USING SQL ON THE MQSMFCSV DATA

Part I – Getting started and some simple examples

Abstract

This document is about how I started using SQL against the MQ SMF data, getting answers to questions using SQL.

Carolyn Elkins

elkinsc@us.ibm.com

Contents

Using the data from MQSMFCSV

A problem that I have long had with the interpretation of MQ SMF data is simply that many of the processing programs do a lot of interpretation for you and do not present the raw data in a consumable format. At times this means that data can be incorrectly presented, creating errors in interpretation. Quite possibly the best example of this is a now infamous 'problem' where the CPU fields were being rounded up to a full CPU second. The customer saw that in their testing the cost of small persistent messages compared to the same sized nonpersistent messages was "the same" when in fact the CPU consumption was almost 3 times as high – both 27 CPU microseconds and 75 CPU micro seconds were being rounded to the same value.

The MQ supplied program, CSQ4SMFD, produces a dump like form of the data, but I have only found that useful when you are verifying the results from a report.

I kept saying that what I needed was a program that would format the records and leave the interpretation to me. Something I could use in conjunction with the reports from MP1B, allowing me to interrogate the SMF data directly. Be careful what you ask for, my grandmother used to say, because the devil has ears. While not Satan, Mark Taylor did listen and provided a tool. And he was rather surprised that others were quite interested as well.

You can find the tool at http://github.com/ibm-messaging/mq-smf-csv including source code, examples and ready-to-run executables for some platforms.

Not only is it exactly what I was asking for, he's already made a number of improvements to provide much greater ease of use. And I have already successfully used it to assist with live customer SMF data – a very good use of time.

Using DB2

One of the most useful features is the ability to use SQL against the data as it can help locate specific problem areas and provide detailed information like no other. While I have used Microsoft Access for the CSV files that are produced by MP1B, MS Access will not work with the number of columns or characters produced by the field by field translation done by MQSMFCSV. As a result, Mark altered the MQSMFCSV program to create the DDL for tables that match the SMF records. All I had to do is to learn to load the data and begin using SQL requests against the data.

I am certainly not a DDL, DB2, or SQL expert, so there may be far more efficient ways to extract some of the data. I know that proper indexing will be particularly helpful, especially if you load many days' worth of data into the tables. The DDL generated by MQSMFCSV does not include indexes for the tables, but where I have added an index or have a suggestion for one I will mention that in the examples. How the data will be used in your environment is the key to determining whether it will be useful to add an index or two.

Also note that the examples used in this document are using DB2 DataStudio on Windows. My teammate Mitch Johnson has already written up how to upload the CSV files and DDL to use DB2 on z/OS should that be required.

Simple Example 1 – How many messages were put to and retrieved from a specific queue?

