Microsoft

Migrating to PDW

Agenda

- Planning your Migration
- Migrating DDL
- Migrating SQL Code
- Augmenting ETL Frameworks
- Altering SSAS designs
- Supporting multiple environments

Planning Your Migration

Understanding Migration

Significant Undertaking

- Large # of Databases
- Large # of Objects
- Large # of Reports
- End-user training
- Operations

Areas of complexity

- Code compliance
- ETL (re)design
- 3rd Party Products
- Data Consumption

Approaches to Migration

Delivery Method
- Micro increment
- Phased increment
- Whole product

Philosophies
- Lift & Shift
- Redesign & Rebuild

Horizontal Slicing
- End to End
- Domain Driven

Vertical Slicing
- Layers
- Technically Driven

Migrating for Coverage

Goals
- Breadth
- Compatibility
- Risk Mitigation

Non-goals
- Performance

Tends to involve
- Minimal Change
- Lift & Shift
- Technology Focus

Test Strategy
- Parallel Run
- Black Box Tests

Migrating for Performance

Goals
- Performance
- Maintain-ability
- Depth

Non-goals
- Risk Management
- Solution Breadth

Tends to Involve
- Greater change
- Re-design & rebuild
- Horizontal Approach

Test Strategy
- White-Box Testing
- Non-Functional Tests

Risk Management

PDW Migration Advisor

- Internal MS Tool
- Evaluates Code Use
- Summarises Findings
  - Code Impact
  - Object Impact
  - Rule Impact

3rd Party Tools

- BI Tools
- Data Integration
- Appliance Monitoring
- Backup

PDW Migration Advisor Limitations

- SQL Server Sources
- Identifies Issues
- Does not fix

DevOps Tasks

- Source Code Management with TFS
- Release Management
- Automation
- Environment
  - Hadoop Integration
  - Cloud Environment and configuration
  - Loading and Backup servers
  - Dev & Test

