TIME XTENDER
DATA WAREHOUSE
AUTOMATION IN ACTION

July 18, 2016
Copyright
© 2016 TimeXtender A/S. All Rights Reserved.

Trademarks
Microsoft®, Windows® and other names of Microsoft products are either registered trade-marks or trademarks of Microsoft Corporation in the United States and/or other countries. All other product names mentioned in this documentation may be trademarks or registered trademarks of their respective companies.
Contents

Introduction .................................................................................................................................................. 6
About this book.............................................................................................................................................. 6
Who this book is for ..................................................................................................................................... 6
What you need to get started .......................................................................................................................... 6
Introducing TX DWA .................................................................................................................................. 6

Core Concepts .............................................................................................................................................. 7

Data Warehouse Design ................................................................................................................................. 7
  Dimensional Model ....................................................................................................................................... 7
  Dimensions .................................................................................................................................................... 7
  Fact Tables ..................................................................................................................................................... 12

Agile DWH Design Process .......................................................................................................................... 13
  Step 1: Gain an Understanding of the Client’s Business .............................................................................. 13
  Step 2: Establish the Client’s Objectives for a Data Warehouse ................................................................. 13
  Step 3: DWH Development Sprints ........................................................................................................ 14
  Step 4: Final Project Review, Validation, Completion ................................................................................. 16

Data Warehouse Implementation .................................................................................................................. 16
Prerequisites .................................................................................................................................................. 16

Adventure Works DWH Project ..................................................................................................................... 16
  Background .................................................................................................................................................. 16
  Adventure Works Transaction Processing Database Design for Sales ...................................................... 17
  Adventure Works Transaction Processing Sales Database Design Diagram ........................................... 17
  Sales Sprint 1 User Story ............................................................................................................................ 18
  Sprint 1 DWH Model .................................................................................................................................. 18
  Data Warehouse Architecture ..................................................................................................................... 19

TX DWA Interface Overview ......................................................................................................................... 20
  The Interface ................................................................................................................................................ 20
  Navigating the Object Tree ........................................................................................................................ 21

Creating a Project ....................................................................................................................................... 22
  Task: Create the Project ............................................................................................................................ 22

The Data Warehouse .................................................................................................................................. 23
  Task: Create the Data Warehouse ............................................................................................................. 23
Create DWH Table Structure .......................................................................................................................... 24
Deploying ...................................................................................................................................................... 28
Introduction

About this book
The main objective of this book is to help anybody getting started and being successful using TX DWA. The intention is to let business scenarios guide you through the steps needed to build a basic Data Warehouse and Business Intelligence (DWH/BI) solution. We will combine fundamental DWH/BI concepts and best practices with hands-on exercises using the TX DWA platform and introduce you to the important features available in the product. Every major step in a basic DWH/BI solution will be covered ranging from connecting to data sources, preparing the data model and to setting up the final presentation layer.

Working through this book should demonstrate the power of TX DWA and enable you to apply it in a real DWH/BI project. Ideally, you will be proficient in performing the common operations and hopefully enough knowledge to help you go beyond simple and deliver on more advanced requirements.

Who this book is for
The target audience for this book are those who wants practical experience with TX DWA and how to build DWH/BI solutions. You may be someone who joined an organization that already has TX DWA and want help solving specific challenges. You may be interested in becoming a Data Warehouse Automation (DWA) professional and want practical experience. Or you may have requested a TX DWA trial to experience all the benefits for yourself before buying. We believe all of these audiences will find value in this book.

We do not expect you to have in-depth knowledge of developing DWH/BI solutions to understand this book. Although, you will be challenged if you have no prior exposure to databases at all.

Typical TX DWA user roles:
1. DW/BI Developers
2. IT Professionals
3. Business Analysts

What you need to get started
You need a computer running the Windows operating system, SQL Server with Analysis Services and TX DWA. You can download a trial here: http://go.timextender.com/tx-dwa-trial

Introducing TX DWA
TimeXtender’s TX DWA is the world’s leading Data Warehouse Automation (DWA) platform. With TX DWA, you build a complete Data Warehouse and Business Intelligence (DWH/BI) solution in a drag-and-drop environment and let the software generate the code.

