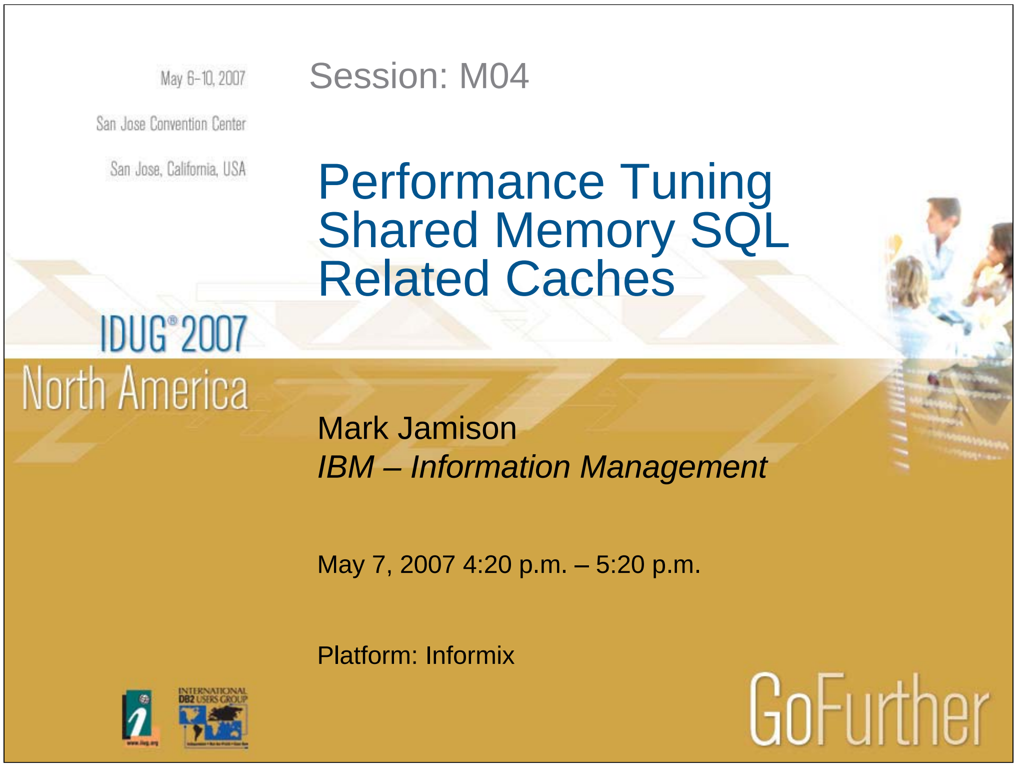


May 6–10, 2007
San Jose Convention Center
San Jose, California, USA

Session: M04

Performance Tuning Shared Memory SQL Related Caches

IDUG® 2007
North America

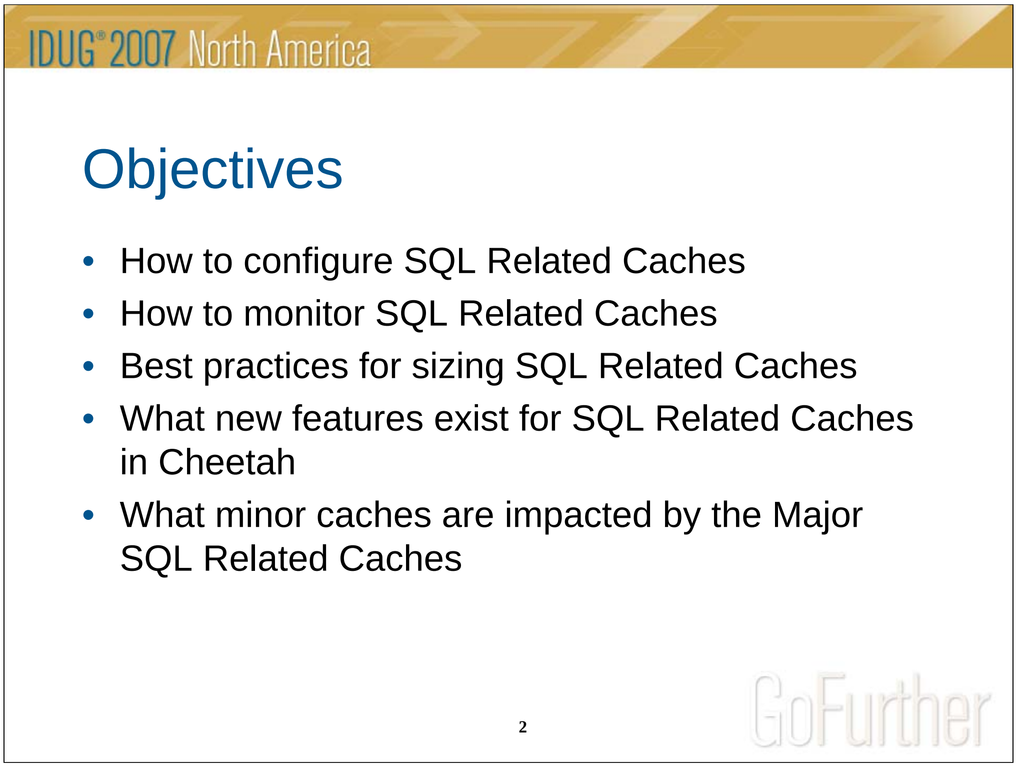
Mark Jamison
IBM – Information Management

May 7, 2007 4:20 p.m. – 5:20 p.m.

Platform: Informix

GoFurther

Performance is a key database requirement, and one of the primary places to improve performance is in monitoring and tuning the various SQL related caches that are available to tune in IDS 10.00. This presentation will discuss the 4 primary caches that can be tuned, best practices for tuning them and the best way to monitor them. Additional methods to monitor these caches in Cheetah will also be discussed.

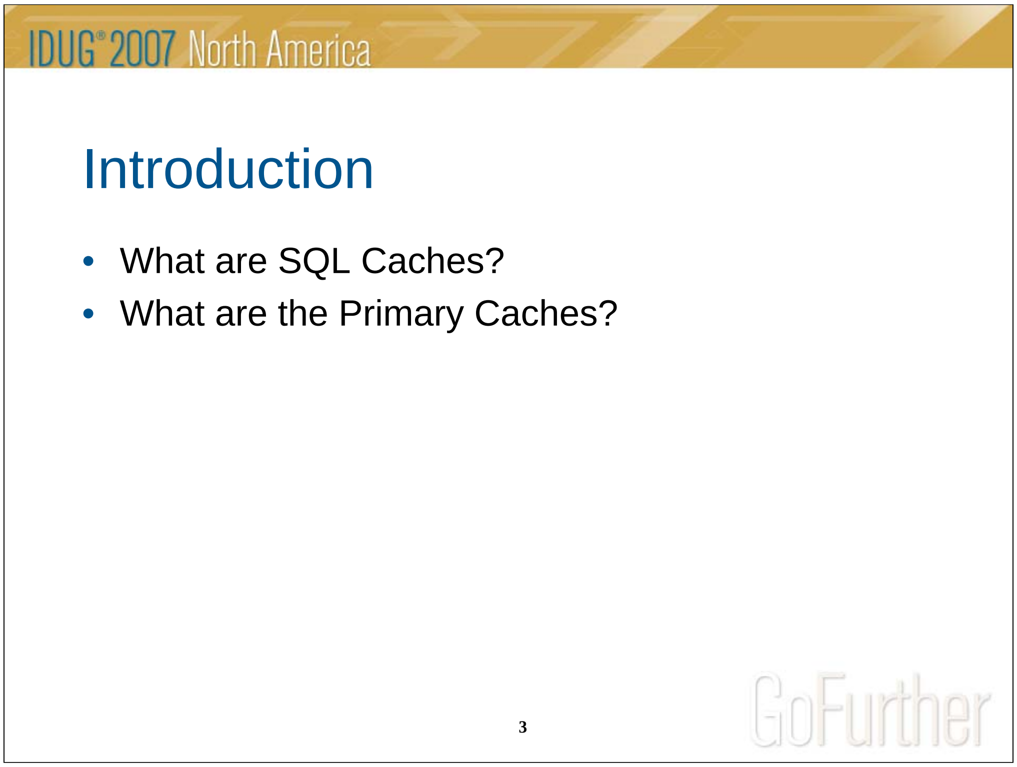


Objectives

- How to configure SQL Related Caches
- How to monitor SQL Related Caches
- Best practices for sizing SQL Related Caches
- What new features exist for SQL Related Caches in Cheetah
- What minor caches are impacted by the Major SQL Related Caches

GoFurther

1. Introduction to IDS SQL Related Caches.
2. The Data Dictionary Cache
    - Configuring the Data Dictionary Cache.
    - Monitoring the Data Dictionary Cache.
    - Cleaning the Data Dictionary Cache.
    - Best practices for sizing the Data Dictionary Cache.
3. The Data Distribution Cache
    - Configuring the Data Distribution Cache.
    - Monitoring the Data Distribution Cache
    - Cleaning the Data Distribution Cache
    - Best practices for Sizing the Data Distribution Cache
    - What other caches are affected by the distribution cache.
4. The UDR Cache
    - Configuring the UDR Cache
    - Monitoring the UDR Cache
    - Cleaning the UDR Cache
    - Best practices for Tuning the UDR Cache
5. The SQL Statement Cache
    - Configuring the SQL Statement Cache
    - Monitoring the SQL Statement Cache.
    - Cleaning the SQL Statement Cache.
    - Best practices for Tuning the SQL Statement Cache.
6. Changes in Cheetah that affect caching parameters.

