[

开篇介绍

在这篇日志中 如何在 ETL 项目中统一管理上百个 SSIS 包的日志和包配置框架 我介绍到了包级别的日志管理框架，那么这个主要是针对包这一个层级的 Log 信息，包括包开始执行和结束时间，以及各个包的执行成功或者失败状态。

但是我们可以更加深一层次的将日志记录 Logging 以及数据信息 Auditing 信息延伸到包中的重要 Task 中。

通常情况下，SSIS 包从各个数据源加载数据到 Staging 表中，数据源可以是文件，也可以是其它数据库。然后经过数据仓库 SCD 以及 Lookup 等操作，将 Staging 中的数据清理并整理加载到各个维度以及事实表中。

假设我们需要知道在当前操作中，各个 Staging 表加载了多少数据，使用了多长时间。各个处理维度和事实的 Task 使用了多少时间，新增了多少数据，修改了多少数据。这些我们也是有能力做到的，如果再配合 如何在 ETL 项目中统一管理上百个 SSIS 包的日志和包配置框架 这篇文章中提到的包级别日志记录，那么我们将非常清晰的知道我们的 SSIS 包无论是在包级别，还是在各个重要 Task 级别的各种日志，数据信息。这些信息对于我们的包维护，性能分析，错误纠正，错误修复都是非常有价值的。

比如，我可以很轻松的通过自定义的报表浏览哪些 Task 在同等记录情况下最耗时间，各个 Task 在整个包的执行过程中所用的时间比。

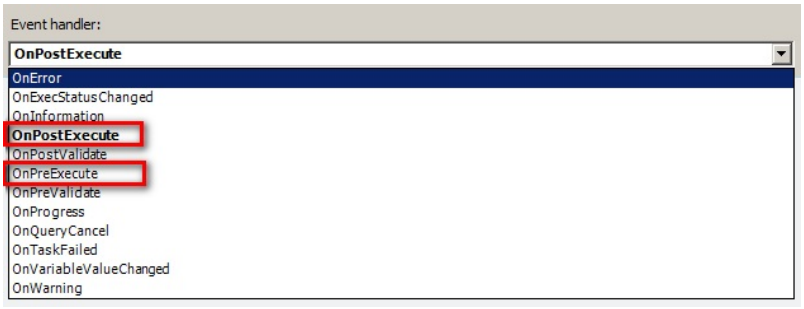
TASK_NAME	TABLE_NAME	ExistingRowsBefore	StartTime	DeletedRows	UpdatedRows	InsertedRows	ExistingRowsAfter	EndTime	ExecutionStatus
EST_TRUNCATE_SALES_ORDER_DETAIL	SalesOrderDetail	542	2014-04-17 22:53:51.817	542	0	0	0	2014-04-17 22:53:51.933	1
DFT_LOADING_DATA	SalesOrderDetail	0	2014-04-17 22:53:51.987	0	0	542	542	2014-04-17 22:53:52.380	1

不同的 ETL 项目在 Auditing 上会采取不同的策略，比如以文件加载为主的 ETL 是允许有部分错误数据加载失败的，但是以数据仓库为主的 ETL 则不希望出现错误数据加载的。因此在设计 Auditing 的时候要考虑到这些情况，比如设计的时候多出一个 失败数据总数的记录用于跟踪文件数据等。

在这里只讲如何实现 Auditing，简单的介绍一下核心操作，大家可以在这个基础之上去扩充。

关键点

实现 Auditing 的关键点就是要借用控制流 Task 中的 Event Handler 下的 OnPostExecute 和 OnPreExecute 功能。

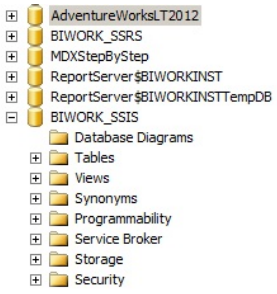


1. OnError 功能已经在这篇 如何在 ETL 项目中统一管理上百个 SSIS 包的日志和包配置框架 文章中详细的介绍到了。
2. OnPreExecute - 在 Task 执行之前触发的事件。
3. OnPostExecute - 在 Task 执行完成之后触发的事件。

