Tech Note 1011
Fine-Tuning the Intelligence Application

Topic#: 002843
Created: January 2014

Introduction

If your Intelligence application is experiencing issues related to any query timeouts, running out of SQL memory, or queries failing due to deadlock and backfill taking longtime to finish, you can tune Intelligence objects in order to optimize their performance.

This *Tech Note* also provides recommendations for periodic maintenance of the Intelligence database.

Application Version

- Intelligence 1.x

Tuning DataSource Objects

The first step is to increase the timeouts listed in Intelligence Technotes **787** & **839**. If issue still persists, verify following:

By default the retrieval query brings back the *entire* data set into the Intelligence Staging database, and then transforms this data set according to the configured Intelligence model. You can build a custom query expression to create custom source data items from the external data source system.

By creating a custom query that filters out unnecessary data from the source system, you reduce the time necessary to move this data across the network, and reduce the processing time in the Staging database. This feature is available at the data source object level and can be configured under the **Custom Queries** tab.

Custom query source data items help you to prepare and optimize the data structure of the external source system for data extraction operations required by the Intelligence model.

Tuning Dimension Objects

Dimensional modeling has to be pursued from a different angle when compared to transactional databases, especially when dealing with the concepts of de-normalization. In general, a dimensional model consists of a few (typically less than a dozen) dimension objects related to a few measure objects, depending upon your application requirements.

The following list provides some of the configurable parameters for a dimension object that you can tune for optimization.

- **Refresh Rate**: The dimension refresh rate determines how often the Intelligence system updates dimension data from the data

source. The dimension refresh period is calculated from the time the dimension object is deployed.

Tune this parameter as per how often source data changes. It is **important** to note that by default all records of a dimension are updated at each refresh execution. It is **NOT** a good idea to set the refresh rate to 1 minute for all objects, especially if the dimensions get very large.

The goal is to distribute the computing resources for the updates of the dimensions and measures. Dimensions that change less frequently, or that are used by measures that are not updated frequently can be set at a slower Refresh Rate.

- **Incremental Update**: For dimensions with large row counts, you may configure the dimension to update incrementally. Incremental updates are based on the **DateTime** field of the dimension. The initial population of dimension data only considers records from the data collection start time and then, at each refresh, records corresponding to the update period.

  For example, if you configure a dimension with a refresh rate of 15 minutes and an update period of 1 hour, then every 15 minutes, the system will query the past hour of data and update the dimension table accordingly.

- **Dimension Links**: Dimension links allow you to create links between dimensions. When defining a measure, you can add linked dimensions to the measure context even if the measure source data has no direct reference to the linked dimension. The measure source data will be inferred from the dimension that defined the dimension link. You can add up to 30 dimension links including the inherited dimension links.

  However, be aware that dimension links can have a significant impact on performance. Use them only when necessary.

Tuning Measure Objects

Measures are typically numerical values computed from source data and put into context with dimensions. The following list shows some of the configurable parameters that can be tuned for optimization.

- **Refresh Rate**: You must provide a refresh rate and update period for the Measure. The Measure refresh rate determines how often the Intelligence system updates a Measure object's data.

  When you deploy a Measure object, the Intelligence Service will first backfill historical data according to the Data Collection Start Time. Measure records are populated in chronological order up to the current time. Next, the Intelligence Service refreshes the measure according to the Refresh Rate, using the Data Collection Start Time as the base time for computing refresh times.

  For **example**, if the Data Collection Start Time is January 1, 2014 00:05:00 and the Refresh Rate is 15 minutes, the Intelligence Service will refresh the measure every day at 00:05:00, 00:20:00. 00:35:00, etc. for a specified updated period from the current time.

  Tune this parameter according to how often the source data changes and avoid refreshing data at smaller time frames. It is **NOT** a good idea to set the refresh rate to 1 minute for all Measures and Dimensions.

  The goal is to distribute the computing resources for the updates of the dimension and measures.

- **Update Period**: This is the time period for which Measure data has to be updated. Be sure to specify that update period equal to more than the refresh rate.

  For **example**, if the measure represents downtime data, and if operators can change the reason codes of downtime at the end of

their shift of 8 hours, set the update period to 8, or even 9 hours.

This means that at each refresh cycle, the Intelligence Service queries the last 9 hours of data and updates the Data Store accordingly.

Note that if your measure data source includes a **Modification Field**, you should set the Update Period equal to the Refresh Rate. The modification field will indicate any record that has changed since the last refresh of the measure table.