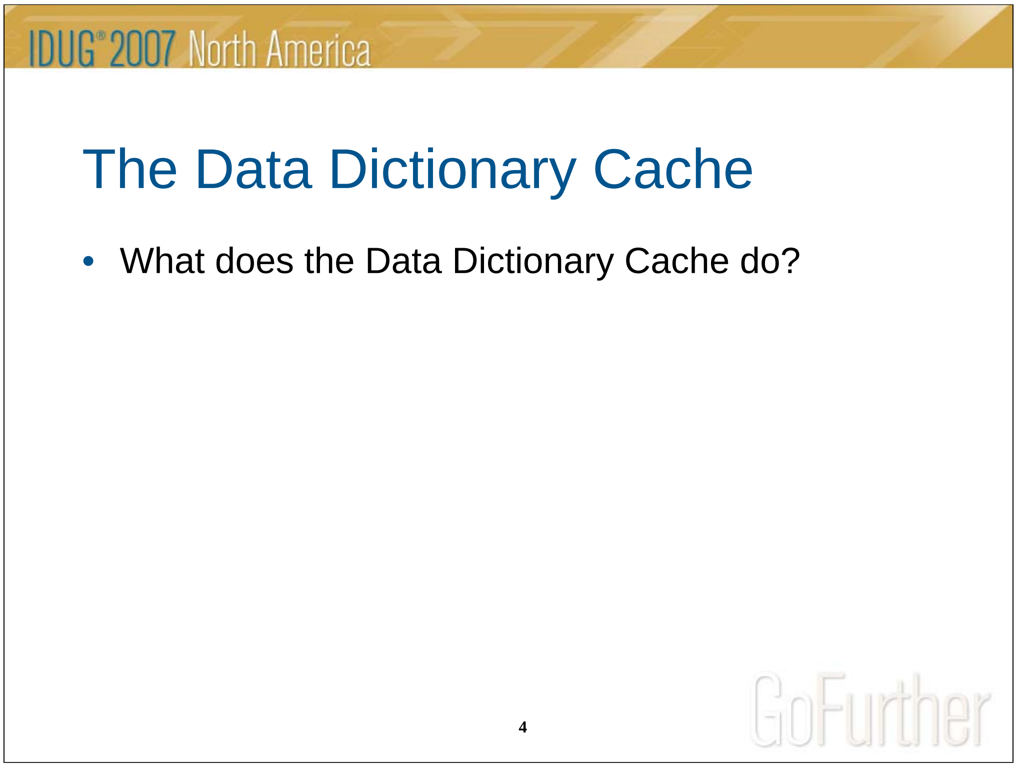


Introduction

- What are SQL Caches?
- What are the Primary Caches?

SQL caches are areas of shared memory used by the Optimizer in order to speed up query and UDR execution, and to speed up the time it takes for the Optimizer to choose a good path. There are 4 Primary caches the Optimizer uses. These 4 are:

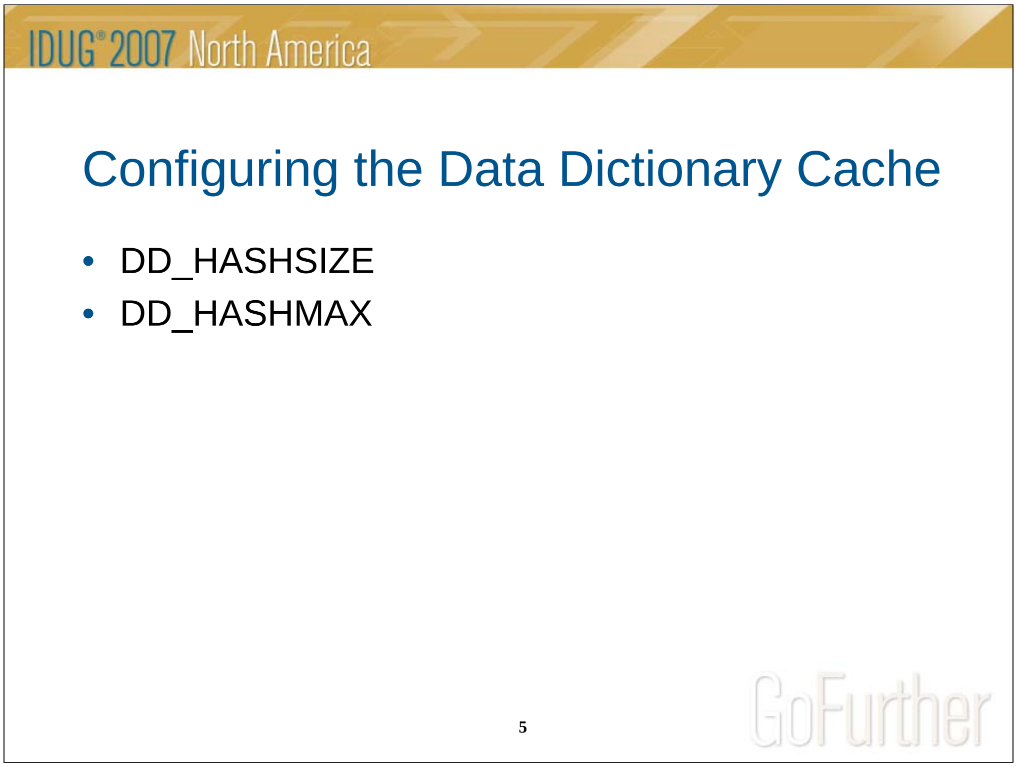
- The Data Dictionary Cache
- The Data Distribution Cache
- The UDR Cache
- The SQL Statement Cache



The Data Dictionary Cache

- What does the Data Dictionary Cache do?

The dictionary cache retrieves and caches information about the tables accessed by IDS. This includes information such as column names, data types, indexes, and extents. Although IDS also places the actual pages for the system catalog tables, also known as partition pages, in the buffer pool, the dictionary cache is always used and generally presents a huge performance advantage for repeated access to the table information.



Configuring the Data Dictionary Cache

- DD_HASHSIZE
- DD_HASHMAX

There are two onconfig parameters that affect the tuning of the Data Dictionary Cache. These two variables are:

DD_HASHSIZE

DD_HASHMAX

DD_HASHSIZE specifies the number of hash buckets or lists in the data dictionary cache, and must be a prime number, while DD_HASHMAX specifies the number of table entries per hash bucket, and is expected to be a non-prime number generally between 4 and 20. The default values for each are as follows:

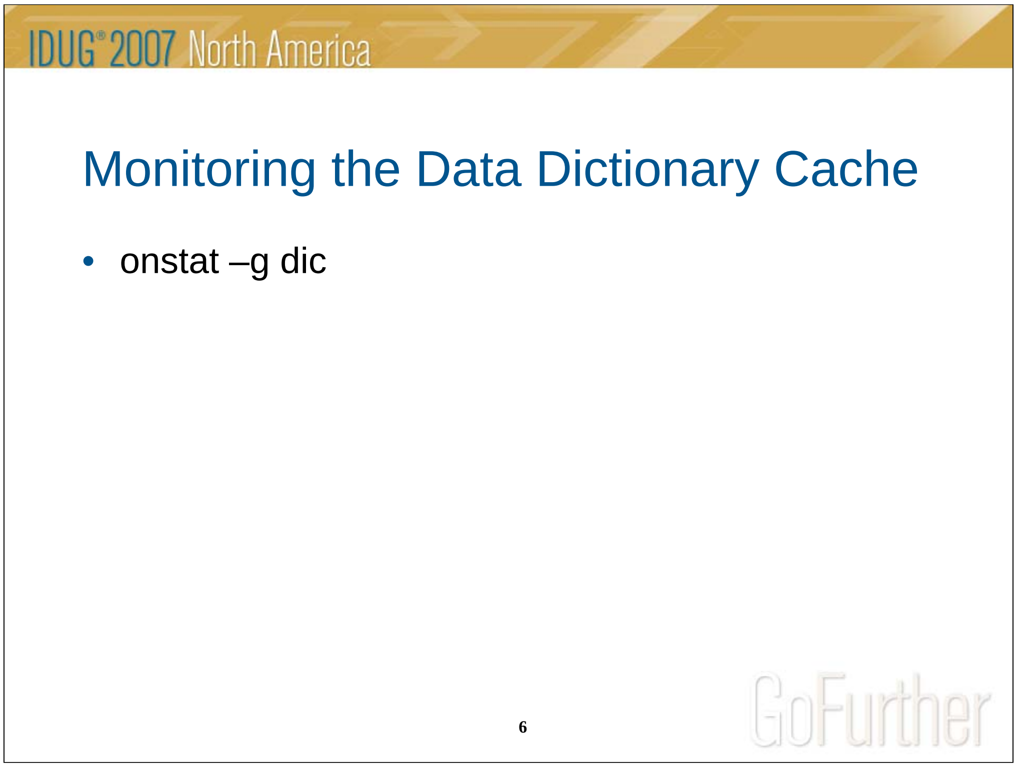
DD_HASHSIZE 31

DD_HASHMAX 10

This means that there are 31 Buckets and each bucket can contain 10 tables, which in turn means that the instance can cache only 310 tables before it runs out of memory, and begins to grow dynamically. If IDS runs out of memory in the dictionary cache to add additional entries, you will have two potential problems:

•Performance: If your Data Dictionary cache is too small for your instance the instance will be constantly attempting to clean the cache to get space back, or constantly growing if all of the tables are still being used. In effect wasting I/O and potentially causing a performance bottleneck.

•Corruption: The potential for shared memory corruption while performing the work listed above.