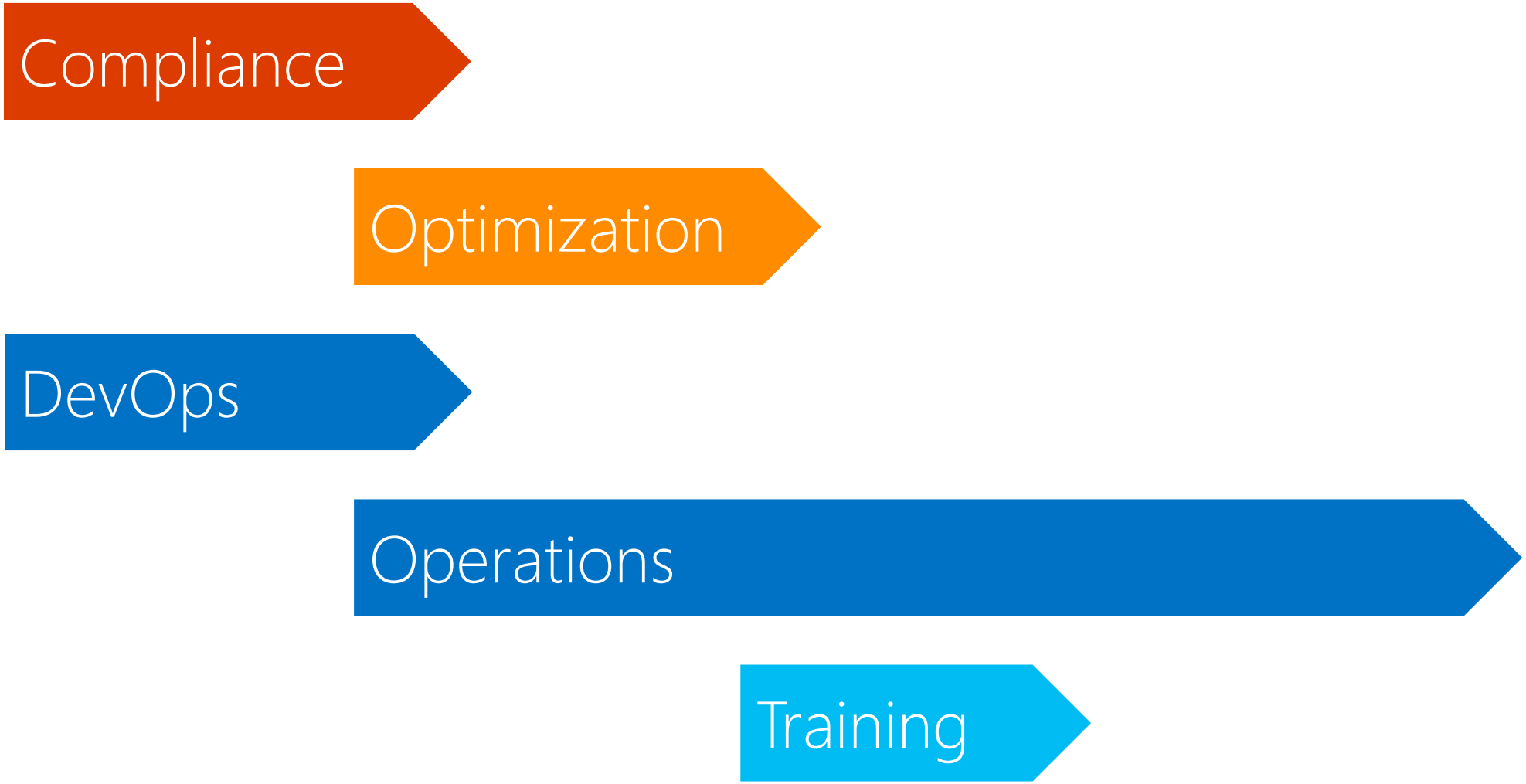
Operations

- Appliance Configuration
- Appliance Monitoring
- Backup / Restore
- Permissions Model
- HA & DR planning & design

Recommendations

- Build confidence
- Demonstrate value
- Be pragmatic
- Release frequently
- Identify risk areas early
- Include operations
- Executive sponsorship

Crawl, Walk, Run, Sprint



Compliance

- PDW supports subset of T-SQL functionality
- Minimise change
- Maximise coverage

Optimisation

- PDW supports enhanced T-SQL capabilities
- Non-Functional Tuning
- Maximise Parallelism opportunities

Migrating DDL

DDL not implemented by PDW

- Default Constraints*
- Check Constraints
- User Defined Types
- Data Types
- Computed Columns
- Primary Keys
- Foreign Keys
- Indexed Views
- Identity / Sequences

Partial support for default constraints
(strings and constants only on Create / Alter table)

Nulls & Column Lengths

DMS Row Conversion

- Columns are converted to ODBC native types
- Columns are padded to their maximum length
- Columns are re-sequenced (fixed columns first)
- Nullable Columns are assigned a null identifier & null terminator
- Distribution key column is hashed
- Row must fit into a 32KB buffer

Impact on DMS Throughput

- Every Nullable field adds eight bytes to the row length
- Wide Column Definitions can have a significant impact on buffer row density

Collation Support

Collation is fixed at the following levels

- Appliance
- Database

Specify Collation at the Column Level only

Appliance and Database Collation fixed to LATIN1_GENERAL_100_CI_AS_KS_WS

Migrating DDL

PDWScripter Utility

- Generates SQL Objects held in PDW
- Utility offered “as-is”

Incompatible Data Types

- Over time data types in the source database may have been inconsistently applied
- In SMP environments implicit type conversion may have masked this issue
- For PDW we need to reconcile the types
- Joining across incompatible types leads to unnecessary data movement

Finding Incompatible Types

```
WITH T
AS
(
SELECT T.NAME,C.NAME CNAME,Y.NAME YNAME,C.IS_NULLABLE
       ,C.MAX_LENGTH,C.PRECISION,C.SCALE,C.COLUMN_ID
FROM SYS.COLUMNS C
JOIN SYS.TABLES T ON C.OBJECT_ID = T.OBJECT_ID
JOIN SYS.TYPES Y ON C.USER_TYPE_ID = Y.USER_TYPE_ID
)
SELECT *
FROM T as T1
FULL JOIN T as T2 ON T1.CNAME = T2.CNAME
WHERE   T1.YNAME                <>T2.YNAME
OR      T1.IS_NULLABLE          <>T2.IS_NULLABLE
OR      T1.MAX_LENGTH           <>T2.MAX_LENGTH
OR      T1.[PRECISION]          <>T2.[PRECISION]
OR      T1.SCALE                <>T2.SCALE
;
```

This script is not exhaustive!

