Practical ETL 2

# Transform

## Preface

In this class we are going to be using stored procs to illustrate the logic and data flow behind transform and load steps. Stored procs are generally a difficult, inefficient way to handle ETL except for the simplest of implementations. ETL tools such as SSIS are better suited to this task.

## Extracts

In this class we are going to create the Cities, Customers, and Orders transformations.

In our previous class, we looked at how to create an extract by looking at the customers extract. The remaining extracts were assigned as homework, but we are going to need the orders stage table for this lesson, so let's create it now:

CREATE TABLE dbo.Orders\_Stage (

    OrderDate    DATE,

Quantity     INT,

UnitPrice    DECIMAL(18,2),

TaxRate      DECIMAL(18,3),

CustomerName NVARCHAR(100),

CityName     NVARCHAR(50),

StateProvinceName NVARCHAR(50),

CountryName  NVARCHAR(60),

StockItemName NVARCHAR(100),

LogonName    NVARCHAR(50)

);

*NOTE: The transformations we will be creating in this class cannot be executed successfully until the extracts they are dependent on are built and executed.*

## Preload Tables

Just like with our extract, we need a staging table for our transfom steps to load into. We decided to call these "preload" tables to differentiate them from our staging tables. In practice, many different naming conventions exist.

Post-transform staging tables should reflect the structure of the destination table, so let's create them now:

CREATE TABLE dbo.Cities\_Preload (

CityKey INT NOT NULL,

CityName NVARCHAR(50) NULL,

StateProvCode NVARCHAR(5) NULL,

StateProvName NVARCHAR(50) NULL,

CountryName NVARCHAR(60) NULL,

CountryFormalName NVARCHAR(60) NULL,

CONSTRAINT PK\_Cities\_Preload PRIMARY KEY CLUSTERED ( CityKey )

);

CREATE TABLE dbo.Customers\_Preload (

CustomerKey INT NOT NULL,

CustomerName NVARCHAR(100) NULL,

CustomerCategoryName NVARCHAR(50) NULL,

DeliveryCityName NVARCHAR(50) NULL,

DeliveryStateProvCode NVARCHAR(5) NULL,

DeliveryCountryName NVARCHAR(50) NULL,

PostalCityName NVARCHAR(50) NULL,

PostalStateProvCode NVARCHAR(5) NULL,

PostalCountryName NVARCHAR(50) NULL,

StartDate DATE NOT NULL,

EndDate DATE NULL,

CONSTRAINT PK\_Customers\_Preload PRIMARY KEY CLUSTERED ( CustomerKey )

);

CREATE TABLE dbo.Orders\_Preload (

CustomerKey INT NOT NULL,

CityKey INT NOT NULL,

ProductKey INT NOT NULL,

SalespersonKey INT NOT NULL,

DateKey INT NOT NULL,

Quantity INT NOT NULL,

UnitPrice DECIMAL(18, 2) NOT NULL,

TaxRate DECIMAL(18, 3) NOT NULL,

TotalBeforeTax DECIMAL(18, 2) NOT NULL,

TotalAfterTax DECIMAL(18, 2) NOT NULL,

);

Notice that for the dimension tables the structures of the tables are identical to their data mart counterparts. In our fact table, all our foreign key constraints have been removed, but the structure is otherwise the same. Also notice that none of the keys have been created as identities, just like the tables they're based on, but that the keys are not nullable.

So if we're not using identities, where are these surrogate keys coming from?

## SEQUENCE

<https://docs.microsoft.com/en-us/sql/t-sql/statements/create-sequence-transact-sql>

A sequence is an object that works very similar to an identity – it is an auto-incrementing number that can be used to create surrogate keys. However, unlike an identity, a sequence is a stand-alone object that can be called. This has a few benefits:

* A sequence is not affected by TRUNCATE
* We can insert into our table without having to generate a new key, which allows us to use a mix of generated and existing keys

Let's create sequences for our Cities and Customers tables:

CREATE SEQUENCE dbo.CityKey START WITH 1;

CREATE SEQUENCE dbo.CustomerKey START WITH 1;

The create sequence statement is simple. We provide a unique object name. We may also optionally provide a seed (START WITH), otherwise it defaults to the min value for our datatype. To retrieve the next value from a sequence, we use NEXT VALUE FOR <sequence name>. We can use this in our insert statement, as we will see in our transformation.