IDUG® 2007 North America

Monitoring the Data Dictionary Cache

- onstat –g dic

The data dictionary cache can be monitored via the following onstat command:

onstat –g dic

Example:

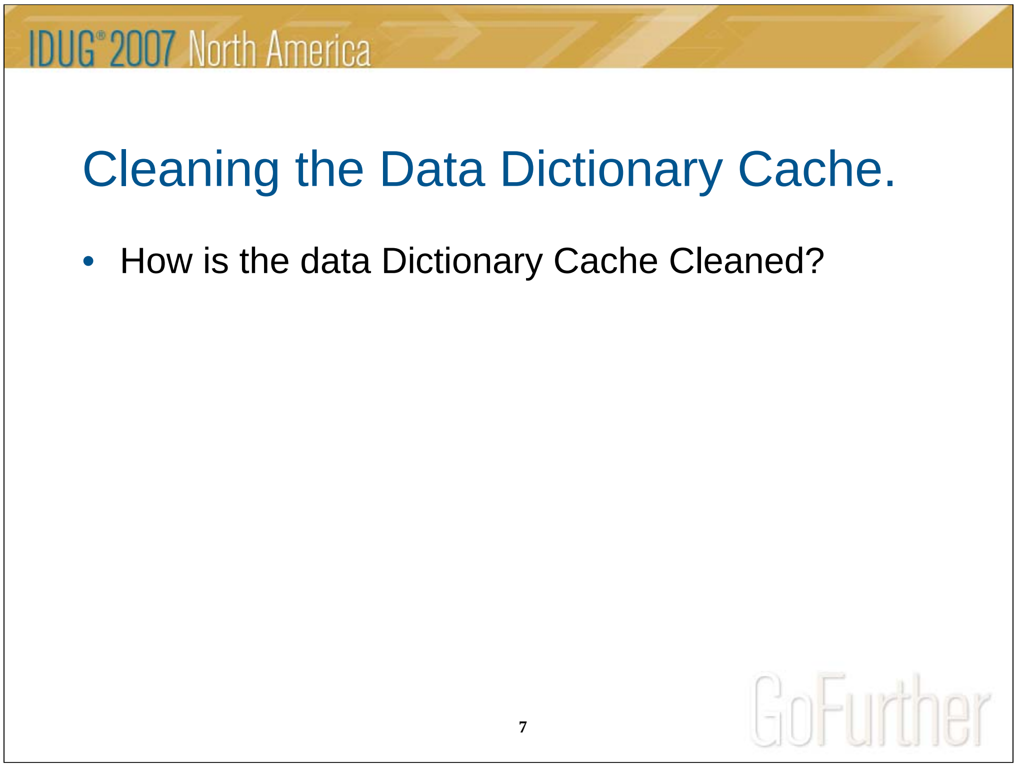
```
(informix) /home/informix > onstat -g dic
IBM Informix Dynamic Server Version 11.10.FC1     -- On-Line -- Up 4 days 22:08:26 -- 42460 Kbytes
Dictionary Cache:  Number of lists: 31, Maximum list size: 10
list#  size  refcnt  dirty?  heapptr         table name
--------------------------------------------------------
   0    2     0      no     c000000003226838   stores_demo@ids10_shm:informix.sysxasourcetypes
              0      no     c00000000312b838   stores_demo@ids10_shm:informix.sysindexes
   1    1     0      no     c0000000031f9038   stores_demo@ids10_shm:informix.sysdefault
  …
  …
  …
  30    1     0      no     c000000003221838   stores_demo@ids10_shm:informix.systracemsgs
Total number of dictionary entries: 62
```

The **onstat –g dic** output has the following fields:

Field	Description
Number of lists	Number of buckets that DD_HASHSIZE specifies
Maximum list size	Number of tables that allowed in each bucket
List #	Bucket number
Size	Number of tables in the bucket
Refcnt	Number of user sessions currently attached to the entry.
Dirty	Flag indicating Data Dictionary entry is no longer valid.
Heapptr	Heap Pointer
Table Name	Name of the table that data dictionary entry describes.

If you need to check the individual Data Dictionary entry, or entries, for a table, you can accomplish this by doing the following:

onstat –g dic <table name>



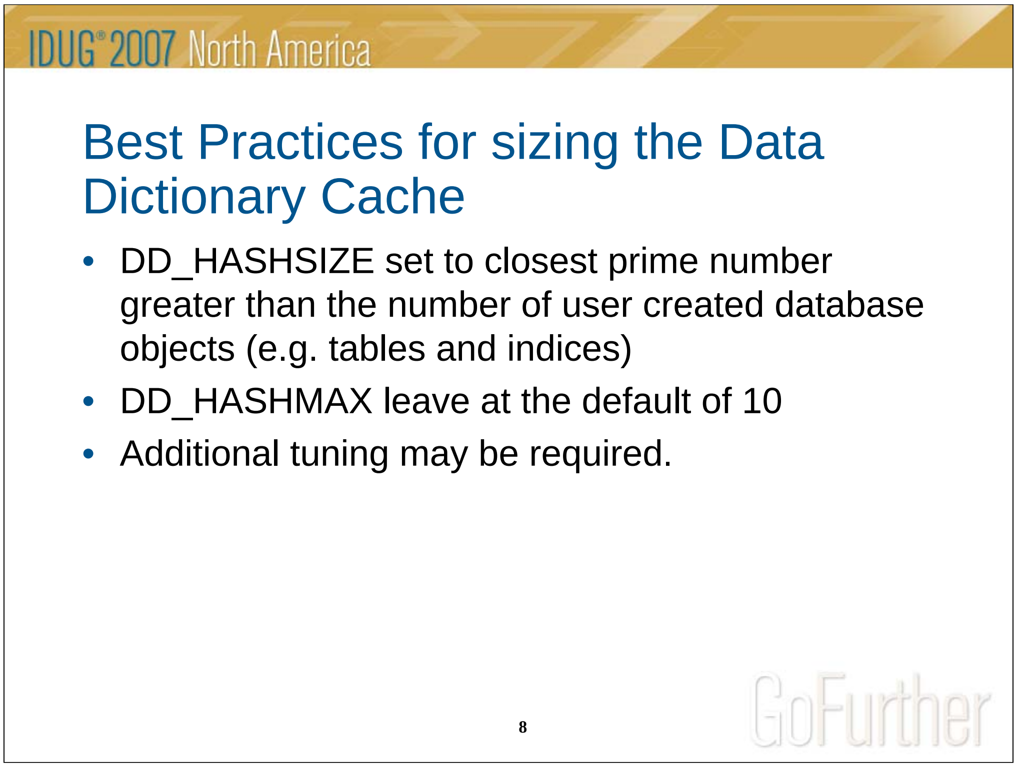
Cleaning the Data Dictionary Cache.

- How is the data Dictionary Cache Cleaned?

The data dictionary cache is cleaned by using a Least Recently Used algorithm. The cache is most commonly cleaned only under the following conditions:

•An entry is freed resulting in a refcnt of 0, and DD_HASHMAX having been exceeded for that list.

•An entry is freed resulting in a refcnt of 0, and the entry is marked as Dirty.



IDUG® 2007 North America

Best Practices for sizing the Data Dictionary Cache

- DD_HASHSIZE set to closest prime number greater than the number of user created database objects (e.g. tables and indices)
- DD_HASHMAX leave at the default of 10
- Additional tuning may be required.

GoFurther

When configuring the value for DD_HASHSIZE, a good rule of thumb is to set DD_HASHSIZE to the prime number closest to the number of user tables for the entire instance. The prime number recommendation is because it is the most efficient number, from a performance standpoint, for the hashing algorithm used to cache Data Dictionary caches. The reason to suggests a prime close to the number of user tables in the instance is because of the desire to limit the number of entries per bucket, making retrieval of a cache entry extremely fast, however any value much higher than the number of user tables starts to make the cost greater than the benefit. Since the DD_HASHSIZE is set to a large value, it then follows that DD_HASHMAX be set to a small value, in fact there is little reason to change the value at all, rather leaving it at the default value of 10. The primary reason not to lower the value is because any time you have a hashing algorithm, you can occasionally have different entries hash to the same location. So if we have a database instance with 2000 user created tables, a good starting point from configuration would be the following values:

DD_HASHSIZE 2003

DD_HASHMAX 10

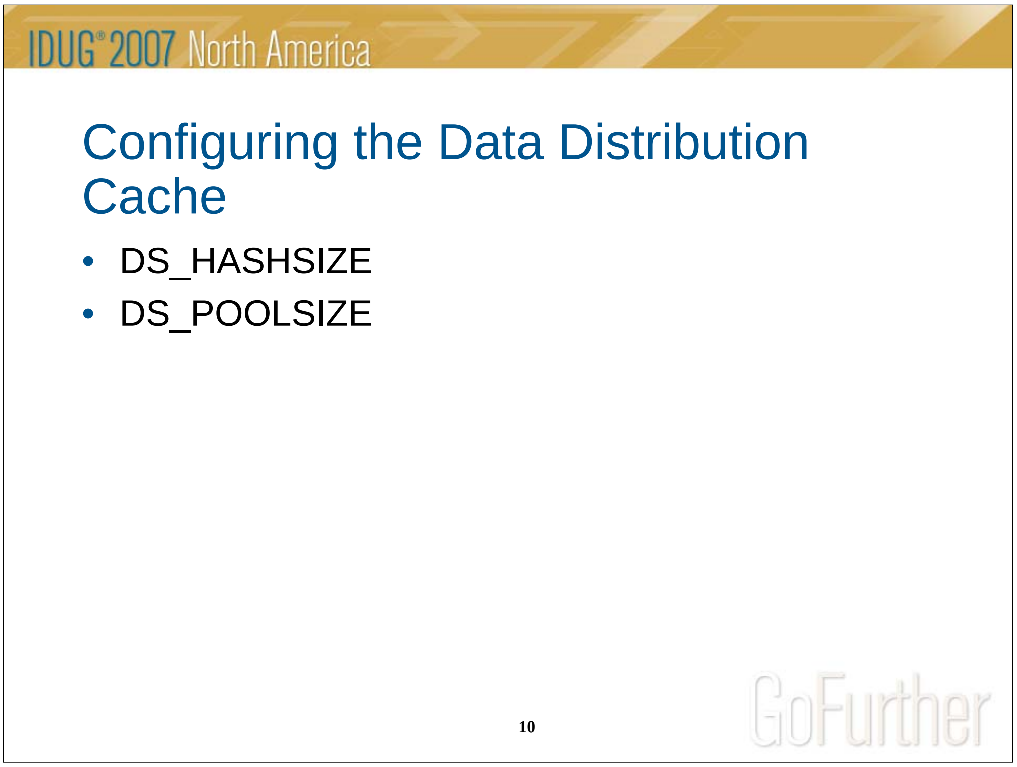
The Data Distribution Cache

- What does the Data Distribution Cache do?