通过这样两个事件我们很容易实现对 Task 执行前后表数据变化的操作记录。

数据源，目标表及其它数据库对象

测试数据源是 AdventureWorksLT2012



BIWORK_SSIS 数据库中的目标表

```
IF OBJECT_ID('dbo.SalesOrderDetail') IS NOT NULL

DROP TABLE dbo.SalesOrderDetail

GO
```

```
CREATE TABLE [dbo].[SalesOrderDetail](

    [SalesOrderID] [int] NOT NULL,

    [SalesOrderDetailID] [int] NOT NULL,

    [OrderQty] [smallint] NOT NULL,

    [ProductID] [int] NOT NULL,

    [UnitPrice] [money] NOT NULL,

    [UnitPriceDiscount] [money] NOT NULL,

    [LineTotal] [numeric](38, 6) NOT NULL,

    [rowguid] [uniqueidentifier] NOT NULL,

    [ModifiedDate] [datetime] NOT NULL

) ON [PRIMARY]

GO
```

Task 执行状态表

EXECUTION_ID 应该使用 [如何在 ETL 项目中统一管理上百个 SSIS 包的日志和包配置框架] 中的 PROCESS_LOG_ID，这样就将 SSIS 包日志和 TASK 关联起来了。

```
USE BIWORK_SSIS

GO


IF OBJECT_ID('TASK_EXECUTION_STATUS') IS NOT NULL

DROP TABLE TASK_EXECUTION_STATUS

GO


CREATE TABLE TASK_EXECUTION_STATUS

(

    EXECUTION_ID NVARCHAR(255),

    PACKAGE_NAME NVARCHAR(100),

    TASK_ID NVARCHAR(250),

    TASK_NAME NVARCHAR(250),

    TABLE_NAME NVARCHAR(250),

    ExistingRowsBefore BIGINT,

    StartTime DATETIME,

    DeletedRows BIGINT,

    UpdatedRows BIGINT,

    InsertedRows BIGINT,

    ExistingRowsAfter BIGINT,

    EndTime DATETIME,

    ExecutionStatus INT
```

)

获取表的条数

```
IF OBJECT_ID('dbo.GET_TABLE_COUNT') IS NOT NULL

DROP PROCEDURE dbo.GET_TABLE_COUNT

GO


CREATE PROCEDURE dbo.GET_TABLE_COUNT

@TABLE_NAME NVARCHAR(50),

@ROW_COUNT BIGINT OUTPUT

AS

BEGIN


    SELECT @ROW_COUNT = SUM(PART.rows)

    FROM sys.tables TBL

    INNER JOIN sys.partitions PART ON TBL.object_id = PART.object_id

    INNER JOIN sys.indexes IDX ON PART.object_id = IDX.object_id

    AND PART.index_id = IDX.index_id

    WHERE TBL.name = @TABLE_NAME

    AND IDX.index_id < 2

    GROUP BY TBL.object_id, TBL.name


    RETURN @ROW_COUNT

END

GO
```

记录时间

这个存储过程用来每次在执行 Task 之前获取目标表中的条数，并且插入 Task 启动时间 -

```
IF OBJECT_ID('dbo.USP_INSERT_TASK_EXECUTION','P') IS NOT NULL

DROP PROCEDURE dbo.USP_INSERT_TASK_EXECUTION

GO


CREATE PROCEDURE USP_INSERT_TASK_EXECUTION

    @TARGET_TABLE_NAME NVARCHAR(50),

    @EXECUTION_ID NVARCHAR(255)  ,

    @PACKAGE_NAME NVARCHAR(100),

    @TASK_ID NVARCHAR(255),

    @TASK_NAME NVARCHAR(250)

AS

BEGIN
```

```
DECLARE @ExistingRowsBefore BIGINT

EXECUTE dbo.GET_TABLE_COUNT
@TABLE_NAME = @TARGET_TABLE_NAME,
@ROW_COUNT = @ExistingRowsBefore OUTPUT

INSERT INTO TASK_EXECUTION_STATUS
(
    EXECUTION_ID,
    PACKAGE_NAME,
    TASK_ID,
    TASK_NAME,
    TABLE_NAME,
    ExistingRowsBefore,
    StartTime,
    DeletedRows,
    UpdatedRows,
    InsertedRows,
    ExistingRowsAfter,
    EndTime,
    ExecutionStatus
)
VALUES
(
    @EXECUTION_ID,
    @PACKAGE_NAME,
    @TASK_ID,
    @TASK_NAME,
    @TARGET_TABLE_NAME,
    @ExistingRowsBefore,
    GETDATE(),
    NULL, --@DeletedRows,
    NULL, --@UpdatedRows,
    NULL, --@InsertedRows,
    NULL, --@ExistingRowsAfter
    NULL, --@EndTime
    0 -- In process
```

```
)
END
```

