

SQLskills Immersion Event

IETS: Advanced Transact-SQL Programming

Module 7: Optimizing Actions

Bob Beauchemin
BobB@SQLskills.com



Overview

- **INSERTing Data**
 - INSERT...SELECT
 - INSERT...EXECUTE
 - INSERT...INTO
- **UPDATEing Data**
 - Update using a JOIN or subquery
- **DELETEing Data**
 - Bulk delete
- **MERGE**
- **TOP, ROWCOUNT and Actions**
- **Output from Action Statements**
- **Inserting in bulk**



2

© SQLskills, All rights reserved.
<http://www.SQLskills.com>



Row Constructors

- Use VALUES clause to construct a set of rows
- Insert multiple rows based on values in a single
- INSERT statement
- SQL 2006 standard compatible

```
DECLARE @t TABLE (id int, name varchar(20));  
INSERT INTO @t VALUES  
    (1, 'Fred'), (2, 'Jim'), (3, 'Sue');
```

Using the INSERT...SELECT Statement

- All rows that satisfy the SELECT statement are inserted
 - Verify that the table that receives new row exists
 - Must ensure that data types are compatible
 - Determine whether default values exist or whether null values are allowed
 - Derived columns, constants, and defaults can be used as input

INSERT...SELECT

```
USE northwind
INSERT customers
SELECT substring (firstname, 1, 3) +
      substring (lastname, 1, 2) AS name
      , lastname, firstname, title
      , address, city, region, postalcode
      , country, homephone, NULL
FROM employees
```

Using INSERT...EXECUTE

- **Allows population from stored procedure**
 - Number and data type of columns must match
 - Procedure must return a single rowset
 - Or all rowsets must have exactly the same shape
 - Procedure can be on remote server
 - Limitations for (N)TEXT
 - SELECT statement in stored procedure cannot contain CTE
 - Useful for storing information from some DBCC commands

SELECT...INTO

- **SELECT...INTO creates a table from resultset**
 - Used for producing intermediate temp table for reshaping data then SELECTing
 - Must have CREATE TABLE privilege, except when creating temp table
 - Can be used to create an empty copy an empty copy of existing table
 - Does not require non-logged mode
 - Can be parallelized in SQL Server 2014

Updating Rows Based on Other Tables

- **Using the UPDATE Statement**
 - Updates a row once
 - Updates columns or variable names that follow the SET keyword
- **Using Joins**
 - Uses the FROM clause
 - Some issues when using this...more later
- **Using Subqueries**
 - Returns a single value for each row

Indirect Updates

```
-- UPDATE from a correlated subquery
UPDATE titles
SET YTD_SALES = (SELECT ISNULL(SUM(qty),0)
                 FROM sales AS s
                 WHERE s.title_id = t.title_id)
FROM titles as t
```

```
-- UPDATE from a JOIN
UPDATE od
SET Discount = Discount + 0.02
FROM [Order Details] AS od
JOIN Products AS p ON od.productid = p.productid
WHERE p.SupplierID = 5
```

Deleting Rows

- **Can delete rows using additional tables**
 - Similar to UPDATE
 - Using Additional FROM Clause
 - First FROM clause indicates table to modify
 - Second FROM clause specifies restricting criteria for the DELETE statement
- **Specifying Conditions in the WHERE Clause**
 - Subqueries determine which rows to delete
- **TRUNCATE TABLE is fastest way to delete all rows in a table**
 - Transactional, but its not logged
 - Triggers do not fire

Plans for Action Statements

- **Consider UPDATE statement**
 - Find rows to be updated
 - Calculate new values
 - Update table
 - Update related structures – indexes, indexed views
- **INSERT and DELETE perform a subset of this**
- **Narrow vs. wide plans**
 - Narrow – all indexes updated in one iterator
 - Wide – multiple update iterators
- **Split-Sort-Collapse optimization to avoid uniqueness violations**
 - Split update into insert + delete
 - Sort by value, action – deletes sort before inserts
 - Remove insert/delete pairs for same value

MERGE Statement

- **Multiple set operations in a single SQL statement**
- **Uses multiple sets as input**
 - MERGE target USING source ON ...
- **Operations can be INSERT, UPDATE, DELETE**
- **Operations based on**
 - WHEN MATCHED
 - WHEN NOT MATCHED [BY TARGET]
 - WHEN NOT MATCHED BY SOURCE
- **ANSI SQL 2006 compliant - with extensions**

