::ISUG

International Sybase Users Group

techcast series

MDA Tables in ASE—Tips and Tricks

April 18, 2006

SYBASE

::ISUG International Sybase Users Group

techcast series

MDA Tables in ASE—Tips and Tricks

Peter Dorfman
Senior Staff Software Engineer
Sybase, Inc.
peter.dorfman@sybase.com

SYBASE

::ISUG International Sybase Users Group

techcast series

MDA Tables in ASE—Tips and Tricks

Rob Verschoor
Senior Evangelist
Sybase, Inc
robv@sybase.com

SYBASE

Topics

- Quick introduction to MDA tables
- Possible applications of MDA tables
  - What's that application doing?
  - Identifying unused indexes
  - Identifying 'hot' tables
- Historical MDA tables
- Archiving historical MDA table data
- Performance impact of MDA tables
- Counter wrap
- Analyzing stored procedure activity
- Miscellaneous topics
- Recent Enhancements
- Q&A

Quick Introduction to MDA Tables

- MDA tables were introduced in ASE 12.5.0.3
  - MDA = Monitoring and Diagnostic Access
  - also called “monitoring tables”
- 39 proxy tables in ‘master’ database **(35 in ASE 12.5)**
  - monSysSQLText, monObjectActivity, monCachedObject (etc.)
  - Can be accessed with regular SQL statements
  - When queried, tables are created on-the-fly from memory structures
  - No data is stored in **master** database
  - The proxy tables may also be created in a different database than **master**
- Must be installed: ‘installmontables’ script
- No license needed: included in ASE base product
- Only small performance impact on ASE (<5%)

Quick Introduction to MDA Tables

- MDA tables provide access to low-level monitoring data
  - Resource usage in ASE per table/query/entire server
  - Current activity in ASE per spid/query/procedure/table
  - Recent activity: recently completed statements, with the resources they required
- Some examples of practically relevant information:
  - Amount of memory occupied in the data cache by each table or index
  - Most frequently used tables/procedures
  - Top-N queries for CPU, I/O, elapsed time,...
  - Find unused indexes
  - SQL text of currently executing/recently executed statements
  - Automatically kill user processes that have been idle for more than X minutes
  - Provide server status information even when **tempdb** is full

What *ARE* the MDA Tables?

- The MDA tables are proxy tables
- Defined in the master database
- Defined in installmontables script
- Consume no space
- Table definitions map to ASE RPC calls

```
create existing table monProcessLookup (
   SPID                      smallint,
   KPID                      int,
   Login                     varchar(30) NULL,
   Application               varchar(30) NULL,
   ClientHost                varchar(30) NULL,
   ClientIP                  varchar(64) NULL,
   ClientOSPID               varchar(30) NULL,
)
external procedure
at "loopback...$monProcessLookup"
```

Reference to Local server

RPC Call

Quick Introduction to MDA Tables

- For more MDA basics, and a brief discussion of all tables:
  - See presentations from past Techwave conferences (**www.sypron.nl/mda**)
- In this presentation:
  - We want to go one step further than just the basics
  - Look at practical applications of MDA tables
  - Things that are useful for you as a DBA

Possible Applications of MDA Tables

What's That Application Doing? SQL Text

- Does this sound familiar?
  - A third-party 'black box' application runs on your ASE server
  - You have the feeling it sometime slows down the entire server...
  - … but you don't know which queries it is sending to ASE
- Classic solutions:
  - Use "cmdtext" auditing to intercept the application's T-SQL commands
  - Use traceflag 11202 (writes all incoming client language to the errorlog)
  - Use third-party tools to find T-SQL commands by intercepting network packets
  - dbcc sqltext()
  - …but all these methods have significant limitations or drawbacks

What’s That Application Doing?

- Solution: MDA tables:

  monProcessSQLText **&** monSysSQLText
- monProcessSQLText: currently executing SQL
- monSysSQLText: recently executed SQL, now completed
  - Historical table
  - Lets you “look back” in time
  - By copying rows regularly into an ‘archive’ table, complete history can be preserved

What's That Application Doing? Statement Statistics

- Also: monSysStatement: info about completed SQL statement
  - Number of logical I/Os
  - Number of physical I/Os
  - Number of network packets sent/received
  - Number of milliseconds of 'waiting time' during statement execution
  - Exact starttime & endtime of execution
  - Not the SQL Text itself; for this, see **monSysSQLText**
  - Historical table
  - Lets you "look back" in time
  - By copying rows regularly into an 'archive' table, complete history can be preserved

Measuring Statement I/O and CPU Time

- Most I/O intensive statement

```
select * into #ts from master..monSysStatement

select KPID, BatchID, LineNumber, LogicalReads, Elapsed = datediff(ms,
   StartTime, EndTime) from #ts where LogicalReads > 10000
   order by 4 desc
```

```
 KPID         BatchID      LineNumber   LogicalReads Elapsed
 ----------- ----------- ----------- ------------ -----------
   574619857           9          13      5509405       68613
   575602932          10           3       360656        7956
   575209751          10           3        86241         606
   575406493          10           3        59546         983
   576258422           2          62        39963         223
   575275534           1          62        39959         216
   575930476           2          62        39955         250
   332857800       65454           1        15884        1600
   575275534           9           1        12758         176
```