The data distribution cache retrieves and caches distribution information generated by **UPDATE STATISTICS** in the **MEDIUM** or **HIGH** mode. The first time the optimizer accesses these distributions to make cost estimates for a column, IDS retrieves the statistics recorded from the **sysdistrib** system catalog table on disk and places the subsequent information in the cache.

Although IDS places the actual pages for the **sysdistrib** table, in the buffer pool, the distribution cache is always used and presents the following benefits for the distributions info:

- It is organized in a more efficient format
- It is organized for fast retrieval
- It bypasses the overhead of the buffer pool management
- It frees the buffer pool for more important usage, like actual user data and index pages.
- It reduces I/O operations to the system catalog table, reducing the likelihood of contention.



IDUG® 2007 North America

Configuring the Data Distribution Cache

- DS_HASHSIZE
- DS_POOLSIZE

GoFurther

There are two **ONCONFIG** variables used for configuring the Data Distribution cache. These two variables are:

DS_HASHSIZE

DS_POOLSIZE

DS_HASHSIZE specifies the number of hash buckets or lists in the data distribution cache, and must be a prime number, while **DS_POOLSIZE** specifies the total number of entries for all hash buckets, and is expected to be a non-prime multiple of **DS_HASHSIZE**. The default values for each are as follows:

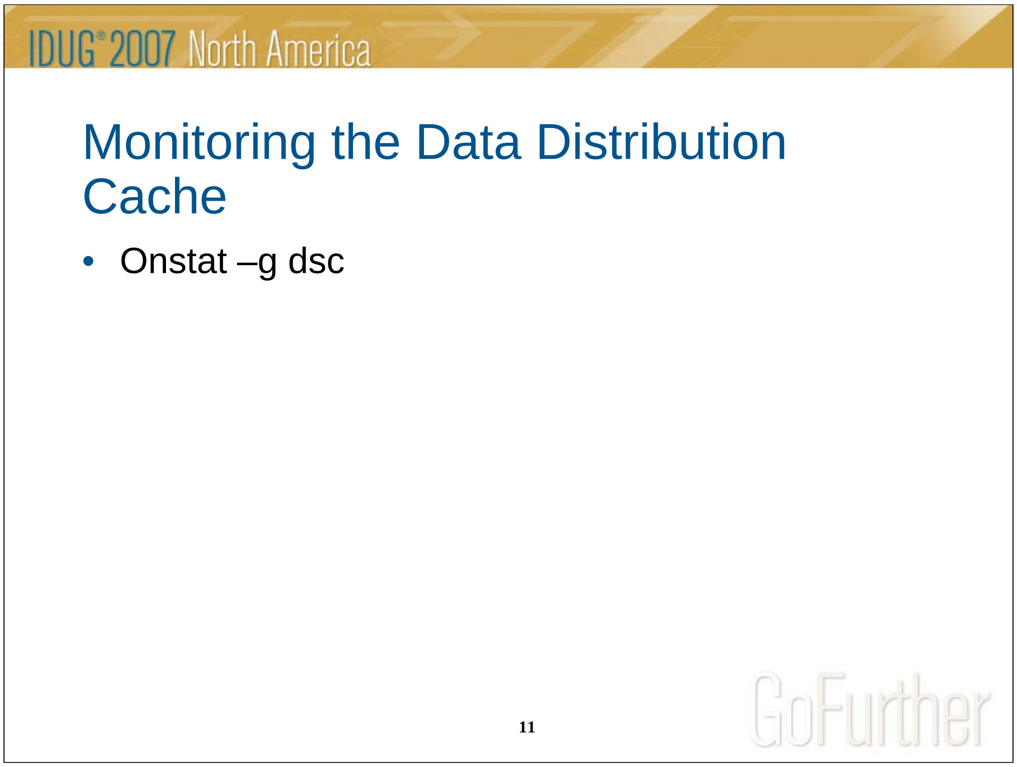
DS_HASHSIZE 31

DS_POOLSIZE 127

This means that there are 31 Buckets and each bucket can contains approximately 4 columns with update statistics high run. If Informix runs out of memory in the distribution cache to add additional entries, you will have two potential problems:

•Performance: If your data distribution cache is too small for your instance the instance will be constantly attempting to clean the cache to get space back. In effect wasting I/O and potentially causing a performance bottleneck.

•Corruption: As with any process in which you do excessive cleaning, there always exists the chance for introduction of shared memory corruption.


IDUG® 2007 North America
GoFurther

Monitoring the Data Distribution Cache

- Onstat –g dsc

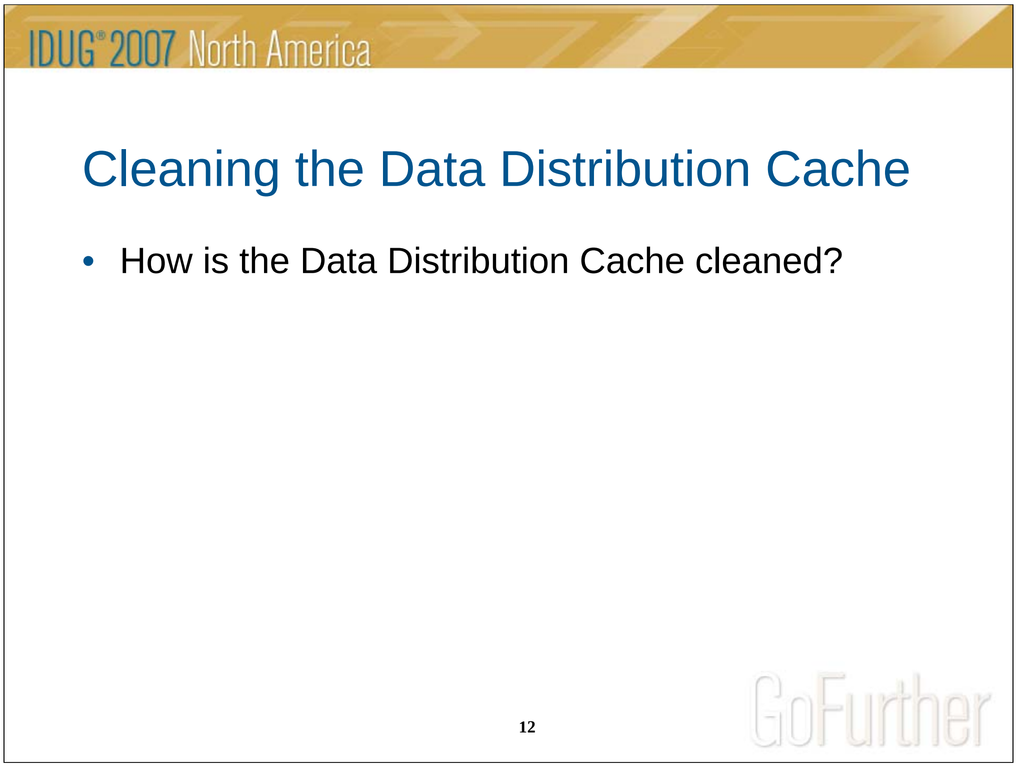
The data distribution cache can be monitored via the following onstat command:
onstat –g dsc

Example:

```
(informix) /home/informix > onstat -g dsc
IBM Informix Dynamic Server Version 11.10.FC1     -- On-Line -- Up 04:05:34 -- 42460 Kbytes
Distribution Cache:
    Number of lists          : 3
    DS_POOLSIZE              : 9
Distribution Cache Entries:
list#  id  ref_cnt   dropped?   heap_ptr       distribution name
-----------------------------------------------------------------------------------------
0      0      0      0      c0000000031ca438    stores_demo:informix.syscolauth.grantee
1      0      0      0      c0000000031bd838    stores_demo:informix.sysdistrib.colno
1      0      0      0      c000000003155038    stores_demo:informix.sysdistrib.tabid
2      0      0      0      c0000000031c9838    stores_demo:informix.syscolauth.colno
2      0      0      0      c0000000031c9438    stores_demo:informix.syscolauth.tabid
Total number of distribution entries: 5.
    Number of entries in use    : 0
```

The **onstat –g dsc** output has the following fields:

Field	Description
Number of lists	Number of buckets that DS_HASHSIZE specifies
DS_POOLSIZE	Number of entries allowed
List #	Bucket number
Id	Not used
Refcnt	Number of user sessions currently attached to the entry.
Diropped	Flag indicating Data Distribution entry is no longer valid.
Heapptr	Heap Pointer
Table Name	Name of the table and column that the data distribution entry describes.