More on MERGE

- **MERGE statement can reference a \$action column**
 - Used when MERGE used with OUTPUT clause
- **Multiple WHEN clauses possible**
 - For MATCHED and NOT MATCHED BY SOURCE
 - Only one can be an update
 - Only one WHEN clause for NOT MATCHED BY TARGET
- **MERGE can be used with any table source**
- **A MERGE statement causes triggers to be fired once**
- **Rows affected includes total rows affected by all clauses**

MERGE Performance

- **MERGE statement is transactional**
 - No explicit transaction required
- **One Pass Through Tables**
 - At most a full outer join
 - Matching rows = when matched
 - Left-outer join rows = when target not matched
 - Right-outer join rows = when source not matched

MERGE and Determinism

- **UPDATE using a JOIN is non-deterministic**
 - If more than one row in source matches ON clause, either/any row can be used for the UPDATE
- **MERGE is deterministic**
 - If more than one row in source matches ON clause,
 - its an error

TOP With INSERT,UPDATE,DELETE

- **This is meant to replace SET ROWCOUNT**
 - Cannot UPDATE/DELETE the higher/lowest because no ORDER BY in these statements
 - Allow batching with optimization

```
DECLARE @batch_size AS INT
SET @batch_size = 100

WHILE (1=1)
BEGIN
    UPDATE TOP(@batch_size) licenses SET expired = 1
    WHERE expire_date < GETDATE()
    AND expired = 0
    IF @@rowcount < @batch_size BREAK
END
GO
```


DML with Results (OUTPUT Clause)

- **New OUTPUT clause which facilitates the return of data by referencing inserted and deleted tables**
 - In INSERT statements, refer to INSERTED to retrieve columns from the newly inserted rows
 - In DELETE statements, refer to DELETED to retrieve columns from the deleted rows
 - In UPDATE and MERGE statements, refer to DELETED to return the old image of the updated rows, and to INSERTED to return their new image

DML with Results (OUTPUT Clause)

- **OUTPUT DELETED.* in the DELETE statement below, returns all columns deleted from the ShoppingCartItem table to the table variable**

```
DECLARE @MyTableVar TABLE (...)  
  
-- OK for table or variable  
-- must name columns if column list  
DELETE Sales.ShoppingCartItem  
OUTPUT DELETED.* INTO @MyTableVar  
WHERE ProductID = 862
```

DML with Results (OUTPUT Clause)

- **Restrictions and Requirements**
 - Not supported in any DML statements referencing local or distributed partitioned views, or remote tables
 - Columns returned from OUTPUT reflect the data as it is once the INSERT, UPDATE, or DELETE statement has completed, but before triggers are executed
 - OUTPUT can be used with an UPDATE or DELETE statement within a cursor. (no guarantee on sort order)
 - If parameters or variables are modified in an UPDATE statement, the OUTPUT clause always returns the pre-change value
 - E.g don't use @@IDENTITY, use INSERTED

