Sybase Adaptive Server® IQ Multiplex

Performance and Tuning Recommendations Adaptive Server IQ-M Version 12.4.2

by Lou Stanton
Technology Specialist
Sybase Business Intelligence Division
**Author’s Note**

This document is intended to provide supplementary information and guidance for improving the performance of an Adaptive Server® IQ Multiplex (IQ-M) database. Some of the information in this document is contained in the product documentation, particularly in the *Adaptive Server IQ-M Administration and Performance Guide*. I have supplemented that information with additional guidance and recommendations gained from my experience as well as the experience of other Sybase employees.

This paper is an update that addresses features and functionality of Adaptive Server IQ Multiplex version 12.4.2. The previous version of this document addressed version 12.4. As the IQ-M server continues to be developed and improved, some changes and recommendations may be published in future Release Bulletins, and EBF Cover Letters may supersede some recommendations contained in this document. Every attempt will be made to keep this document current to the latest production release of IQ-M.

**Acknowledgments**

This document grew out of many discussions with and input from Sybase employees, especially David Walrath, Steve Kirk and Richard Soundy.

**Distributing this Document**

This document may be freely distributed.
Introduction

This paper is intended to provide server tuning guidance for the database administrator who is about to create or who has built a database with Adaptive Server IQ-M. The document is arranged in several parts, each addressing different performance areas.

Experience working with IQ-M customers suggests that many performance problems can be directly related to the following three areas:

• Insufficient or improper indexes or columns
• Misallocation or overallocation of memory for IQ-M caches
• Incorrect setting of database options

Each part of this document contains discussion and specific recommendations for improving performance in these areas.

Tuning a database by creating indexes, configuring memory and setting database options may not solve all your performance problems. Many performance problems can be traced to the very design of a database. If, after implementing changes suggested in this document, you continue to experience performance problems, then you should review the last part of this document relating to database design. This part covers several subjects including database configuration options and design. If you are about to create an IQ-M database, you might consider reading this part first to avoid locking in poor performance by using the wrong configurations and table design.

Changes from Previous Version of this Document

Version 12.4.2 introduces a number of important features affecting IQ-M server performance that are addressed to varying degree, including:

• New optimized FP index
• Parallel index create
• New database options
• Server request logging

There are also a number of “beneath the covers” changes that improve IQ-M server performance. You should consult Part 6 of the Release Bulletin for more details.

Topics have been reordered and new material has been added to reflect new features.
# Table of Contents

Author’s Note ........................................................................................................2
Introduction ...........................................................................................................3
Changes from Previous Version of this Document .............................................3

Part 1 – Design Recommendations for an IQ-M Database................................6
  Database Configuration Options.................................................................6
  Designing Tables .........................................................................................7
  User tables in the Catalog Store ..............................................................8

Part 2 – Indexing Techniques ............................................................................9
  Default Index (or Fast Projection Index) – FP ..............................................9
  Flat FP Index .............................................................................................11
  Low Fast Index – LF .................................................................................12
  High Group Index – HG ............................................................................13
  High Non Group Index – HNG .................................................................13
  Index Selection Summary .........................................................................14
  Indexing Tips .............................................................................................14

Part 3 – Allocating Memory for the IQ-M Caches .............................................16
  Memory Components in the IQ-M Server ................................................16
  Conduct a memory inventory ...................................................................16
  IQ-M Caches .............................................................................................17
  Setting the Caches ....................................................................................18
  Memory Tips .............................................................................................19

Part 4 – Database Options Effecting Performance ..........................................20

Part 5 - Load Performance ..............................................................................22
  Parallel (Multi-Threaded) Loading ...........................................................22
  Multi-File Loading for a Table .................................................................22
  Improving Incremental Data Loads ...........................................................23
Part 1 – Design Recommendations for an IQ-M Database

Design is crucial for performance. Understanding the database server’s strengths and weaknesses is critical to the design process. Mistakes made in the design or configuration of a database can almost guarantee poor performance that cannot be fixed with indexes, memory or database options.

Database Configuration Options

Before you create your IQ-M database, you will need to decide upon a number of critical database configuration options. You cannot change these options after you build your database. In version 12.4.2, many critical database parameter defaults were changed to provide the best performance configuration for a database.

Case    Default: Respect

You’ll get better query performance with a Case Respect database, especially when used with the ISO_BINENG collation. Just be sure this is the best configuration for your database. For example, you might consider a case insensitive database (case ignore) if the application searched classified newspaper ads where data capitalization is random or unpredictable.

IQ-M Page Size    Default: 65,534

The IQ-M page size sets the size of a buffer in the IQ-M Main and Temp Caches and the page size of the IQ-M stores on disk. For most applications, the default is the correct size. However, there are performance benefits with a larger IQ-M page size. The decision to change the IQ-M page size depends upon three factors: the size of the largest table (in rows), the amount of RAM available for the IQ-M caches and the expected number of concurrent users actively running queries. You might consider a larger IQ-M page size (131,072 or 262,144) when the largest table has more than 1 billion rows and if you have 4 gigabytes or more of memory with 20 active users or less. For more information, see chapters 3 and 12 of the Administration and Performance Guide.

Page Size    Default: 4096

Page size refers to the size of the page in the IQ-M Catalog Store. This is now the default size and should not be changed. Using a smaller page size may limit the number of columns you can define in a table.