Capturing SQL Text

```
select * into #tsql from master..monSysSQLText

select SQLText from #tsql where KPID= 574619857
order by  BatchID, SequenceInBatch

    SQLText
    ---------------------------------------------------------

    select  admnr = lvd.id_lm_adres,
            lvd.id_logmiddel
        from
              logmi.dbo.lm_voorraad lvd,
              ravar.dbo.adm_relatie adm
        where
               lvd.cdsys_lm_adrestype  = "A"
        and    lvd.cdsys_lm_opslagstat = "O"
        and    lvd.id_lm_adres         = adm.admnr
        and    adm.dat_ingang         <= @vandaag
        and   (adm.dat_einde          >= @vandaag
         or    adm.dat_einde           = null)
```

Monitoring Table Activity

- Monitoring activity on a specific table

```
select SQLText from master..monSysSQLText
where SQLText like ‘%MyTable%’
```

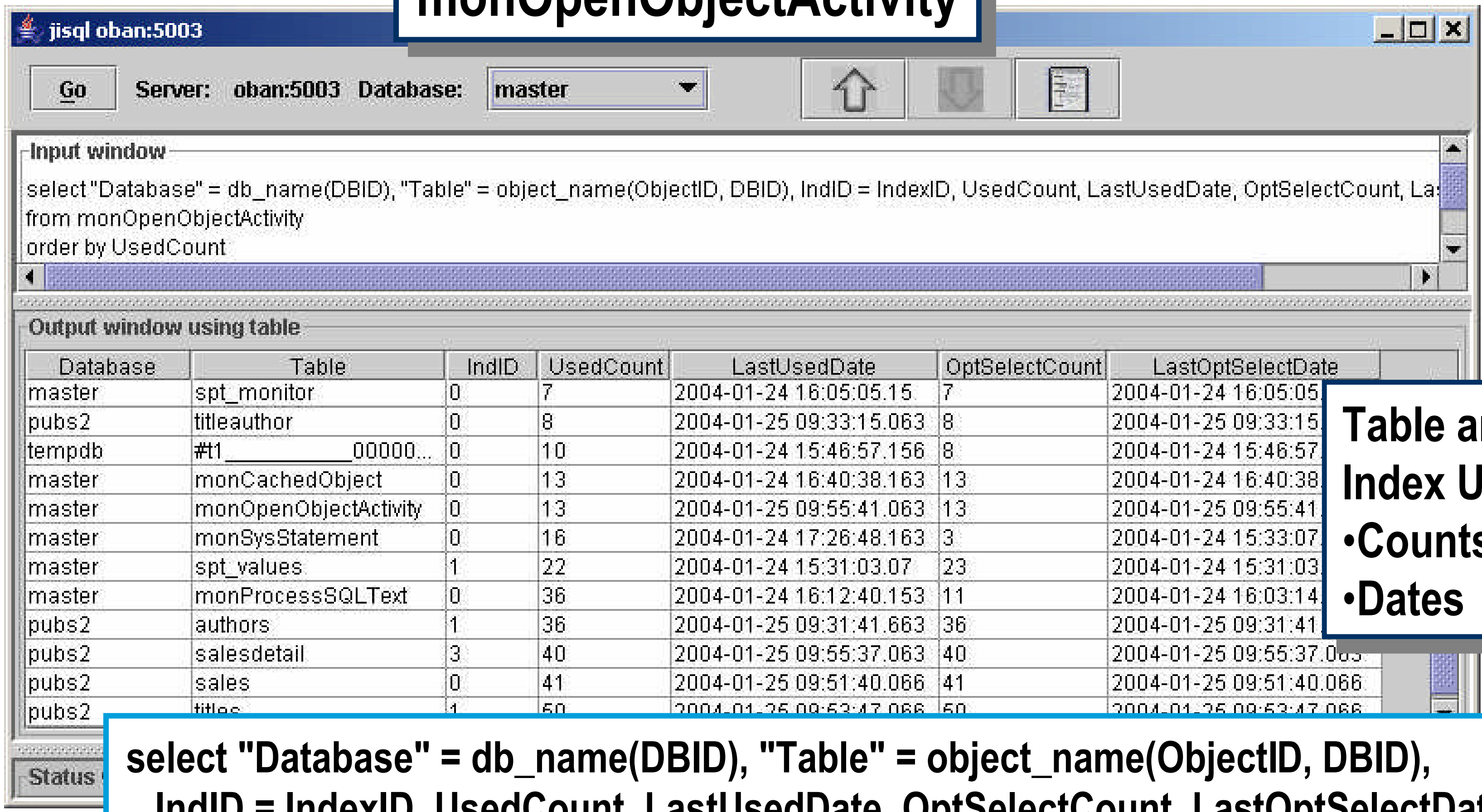
- Also handy for RepServer DBAs:
  - Quick way to figure out exactly which SQL is executed against your replicate DB
  - Especially handy when developing/debugging custom function strings

Monitoring Index Utilization

- Have you ever wanted to see
  - Which indexes are never used?
  - How frequently they are used?
  - How many inserts, deletes, updates, physical or logical I/O they incur?
- monOpenObjectActivity table provides:
  - Table usage count
  - Index usage count
  - Last used dates
  - Physical, logical I/O
  - Row-level insert/delete/update counts
  - Lock wait counts for tables and indexes
- NOTE: Statistics are reset when server is booted or object descriptor is reused in memory.