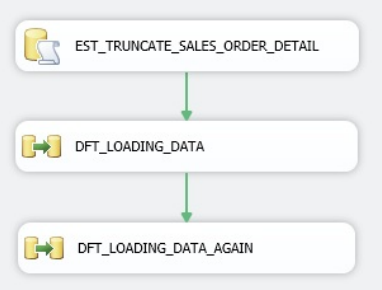
更新 Task 状态表

当 @DeletedRows = -1 的时候，表明操作是 Truncate 操作。

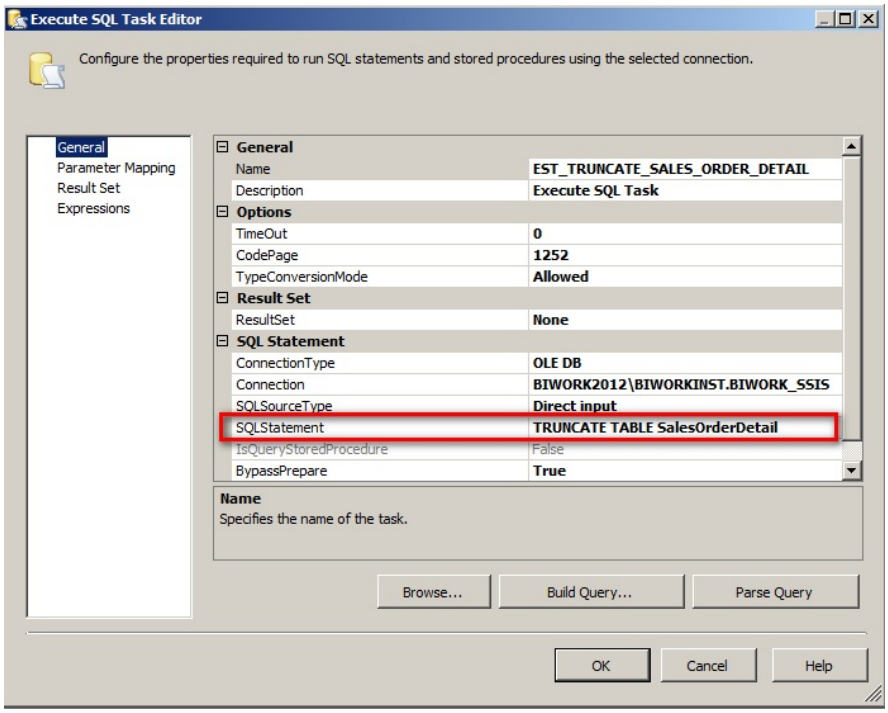
```
IF OBJECT_ID('dbo.USP_UPDATE_TASK_EXECUTION' ) IS NOT NULL
DROP PROCEDURE dbo.USP_UPDATE_TASK_EXECUTION
GO

CREATE PROCEDURE dbo.USP_UPDATE_TASK_EXECUTION
@ExecutionID NVARCHAR(250),
@TaskID NVARCHAR(250),
@DeletedRows BIGINT,
@UpdatedRows BIGINT,
@InsertedRows BIGINT
AS
BEGIN

    UPDATE dbo.TASK_EXECUTION_STATUS
    SET DeletedRows = (CASE WHEN @DeletedRows = -1 THEN ExistingRowsBefore ELSE @DeletedRows END),
        UpdatedRows = (CASE WHEN @DeletedRows = -1 THEN 0 ELSE @UpdatedRows END),
        InsertedRows = (CASE WHEN @DeletedRows = -1 THEN 0 ELSE @InsertedRows END),
        ExistingRowsAfter = (CASE WHEN @DeletedRows = -1 THEN 0 ELSE (ExistingRowsBefore + @InsertedRows - @DeletedRows) END),
        EndTime = GETDATE(),
        ExecutionStatus = 1
    WHERE EXECUTION_ID = @ExecutionID
        AND TASK_ID = @TaskID
END
GO
```