Blank Padding    Default: On

The blank padding option controls how character fields will be treated for comparison tests in query. The default configuration option should be left ON when creating an IQ-M database. (Earlier server versions had this parameter set to "Off" by default.)
ASE Compatible

Despite information to the contrary in the original Adaptive Server IQ-M documentation set, this database option is not functional and should not be used. For the most part, it has been removed from the current documentation.

Collation

Default: ISO_BINENG

Collation (or collation sequence) is a combination of a character set and a sort order for characters in a database. Choosing the proper collation ensures that the sorting and comparison operations produce the proper results for the native language of the user. Chapter 9 of the Administration and Performance Guide provides details on collation. English language databases will perform best with ISO_BINENG collation in combination with the Case Respect option. These are both defaults for creating IQ-M databases.

Block Size

Default: Based on IQ-M Page Size

The block size parameter is intended to set the IO transfer block size in bytes for the IQ-M Main and Temporary stores of the database. However, the block size in an IQ-M database may vary and is based upon the amount of compression achieved on a particular page in memory before it is written to disk. If not specified in Create Database command, the IQ-M page size determines the block size. (See block size in Chapter 12 of the Administration and Performance Guide.) Unless there are overriding reasons to specify a block size, allow the IQ-M page size parameter to set the block size for your database. Changing the default block size will affect data compression on disk.

Transaction Log

Default: On

This option permits a database to be created with or without a transaction log. This refers to the transaction log for the Catalog Store, not the IQ-M Main Store. Never create a database without a transaction log.

Designing Tables

These recommendations address performance considerations when designing tables in a database.

Define Primary Keys. The IQ-M query optimizer can make better decisions when primary keys are defined on tables. For single column primary keys, use a Primary Key constraint in the Create Table command. This constraint will create a HG index automatically on the column. This type of HG index is very space efficient since it will NOT have a Group Array (G-Array) and, as a result, will use less disk space. Note: The HG index created by the Primary Key constraint can only be dropped using the Alter Table Drop Primary Key command.

There is currently no support for multicolumn or concatenated primary keys. In IQ-M version 12.4.2 (and earlier versions), there is no benefit in creating a multicolumn Primary Key. Multicolumn keys are currently NOT enforced as unique, and no real IQ-M indexes are

---

1 Multicolumn Primary Keys will be supported in version 12.4.3.
created. For multicoloumn keys, create appropriate indexes on the individual columns including the key.

**Foreign Keys.** NOT RECOMMENDED. The IQ-M *does not* currently support referential integrity (RI), however, you may define foreign keys using the UNENFORCED key word. There is little, if any, performance benefit to be gained, and there is a potential for the query engine to return incorrect results. When key relationships are defined between tables, the IQ-M query parser may rewrite a query involving the tables defined with PK-FK relationships and change a join to an ANSI style join. The parser assumes there is RI between the tables and may use join elimination to resolve the query. You may end up with incorrect results with these queries if orphan records exist in the dependent table (i.e., Primary Key does not exist in the parent table).

**Unsigned Integer Data Type for Keys.** If possible, design table keys, primary and foreign, using the unsigned integer data type. This data type uses only 4 bytes and performs the best in table joins.

**Date Columns.** You have several options for storing datetime data. IQ-M has DATE, TIME and TIMESTAMP data types available. (Note: DATETIME and SMALLDATETIME data types are also available but are exactly the same as TIMESTAMP. They were included as user-defined data types for compatibility with Sybase Adaptive Server Enterprise.) If your application is not using the time aspect of a datetime column, store it as a DATE. The DATE data type uses only 4 bytes versus 8 bytes for DATETIME.

**Char vs. Varchar Data Type.** For columns less than 256 bytes, use the char data type over Varchar. There is a slight performance improvement. For columns larger than 255 bytes, however, you must use the Varchar data type.

**IQ UNIQUE Constraint.** In the Create Table command, define columns using the IQ UNIQUE constraint wherever possible. There is a detailed discussion of this constraint in Part 2 of this document in the section about FP indexes.

**Create New Columns to Avoid String Searches.** If user queries typically search the same byte string in a column using an expression such as SUBSTR(part_no, 1, 4), consider creating a new column in that table consisting of the bytes searched by the function. Index the new column with the appropriate index (LF or HG). Using a LF or HG index will return rows much faster than the SUBSTR function. The SUBSTR function typically will require a scan of the entire FP index.

**User Data Tables in the Catalog Store**

Creating user data tables in the Catalog Store is not recommended, especially if there will be joins between the IQ-M Main Store and tables in the Catalog Store. The bridge (or connection) between the Catalog Store and the IQ-M Main store is very “narrow” and will not feed data quickly to the IQ-M query engine. Queries involving joins between tables in the IQ-M Main Store and Catalog Store typically run poorly. Also, with the Update verb available in version 12.4.2, there is no longer a need to create tables in the Catalog Store.
Part 2 – Indexing Techniques

Proper index selection is essential for query performance in an IQ-M database. Unlike traditional relational database management systems (RDBMS), it is conceivable that every column in an IQ-M database will be indexed, and some columns may have as many as three indexes. For a database administrator with OLTP database experience, the idea of indexing every column is unusual and perhaps unthinkable. In an OLTP database, indexes are additional overhead using more space in the database and impact performance when data is inserted or deleted. Regardless, indexes are critical for query performance and maintaining entity integrity. To maintain performance in an OLTP database, a database administrator must weigh the costs of additional space requirements and transaction performance versus data integrity and response time. It’s the classic performance and tuning dilemma where you improve one process and slow down another.