Please note that setting a **long Update Period** will likely decrease the performance of refreshes since there will be more data to process. Use this feature wisely.

- **Measure Period**: Measure Period forces the computation of Measure calculations at the specified period. This computation creates individual records in the Measure table for each time slice representing the Measure Period.

  For **example**, a Measure object configured with a measure period of **one hour** stores calculations in the Measure table for every hour since the data collection start time. The system also triggers a check on the configured refresh rate to verify that there is a measure period or dimension time-slices identified since the last refresh. New measure values are calculated for each time-slice defined by measure period and time slicing dimensions and these values are stored in measure table.

  Note that using a small measure period forces the computation of all calculations in the Measure at each time slice.

  For **example**, if you set a Measure Period of 1 minute, the system computes *each* Measure calculation every minute and creates a record in the data store. This represents a minimum of 1440 records per day, per calculation data source (or tag for Historian data). This configuration takes significant processing resources and could represent a lot of data storage.

- **Incremental Update:** When a Measure data source has a column identifying when each of records were created or updated, using the **Incremental Update** feature to capture all changes in the source is strongly recommended. Doing so minimizes the number of records to process.

The Measure object configuration interface allows you to specify the name of the field representing the source record modification timestamp. Ideally, there should be an index on that field in the source system. However, if there is no index on that field, it is likely that a table scan will be done by the source database engine. If you cannot define an index on that field, you can specify the time span of source records that will be considered by setting the Interval value.

For **example**, if you set the interval to one month, the Intelligence Service will first query one month of data based on the Period Field, and then identify the modified records in that data set based on the Modification Field.

Tuning the Intelligence Database

SQL Server is easy to implement, and when the Intelligence database is created and has started collecting data, we often forget that a job is completed.

It is important to note that the SQL Server database engine requires supervision and maintenance. The following list describes rebuilding Indexes, purging transaction logs and performing defragmentation.

Rebuilding Indexes for Intelligence Database

Intelligence creates a series of indexes by default. Depending on your application, several indexes might not be used. However, they can

take significant amounts of space, sometimes ten- to fifteen times the space used by data.

It may therefore be appropriate to disable non-clustered indexes. You can revisit these indexes and enable them based on your application requirements.

Rebuild the Clustered Indexes Periodically

A Clustered Index defines how the data is physically organized in the data file.

For example, a Clustered Index on **aaValuePeriodEndTime_UTC** means that data will be stored according to this field.

You can only have once clustered index per table. All others are non clustered, which means that the result of a search in a non-clustered index is actually a pointer to the physical record identified. A search of a Clustered Index returns the physical records, and not a pointer to the record.

Our recommendation is to disable default indexes and **add** targeted indexes for Measure objects that help the client applications such as dashboards, reports, etc. Doing this will optimize data retrieval.

In most cases, an index including **aaValuePeriodStartTime** or **aaValuePeriodEndTime** is sufficient. They can use either UTC or local time, depending on what the client queries are using.

If you are working with extracts, and those extracts are incremental, the extract queries will likely be based on the **aaValueTimestamp** field. In this case an index on that field will be useful.

With the help of **SQL Server tools** (Database Engine Tuning Advisor) new indexes can be identified for query optimization. A Google Search on **creating cluster indexes** describes how Cluster Indexes can be implemented programmatically.

Truncating Transaction Logs for the Intelligence Database

Truncate the transaction logs periodically. It is not necessary to backup the full Transaction log.

A full Transaction log is not really necessary for Intelligence database, since the database can be reconstructed from sources. If you backup the database on a daily basis and there is an unrecoverable issue with the database, you can restore the latest backup and Intelligence Service will backfill the missing day(s).

Defragmenting the Intelligence Database

Periodically check the data for any fragmentation. If data fragmentation is greater than 30%, defragment the database.

It is always good practice to backup the database prior to making any modifications. Please refer to the Tech Article **Defragmenting SQL Server Databases** on the Wonderware Developer Network for additional info on defragmentation.

S. Mariyala, C.M. Pouyez

Tech Notes are published occasionally by Wonderware Technical Support. Publisher: Invensys Systems, Inc., 26561 Rancho Parkway South, Lake Forest, CA 92630. There is also technical information on our software products at **Wonderware Technical Support.**

For technical support questions, send an e-mail to **wwsupport@invensys.com**.

Back to top