Double check query joins and ELT components for signs of movement due to incompatible data types

Migrating SQL Code

Migration Maxims

General Maxim for PDW

- If it worked in SQL 2000 it probably works on PDW
- If it did not work / not available in SQL 2000 then it probably does not work with PDW

Notable Exceptions

- Windowing Functions
- PDW specific functionality (CTAS, Rename)
- Clustered Columnstore Index

Stored Procedures in PDW

- Aren't compiled code
- Containers for SQL Logic
- PDW re-writes the SQL in the procedures
- MPP query plans aren't cached
- Recompile option irrelevant & unsupported
- Quoted Identifier is always ON
- ANSI Nulls is always ON

Stored Procedures

Following procedure types aren't supported

- Temporary - deprecated
- Numbered - deprecated
- Extended - deprecated
- Encrypted
- CLR – dependency on CLR support

Stored Procedures

Additional Restrictions

- TVPs – uses table variables (no stats)
- Read only parameters – used primarily for TVP
- Default parameters
- Execution contexts (Execute AS)
- Return Statement

User Schemas

You can have any schema you like…
…as long as it is dbo…

Workaround

- Prepend schema name to table names
- Use database roles for security

SQL Code – not in PDW today

- Functions (UDFs)
- Cursors
- Triggers
- Insert … Execute
- Print
- Raiserror
- Recursive CTE
- CTE with DML operations
- CLR
- Table Valued Parameters
- Updating Through Views
- Merge
- $partition
- DML using ANSI join
- Read Committed Isolation
- Try…Catch

SET Statements

- ANSI fixed to always support ANSI behaviour
- NOCOUNT is not recognised – always OFF
- DATEFIRST always 7 – Sunday first day of week
- DATEFORMAT always mdy
- TRANSACTION ISOLATION LEVEL Read Uncommitted Only

Variable Assignment

- Use SET
- Select not supported `SELECT @v =1` not allowed
- Assigning a value from a query

```
DECLARE @v int = 0;
SET     @v = (Select max(database_id) from sys.databases);
SET     @v = 1;
SET     @v = @v+1;
SET     @v +=1;
```

Only one variable can be assigned at a time

Common Table Expressions

- Partially Supported by PDW
- No recursion
- Cannot be used to update or delete data
- Can be used by CTAS, CETAS & CRTAS

Dynamic SQL Execution

- No (MAX) type
- Max String length is 8000 bytes

```
DECLARE
 @sql_fragment1 VARCHAR(8000)=' SELECT name '
,@sql_fragment2 VARCHAR(8000)=' FROM sys.system_views '
,@sql_fragment3 VARCHAR(8000)=' WHERE name like ''%loader%''';

EXEC(@sql_fragment1+@sql_fragment2+@sql_fragment3);
```

Pivot & UnPivot

- Are not supported
- Equivalent functionality is available

Pivot Rows to Columns

```
WITH SalesPVT AS
(SELECT EnglishProductCategoryName
,CASE WHEN [CalendarYear] = 2001 THEN SUM(SalesAmount) ELSE 0 END AS 'CY_2001'
,CASE WHEN [CalendarYear] = 2002 THEN SUM(SalesAmount) ELSE 0 END AS 'CY_2002'
,CASE WHEN [CalendarYear] = 2003 THEN SUM(SalesAmount) ELSE 0 END AS 'CY_2003'
,CASE WHEN [CalendarYear] = 2004 THEN SUM(SalesAmount) ELSE 0 END AS 'CY_2004'
FROM  dbo.factInternetSales s
JOIN  dbo.DimDate d                ON s.OrderDateKey            = d.DateKey
JOIN  dbo.DimProduct p             ON s.ProductKey              = p.ProductKey
JOIN  dbo.DimProductSubCategory u ON p.[ProductSubcategoryKey] = u.[ProductSubcategoryKey]
JOIN  dbo.DimProductCategory c     ON u.[ProductCategoryKey]    = c.[ProductCategoryKey]
GROUP BY [EnglishProductCategoryName]
,        [CalendarYear]
)
SELECT
EnglishProductCategoryName
,SUM(CY_2001)AS 'CY_2001'
,SUM(CY_2002)AS 'CY_2002'
,SUM(CY_2003)AS 'CY_2003'
,SUM(CY_2004)AS 'CY_2004'
FROM SalesPVT
GROUP BY EnglishProductCategoryName
```

