Practical ETL 1

# Planning

## Example Model

Before we begin loading our example model, let’s review the model we created over the last few classes. We created a fact table to store order details and the five dimensions it is dependent on. We also created a procedure to load out date dimension. We can recreate this with the following DDL:

CREATE TABLE dbo.DimCities(

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\_DimCities PRIMARY KEY CLUSTERED ( CityKey )

);

CREATE TABLE dbo.DimCustomers(

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\_DimCustomers PRIMARY KEY CLUSTERED ( CustomerKey )

);

CREATE TABLE dbo.DimProducts(

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\_DimProducts PRIMARY KEY CLUSTERED ( ProductKey )

);

CREATE TABLE dbo.DimSalesPeople(

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\_DimSalesPeople PRIMARY KEY CLUSTERED (SalespersonKey )

);

CREATE TABLE dbo.DimDate(

DateKey INT NOT NULL,

DateValue DATE NOT NULL,

Year SMALLINT NOT NULL,

Month TINYINT NOT NULL,

Day TINYINT NOT NULL,

Quarter TINYINT NOT NULL,

StartOfMonth DATE NOT NULL,

EndOfMonth DATE NOT NULL,

MonthName VARCHAR(9) NOT NULL,

DayOfWeekName VARCHAR(9) NOT NULL,

CONSTRAINT PK\_DimDate PRIMARY KEY CLUSTERED ( DateKey )

);

CREATE TABLE dbo.FactOrders(

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,

CONSTRAINT FK\_FactOrders\_DimCities FOREIGN KEY(CityKey) REFERENCES dbo.DimCities (CityKey),

CONSTRAINT FK\_FactOrders\_DimCustomers FOREIGN KEY(CustomerKey) REFERENCES dbo.DimCustomers (CustomerKey),

CONSTRAINT FK\_FactOrders\_DimDate FOREIGN KEY(DateKey) REFERENCES dbo.DimDate (DateKey),

CONSTRAINT FK\_FactOrders\_DimProducts FOREIGN KEY(ProductKey) REFERENCES dbo.DimProducts (ProductKey),

CONSTRAINT FK\_FactOrders\_DimSalesPeople FOREIGN KEY(SalespersonKey) REFERENCES dbo.DimSalesPeople (SalespersonKey)

);

GO

CREATE PROCEDURE dbo.DimDate\_Load

@DateValue DATE

AS

BEGIN;

INSERT INTO dbo.DimDate

SELECT CAST( YEAR(@DateValue) \* 10000 + MONTH(@DateValue) \* 100 + DAY(@DateValue) AS INT),

@DateValue,

YEAR(@DateValue),

MONTH(@DateValue),

DAY(@DateValue),

DATEPART(qq,@DateValue),

DATEADD(DAY,1,EOMONTH(@DateValue,-1)),

EOMONTH(@DateValue),

DATENAME(mm,@DateValue),

DATENAME(dw,@DateValue);

END

## Processes

Before we create our ETL, we should plan out what modules will be required.

As we know, ETL processes usually include the following processes: Initialize; Extract; Transform; Validate; Load; Purge; and Archive. Our example is simple enough to not require extra validation and we will not be purging our archiving our data. Let’s review the remaining steps to see what might be required.

### Initialize

Initialize steps prepare the environment for processing. For our simple example, we will execute the stored procedure we created to load the date dimension to ensure that the table is up to date.

We will create a single step called **Initialize**.

### Extract

Typically we create one extract step per source data set. In our example, we will be require the following source data sets from WideWorldImporters:

* Customers – Query that joins Customers, CustomerCategories, Cities, StateProvinces, and Countries.
* Products – Query that joins StockItems and Colours
* Salespeople – Query of People where IsSalesperson is 1
* Orders – Query that joins Orders, OrderLines, Customers, and People

We will create an extract for each of these four sources: **Customers\_Extract; Products\_Extract; Salespeople\_Extract;** and **Orders\_Extract**. We will also create stage tables for each of these extracts.

Note that we have five tables that need to be loaded (excluding DimDate) but we only have four sources. Our customer list extract will be able to satisfy both the DimCustomers and DimCities tables.

### Transform

Often a transform step is created for each source-target relationship. Alternatively, if we combine transform steps to consolidate integration logic, we may end up with one per target instead.

