	448
SOAP/JDK	8	1,788,195	7	52	472	3,789	227,313	13,638,775	5	316	467
SOAP/JDK	9	1,788,195	11	39	699	2,558	153,493	9,209,588	6	213	693
SOAP/JDK	10	1,788,195	7	39	459	3,896	233,751	14,025,059	6	325	453
SOAP/Apache	1	1,788,195	18	47	1127	1,587	95,201	5,712,069	584	132	543
SOAP/Apache	2	1,788,195	9	24	564	3,171	190,234	11,414,011	79	264	485
SOAP/Apache	3	1,788,195	7	46	466	3,837	230,240	13,814,382	8	320	458
SOAP/Apache	4	1,788,195	7	25	445	4,018	241,105	14,466,297	3	335	442
SOAP/Apache	5	1,788,195	8	25	505	3,541	212,459	12,747,529	5	295	500
SOAP/Apache	6	1,788,195	8	8	488	3,664	219,860	13,191,602	4	305	484
SOAP/Apache	7	1,788,195	8	34	514	3,479	208,739	12,524,323	3	290	511
SOAP/Apache	8	1,788,195	7	44	464	3,854	231,232	13,873,927	4	321	460
SOAP/Apache	9	1,788,195	9	12	552	3,239	194,369	11,662,141	4	270	548
SOAP/Apache	10	1,788,195	8	34	514	3,479	208,739	12,524,323	4	290	510
BULK	1	1,788,195	21	45	1305	1,370	82,216	4,932,952	807	114	498
BULK	2	1,788,195	11	1	661	2,705	162,317	9,739,035	132	225	529
BULK	3	1,788,195	13	44	824	2,170	130,208	7,812,502	207	181	617
BULK	4	1,788,195	8	50	530	3,374	202,437	12,146,230	5	281	525
BULK	5	1,788,195	8	41	521	3,432	205,934	12,356,050	5	286	516
BULK	6	1,788,195	8	49	529	3,380	202,820	12,169,191	4	282	525
BULK	7	1,788,195	8	50	530	3,374	202,437	12,146,230	5	281	525
BULK	8	1,788,195	9	6	546	3,275	196,505	11,790,297	6	273	540
BULK	9	1,788,195	8	40	520	3,439	206,330	12,379,812	5	287	515
BULK	10	1,788,195	10	23	623	2,870	172,218	10,333,069	12	239	611
BULK – 1 Million	1	1,788,195	25	15	1515	1,180	70,820	4,249,176	990	98	525
BULK – 1 Million	2	1,788,195	10	13	613	2,917	175,027	10,501,635	143	243	470
BULK – 1 Million	3	1,788,195	8	9	489	3,657	219,410	13,164,626	14	305	475
BULK – 1 Million	4	1,788,195	8	39	519	3,445	206,728	12,403,665	4	287	515
BULK – 1 Million	5	1,788,195	7	47	467	3,829	229,747	13,784,801	5	319	462
BULK – 1 Million	6	1,788,195	8	49	529	3,380	202,820	12,169,191	4	282	525
BULK – 1 Million	7	1,788,195	9	15	555	3,222	193,318	11,599,103	5	268	550
BULK – 1 Million	8	1,788,195	7	38	458	3,904	234,261	14,055,681	5	325	453
BULK – 1 Million	9	1,788,195	8	4	484	3,695	221,677	13,300,624	5	308	479
BULK – 1 Million	10	1,788,195	8	30	510	3,506	210,376	12,622,553	6	292	504

Raw 4.9 Million Data Set

The following table contains the raw data recorded for each 4.9 million record experiment. The columns are the same as for the 1.8 million experiment.