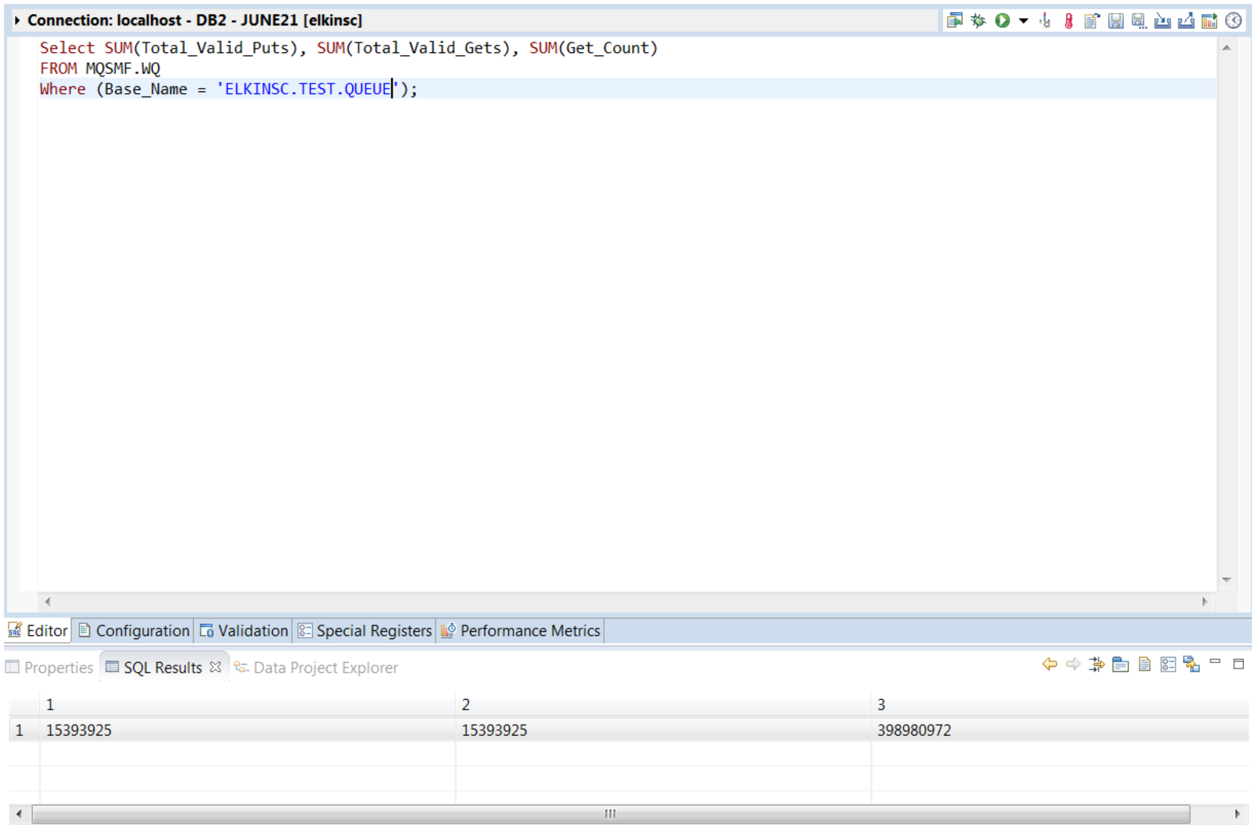
This simple query can be quite useful to determine whether the message volume is what is expected during testing or matches what is reported by a monitoring tool. Most monitoring tools use the 'RESET QSTATS' command to retrieve the number of puts and gets, but that number is only valid if a single monitoring tool is the only application using this command.

The query also includes the total number of MQGET requests issued, which can be used to compare against the number of requests made that actually returned data. This information can help with performance tuning. If for example the number of MQGETs issued during the period being examined is much higher than the number of valid MQGETs then the application may be 'polling' for data rather than waiting for data to arrive. It can also indicate that the wait interval specified may be too short, and lengthening that interval may help conserve CPU time or there may be too many instances of the getting application for the volume of messages and all are competing.

The query used was:

```
    Select SUM(Total_Valid_Puts), SUM(Total_Valid_Gets), SUM(Get_Count)
    FROM MQSMF.WQ
    Where (Base_Name = 'ELKINSC.TEST.QUEUE');
```

The results from the query issued via the IBM DataStudio tool:


Connection: localhost - DB2 - JUNE21 [elkinsc]

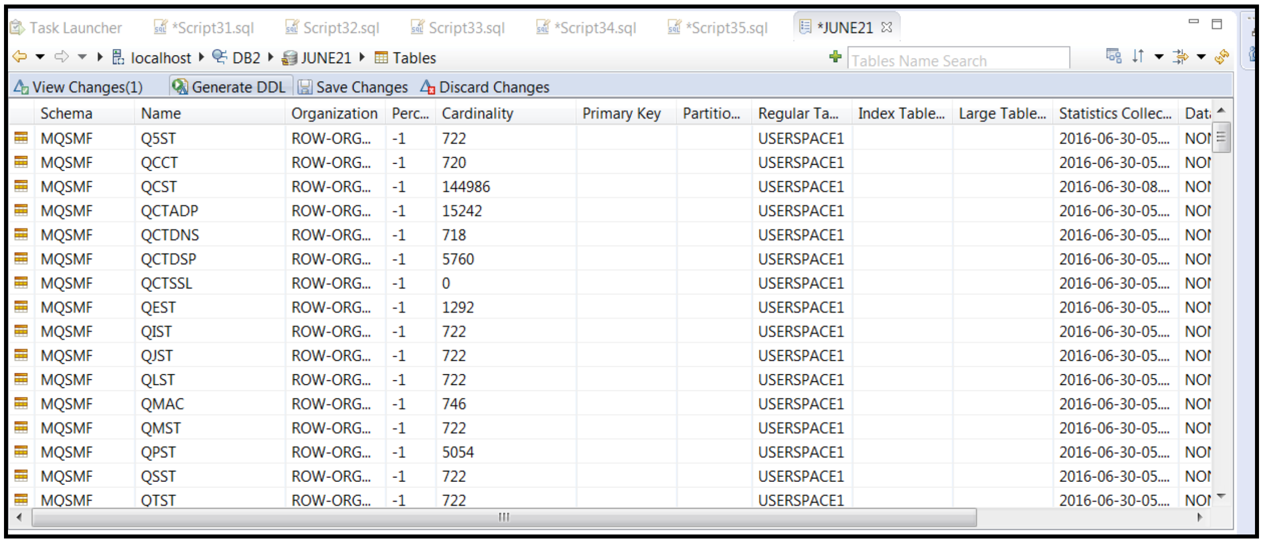
```
Select SUM(Total_Valid_Puts), SUM(Total_Valid_Gets), SUM(Get_Count)
FROM MQSMF.WQ
Where (Base_Name = 'ELKINSC.TEST.QUEUE');
```