Monitoring Index Utilization

monOpenObjectActivity



```
select "Database" = db_name(DBID), "Table" = object_name(ObjectID, DBID), IndID = IndexID, UsedCount, LastUsedDate, OptSelectCount, La
from monOpenObjectActivity
order by UsedCount
```

Database	Table	IndID	UsedCount	LastUsedDate	OptSelectCount	LastOptSelectDate
master	spt_monitor	0	7	2004-01-24 16:05:05.15	7	2004-01-24 16:05:05
pubs2	titleauthor	0	8	2004-01-25 09:33:15.063	8	2004-01-25 09:33:15
tempdb	#t1___________00000...	0	10	2004-01-24 15:46:57.156	8	2004-01-24 15:46:57
master	monCachedObject	0	13	2004-01-24 16:40:38.163	13	2004-01-24 16:40:38
master	monOpenObjectActivity	0	13	2004-01-25 09:55:41.063	13	2004-01-25 09:55:41
master	monSysStatement	0	16	2004-01-24 17:26:48.163	3	2004-01-24 15:33:07
master	spt_values	1	22	2004-01-24 15:31:03.07	23	2004-01-24 15:31:03
master	monProcessSQLText	0	36	2004-01-24 16:12:40.153	11	2004-01-24 16:03:14
pubs2	authors	1	36	2004-01-25 09:31:41.663	36	2004-01-25 09:31:41
pubs2	salesdetail	3	40	2004-01-25 09:55:37.063	40	2004-01-25 09:55:37.063
pubs2	sales	0	41	2004-01-25 09:51:40.066	41	2004-01-25 09:51:40.066
pubs2	titles	1	50	2004-01-25 09:53:47.066	50	2004-01-25 09:53:47.066

Table and Index Usage
- Counts
- Dates

```
select "Database" = db_name(DBID), "Table" = object_name(ObjectID, DBID),
  IndID = IndexID, UsedCount, LastUsedDate, OptSelectCount, LastOptSelectDate
from master..monOpenObjectActivity
order by UsedCount
```

Monitoring Index Utilization: Unused Indexes

Find Indexes in a Database that have not been used since the server was started

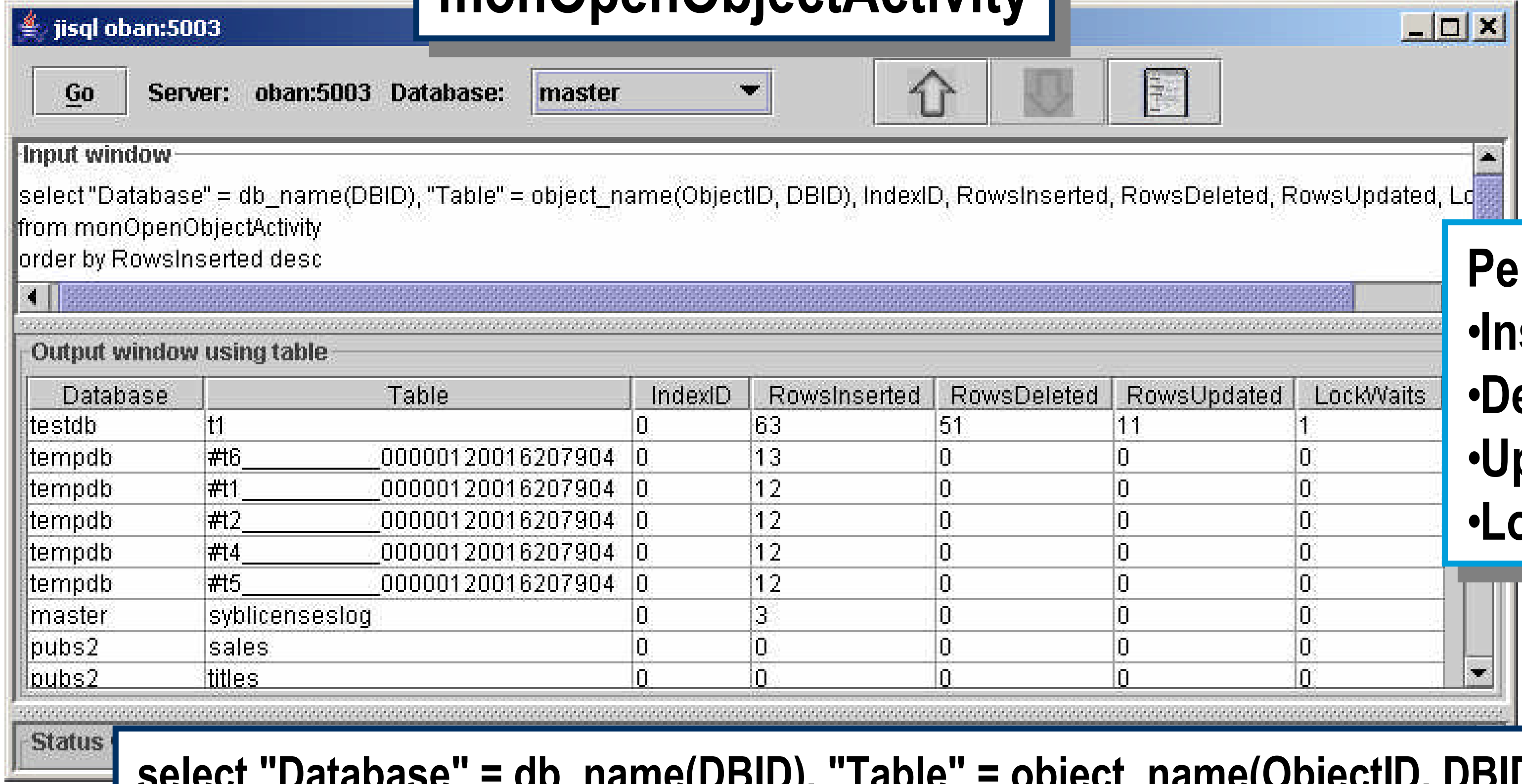
```
select DB = convert(char(20), db_name()),
  TableName = convert(char(20), object_name(i.id, db_id())),
  IndexName = convert(char(20),i.name),
  IndID = i.indid
from master..monOpenObjectActivity a,
  sysindexes i
where a.ObjectID =* i.id
  and a.IndexID =* i.indid
  and (a.UsedCount = 0 or a.UsedCount is NULL)
  and i.indid > 0
  and object_name(i.id, db_id()) not like "sys%"
order by 2, 4 asc
```

