SQL server supports many installation scenarios to support "built in" data movements and "Always On" user support.

About "Bucket's" or "Containers" List: (Security & Admin "Boundaries" & "Sharing")

Source: Can connect to just about any type….	Read Only Replica database
Target: Can connect to just about any type….	Partitioned tables
Replication: Snapshot, Transactional, Merge	Not Shareable/Useable
Mirrored database	On/Off Premise (Azure SQL & NoSQL containers)
Primary database	SQL FCI (Failover Cluster Instance)
Replica database	Files or Folders

"Mirrored & Clustered" Bucket Topology: ("Out of the Box" solutions)

1. Mirrors
2. AO/HA Group w/Read Only Replicas
3. SQL FCI's (Failover Cluster Instances) within a WFCS (Windows Failover Cluster Service)
4. Azure hosting solutions (SQL, NoSQL,

Data Movement Scenarios

Publisher	Bulk (Bulk recovery model with minimal transaction logs)
Distributor	Log Shipping
Subscriber	AO/HA Replicas (Synchronous vs. Asynchronous)
Replication: Snapshot, Transactional, Merge	Partition SWITCH
SELECT INTO	Visual Studio ETL/SSIS

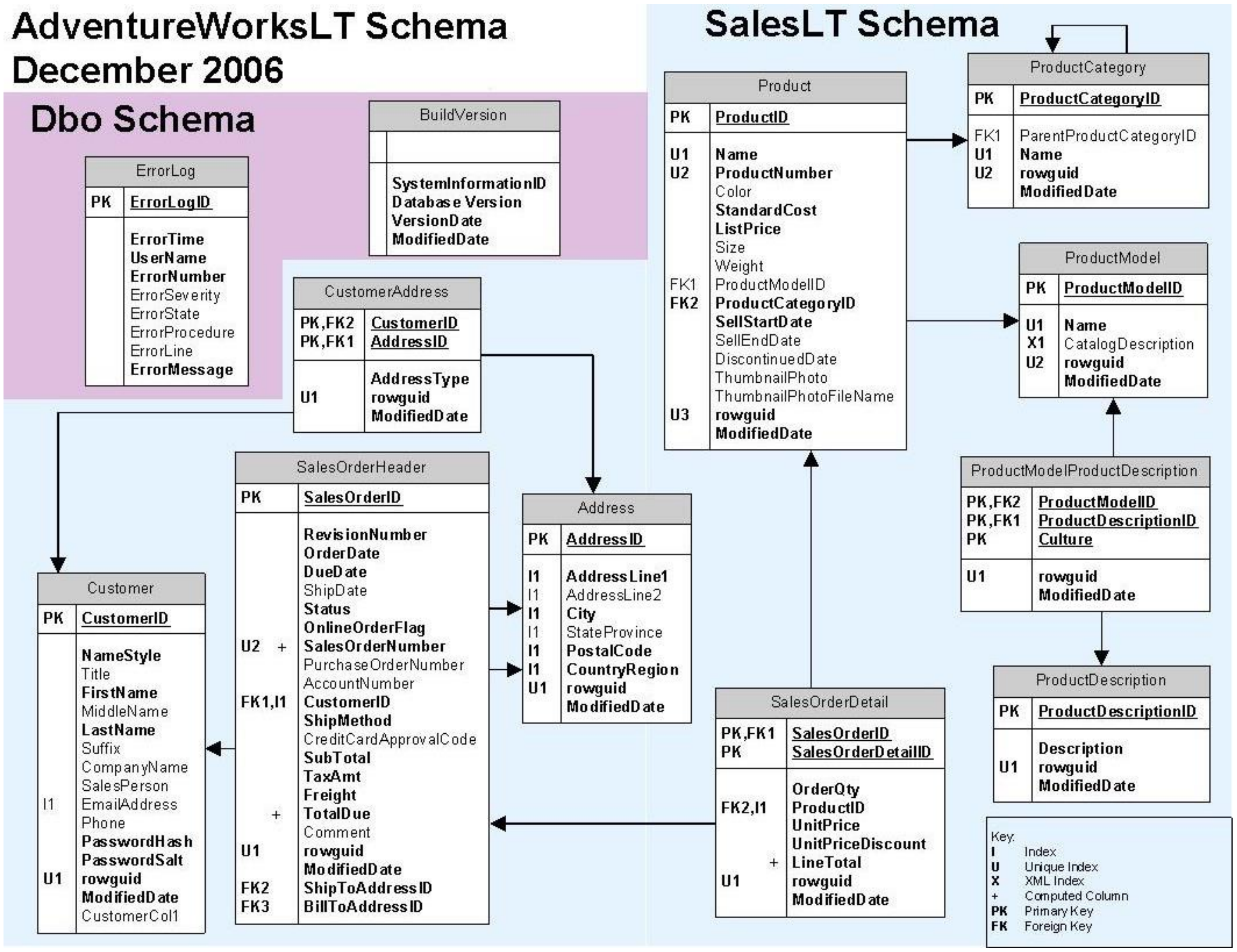
Data 2 Data Movements

1. T-SQL: SELECT INTO: Tutorial: : https://technet.microsoft.com/en-us/library/ms190750(v=sql.105).aspx
1. Import/Export: Tutorial: https://msdn.microsoft.com/en-us/library/ms140052.aspx)
2. SSIS ETL (Extract Transform Load) Jobs: Tutorial: http://msdn.microsoft.com/en-us/library/ms169917.aspx
2. BULK movements: Tutorial: https://technet.microsoft.com/en-us/library/ms191516(v=sql.110).aspx
3. XML (Create, Consume, Validate, Shred): Tutorial: http://msdn.microsoft.com/en-us/library/ms169917.aspx
4. Log Shipping: SBS p421 – install 2nd SQL named instance Win 10 VM.
5. Replication: SBS p 238 – install 3rd SQL named instance Win 10 VM.

Tool Box

1. SQLCMD: Sybex p161
2. PowerShell (Invoke-SQLCMD) Sybex p161
3. Shared Folders
4. SSMS
5. Import/Export Tool
6. BCP (Bulk Copy Program)
7. SSDT (Visual Studio), Integration Services & the "Catalog"

What are the "Relational Integrity" point of interest (PK & FK RELATIONS?)



Create files for labs….