SSIS 包中的流程实现

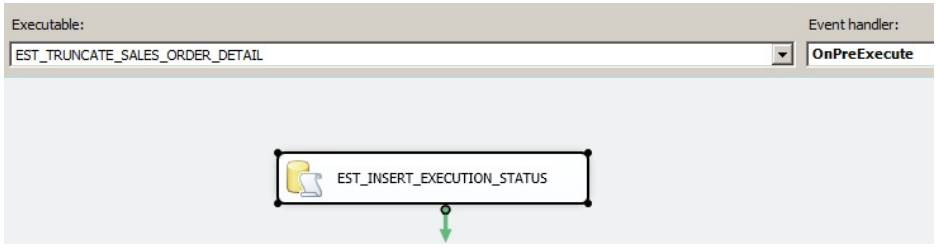
SSIS 包 - 第二个和第三个 Task 的功能完全一样，只为了演示的目的。



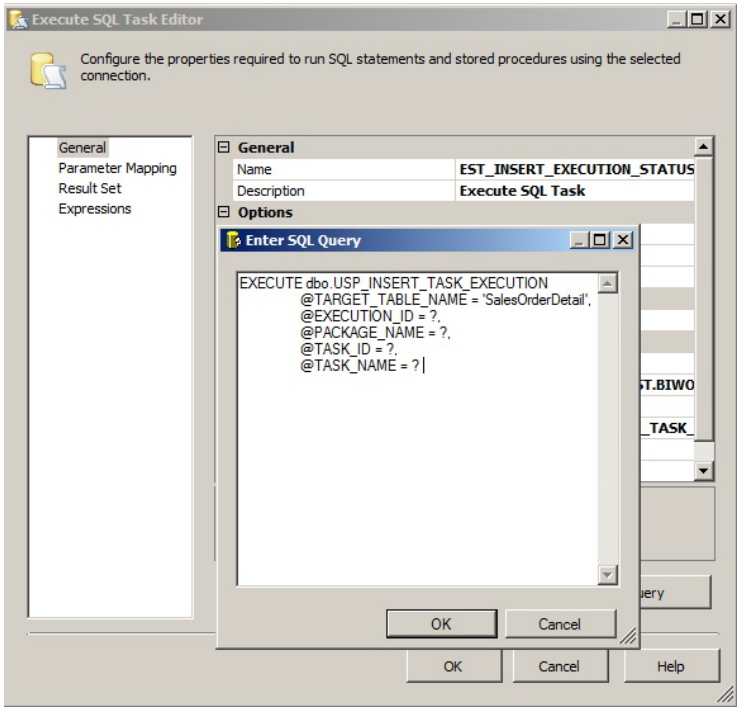
EST_TRUNCATE_SALES_ORDER_DETAIL Task - 在加载数据之前删除表数据。