IDUG® 2007 North America

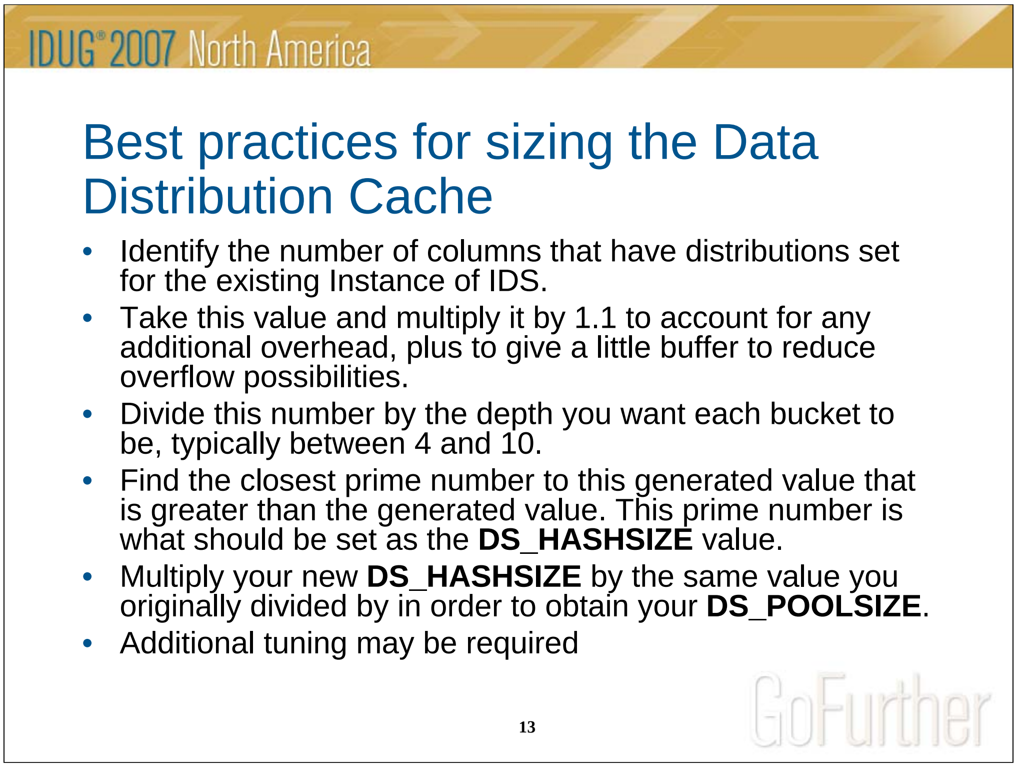
Cleaning the Data Distribution Cache

- How is the Data Distribution Cache cleaned?

12

GoFurther

The cache is cleaned when the Data Distribution cache has consumed more than 75% of the total number of entries cleaning begins. Cleaning continues until such time as there is 50% or less of the Data Distribution cache. The **UPDATE STATISTICS** command will also kick off a cleaning process, removing entries with a refcnt of zero.



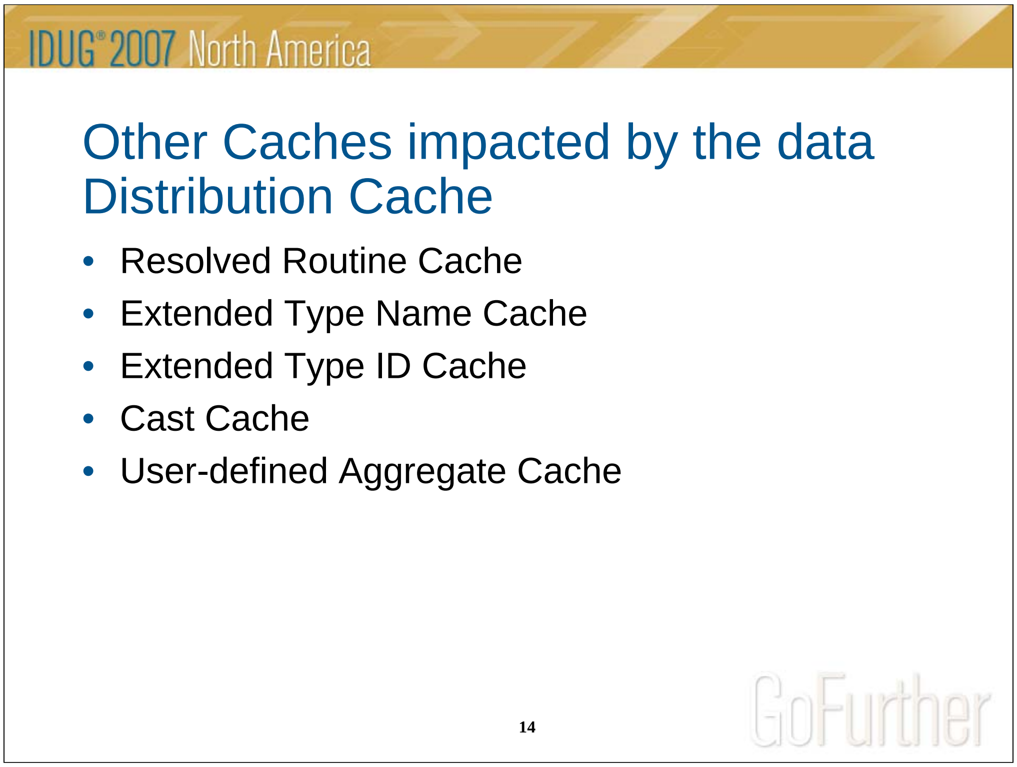
Best practices for sizing the Data Distribution Cache

- Identify the number of columns that have distributions set for the existing Instance of IDS.
- Take this value and multiply it by 1.1 to account for any additional overhead, plus to give a little buffer to reduce overflow possibilities.
- Divide this number by the depth you want each bucket to be, typically between 4 and 10.
- Find the closest prime number to this generated value that is greater than the generated value. This prime number is what should be set as the **DS_HASHSIZE** value.
- Multiply your new **DS_HASHSIZE** by the same value you originally divided by in order to obtain your **DS_POOLSIZE**.
- Additional tuning may be required

I would recommends a ***prime*** *number value for* ***DS_HASHSIZE*** because the insertion of Data Distribution entries is done via a hashing algorithm and a prime number is the most efficient type of number, from a performance standpoint, for a hashing algorithm. The reason to suggests a prime relative to the number of user columns that have distributions generated is because of the desire to limit the number of entries per bucket, making retrieval of a cache entry extremely fast, and eliminating overflow problems. Since the **DS_HASHSIZE** is set to a large value, it then follows that **DS_POOLSIZE** be set to a relatively small multiple of **DS_HASHSIZE**, typically in the neighborhood of 4 to 10 times the value of **DS_HASHSIZE**. ***DS_POOLSIZE*** *should not be limited to a prime number*, because there is no performance gain to be had. The primary reason not to use a multiple lower than 4, is because any time you have a hashing algorithm, you can occasionally have different entries hash to the same location, and you need to allow for some overflow per bucket to account for that possibility. So if we have an instance with 2000 columns which have distribution information a good baseline to start with would be the following:

DS_HASHSIZE 557

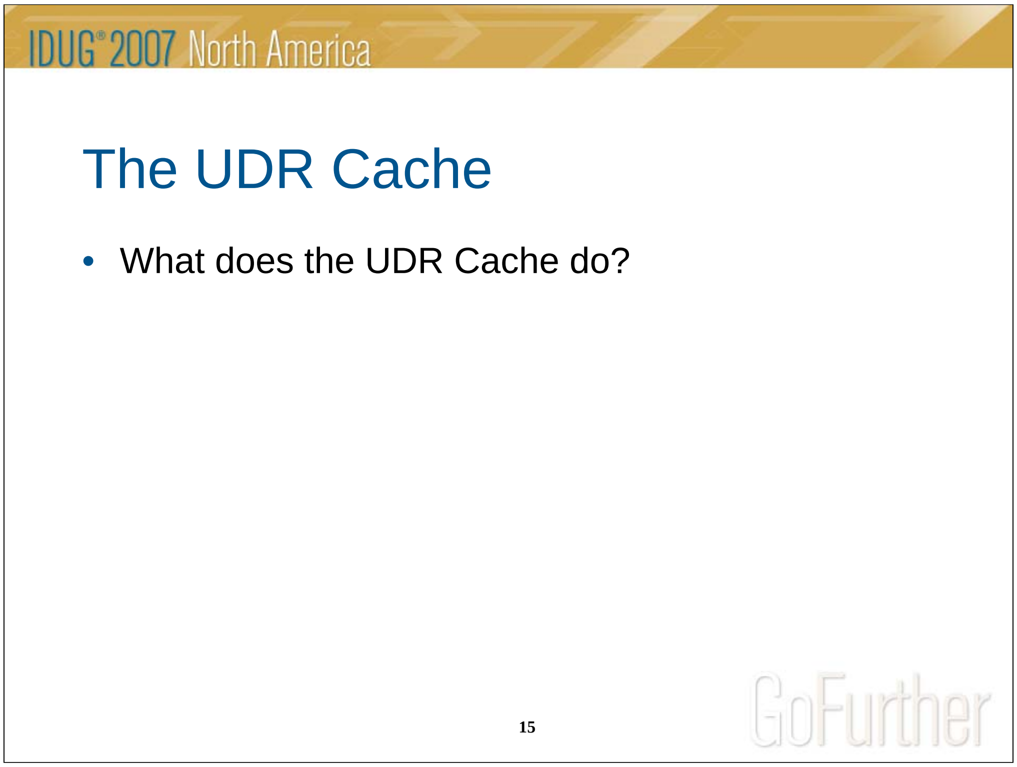
DS_POOLSIZE 2228



Other Caches impacted by the data Distribution Cache

- Resolved Routine Cache
- Extended Type Name Cache
- Extended Type ID Cache
- Cast Cache
- User-defined Aggregate Cache

All of the above are configured by using the values from DS_HASHSIZE and DS_POOLSIZE.

