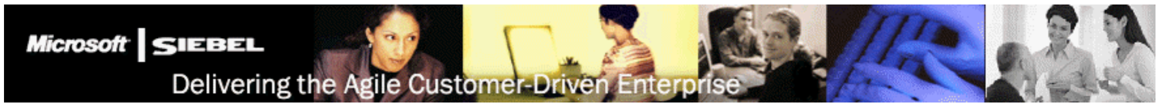

Technical Note #30: EIM Tuning

Technical Note	#xx: Title
Last Modified:	February 1, 2005
Version	1: For latest version see www.siebelonmicrosoft.com
Area:	
Siebel Releases:	Siebel 7.x
Windows Releases	ALL
SQL Server Releases	SQL Server 2000

EIM is one of the most complicated modules within the Siebel application. It validates the data, particularly the primary key and foreign key relationships, and generates a unique ROWID when a row is inserted.

The EIM query is batch oriented, and it is typically more complicated than a normal Siebel query. EIM has better logging, which is essential for resolving problems. EIM reads the system catalog on each run. You can alter the indexes on the EIM tables, but the EIM query is less configurable than a business rule for a normal Siebel query.

There are two types of EIM processing. The first is the Initial EIM, and its purpose, as the name implies, is the initial loading of Siebel databases. A project goal is loading this data as quickly as possible, but for major corporations there is an immense amount of data to be loaded, and the process can take months.

The Initial EIM process will vary for each business but fundamentally it involves the following steps: Data is extracted from various legacy systems into flat files. The flat files are input to some kind of process for data scrubbing and parsing, and output data is loaded into staging tables, referred to as EIM tables. The Siebel EIM process is run, and the Siebel base tables are loaded. This cycle is done in batches with backups at appropriate points throughout the process. The database is tuned as it grows. The Initial EIM is completed before the project is implemented.

Once the base tables are loaded, they can be updated through a batch process called Ongoing EIM. This process has a regular production schedule, and there is normally the demand to finish the Ongoing EIM within a specific time frame. Data can come from various sources. It then goes through various preparation processes which build batch files that the Ongoing EIM uses to insert and update the base tables. Unlike the Initial EIM which has few business people accessing and updating data during the data load, the Ongoing EIM may have many business people using the Siebel system at the same time.

Optimizing the EIM Batch

What typically happens during Initial EIM is that the first few thousand records load quickly. As more records are loaded performance deteriorates. The number of b-trees in the indexes grow, and there are more levels in the index to navigate for each add or update. See the discussion on fragmentation later in this section for additional suggestions. But it is possible that there are other optimization problems. Use this process for optimizing the EIM batch:

1. Turn on 'Level 8' EIM logging.
2. Turn on SQL Profiler for the run.
3. Run an EIM batch.
4. Load the EIM log into Excel. Sort the log on execution time to find what is the longest execution time. Find this event In the unsorted EIM log.

5. Load the SQL Profile trace into Excel. Sort on the longest duration, then on reads. These queries should match those in the EIM log.
6. Review the execution plans. Isolate the problem. Determine how the problem can be fixed. Keep in mind that a combination of solutions may bring about the most improvement.

Performance Tuning for the Initial EIM

There are a number of factors to consider in performance tuning for the Initial EIM:

- The loading strategy
- Indexes
- Turn off optimizer hints
- Fragmentation

The Loading Strategy

It's important to analyze what data is being changed and how the data will be updated. What base tables are updated? Can the batches be separated based on the tables that are updated? If you can do this, you can run batches in parallel with less impact on performance. Will the batches only be inserts of new rows? Will the batch be a mix of inserts, updates, and deletions? Consider separating the batches so that there are separate batches for inserting, updating, deleting, and merging. Will there be so much updating that fragmentation may affect performance? Think about when your organization will place heavy demands on the Siebel system during its normal business cycle.

The default EIM rules may not be the most efficient or the fastest. For example, the default rules require that if a row insert is attempted, and the row already exists, then the row is updated. That is more expensive to process than changing the EIM rule to fail if the row already exists.

Indexes

- Consider dropping all non-clustered indexes on the EIM_ tables. EIM performance usually improves when there is only a clustered index on the EIM_ table.
- You may be able to drop indexes on base tables and put the indexes back selectively.
- Indexes can be tuned. The preferred method is using the Index Tuning Wizard rather than manually tuning the indexes. You can also now use the Database Tuning Advisor found in installations of SQL Server 2005. It is backward compatible to SQL Server 2000 databases and is very effective for tuning indexes.

Turn Off Optimizer Hints

Siebel uses optimizer hints in its queries. Performance may be improved by turning these hints off. Testing should tell you if this strategy is effective.