1. Use SELECT INTO to create new table to export.
2. Export the table or tables to a .csv or .txt file to a folder for re-use.
3. Create and XML column as xml data type.
    a. Create XML from table.
    b. Store XML in column.
    c. Export XML…

T-SQL: SELECT INTO: Tutorial: : https://technet.microsoft.com/en-us/library/ms190750(v=sql.105).aspx

The SELECT INTO statement creates a new table and populates it with the result set of the SELECT statement. SELECT INTO can be used to combine data from several tables or views into one table.

The following example creates the table dbo.EmployeeAddresses by selecting seven columns from various employee and address-related tables.

USE AdventureWorks2008R2;

GO

SELECT c.FirstName, c.LastName, e.JobTitle, a.AddressLine1, a.City,

   sp.Name AS [State/Province], a.PostalCode

INTO dbo.EmployeeAddresses

FROM Person.Person AS c

   JOIN HumanResources.Employee AS e

   ON e.BusinessEntityID = c.BusinessEntityID

   JOIN Person.BusinessEntityAddress AS bea

   ON e.BusinessEntityID = bea.BusinessEntityID

   JOIN Person.Address AS a

   ON bea.AddressID = a.AddressID

   JOIN Person.StateProvince as sp

   ON sp.StateProvinceID = a.StateProvinceID;

GO

Import/Export Tool

See different ways to open utility from the web: https://msdn.microsoft.com/en-us/library/ms140052.aspx

BCP Examples: (bcp Utility: https://msdn.microsoft.com/en-us/library/ms162802.aspx)

The **b**ulk **c**opy **p**rogram utility (**bcp**) bulk copies data between an instance of MicrosoftSQL Server and a data file in a user-specified format. The **bcp** utility can be used to import large numbers of new rows into SQL Server tables or to export data out of tables into data files.

The following bcp out command creates a data file named Currency Types.dat:

```
bcp AdventureWorks2012.Sales.Currency out "Currency Types.dat" -T -c
```

Partition SWITCH Example

Example 1

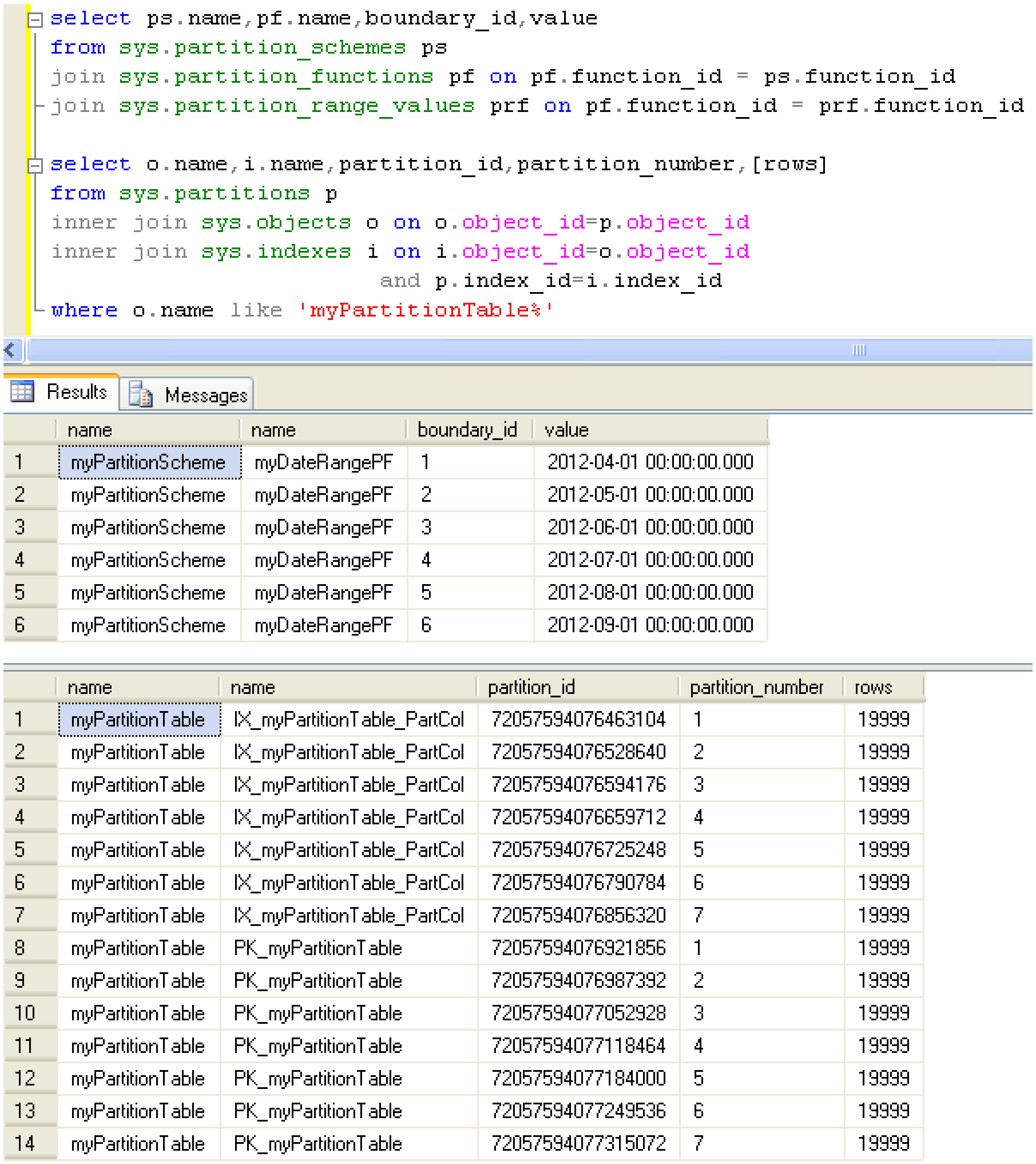
How partitioning can be used in the archiving process to provide users with uninterrupted access to the other data in the table while at the same time making the archiving process as fast as possible.