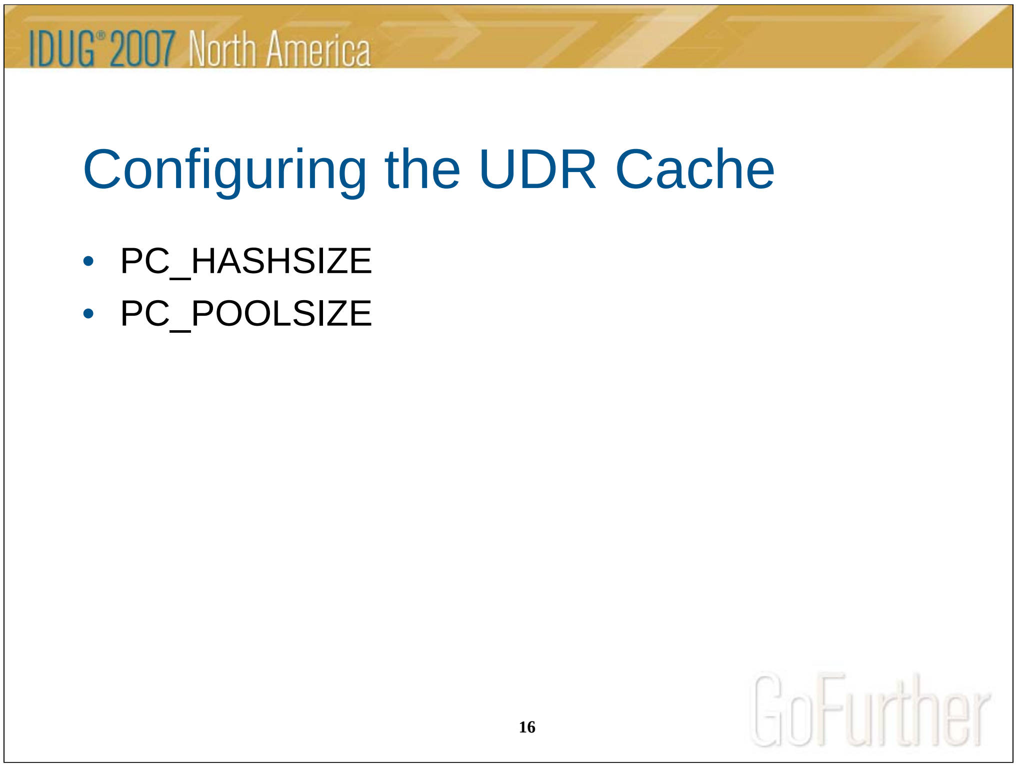


The UDR Cache

- What does the UDR Cache do?

GoFurther

The UDR cache is used to store frequently executed Stored Procedures and Functions (both internal and external). When a session executes an SPL, or function, for the first time, the engine grabs the p-code as found in the system catalog tables. It then converts this p-code into a binary executable, if not already an external executable, and then places the entry into the UDR cache. Subsequent calls of a UDR result in the UDR cache being checked first so that, if possible, the conversion of p-code process can be avoided.



IDUG® 2007 North America

Configuring the UDR Cache

- PC_HASHSIZE
- PC_POOLSIZE

GoFurther

There are two **ONCONFIG** variables used for configuring the UDR cache. These two variables are:

PC_HASHSIZE

PC_POOLSIZE

PC**_HASHSIZE** specifies the number of hash buckets or lists in the UDR cache, and must be a prime number, while **PC_POOLSIZE** specifies the total number of entries for all hash buckets, and is expected to be a non-prime multiple of **PC_HASHSIZE**. The default values for each are as follows:

PC_HASHSIZE 31

PC_POOLSIZE 127

This means that there are 31 Buckets and each bucket can contains approximately 4 UDR’s. If Informix runs out of memory in the UDR cache to add additional entries, you will have two potential problems:

•Performance: If your UDR cache is too small for your instance the instance will be constantly attempting to clean the cache to get space back. In effect wasting I/O and potentially causing a performance bottleneck.

•Corruption: As with any process in which you do excessive cleaning, there always exists the chance for introduction of shared memory corruption.

Monitoring the UDR Cache

- onstat –g prc

GoFurther

The UDR cache can be monitored via the following onstat command:
onstat –g prc

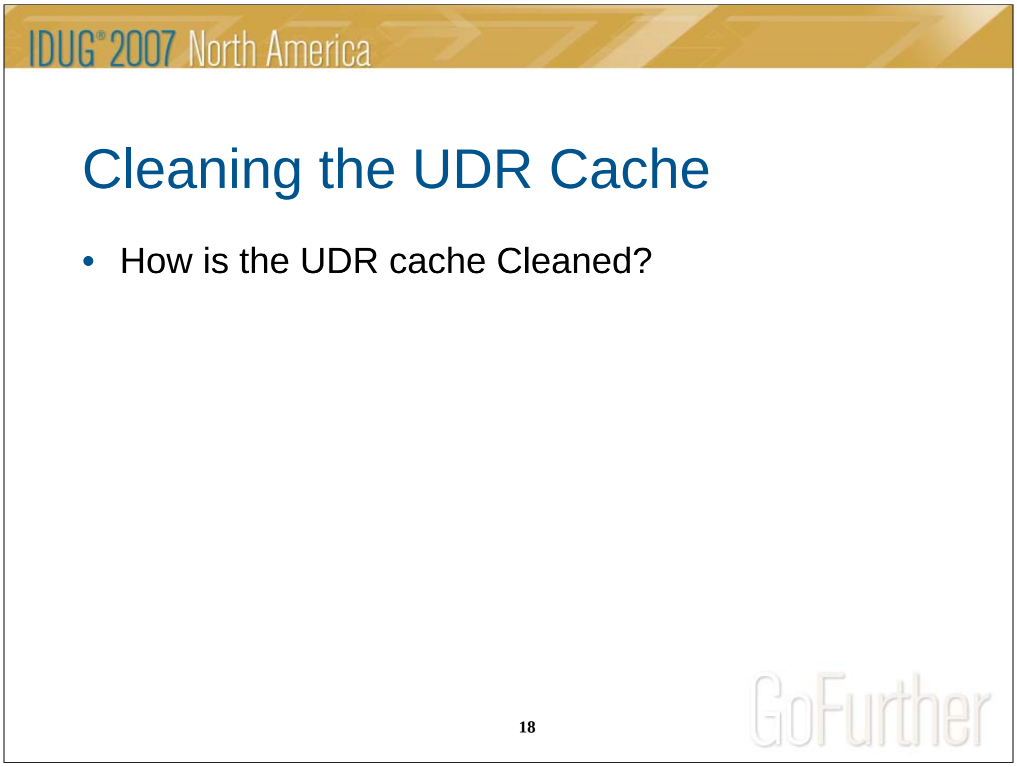
Example:

```
(informix) /usr/informix > onstat -g prc
IBM Informix Dynamic Server Version 11.10.FC1     -- On-Line -- Up 1 days 00:27:48 -- 248928 Kbytes
UDR Cache:
    Number of lists          : 31
    PC_POOLSIZE              : 127
UDR Cache Entries:

list#  id  ref_cnt   dropped?   heap_ptr        udr name
--------------------------------------------------------------
0      27     0       0       c00000001f06f438              sysadmin@cobra1110fc1_shm:.destroy
2      367    0       0        c00000001f083c38              sysadmin@cobra1110fc1_shm:.tabnames_save_diffs
3      133    0       0        c00000001ec0d438              sysadmin@cobra1110fc1_shm:.assign
3      33     0       0       c00000001f106838              sysadmin@cobra1110fc1_shm:.destroy
4      28     0       0       c00000001eba3838              sysadmin@cobra1110fc1_shm:.assign
…
Total number of udr entries: 21.
    Number of entries in use    : 2
```

The **onstat –g prc** output has the following fields:

Field	Description
Number of lists	Number of buckets that PC_HASHSIZE specifies
PC_POOLSIZE	Number of entries allowed
List #	Bucket number
Id	Procedure ID
Refcnt	Number of user sessions currently attached to the entry.
Dropped	Flag indicating Data Distribution entry is no longer valid.
Heap_ptr	Heap Pointer
Udr Name	Name of the UDR.



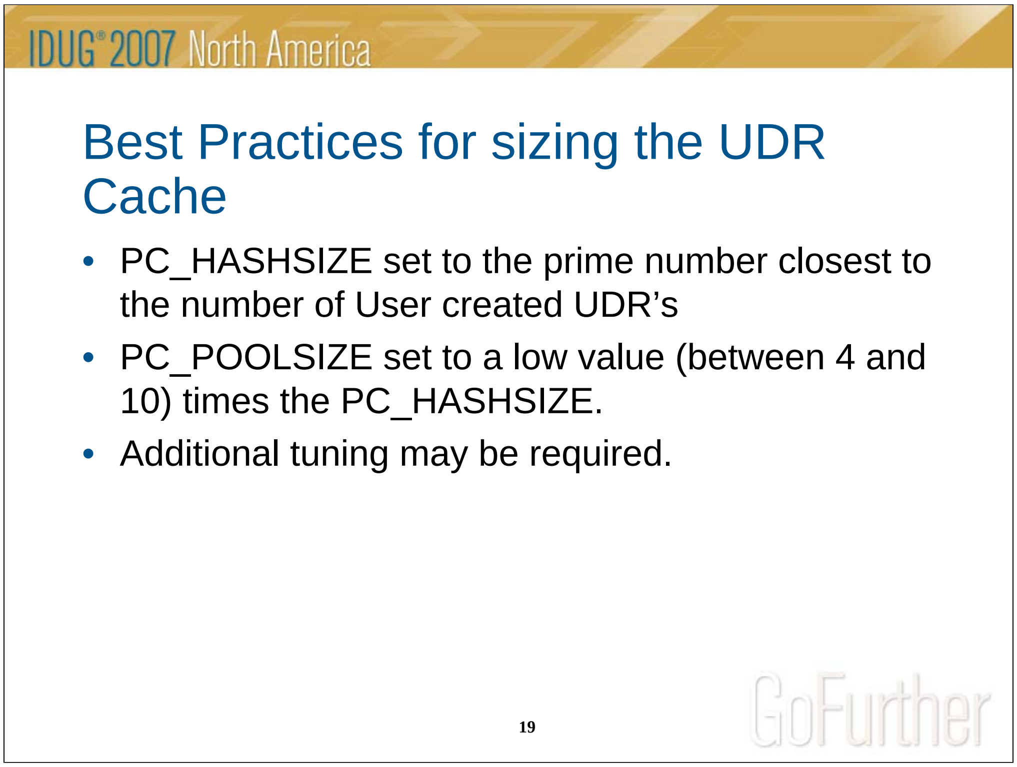
IDUG® 2007 North America

Cleaning the UDR Cache

- How is the UDR cache Cleaned?