Outer join finds all indexes

Report indexes that Have not been used

Monitoring Table Usage

monOpenObjectActivity



Server: oban:5003 Database: master

```
select "Database" = db_name(DBID), "Table" = object_name(ObjectID, DBID), IndexID, RowsInserted, RowsDeleted, RowsUpdated, Lc
from monOpenObjectActivity
order by RowsInserted desc
```

Database	Table	IndexID	RowsInserted	RowsDeleted	RowsUpdated	LockWaits
testdb	t1	0	63	51	11	1
tempdb	#t6____________00000120016207904	0	13	0	0	0
tempdb	#t1____________00000120016207904	0	12	0	0	0
tempdb	#t2____________00000120016207904	0	12	0	0	0
tempdb	#t4____________00000120016207904	0	12	0	0	0
tempdb	#t5____________00000120016207904	0	12	0	0	0
master	syblicenseslog	0	3	0	0	0
pubs2	sales	0	0	0	0	0
pubs2	titles	0	0	0	0	0

Per Table
- Inserts
- Deletes
- Updates
- Lock Waits

```
select "Database" = db_name(DBID), "Table" = object_name(ObjectID, DBID),
   IndexID, RowsInserted, RowsDeleted, RowsUpdated, LockWaits
from monOpenObjectActivity
order by RowsInserted desc
```

Identifying ‘Hot’ Tables

- What makes a table “hot”?
  - Logical reads?
  - Physical reads?
  - Number of queries?
  - Lock usage?
- monOpenObjectActivity reports a number of measures of table and index activity
- Example

```
select * into #t
from  master..monOpenObjectActivity
go

select TableName = object_name(ObjectID, DBID), IndexID,
   LogicalReads, PhysicalReads, Operations, LockWaits
from #t
order by 3 desc
go
```

Identifying 'Hot' Tables

```
TableName                      IndexID LogReads PhysReads Operations LockWaits
------------------------------ ------- -------- --------- ---------- ---------
products_tb                          0   282294      9043        609        97
products_tb                          2    36450         0          0         0
cust_tab                             0    12315         0         17         2
cust_tab                             2      239         0          0         0
```

Understanding and Using Historical Tables

Using Historical Tables

- Which MDA tables are “historical” tables?
- What are Historical Tables?
- How do they work?
- What is the correct size to configure them?
- Archiving historical table data
- Tips on using historical tables

Which Tables are Historical Tables?

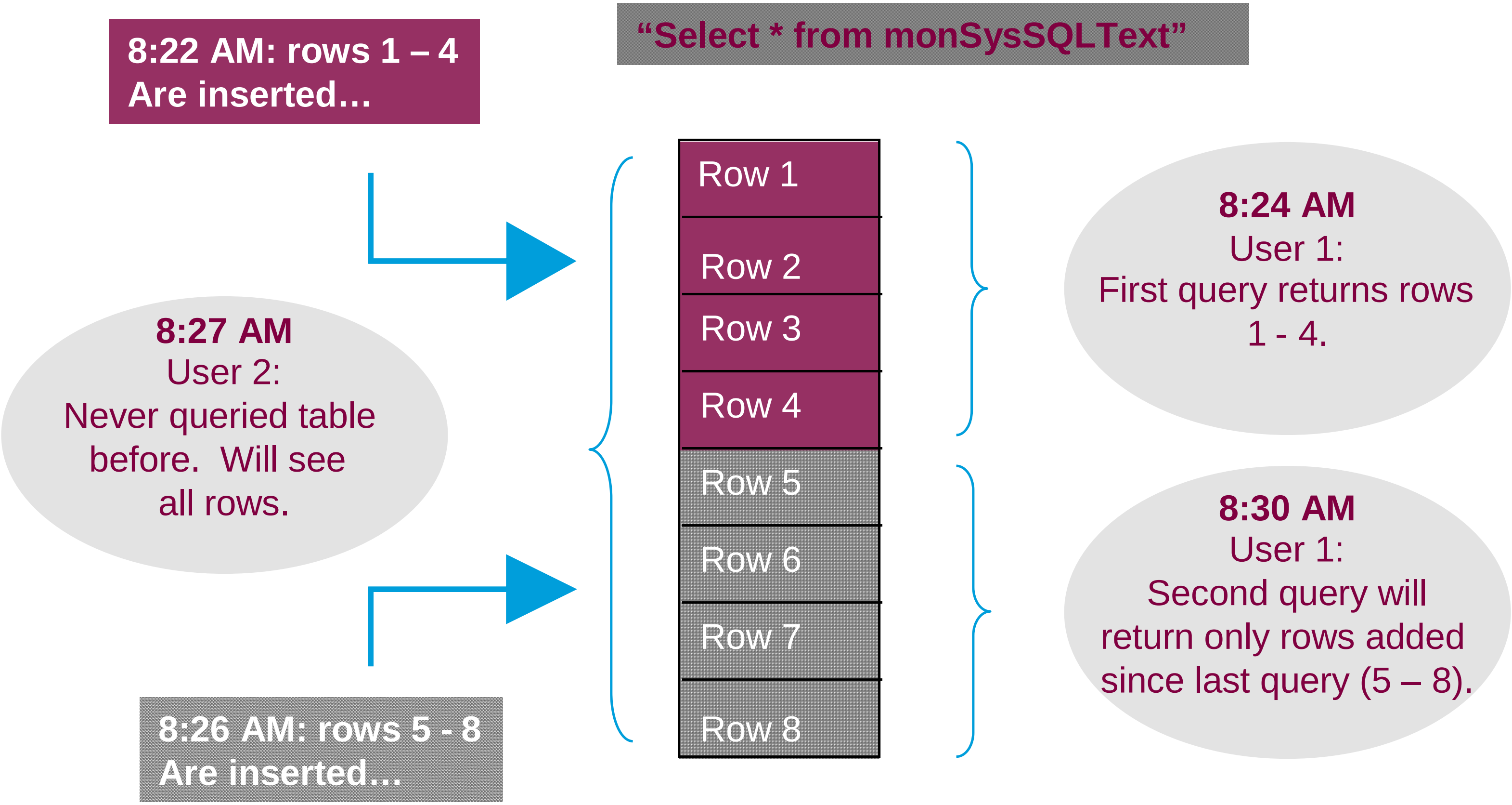
- monSysSQLText
  - Records every SQL command executed on the server
- monSysPlanText
  - Records the Query Plan for every SQL command executed on the server
- monSysStatement
  - Reports the statistics for every statement within every query, batch, stored procedure, trigger, etc. executed on the server
- monErrorLog
  - Records every row written to the server errorlog
- monDeadLock
  - Records information on every deadlock that occurs on the server

