Create a Data Loading Strategy

Presentation by Ryan P. Casey at AIM Report Writing

www.aimreportwriting.com  blog.aimreportwriting.com  @RyanCaseyMBA
Introduction

Create a Data Loading Strategy

Presentation by Ryan P. Casey at AIM Report Writing

www.aimreportwriting.com  blog.aimreportwriting.com  @RyanCaseyMBA
Hire the Intelligence Choice: Business intelligence architect seeking corporate-to-corporate contract opportunities. As a Friend of Redgate Member and a regular SQL Saturday Speaker, Ryan delivers:

- 13+ Fortune 500 Corporations Implementing Data Centers and Data Warehouses
- 15+ years SQL Server and Oracle (T-SQL, PL-SQL), (Excel, Flat Files, XML)
- 12+ years Visual Studio (SSDT Business Intelligence) / (ASP.Net C#, VB.Net)
- 11+ years (Excel), (SSRS, SSIS, SSAS), (Performance Point, Power Pivot, Power Query)

AIM Report Writing combines the collaborative and creative arts included in Technical Communication, the business acumen of a Harvard MBA program, and the technical discipline of object oriented programming and business intelligence (data warehouses).
Presentation Outline

Section 1

- Methods for Loading Large Amounts of Data
  - SSIS Pipeline (Bulk Load) with Type 1 and Type 2 Checksum
  - Merge with Type 1 and Type 2 Checksum with Output
  - SSIS Method using SQL Task to Execute Merge Stored Procedure
  - Others Worth Mentioning
    - Eager Write
Presentation Outline
Section 2

- Load Settings and Features
  - Understanding Minimally Logged Operations
  - Delayed Durability
  - Enabling the Query Store
  - Enhancements to TempDB
  - Statistics On and Updated
  - PolyBase
Presentation Outline

Section 3

- Optimizing Data Loading
  - Surrogate, Primary, Business, Foreign Keys and Indexes
  - Merge Indexes
  - Drop / Build Indexes
  - Server / Query Optimization
Section 1
Methods for Loading Large Amounts of Data
Setting Up This Demonstration

Table Changes

- WideWorldImporters → Sales.Customers
- WideWorldImportersDW → Dimension.Customer
  - Type 2
    - isCurrent
    - Start Date
    - End Date
    - RowChecksum_Type2
  - Type 1
    - Insert Date
    - Update Date
    - RowChecksum_Type1
Setting Up This Demonstration

View Creation

- WideWorldImporters → dbo.LoadTable_DimensionCustomer

**SQL View**: Adds Checksums to Source

```sql
-- View Example
SELECT TOP 5
    -- Type II Columns
    [isCurrent_Type2] -- View Filters on isCurrent_Type2,
    [RowChecksum_Type2]
    -- Type I Columns
    [insertdate_Type1],
    [RowChecksum_Type1]
    -- Table Columns Columns
    , [WideWorldImporters].[dbo].[LoadTable_DimensionCustomer]
FROM [WideWorldImporters].[dbo].[LoadTable_DimensionCustomer]
```

**SSIS Lookup**: Split Inserts and Updates

**SSIS Conditional Split**: Split Type I and Type II

**SQL Merge**: When Matched
SSIS Pipeline (Bulk Load) with Type 1 and Type 2 Checksum

- **SSIS Lookup**
  - Split Inserts / Updates

- **Update Pipeline**
  - Conditional Split
    - Type 2 Checksum
    - Type 1 Checksum

- **Insert Pipeline**
  - Merge
    - New Rows and Type 2 Inserts
Merge using Output with Type 1 and Type 2 Checksum

- SQL Merge Code
- Update
  - WHEN MATCHED
    - Type 2 Checksum
  - WHEN NOT MATCHED
    - Type 2 Inserts
    - Insert Into