GoFurther

The cache is cleaned when the Data UDR cache has consumed more than 75% of the total number of entries cleaning begins. Cleaning continues until such time as there is 50% or less of the UDR cache. The **UPDATE STATISTICS** command will also kick off a cleaning process, removing entries with a refcnt of zero.



Best Practices for sizing the UDR Cache

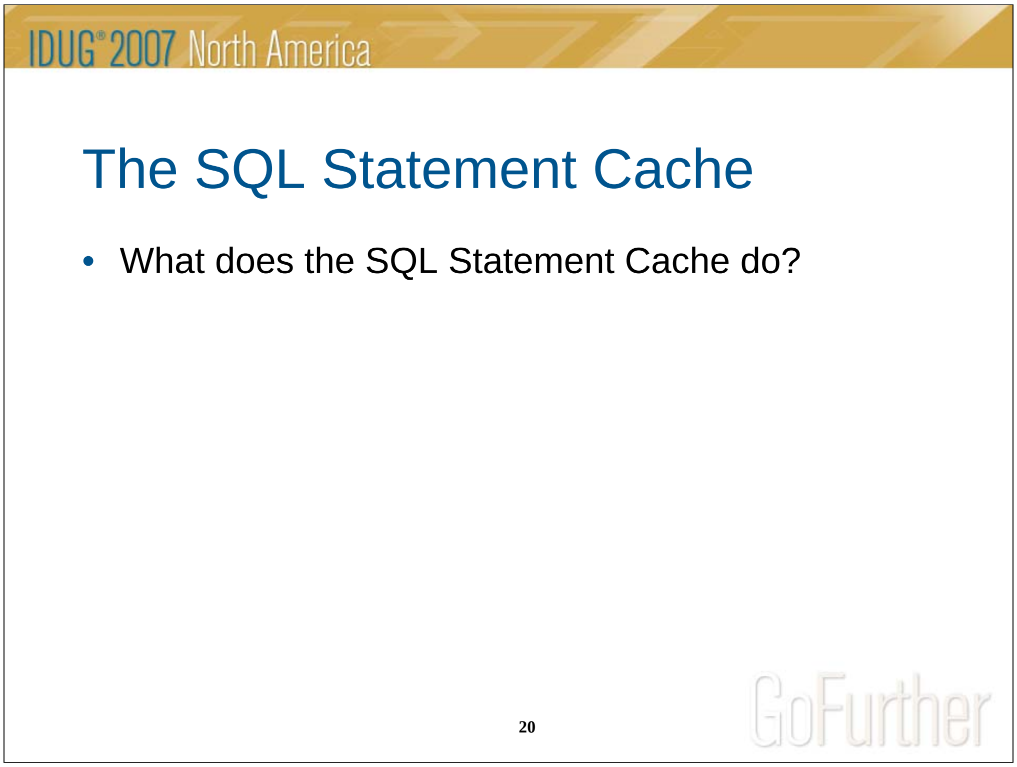
- PC_HASHSIZE set to the prime number closest to the number of User created UDR’s
- PC_POOLSIZE set to a low value (between 4 and 10) times the PC_HASHSIZE.
- Additional tuning may be required.

*I recommend a **prime** number value for **PC_HASHSIZE*** because the insertion of UDR entries are done via a hashing algorithm and a prime number is the most efficient type of number, from a performance standpoint, for a hashing algorithm. The reason to suggest a prime relative to the number of user created UDRS is because of the desire to limit the number of entries per bucket, making retrieval of a cache entry extremely fast, and eliminating overflow problems. **PC_POOLSIZE** be set to a relatively small multiple of **PC_HASHSIZE**, typically in the neighborhood of 4 to 10 times the value of **PC_HASHSIZE**. ***PC_POOLSIZE** should not be limited to a prime number*, because testing does not appear to show any benefit for setting it to a prime number. The primary reason not to use a multiple lower than 4, is because any time you have a hashing algorithm, you can occasionally have different entries hash to the same location, and you need to allow for some overflow per bucket to account for that possibility. So if we have an instance with 200 User created UDR’s, then a good baseline would be the following values:

PC_HASHSIZE 211

PC_POOLSIZE 844

Please note that due to the large number of internal UDR’s it may be necessary to tune significantly higher than the recommendations above, especially if there are a very small number of user created UDR’s.



The SQL Statement Cache

- What does the SQL Statement Cache do?

The SQL Statement cache is used to store the parsed and optimized SQL that is run frequently. This allows for certain memory structures to be shared by sessions. So in addition to having access to pre-optimized queries, the session also saves memory usage. When a sessions executes any SQL, it does the following

- IDS checks to see if the entry is fully realized in the Statement cache.
  - If it is fully realized in the cache, it uses the parsed and optimized query information, and increments the hit count.
  - If it is in the cache but not fully realized, then the hit count is incremented, if after the increment the hit count is sufficient to qualify for a fully realized cache, then the parsed and Optimized SQL form this session is inserted into the cache.
  - If it is not in the cache at all , then it adds a key entry and sets the hit count to 1.

Configuring the SQL Statement Cache

- STMT_CACHE_HITS
- STMT_CACHE
- STMT_CACHE_NUMPOOL
- STMT_CACHE_SIZE
- STMT_CACHE_NOLIMIT

There are 5 ONCONFIG variables used for configuring the SQL Statement Cache. These Variables are:

STMT_CACHE
STMT_CACHE_HITS
STMT_CACHE_NUMPOOL
STMT_CACHE_SIZE
STMT_CACHE_NOLIMIT

STMT_CACHE specifies whether the SQL Statement cache is enabled or not. Valid values are (0) off (1) Application enabled and (2) on.

STMT_CACHE_NOLIMIT specifies whether the variable **STMT_CACHE_SIZE** is a hard limit (0) or a soft limit (1).

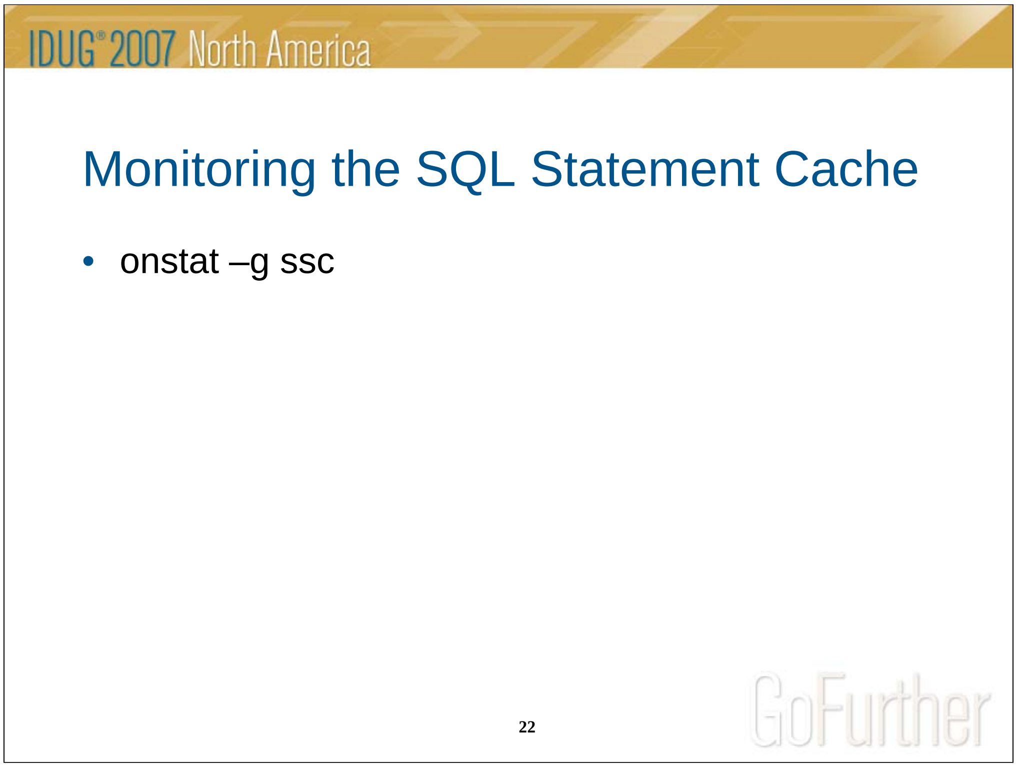
STMT_CACHE_HITS specifies the total number of times an SQL statement must be executed before it can be fully cached.

STMT_CACHE_NUMPOOL specifies the total number of SQL Statement Cache pools held by the engine.

STMT_CACHE_SIZE specifies the beginning size of the SQL Statement Cache if **STMT_CACHE_NOLIMIT** is set to 1, otherwise it specifies the total size of the SQL Statement Cache. Values represent Kilobytes.

The defaults for these ONCONFIG variables are as follows:

STMT_CACHE 0
STMT_CACHE_NOLIMIT 1
STMT_CACHE_HITS 0
STMT_CACHE_NUMPOOL 1
STMT_CACHE_SIZE 512



Monitoring the SQL Statement Cache

- onstat –g ssc

The primary method to monitor the SQL Statement cache is by use of the onstat –g ssc command.