Use the code that follows to setup a partitioned table and load some data into it.

```
-- Create partition function and scheme
CREATE PARTITION FUNCTION myDateRangePF (datetime)
```

```
AS RANGE LEFT FOR VALUES ('20120401', '20120501','20120601',
                          '20120701', '20120801','20120901')
GO
CREATE PARTITION SCHEME myPartitionScheme AS PARTITION myDateRangePF ALL TO ([PRIMARY])
GO
-- Create table and indexes
CREATE TABLE myPartitionTable (i INT IDENTITY (1,1),
                               s CHAR(10) ,
                               PartCol datetime NOT NULL)
    ON myPartitionScheme (PartCol)
GO
ALTER TABLE dbo.myPartitionTable ADD CONSTRAINT
    PK_myPartitionTable PRIMARY KEY NONCLUSTERED (i,PartCol)
  ON myPartitionScheme (PartCol)
GO
CREATE CLUSTERED INDEX IX_myPartitionTable_PartCol
  ON myPartitionTable (PartCol)
  ON myPartitionScheme(PartCol)
GO
-- Polulate table data
DECLARE @x INT, @y INT
SELECT @y=3
WHILE @y < 10
BEGIN
 SELECT @x=1
 WHILE @x < 20000
 BEGIN
    INSERT INTO myPartitionTable (s,PartCol)
              VALUES ('data ' + CAST(@x AS VARCHAR),'20120' + CAST (@y AS VARCHAR)+
'15')
    SELECT @x=@x+1
 END
 SELECT @y=@y+1
END
GO
```

Now that we have a partitioned table with some data in it, let's take a look at the underlying structure of this table.



```sql
select ps.name,pf.name,boundary_id,value
from sys.partition_schemes ps
join sys.partition_functions pf on pf.function_id = ps.function_id
join sys.partition_range_values prf on pf.function_id = prf.function_id

select o.name,i.name,partition_id,partition_number,[rows]
from sys.partitions p
inner join sys.objects o on o.object_id=p.object_id
inner join sys.indexes i on i.object_id=o.object_id
                              and p.index_id=i.index_id
where o.name like 'myPartitionTable%'
```

Results Messages

	name	name	boundary_id	value
1	myPartitionScheme	myDateRangePF	1	2012-04-01 00:00:00.000
2	myPartitionScheme	myDateRangePF	2	2012-05-01 00:00:00.000
3	myPartitionScheme	myDateRangePF	3	2012-06-01 00:00:00.000
4	myPartitionScheme	myDateRangePF	4	2012-07-01 00:00:00.000
5	myPartitionScheme	myDateRangePF	5	2012-08-01 00:00:00.000
6	myPartitionScheme	myDateRangePF	6	2012-09-01 00:00:00.000

	name	name	partition_id	partition_number	rows
1	myPartitionTable	IX_myPartitionTable_PartCol	72057594076463104	1	19999
2	myPartitionTable	IX_myPartitionTable_PartCol	72057594076528640	2	19999
3	myPartitionTable	IX_myPartitionTable_PartCol	72057594076594176	3	19999
4	myPartitionTable	IX_myPartitionTable_PartCol	72057594076659712	4	19999
5	myPartitionTable	IX_myPartitionTable_PartCol	72057594076725248	5	19999
6	myPartitionTable	IX_myPartitionTable_PartCol	72057594076790784	6	19999
7	myPartitionTable	IX_myPartitionTable_PartCol	72057594076856320	7	19999
8	myPartitionTable	PK_myPartitionTable	72057594076921856	1	19999
9	myPartitionTable	PK_myPartitionTable	72057594076987392	2	19999
10	myPartitionTable	PK_myPartitionTable	72057594077052928	3	19999
11	myPartitionTable	PK_myPartitionTable	72057594077118464	4	19999
12	myPartitionTable	PK_myPartitionTable	72057594077184000	5	19999
13	myPartitionTable	PK_myPartitionTable	72057594077249536	6	19999
14	myPartitionTable	PK_myPartitionTable	72057594077315072	7	19999

As you can see from the query results we have 7 partitions in this table and we would like to remove the oldest partition. To accomplish this we will use the SWITCH PARTITION clause of the ALTER TABLE statement. A good description of this statement can be found here.

When using the SWITCH PARTITION clause there are some requirements that must be adhered to but the main point is that the source and target table schemas must match (with the exception of both needing to be partitioned).

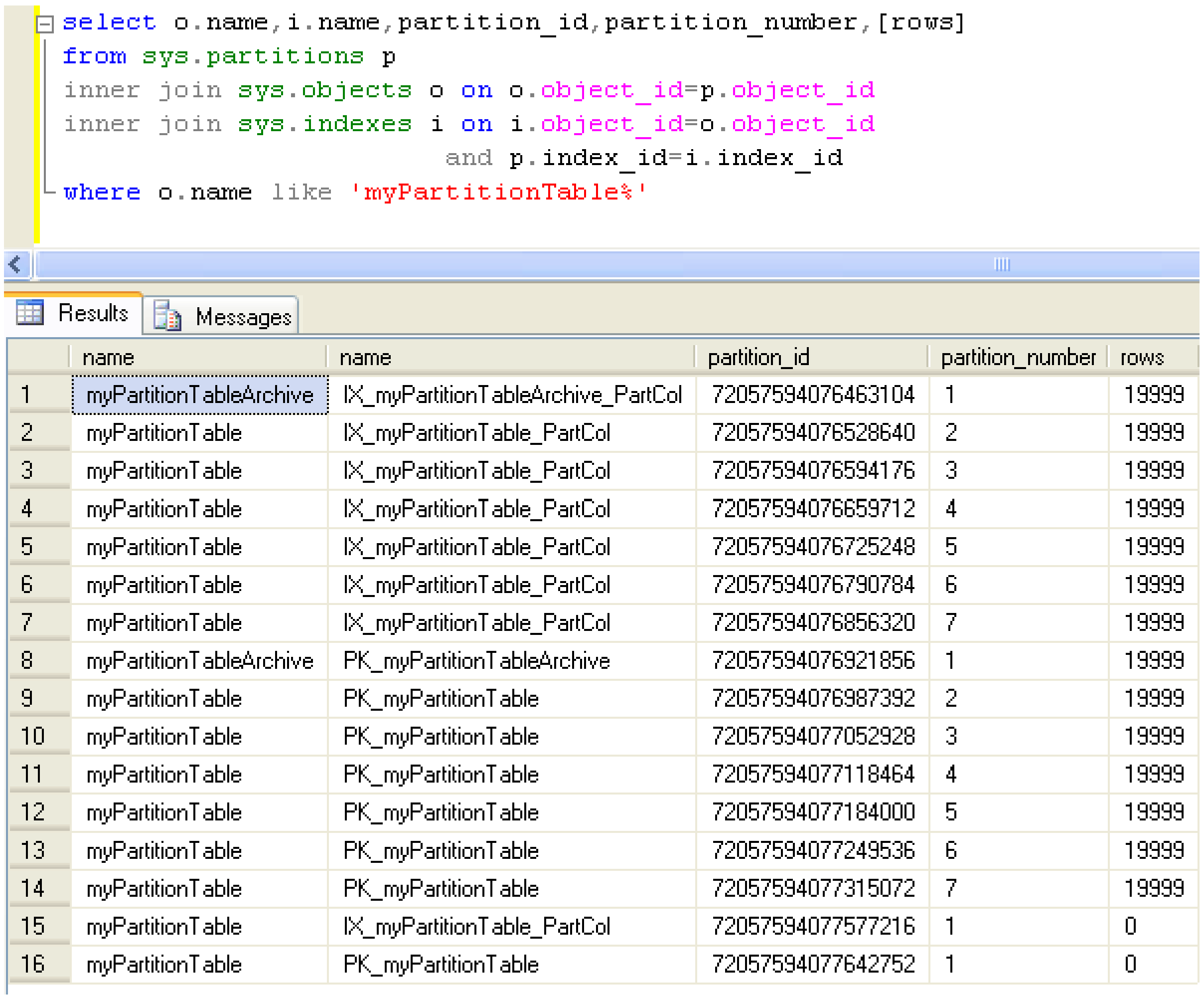
Here is the code we can use to create our archive table.

```
CREATE TABLE myPartitionTableArchive (i INT NOT NULL,
                                      s CHAR(10) ,
                                      PartCol datetime NOT NULL)
GO
ALTER TABLE myPartitionTableArchive ADD CONSTRAINT
    PK_myPartitionTableArchive PRIMARY KEY NONCLUSTERED (i,PartCol)
GO
CREATE CLUSTERED INDEX IX_myPartitionTableArchive_PartCol
  ON myPartitionTableArchive (PartCol)
GO
```