**Minimally Logged / Trace Flag 610**
SSIS Method

Using Execute SQL Task using Merge

- SSIS is used in its strengths in terms of being able to Stage using SSIS Data Flows
  - Flat Files, Excel, CSV
  - SQL Server, Oracle, XML, JSON
  - AS400, iSeries, ODBC
- Control the ETL Data Warehouse flow / sequence using Execute SQL Task
- Share meta data throughout the entire ETL Loading Process
- Advantage and flexibility with lower maintenance by using SQL Server Built-In Features, not a custom developed framework as well as enforcing transactions rules and allowing automatic rollback.
Other Methods Worth Mentioning

- SELECT INTO (Eager Write – “a hidden gem”) SQL Server 2014
  - Eager write allows query results using `SELECT INTO #tempdb` to be stored in RAM by not flushing the query results to disk.
  - Eager write is aware that using `SELECT INTO #tempdb` is used for temporary storage that will be read almost immediately

- Bulk Insert
- INSERT / SELECT
- BCP
  - Bulk Copy Program
  - Command Line
Section 2
Load Settings and Features
Understanding Minimally Logged Operations

- We want to Index Reads (source) and Minimally Log Writes (destination)
- Minimally Logged Operations
  - Track Extent Allocations and Metadata Changes Only (not every row change)
- Trace Flag 610
  - Required (enabled) for Some Bulk Loading Methods with Minimally Logged
- Other Minimally Logged and Metadata-Only Operations
  - Drop Table
  - Merge
  - Create, Drop, and Rebuild Indexes
  - Partition Switch (Sliding Window Technique)
### Overview: Minimal Logging Conditions

<table>
<thead>
<tr>
<th>Table Indexes</th>
<th>Rows in Table</th>
<th>Hints</th>
<th>Without Trace Flag 610</th>
<th>With Trace Flag 610</th>
<th>Concurrent Possible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>Any</td>
<td>TABLOCK</td>
<td>Minimal</td>
<td>Minimal</td>
<td>Yes</td>
</tr>
<tr>
<td>Heap</td>
<td>Any</td>
<td>None</td>
<td>Full</td>
<td>Full</td>
<td>Yes</td>
</tr>
<tr>
<td>Heap + Index</td>
<td>Any</td>
<td>TABLOCK</td>
<td>Full</td>
<td>Depends *3</td>
<td>No</td>
</tr>
<tr>
<td>Cluster</td>
<td>Empty</td>
<td>TABLOCK, ORDER *1</td>
<td>Minimal</td>
<td>Minimal</td>
<td>No</td>
</tr>
<tr>
<td>Cluster</td>
<td>Empty</td>
<td>None</td>
<td>Full</td>
<td>Minimal</td>
<td>Yes *2</td>
</tr>
<tr>
<td>Cluster</td>
<td>Any</td>
<td>None</td>
<td>Full</td>
<td>Minimal</td>
<td>Yes *2</td>
</tr>
<tr>
<td>Cluster</td>
<td>Any</td>
<td>TABLOCK</td>
<td>Full</td>
<td>Minimal</td>
<td>No</td>
</tr>
<tr>
<td>Cluster + Index</td>
<td>Any</td>
<td>None</td>
<td>Full</td>
<td>Depends *3</td>
<td>Yes *2</td>
</tr>
<tr>
<td>Cluster + Index</td>
<td>Any</td>
<td>TABLOCK</td>
<td>Full</td>
<td>Depends *3</td>
<td>No</td>
</tr>
</tbody>
</table>

*1 If you are using the INSERT ... SELECT method, the ORDER hint does not have to be specified, but the rows must be in the same order as the clustered index. If using BULK INSERT the order hint must be used.

*2 Concurrent loads only possible under certain conditions. See “Bulk Loading with the Indexes in Place”. Also, only rows written to newly allocated pages are minimally logged.

*3 Depending on the plan chosen by the optimizer, the non-clustered index on the table may either be fully- or minimally logged.
Minimally Logged Operations

- To support high-volume data loading scenarios, SQL Server implements minimally logged operations. Unlike fully logged operations, which use the transaction log to keep track of every row change, minimally logged operations keep track of extent allocations and metadata changes only.