A **sequence** is an object that stores an auto-incrementing number. It is not directly associated with any table or column.

It is useful in ETL because it permits us to truncate our stage tables and insert current records without being forced to generate new keys for them.

## Transform – Type 1 SCD

Now that we have our required staging tables and sequences, let's create the transformation for Cities, which we decided to implement as a Type 1 SCD. Let's create the transformation, then walk through the code:

CREATE PROCEDURE dbo.Cities\_Transform

AS

BEGIN;

SET NOCOUNT ON;

SET XACT\_ABORT ON;

TRUNCATE TABLE dbo.Cities\_Preload;

BEGIN TRANSACTION;

INSERT INTO dbo.Cities\_Preload /\* Column list excluded for brevity \*/

SELECT NEXT VALUE FOR dbo.CityKey AS CityKey,

cu.DeliveryCityName,

cu.DeliveryStateProvinceCode,

cu.DeliveryStateProvinceName,

cu.DeliveryCountryName,

cu.DeliveryFormalName

FROM dbo.Customers\_Stage cu

WHERE NOT EXISTS ( SELECT 1

FROM dbo.DimCities ci

WHERE cu.DeliveryCityName = ci.CityName

AND cu.DeliveryStateProvinceName = ci.StateProvName

AND cu.DeliveryCountryName = ci.CountryName );

INSERT INTO dbo.Cities\_Preload /\* Column list excluded for brevity \*/

SELECT ci.CityKey,

cu.DeliveryCityName,

cu.DeliveryStateProvinceCode,

cu.DeliveryStateProvinceName,

cu.DeliveryCountryName,

cu.DeliveryFormalName

FROM dbo.Customers\_Stage cu

JOIN dbo.DimCities ci

ON cu.DeliveryCityName = ci.CityName

AND cu.DeliveryStateProvinceName = ci.StateProvName

AND cu.DeliveryCountryName = ci.CountryName;

COMMIT TRANSACTION;

END;

Here are a few important things to note:

* We are using the **business key** to match records.
* We are using our existing surrogate key if one exists.
* We are using our **sequence** to create **new surrogate keys** as required.

*NOTE: There is a more advanced way to combine data sets that allows us to perform inserts, updates, and/or deletes simultaneously – the MERGE statement. This is outside the scope of this course. If you are interested in learning more about it, it can be found here:* <https://docs.microsoft.com/en-us/sql/t-sql/statements/merge-transact-sql>

## Transform – Type 2 SCD

Let's create the transformation for Customers, which is a Type 2 SCD.

The logic for Type 2 SCDs is a bit more complicated because we need to track changes to our records. If our non-key attributes change we need to create a new record, but also update the previous record.

Once again, let's create the code then walk through it:

CREATE PROCEDURE dbo.Customers\_Transform

AS

BEGIN;

SET NOCOUNT ON;

SET XACT\_ABORT ON;

TRUNCATE TABLE dbo.Customers\_Preload;

DECLARE @StartDate DATE = GETDATE();

DECLARE @EndDate DATE = DATEADD(dd,-1,GETDATE());

BEGIN TRANSACTION;

-- Add updated records

INSERT INTO dbo.Customers\_Preload /\* Column list excluded for brevity \*/

SELECT NEXT VALUE FOR dbo.CustomerKey AS CustomerKey,

stg.CustomerName,

stg.CustomerCategoryName,

stg.DeliveryCityName,

stg.DeliveryStateProvinceCode,

stg.DeliveryCountryName,

stg.PostalCityName,

stg.PostalStateProvinceCode,

stg.PostalCountryName,

@StartDate,

NULL

FROM dbo.Customers\_Stage stg

JOIN dbo.DimCustomers cu

ON stg.CustomerName = cu.CustomerName

AND cu.EndDate IS NULL

WHERE stg.CustomerCategoryName <> cu.CustomerCategoryName

OR stg.DeliveryCityName <> cu.DeliveryCityName

OR stg.DeliveryStateProvinceCode <> cu.DeliveryStateProvCode

OR stg.DeliveryCountryName <> cu.DeliveryCountryName

OR stg.PostalCityName <> cu.PostalCityName

OR stg.PostalStateProvinceCode <> cu.PostalStateProvCode

OR stg.PostalCountryName <> cu.PostalCountryName;