Now that we have an empty archive table we can switch partition 1 from our partitioned table with the main partition of this table. Here is the code to do this.

```
ALTER TABLE myPartitionTable SWITCH PARTITION 1 TO myPartitionTableArchive
GO
```

After running the statement above if we take a look at the sys.partitions catalog view we can see that the first partition in our partitioned table is now empty and the archive table holds these records.



```
select o.name,i.name,partition_id,partition_number,[rows]
from sys.partitions p
inner join sys.objects o on o.object_id=p.object_id
inner join sys.indexes i on i.object_id=o.object_id
                        and p.index_id=i.index_id
where o.name like 'myPartitionTable%'
```

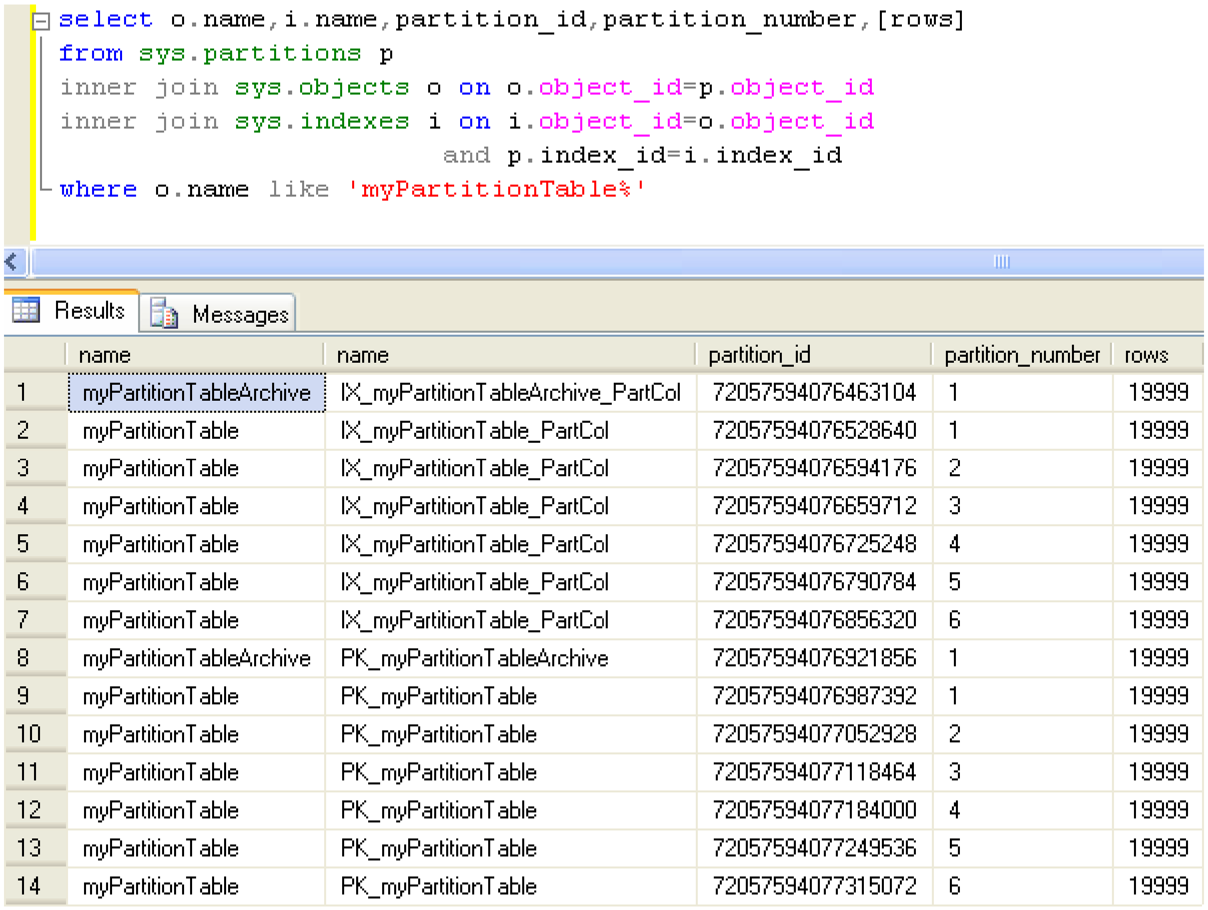
Results | Messages

	name	name	partition_id	partition_number	rows
1	myPartitionTableArchive	IX_myPartitionTableArchive_PartCol	72057594076463104	1	19999
2	myPartitionTable	IX_myPartitionTable_PartCol	72057594076528640	2	19999
3	myPartitionTable	IX_myPartitionTable_PartCol	72057594076594176	3	19999
4	myPartitionTable	IX_myPartitionTable_PartCol	72057594076659712	4	19999
5	myPartitionTable	IX_myPartitionTable_PartCol	72057594076725248	5	19999
6	myPartitionTable	IX_myPartitionTable_PartCol	72057594076790784	6	19999
7	myPartitionTable	IX_myPartitionTable_PartCol	72057594076856320	7	19999
8	myPartitionTableArchive	PK_myPartitionTableArchive	72057594076921856	1	19999
9	myPartitionTable	PK_myPartitionTable	72057594076987392	2	19999
10	myPartitionTable	PK_myPartitionTable	72057594077052928	3	19999
11	myPartitionTable	PK_myPartitionTable	72057594077118464	4	19999
12	myPartitionTable	PK_myPartitionTable	72057594077184000	5	19999
13	myPartitionTable	PK_myPartitionTable	72057594077249536	6	19999
14	myPartitionTable	PK_myPartitionTable	72057594077315072	7	19999
15	myPartitionTable	IX_myPartitionTable_PartCol	72057594077577216	1	0
16	myPartitionTable	PK_myPartitionTable	72057594077642752	1	0

We can now merge this first empty partition with the second partition using the following command.

```
ALTER PARTITION FUNCTION myDateRangePF () MERGE RANGE ('20120401')
GO
```

Taking a look at the sys.partitions catalog view after this statement is executed we can see that we no longer have the empty partition.



```
select o.name,i.name,partition_id,partition_number,[rows]
from sys.partitions p
inner join sys.objects o on o.object_id=p.object_id
inner join sys.indexes i on i.object_id=o.object_id
                           and p.index_id=i.index_id
where o.name like 'myPartitionTable%'
```

Results | Messages

	name	name	partition_id	partition_number	rows
1	myPartitionTableArchive	IX_myPartitionTableArchive_PartCol	72057594076463104	1	19999
2	myPartitionTable	IX_myPartitionTable_PartCol	72057594076528640	1	19999
3	myPartitionTable	IX_myPartitionTable_PartCol	72057594076594176	2	19999
4	myPartitionTable	IX_myPartitionTable_PartCol	72057594076659712	3	19999
5	myPartitionTable	IX_myPartitionTable_PartCol	72057594076725248	4	19999
6	myPartitionTable	IX_myPartitionTable_PartCol	72057594076790784	5	19999
7	myPartitionTable	IX_myPartitionTable_PartCol	72057594076856320	6	19999
8	myPartitionTableArchive	PK_myPartitionTableArchive	72057594076921856	1	19999
9	myPartitionTable	PK_myPartitionTable	72057594076987392	1	19999
10	myPartitionTable	PK_myPartitionTable	72057594077052928	2	19999
11	myPartitionTable	PK_myPartitionTable	72057594077118464	3	19999
12	myPartitionTable	PK_myPartitionTable	72057594077184000	4	19999
13	myPartitionTable	PK_myPartitionTable	72057594077249536	5	19999