IQ-M indexing is completely different from indexing in a traditional RDBMS. The indexes are the data and must be created aggressively to ensure query performance. IQ-M indexes use much less space compared to OLTP B-tree indexes and do not require special maintenance after data changes. Most importantly, the IQ-M query optimizer depends heavily upon indexes to provide statistics about the data to determine the optimal query execution plan.

A review of the IQ-M indexes and how they affect loading, storage and query performance follows. Note that there have been significant changes to the FP index in version 12.4.2.

Default Index (or Fast Projection Index) – FP

The FP index is created on all columns automatically, hence it is also known as the default index. This index stores the raw data for a column and is used in many aspects of query execution. You cannot delete this index from a column without dropping the column from the table using the Alter Table command.

As IQ-M has evolved, the FP index has taken on greater significance for performance and is more than just the raw data for a column. As of version 12.4.2, there are now two “optimized” storage options for the FP index, which can be used with any column with cardinality less than 65,536 values. These optimized FP indexes have important storage and query performance implications. The regular or flat FP index (raw data) as well as the new optimized FP indexes are still used in a number of critical query operations, including projection of data, ad-hoc table joins (as opposed to joins using join indexes) and string searches with the Substring function or Like clause. The optimized FP indexes, however, provide the optimizer with column cardinality information, something that only Low Fast (LF) and High Group (HG) indexes could previously provide. The result is the new optimized FP indexes can be used on columns that may not otherwise have had an LF or HG index, allowing the optimizer to make better decisions and, at the same time, saving storage space.

In IQ-M version 12.4.2, there are three new types of FP indexes: regular, optimized and wide. The flat FP refers to an FP index that is just the raw data built for columns under 256 bytes. The optimized FP indexes store data in a lookup table with the actual rows containing a 1- or
2-byte pointer to the lookup table. Optimized FP indexes can be used with a column width less than 256 bytes and cardinality less than 65,536 values. The wide FP index type is used on Varchar columns greater than 255 bytes. Except for the wide FP, the FP index storage (regular or optimized) type is determined by the IQ UNIQUE constraint parameter (or lack thereof) in a Create Table command.

The next section describes how the IQ UNIQUE constraint functions. The sections after describe each FP index type.

**IQ UNIQUE Constraint**

The IQ UNIQUE Constraint is an optional statement used in the Create Table command to provide the IQ-M server an estimate of the number of distinct values in a column. For example:

```sql
Create Table customer (
    customer_key unsigned int Primary Key,
    ...
    customer_last_name varchar (30) IQ UNIQUE (200000),
    city varchar (30) IQ UNIQUE (50000),
    state char (2) IQ UNIQUE (50),
    ...
)
```

In the example above, the database administrator created the customer table and estimated there would be 200,000 distinct last names for customers; 50,000 distinct cities; and 50 distinct states. The number passed as the IQ UNIQUE constraint parameter has important implications:

- When the value in the IQ UNIQUE constraint is less than 256, the server will attempt to create a 1-byte optimized FP store for that column.
- If the value is between 256 and 65,535, the server will attempt to create a 2-byte optimized FP store.
- A flat FP index is created whenever the constraint is not specified or the value passed is greater than 65,535.
- In all cases, when IQ UNIQUE is specified and there is no enumerated index for the column, the IQ-M optimizer will use the IQ UNIQUE value for column cardinality. That cardinality value will be used in query plans to determine selectivity when the column is used in a Where clause predicate.

Example: In the customer table above, if a query used the customer_last_name column in a Where clause predicate and there was no LF or HG index, the optimizer would deduce column selectivity based upon 200,000 distinct values.

**Important Notes about IQ UNIQUE**

The phrase “will attempt” means just that. The server uses the value in the constraint to determine the type of FP index to create for a column. With no hint (no IQ UNIQUE constraint), the server creates a flat FP. If during data modification, a threshold value (255 or 65,535) is exceeded for an optimized FP, the storage type will roll over to the next higher type (1 byte to 2 byte, 2 byte to a flat FP). Columns will not roll back to a smaller store type even if the cardinality is reduced.

---

2 Enumerated indexes are LF and HG indexes. With either of these indexes, the exact cardinality of the column is stored internally and used by the optimizer during queries.
In query plans, the optimizer will use the value specified in the IQ UNIQUE constraint for column selectivity if there is no enumerated index for the column. This occurs when the column is used as a search argument (Where clause predicate) in a query. Consider this implication before using IQ UNIQUE on any column that does not have an LF or HG index. The database administrator should take into account the accuracy of the parameter and long-term cardinality stability of the column before using the constraint in Create Table. Significant cardinality changes over time could lead to poor query response as the optimizer may be using incorrect information. If you would like to have selectivity information available to the optimizer, there is an alternative method of passing statistics to the optimizer without using IQ UNIQUE. Rather than hard-coding cardinality using IQ UNIQUE, use syntax in the Where clause to pass estimates of the number of qualifying rows for a join or search argument in queries.3

The IQ UNIQUE constraint can only be used in Create Table syntax and cannot be added, changed or dropped using the Alter Table command. If you have an existing table and want to take advantage of these new FP stores, you will need to drop, recreate and reload the table4.

**Flat FP Index**

This is the storage method for columns with width less than 256 bytes where no IQ UNIQUE constraint is specified, or if the IQ UNIQUE constraint parameter is greater than 65,535. This index is just the raw data stored in a column. Regardless of the data type selected (char vs. Varchar), trailing blanks in the data are compressed. The amount of storage space used on disk will depend on the data itself.