TX DWA automates the entire cycle of implementing and maintaining a DWH/BI solution. Allowing for dramatic increases in productivity, speed and agility. It lowers the barriers for new developers to take over existing projects and provides a comforting safety net with powerful version controls in a team based development environment. Security and documentation features ensures that your business can remain in compliance, while continuously aligning the DWH/BI solution to the ever-changing needs of the business.

The platform offers proven benefits including:

- Delivers business value within weeks
- Builds the data warehouse from A-Z including OLAP cubes for ease of reporting and analysis
- Finally, a practical way to do Agile BI & DWH development
- Automated documentation
- Data lineage and history
- Implementation team consists of one or two people rather than 10 to 20 people needed for doing a data warehouse using traditional methods
- Respond to new and changing requirements immediately
- Greatly reduces development risk
- Up to an order of magnitude less time and expense to build and maintain your data warehouse
- Extremely quick return on investment
- Extremely low cost of ownership
- Qlik modeler and script generator

Core Concepts

Data Warehouse Design
This section gives an overview of the primary modern data warehouse design concepts used by the TimeXtender team to implement a Data Warehouse.

Dimensional Model
This concept describes a longstanding technique for presenting analytical data in a format that is easily understandable and delivers fast query performance. This model builds on a single business process represented by a transactional fact table in the center surrounded by the various dimensional tables. This model can be represented visually in what’s called a Star Schema seen here.

Additional crucial details of the dimensional model are described in the sections below.

Dimensions
The Dimensions are the tables that include the descriptive, contextual information. For example, the Customer dimension is the table that includes all the details recorded for each customer. Each detail is represented as a column and is known as an attribute. So in this case, the customer table would include attributes such as the Name, Street Address, City, State, Region, Country, Contact etc. Dimension tables typically have fewer rows but more columns than fact tables. The dimension attributes are the primary source of query constraints, groupings, and report labels. You should avoid using operational codes in dimension tables to improve the intelligibility of reports.

Nulls
Typically, nulls should be replaced by a descriptive string in Dimension tables. In TX DWA you can use a fixed transformation to replace empty fields with “NA” or “Unknown”. This is because different databases handle constraining on nulls inconsistently.

Surrogate Keys vs. Natural Keys
Keys assign each entry, or row, in the dimension a unique code. This code is typically a number or sometimes a set of numbers and letters in order to uniquely identify a specific row or entry. In an operational database, or our client’s
source data, every table entry will be assigned unique number. Customers will have a customer number and purchase orders will have a purchase order number etc. This is known in TX DWA as the Natural Key. However, when building a Data Warehouse, it is important that an additional key is assigned to each row we bring into the DWH. This “Surrogate key” is created by TX DWA automatically and should not be used outside of the Data Warehouse. This way, if the client decides to change how they assign keys, or they acquire a company for example, the new or modified keys will not affect the data warehouse because the surrogate key acts independently of the natural key.

Conformed Dimensions

Another key concept of the Dimensional Model is Conformed Dimensions. This is the idea that all the Dimensions in the Data Warehouse conform to specific structure so they may be reused across multiple fact tables. This not only reduces the size of the database, it also improves the simplicity and query time as well. For Example, a conformed customer dimension would be used to identify customers in the sales fact table and the Invoice fact table as well.

Denormalized Flattened Dimensions

A typical operational database does not use the Dimensional model because it is not efficient for continuous data entry. Operational databases use what’s known as Third Normal Form, or 3NF. This model attempts to reduce duplication as much as possible by “snowflaking” dimensions out into a large number of tables. For example, product items in the product table may be grouped by brands and categories. So 3NF would have a Product table, Product Brand table, and Product Category Table. Then the category table may have a subcategory table from there and so on.

However, this “snowflaking” of tables is extremely inefficient for analysis simply due to the large number of joins that need to be done every time a query is run. Instead, the Dimensional Model uses what’s called Denormalized Flattened Dimensions. This is where all the related table attributes are brought together into a single table and hierarchical descriptive information is stored redundantly. Using the previous example this would take the brand, category, and subcategory attributes and bring them all into the Product table. This can be seen in the example table here.