14	myPartitionTable	PK_myPartitionTable	72057594077315072	6	19999

Now that we have the data we would like to archive in a separate table we can use bcp to export the data to a text file. There is a great tip here that describes how you can do this using TSQL. For our example you can run the following command and a sample output can be found here. Note: you have to replace the text between the # symbols. Once you have exported the data you can back it up to tape or leave it on your server for fast access, whichever works best for your situation.

```
EXEC xp_cmdshell 'bcp "select * from myPartitionTableArchive"
queryout "C:\myPartitionTableArchive_#DATEHERE#.txt" -T -S#SERVERNAME# -c -t,'
GO
DROP TABLE myPartitionTableArchive
GO
```

One final thing that also has to be done when working with partitioned tables is adding new partitions. Now that we have archived the oldest month of data let's assume we are moving into October and we would like to create a new partition for any new data that is added to the table this month. We can accomplish this by altering the partition function and splitting the last partition. Please note that this could also be done even if there was October data already loaded into the table. The TSQL to do this is as follows.

```
-- Split last partition by altering partition function
-- Note: When splitting a partition you need to use the following command before
issuing the
         ALTER PARTITION command however this is not needed for the first split command
issued.
--    ALTER PARTITION SCHEME myPartitionScheme NEXT USED [PRIMARY]
ALTER PARTITION FUNCTION myDateRangePF () SPLIT RANGE ('20121001')
GO
```

Example 2 (Partitioning/SWITCH)

SQL Server 2005 supports CREATE TABLE statement for constructing new tables. Each table has one default partition which contains all data. However, you can create up to 1000 partitions per table. To add partitions, you need to define a partition function and the corresponding partition scheme. A partition scheme uses the partition function to map data ranges to appropriate file groups. If the partition function defines three ranges, then partition scheme must map them to four file groups. Three ranges defined by the partition function and the forth, catchall range, for all data outside the boundaries of the predefined ranges.

Consider an example of adding table partitions using AdventureWorksDW sample database which ships with SQL Server 2005. Fact Internet Sales table only contains about 60000 records, but for the sake of example, populate it with 60 million sample records to resemble a real life data warehousing scenario. The table records span calendar years 2001 through 2004. You will create partitions for each year. Depending on your data volume, you could, create more granular partitions, perhaps at a month or week level, as appropriate.

First step in creating table partitions is adding file groups to your database. You could place files in multiple file groups on the same disk or on different disks or disk arrays. If you plan on using multiple aligned tables and data retrieval statements will almost always affect a small subset of rows in each table that could be grouped in partitions you should consider using multiple disk arrays. However, if you anticipate many SELECT statements retrieving a large majority of rows from each table, then performing input / output operations against multiple disk arrays could actually worsen performance. In such a case, you can still benefit from partitioning but should try to create multiple filegroups on the same disk array.