Because much less information is tracked in the transaction log, a minimally logged operation is often faster than a fully logged operation if logging is the bottleneck. Furthermore, because fewer writes go to the transaction log, a much smaller log file with a lighter I/O requirement becomes viable.

- Understand that an operation can be a bulk load operation without being minimally logged. For example, you can bulk load data into clustered indexes or even heaps without getting minimal logging. Minimal logging typically provides an extra speed benefit, but even without the minimal logging, bulk load has less overhead than traditional row by row inserting of data.

- Contrary to the SQL Server myths, a minimally logged operation can participate in a transaction. Because all changes in allocation structures are tracked, it is possible to roll back minimally logged operations.
Trace Flag 610

The trace flag can be turned on by using one of the following methods:

- Adding to the SQL Server startup parameters
- At Runtime
  - This enables the trace flag for a specific session. This is useful if you want to enable 610 for only a subset of load scenarios on the instance, and it applies only to the Transact-SQL connection that issues it.
  - Use turns on the trace flag for all connections to the server until it is turned off or until the next server restart.

Before you start using this trace flag, be aware of these limitations

- Not every row inserted in a cluster index with trace flag 610 is minimally logged. When the bulk load operation causes a new page to be allocated, all of the rows sequentially filling that new page are minimally logged. Rows inserted into pages that are allocated before the bulk load operation occurs are still fully logged, as are rows that are moved as a result of page splits during the load. This means that for some tables, you may still get some fully logged inserts.
- If trace flag 610 causes minimal logging to occur, you should generally see a performance improvement. But as always with trace flags, make sure you test for your specific environment and workload.

I/O Impact of Minimal Logging Under Trace Flag 610

- When you commit a bulk load transaction that was minimally logged, all of the loaded pages must be flushed to disk before the commit completes. Any flushed pages not caught by an earlier checkpoint operation can create a great deal of random I/O. Contrast this with a fully logged operation, which creates sequential I/O on the log writes instead and does not require loaded pages to be flushed to disk at commit time.
- If your load scenario is small insert operations on btrees that do not cross checkpoint boundaries, and you have a slow I/O system, using minimal logging can actually slow down insert speeds.
Delayed Durability (SQL Server 2014)

- Why can Delayed Durability give a throughput boost?
- Here is a workload with a small table where 50 concurrent clients are updating the same rows, and the database log is on a slow I/O subsystem.
- The change point is where delayed durability was enabled on all transactions. Before the change, the number of Transactions/sec is equal to the number of Log Flushes/sec, as each transaction is holding locks that block all other transactions.

Why the jump in Transactions/sec when using Delayed Durability?

Under normal circumstances, when a transaction commits, the commit doesn’t complete until the log block containing the LOP_COMMIT_XACT log record for the transaction has been flushed to disk and the write is acknowledged back to SQL Server as having completed, providing the durability of the transaction (the D in the ACID properties of the transaction). The transaction’s locks cannot be dropped until the log flush completes.

Before the change point, all the transactions are waiting for the one that is committing, so Transactions/sec is tied to Log Flushes/sec. With delayed durability, the transaction commit proceeds without log block flush – making the transaction’s durable (ACID) delayed. Under delayed durability, log blocks are only flushed to disk when they reach their maximum size of 60KB. This means that transactions commit a lot faster and hold their locks for less time. Thus, in this example, Transactions/sec increases greatly. Also the Log Flushes/sec decreased greatly.