In our example, most of our source-target relationships are 1:1, with two exceptions:

1. Our customer extract will supply data for both DimCities and DimCustomers
2. Our FactOrders table will need data from both the orders extract and the customers extract (to get city data)

Our first case is an example of one source feeding multiple targets, so we will still create separate transform steps for that. In our second case we will be integrating data from two sources. It is certainly possible to handle this as two separate steps, but our case will be very simple, so we will combine the two to save effort.

This leaves us with five transform steps: **Customers\_Transform; Cities\_Transform; Products\_Transform; Salespeople\_Transform;** and **Orders\_Transform**. All of our steps will be dependent on the extract of the same name, except Cities\_Transform, which is dependent on the customers extract, and Orders\_Transform, which is dependent on the customers and orders extracts.

We will also create stage tables for each of these transformations to write to.

### Load

One load step is typically created for each target table, and that’s what we will be doing. We will create **Customers\_Load; Cities\_Load; Products\_Load; Salespeople\_Load;** and **Orders\_Load**.

## Process Diagram

Even our simple example is already getting a little complicated. When creating a new ETL process, it is a good idea to diagram it to clarify all the required steps and their dependencies.

Let’s have a look:



We will need to create all of the processes and tables highlighted in green. The tables/sources listed in grey already exist.

Let’s review the important details:

* One initialize step for the entire process
* Four extracts, all working from WideWorldImporters, are all dependent on the initialize step.
* Each extract writes into a stage table.
* Five transform steps. Four of the steps have straightforward dependencies, working from a single stage table each. The orders transformation requires data from both orders and customers, so it is dependent on more than one extract step.
* Each transformation writes into a stage table, which we have called “preload” to help differentiate between our different stage tables.
* Five load steps. The four loads that populate dimension tables have a simple dependency on their transform step. Since the fact table has a foreign key reference to each of our dimension tables, it is dependent on each of the dimension loads as well as its transform step.

# Developing ETL

## Extract and Stage

Since our Initialize step is really just setting up our date dimension, we are effectively done that step. Let’s start by creating our first extract process. As we do that, we will also need to set up a stage table to write into.

Let’s start with the Customers extract.

Reviewing the requirements, we know that this extract will be getting customer information, along with their address information. We will be using this extract to serve DimCustomers and DimCities.

We will need to write a query to get data from the following tables and columns:

**From Sales.Customer**

* CustomerName

**From Sales.CustomerCategory**

* CustomerCategoryName

**From Application.Cities**

* CityName (for both Postal and Delivery)

**From Application.StateProvinces**

* StateProvinceCode (for both Postal and Delivery)
* StateProvinceName (for both Postal and Delivery)

**From Application.Countries**

* CountryName (for both Postal and Delivery)
* FormalName (for both Postal and Delivery)

Note that we are not retrieving the IDs for any of these tables. In our ETL, we will be dealing with business keys, not the surrogate keys for our source.

### Customers Stage

To build our stage table, we will create the columns needed to store the results of our query. The structure and naming of the table should be very closely tied to our source.

CREATE TABLE dbo.Customers\_Stage (

CustomerName NVARCHAR(100),

CustomerCategoryName NVARCHAR(50),

DeliveryCityName NVARCHAR(50),

DeliveryStateProvinceCode NVARCHAR(5),

DeliveryStateProvinceName NVARCHAR(50),

DeliveryCountryName NVARCHAR(50),

DeliveryFormalName NVARCHAR(60),

PostalCityName NVARCHAR(50),

PostalStateProvinceCode NVARCHAR(5),

PostalStateProvinceName NVARCHAR(50),

PostalCountryName NVARCHAR(50),

PostalFormalName NVARCHAR(60)

);

Note that we do not have a surrogate key, a primary key, or any kinds of indexing. When we read from a stage table, we read the entire table, so there’s no need for us to try to perform seeks instead of scans.

We have also created a very permissive structure, with no constraints of any kind (including NOT NULL). It is preferable to get as much data as possible staged, then deal with cleaning or reporting bad data later in the ETL.

**Best Practice**

In an extract step, prioritize loading all data over validating data integrity. It is easier to validate, clean, and report on bad data after it has been loaded into the stage tables.