**Optimized FP Indexes**

In IQ-M version 12.4, an “optimized” 1-byte FP store was introduced for columns with cardinality less than 256. With version 12.4.2, a 2-byte optimized store was added for columns with cardinality up to 65,535 values. The IQ UNIQUE constraint determines which type of store will be created. These optimized FP stores can improve query response by providing additional information to the optimizer, reduce disk IO and substantially reduce the amount of storage space on disk.

After a table is modified, a column defined with IQ UNIQUE constraint will be populated with unique values stored in a lookup table with a 1- or 2-byte pointer, depending on cardinality. The data value for each row will be a 1- or 2-byte pointer referencing a value in a lookup table.

As with any performance-enhancing feature, there is a cost. Creating or maintaining an optimized FP index results in a performance penalty paid during data modification. The server uses the IQ-M Temporary Store memory (and possibly the IQ-M Temporary Store disk devices) to manage the optimized FP stores. If the cardinality exceeds 65,535, there will be a one-time additional cost of reformatting data pages to roll over an optimized store to a flat FP. As mentioned before, there is no rolling back to a more efficient FP store when cardinality is reduced.

The space savings for the optimized FP stores tend to outweigh any loading costs. Consider this small example: In a customer table with 100,000 rows, there are 9,170 unique last names in a char (20) column. Stored as a flat FP index, the data in the column uses 872 kilobytes. The same data stored as a 2-byte FP uses 492 kilobytes of space. That represents almost a 50% savings, and the savings increases as the number of rows increases.

---

4 This restriction will be removed in version 12.4.3
**Wide FP**

Columns defined as `VARCHAR > 255` will be stored as a wide FP. For wide FP indexes, disk storage is allocated in 255-byte increments. If a column were defined as `VARCHAR (2000)` and the value for a row were 50 bytes, 256 bytes would be allocated on disk for that row. If the data length of a value were 300 bytes, 512 bytes would be allocated on disk, etc. Reminder: The only query functions permitted on wide FP-type columns are projection (with or without a Substring function) and the search arguments Like and Substring.

**Low Fast Index – LF**

The LF index is one of the “fast” indexes for queries. It’s designed for columns with low cardinality (under 1,500 values). This index cannot be used with columns defined with float, real or double data types. LF (along with the HG index) is also one of the IQ-M enumerated indexes. After any DML operation on a table is committed, the exact number of values in the column is known and stored for use by the optimizer. LF indexes cannot be used if column cardinality exceeds 9,999 values, and any insert or update operation causing cardinality to exceed that value will fail with a rollback. The LF index can be declared as unique.

The LF index consists of a B-tree and bitmap for each distinct value in the column. If there are 1,200 distinct values, there will be 1,200 bitmaps. Used in conjunction with the internal table row ID, IQ-M stores a “1” (or bit “on”) in the bitmap to represent the data value for a particular row.

**Data Loading** – Each distinct value of LF index uses one buffer in memory in the IQ-M Main Cache. It follows that as the number of distinct values in a column increases, more memory is used for the IQ-M Main Cache.

**Data Storage** – With compression of the bitmaps, the LF index is stored efficiently using typically 20% the size of the raw data. Storage size is directly related to the number of distinct values in the column and will increase with column cardinality. For this reason, the recommended limit is approximately 1,500 unique values.

**Query Performance** – The query engine uses the LF index extensively. After data loading, column cardinality is stored for use by the optimizer. Cardinality knowledge plays a critical part in determining the best join and/or group by algorithm in query plans. Cardinality statistics are also used with `COUNT ()` and `COUNT (DISTINCT)` functions, and the internal storage of the LF index provides fast performance with the `MIN()` and `MAX()` functions. The LF index is critical for query performance involving these Where clause predicates: Equality, Inequality, In lists and sometimes range searches (i.e., `>`, `<`, `>=`, `<=`, BETWEEN and NOT BETWEEN).

**Small Table Exception** – The above notwithstanding, there are storage and query performance reasons for not using an LF index on low cardinality columns for tables with less than 10,000 rows. Because of the differences in the way these indexes are stored, it’s advantageous to use HG indexes over LF indexes on smaller tables. HG indexes will use fewer pages on disk, and the result is less disk IO during queries.
High Group Index – HG

HG is the other enumerated index and also is considered a “fast” index for queries. HG indexes are intended for use on columns with cardinality greater than 1,500 values. It can be declared as unique. The HG index consists of a B-tree and a G-Array (Group Array). If an HG index is declared as unique (or as a Primary Key), it will be built without the G-Array.

Data Loading – HG indexes have the greatest impact on load performance, and careful consideration should be given to selecting columns for this index type. HG indexes depend heavily upon the IQ-M Temporary Cache during inserts by accumulating and sorting data before loading into the IQ-M Main Store. Only after all rows have been read and sorted will data actually begin to be sent and loaded into the IQ-M Main Store. If there is insufficient memory in the IQ-M Temporary Cache to sort all HG indexes, you will incur IO as buffers are paged out to the IQ-M Temporary Store. For this reason, you want a large IQ-M Temporary Cache for loading HG indexes. (See the discussion about memory configuration in Part 3 regarding loading performance and HG indexes.)

Data Storage – The HG index tends to use the most disk space of all the indexes and could use between 80% to 120% of the size of the raw data for the column. If the HG index is declared unique it is built without the G-Array and will use less disk space.