<table>
<thead>
<tr>
<th>Product Key</th>
<th>Product Description</th>
<th>Brand Name</th>
<th>Category Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PowerAll 20 oz</td>
<td>PowerClean</td>
<td>All Purpose Cleaner</td>
</tr>
<tr>
<td>2</td>
<td>PowerAll 32 oz</td>
<td>PowerClean</td>
<td>All Purpose Cleaner</td>
</tr>
<tr>
<td>3</td>
<td>PowerAll 48 oz</td>
<td>PowerClean</td>
<td>All Purpose Cleaner</td>
</tr>
<tr>
<td>4</td>
<td>PowerAll 64 oz</td>
<td>PowerClean</td>
<td>All Purpose Cleaner</td>
</tr>
<tr>
<td>5</td>
<td>ZipAll 20 oz</td>
<td>Zippy</td>
<td>All Purpose Cleaner</td>
</tr>
<tr>
<td>6</td>
<td>ZipAll 32 oz</td>
<td>Zippy</td>
<td>All Purpose Cleaner</td>
</tr>
<tr>
<td>7</td>
<td>ZipAll 48 oz</td>
<td>Zippy</td>
<td>All Purpose Cleaner</td>
</tr>
<tr>
<td>8</td>
<td>Shiny 20 oz</td>
<td>Clean Fast</td>
<td>Glass Cleaner</td>
</tr>
<tr>
<td>9</td>
<td>Shiny 32 oz</td>
<td>Clean Fast</td>
<td>Glass Cleaner</td>
</tr>
<tr>
<td>10</td>
<td>ZipGlass 20 oz</td>
<td>Zippy</td>
<td>Glass Cleaner</td>
</tr>
<tr>
<td>11</td>
<td>ZipGlass 32 oz</td>
<td>Zippy</td>
<td>Glass Cleaner</td>
</tr>
</tbody>
</table>

SUPERTYPE / SUBTYPE

A generalization hierarchy is a form of abstraction that specifies two or more entities that share common attributes can be generalized into a higher level entity called a supertype. The lower-level entities become the subtypes, or categories, to the supertype.

For example, Cars and trucks represent categories of the Vehicle entity. In this example, Vehicle would be the supertype, while Cars, Trucks and Motorcycles are the subtypes.

The supertype entity contains attributes that are common to all of its subtypes. All vehicles share the same attributes such as color, weight, price etc. However, Cars have attributes that trucks and motorcycles don’t have and vice versa.

However, using this design to keep history is very difficult, particularly when using Type 2 slowly changing dimensions. To overcome the limitations for keeping history that the transaction processing systems have, the denormalization technique takes all of the attributes from the transaction processing supertype design and puts them all into a single table.
Slowly Changing Dimensions
While fact tables are changing constantly recording entries every minute, dimensions are seen as somewhat static. However, dimensions are not fully static, people change addresses, phone numbers are updated, and product descriptions are modified and so on. So it is important to address these changes with the business users early on in the development process. The business user must decide if they want to record these changes when they happen. The ways we handle a slowly changing dimension (SCD) are broken down into multiple “types”. It is important to note that SCD types are specific to the attributes themselves, not the entire Table. For example, address could be a type 2 SCD while the Birth Date would be Type 1. In this guide we will review Type 0, Type 1, and Type 2. There are a number of other SCD types, most of which are combinations of these three types. However, they are outside the scope of this guide and will not be addressed.

TYPE 0
This is not really a slowly changing dimension because in this type of dimension the attributes never change. Examples of attributes in this type of dimension are Birth Date and Original Credit Score.

TYPE 1
A type 1 SCD does support updating of values, however, it overwrites the values and does not track the change. In other words, once a change is made, the old value is lost forever.

TYPE 2
A type 2 SCD tracks historical value changes in dimensions through a number of ways. First, two additional columns are created in the table, a “From Date and a “To Date”. Second, changes are tracked by creating an additional row in the table with the unchanged values remaining the same, and the changed values with the updated data. The old row’s “To Date” is updated to the date prior to the change and the new rows “From Date” is update with the date the change was made. For Example, John Doe moves from Oregon to Idaho. A Type 2 SCD would create a new instance of John Doe in the table so the table now has two John Doe’s with identical information SS number and Customer ID etc. The first instance would have Oregon as the value in the State attribute with the To Date updated to Yesterday. The second instance would have Idaho under the State attribute and a From Date of Today’s date. This way the business users can track customer address changes.