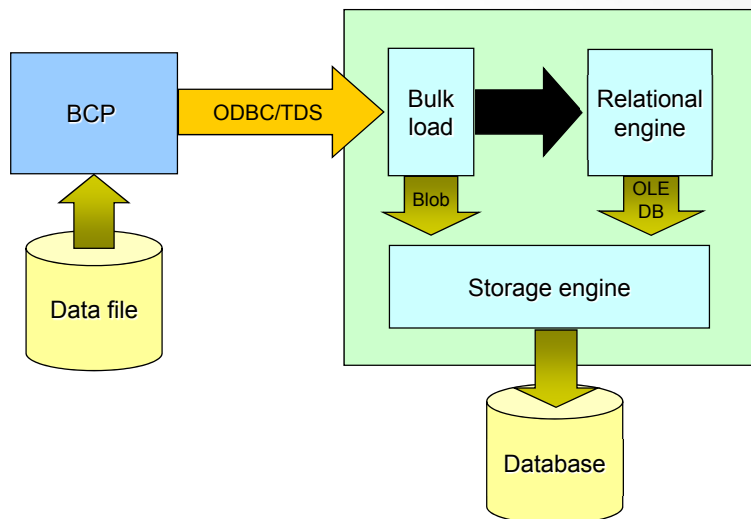
Bulk Loading Overview

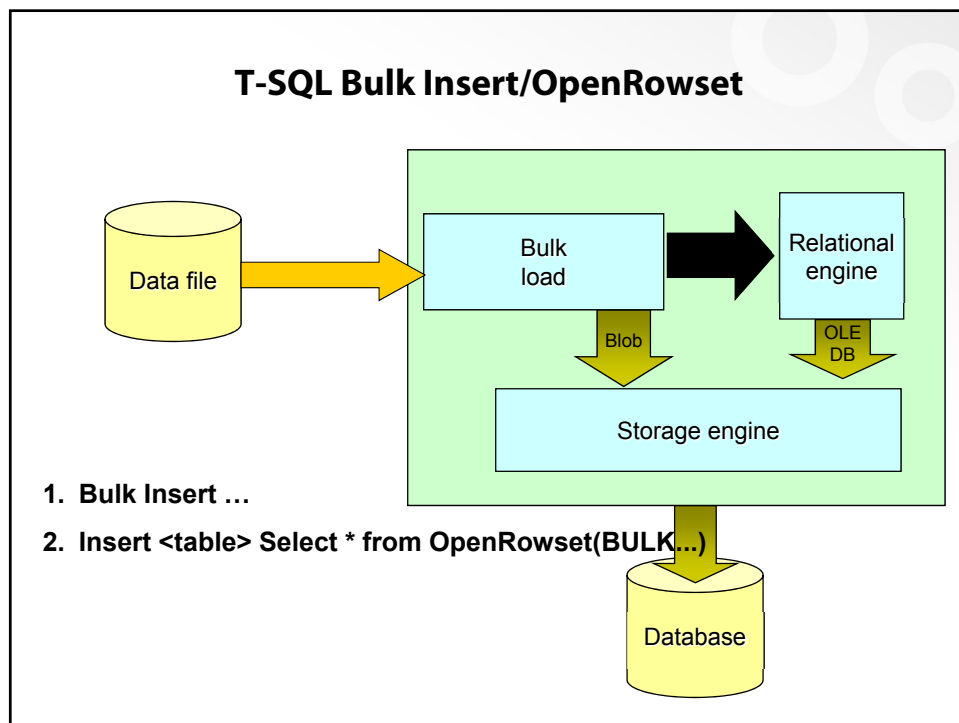
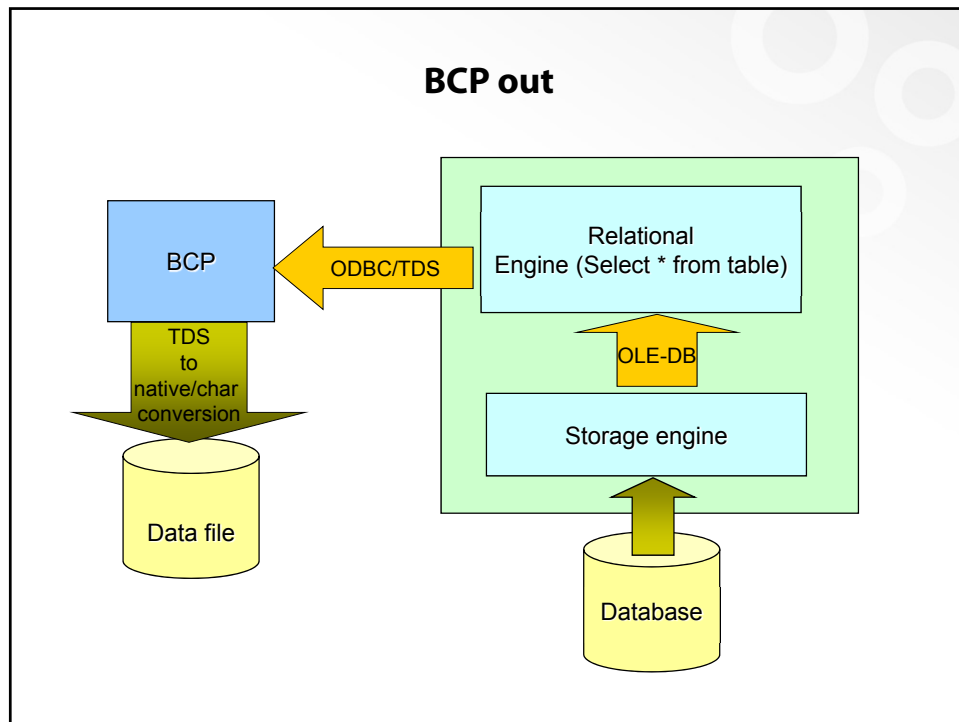
- **Set of T-SQL commands and tools to import/export data into/out of SQL Server**
- **Providing:**
 - Robust error handling and data validation
 - Allowing simple data transformations during load
 - Supporting flexible layout of source data through (structured) format files
 - Some provide ability to load BLOBs as singletons
- **Performance Optimizations**
 - Optimized insert through special logging, locking, and constraint checking
 - Skipping constraints/triggers
 - Some support parallel Data loading