Performance Implications – Like the LF index, the optimizer uses the HG index extensively. The number of distinct values is stored after loading for use in determining join and group by algorithms as well as for count(), count (distinct), min() and max() functions. Except for range searches, HG indexes are critical for performance in Where clause predicates.

High Non Group Index – HNG

The HNG index is referred to as a “bit-wise” index. This index is used for both low and high cardinality columns for performance with aggregates (sum and avg) and range searches. For numeric data, the values are stored as a 32-bit word (64 bit for 64-bit versions of IQ-M), with each significant bit stored vertically and independently. For character data, each byte in a string is stored with each significant bit stored vertically and independently.

Data Loading – This index loads very fast. Each bit of an HNG index requires one buffer in the IQ-M Main Cache.

Data Storage – With disk compression of the bits, HNG stores efficiently on disk using between 2% to 20% of the size of the raw data.

Query Performance – The HNG index should always be used for high cardinality columns involving range searches in the Where clause and with the aggregate function sum and avg. This index is also useful for low cardinality columns for the same reasons. You should consider creating an HNG index for low cardinality columns with more than 300 distinct values. Depending upon the values of a particular range search, the optimizer may choose to use the HNG index over the LF index for better performance.
Index Selection Summary

In summary, create an IQ-M index for each column as follows:

Low Cardinality Columns ( <1500 values) – LF Index
It’s appropriate for all query functions [equality, inequality, grouping, min(), etc.] and uses very little space on disk. If the column is a single column Primary Key, allow the server to create the HG index instead and do not create the LF index.

High Cardinality Columns – HG Index
The HG index should be used if the column is used in joins; Group by expressions; Where clause predicates for equality; inequality; IN lists; and in the select list for the functions count(), count(distinct), min() and max().

All Columns – HNG Index
HNG index is used in range searches and aggregate functions sum() and avg().

Do all columns really need an additional index (other than the FP)? No. You don’t need an index on any column that is only projected (used in Select statement) or that is only used in Substring or Like searches. Columns with certain data types cannot use certain indexes including Varchar (> 255) and bit. If you are certain a column will only be used for these operations, then no additional index is needed.

Indexing Tips

1. Use the IQ UNIQUE constraint in Create Table. You will improve data compression and query response on low cardinality columns (less than 65,536). Expect some impact on load performance, however, as cardinality increases. If the target column has an HG or LF index, then the cardinality parameter used for the IQ UNIQUE constraint of the data must be exact. Cardinality values near critical thresholds (256 and 65,536) are the most important. Recognize that values passed in IQ UNIQUE constraints on columns without an enumerated index will be used by the optimizer for selectivity estimates when that column is used in a Where clause. Use this constraint with care in these situations, especially if cardinality will change over time.

2. Create IQ-M indexes before loading data. In most cases, indexing will have only a minor effect on overall load performance, and you’ll save time in the long run. With version 12.4.2, you can create indexes in parallel after data has been loaded. Limit the number of index creates to the number of processors on your server for better performance.

3. Choose IQ-M Indexes by committee. That is, when you are deciding which IQ-M indexes to create, assemble the people with the knowledge of cardinality and how the data will be used in the warehouse. Rarely is this knowledge vested solely in the database administrator.

4. When in doubt, index everything. If you can’t get consensus on how a column may be used in queries, use the index selection criteria above. (Remember, however, that HG indexes are expensive to load and store.) You can always drop unnecessary indexes later. You can monitor user query activity by using the server switches (new in version 12.4.2) to pipe SQL statements to a file.6
5. **Specify LF or HG index as unique, if appropriate.** This information is useful to the optimizer at query time and, in the case of HG indexes, you'll get more efficient storage on disk.

6. **Use HNG with LF and HG indexes for range searches.** HNG indexes are cheap to store and fast to load. For range searches, the HNG index can be faster than an LF index (in certain cases) and is a must for high cardinality columns.
**Part 3 – Allocating Memory for the IQ-M Caches**

After index selection, memory allocation to IQ-M caches is perhaps the primary reason behind complaints of load and query performance. To achieve the best performance, you will need to balance the overall needs of the server (OS), the IQ-M server and the two IQ-M caches. Starve anyone of these components for memory and your performance will be adversely affected. Additionally, you will need to change some database options affecting memory. All these topics are addressed in detail in the following discussion.

Also, Sybase strongly recommends an IQ-M database run on a dedicated server. Sharing system resources between an IQ-M server and other applications will impact the performance.

**Memory Components in the IQ-M Server**

There are four memory components associated with your IQ-M data server: the IQ-M server process itself, two IQ-M caches [Main and Temporary (or Private)] and the heap memory overhead used for loading data.

**Conducting a Memory Inventory**

To determine adequate memory allocation, first assess memory usage of all processes running on the host server, then add a small fudge factor, and subtract that total from the total available memory. The remaining memory, within certain limits, is available for your IQ-M caches.

**Operating system (OS) and Other Applications.**

With the IQ-M server running, determine the amount of memory consumed by the OS and any other non-IQ-M applications and add 10%. In Unix, you can use vmstat (or other system utility) or the NT performance monitor. It is strongly recommended that no other processes run on a server with the IQ-M server.