What are Historical Tables?

- The historical MDA tables contain a record of “events” within the ASE
  - E.g., SQL submitted for a query, a statement executed within a batch, error message added to the errorlog
- The data for these tables is stored in memory in fixed-sized arrays
  - Size is configurable using sp_configure
- Data in Historical tables is transient
  - The arrays are managed as “ring buffers”: After the last entry in the array is written the first entry will be overwritten
- Historical tables are “stateful.”
  - The ASE remembers which records a process has already seen
  - Subsequent queries on same table will return only new records
- Why are they stateful?
  - This allows applications to accurately collect or “drain” the rows in these tables without finding duplicates

Queries on Historical Tables

- **The ASE maintains the connection's *currency* in the MDA table**
- **Currency is reset for each new connection**



"Select * from monSysSQLText"

8:22 AM: rows 1 – 4 Are inserted…

Row 1

Row 2

Row 3

Row 4

Row 5

Row 6

Row 7

Row 8

8:24 AM
User 1:
First query returns rows 1 - 4.

8:27 AM
User 2:
Never queried table before. Will see all rows.

8:30 AM
User 1:
Second query will return only rows added since last query (5 – 8).

8:26 AM: rows 5 - 8 Are inserted…

Setting the Size of Historical Tables

- Depends on which table you are configuring
- These sp_configure parameters determine the number of rows in the historical tables
  - `errorlog pipe max messages`
  - `plan text pipe max messages`
  - `sql text pipe max messages`
  - `statement pipe max messages`
  - `deadlock pipe max messages`
- The value of the parameter is the number of rows *per engine*
- Correct size depends on
  - Rate at which rows are written to table
  - Frequency with which queries will be run against the table

Rate x Frequency = Size
E.g.: 5000/min x 5 min = 25000

- For example:
  - 2 engines
  - 5000 rows per minute per engine
  - Select * from monSysStatement every 5 minutes
  - Statement pipe max messages should be greater than or equal to 25000
  - Result set size??? (50000 rows!)

Reasonable size on busy system??
Could be >> 100000

- Errorlog and deadlock pipes are usually much smaller than plan text, sql text and statement pipes

Tips on Using Historical Tables

- Do not use in subqueries or joins
- Save contents of tables to an archive table or database for analysis
- When collecting long-term data, archive data on a regular basis and size tables to avoid data loss
- How do you know whether the table for the buffer has wrapped?
  - If # of rows returned = size of buffer * # of engines
  - In other words, if you get the entire size of the buffer, some rows were probably lost
  - Only a “rule of thumb”
  - Currently, it is not possible to determine how many rows were lost

Archiving Historical Table Data

Archiving Historical Table Data

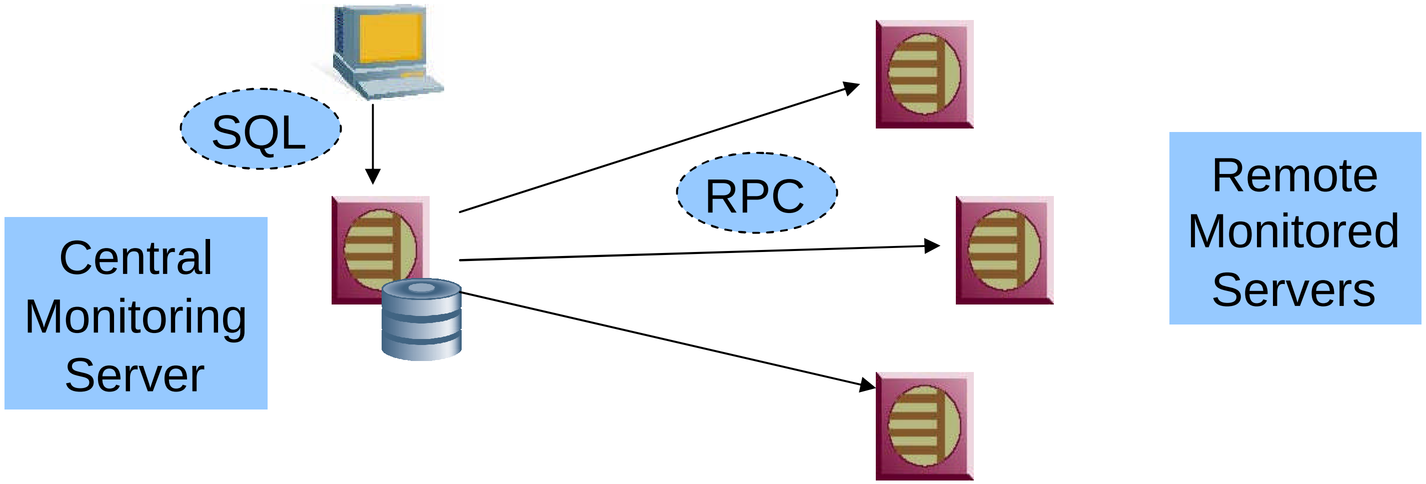
- Why is an historical archive useful?
- Because data in historical tables is transient
  - Capture data for later or detailed analysis
- Because repeated queries on historical tables will not return the same rows
  - Even in subqueries or joins
  - Makes analytical queries directly against MDA tables difficult
- Solution
- Data from historical tables should be moved to separate storage for analysis
- Create a monitoring data repository for historical diagnostics or capacity planning