**WARNING!** Transactions aren’t durable when they commit. If the last log block contains the LOP_COMMIT_XACT log record for a long-running transaction, if the system crashes, and that log block isn’t on disk, that whole transaction will roll back during crash recovery. So the potential for work/data loss is greater than just 60KB. Log backups will not back up that unflushed log block, as it’s not on disk, so non-durable committed transactions may not be contained within a log backup. Non-durable transactions that have committed are not protected by synchronous database mirroring or a synchronous availability group either, as these rely on log block flushes.
SQL Server Statistics

- **Statistics On and Updated Effect**
  - Linked Servers
  - Columns being used in JOIN, WHERE, ORDER BY, or GROUP clauses
  - SQL Server Query Optimizer creates single column statistics when the AUTO_CREATE_STATISTICS database property is set to ON, or when you create indexes on the table or views (statistics are created on the key columns of the indexes)

- **SQL To Enable and Disable (SQL Exemplar)**
  - Turning on Statistics info
    - SET STATISTICS IO ON-- returns info regarding amount of disk activity
    - SET STATISTICS TIME ON-- returns run time in milliseconds
    - SET STATISTICS PROFILE ON-- returns execution plan info in addition to regular result set
    - SET STATISTICS XML ON-- produces an xml output document for each statement that executes
Substantial Data Changes with Insert, Update, Delete, or Merge

Substantial data change operations (like insert, update, delete, or merge) change the data distribution in the table or indexed view and make the statistics go stale or out-of-date, as it might not reflect the correct data distribution in a given column or index. SQL Server Query Optimizer identifies these stale statistics before compiling a query and before executing a cached query plan. The identification of stale statistics are done by counting the number of data modifications since the last statistics update and comparing the number of modifications to a threshold as mentioned below.

- A database table with no rows gets a 1st / new row
- A database table had fewer than 500 rows when statistics was last created or updated and is increased by another 500 or more rows
- A database table had more than 500 rows when statistics was last created or updated and is increased by 500 rows + 20 percent of the number of
SQL Server Statistics DMVs

```sql
USE the dynamic management function (sys.dm_db_stats_properties) to retrieve statistics properties with further details.
```