-- Add existing records, and expire as necessary

INSERT INTO dbo.Customers\_Preload /\* Column list excluded for brevity \*/

SELECT cu.CustomerKey,

cu.CustomerName,

cu.CustomerCategoryName,

cu.DeliveryCityName,

cu.DeliveryStateProvCode,

cu.DeliveryCountryName,

cu.PostalCityName,

cu.PostalStateProvCode,

cu.PostalCountryName,

cu.StartDate,

CASE

WHEN pl.CustomerName IS NULL THEN NULL

ELSE @EndDate

END AS EndDate

FROM dbo.DimCustomers cu

LEFT JOIN dbo.Customers\_Preload pl

ON pl.CustomerName = cu.CustomerName

AND cu.EndDate IS NULL;

-- Create new records

INSERT INTO dbo.Customers\_Preload /\* Column list excluded for brevity \*/

SELECT NEXT VALUE FOR dbo.CustomerKey AS CustomerKey,

stg.CustomerName,

stg.CustomerCategoryName,

stg.DeliveryCityName,

stg.DeliveryStateProvinceCode,

stg.DeliveryCountryName,

stg.PostalCityName,

stg.PostalStateProvinceCode,

stg.PostalCountryName,

@StartDate,

NULL

FROM dbo.Customers\_Stage stg

WHERE NOT EXISTS ( SELECT 1 FROM dbo.DimCustomers cu WHERE stg.CustomerName = cu.CustomerName );

-- Expire missing records

INSERT INTO dbo.Customers\_Preload /\* Column list excluded for brevity \*/

SELECT cu.CustomerKey,

cu.CustomerName,

cu.CustomerCategoryName,

cu.DeliveryCityName,

cu.DeliveryStateProvCode,

cu.DeliveryCountryName,

cu.PostalCityName,

cu.PostalStateProvCode,

cu.PostalCountryName,

cu.StartDate,

@EndDate

FROM dbo.DimCustomers cu

WHERE NOT EXISTS ( SELECT 1 FROM dbo.Customers\_Stage stg WHERE stg.CustomerName = cu.CustomerName )

AND cu.EndDate IS NULL;

COMMIT TRANSACTION;

END;

Here are important things to note:

* Our StartDate is always today's date. In a real-world ETL, this would typically be configurable.
* Our EndDate is always StartDate – 1 day.
* Since this is a Type 2 SCD, there can be **multiple records per business key**. It is very important that we always use **EndDate IS NULL** to ensure that we are always working with the **current record**.
* We are comparing our extracted records against the current records in our data mart table:
  + When there is a match, but **no non-key attributes have changed**, we **add the record as-is** so it is available for our fact transform.
  + When there is a match and a **non-key attribute has changed**, we **create a new record** and **expire the previous record**.
  + When there is **no match in the data mart**, we **create a new record**.
  + When there is **no match in our extract** for a record in the data mart, we **expire the existing record**.
* As with our Type 1 SCD, we are using our business key to match records.

## Transform – Fact

Our fact table is dependent on all our dimensions. You will be creating our other two transformations as homework, but we need to create the stage tables to finish our fact transformation, so let's do that now:

CREATE TABLE dbo.Products\_Preload (

ProductKey INT NOT NULL,

ProductName NVARCHAR(100) NULL,

ProductColour NVARCHAR(20) NULL,

ProductBrand NVARCHAR(50) NULL,

ProductSize NVARCHAR(20) NULL,

StartDate DATE NOT NULL,

EndDate DATE NULL,

CONSTRAINT PK\_Products\_Preload PRIMARY KEY CLUSTERED ( ProductKey )

);