**The IQ-M Server process.**

The amount of memory used by the server varies based upon the OS it is running on and the number of users connected and running queries. With the number of user connections being constant, IQ-M will use the least amount of memory on NT and the most on a 64-bit OS. Since this document is OS-agnostic, you will need to determine for yourself how much memory the server needs using your OS tool of choice. As users connect and perform queries (not loading data), the server memory may grow about 5 megabytes per user. If possible, determine the amount of memory the IQ-M server will need based upon the average number of users who will be actively running queries. Assess this amount and again add 10%.

**Additional Memory for File Systems.**

If you happen to be using UFS (Unix File Systems) or NTFS for your IQ-M database stores, set aside about 10% of total real memory for file system cache. Data must be buffered from disk while being read (or written) to disk. Don’t starve the file system cache as it plays an important role in performance, particularly on NT. (On Sun and NT, you can disable the file system cache using a database option. See Part 4, Database Options Affecting Performance.)

**IQ-M Database Load Overhead.**
When you load data into a table from an OS file, you will need memory to buffer data between the disk file and the IQ-M caches. By default, the amount of memory needed in megabytes is about equal to the record length of an input record divided by two. For example, a 500-byte input record will need about 250 megabytes for load overhead. This represents a significant amount of memory for servers with limited memory. You can control the amount of memory needed using the database option "Load_Memory_MB." The default value is 0 (meaning use as much as needed up to 500 megabytes), and the maximum value is 500 (megabytes). If memory on your server is at a premium, set this option to a reasonable value below 500. The impact of reducing the setting is more IO reading raw data from disk. If you set Load_Memory_MB too low, a table load could fail. The command to set this option to 100 megabytes for example, is:

```
Set Option public.load_memory_mb = 100
```

If you load data from another server, the load overhead is much smaller. A good estimate is input record length divided by four instead of two.

**IQ-M Caches**

The IQ-M server has two caches, Main and Temporary, which need to be sized correctly for optimal performance. The default cache sizes of 8 megabytes and 4 megabytes, Main and Temporary, respectively, are much too small for a production database. Before setting your caches, some discussion about how IQ-M uses the caches may help in understanding how to set them.

There are two scenarios to address for the caches: loading data and performing queries. If you intend to perform table loads while users conduct queries, then you’ll need to consider both scenarios when you size caches.

**IQ-M Main Cache – Data Loading**

As data is read from disk the Flat and Wide FP indexes, and all LF and HNG indexes are built in the IQ-M Main Cache. Buffers (pages) in the Main Cache process the data (see discussion about how these indexes use memory during loads in Part 2 - Sybase IQ-M Indexes) and fill buffers with indexes. When a buffer in memory is full (that’s an IQ-M page size buffer), it’s compressed and sent out to the IQ-M Main Store. Contrary to what you may think, the IQ-M Main Cache only needs to be big enough to accommodate the number of buffers for processing the FP, LF and HNG indexes. If there is additional space in the IQ-M Main Cache, full pages sit in memory until flushed to disk. At the end of the load, all the buffers containing data must be flushed to disk before the load can complete and commit. As a result, if your Main Cache is very large you’ll need to wait for all buffers to be flushed. The point of all this is a large Main Cache for loading is unnecessary and will slow down loading. Make it too small and the load will fail with an out-of-memory message.

**IQ-M Temporary Cache – Data Loading**

The Temporary Cache processes the HG indexes during loading. For each HG index, the data is sorted and saved in the IQ-M Temporary Cache and, if necessary, the IQ-M Temporary Store. No writing of HG index pages to the IQ-M Main Store occurs until all the raw data has been read, sorted and saved. The space required to process HG indexes can be
computed (very approximately) by multiplying the width of the column plus 8 (for the row ID, which is also needed) times the number of records to load. Do this for each HG index in the table (but only account for row ID once) to determine the amount of Temporary memory (and disk) required.

Sorting and saving will occur first in the Temporary Cache until all the buffers are used, and then buffers will be paged out to the IQ-M Temporary Store. If during loads you run the IQ-M Monitor for the Private7 store, you’ll start to see IO activity when the Temporary Cache is full. Recall that loading HG indexes always takes the longest time, so your strategy should be to speed up the HG index loading by minimizing IO to the Temp Store. For best loading performance, you should always make the IQ-M Temporary Cache larger (often much larger) than the Main Cache during table inserts.

IQ-M Main Cache – Queries
The Main Cache will receive and process pages of data as needed from the IQ-M Main Store. If data can be found in cache, then it will be processed quicker than requiring an IO fetch to disk. If the Main Cache is large in theory, there will be fewer IO to disk. In data warehousing, however, the chances of finding a particular data page in cache are remote when you are dealing with millions of rows of data. Chances are there’s usually more activity in the Temporary Cache.

IQ-M Temporary Cache – Queries
The Temporary Cache is where joins, sorting, grouping, subqueries and temp tables are handled. If your query environment involves these activities, then you will need to consider allowing for enough space in your Temporary Cache to minimize paging buffers to the Temporary Store. If queries tend to use only a single table, you won’t have too much activity in the Temporary Cache. As in loading data, the overall strategy is to minimize IO, especially with the Temporary Cache and Temporary Store.

Setting the Caches
You will need to execute two Set Option commands to set the IQ-M cache sizes, one for the Main Cache and the other for the Temporary Cache. You must restart the IQ-M server for these changes to take effect:

\[
\text{Set Option public.main_cache_memory_mb = x} \\
\text{Set Option public.temp_cache_memory_mb = x}
\]

where \(x\) is the size in megabytes.