Degenerate Dimensions
A degenerate dimension has no related dimension table. This is typically a purchase order number or transaction number. It does not have its own dimension because all the details of the dimension are already established within the fact table. When you group a query by a degenerate dimension such as a PO number, the results displayed will be all the details of the PO.

Bridge Tables to Resolve Many-to-Many Relationships
There are a number of attributes that could potentially have any number of values, such as, customer contacts, industry groups or patient diagnosis. In this case, you can use a bridge table that contains all of the foreign keys of the Company
table and contains all of the foreign keys from the Industry table. The fact table would then reference the IndustryGroupKey to identify combination of companies that belong to a particular IndustryGroup. In the following example it is possible to associate any number of industries with a company. It is now simple to qualify queries for a specific industry. Since there is one column called “IndustryName” is also simple to group results by industry.

**Outrigger Dimensions**

When a dimension table contains a reference to another dimension it is known as an outrigger dimension. In the following example, we see a fact table with a customer dimension. Linked to the customer dimension is an outrigger with a lot of demographic-based attributes pertaining to the county that the customer lives in.

This is considered an acceptable circumstance for leveraging a snowflake/outrigger design because there are so many county demographic attributes and they are all at such a higher granularity than the rest of the attributes in the customer dimension. Another justification is that the data for the county demographic attributes is loaded at a different time from the rest of the customer attributes.

If necessary, a view can be placed on top of the dimension and outrigger tables so that they appear to the user as a single dimension via the client application.

**Role Playing Dimensions**

When a single dimension simultaneously appears several times in the same fact table, it is known as a Role Playing Dimension. This is done with a single physical table presented as separately labeled views. For instance, a fact table can have several dates, each of which is represented by a foreign key to the date dimension. It is essential that each foreign key refers to a separate view of the date dimension so that the references are independent. These separate dimension views (with unique attribute column names) are called “Roles”.

The underlying dimension will exist in the database as a single physical table, but each of the roles should be presented to the data access tools in a separately labeled relational view.
The example shown here shows a single date dimension split into two views, Order Date and Requested Ship Date. Now the fact table can refer to two separate foreign keys to indicate the proper date type.

**ADVANCED ROLE PLAYING DIMENSIONS**

Suppose that a transoceanic shipping company transports bulk goods in containers from foreign to domestic ports. The items in the containers are shipped from an original shipper to a final consignor. The trip can have multiple stops at intermediate ports. It is possible that the containers may be off-loaded from one ship to another at a port. Likewise, it is possible that one or more of the legs may be by truck rather than ship.

In this example the consignor, foreign transporter, foreign consolidator, shipper, domestic consolidator, domestic transporter, and consignee are all roles played by a master business entity dimension that contains all the possible business parties associated with a voyage. Also, the Voyage origin port, voyage destination port, leg origin port, and leg destination port are all views on the Port Dimension.

Shipping transport schemas like this one characteristically have a large number of dimensions. When all the parties to the voyage have been added, the design can swell to 15 or even 20 dimensions. This method can drastically reduce the number of tables and duplicate entries in the DWH.
Fact Tables

According to the Dimensional Model the Fact Table handles the recording and storage of the performance measurements resulting from an organization's business process events. Some examples include the Sales Order table that records all sales transactions, or the Invoice table that records all the items that have been billed to the customer, etc.

While fact tables typically have much fewer columns than dimension tables, it is not uncommon for fact tables to have a row count in the millions or sometimes billions. As seen in the example here, Fact Tables should only include foreign keys of dimensions, possibly a degenerate dimension or two, and the measures of the transaction.

Measure

Describes the measurements taken during the business process such as quantity ordered or sales amount.

Nulls

In contrast to Dimensions, nulls exist nicely in fact tables and operations typically behave as you would expect.

Grain

Each row in the fact table should represent a single event in the physical world. This level of detail is referred to as the "grain" and must remain constant throughout the table. Modern data warehouse practices call for developers to always use the finest, most atomic, grain. This way business users can always slice down to any level they like and are not limited to summary data.