UnPivot

```
SELECT EnglishProductCategoryName
,      CASE c.Number
WHEN 1 THEN CY_2001
WHEN 2 THEN CY_2002
WHEN 3 THEN CY_2003
WHEN 4 THEN CY_2004
END as Sales
FROM AnnualSales_pvt
JOIN #Nums c ON 1=1
WHERE  CASE c.Number
WHEN 1 THEN CY_2001
WHEN 2 THEN CY_2002
WHEN 3 THEN CY_2003
WHEN 4 THEN CY_2004
END IS NOT NULL;
```

```
CREATE TABLE #Nums
WITH(LOCATION = USER_DB
,DISTRIBUTION = REPLICATE
)
AS
WITH
  L0   AS(SELECT 1 AS c UNION ALL SELECT 1),
  L1   AS(SELECT 1 AS c FROM L0 AS A, L0 AS B),
  L2   AS(SELECT 1 AS c FROM L1 AS A, L1 AS B),
  L3   AS(SELECT 1 AS c FROM L2 AS A, L2 AS B),
  L4   AS(SELECT 1 AS c FROM L3 AS A, L3 AS B),
  L5   AS(SELECT 1 AS c FROM L4 AS A, L4 AS B),
  Nums AS(SELECT ROW_NUMBER() OVER(ORDER BY c)
AS n FROM L5)
SELECT n AS Number
FROM   Nums
WHERE  n <= @n;
```

Based on fn_nums by Itzik Ben-Gan

Surrogate Key (Identity)

- Load Data Into a Replicated Staging Table
- Cross Join assigns Max ID value to new rows

```
INSERT INTO dbo.DimProduct
SELECT
ROW_NUMBER() OVER (ORDER BY p.ProductAlternateKey) + maxID.maxID
,p.*
FROM (    SELECT *
          FROM   dbo.stg_DimProduct
     ) AS p
CROSS JOIN
     (    SELECT CAST(ISNULL((MAX(Product_ID)),0) AS INT) AS maxID
          FROM   dbo.DimProduct_ID
     ) as maxID;
```

Update

- ANSI Joins not currently supported
- Unaffected by SET ROWCOUNT
- Cannot contain TOP
- Single table Implicit Joins and Sub-Selects are supported
- Cannot use Views or CTEs

Update w/Implicit Join

```
CREATE TABLE AnnualCategorySales
([EnglishProductCategoryName] NVARCHAR(50) NOT NULL
,[CalendarYear]               SMALLINT     NOT NULL
,[TotalSalesAmount]           MONEY        NOT NULL
)
```

Update w/ Implicit Join

```
CREATE TABLE CTAS_ACS
WITH (DISTRIBUTION = REPLICATE)
AS
SELECT
 ISNULL(CAST([EnglishProductCategoryName] AS NVARCHAR(50)),0) AS [EnglishProductCategoryName]
,ISNULL(CAST([CalendarYear] AS SMALLINT),0) AS [CalendarYear]
,ISNULL(CAST(SUM([SalesAmount]) AS MONEY),0) AS [TotalSalesAmount]
FROM  dbo.factInternetSales s
JOIN  dbo.DimDate d                ON s.OrderDateKey            = d.DateKey
JOIN  dbo.DimProduct p             ON s.ProductKey              = p.ProductKey
JOIN  dbo.DimProductSubCategory u ON p.[ProductSubcategoryKey] = u.[ProductSubcategoryKey]
JOIN  dbo.DimProductCategory c     ON u.[ProductCategoryKey]    = c.[ProductCategoryKey]
WHERE [CalendarYear] = 2004
GROUP BY [EnglishProductCategoryName]
,        [CalendarYear];

UPDATE AnnualCategorySales
SET   AnnualCategorySales.TotalSalesAmount = CTAS_ACS.TotalSalesAmount
FROM  CTAS_ACS
WHERE CTAS_ACS.[EnglishProductCategoryName] = AnnualCategorySales.[EnglishProductCategoryName]
AND   CTAS_ACS.[CalendarYear]               = AnnualCategorySales.[CalendarYear];
```