Protocol	# of Threads	# Records Copied	Elapsed Minutes	Elapse Seconds	Total Seconds	Records Per Second	Records Per Minute	Records Per Hour	Secs Minus Slowest	Avg Table	Slowest Table
REST	1	4,905,440	37	3	2223	2,207	132,401	7,944,032	1939	184	284
REST	2	4,905,440	18	18	1098	4,468	268,057	16,083,410	840	372	258
REST	3	4,905,440	9	0	540	9,084	545,049	32,702,933	155	757	385
REST	4	4,905,440	9	41	581	8,443	506,586	30,395,153	328	704	253
REST	5	4,905,440	8	4	484	10,135	608,112	36,486,744	233	845	251
REST	6	4,905,440	8	22	502	9,772	586,308	35,178,454	184	814	318
REST	7	4,905,440	7	59	479	10,241	614,460	36,867,608	160	853	319
REST	8	4,905,440	7	25	445	11,023	661,408	39,684,458	110	919	335
REST	9	4,905,440	8	4	484	10,135	608,112	36,486,744	119	845	365
REST	10	4,905,440	6	39	399	12,294	737,660	44,259,609	9	1,025	390
SOAP/JDK	1	4,905,440	27	34	1654	2,966	177,948	10,676,895	1440	247	214
SOAP/JDK	2	4,905,440	15	9	909	5,397	323,791	19,427,485	689	450	220
SOAP/JDK	3	4,905,440	10	46	646	7,594	455,614	27,336,817	419	633	227
SOAP/JDK	4	4,905,440	9	0	540	9,084	545,049	32,702,933	311	757	229
SOAP/JDK	5	4,905,440	8	20	500	9,811	588,653	35,319,168	240	818	260
SOAP/JDK	6	4,905,440	7	49	469	10,459	627,562	37,653,697	158	872	311
SOAP/JDK	7	4,905,440	8	9	489	10,032	601,894	36,113,669	159	836	330
SOAP/JDK	8	4,905,440	8	40	520	9,434	566,012	33,960,738	119	786	401
SOAP/JDK	9	4,905,440	8	0	480	10,220	613,180	36,790,800	144	852	336
SOAP/JDK	10	4,905,440	6	54	414	11,849	710,933	42,656,000	47	987	367
SOAP/Apache	1	4,905,440	55	6	3306	1,484	89,028	5,341,677	2899	124	407
SOAP/Apache	2	4,905,440	23	37	1417	3,462	207,711	12,462,656	1090	288	327
SOAP/Apache	3	4,905,440	11	27	687	7,140	428,423	25,705,362	451	595	236
SOAP/Apache	4	4,905,440	10	17	617	7,950	477,028	28,621,692	342	663	275
SOAP/Apache	5	4,905,440	8	50	530	9,256	555,333	33,319,970	256	771	274
SOAP/Apache	6	4,905,440	8	44	524	9,362	561,692	33,701,496	189	780	335
SOAP/Apache	7	4,905,440	8	54	534	9,186	551,173	33,070,382	183	766	351
SOAP/Apache	8	4,905,440	8	13	493	9,950	597,011	35,820,657	134	829	359
SOAP/Apache	9	4,905,440	7	56	476	10,306	618,333	37,099,966	106	859	370
SOAP/Apache	10	4,905,440	6	26	386	12,708	762,504	45,750,218	22	1,059	364
BULK	1	4,905,440	52	34	3154	1,555	93,318	5,599,107	2796	130	358
BULK	2	4,905,440	22	35	1355	3,620	217,215	13,032,903	1040	302	315
BULK	3	4,905,440	15	19	919	5,338	320,268	19,216,087	596	445	323
BULK	4	4,905,440	13	3	783	6,265	375,896	22,553,747	444	522	339
BULK	5	4,905,440	13	33	813	6,034	362,025	21,721,506	401	503	412
BULK	6	4,905,440	10	25	625	7,849	470,922	28,255,334	266	654	359
BULK	7	4,905,440	9	32	572	8,576	514,557	30,873,399	195	715	377
BULK	8	4,905,440	10	9	609	8,055	483,295	28,997,675	199	671	410
BULK	9	4,905,440	9	49	589	8,328	499,705	29,982,316	162	694	427
BULK	10	4,905,440	10	16	616	7,963	477,803	28,668,156	191	664	425
BULK – 1 Million	1	4,905,440	38	49	2329	2,106	126,375	7,582,475	2054	176	275
BULK – 1 Million	2	4,905,440	20	57	1257	3,902	234,150	14,048,993	960	325	297
BULK – 1 Million	3	4,905,440	14	49	889	5,518	331,076	19,864,549	581	460	308
BULK – 1 Million	4	4,905,440	11	47	707	6,938	416,303	24,978,195	417	578	290
BULK – 1 Million	5	4,905,440	10	13	613	8,002	480,141	28,808,457	297	667	316
BULK – 1 Million	6	4,905,440	9	23	563	8,713	522,782	31,366,934	226	726	337
BULK – 1 Million	7	4,905,440	9	0	540	9,084	545,049	32,702,933	170	757	370
BULK – 1 Million	8	4,905,440	8	59	539	9,101	546,060	32,763,607	156	758	383
BULK – 1 Million	9	4,905,440	8	51	531	9,238	554,287	33,257,220	134	770	397
BULK – 1 Million	10	4,905,440	7	40	460	10,664	639,840	38,390,400	61	889	399

How Can I Reproduce These Results?

We ran each experiment using Capstorm's CopyStorm product for three reasons:

- It has parameters that support each experiment without additional work on our part.
- The multiple session experiment could be run with a short shell script.
- The results were stored in a relational database so comparison for accuracy was easy.

Except for storing the data in a relational database, the rest of the results are fairly easy to reproduce as long as your communication with Salesforce is reliable during the experiments. If not, then the addition of recovery code makes a solution a bit harder.

To test 10 concurrent tables, fundamentally all that is needed are REST requests using *curl* that are submitted in parallel. For more than 10 tables, the problem is a bit harder because you will run into Salesforce governor concurrency issues and will need a queuing system (and recovery code) to get high performance. In addition, if your tables are much larger than our test cases, multiple SOQL statements with LIMIT parameters will be needed to avoid SOQL timeouts.