Denormalized Header & Detail Facts

It is quite common in operational databases to encounter facts of differing granularity. On an order, for example, there may be a shipping charge that applies to the entire order that isn’t available at the individual product-level line item in the operational system. Allocating the parent order facts to the child line item level is critical if we want the ability to slice and dice and roll up all order facts by all dimensions, including product, which is a common requirement.

The technique for creating a new sales detail line includes adding a "Product" in the Product dimension for the Header fact that you want to allocate to the detail line item. This "Product" does not exist in the source product table, you have to manually create it. These artificial "Products" typically assigned minus surrogate keys. Examples include: -11 for Freight fact from header; -12 for Tax fact from header; etc.

Consolidated Transaction Fact Table

In a typical dimensionally modeled data warehouse you have multiple fact tables at an atomic grain that all connect to the same conformed dimensions. However, the TX Data Warehouse Master Model brings the goal of simplicity to the next level. Using the denormalization methodology addressed earlier, the Master Model groups all the transactional fact tables into one large table known as the Consolidated Transaction Fact Table. By assigning each fact table a “FactType” the facts can be consolidated and then split out into separate views for the end user. This not only improves simplicity of
the data warehouse but also drastically reduces the number of relations needed. You can see how this table and the other methodologies mentioned in this section come together in the entity relationship diagram of the TX Master Model shown here.

Agile DWH Design Process
The TimeXtender team utilizes a methodology known as Agile for its primary data warehouse development process. Agile is a software development methodology that focuses customer collaboration to develop smaller pieces of working software in short sprints, building the project until it is complete. With this method, the client not only gets a very short return on investment by receiving a working product early on. But the developers also get the constant feedback of the client, their concerns, likes, and dislikes, as the project is developed. References to additional resources on agile methodologies is listed in

The process used by the TimeXtender team is broken down into various steps which are described below.

Step 1: Gain an Understanding of the Client’s Business
Through a combination of outside research and conversations with the client, develop an understanding of the WHO, WHAT, WHEN, WHERE, and WHY of the business. This should allow you to start making a list the client’s business processes e.g. receiving purchase order, manufacturing goods, shipping goods, sending invoices etc. You can use this information to begin filling out an Enterprise Bus Matrix.

Enterprise Bus Matrix
This is table helps developers visualize the high level requirements of the data warehouse during the early stage of development. It includes a list of the business process and associated conformed dimensions. See the example shown here.

<table>
<thead>
<tr>
<th>BUSINESS PROCESSES</th>
<th>Item</th>
<th>Product</th>
<th>Warehouse</th>
<th>Sales</th>
<th>Promotions</th>
<th>Customer</th>
<th>Employee</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue Purchase Orders</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive Warehouse Deliveries</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warehouse Inventory</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receive Store Deliveries</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Store Inventory</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Retail Sales</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Retail Sales Forecast</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Retail Promotion Tracking</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customer Returns</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return to Vendor</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frequent Shoppers Sign-Ups</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 2: Establish the Client’s Objectives for a Data Warehouse
In this step you want to identify why the client wants a data warehouse. What key information does the client need in order to make their business more profitable? In the Agile Methodology, this vital question is answered through a list of client requirements known as Business User Stories and Key Performance Indicators (KPIs). In your initial meeting with the client, you should write down as many user stories and KPIs as possible. You should continue to acquire more user stories through future conversations with the client and development sprints. In this initial meeting you also want to acquire any documentation from the client that maps out their current operational database. This should include a list of necessary tables, related tables and associated keys.

Business User Story
A user story is a short, simple, unambiguous description of what the client wants and why he wants it written in the language and perspective of the client. They should typically follow this format:
"As a <type of user/role>, I want <feature> so that <reason>"

The primary goal of the user story is to help guide the development team to incorporate all of the necessary features in order to best serve the business users. Developers should collaborate closely with the business owners to clarify the details as the project develops.

EXAMPLES
“As a Sales Manager, I want to know what sales reps have the highest sales per month so that I can establish an incentive program”

“As the CEO, I want to see our profitability over time so that I can ensure the company is growing steadily”

“As a Production Manager, I want to see returns per month by product and reason so I can determine what product might be failing in the field”.