Delete

- Only supports joins in sub-queries
- Also use sub-select
- Unaffected by SET ROWCOUNT
- Cannot use Views or CTEs

Delete Examples

```
DELETE
FROM   CTAS_ACS
WHERE  EXISTS
(
SELECT *
FROM   CTAS_ACS_2001
WHERE  CTAS_ACS.CalendarYear = CTAS_ACS_2001.CalendarYear
AND    CTAS_ACS.[EnglishProductCategoryName] = CTAS_ACS_2001.[EnglishProductCategoryName]
);

DELETE FROM dbo.FactInternetSales
WHERE ProductKey IN
(
SELECT    T1.ProductKey
FROM      dbo.DimProduct T1
JOIN      dbo.DimProductSubcategory T2
ON        T1.ProductSubcategoryKey = T2.ProductSubcategoryKey
WHERE     T2.EnglishProductSubcategoryName = 'Road Bikes'
);
```

Cursors

- Cursors are a curse
- Most Cursors are “Firehose” i.e. forward only

If you can’t replace the cursor with set based operations…

- Replace with a loop

Cursor Example for Backup

```
CREATE TABLE #dbs WITH (location=USER_DB,DISTRIBUTION = REPLICATE)
AS
SELECT
 ROW_NUMBER() OVER(ORDER BY database_id ) AS Sequence
,name
,'BACKUP DATABASE '+name+' TO DISK = ''\\UNC_Share\Destination_Folder''' AS sql_code
FROM    sys.databases
WHERE   name <>'tempdb'

DECLARE @nbr_statements INT = (SELECT COUNT(*) FROM #dbs)
DECLARE @i INT = 1
WHILE   @i <= @nbr_statements
BEGIN
    DECLARE @sql_code NVARCHAR(4000) = (SELECT sql_code FROM #dbs WHERE Sequence = @i)
    EXEC    sp_executesql @sql_code
    SET     @i +=1
END
```

Migrating Control Flow

- TRY..CATCH
- RAISERROR
- ERROR_* Functions aren't supported

Options

- BREAK

Nesting Level

- PDW 8 Nesting Levels
- SMP 32 Nesting Levels

Every EXEC increases nest level depth

- Includes dynamic SQL

Some SMP applications are may be

- business logic heavy
- Modular

Nest level maybe deeper than PDW supports

- Flatten Code

Renaming

- Sp_rename / sp_renamedb aren't supported
- Use PDW specific syntax instead - RENAME
- Requires exclusive access
- Works for tables & databases only
  - Drop / Create required for indexes and views
- Rename is not propagated
  - Only affects the base table
  - Views will be invalid until table is re-created
- Commonly used post CTAS operations
  - Whole table dimension updates

Rename Examples

```
RENAME DATABASE Instructor TO Instructor2;


RENAME OBJECT dbo.item TO item_out;
RENAME OBJECT dbo.item_in TO item;
```

Group by

- Partially Supported
- PDW Hint
  - WITH(DISTRIBUTED_AGG)

Unsupported

- Rollup
- Cube
- Grouping Sets

```
SELECT
 CustomerKey
,SUM(SalesAmount) AS TotalSalesAmount
FROM     dbo.FactInternetSales
GROUP BY CustomerKey WITH (DISTRIBUTED_AGG)
ORDER BY CustomerKey DESC;
```

Rollup Example

```
SELECT [SalesTerritoryCountry]
,      [SalesTerritoryRegion]
,      SUM(SalesAmount) AS TotalSalesAmount
FROM   dbo.factInternetSales s
JOIN   dbo.DimSalesTerritory t     ON s.SalesTerritoryKey       = t.SalesTerritoryKey
GROUP BY
       [SalesTerritoryCountry]
,      [SalesTerritoryRegion]
UNION ALL
SELECT [SalesTerritoryCountry]
,      NULL
,      SUM(SalesAmount) AS TotalSalesAmount
FROM   dbo.factInternetSales s
JOIN   dbo.DimSalesTerritory t     ON s.SalesTerritoryKey       = t.SalesTerritoryKey
GROUP BY
       [SalesTerritoryCountry]
UNION ALL
SELECT NULL
,      NULL
,      SUM(SalesAmount) AS TotalSalesAmount
FROM   dbo.factInternetSales s
JOIN   dbo.DimSalesTerritory t     ON s.SalesTerritoryKey       = t.SalesTerritoryKey;
```