After evaluating the amount of memory for your IQ-M caches, start by dividing the available memory 40-60 to the Main and Temporary Caches, respectively. You should consider a 30-70 proportion (Main to Temporary) if your queries involve complex table joins, large group by statements or heavy use of temporary tables, or if you intend to perform table inserts while users are conducting queries. Use the IQ-M Monitor with the “--io” option to determine activity between the Temporary Cache and Temporary Store. Adjust the size of the Temporary Cache to reduce IO activity between the Temporary Cache and Temporary Store. Depending upon the platform, there are limits to the amount of memory you can allocate to

---

7 Private refers to the IQ-M Temporary Store. Earlier versions of Sybase IQ-M had “Private” index spaces instead of shared Temporary Stores.
IQ-M caches. This is especially true with 32-bit versions of IQ-M. Consult the Release Bulletin and Installation and Feature Guide for your platform to determine the memory limit. 64-bit versions of IQ-M effectively have no cache size limit.

**Memory Tips**

1. Don’t overallocate memory for IQ-M caches. Be conservative in assessing the memory used by all the applications running on the server.

2. With version 12.4.2, there is even more emphasis on IQ-M Temporary Cache activity. The IQ-M Temporary Cache is used heavily for loading HG indexes and building optimized FP indexes as well as processing joins, temporary tables and some GROUP BY statements. Make sure the Temporary Cache has sufficient memory, even at the expense of the Main Cache.

3. A large Main Cache may slow down table loading. Use the memory for the Temporary Cache particularly if the loading process is conducted during off-hours.

4. Use the Load_Memory_MB database option as needed to limit memory used when loading data from files.

5. An IQ-M database should have a dedicated server with only one database per IQ-M server.
Part 4 – Database Options Affecting Performance

There are a number of IQ-M database options that may affect the performance of your server. Several of the database options below were addressed earlier in this paper and are repeated here. In version 12.4.2, several new performance options were added, and one infamous option, Optimize_For_This_Many_Users, was removed. You will find detailed information in Chapter 5 of the *Adaptive Server IQ-M Reference Manual*.

**Displaying Options** – You can list all current database options alphabetically with the query:

```
Select * from sys.sysoptions order by 2
```

You can also display options with the dbisql GUI client by executing the command SET. A separate scrollable window appears listing all options.

**Types of Options** – There are three types of options: public, temporary and user. There are also a series of options specifically for dbisql clients. These are discussed in Chapter 5 of the *Adaptive Server IQ-M Reference Manual*.

**Disk Striping** – The option is ON by default. When there are multiple devices for the IQ-M Main Store, the server will write data to all devices in a round-robin fashion. If your disk hardware is striping, then data will be striped on stripes. With Disk_Striping set to ON, you will not be able to drop devices from a database.

```
Disk_Striping          Allowable Values: On, Off        Default: On
```

**OS_File_Cache_Buffering** – Raw devices for the IQ-M stores are recommended on all platforms. If you choose file system for your IQ-M Main or Temporary Stores you will need to allow memory for file system caches. However, for NT file systems or Sun Solaris with UFS file systems, you can disable file system buffering by setting this option to OFF. Turning off file system buffering saves a data copy from the file system to the Main IQ-M buffer cache improving performance. See the notes for this option in Chapter 5 of the *Adaptive Server IQ-M Reference Manual* before implementing, particularly for Sun Solaris platforms.

```
OS_File_Cache_Buffering          Allowable Values: On, Off        Default: On
```

**Force_No_Scroll_Cursors** — This option controls query result set buffering. By default, query results are buffered to permit a client tool to scroll forward or backward through an entire result set from a query. The effect of buffering is a slower return of rows to the client, especially with large result sets. For most sites, this option should be turned ON to disable buffering and improve query response. If you attempt to scroll with this option ON, you will likely get an error message “feature not implemented.”

```
Force_No_Scroll_Cursors          Allowable Values: On, Off        Default: Off
```
Main_Cache_Memory_MB — This option sets the size of the IQ-M Main Cache in megabytes. This option is discussed in detail in Part 2.

Main_Cache_Memory_MB Allowable Values: 1 to 4,194,303 Default 8

Temp_Cache_Memory_MB — This option sets the size of the IQ-M Temporary Cache in megabytes. This option is discussed in detail in Part 2.

Temp_Cache_Memory_MB Allowable Values: 1 to 4,194,303 Default 4

Load_Memory_MB — This option controls the amount of memory in the heap when loading data. It has no direct effect on query response. (This single option replaces four other options used to control memory during loads: raw IO buffers and MT buffers.) When loading data records are read from disk files and buffered using memory pages outside of the IQ-M caches, the amount of memory (heap) used is directly related to the input record length. The default allows the heap to use as much memory as needed up to a limit of 500 megabytes. You can calculate approximately the amount of memory used (in megabytes) by dividing the length of the input record by two. For example, a 700-byte input record will use about 350 megabytes of memory. If you fail to control the heap, you may encounter an “out of virtual memory” error during loads. To control the heap size, set this option to a value at or below 500 megabytes. If you set memory too low, you may encounter the error message “The load user approximately <x> MB, but only <y> MB was specified,” where x is the amount required for the heap and y is the limit set. Additionally, setting this option too low may affect the degree of parallelism for loading and slow down the loading process.

Load_Memory_MB Allowable Values: 0 to 500 Default: 0

Append_Load — This new option controls where new rows are placed during Load Table and Insert commands. If rows have been deleted from a table, the default server behavior is to backfill empty rows before appending rows to the table. By setting this option “ON”, all new rows will be appended to the table.