<table>
<thead>
<tr>
<th>TableName</th>
<th>StatisticsName</th>
<th>last_updated</th>
<th>rows_sampled</th>
<th>rows</th>
<th>unfiltered_rows</th>
<th>steps</th>
<th>modification_counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>SalesOrderHeader</td>
<td>_WA_Sys_00000008_3C34F16F</td>
<td>2012-03-29 13:52:23 93000000</td>
<td>31465</td>
<td>31465</td>
<td>31465</td>
<td>156</td>
<td>0</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>_WA_Sys_00000009_3C34F16F</td>
<td>2012-03-29 13:52:23 93000000</td>
<td>31465</td>
<td>31465</td>
<td>31465</td>
<td>156</td>
<td>0</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>_WA_Sys_00000010_3C34F16F</td>
<td>2012-03-29 13:52:23 93000000</td>
<td>31465</td>
<td>31465</td>
<td>31465</td>
<td>156</td>
<td>0</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>_WA_Sys_00000011_3C34F16F</td>
<td>2012-03-29 13:52:23 93000000</td>
<td>31465</td>
<td>31465</td>
<td>31465</td>
<td>156</td>
<td>0</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>_WA_Sys_00000012_3C34F16F</td>
<td>2012-03-29 13:52:23 93000000</td>
<td>31465</td>
<td>31465</td>
<td>31465</td>
<td>156</td>
<td>0</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>_WA_Sys_00000013_3C34F16F</td>
<td>2012-03-29 13:52:23 93000000</td>
<td>31465</td>
<td>31465</td>
<td>31465</td>
<td>156</td>
<td>0</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>_WA_Sys_00000014_3C34F16F</td>
<td>2012-03-29 13:52:23 93000000</td>
<td>31465</td>
<td>31465</td>
<td>31465</td>
<td>156</td>
<td>0</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>_WA_Sys_00000015_3C34F16F</td>
<td>2012-03-29 13:52:23 93000000</td>
<td>31465</td>
<td>31465</td>
<td>31465</td>
<td>156</td>
<td>0</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>_WA_Sys_00000016_3C34F16F</td>
<td>2012-03-29 13:52:23 93000000</td>
<td>31465</td>
<td>31465</td>
<td>31465</td>
<td>156</td>
<td>0</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>_WA_Sys_00000017_3C34F16F</td>
<td>2012-03-29 13:52:23 93000000</td>
<td>31465</td>
<td>31465</td>
<td>31465</td>
<td>156</td>
<td>0</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>_WA_Sys_00000018_3C34F16F</td>
<td>2012-03-29 13:52:23 93000000</td>
<td>31465</td>
<td>31465</td>
<td>31465</td>
<td>156</td>
<td>0</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>_WA_Sys_00000019_3C34F16F</td>
<td>2012-03-29 13:52:23 93000000</td>
<td>31465</td>
<td>31465</td>
<td>31465</td>
<td>156</td>
<td>0</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>_WA_Sys_00000020_3C34F16F</td>
<td>2012-03-29 13:52:23 93000000</td>
<td>31465</td>
<td>31465</td>
<td>31465</td>
<td>156</td>
<td>0</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>_WA_Sys_00000021_3C34F16F</td>
<td>2012-03-29 13:52:23 93000000</td>
<td>31465</td>
<td>31465</td>
<td>31465</td>
<td>156</td>
<td>0</td>
</tr>
<tr>
<td>SalesOrderHeader</td>
<td>_WA_Sys_00000022_3C34F16F</td>
<td>2012-03-29 13:52:23 93000000</td>
<td>31465</td>
<td>31465</td>
<td>31465</td>
<td>156</td>
<td>0</td>
</tr>
</tbody>
</table>
```
Other Features Worth Mentioning

- **Enabling the Query Store**
  - SET = ON, OFF : SET = Read_Only, Read_Write : Read_Only Limitation

- **Enhancements to TempDB**
  - These operations heavily use tempdb
    - Repeated create and drop of temporary tables (local or global)
  - SQL Server 2016 Supports Multiple tempdb (1 per processor)

- **PolyBase**
  - Query (load data) Hadoop in SQL Server 2016
Section 3
Optimizing Data Loading
Index Considerations

- What to consider when indexing the data warehouse?
  - We want to Index Reads (source) and Minimally Log Writes (destination)
  - What type of Data Warehouse do we have?
    - Primarily archive or primarily near real-time?

How large are the dimension and fact tables?
- Are the fact tables partitioned?

- Who will be accessing the data and how?
  - Excel, SQL Server Management Server (SSMS), Reporting Services, Web Services?

- Will access be ad hoc or via structured application interfaces?
Index Strategy Overview