GROUP BY CUBE

- Can get very long & complex

Combine the following ingredients

- Metadata (tables or generated)
- While loop
- Dynamic SQL
- Temporary Table

Cube Step #1

```
CREATE TABLE #Cube
WITH (DISTRIBUTION = REPLICATE,LOCATION = USER_DB)
AS
WITH GrpCube AS
(SELECT    CAST(ISNULL(Country,'NULL')+','+ISNULL(Region,'NULL') AS NVARCHAR(50)) as 'Cols'
,          CAST(ISNULL(Country+',','')+ISNULL(Region,'') AS NVARCHAR(50))  as 'GroupBy'
,          ROW_NUMBER() OVER (ORDER BY Country) as 'Seq'
FROM       ( SELECT 'SalesTerritoryCountry' as Country
             UNION ALL
             SELECT NULL
           ) c
CROSS JOIN ( SELECT 'SalesTerritoryRegion' as Region
             UNION ALL
             SELECT NULL
           ) r
)
SELECT Cols
,      CASE WHEN SUBSTRING(GroupBy,LEN(GroupBy),1) = ','
            THEN SUBSTRING(GroupBy,1,LEN(GroupBy)-1)
            ELSE GroupBy
       END AS GroupBy  --Remove Trailing Comma
,Seq
FROM GrpCube;
```

First Step :

- Generate the Cube of Columns

Cube Step #2

```
DECLARE
 @SQL NVARCHAR(4000)
,@Columns NVARCHAR(4000)
,@GroupBy NVARCHAR(4000)
,@i INT = 1
,@nbr INT = 0
;
CREATE TABLE #Results
(
 [SalesTerritoryCountry] NVARCHAR(50)
,[SalesTerritoryRegion]  NVARCHAR(50)
,[TotalSalesAmount]      MONEY
)
WITH (DISTRIBUTION = REPLICATE,LOCATION = USER_DB)
;
```

Second Step :

- Initialise variables
- Create the target

Cube Step #3

```
SET @nbr =(SELECT MAX(Seq) FROM #Cube);

WHILE @i<=@nbr
BEGIN
    SET @Columns = (SELECT Cols    FROM #Cube where seq = @i)
    SET @GroupBy = (SELECT GroupBy FROM #Cube where seq = @i)

    SET @SQL ='INSERT INTO #Results
              SELECT '+@Columns+'
              ,      SUM(SalesAmount) AS TotalSalesAmount
              FROM  dbo.factInternetSales s
              JOIN  dbo.DimSalesTerritory t
              ON s.SalesTerritoryKey = t.SalesTerritoryKey
              '+CASE WHEN @GroupBy <>''
                     THEN 'GROUP BY '+@GroupBy ELSE '' END

    EXEC sp_executesql @SQL
    SET @i +=1
END
SELECT * FROM #Results
order by 1,2,3
```

Third Step :

- Loop over Cube
- Insert into Temp
- Return Results

Augmenting ETL Frameworks

Logging & Debugging

- Print is not supported
- Attach Labels instead

Visible in

- Management Console
- sys.dm_pdw_exec_requests

Most DML operations support Label option

- Select
- Insert
- Update
- Delete
- CTAS
- CETAS

Label Option

```
CREATE TABLE DimProduct_New
WITH (DISTRIBUTION = REPLICATE)
AS
SELECT *
FROM DimProduct
OPTION (LABEL = 'CTAS : DimProduct : Step 01')
```

Labels in the Management Console

parallel data warehouse tuki2a: session sid6511

LOGIN JRJ
LOGON DATE 4/20/2014 3:38:35 AM
STATUS Idle

CLIENT ID 172.16.252.100:51598
APPLICATION NAME Microsoft SQL Server Data Tools, T-SQL Editor