	1	2	3
1	15393925	15393925	398980972

There were 15,393,925 valid puts, the same number of valid gets, and a substantially higher number of total gets – 398,980,972 – issued against the queue during the date and time examined. Given the kind of processing that was being tested, the total MQGET to valid MQGET ratio of 26 times indicates that there may be too many getting applications waiting for work. It could also indicate that the applications are 'polling' rather than using an MQGET with a wait option.

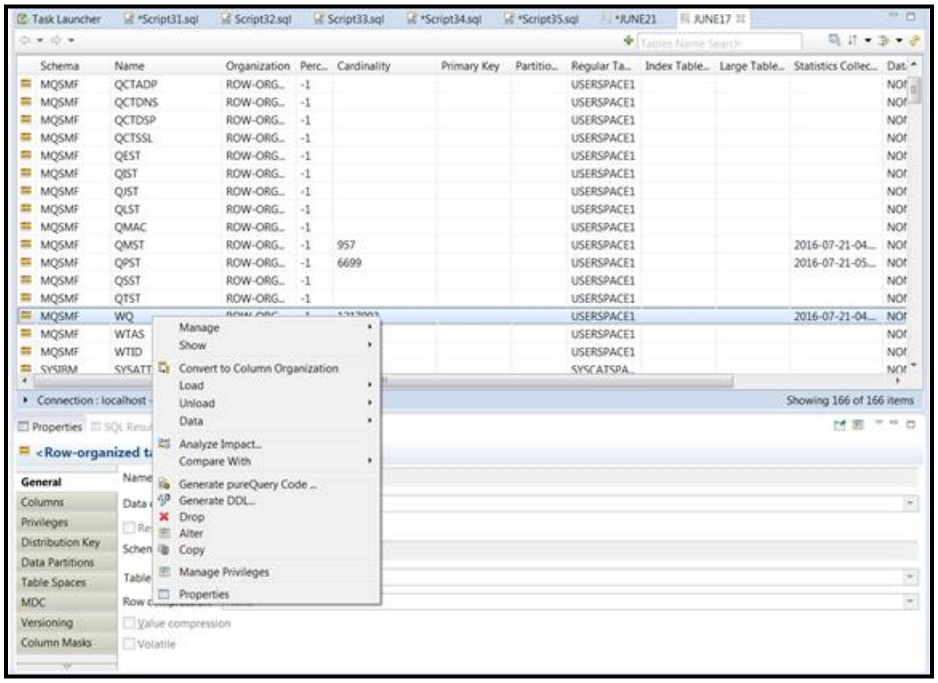
Note that if this table has many rows or there are going to be a lot of queries based on the queue name, which I found myself doing quite a bit, then adding that as an index to the table is helpful. To add an index using IBM DataStudio:

1) Expand the Tables list for the SMF database. You should see a list that looks like this:



Schema	Name	Organization	Perc...	Cardinality	Primary Key	Partitio...	Regular Ta...	Index Table...	Large Table...	Statistics Collec...	Dat
MQSMF	Q5ST	ROW-ORG...	-1	722			USERSPACE1			2016-06-30-05....	NO
MQSMF	QCCT	ROW-ORG...	-1	720			USERSPACE1			2016-06-30-05....	NO
MQSMF	QCST	ROW-ORG...	-1	144986			USERSPACE1			2016-06-30-08....	NO
MQSMF	QCTADP	ROW-ORG...	-1	15242			USERSPACE1			2016-06-30-05....	NO
MQSMF	QCTDNS	ROW-ORG...	-1	718			USERSPACE1			2016-06-30-05....	NO
MQSMF	QCTDSP	ROW-ORG...	-1	5760			USERSPACE1			2016-06-30-05....	NO
MQSMF	QCTSSL	ROW-ORG...	-1	0			USERSPACE1			2016-06-30-05....	NO
MQSMF	QEST	ROW-ORG...	-1	1292			USERSPACE1			2016-06-30-05....	NO
MQSMF	QIST	ROW-ORG...	-1	722			USERSPACE1			2016-06-30-05....	NO
MQSMF	QJST	ROW-ORG...	-1	722			USERSPACE1			2016-06-30-05....	NO
MQSMF	QLST	ROW-ORG...	-1	722			USERSPACE1			2016-06-30-05....	NO
MQSMF	QMAC	ROW-ORG...	-1	746			USERSPACE1			2016-06-30-05....	NO
MQSMF	QMST	ROW-ORG...	-1	722			USERSPACE1			2016-06-30-05....	NO
MQSMF	QPST	ROW-ORG...	-1	5054			USERSPACE1			2016-06-30-05....	NO
MQSMF	QSST	ROW-ORG...	-1	722			USERSPACE1			2016-06-30-05....	NO
MQSMF	QTST	ROW-ORG...	-1	722			USERSPACE1			2016-06-30-05....	NO