它的 OnPreExecute 事件中添加了一个 Execute SQL Task 组件用来向 Task Execution 表插入操作前的记录。

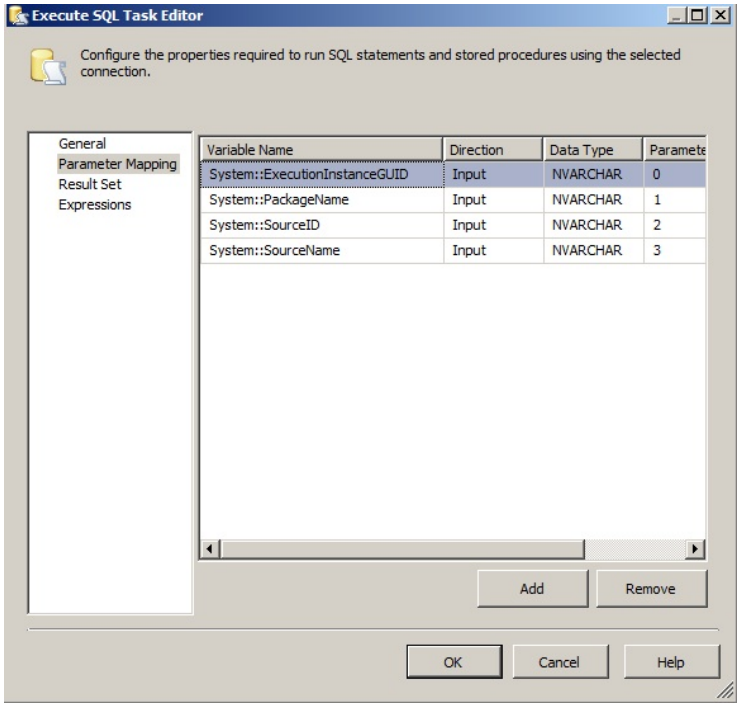


调用 USP_INSERT_TASK_EXECUTION 存储过程根据表名查询记录数。



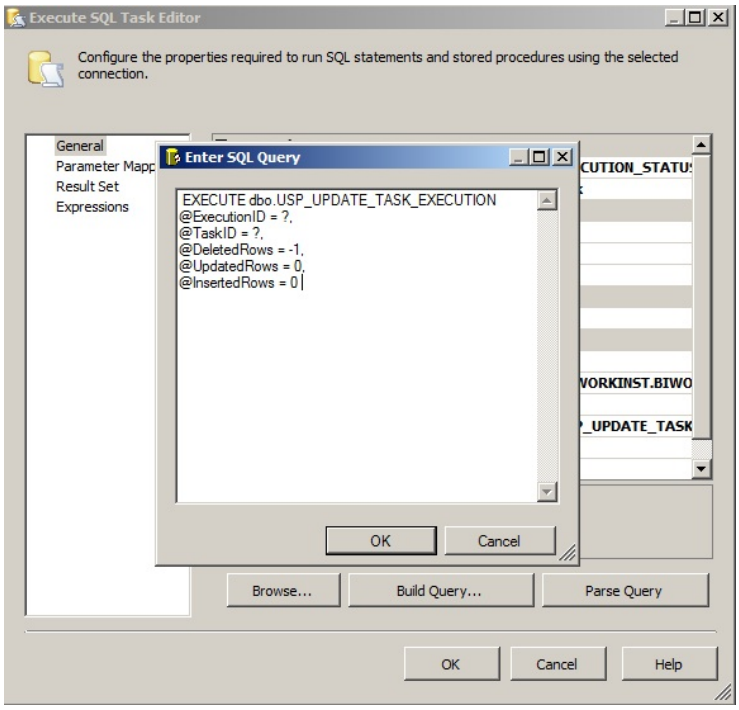
参数 Mapping 关系，注意这里要用 Source ID，Source Name 而不是 Task ID，Task Name。因为 Task 是指当前执行这些 SQL 的 Task 自身，而

我们要监控的不是这个事件下的 Task ，而是控制流中的 Task 组件。

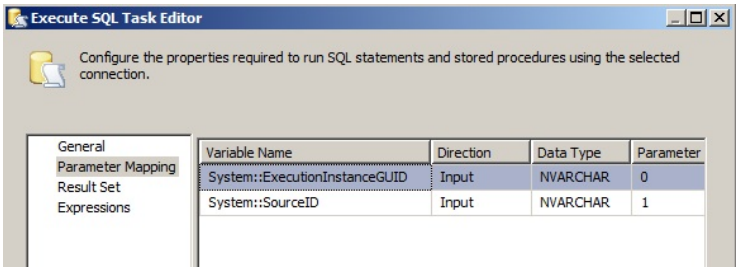


EST_TRUNCATE_SALES_ORDER_DETAIL 中 OnPostExecute 的配置 -

这里就是 Update 操作了，因为 EST_TRUNCATE_SALES_ORDER_DETAIL 是 Truncate 表操作，所以这里给了 DeletedRows = -1。