QUERIES LOCKS WAITS

ID	START TIME	END TIME	DURATION	STATUS	LABEL	ERROR
QID64115	4/20/2014 3:38:43 AM	4/20/2014 3:38:44 AM	000:00:00:910	Completed	Procedure : Statement_05 : Comment	
QID64114	4/20/2014 3:38:42 AM	4/20/2014 3:38:43 AM	000:00:00:847	Completed	Procedure : Statement_04 : Comment	
QID64113	4/20/2014 3:38:41 AM	4/20/2014 3:38:42 AM	000:00:00:875	Completed	Procedure : Statement_03 : Comment	
QID64112	4/20/2014 3:38:41 AM	4/20/2014 3:38:41 AM	000:00:00:719	Completed	Procedure : Statement_02 : Comment	
QID64110	4/20/2014 3:38:40 AM	4/20/2014 3:38:44 AM	000:00:04:133	Completed		
QID64111	4/20/2014 3:38:40 AM	4/20/2014 3:38:41 AM	000:00:00:781	Completed	Procedure : Statement_01 : Comment	
QID64108	4/20/2014 3:38:40 AM	4/20/2014 3:38:40 AM	000:00:00:000	Completed		
QID64109	4/20/2014 3:38:40 AM	4/20/2014 3:38:40 AM	000:00:00:000	Completed		
QID64106	4/20/2014 3:38:40 AM	4/20/2014 3:38:40 AM	000:00:00:016	Completed		
QID64107	4/20/2014 3:38:40 AM	4/20/2014 3:38:40 AM	000:00:00:016	Completed		
QID64105	4/20/2014 3:38:35 AM	4/20/2014 3:38:35 AM	000:00:00:000	Completed		

Monitoring Labels

```
SELECT
 r.Session_ID        as Session_ID
,r.request_id        as Request_ID
,r.[status]          as Request_Status
,[label]             as Request_QueryLabel
,submit_time         as Request_SubmitTime
,start_time          as Request_StartTime
,end_compile_time    as Request_EndCompileTime
,end_time            as Request_EndTime
,total_elapsed_time as Request_TotalElapsedDuration_ms
,DATEDIFF(ms,submit_time,start_time)      as Request_InitiateDuration_ms
,DATEDIFF(ms,start_time,end_compile_time) as Request_CompileDuration_ms
,DATEDIFF(ms,end_compile_time,end_time)   as Request_ExecDuration_ms
,command             as Request_Command
,database_id         as Request_Database_ID
,SYSDATETIME()as Log_Date
FROM sys.dm_pdw_exec_requests r
WHERE r.[Label] = 'CTAS : DimProduct : Step 01'
```

DMV Limits on History

- sys.dm_pdw_exec_sessions
  - 10,000 most recent sessions
- sys.dm_pdw_exec_requests
  - 10,000 most recent requests
- sys.dm_pdw_errors
  - 10,000 most recent errors
- sys.dm_pdw_sql_requests
  - 1,000 most recent SQL requests
- sys.dm_pdw_request_steps
  - All steps for 1,000 most recent SQL requests
- sys.dm_pdw_dms_workers
  - All workers for 1,000 most recent SQL requests

Capturing History

- DMV data is transient; thresholds easily exceeded by load
- Need to retain the history ourselves
- Labels are a good start
- More information is generally required
- Persisting data in user tables is popular

- Consider "stamping" stored procedures with a dummy parameter value (e.g. SSIS execution GUID).
- This value will appear in command text of sys.dm_pdw_exec_requests
- By scanning command text for the dummy value all queries associated to the session can be identified

Capturing History

```
CREATE VIEW [dbo].[vSessionRequestMetaData]
AS
SELECT
 s.Session_ID       as Session_ID
,s.[status]         as Session_Status
,s.login_name       as Session_LoginName
,s.login_time       as Session_LoginTime
,r.request_id       as Request_ID
,r.[status]         as Request_Status
,submit_time        as Request_SubmitTime
,start_time         as Request_StartTime
,end_compile_time   as Request_EndCompileTime
,end_time           as Request_EndTime
,total_elapsed_time as Request_TotalElapsedDuration_ms
,DATEDIFF(ms,submit_time,start_time)     as Request_InitiateDuration_ms
,DATEDIFF(ms,start_time,end_compile_time)as Request_CompileDuration_ms
,DATEDIFF(ms,end_compile_time,end_time)  as Request_ExecDuration_ms
FROM sys.dm_pdw_exec_requests r
JOIN sys.dm_pdw_exec_sessions s  ON  r.session_id = s.session_id
LEFT JOIN sys.dm_pdw_errors e    ON  r.error_id   = e.error_id
                                 AND r.session_id = e.session_id
                                 AND r.request_id = e.request_id;
```

```
,[label]       as Request_QueryLabel
,command       as Request_Command
,database_id   as Request_Database_ID
,e.source      as Error_Source
,e.[type]      as Error_Type
,e.create_time as Error_CreateTime
,e.pdw_node_id as Error_PDWNodeID
,e.spid        as Error_SPID
,e.thread_id   as Error_Thread
,e.details     as Error_Details
,SYSDATETIME()  as Log_Date
```