CREATE TABLE dbo.SalesPeople\_Preload (

SalespersonKey INT NOT NULL,

FullName NVARCHAR(50) NULL,

PreferredName NVARCHAR(50) NULL,

LogonName NVARCHAR(50) NULL,

PhoneNumber NVARCHAR(20) NULL,

FaxNumber NVARCHAR(20) NULL,

EmailAddress NVARCHAR(256) NULL,

CONSTRAINT PK\_SalesPeople\_Preload PRIMARY KEY CLUSTERED (SalespersonKey )

);

The code for our fact transformation is relatively simple. Let's create it and walk through it:

CREATE PROCEDURE dbo.Orders\_Transform

AS

BEGIN;

SET NOCOUNT ON;

SET XACT\_ABORT ON;

TRUNCATE TABLE dbo.Orders\_Preload;

INSERT INTO dbo.Orders\_Preload /\* Columns excluded for brevity \*/

SELECT cu.CustomerKey,

ci.CityKey,

pr.ProductKey,

sp.SalespersonKey,

CAST(YEAR(ord.OrderDate) \* 10000 + MONTH(ord.OrderDate) \* 100 + DAY(ord.OrderDate) AS INT),

SUM(ord.Quantity) AS Quantity,

AVG(ord.UnitPrice) AS UnitPrice,

AVG(ord.TaxRate) AS TaxRate,

SUM(ord.Quantity \* ord.UnitPrice) AS TotalBeforeTax,

SUM(ord.Quantity \* ord.UnitPrice \* (1 + ord.TaxRate/100)) AS TotalAfterTax

FROM dbo.Orders\_Stage ord

JOIN dbo.Customers\_Preload cu

ON ord.CustomerName = cu.CustomerName

JOIN dbo.Cities\_Preload ci

ON ord.CityName = ci.CityName

AND ord.StateProvinceName = ci.StateProvName

AND ord.CountryName = ci.CountryName

JOIN dbo.Products\_Preload pr

ON ord.StockItemName = pr.ProductName

JOIN dbo.SalesPeople\_Preload sp

ON ord.LogonName = sp.LogonName;

END;

Here are a few things to note:

* We use the staged data to look up our surrogate keys by business key
* For each **measure**, we need to decide how to **aggregate our data**. In a real-world scenario, this would be decided by consulting with your BA and/or SME.

# Load

## Load – Dimensions

Compared to our transform steps, the load step is quite simple.

For our dimensions, we are going to need to update any records that have changed. Rather than attempting to determine which columns have changed, we can accomplish this with a delete and insert. The process is the same for both Type 1 and Type 2 SCDs, so we will walk through our Customers load as an example:

CREATE PROCEDURE dbo.Customers\_Load

AS

BEGIN;

SET NOCOUNT ON;

SET XACT\_ABORT ON;

BEGIN TRANSACTION;

DELETE cu

FROM dbo.DimCustomers cu

JOIN dbo.Customers\_Preload pl

ON cu.CustomerKey = pl.CustomerKey;

INSERT INTO dbo.DimCustomers /\* Columns excluded for brevity \*/

SELECT \* /\* Columns excluded for brevity \*/

FROM dbo.Customers\_Preload;

COMMIT TRANSACTION;

END;

Here are a few things worth noting:

* We are executing our update as a delete and insert.
* We are using **surrogate keys**, which we collected/created in the preload table.

## Load – Facts

Since individual fact records are not dependent on each other, loading a fact table is usually a simple insert. However, it is important to note that because we have decided to add foreign keys to our fact table, we must wait until all of our dimensions are finished loading before we can load our fact.

Let's look at the code to do this:

CREATE PROCEDURE dbo.Orders\_Load

AS

BEGIN;

SET NOCOUNT ON;

SET XACT\_ABORT ON;

INSERT INTO dbo.FactOrders /\* Columns excluded for brevity \*/

SELECT \* /\* Columns excluded for brevity \*/

FROM dbo.Orders\_Preload;

END;

# Homework

## Understand

1. Create the SalesPeople transform. Remember that we decided to create it as a Type 1 SCD.
2. Create the Products transform. Remember that we decided to create it as a Type 2 SCD.
3. Create the load for DimSalesPeople.
4. Create the load for DimProducts.
5. Test your code by executing