Archiving Historical Table Data

- A possible approach: a ‘collector’ stored proc which frequently extracts data from the MDA tables
  - ‘sp_mda_collect‘
    - sp_mda_collect ‘start’ **[**, ‘hh:mm:ss’ **] -- runs in a loop (default interval = 30 sec.)**
    - sp_mda_collect ‘stop’ **-- run from a different session, stops the original procedure**
    - sp_mda_collect ‘status’ **-- displays #rows saved in archive tables**
    - **(download from** www.sypron.nl/mda)
  - Uses a separate database to collect the historical data in permanent tables
  - The permanent tables have the same layout as the historical MDA tables
  - Added a composite unique index **with ignore_dup_key** on key columns (SPID, KPID, etc.) to filter out duplicates (in case the proc needs to be restarted…)

Enterprise Monitoring Repository and Center

- To access MDA tables from a remote server
  - Create the MDA proxy tables on a central server
  - Map MDA proxy tables to each monitored server
- Reduces load on monitored ASE servers
- Provides central source of monitoring data for your enterprise
- Allows easy archiving of enterprise data to permanent storage in database on repository server



Creating an Enterprise Monitoring Center

- Create monitoring database on central server
- Copy and edit installmontables script
  - Two options:
    - Create separate monitoring database for each monitored server
    - Add server name to MDA table names to create unique table names for each server within a single database
- Set the use database command to use the correct database
- Use sp_addserver to register the remote monitored server with the repository server
  - Use sp_addexternlogin to coordinate login credentials if needed
- Change the “loopback” server name to the remote server name of the monitored server in your central server
- Copy object ID’s and names from remote servers on a regular basis

Modifying installmontables Script

- **Creating MDA proxy tables in a separate database for each monitored server**

```
…
…
use monitor_svrtest1
go
…
…

create existing table
monProcedureCache (
        Requests          int,
        Loads             int,
        Writes            int,
        Stalls            int,
)
external procedure
at “svrtest1...$monProcedureCache"
go
```

Use a separate database for each Monitored server

Proxy table points to monitored server

Modifying installmontables Script

- **Creating MDA proxy tables in a single database for all monitored server**

```
…
…
use monitordb
go
…
…

create existing table
monProcedureCache_svrtest1 (
        Requests         int,
        Loads            int,
        Writes           int,
        Stalls           int,
)
external procedure
at “svrtest1...$monProcedureCache"
go
```

Database in which all proxy tables
Will be created

Unique table name constructed by a
Appending server name

Proxy table points to monitored
server

Performance Impact of MDA Tables

Performance Impact of MDA Tables

- Two questions
  - Impact of data collection?
  - Impact of querying MDA tables?
- General performance impact: 5% or less
- Depends on a number of factors
  - Configuration of server (e.g., number of engines, memory size, processor speed)
  - Load on server
  - Configuration of Monitoring parameters
- Different monitoring configuration settings have different performance impacts
- Fully enabling all options will have greatest impact

Performance Impact: Configuration Settings

- Lowest impact
  - Enable monitoring with no other options

⇨ Tables enabled

  - monEngine, monDataCache, monProcedureCache, monOpenDatabases, monSysWorkerThread, monNetworkIO, monLocks, monCachePool, monIOQueue, monDeviceIO, monProcessWorkerThread, monProcessNetIO

- Medium Impact Parameters
  - wait event timing
  - plan text pipe active
  - sql text pipe active
  - errorlog pipe active
  - deadlock pipe active

⇨ Tables enabled

  - monSysWaits, monProcessWaits
  - monSysPlanText, monSysSQLText, monErrorLog, monDeadLock

- Highest Impact Parameters
  - statement pipe active
  - statement statistics active
  - per object statistics
  - statement pipe active

⇨ Tables enabled

  - monOpenObjectActivity, monProcessObject, monProcessActivity, monSysStatement, monProcessStatement

** Note: Actual impact depends on system load patterns and configuration

Performance Improvements in ASE 15.0 ESD#2

- Up to ASE 15.0 ESD#1:

```
create existing table monLocks (
...
)
external procedure
at “loopback...$monLocks"
go
```

'external procedure':
When querying the MDA table, ASE creates a 'backdoor' client connection back into ASE itself
('loopback' = current ASE server)

- As of ASE 15.0 ESD#2:

```
create existing table monLocks (
...
)
materialized
at "$monLocks"
go
```

New option 'materialized'

Does not require the additional 'backdoor' client connection anymore

Less overhead, better performance:
- no additional connection management
- saves network I/O

Why?
- Reduces resource usage
- Increases query speed

Understanding and Handling Counter Wrap

What is Counter Wrap?