更新的时候直接根据 Execution ID 和 Task ID 就可以了。

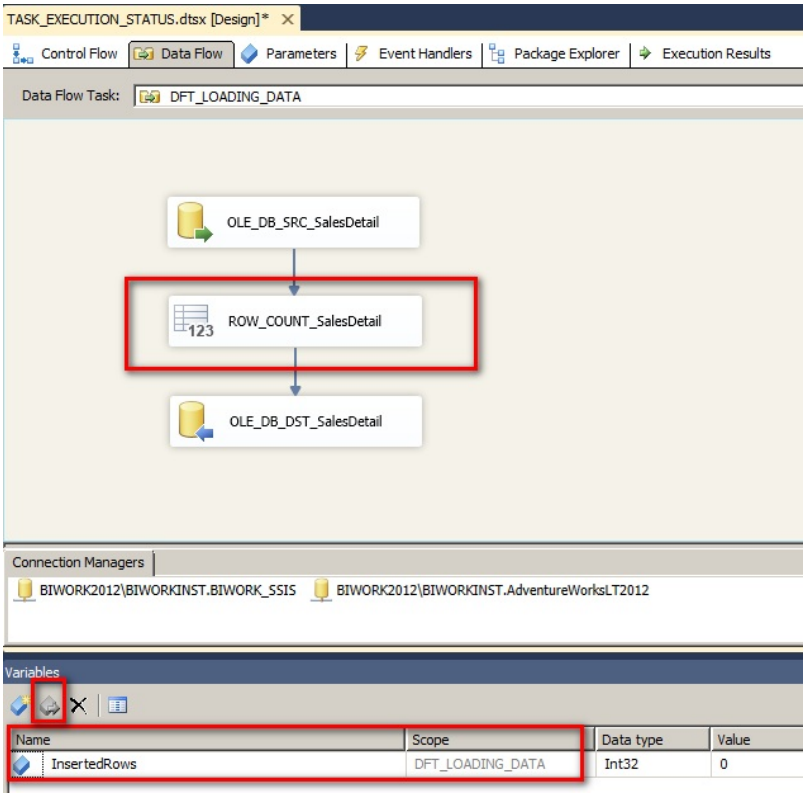


第二个 Task 也要配置 OnPreExecute 和 OnPostExecute 事件，也就是说每一个你需要监控的 Task 都要配置。感觉比较复杂，但是一次配置完成以后，受用可是长期的。

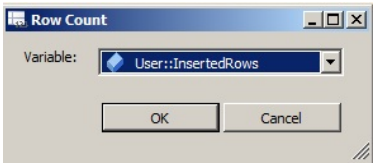
要注意的是第二个 Task 是从数据源加载数据，这样需要在加载的过程中获取记录数，通过 ROW COUNT 可以实现将数据流的条数赋值给变量保存。

另外要注意的是 - 这个变量的 SCOPE 是控制流组件自身，即作用域。因为可能要有很多 Task 需要用到记录条数的变量，全部放到包级别中这个变量会非常多，并且容易出错。可以理解为 InsertedRows 是局部变量，它的生命周期就是 Task 本身。

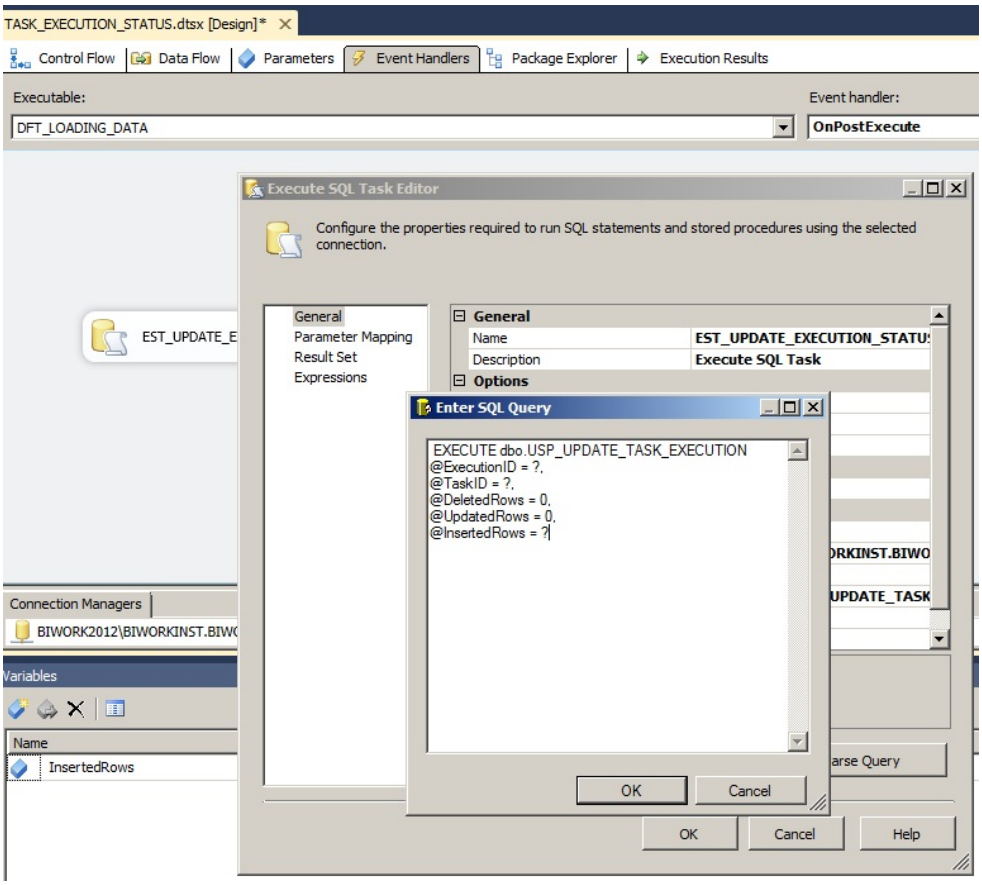
如果创建的变量位于包级别 SCOPE，可以点击下方的小方框 Move 到当前 Task 的 SCOPE 中。