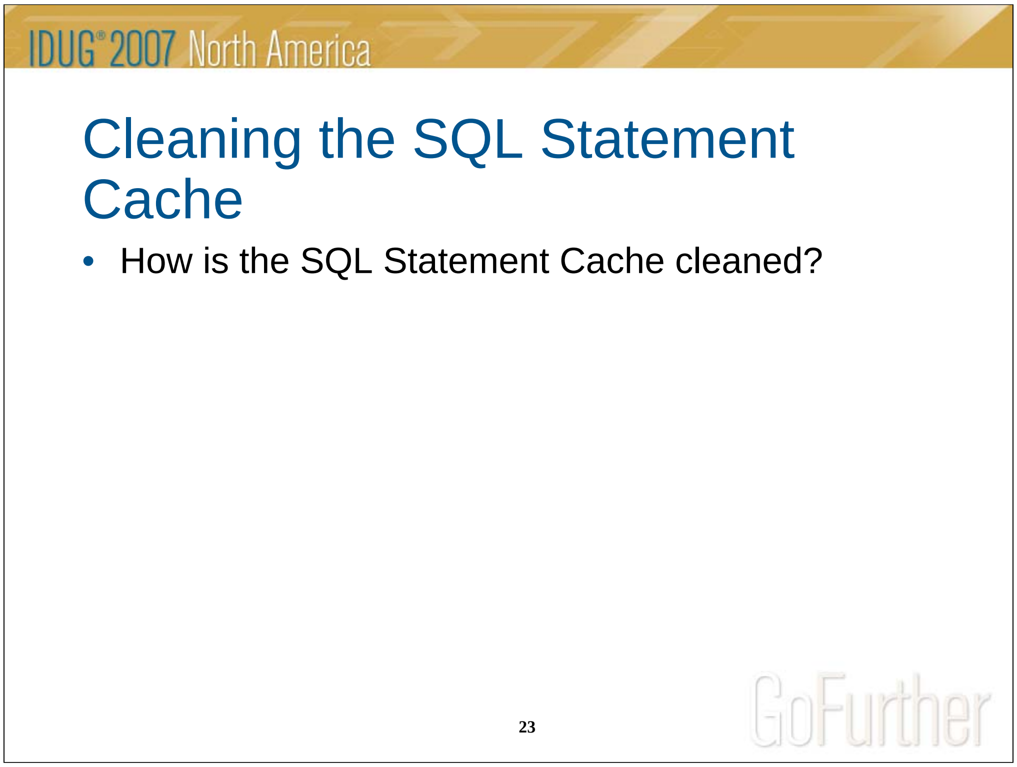
Example:

```
(informix) /usr/informix > onstat -g ssc

IBM Informix Dynamic Server Version 11.10.FC1     -- On-Line -- Up 00:14:32 -- 257120 Kbytes
Statement Cache Summary:
#lrus   currsize  maxsize   Poolsize  #hits   nolimit
6       11552     524288    24576     0       1
Statement Cache Entries:
lru hash ref_cnt hits flag heap_ptr      database          user
-----------------------------------------------------------------------------------
  0  213       0    0   -F c00000001edad038      sysmaster         informix
  select * from syssqlcacheprof
    Total number of entries: 1.
```

The **onstat –g ssc** output has the following fields:

Field	Description
lru	lru used.
hash	hash number for entry
Ref_cnt	Number of user sessions currently attached to the entry.
flag	Flag entry (F) is fully cached
Heap_ptr	Heap Pointer
user	user name who first executed SQL.


IDUG® 2007 North America
GoFurther

Cleaning the SQL Statement Cache

- How is the SQL Statement Cache cleaned?

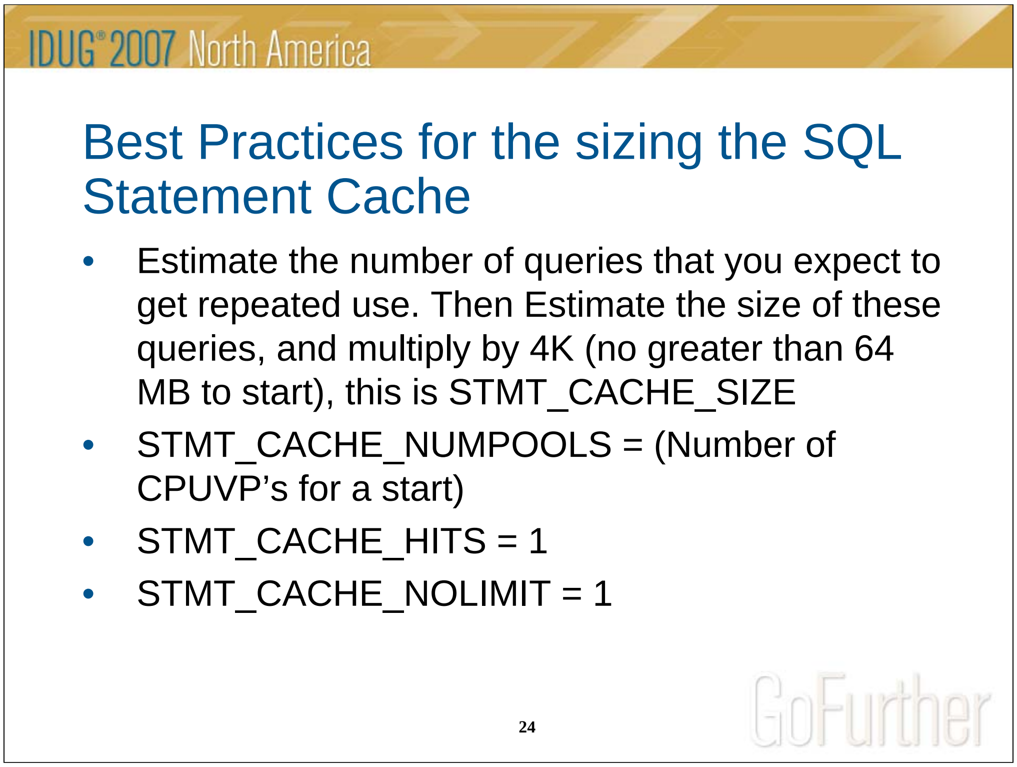
The SQL statement cache is cleaned under the following circumstances

•STMT_CACHE_NOLIMIT is disabled, and the cache is full.

•A process like Update Statistics is run.

•A DBA issues the command

```
onmode –e flush
```

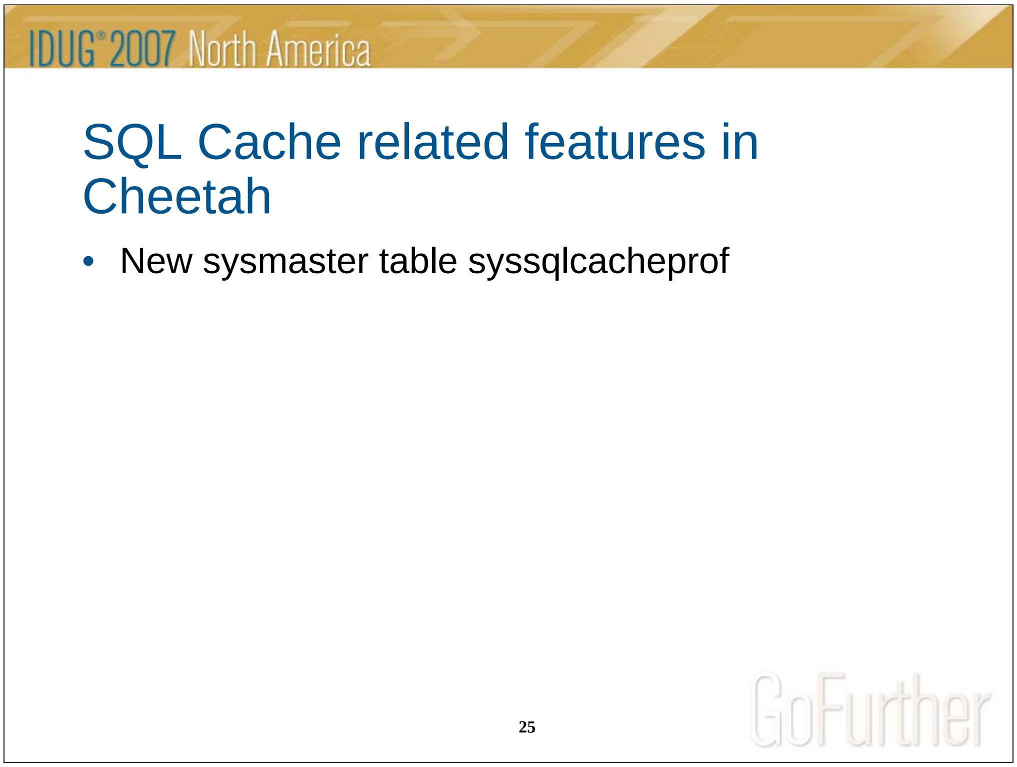


Best Practices for the sizing the SQL Statement Cache

- Estimate the number of queries that you expect to get repeated use. Then Estimate the size of these queries, and multiply by 4K (no greater than 64 MB to start), this is STMT_CACHE_SIZE
- STMT_CACHE_NUMPOOLS = (Number of CPUVP's for a start)
- STMT_CACHE_HITS = 1
- STMT_CACHE_NOLIMIT = 1

There is no easy Best practice recommendations for SQL Statement caches. The above is primarily intended to give you a safe, but small, starting point by which to continue monitoring and tuning.

Note another valid method is to set the above and set the STMT_CACHE to 2. Then enable this on an application by application basis.



IDUG® 2007 North America

SQL Cache related features in Cheetah

- New sysmaster table syssqlcacheprof

GoFurther

The biggest cache related feature is the addition of the syssqlcacheprof table in the sysmaster database. This table gives you important information like cache memory usage, the number of times the cache has been successfully hit, the number of times the cache has not contained the necessary info, and the number of times the cache has had entries removed. This table will allow for more intelligent tuning after starting with the best practice guidelines for each cache.

Q&A

- Discuss questions from the presentation

Session: M04
Performance Tuning Shared
Memory SQL Related Caches

Mark Jamison
IBM Information Management
mjamison@us.ibm.com