Index the Reads and Minimally Log the Writes

<table>
<thead>
<tr>
<th>Dimensions</th>
<th>Non-Clustered on Dimension Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension (Surrogate) Key</td>
<td>Non-Clustered on Dimension Key</td>
</tr>
<tr>
<td>Business Key</td>
<td>Clustered on Business Key</td>
</tr>
<tr>
<td>Type II Slowly Changing Dimension</td>
<td>4-Part Non-Clustered on Business Key, Record Start Date, Record End Date, and Surrogate Key</td>
</tr>
<tr>
<td>Data and Time Dimensions</td>
<td>Cluster on Date and Time Dimension using a smart key (YYYYMMDD, HHMMSSSSSS)</td>
</tr>
<tr>
<td>Dimension Hierarchies</td>
<td>Non-Clustered on Dimension Hierarchies like Category --&gt; Sub-Category --&gt; ProductID</td>
</tr>
<tr>
<td>Report Predicate</td>
<td>Non-Clustered on Report Predicate (columns used in the where, or join clause)</td>
</tr>
<tr>
<td>Report Select Criteria</td>
<td>Covering Index on Report Selection Criteria</td>
</tr>
<tr>
<td>Other Columns</td>
<td>Non-Clustered on Columns the will be continuously Searching, Sorting, and Grouping</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Facts</th>
<th>Clustering on Most often Used Date and Time Keys using a smart key (YYYYMMDD, HHMMSSSSSS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data and Time Keys</td>
<td>Clustering on Most often Used Date and Time Keys using a smart key (YYYYMMDD, HHMMSSSSSS)</td>
</tr>
<tr>
<td>Partitioning</td>
<td>If the fact table is partitioned on the date column, use this column as the clustering key to align the index and table</td>
</tr>
<tr>
<td>Foreign Keys</td>
<td>Non-Clustered on each of the foreign keys in the index</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Staging</th>
<th>Typically, there is no indexing in Staging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensions, Facts, and All Other Tables</td>
<td>Typically, there is no indexing in Staging</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Merge</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Table</td>
<td>Unique Non-Clustered Covering on Join Columns in the Source Table</td>
</tr>
<tr>
<td>Target Table</td>
<td>Unique Clustered on Join Columns in the Target Table</td>
</tr>
<tr>
<td>ON Clause</td>
<td>Specify only Search Conditions in the ON clause from the target table that are compared to the corresponding columns in the source table</td>
</tr>
</tbody>
</table>
Index Strategy Diagram

**dimDate**
- DateKey (MMDDYYYY) Clustering Index
- Date (datetime)
- Year
- Quarter
- Month

**dimCustomer**
- CustomerKey (SK) Non-Clustered Index
- CustomerID (BK) Clustered Index
- Customer Name Non-Clustered Index

**dimTime**
- TimeKey (HHMMSSSSS)
- Hour
- Minute

**factSales**
- DateKey (MMDDYY) Clustering Index
- TimeKey (HHMMSSSSS) Non-Clustered Index
- CustomerKey Non-Clustered Index on CustomerKey & DateKey
- ProductKey Non-Clustered Index
- Quantity Sold
- Unit Price

**dimProduct**
- ProductKey (SK) Non-Clustered Index
- ProductID (BK) Clustered Index
- Product Category Non-Clustered Index
Dimension Table Index Strategy

- **Non-Clustered on Dimension Key**
  - primary key, which is a surrogate key with an identity
  - not a “natural” or transactional key such as customer name or customer ID
  - Don't cluster on the dimension key

- **Clustered on Business key**
  - Natural” or transactional key such as customer name or customer ID
  - Enhances the query response when the business key is used in the WHERE clause
  - Often used to search the dimensional data, and having the dimension records pre-sorted makes the query response faster
Dimension Table Index Strategy (Covering Indexes)

- Four-Part Non-Clustered on Large Type 2 Slowly Changing Dimensions
  - business key, the record begin date, the record end date, and the surrogate key
  - For efficiency and to prevent escalating storage requirements, INCLUDE the record end date and the surrogate key when creating the index instead of making them part of the index key, as shown in the following command:

```sql
CREATE NONCLUSTERED INDEX MyDim_CoveringIndex
    ON (NaturalKEY, RecordStartDate) INCLUDE (RecordEndDate, SurrogateKEY);
```

INCLUDE EndDate and CustomerKey (SK) instead of making them part of the index key, so SQL Server stores these values at the leaf level only of the index tree, reducing the storage requirements.

INCLUDING these 2 columns in the index (covering index), SQL Server can get the data that it needs solely from the index during the load without having to make an additional trip for data from the underlying dimension table.
Dimension Table Index Strategy

- Cluster on **Date Dimension and the Time Dimension**
  - use a smart key (int, bigint)    **Date**: YYYYMMDD    **Time**: HHMMSSSSSS
    - Maintains index order, range queries simplified in the fact table
    - One less join when querying because the primary key will contain the date (or time)

- Non-Clustered on **Dimension Hierarchies**
  - Category → SubCategory → ProductID hierarchy in the Product Dimension
  - Consider indexing the components of the hierarchy if it will enhance query performance and won’t inhibit data loading

- Non-Clustered on **Other Dimension Columns** that will be…
  *Used Continuously for Searching, Sorting, or Grouping*
Fact Table Index Strategy

- Indexing the fact table is similar to indexing a dimension
  - However, you must account for partitioning!

- Business intelligence (BI) analysis often involves a date/time component
  - Index and cluster on the Date Key or a combined Date plus Time Key
  - Clustering on this key will help with cube-building

- Date already stored in calendar (date time) order
  - Historical queries will have an execution advantage

- When the fact table has more than one date or date time column,
  - Cluster on the column that’s used most often for querying or cube-building
Fact Table Index Strategy

- If the **Fact Table is Partitioned** on the Date Column, use that **column as the clustering key**

- When you use the same column to create the clustered index, that you used to create the partitions and creating the index (in the same file group) that holds the partitioned fact table, SQL Server will automatically partition the index the same way that the fact table is partitioned
  - the index will have the same partitioning function and column as the fact table

- When the index is partitioned the same way the fact table is partitioned, the table and its index are said to be aligned
  - makes for an optimal operational situation, especially if you anticipate creating additional partitions or making frequent partition switches
Fact Table Index Strategy

- Create a Non-Clustered index on each of the foreign keys in the fact table
- Consider combining the foreign key and the date key (in that order)
- Creating a non-clustered key on the foreign keys
  - works especially well if one or more of the associated dimensions are a type 2 slowly changing dimension
  - Rows with the same foreign key value will be searched in ascending date order, which will enhance the historical query response

Note: Always retain relational integrity when dealing with the foreign keys
Staging Table Index Strategy

- Staging tables may bear little resemblance to tables in the data warehouse
  - Landing places for transactional and non-transactional data sources as they enter the data warehouse environment
  - Intermediate steps in the checking, scrubbing, separating, decoding, validating, and verifying portions of the ETL process
  - Store operational audit logs

- Typically, indexing these tables is severely limited

  - Most, if not all staging tables have no indexes
  - Data integrity is managed and maintained by the ETL process
Merge Indexes

- To improve the performance of the MERGE statement
  - Create an index on the join columns in the source table that is unique and covering.
  - Create a unique clustered index on the join columns in the target table.

- Specify only search conditions in the ON <merge_search_condition> clause that determine the criteria for matching data in the source and target tables.
  - That is, specify only columns from the target table that are compared to the corresponding columns of the source table. Do not include comparisons to other values such as a constant.
Drop and Build Indexes
Index the Reads and Minimally Log the Writes

- Enable / Disable All Indexes Scripts on Dimension.Customer
Request, Session Optimization
Blocking and Buffer Cache

Blocking
- View using sys.dm_exec_requests, sys.dm_exec_sessionsm and sys.dm_exec_text
- Table tblBlockingDetails to store blocking details with (SQL Handle, Plan Handle)
- Stored Procedure uspInsertBlockingRecords to insert blocking details into table from view
- Trigger Alert_tblBlockingDetails_Insert to send email and text blocking alerts
- Job InsertBlockingRecords to schedule this stored procedure every 3 minutes

Buffer Cache
- sys.dm_os_sys_info
- sys.dm_os_buffer_descriptors
- sys.allocation_units
- sys.partitions
Server / Query Optimization
Page Free Space and Performance Counters

- PFS (Page Free Space) Contention
  - Table variables that use tempdb for storage purposes
  - Heavy and significant use of these activities may lead to the contention problems
    - Work tables associated with CURSORS
    - Work tables associated with an ORDER BY clause
    - Work tables associated with an GROUP BY clause
    - Work files associated with HASH PLANS

- Worth Mentioning
  - SQL Server Configuration
  - Query Plan Estimate and Actual
  - Query Store
  - Optimizing I/O and File Layout

Performance Counters

SELECT [object_name], [counter_name], [cntr_value]
FROM sys.dm_os_performance_counters
WHERE [object_name] LIKE '%Manager%
AND [counter_name] = 'Page life expectancy'
Conclusion

Create a Data Loading Strategy

Presentation by Ryan P. Casey at AIM Report Writing

www.aimreportwriting.com  blog.aimreportwriting.com  @RyanCaseyMBA
Questions and Answers

Create a Data Loading Strategy

Presentation by Ryan P. Casey at AIM Report Writing

www.aimreportwriting.com  blog.aimreportwriting.com  @RyanCaseyMBA