### Customers Extract

We are going to create our extract as a stored procedure. Let’s create this procedure and then walk through the important elements:

CREATE PROCEDURE dbo.Customers\_Extract

AS

BEGIN;

SET NOCOUNT ON;

SET XACT\_ABORT ON;

DECLARE @RowCt INT;

TRUNCATE TABLE dbo.Customers\_Stage;

WITH CityDetails AS (

SELECT ci.CityID,

ci.CityName,

sp.StateProvinceCode,

sp.StateProvinceName,

co.CountryName,

co.FormalName

FROM WideWorldImporters.Application.Cities ci

LEFT JOIN WideWorldImporters.Application.StateProvinces sp

ON ci.StateProvinceID = sp.StateProvinceID

LEFT JOIN WideWorldImporters.Application.Countries co

ON sp.CountryID = co.CountryID )

INSERT INTO dbo.Customers\_Stage (

CustomerName,

CustomerCategoryName,

DeliveryCityName,

DeliveryStateProvinceCode,

DeliveryStateProvinceName,

DeliveryCountryName,

DeliveryFormalName,

PostalCityName,

PostalStateProvinceCode,

PostalStateProvinceName,

PostalCountryName,

PostalFormalName )

SELECT cust.CustomerName,

cat.CustomerCategoryName,

dc.CityName,

dc.StateProvinceCode,

dc.StateProvinceName,

dc.CountryName,

dc.FormalName,

pc.CityName,

pc.StateProvinceCode,

pc.StateProvinceName,

pc.CountryName,

pc.FormalName

FROM WideWorldImporters.Sales.Customers cust

LEFT JOIN WideWorldImporters.Sales.CustomerCategories cat

ON cust.CustomerCategoryID = cat.CustomerCategoryID

LEFT JOIN CityDetails dc

ON cust.DeliveryCityID = dc.CityID

LEFT JOIN CityDetails pc

ON cust.PostalCityID = pc.CityID;

SET @RowCt = @@ROWCOUNT;

IF @RowCt = 0

BEGIN;

THROW 50001, 'No records found. Check with source system.', 1;

END;

END;

Here are a few things worth noting:

* As we know, it is a good policy to always set NOCOUNT and XACT\_ABORT in our stored procedures, so we have done that here.
* We are truncating our table instead of deleting. Stage tables are excellent candidates for truncate: we don’t need a WHERE clause; there is no identity; and there are no foreign keys.
* We are using the city details twice – once for delivery city and once for postal city. Rather than join all of these tables twice, we are building up the details in a CTE and using the results twice. The result is the same, but it makes developing and maintaining the code simpler.
* We are using outer joins instead of inner joins. All of the FKs we are joining across are not nullable, so we would have been fine with inner joins, but this provides us with some future-proofing.
* We have provided some very simple validation, throwing an error if no records are loaded. This is just an example of validation that might be done. Depending on your source, it may be permissible for no records to be found.

# Homework

## Understand

1. Create the Products extract and stage table. This will require the following:
   1. Warehouse.StockItems.StockItemName
   2. Warehouse.StockItems.Brand
   3. Warehouse.StockItems.Size
   4. Warehouse.Colors.ColorName
2. Create the SalesPeople extract and stage table. All required details are in the Application.People table. Only records where “IsSalesperson” = 1 should be included. We will require the following columns:
   1. FullName
   2. PreferredName
   3. LogonName
   4. PhoneNumber
   5. FaxNumber
   6. EmailAddress
3. Create the Orders extract and stage table. This query will use Sales.Orders as the main table. It should accept @OrderDate as a parameter, and only select records that match that date. It will require the following:
   1. Sales.Orders.OrderDate
   2. Sales.OrderLines.Quantity
   3. Sales.OrderLines.UnitPrice
   4. Sales.OrderLines.TaxRate
   5. Sales.Customers.CustomerName
   6. Application.Cities.CityName (based on customer delivery address)
   7. Application.StateProvinces.StateProvinceName (based on customer delivery address)
   8. Application.Countries.CountryName (based on customer delivery address)
   9. Warehouse.StockItems.StockItemName
   10. Application.People.LogonName (based on SalesPersonID)
4. Test your extracts by executing each one. When testing the orders extract, you can use '2013-01-01' as the date.