- All MDA counter columns are 32-bit signed integers
  - Maximum value is 2147483647
- When signed integers are incremented above maximum value they become negative
  - 2147483647 + 1 => - 2147483646
- Internal adjustments prevent MDA counter values from becoming negative
  - Therefore counter ranges are from 0 to 2147483647
- When the ASE increments an MDA counter past the maximum value it will return to 0 and start increasing again

Handling Counter Wrap with Delta Values

- If counter has wrapped, add difference between start value and maximum value + 1 to the current value of the counter

```
Select CacheName,
    CacheSearches =
    case
      when e.CacheSearches < s.CacheSeaches
      then
        (2147483648 - s.CacheSearches) + e.CacheSearches)
      else
        (e.CacheSearches – s.CacheSearches)
    end
    from #cacheStart s, #cacheEnd e
    where s.CacheID = e.CacheID
```

- Again: As long as change in counter values is <= 2147483647, delta values will be accurate

Which MDA Table Columns Can Wrap?

- Not all MDA columns are likely to wrap
  - Some counter values increment slowly
  - Some numeric columns are not counters
- Columns that can wrap pretty quickly
  - **monDataCache**
    - **CacheSearches**
    - **LogicalReads**
  - **monNetworkIO**
    - **BytesSent**
    - **BytesReceived**
  - **monSysWaits**
    - **Waits**
- Others wrap less quickly
  - monEngine.ContextSwitches
  - monNetworkIO.PacketsSent

sp_sysmon, subqueries, joins…

Using MDA Tables and sp_sysmon

- Monitor Counters are a set of counters used by sp_sysmon and Monitor Server
- Some MDA table columns are derived from Monitor Counters
- sp_sysmon resets the value of Monitor Counters when it starts
- This can have an impact on applications using MDA tables or Monitor Server
- MDA table columns that come from Monitor Counters are documented.
  - Attributes column = “counter, reset”

Using MDA Tables and sp_sysmon

- The ASE 12.5.3 release introduced changes to sp_sysmon that allow it to run without clearing monitor counters
- Also enhanced so that when multiple applications are using monitor counters the collection of monitor data will not be terminated until all applications are finished
- It is safe to run sp_sysmon when using the MDA tables as long as sp_sysmon is run with the ‘noclear’ option.

```
> exec sp_sysmon “00:01:00”, noclear
```

Subqueries, Joins and Self-Joins

- Rule of thumb: Don’t use joins or subqueries when querying the MDA tables
- Why? Because the MDA table data is transient and reflects the ASE’s instantaneous state, joins and subqueries may not give the expected result.
  - Sequential queries on same table can give different results
- Because of the currency mechanism, self-joins or subqueries involving one of the historical tables more than once *will not* work.
  - Currency is reset by first query and the same rows will not be seen by the subquery or inner join table
- Solution: Copy MDA table data to a work table or permanent repository when complex analysis is required.

Analyzing Stored Procedure Performance

Analyzing Stored Procedure Performance

- Historical Server provides stored procedure performance information
- MDA tables do not provide a table with historical stored procedure statistics (yet, we’re working on it!)
- The monSysStatement table can be used to report this information

```
select * into #s from master..monSysStatement
where ProcedureID !=0

select ProcName = isnull(object_name(ProcedureID,
   DBID), "UNKNOWN"),
  DBName = isnull(db_name(DBID), "UNKNOWN"),
  ElapsedTime = datediff(ms, min(StartTime),
   max(EndTime))
from #s
group by SPID, DBID, ProcedureID, BatchID
```

Stored Procedure Statistics

```
1> select ProcName = isnull(object_name(ProcedureID, DBID), "UNKNOWN"),
2> DBName = isnull(db_name(DBID), "UNKNOWN"),
3> ElapsedTime = datediff(ms, min(StartTime), max(EndTime))
4> from #s
5> group by SPID, DBID, ProcedureID, BatchID
6> order by 3
7> go

 ProcName                       DBName                         ElapsedTime
 ------------------------------ ------------------------------ -----------
 p_sybbugstatus                 engcomdb                              1096
 sybrev_fetch_revstatus         engcomdb                               983
 p_sybbugstatus                 engcomdb                               923
 p_sybbugstatus                 engcomdb                               836
 p_sybbugstatus                 engcomdb                               683
 p_sybbugstatus                 engcomdb                               620
 p_sybbugstatus                 engcomdb                               586
 p_sybbugstatus                 engcomdb                               543
 p_sybbugstatus                 engcomdb                               533
 p_sybbugstatus                 engcomdb                               526
….
….

0
```

Stored Procedure Performance Averages

Aggregate performance statistics can be derived from the output of the previous query

```
/*
** Build a detail table
*/
select ProcName = isnull(object_name(ProcedureID, DBID),
"UNKNOWN"),
  DBName = isnull(db_name(DBID), "UNKNOWN"),
  ElapsedTime = datediff(ms, min(StartTime), max(EndTime))
into #t1
from master..monSysStatement
group by SPID, DBID, ProcedureID, BatchID
having ProcedureID != 0

/*
** Calculate aggregate values
*/
select ProcName, DBName, "Avg" = avg(ElapsedTime),
  NumExecs = count(*)
from #t1
group by ProcName, DBName
order by 3 desc
```