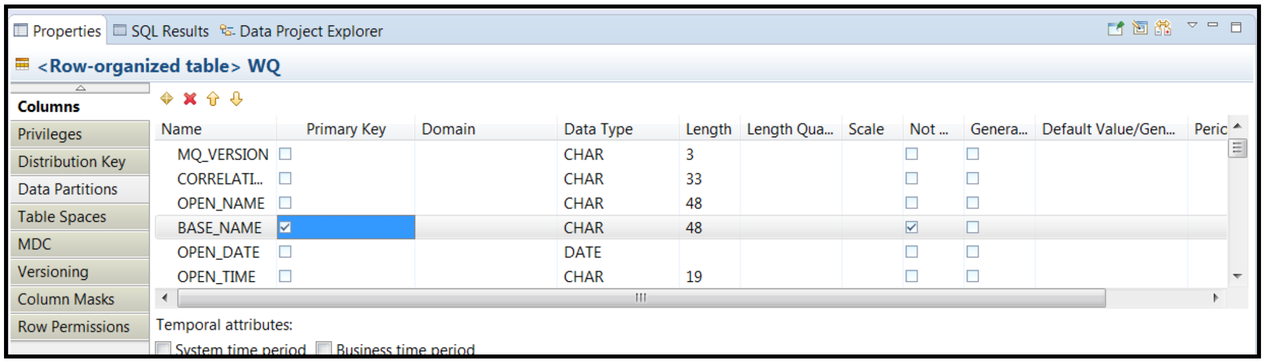
2) Scroll down to the WQ table, right click on the table, and select ‘Alter’.



Schema	Name	Organization	Perc...	Cardinality	Primary Key	Partitio...	Regular Ta...	Index Table...	Large Table...	Statistics Collec...	Dat
MQSMF	QCTADP	ROW-ORG...	-1				USERSPACE1				NO
MQSMF	QCTDNS	ROW-ORG...	-1				USERSPACE1				NO
MQSMF	QCTDSP	ROW-ORG...	-1				USERSPACE1				NO
MQSMF	QCTSSL	ROW-ORG...	-1				USERSPACE1				NO
MQSMF	QEST	ROW-ORG...	-1				USERSPACE1				NO
MQSMF	QIST	ROW-ORG...	-1				USERSPACE1				NO
MQSMF	QJST	ROW-ORG...	-1				USERSPACE1				NO
MQSMF	QLST	ROW-ORG...	-1				USERSPACE1				NO
MQSMF	QMAC	ROW-ORG...	-1				USERSPACE1				NO
MQSMF	QMST	ROW-ORG...	-1	957			USERSPACE1			2016-07-21-04...	NO
MQSMF	QPST	ROW-ORG...	-1	6699			USERSPACE1			2016-07-21-05...	NO
MQSMF	QSST	ROW-ORG...	-1				USERSPACE1				NO
MQSMF	QTST	ROW-ORG...	-1				USERSPACE1				NO
MQSMF	WQ	[illegible]	[illegible]	[illegible]			USERSPACE1			2016-07-21-04...	NO
MQSMF	WTAS						USERSPACE1				NO
MQSMF	WTID						USERSPACE1				NO
SYSIBM	SYSATT						SYSCATSPA...				NO

3) Under the ‘Properties’ tab, click on the ‘Columns’ box.

4) Scroll thru the columns, selecting 'Base_Name' as the primary key.