Test batches to determine if index hints are effective. Index hints can be turned off with these commands:

```
USEINDEXHINTS,FALSE
USEESSENTIALHINTS,FALSE
```

Avoid UPDATES

It is possible to modify the Siebel ifb files to ensure that no updates are performed. An example of this for the S_CONTACT table would be:

```
INSERT ROWS=S_CONTACT,TRUE
UPDATE ROWS=S_CONTACT,FALSE
```

Fragmentation

As databases are loaded, the data can become fragmented resulting in poor performance because more data pages must be read by SQL Server to process a query.

Setting the index options FILLFACTOR and PAD_INDEX reduce fragmentation. The FILLFACTOR option reserves space on each leaf page of an index, while the PAD_INDEX option reserves space in the intermediate index pages. On a table with a clustered index, the leaf level contains the data, so FILLFACTOR controls the space left in the table. These options only apply when the index is built. As rows are added and deleted from the table, the table can become fragmented. When the index is rebuilt, the indexes are rebuilt with the reserve space specified by the two options.

The trick is determining what the appropriate settings should be for FILLFACTOR and PAD_INDEX. If these settings are too low, there will be additional io because the data page is not packed fully as records are inserted. If these settings are too high, it causes unnecessary page splits and rebalancing.

It is not possible to give specific advice on values for these settings. Each customer will be different. In deciding these settings consider how much data is being loading and the maintenance schedule for defragmenting the database. DBCC SHOWCONTIG reports database fragmentation.

The following script generates the commands to defragment a database:

```
select 'DBCC DBREINDEX ('+so.name+','+si.name+',60)' +
char(13)+'go'+char(13)
from sysobjects so, sysindexes si
where
so.id=si.id and
so.id=object_id('S_INSITM2_FN_IF')
-- so.id > 100  to defrag whole database...
```

Performance Tuning for Ongoing EIM

Many of the suggestions made in the previous section for the Initial EIM can apply to the Ongoing EIM.

It's important to analyze what data is being changed and how the data will be updated. Here are some questions and suggestions to consider:

- What base tables are updated? Can the batches be separated based on the tables that are updated? If you can do this, you can run batches in parallel. This helps complete the processing within the available batch window with less impact on performance.
- Will the batches only be inserts of new rows? Will the batch be a mix of inserts, updates, and deletions? Consider separating the batches so that there are separate batches for inserting, updating, deleting, and merging.
- What is the best size for a batch? Many customers start with a batch size between 5,000 and 10,000 records and perform a baseline test. The batch size is adjusted up or down depending on the results and tested until a good batch size is found.

Blocking, Locking and Deadlocks

Blocking occurs when two active tasks try to access the same resource at the same instant. Work is serialized when blocking occurs. If two different people are trying to save a file to the same network location and name, the network processes one of the two requests, and blocks the other until the first one is completed. When the first one task completes, the network attempts to complete the second task.

Locking is when a task takes ownership of a resource prior to performing a task that requires the resource. If a second task requires the resource, it is prevented because of the lock by the first task. For example, when two users try to open the same Word document on the network at the same time, locking prevents one of them from opening the Word document.

Blocking and locking are normal and necessary and protect data integrity. Blocking and locking issues may contribute to performance problems, particularly if queries are timing out. There are clues that blocking and locking are occurring:

- Throughput is limited in the system, but disk I/O and the processor are under utilized.

- The query plan is usually a table scan.
- There are complaints that system response is slow.

There are tools for troubleshooting blocking and locking:

- Knowledge Base Article 298475 – This knowledge base article covers troubleshooting application performance issues in SQL Server.
- Exec sp_who - Provides information about current Microsoft® SQL Server™ users and processes. The information returned can be filtered to return only those processes that are not idle.
- Exec sp_lock – reports information about locks.

A **Deadlock** (also called deadly embrace) occurs when two transactions have conflicting locks on tables that prevent both transactions from proceeding. For example, Transaction 1 requires database A, and places a lock on it. Transaction 2 requires database B, and places a lock on it. Both transactions are processing. Transaction 1 reaches a point in processing where it requires database B, which is still locked by Transaction 2. Database A is still locked by Transaction 1. Transaction 1 is blocked. Transaction 2 continues to process, but reaches a point where it requires database A. Transaction 2 is locked. Neither transaction can proceed, and a deadlock occurs.

The clues that a dead lock is occurring are similar to the ones for locking:

- Throughput is limited in the system, but disk I/O and the processor are under utilized.
- There is lower disk I/O than expected for the given load on the system.
- The query plan is usually a table scan.
- There are complaints that system response is slow.

There is a major cause of deadlocks within the Siebel application for EIM processing. ROWID is a high order column in the clustered index and batch number is a high order column in the non-clustered index of IF tables. When EIM updates the batch to flag the status of the IF table, a bad clustered index can cause unnecessary lock escalation. Putting BATCHNUMBER on the clustered index has the effect of creating a fire wall for each separate EIM job and keeps the EIM jobs from conflicting with each other.

SQL Server provides the stored procedure sp_indexoption to influence the lock behavior of a table.

Ongoing monitoring can help detect blocking, locking, and deadlocks. Periodically run predefined queries as a ping that verifies uptime and a monitor for performance history.