Append_Load Allowable Values: On, Off Default: Off
Part 5 - Load Performance

Loading performance (which includes any data modification activity) is closely tied to the IQ-M server memory configurations that include both the IQ-M caches and the heap. Memory configuration for loading was addressed in detail in Part 3. This part addresses other areas where you can improve load performance, including parallel loading, multifile loading and parallel index creation (new in version 12.4.2).

Parallel (Multithreaded) Loading

With an SMP server, you can improve load performance by employing parallelism during certain phases of a single table load. Certain phases of a load process can be multithreaded using some or all of the processors. While this has always been the case with the IQ-M server, the 12.4.2 server version now issues an advisory message when a load is NOT performed in parallel. Nothing changed in this version to inhibit or improve parallelism, only this advisory note now appears in the IQ-M Message File when the server cannot perform the load in parallel:

No row delimiter has been provided. The insert/load will be single threaded.

The secret to achieving multithreaded loading is simple –

• For variable length records, provide a column delimiter after each column, including the last column, and
• In the Load Table command, specify the row terminator using the "Row Delimited by" statement.

Example:

```
Load Table customer (  
customer_key,  
customer_last_name,  
...  
zipcode)  
From "data/customer.dat"  
Row Delimited by '
'  
...
```

In the above example, the raw data file, customer.dat, is assumed to be comma delimited, and that includes the last field. If the last field (zipcode in this case) is not delimited with a comma, the Load Table command as written will fail.

Multifile Loading for a Table

When you have multiple files to load into a single table and the table has HG indexes, it's always better to load all the files as a single transaction rather than loading and committing each individual file. The syntax is simple: Include all files in the “From” statement of the Load Table command separating each file name with a comma.

Example:

```
Load Table customer (...)  
From "datafile1.txt", "datafile2.txt", "datafile3.txt"  
...  
```
Any data modification transaction involving a table with an HG index requires sorting and some rebuilding of the HG index B-tree to accommodate the new rows. It’s far more efficient to pay the sorting price once with a single transaction than repeatedly manipulating the HG index B-tree for every file.

In a multifile load, the server accesses the files in the “From” statement sequentially as written. No existence or access checking of the files is performed until the server is ready to access the next file. Should any of the files be inaccessible, the entire transaction will by default be rolled back. You have control over this default behavior by using the load option command “On File Error.” This option permits you to specify two alternatives: FINISH or CONTINUE. If a file access error occurs, FINISH directs the server to complete loading of all the files accessed to that point. If CONTINUE is used, the server will skip the problem file, return a non-fatal error to the IQ-M Message file and then continue on to the next file. (CONTINUE cannot be used with partial-width inserts.) Using the previous example:

```
Load Table customer (...) 
From “datafile1.txt”, “datafile2.txt”, “datafile3.txt”
On File Error CONTINUE
...;
commit;
```

directs the server to skip any inaccessible file, go on to the next file and commit when done.

**Improving Incremental Data Loads**

A key to incremental loading performance is to avoid backfilling empty rows that exist when rows have been deleted. After data has been loaded into a table, IQ-M’s default behavior is to search for and fill any empty rows (row IDs) before adding (appending) data to the end of the table. Backfilling is slow and will affect load performance. A new option for loading tables is available in version 12.4.2 that places all new rows at the end of the table. Set the option Append_load = “On” as a public option or as a temporary option in your load script. This new option replaces the “Start Row ID” load option as the method for appending rows to a table.
International Contacts

Argentina  +54 1 990 0421  
Australia  +61 2 9936 8800  
Austria  +43 1504 8510  
Belgium  +32 2 713 1503  
Brazil  +55 11214 4044  
Canada  +1 905 273 8500  
Chile  +56 2 330 6700  
China  +86 10 6856 8488  
Czech Republic  +420 2 2431 0808  
Denmark  +45 3927 7913  
Finland  +358 9 7250 200  
France  +33 1 41 91 96 80  
Germany  +49 69 9508 6182  
Hong Kong  +852 2506 6000  
India  +91 80 228 3850  
Indonesia  +62 21 528 6520  
Italy  +39 02 696 820 64  
Japan  +81 3 5210 6000  
Korea  +82 2 3451 5200  
Malaysia  +60 3 242 4218  
Netherlands  +31 20 346 9290  
New Zealand  +64 4473 3661  
Norway  +47 231 621 45  
Peru  +51 1 221 4190  
Philippines  +63 2 739 2550  
Portugal  +351 21 415 4078  
Singapore  +65 338 0018  
Spain  +34 91 749 7605  
Sweden  +46 8 587 70 433  
Switzerland  +41 1 800 9220  
Taiwan  +886 22 514 8292  
United Kingdom  +44 870 240 2255

Sybase Incorporated  
Worldwide Headquarters  
6475 Christie Avenue  
Emeryville, CA 94608 USA  
Tel: 1-800-8-SYBASE  
Fax: 1-510-922-3210  
www.sybase.com/bi

Copyright © 2000 Sybase, Inc.  All rights reserved.  Unpublished rights reserved under U. S. copyright laws.  Sybase, the Sybase logo and Adaptive Server IQ are trademarks of Sybase, Inc.  All other trademarks are property of their respective owners.  ® indicates registration in the United States.  Specifications are subject to change without notice.  Printed in the U.S.A.
Part #:L01583