Capturing History

```
CREATE PROC [dbo].[csp_ETL_LogState]
@pRunGUID NVARCHAR(38),@pPkgName NVARCHAR(256),@pLogGUID NVARCHAR(38)
AS
INSERT INTO dbo.ETL_ProcessLog
SELECT  @pRunGUID as RunGUID
,       @pPkgName as PkgName
,       @pLogGUID as LogGUID
,       m.*
,       SYSDATETIME() AS Log_Date
FROM dbo.vSessionRequestMetadata m
JOIN (Select Session_ID
from dbo.vSessionRequestMetadata where Request_Command like '%'+@pLogGUID+'%'
AND Session_ID <> Session_ID()
GROUP BY Session_ID
) s
ON m.Session_ID = s.Session_ID
OPTION (Label='dbo.csp_ETL_LogState : Step 1 : Insert into ETL_ProcessLog')
```

Field Note on Logging Tables

Tend to be

- Quite Small
- Write Heavy

When Replicated

- Slows ELT process
- Can cause deadlocks

Top Tip
Distribute the table for enhanced write speed

Loading Parallelism Limits

- 10 Concurrent Loads
- 40 Queued Loads
- 51st Load – exception

1 dwloader running = 1 Load

Each SSIS PDW Destination Adapter = 1 Load

Dual Loading Considerations

Where and when to split the load

- Pre-Staging?
- Post-Staging?

Single Shared Surrogate Key?

- Shared meaning surrogate key
- Split brain surrogate key

Dual Loading

Execution

- Deferred DR Run
- Parallel DR Run

Reconciliation

- Values i.e. Sum of sales for the month
- Counts i.e. Number of rows

Housekeeping

- When do you clean up staging tables?
- After the ETL Run?

Consider this approach:

- Drop tables at the start of the next ETL Run
- Use Staging tables for reconciliation
- Use Partition Switch out tables for fast failback
- Partition switch much faster than Restore DB

Altering SSAS designs

SSAS Options

Multidimensional

- MDX
- MOLAP
- ROLAP
- NUMA aware

Tabular

- DAX
- InMemory (MOLAP)
- DirectQuery (ROLAP)
- Power BI
- Not NUMA aware
- DirectQuery limited by DAX

Connecting SSAS

Networking

- IB Network
- 10GbE
- 1GbE

Product Version

- 2012 SP1+ preferred
- Distinct Count optimisation
  - *EnableRolapDistinctCountOnDataSource*
  - msmdsrv.ini file

Supporting Environments

Requirements

Environments

- Development
- Test
- UAT
- Production
- Disaster Recovery

Boundary

- Physical
- Logical

Challenges

Backup and Restore not always viable:

- Physical Constraints
  - Development environment may be smaller than production
  - May not have >1 PDW appliance
  - Only 1 database backup running at a time
  - No point in time backups
- Security Constraints
  - Data may be sensitive or contain personally identifiable information

Data Delivery Considerations

- Data Sub-setting
- Obfuscation Requirements
- Tokenisation
- Data Latency
- Reconciliation

Code Management

- PDWScripter Utility available
  - Generates Scripts of all objects in PDW
- SSDT Projects system does not support PDW
  - Use a SQL project as a container
  - Add files to projects as User Scripts (not in build)
  - Correct file extension to .dsql
  - Integrate code with TFS / GIT
- Alter Scripts required for release management
  - Some object types (e.g. Views are Create/Drop only)
- Automate deployment with Powershell