Name	Primary Key	Domain	Data Type	Length	Length Qua...	Scale	Not ...	Genera...	Default Value/Gen...	Peric
MQ_VERSION	☐		CHAR	3			☐	☐		
CORRELATI...	☐		CHAR	33			☐	☐		
OPEN_NAME	☐		CHAR	48			☐	☐		
BASE_NAME	✓		CHAR	48			✓	☐		
OPEN_DATE	☐		DATE				☐	☐		
OPEN_TIME	☐		CHAR	19			☐	☐		

Temporal attributes:
☐ System time period ☐ Business time period

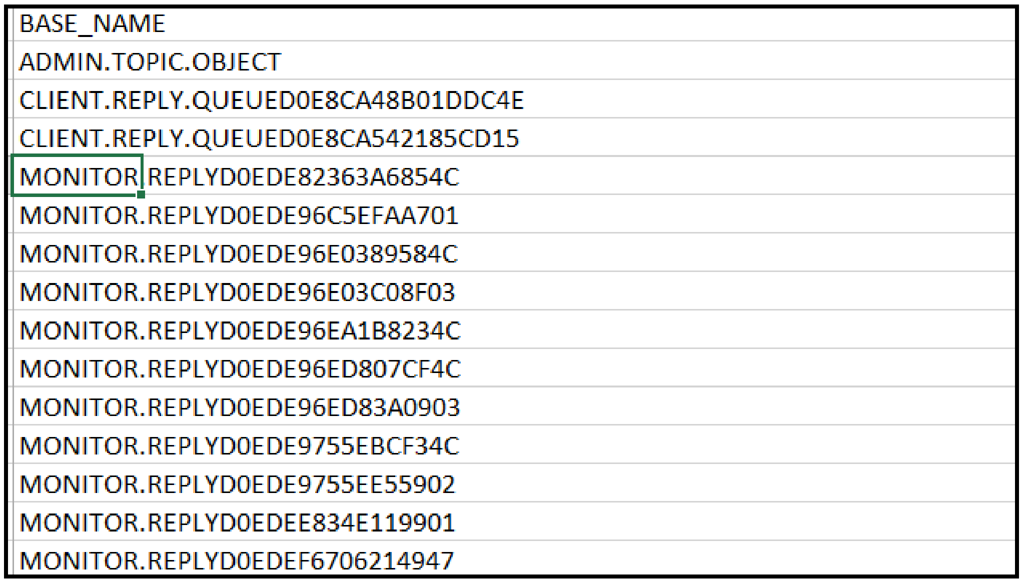
Simple Example 2 – What queues are using a specific bufferpool and/or pageset?

As an administrator I would expect to know this, but it's often useful to check on what's really being used. Especially with a queue manager that was inherited from a prior admin, when looking at dynamic queues, when workload patterns have changed, or when you tend to use 'define like' too much (guilty as charged!). When using older versions of the MP1B program the information could be located as there was one line in the task report giving the queue name and one giving the bufferpool and pageset information. It was just a matter of finding the queue name and the bufferpool line and correlating the data. Those searches no longer work as well on the newer task reports, as the queue name is repeated on every line so ferreting out that information is more challenging when you have a multi-million-line report!

The sample bufferpool query used looks as follows:

```
Select DISTINCT(Base_Name)
FROM MQSMF.WQ
Where (BufferPool_ID = 4);
```

The output was exported to a CSV file because there were many more rows than expected. The results looked as follows (and I've changed the high level qualifiers to protect privacy):



BASE_NAME
ADMIN.TOPIC.OBJECT
CLIENT.REPLY.QUEUED0E8CA48B01DDC4E
CLIENT.REPLY.QUEUED0E8CA542185CD15
MONITOR.REPLYD0EDE82363A6854C
MONITOR.REPLYD0EDE96C5EFAA701
MONITOR.REPLYD0EDE96E0389584C
MONITOR.REPLYD0EDE96E03C08F03
MONITOR.REPLYD0EDE96EA1B8234C
MONITOR.REPLYD0EDE96ED807CF4C
MONITOR.REPLYD0EDE96ED83A0903
MONITOR.REPLYD0EDE9755EBCF34C
MONITOR.REPLYD0EDE9755EE55902
MONITOR.REPLYD0EDEE834E119901
MONITOR.REPLYD0EDEF6706214947

In fact, during the test period there were tens of thousands of temporary dynamic queues created and destroyed by the monitoring tools and clients in use, all of these competing for resources against some application private queues using this particular bufferpool. Even though it was not constrained for space, the heavy use and the volume of temporary dynamic queues appeared to be contributing to slowdowns in some of the permanent queues. The model queues were altered to use a different storage class, and there was a slight smoothing out of response times for the applications not using TD queues. The monitoring interval was probably also set too low, at a query per second against all their queues (and there are many defined, though few actually used).