The following statements create filegroups for each year of internet sales in Adventure Works DW database:

```
ALTER DATABASE AdventureWorksDW ADD FILEGROUP [Filegroup_2001]
GO
ALTER DATABASE AdventureWorksDW ADD FILEGROUP [Filegroup_2002]
GO
ALTER DATABASE AdventureWorksDW ADD FILEGROUP [Filegroup_2003]
GO
ALTER DATABASE AdventureWorksDW ADD FILEGROUP [Filegroup_2004]
GO
```

Each filegroup can have one or multiple files associated with it. Next, add one file to each filegroup so that you can store partition data in each filegroup:

```
ALTER DATABASE AdventureWorksDW
  ADD FILE
  (NAME = N'data_2001',
  FILENAME = N'C:\mssql2005\data\data_2001.ndf',
  SIZE = 5000MB,
  MAXSIZE = 10000MB,
  FILEGROWTH = 500MB)
  TO FILEGROUP [Filegroup_2001]
GO
ALTER DATABASE AdventureWorksDW
```

```
  ADD FILE
  (NAME = N'data_2002',
  FILENAME = N'D:\mssql2005\data\data_2002.ndf',
  SIZE = 5000MB,
  MAXSIZE = 10000MB,
  FILEGROWTH = 500MB)
  TO FILEGROUP [Filegroup_2002]
GO
ALTER DATABASE AdventureWorksDW
  ADD FILE
  (NAME = N'data_2003',
  FILENAME = N'E:\mssql2005\data\data_2003.ndf',
  SIZE = 5000MB,
  MAXSIZE = 10000MB,
  FILEGROWTH = 500MB)
  TO FILEGROUP [Filegroup_2003]
GO
ALTER DATABASE AdventureWorksDW
  ADD FILE
  (NAME = N'data_2004',
  FILENAME = N'F:\mssql2005\data\data_2004.ndf',
  SIZE = 5000MB,
  MAXSIZE = 10000MB,
  FILEGROWTH = 500MB)
  TO FILEGROUP [Filegroup_2004]
GO
```

The CREATE TABLE statement normally specifies a particular filegroup on which the table is built. However, with SQL Server 2005 and later, you can reference the partition scheme as opposed to a filegroup, because you can spread each table across multiple filegroups. Partition functions commonly reference a column with DATETIME data type. This makes sense if you want to spread the data based on its creation timeframe. Data warehouse fact tables typically don't contain the DATETIME column. Instead, they normally include a foreign key referencing the date and time value in the time dimension (or, more accurately, the date dimension). Important point to keep in mind is that you're not limited to columns with DATETIME data type for partitioning keys. You could use the INTEGER data type key referencing the date dimension. However, if you use the integer values for partitioning and you want to partition based on date, then your application must ensure that records for a given week, month, quarter or year (depending on your implementation) must fall into certain ranges of integers.

For this example, alter FactInternetSales table slightly to include the partitioning column with DATETIME data type, as follows:

```
ALTER TABLE FactInternetSales
  ADD FullDate DATETIME  -- populate FullDate column with values based on DimTime
table:
UPDATE a
  SET FullDate = FullDateAlternateKey
  FROM  FactInternetSales a INNER JOIN DimTime b  ON a.OrderDateKey = b.TimeKey
```

Next create a partition function as follows:

```
CREATE PARTITION FUNCTION FullOrderDateKeyRangePFN(DATETIME)  AS
  RANGE LEFT FOR VALUES
  (  '20011231 23:59:59.997',
  '20021231 23:59:59.997',
  '20031231 23:59:59.997',
  '20041231 23:59:59.997'  )
```

Then you will need to specify the partition scheme which takes advantage of the partition function and maps each data range to different filegroups:

```
CREATE PARTITION SCHEME FullOrderDateRangePScheme  AS
  PARTITION FullOrderDateKeyRangePFN  TO
  ([Filegroup_2001],
  [Filegroup_2002],
  [Filegroup_2003],
  [Filegroup_2004],
  [PRIMARY]  )
```

Using this partition scheme, all records with FullDate value prior to December 31, 2001 will be placed on [Filegroup_2001] file group; values between January 1st, 2002 and December 31st, 2002 will be place on [Filegroup_2002] and so forth. Any records for which FullDate column has a value after December 31st, 2004 will be placed on the PRIMARY file group.

Note that the "catchall" filegroup has to be specified. In this case, the PRIMARY file group for records that do not fall into any explicitly defined date ranges. Normally as a best practice, you should reserve the PRIMARY filegroup for system objects and have a separate catchall filegroup for user data.

Now that there is a partition function and scheme, you can create a partitioned table. The syntax is very similar to any other CREATE TABLE statement except it references the partition scheme instead of a referencing filegroup:

```
CREATE TABLE [DBO].[FACTINTERNETSALES_PARTITIONED] (
  [PRODUCTKEY]             [INT]   NOT NULL,
  [ORDERDATEKEY]           [INT]   NOT NULL,
  [DUEDATEKEY]             [INT]   NOT NULL,
  [SHIPDATEKEY]            [INT]   NOT NULL,
  [CUSTOMERKEY]            [INT]   NOT NULL,
  [PROMOTIONKEY]           [INT]   NOT NULL,
  [CURRENCYKEY]            [INT]   NOT NULL,
  [SALESTERRITORYKEY]      [INT]   NOT NULL,
  [SALESORDERNUMBER]       [NVARCHAR](20)   NOT NULL,
  [SALESORDERLINENUMBER]   [TINYINT]   NOT NULL,
  [REVISIONNUMBER]         [TINYINT]   NULL,
  [ORDERQUANTITY]          [SMALLINT]   NULL,
  [UNITPRICE]              [MONEY]   NULL,
  [EXTENDEDAMOUNT]         [MONEY]   NULL,
  [UNITPRICEDISCOUNTPCT]   [FLOAT]   NULL,
  [DISCOUNTAMOUNT]         [FLOAT]   NULL,
  [PRODUCTSTANDARDCOST]    [MONEY]   NULL,
  [TOTALPRODUCTCOST]       [MONEY]   NULL,
  [SALESAMOUNT]            [MONEY]   NULL,
  [TAXAMT]                 [MONEY]   NULL,
  [FREIGHT]                [MONEY]   NULL,
  [CARRIERTRACKINGNUMBER] [NVARCHAR](25)   NULL,
  [CUSTOMERPONUMBER]       [NVARCHAR](25)   NULL,
  [FULLDATE]               [DATETIME]   NULL)
  ON FullOrderDateRangePScheme  (FullDate)
```

Now that the table has been created on a partition scheme, populate it based on the existing table FactInternetSales and subsequently examine the row count in each partition:

```
/* normally you should avoid using "SELECT *" construct.
   But since this is an example of populating a sample table
   "SELECT *" won't cause any performance or usability issues.
   In production environments you will typically use BULK INSERT
   statement to populate partitioned tables based on flat files.
*/
INSERT FACTINTERNETSALES_PARTITIONED
SELECT *
FROM   FACTINTERNETSALES

/* next we can use $PARTITION function to retrieve row counts
  for each partition:
*/
SELECT $PARTITION.FullOrderDateKeyRangePFN(FULLDATE)  AS PARTITIONID,
       COUNT(* )                                   AS ROW_COUNT
FROM     DBO.FACTINTERNETSALES_PARTITIONED
GROUP BY $PARTITION.FullOrderDateKeyRangePFN(FULLDATE)
ORDER BY PARTITIONID
```