Bulk Loading in SQL Server

- **Bulk Copy client application (BCP.EXE)**
 - Supports both import/export of data
 - Data conversion validation at client
- **Bulk Insert**
 - T-SQL command
 - Only allows import of data
- **OPENROWSET BULK Function**
 - Enables data validation without loading the data
 - Support simple (inline) transformations during import
 - Bulk loading of data through normal INSERT path
 - Loading of single BLOB objects
- **SSIS SQL Bulk Load task (fastest)**

BCP in





OpenRowset BULK Rowset Provider

```
OPENROWSET
(BULK 'data_filename'
 [ , CODEPAGE = 'ACP' | 'OEM' | 'RAW' | 'code_page' ]
 [ , FIELDTERMINATOR = 'field_terminator' ]
 [ , FIRSTROW = first_row ]
 [ , FORMATFILE = 'format_file_path' ]
 [ , LASTROW = last_row ]
 [ , ROWTERMINATOR = 'row_terminator' ]
 [ , MAXERRORS = 'max_errors' ]
 [ , ERROR_FILE = 'file_name' ]
 [ , {SINGLE_BLOB | SINGLE_CLOB | SINGLE_NCLOB} ]
 )
```

Bulk Loading using OpenRowset

▪ Benefits:

- Full power of SELECT
- Data validation with no load

```
INSERT      into table <target-table>
WITH        (IGNORE_CONSTRAINTS, IGNORE_TRIGGERS)
SELECT      c1, c2, c3
FROM        OpenRowset(BULK, 'c:\data.txt')
AS          MyTable (c1, c2, c3)
WHERE c1 > 10
```

Implementation Guidelines

- Meant for import/export of single table data.
- Export does not output schema information so you need to create table before import.
- If exporting related tables (e.g. foreign key), the export may be inconsistent
 - With Snapshot Isolation level, you can get consistent export. Single-statement exports can also get consistent results in a versioning-enabled database
- Blocks if there are rows with X lock.
- Not a substitute for physical backup

"Non-Logged" Operations

- Better be named "Minimally Logged" Operations!
- In a "minimally" logged situation only page extents are logged, to undo page allocations during rollback
- Possible "Non-Logged" Operations:
 - SELECT INTO
 - BULK INSERT
 - INSERT SELECT FROM OPENROWSET
 - CREATE INDEX
 - WRITETEXT, UPDATETEXT
 - TRUNCATE TABLE
 - ODBC, OLE DB, and ADO.NET based BCP
 - IRowsetFastLoad (used by SSIS & SQLXMLBulkLoad)
- Non-logged operations depend on the Database Recovery Model

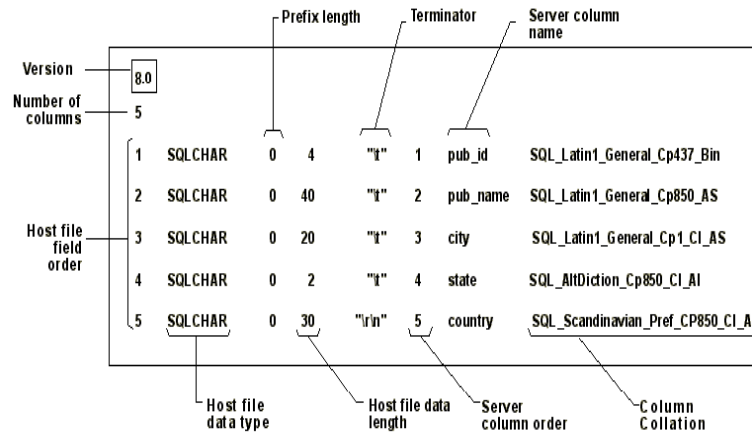
Performance Advantages...