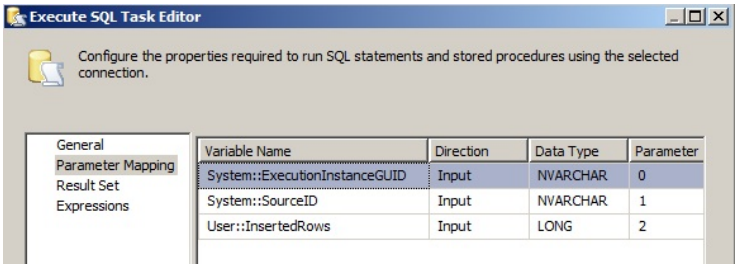
变量的赋值。



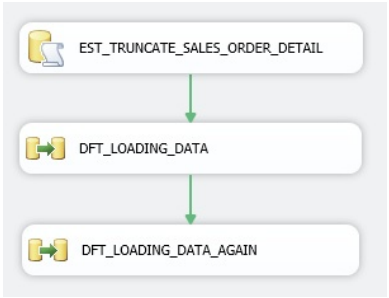
OnPreExecute 的配置和上面的 Task 一样，复制一份即可，这里是 OnPostExecute 的配置。



需要什么变量就记录什么变量，就配置什么变量。



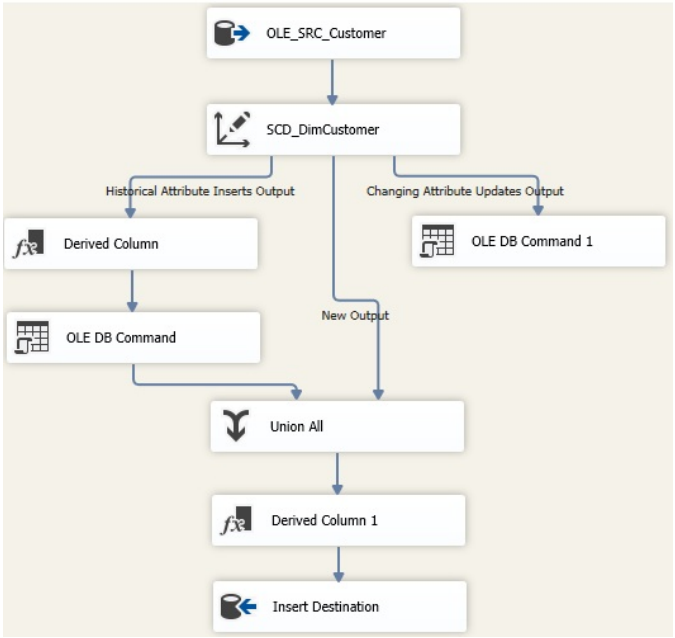
后面的两个 Task 一模一样，只是为了测试使用。



运行两次的结果，数据条数的记录是非常连贯的。

	EXECUTION_ID	PACKAGE_NAME	TASK_ID	TASK_NAME	TABLE_NAME	ExistingRowsBefore	StartTime	DeletedRows	UpdatedRows	InsertedRows	ExistingRowsAfter	EndTime	ExecutionStatus
1	{1F498817-C951-42BB-8593-33E1D7BC4264}	TASK_EXECUTION_STATUS	{EB514C78-4406-4757-8317-93AA59F2A8A3}	EST_TRUNCATE_SALES_ORDER_DETAIL	SalesOrderDetail	0	2014-04-18 00:29:19.617	0	0	0	0	2014-04-18 00:29:19.760	1
2	{1F498817-C951-42BB-8593-33E1D7BC4264}	TASK_EXECUTION_STATUS	{C35F5393-1ED9-44F9-9876-AD352917DC2C}	DFT_LOADING_DATA	SalesOrderDetail	0	2014-04-18 00:29:19.817	0	0	542	542	2014-04-18 00:29:20.343	1
3	{1F498817-C951-42BB-8593-33E1D7BC4264}	TASK_EXECUTION_STATUS	{533a5416-5abb-4901-968c-6848553c5658}	DFT_LOADING_DATA_AGAIN	SalesOrderDetail	542	2014-04-18 00:29:20.390	0	0	542	1084	2014-04-18 00:29:20.693	1
4	{B8DEA955-4E42-4EDF-8312-9AD9A60316C8}	TASK_EXECUTION_STATUS	{EB514C78-4406-4757-8317-93AA59F2A8A3}	EST_TRUNCATE_SALES_ORDER_DETAIL	SalesOrderDetail	1084	2014-04-18 00:43:04.520	1084	0	0	0	2014-04-18 00:43:04.577	1
5	{B8DEA955-4E42-4EDF-8312-9AD9A60316C8}	TASK_EXECUTION_STATUS	{C35F5393-1ED9-44F9-9876-AD352917DC2C}	DFT_LOADING_DATA	SalesOrderDetail	0	2014-04-18 00:43:04.620	0	0	542	542	2014-04-18 00:43:04.910	1