Results:

PartitionID	Row_count
1	1013000
2	2677000
3	24443000
4	32265000

Notice that partition identifiers start at 1. The "catchall" partition located on PRIMARY file group will have partition id equal to 5. This partition isn't retrieved by the above statement because it is empty - it doesn't have any records. You could retrieve all records for a particular partition identifier using the following syntax, again using $PARTITION function:

```
SELECT * FROM dbo.FactInternetSales_Partitioned
  WHERE $Partition.FullOrderDateRangePFN (FullDate) = 2
```

Now if you create an index on this table; by default the index will be partitioned using the same partition scheme as the table:

```
CREATE INDEX ix_FactInternetSales_Partitioned_cl
  ON FactInternetSales_Partitioned (   ProductKey,   FullDate)
  ON FullOrderDateRangePScheme  (FullDate)
```

It is possible to omit the partition scheme specification in this statement if you want the index to be aligned with the table. Note that an index doesn't have to use the same partition scheme and partition function to be aligned with the table. As long as an index is partitioned based on the same data type, has the same number of partitions as the table and each partition has the same data boundaries as the table, the index will be aligned.

Partitioning key of an index doesn't have to be part of the index key. So you could use the same partition scheme to create an index that does not reference FullDate as its index key. For example, the following statement is valid:

```
CREATE INDEX ix_FactInternetSales_Partitioned_ProductKey
  ON FactInternetSales_Partitioned (   ProductKey)
  ON FullOrderDateRangePScheme  (FullDate)
```

Use a partition strategy for an index which is completely different from the underlying table's partition scheme and partition function. Some examples of where this makes sense are:

- The table isn't partitioned, but you wish to partition the index.
- The table is partitioned but you would like to collocate the index data with other tables' indexes because these tables will be frequently joined.
- You have a unique index you wish to partition and the index key isn't part of the table's partition function.

Table and index partition metadata can be retrieved for FactInternetSales_partitioned table using the following statement:

```
SELECT OBJECT_NAME([object_id]) AS table_name, *
  FROM sys.partitions
  WHERE [object_id] = OBJECT_ID('dbo.FactInternetSales_partitioned')
  ORDER BY index_id, partition_number
```

Results:

```
table_name partition_id object_id index_id partition_number hobt_id rows
FactInternetSales_Partitioned 72057594052411300 1294627655 0 1 72057594052411300
2026000
FactInternetSales_Partitioned 72057594052476900 1294627655 0 2 72057594052476900
5354000
```

```
FactInternetSales_Partitioned 72057594052542400 1294627655 0 3 72057594052542400
48886000
FactInternetSales_Partitioned 72057594052608000 1294627655 0 4 72057594052608000
64530000
FactInternetSales_Partitioned 72057594052673500 1294627655 0 5 72057594052673500 0
FactInternetSales_Partitioned 72057594052739000 1294627655 3 1 72057594052739000
2026000
FactInternetSales_Partitioned 72057594052804600 1294627655 3 2 72057594052804600
5354000
FactInternetSales_Partitioned 72057594052870100 1294627655 3 3 72057594052870100
48886000
FactInternetSales_Partitioned 72057594052935600 1294627655 3 4 72057594052935600
64530000
FactInternetSales_Partitioned 72057594053001200 1294627655 3 5 72057594053001200 0
```

It's important to note that in order to enable partition switching, all indexes on the table must be aligned. Partitioned indexes are normally implemented for query performance benefits, whereas partition switching yields great benefits in large table's manageability. So whether you align indexes with underlying tables depends on whether you are implementing table and index partitions primarily for performance tuning or for manageability.

how partition switching works, let's continue with the example presented Creating Partitioned Tables by using the same partition function FullOrderDateRangePScheme for a secondary table:

```
CREATE TABLE [dbo].[FactInternetSales_Partitioned2](
 [ProductKey] [int] NOT NULL,
 [OrderDateKey] [int] NOT NULL,
 [DueDateKey] [int] NOT NULL,
 [ShipDateKey] [int] NOT NULL,
 [CustomerKey] [int] NOT NULL,
 [PromotionKey] [int] NOT NULL,
 [CurrencyKey] [int] NOT NULL,
 [SalesTerritoryKey] [int] NOT NULL,
 [SalesOrderNumber] [nvarchar](20) NOT NULL,
 [SalesOrderLineNumber] [tinyint] NOT NULL,
 [RevisionNumber] [tinyint] NULL,
 [OrderQuantity] [smallint] NULL,
 [UnitPrice] [money] NULL,
 [ExtendedAmount] [money] NULL,
 [UnitPriceDiscountPct] [float] NULL,
 [DiscountAmount] [float] NULL,
 [ProductStandardCost] [money] NULL,
 [TotalProductCost] [money] NULL,
 [SalesAmount] [money] NULL,
 [TaxAmt] [money] NULL,
 [Freight] [money] NULL,
 [CarrierTrackingNumber] [nvarchar](25) NULL,
 [CustomerPONumber] [nvarchar](25) NULL,
 [FullDate] [datetime] NULL
) ON FullOrderDateRangePScheme  (FullDate)
```

Now suppose that FactInternetSales_Partitioned2 has data for 2004, whereas FactInternetSales_Partitioned does not. You need to swap the partitions between two tables so that the main fact table has 2004 data. As a best practice, each table should have a clustered index. Although clustered indexes aren't required for partition switching, if the source table has the clustered index then the destination table must also have an identical clustered index. Execute the following statements to add a clustered index to each table:

```
CREATE CLUSTERED INDEX ix_FactInternetSales_Partitioned_clustered ON
FactInternetSales_Partitioned (
    OrderDateKey,
    ProductKey,
    CustomerKey)
GO
CREATE CLUSTERED INDEX ix_FactInternetSales_Partitioned_clustered ON
FactInternetSales_Partitioned2 (
    OrderDateKey,
    ProductKey,
    CustomerKey)
```

You could use $PARTITION function to verify that 2004 values are stored on partition id 4. Next, execute ALTER TABLE statement to migrate partition 4 from the secondary table to FactInternetSales_Partitioned:

```
ALTER TABLE FactInternetSales_Partitioned2
SWITCH PARTITION 4 TO FactInternetSales_Partitioned PARTITION 4
```

In a matter of a couple of seconds partitions are switched. Note that in order to switch partitions, the partition in the destination partition must be empty.