- **Optimized logging Behavior**
 - Incremental bulk load is bulk-logged when no indexes.
 - If no clustered index, data pages are bulk-logged but not indexes
 - Only logs page allocations, not individual rows
 - SQL Server 2005/2008 optimizations:
 - For incremental bulk load: Generates one log record for all rows in the newly allocated page on a table with indexes
- **Parallel load (with logging optimizations)**
 - Conditions:
 - Heap with no non-clustered indexes
 - Each invocation of command loads one file.
 - Must specify TABLOCK
- **Minimal logging on BTree - INSERT and MERGE statements**
 - Must turn on TF610
- **SELECT INTO can operate in parallel (SQL 2014)**

Format File

- **SQL Server 2000 limitations:**
 - Hard to read or extend
 - Only stores how data is stored in the file
 - Column type information is lost if data file contains character representation
- **SQL Server 2005+ solution:**
 - XML Based Format Files

SQL Server 2000 Format File



XML Format File

```
<?xml version="1.0"?>
<BCPFORMAT
xmlns="http://schemas.microsoft.com/sqlserver/2004/bulkload/format"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <RECORD>
    <FIELD ID="1" xsi:type="CharTerm" TERMINATOR="," MAX_LENGTH="7"/>
    <FIELD ID="2" xsi:type="CharTerm" TERMINATOR="," MAX_LENGTH="100"
COLLATION="SQL_Latin1_General_CP1_CI_AS"/>
    <FIELD ID="3" xsi:type="CharTerm" TERMINATOR="," MAX_LENGTH="100"
COLLATION="SQL_Latin1_General_CP1_CI_AS"/>
    <FIELD ID="4" xsi:type="CharTerm" TERMINATOR="\r\n" MAX_LENGTH="100"
COLLATION="SQL_Latin1_General_CP1_CI_AS"/>
  </RECORD>
  <ROW>
    <COLUMN SOURCE="1" NAME="Col1" xsi:type="SQLSMALLINT"/>
    <COLUMN SOURCE="2" NAME="Col2" xsi:type="SQLNVARCHAR"/>
    <COLUMN SOURCE="3" NAME="Col3" xsi:type="SQLNVARCHAR"/>
    <COLUMN SOURCE="4" NAME="Col4" xsi:type="SQLNVARCHAR"/>
  </ROW>
</BCPFORMAT>
```


XML Format File Generation

- New option '-x' with BCP to generate XML based format
- Examples:
 - Native data
 - `bcpx <table> format nul -x -f bcpxnative.xml -n -T`
 - Character data
 - `bcpx <table> format nul -c -x -f bcpxchar.xml -t, -T`

Data Formats, and Collations

- Bulk copy can use different data formats
 - Native - fastest
 - Unicode Native
 - Character
 - Unicode Character - slowest
- Bulk copy or Bulk insert can be used to change collation
 - Read a file with specific codepage
 - Specify codepage in table definition
 - Specify in format file

Best Practices (1)

- **BCP**
 - Offloads processing from server to client
 - Access privileges to the data file is at client.
 - Multiple batches
- **Bulk Insert**
 - High performance
 - Direct conversion to OLE DB rowset
 - SQL Server or user account needs read access to the data file
 - Multiple batches

Best Practices (2)

- **Bulk loading using OPENROWSET**
 - Data validation with no load
 - Simple (inline) data transformations
 - Full power of the SELECT made available to data in the data file (including joins)
 - Single batch
- **SSIS Bulk Load Task**
 - Fastest
 - File Must Reside On SQL Server Machine

Review

- INSERT...SELECT and INSERT...EXECUTE
- UPDATE by SELECTing from another table
- DELETE by SELECTing from another table
- MERGE statement
- TOP clause and action statements
- OUTPUT from action statements
- Inserting in bulk

Questions?