6	{B8DEA955-4E42-4EDF-8312-9AD9A60316C8}	TASK_EXECUTION_STATUS	{533a5416-5abb-4901-968c-6848553c5658}	DFT_LOADING_DATA_AGAIN	SalesOrderDetail	542	2014-04-18 00:43:04.953	0	0	542	1084	2014-04-18 00:43:05.200	1

记录 SCD 的修改和新增条数只需要在相应的地方添加 ROW COUNT 组件来捕获即可。



当然除了使用 ROW COUNT 组件，在某些特定的情况下也可以使用 @@ROWCOUNT 来获取新增，删除或者修改所影响到的条数。

```
DECLARE @UpdateRowCnt INT

DECLARE @InsertRowCnt INT


--Inserting records from Source to Destination which does not exists

insert into dbo.Client(ClientName,Country,Town)

Select clientName, Country, Town from dbo.ClientSource S

WHERE NOT EXISTS ( Select 1 from dbo.Client CL WHERE CL.ClientName=S.ClientName)

SELECT @InsertRowCnt=@@ROWCOUNT


--Update Already existing records from Source

Update CL

set CL.ClientType=S.CLientType

from dbo.Client CL

INNER JOIN dbo.ClientSource S

ON C1.ClientName=S.ClientName

SELECT @UpdateRowCnt=@@ROWCOUNT
```

最后一个问题

如果每次都在各个 Task 中的 OnPreExecute 和 OnPostExecute 中配置非常麻烦，有没有改进的方法。

答案是有的。

我提供一个思路，有兴趣的话可以自动动手尝试 -

Task 级别的 OnPreExecute 和 OnPostExecute 事件是当 Task 被执行前后被触发的，要注意的是包级别的 OnPreExecute 和 OnPostExecute 也是可以捕获 Task 级别的 OnPreExecute 和 OnPostExecute 事件。

可以定义一张表，表中记录需要被处理的 Task 名称，然后在包级别的 OnPreExecute 和 OnPostExecute 中处理 各个 Task 的 Auditing 信息。不在列表上的，就可以不用处理。

同时还要注意 Task 同步的问题，若是很多 Task 同时执行，并行执行的话，就需要在各自 Task 中定义好变量来记录然后再赋值给 Package 级别

的变量可以避免这一问题。

与本文相关的文章

如何在 ETL 项目中统一管理上百个 SSIS 包的日志和包配置框架

]