Now you can verify the number of records in partition number 4 in FactInternetSales_Partitioned, as follows:

```
SELECT COUNT(*) FROM dbo.FactInternetSales_Partitioned

WHERE $PARTITION.FullOrderDateRangePFN (FullDate) = 4 ;

Result:

-----------


32265000
```

During partition switching, data isn't copied or duplicated. Only metadata is modified. Therefore, now that partitions have been switched the secondary table will no longer have any records for 2004. All indexes associated with a partition are also switched along with the table partition.

Similarly if you wanted to phase out obsolete records for 2001 into the secondary table we could use the following statement:

```
ALTER TABLE FactInternetSales_Partitioned
SWITCH PARTITION 1 TO FactInternetSales_Partitioned2 PARTITION 1
```

The full syntax for switching partitions is as follows:

```
ALTER TABLE source_table

SWITCH [PARTITION source_partition_number]


       TO destination_table

        [PARTITION target_partition_number]
```

Partition schemes used by source and destination table do not have to be the same. In fact, you could switch a partition into a non-partitioned table. For example, create a non-partitioned table with a schema identical to the sample partitioned tables and execute the following statements:

```
CREATE TABLE [dbo].[FactInternetSales_NonPartitioned](
    [ProductKey] [int] NOT NULL,
    [OrderDateKey] [int] NOT NULL,
    [DueDateKey] [int] NOT NULL,
    [ShipDateKey] [int] NOT NULL,
    [CustomerKey] [int] NOT NULL,
    [PromotionKey] [int] NOT NULL,
    [CurrencyKey] [int] NOT NULL,
    [SalesTerritoryKey] [int] NOT NULL,
    [SalesOrderNumber] [nvarchar](20) NOT NULL,
    [SalesOrderLineNumber] [tinyint] NOT NULL,
    [RevisionNumber] [tinyint] NULL,
    [OrderQuantity] [smallint] NULL,
    [UnitPrice] [money] NULL,
    [ExtendedAmount] [money] NULL,
    [UnitPriceDiscountPct] [float] NULL,
    [DiscountAmount] [float] NULL,
    [ProductStandardCost] [money] NULL,
    [TotalProductCost] [money] NULL,
    [SalesAmount] [money] NULL,
    [TaxAmt] [money] NULL,
    [Freight] [money] NULL,
    [CarrierTrackingNumber] [nvarchar](25) NULL,
    [CustomerPONumber] [nvarchar](25) NULL,
    [FullDate] [datetime] NULL
)
ON [Filegroup_2003]    /* note that non-partitioned table was created on
filegroup_2003.  now we can switch 2003 data into the non-partitioned table:  */
ALTER TABLE FactInternetSales_Partitioned
SWITCH PARTITION 3 TO FactInternetSales_NonPartitioned
```

Since the destination table is not partitioned, you don't have to specify a partition number for it. If you do specify partition number for a non-partitioned table, SQL Server completes the partition switch successfully, but returns a warning similar to this:

```
Warning: The specified partition 3 for the table
'AdventureWorksDW.dbo.FactInternetSales_NonPartitioned' was ignored in ALTER TABLE
SWITCH statement because the table is not partitioned.
```

The destination table remains un-partitioned. It does not inherit partition scheme from the source table. However, every table has at least one partition. So technically you could use partition switching to move data between two tables that haven't been explicitly partitioned. For example, the following statement moves data between two tables for which partitions haven't been explicitly defined; therefore partition numbers are not included:

```
ALTER TABLE FactInternetSales_NonPartitioned
SWITCH TO  FactInternetSales_NonPartitioned1
```

You can effectively combine merge and split operations with partition switching. As an example, if you need to add the most recent data to a large fact table you could:

1. Create a staging table on the same filegroup as the target partition, perhaps called Filegroup_2007.

2. Split the most recent partition, adding a boundary value for 2nd half of 2007.

3. Load data into the staging table using BULK INSERT statement or an Integration Services package. Index the staging table and ensure it contains correct data.

4. Switch data from staging table into the fact table into the next to last partition.

Similarly if you were planning on removing data for 1989, which happens to be the first year for which you have data in a fact table, you could:

1. Create an empty historical table on the same filegroup where the obsolete data resides, perhaps called Filegroup_1989.

2. Switch data from the fact table to the historical table.

3. Merge the first partition thereby removing the boundary point for 1989.

4. Move data from the historical table to a different database. Then drop the historical table.

Reference

3. Tutorial: sqlcmd Utility: https://technet.microsoft.com/en-us/library/ms170207(v=sql.105).aspx
4. Tutorial: Inserting Rows by Using SELECT INTO: https://technet.microsoft.com/en-us/library/ms190750(v=sql.105).aspx
5. Tutorial: SQL Server Import and Export Wizard: https://msdn.microsoft.com/en-us/library/ms140052.aspx
6. BCP Utility: https://msdn.microsoft.com/en-us/library/ms162802.aspx
7. Tutorial – Bulk Copy Program: Create a Format File (SQL Server): https://technet.microsoft.com/en-us/library/ms191516(v=sql.110).aspx
8. Exporting Data from or Importing Data to a Temporary Table(BCP): https://technet.microsoft.com/en-us/library/ms191503(v=sql.105).aspx
9. Examples of Bulk Import and Export of XML Documents (SQL Server): https://msdn.microsoft.com/en-us/library/ms191184.aspx
10. SQL Server Partitioning Tips: https://www.mssqltips.com/sql-server-tip-category/65/partitioning/
11. Archiving SQL Server data using partitioning: https://www.mssqltips.com/sqlservertip/2780/archiving-sql-server-data-using-partitioning/
12. How to Partition an existing SQL Server Table: https://www.mssqltips.com/sqlservertip/2888/how-to-partition-an-existing-sql-server-table/
13. Example of Creating Partitioned Tables: https://www.toadworld.com/platforms/sql-server/w/wiki/9651.example-of-creating-partitioned-tables
14. Example - Switching Partitions: https://www.toadworld.com/platforms/sql-server/w/wiki/9658.switching-partitions-example
15. Deploy Integration Services (SSIS) Projects and Packages: https://msdn.microsoft.com/en-us/library/hh213290.aspx
16. Convert to Package Deployment Model Dialog Box: https://msdn.microsoft.com/en-us/library/hh213212.aspx
17. Legacy Package Deployment (SSIS): https://msdn.microsoft.com/en-us/library/ms137592.aspx