The equivalent pageset query is:

```
Select DISTINCT(Base_Name)
FROM MQSMF.WQ
Where (Pageset_ID = 4);
```

In this case the results were the same as the bufferpool query.

NOTE: There are times when the bufferpool and pageset will appear as zero ('0') in the SMF data, the areas that should be reserved for MQ's use. This is even though the storage class being used is pointing to an application bufferpool and pageset. This is most often observed when MQ has not yet had to interact with the buffer manager for this queue, typically when messages are non-persistent and are being put directly to an application's open buffer (known as 'put to waiting getter').

Simple Example 3 – What was the oldest message retrieved from a specific queue?

This example was actually used when trying to evaluate a problem situation. The question came directly from the customer who suspected that the getting application was really not keeping up after some changes had been made and as more messages were coming in the messages were languishing on the queue. This used to be quite difficult to find from the task and queue summary reports, but became quite easy once the data was in a database.

The query used was:

```
Select MAX(Max_Time_On_Queue_us), MIN(Min_Time_On_Queue_us)
FROM MQSMF.WQ
Where (Base_Name  = ‘ELKINSC.TEST.QUEUE’);
```

The results showed that the longest time a message was on the queue was 41,009 microseconds and the shortest time was 0 microseconds. Please note that there are both seconds and microseconds in the SMF record, as shown in this extract from the queue DDL (MQSMF.WQ table):

```
Max_Time_On_Queue_s      INTEGER
Max_Time_On_Queue_us     INTEGER
Min_Time_On_Queue_s      INTEGER
Min_Time_On_Queue_us     INTEGER
```

Unless messages are very, very old use the microseconds field for these types of selects.

Simple Example 4 – How many full batches were sent across the channels?

There are many applications that transmit a lot of data across MQ channels, both cluster and sender/receiver pairs. Using MQ as a backbone for file transfers, like the MQ Family product MFT, or sending replicated data, like InfoSphere Replication Server (aka QREP), are good examples of the type of process that benefit from tuning the amount of data that makes up a 'batch' to MQ when communicating with the network. Some high volume online work can also benefit from tuning the batch size, limit and batch interval - especially those workloads that have extremely high peak periods (market open, black Friday, pension day, etc.).

The first area of investigation is typically around the number of batches sent and how many of them were full. Before MQ Version 8 that information was not readily attainable, but it is now – and can be very helpful.

The first thing I check is how many records there are, what are the total number of batches sent and received, and how many full batches there were. Then I look for how many records actually have full batches, assuming that there are some. Following that I look for additional information about the records that did have full batches to see if their behavior can be improved – and I generally start looking at these channels because there are typically far fewer of them.

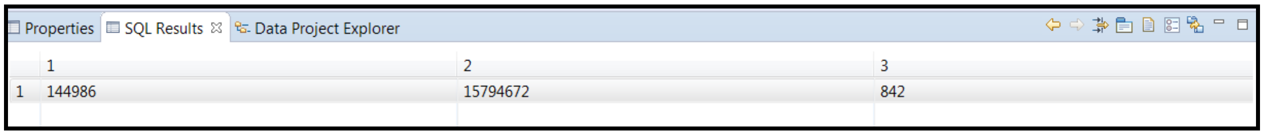
Sample Queries used:

```
SELECT COUNT(Chl_Name), SUM(Batches), SUM(Full_Batches)
FROM MQSMF.QCST;

SELECT COUNT(Chl_Name) FROM MQSMF.QCST
WHERE (Full_Batches > 0);

 SELECT QMGR, Chl_Name, Connection_Name, BatchSize, Batches, Full_Batches,
                        Batch_Interval, Batch_Data_Limit
FROM MQSMF.QCST
WHERE ( Full_Batches > 0);
```