Stored Procedure Performance Averages

Determine average elapsed time and total executions for stored procedures

```
1> select ProcName, DBName, "AvgElapsed" = avg(ElapsedTime),
2> NumExecs = count(*)
3> from #t1
4> group by ProcName, DBName
5> order by 3 desc
6> go
 ProcName                       Database     AvgElapsed     NumExecs
 ------------------------------ ------------ ----------- -----------
 p_sybbugstatus                 engcomdb             483          32
 sn_temp_filters_qts1           qts_db               330          26
 sy_resolution_insert           qts_db               260          44
 p_sybbugreleasematrix          engcomdb             186          21
 create_sn_subscriptions        qts_db               108           9
 p_sybbugsrelease               engcomdb              91          37
 sn_temp_filters_qts2           qts_db                83           2
 sn_temp_filters_qts4           qts_db                73          11
 sn_get_next_key                qts_db                69           5
 create_sn_filters              qts_db                65           5
…
…
```

Identifying Poorly Performing Statements

Identify statements within stored procedures consuming greater than average elapsed time

```
/*
** Build work table
*/
select ProcName = isnull(object_name(ProcedureID, DBID), "UNKNOWN"),
   DBName = convert(char(15), isnull(db_name(DBID), "UNKNOWN")),
   LineNumber,
   ElapsedTime = datediff(ms, StartTime, EndTime)
into #t1
from master..monSysStatement
where ProcedureID != 0
/*
** Calculate aggregate values and find problematic statements
*/
select ProcName, DBName, LineNumber, "AvgElapsed" = avg(ElapsedTime)
from #t1
group by DBName, ProcName, LineNumber
having avg(ElapsedTime) > (select avg(ElapsedTime) from #t1)
order by 4 desc
```

Statements with > Average CPU Time

```
…

…
 ProcName                       DBName          LineNumber  AvgElapsed
------------------------------ --------------- ----------- -----------
row_update                     qts_db                  614        2160
p_sybbugstatus                 engcomdb                 60         240
row_update                     qts_db                  147          98
row_insert                     qts_db                  308          98
e2_CiMember                    qts_db                   71          77
p_sybbugstatus                 engcomdb                 56          76
sy_addl_case_update            qts_db                  138          70
p_sybbugstatus                 engcomdb                125          69
sybrev_report_newcrs           engcomdb                 48          30
log_activity                   qts_db                  155          18
p_sybbugstatus                 engcomdb                 29          16
p_sybbugstatus                 engcomdb                145          15
log_activity                   qts_db                   90          14

….
….
```

Most Frequently Used Stored Procedures

```
1> select * into #t1 from master..monSysStatement
2> go
1> select ProcedureName = isnull(object_name(ProcedureID, DBID), "UNKNOWN"),
2> "Database" = db_name(DBID),
3> "Execs" = count(*)
4> from #t1
5> where ProcedureID != 0
6> group by DBID, ProcedureID
7> order by 3 desc
8> go
 ProcedureName                  Database                       Execs
 ------------------------------ ------------------------------ -----------
 sp_mltypeset                   empdb                                 8138
 p_sybbugstatus                 engcomdb                               888
 sp_help_rep_agent              sybsystemprocs                         462
 p_sybbugsrelease               engcomdb                               205
 sn_get_next_key                qts_db                                 176
 create_sn_filter_criteria      qts_db                                 162
 create_sn_subscriptions        qts_db                                 136
 create_sn_filters              qts_db                                 120
….
….
```

Recent Enhancements

Enhancements in 12.5.1 and 12.5.2

- 360 columns in 12.5.0.3 (first version of MDA tables)
- 5 new columns in 12.5.1
- 2 new columns in 12.5.2
  - **monProcessObject.TableSize** - table size in Kb
  - **monProcessActivity.WorkTables** - total number of work tables created by the process
- Fixes:
  - milliseconds fixed in **monSysStatement.StartTime** / **EndTime**
  - can be used to determine the exact duration of each statement (resolution = 3 milliseconds)

Enhancements in 12.5.3

- New columns in 12.5.3
  - **monProcessActivity.ServerUserID** - Login ID (suid)
  - **monProcessSQLText.ServerUserID** - Login ID (suid)
  - **monSysSQLText.ServerUserID** - Login ID (suid)
  - **monProcessProcedures.LineNumber** – Line number in a stored procedure
- **New columns in 12.5.3 ESD#2**:
  - 4 new columns in **monEngine**: **Yields**, **DiskIOChecks**, **DiskIOPolled**, **DiskIOCompleted**
- Fixes:
  - Various small bugs were fixed

New tables and columns in 15.0

- 4 new tables in ASE 15.0
  - **monOpenPartitionActivity**
    - similar to **monOpenObjectActivity** but from a partitions perspective
  - **monLicense**
    - shows active license keys
  - **monProcedureCacheMemoryUsage, monProcedureCacheModuleUsage**
    - for engineering usage; no useful info for DBAs
- **Various new columns in** monEngine**,** monCachedObject**,** monProcessObject**,** monOpenObjectActivity

Enhancements in 15.0 ESD#2

- monLocks:
  - Report blocking locks; new columns **BlockedBy** and **BlockingState**
- **New CIS option '**materialized**' for MDA proxy tables (see earlier slide)**
- **Improved names of wait events in** monWaitEventInfo
  - Clarified many wait event names
  - Removed most duplicate wait event names

New columns in 12.5.4 and 15.0 ESD#2

- 3 new columns in 12.5.4 and 15.0 ESD#2
  - **monSysStatement.RowsAffected** – like @@rowcount
  - **monSysStatement.ErrorStatus** – like @@error
  - **monProcessStatement.RowsAffected** – like @@rowcount

Q&A

Your Questions are Welcome

Thanks!

robv@sybase.com

peter.dorfman@sybase.com