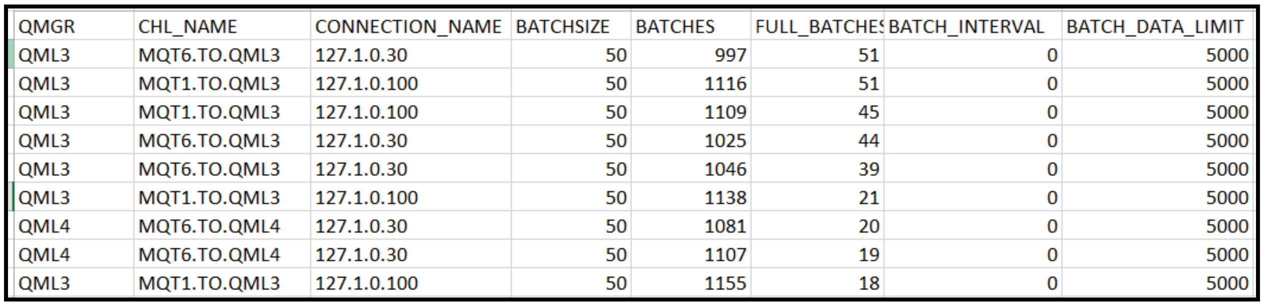
In this particular investigation, the results of the first query showed that during the test period there were 144,986 channel accounting records; 15,794,672 batches sent and received; and 842 full batches. That is remarkably low for a very high volume workload.



	1	2	3
1	144986	15794672	842

The result of the second query was 171 channel accounting records showed full batches transmitted.

The final query showed the channels that had full batches and some of the channel characteristics. The results were exported to a CSV file and sorted by the number of full batches, an excerpt is shown.



QMGR	CHL_NAME	CONNECTION_NAME	BATCHSIZE	BATCHES	FULL_BATCHES	BATCH_INTERVAL	BATCH_DATA_LIMIT
QML3	MQT6.TO.QML3	127.1.0.30	50	997	51	0	5000
QML3	MQT1.TO.QML3	127.1.0.100	50	1116	51	0	5000
QML3	MQT1.TO.QML3	127.1.0.100	50	1109	45	0	5000
QML3	MQT6.TO.QML3	127.1.0.30	50	1025	44	0	5000
QML3	MQT6.TO.QML3	127.1.0.30	50	1046	39	0	5000
QML3	MQT1.TO.QML3	127.1.0.100	50	1138	21	0	5000
QML4	MQT6.TO.QML4	127.1.0.30	50	1081	20	0	5000
QML4	MQT6.TO.QML4	127.1.0.30	50	1107	19	0	5000
QML3	MQT1.TO.QML3	127.1.0.100	50	1155	18	0	5000

After reviewing this information, this application and environment could likely benefit from some iterative testing to find what I think of as the transmission sweet spot. The default values for MQ channels have been used, and that is probably not ideal for this high volume application. We have often seen really positive results from increasing the batch interval – that is the time (in milliseconds) the message channel waits to see if more messages will arrive before sending a batch that is not full.

In this particular test the SMF interval was set to 2 minutes, so as an example in those 2 minutes there were around 1,100 batches sent – somewhat less than 100 per second) and that was for around 30,000 messages (another query to get the detailed record). That means the message arrival rate on the tranmission queue is approximately 250 per second. If the batch interval was set to as low as 10 milliseconds chances are quite good that there will many more full batches, which can actually improve the message throughput during peak periods because there will be fewer data transmissions. This may impact response times during non-peak periods, but has the advantage of 'smoothing out' response times. If there are 'unofficial response time expectations' smothing out the response time between peak and non-peak periods can improve perception of the overall environment.

Another area of investigation is using the batch data limit in place of the number of messages. Feedback from customers that have a wide variation in message sizes has indicated this is a much better option for tuning channel throughput.

Simple Example 5 – What was my longest Log I/O?

After MQ V7.0.1 was released there was an APAR that added fields to the log manager statistics to provide information on the longest I/O, suspend times, etc. These fields were added in response to the number of problems customers were experiencing with long I/O waits on MQ logs – delaying a lot of work. The data gathered is for both logs; if single logging is used the second set will be null values.

One thing that I normally try to identify is if the long I/O times are an anomaly or a constant presence in the queue manager. If the answer is a constant factor, response times are critical, and message traffic is persistent, then working with the I/O subsystem manager may help uncover some underlying issues. It may be that the volume of logging has just become too high to be absorbed and a second queue manager is necessary to handle the workload – the MP1B log statistics report will show the overall amount of logging being done in MB per second. It may be that there is an underlying issue in MQ log placement; we have seen problems caused by all logs for both MQ and another highly active subsystem going to the same physical device. Another area to look for is MQ pageset definition on the on the same devices as the logs (especially if they are at the same time).

When evaluating data from a customer, I look at individual queue managers. I look for the average duration in I/O for each queue manager. It is important to check both the primary and secondary log if dual logging is used.

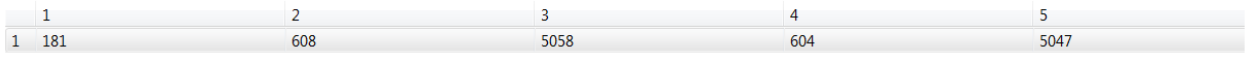
Like with Sample 4, I use multiple queries to look at this data.

Sample Queries used:

```
SELECT COUNT(QMgr), AVG(IO_Max_Duration_1_1_us), MAX(IO_Max_Duration_1_1_us),
                    AVG(IO_Max_Duration_2_1_us), MAX(IO_Max_Duration_2_1_us)
FROM MQSMF.QJSTW
Where QMgr = 'QML1';


SELECT QMgr, IO_CI_1_1, IO_Total_Time_1_1_us, IO_Max_Duration_1_1_us,
             IO_Max_Log_ID_1_1, IO_Max_Suspend_Dur_1_1_us,
             IO_CI_2_1, IO_Total_Time_2_1_us, IO_Max_Duration_2_1_us,
             IO_Max_Log_ID_2_1, IO_Max_Suspend_Dur_2_1_us
FROM (Select * from MQSMF.QJST where QMgr = 'QML1')
Where IO_Max_Duration_1_1_us > 2000;
```

The response from the first query showed:

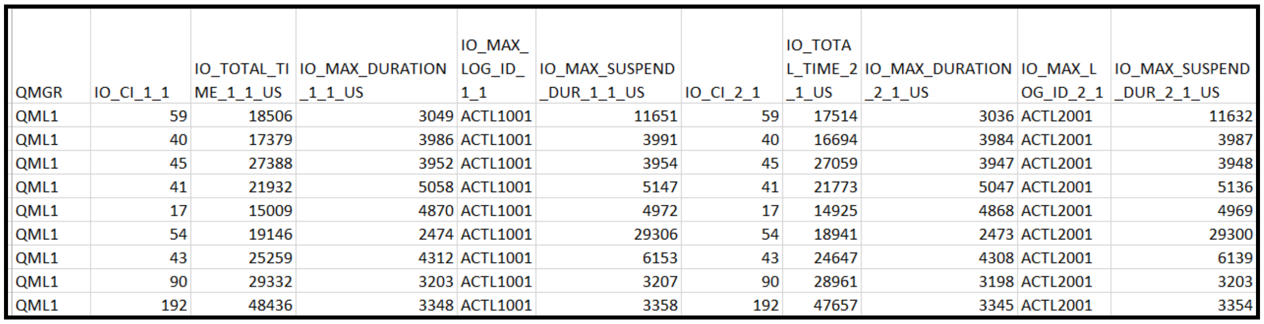


	1	2	3	4	5
1	181	608	5058	604	5047

This means that there were 181 records for the first queue manager being evaluated, the average duration for log copy 1 was 608 microseconds, and the maximum duration was 5,058 microseconds.

The second log copy is almost the same, which is typically a good sign. If I had been told that the longest response time expected from an I/O is 2,000 microseconds then I would follow up that query with the second one to see how often these long I/Os were taking place.

Of the 181 entries for the queue manager in the SMF data submitted, 21 have a maximum I/O duration of greater than 2,000 microseconds.



QMGR	IO_CI_1_1	IO_TOTAL_TIME_1_1_US	IO_MAX_DURATION_1_1_US	IO_MAX_LOG_ID_1_1	IO_MAX_SUSPEND_DUR_1_1_US	IO_CI_2_1	IO_TOTAL_TIME_2_1_US	IO_MAX_DURATION_2_1_US	IO_MAX_LOG_ID_2_1	IO_MAX_SUSPEND_DUR_2_1_US
QML1	59	18506	3049	ACTL1001	11651	59	17514	3036	ACTL2001	11632
QML1	40	17379	3986	ACTL1001	3991	40	16694	3984	ACTL2001	3987
QML1	45	27388	3952	ACTL1001	3954	45	27059	3947	ACTL2001	3948
QML1	41	21932	5058	ACTL1001	5147	41	21773	5047	ACTL2001	5136
QML1	17	15009	4870	ACTL1001	4972	17	14925	4868	ACTL2001	4969
QML1	54	19146	2474	ACTL1001	29306	54	18941	2473	ACTL2001	29300
QML1	43	25259	4312	ACTL1001	6153	43	24647	4308	ACTL2001	6139
QML1	90	29332	3203	ACTL1001	3207	90	28961	3198	ACTL2001	3203
QML1	192	48436	3348	ACTL1001	3358	192	47657	3345	ACTL2001	3354

This is not a significant number of long I/O responses, so in production is likely to be watched and not acted on.