“As the Marketing Manager, I want to see what forms of marketing lead to the most sales so I can allocate budget more effectively”

Key Performance Indicators
A Key Performance Indicator (KPI) is a measurement used to evaluate how effectively a business is reaching its objectives. It is very important that the business identifies KPIs for the C level officers as well as each of the departments or divisions in the organization.

KPI EXAMPLES
- Percent of revenue growth over time
- Average profit generated by a specific product or service
- The percentage of customer that do not renew their subscription
- Average time it takes a product to get from purchase to the customers door

Step 3: DWH Development Sprints
During the initial meeting with the client, you also want to agree upon what measures should be mapped in the DWH first and when this first “sprint” will be complete. This is typically one single business process, consisting of one fact table and a set of associated dimensions. By the end of the sprint, usually in 2-4 weeks, you should have a DWH for the client to use complete with all the data for this single business process. Future sprints will map additional business processes and should take less and less time because your conformed dimensions will be reused. In each successive sprint you and the client will agree on the process to be mapped and the developer should have another process mapped in 1-2 weeks.

These sprints should be broken down into multiple smaller steps.

Design a High Level Dimensional Model
This process will help you develop the model. You should map this out in a bubble chart to assist the development team in visualizing the model.

SELECT THE BUSINESS PROCESS
Along with the client, you agree on which business process to map into the DWH for this sprint. e.g. receiving purchase orders.
DECLARE THE GRAIN
The grain describes what a single row of the fact table represents. e.g. one row equals one purchase order line item. This important to decide early on because it will impact many future development steps.

IDENTIFY THE DIMENSIONS
Make a list of the dimensions or “descriptive context” of the processes. e.g. Purchase order date, Customer, Product, Sales Person etc.

IDENTIFY THE FACTS
The question to be answered in this step is, what details are being recorded as a result of this step? e.g. Purchase order quantity and price.

Design a Detailed Dimensional Model
Using the principles described above, the development team expands the dimensional design with associated attributes and measures. You should begin documenting all the details of your model in a spreadsheet including each table with associated attributes and measures.

IDENTIFY DIMENSIONAL ATTRIBUTES
Review the business user stories and identify which attributes are needed for each dimension.

IDENTIFY THE FACT TABLE MEASURES
This is done by reviewing the business user stories once again to determine what measures and calculated measures need to be included in the fact tables.

IDENTIFY THE SLOWLY CHANGING DIMENSIONS
For each dimensional attribute, identify with the business users which attributes need history tracked and which attributes do not require recorded historical changes.

Translate Business User Stories into Technical User Stories
Once the Dimensional model is mapped out down to the attributes, you can begin translating the Business User Stories into Technical User Stories. The Technical User Stories are the tasks that must be carried out by the developer in order to give the user the result expressed in their business user story. Gather all the business user stories that relate to the business process identified for current sprint. Then, for every business user story, identify the step or steps that need to be taken to give the user the desired result. Often times, the technical user stories surface during a brainstorming session of the development team.

Build out Model in TX DWA
This step will take the majority of man hours however, the details of this step are explained further in the practical section.

Validation
TEST THE MODEL
Once the Model has been built, you must test the DWH to ensure it will give the user their required results. Build out some simple reports in Excel to ensure you can get retrieve the data desired by the user.

REVIEW AND TEST MODEL WITH BUSINESS USER
Present your completed work for the current sprint to the business user and have them run their reporting software against the database to ensure they are getting the results they expect.

Track and Resolve Issues
Record all issues identified in the Validation process and establish steps to correct the issues.
Deploy Completed Model to Production Server
Once Validation is complete and the user has approved all the changes, you can deploy the model onto the client’s production server.

Step 4: Final Project Review, Validation, Completion
Once you have repeated step 3 for all the development sprints and the entire DHW has been mapped you move on to project validation. Return to your original list of user stories to ensure they have all been addressed. Not all user stories are always completed, it may be that the client may decide some are such a low priority that extra time should not be spent on them. However, it is important you ensure all User Stories have at least been addressed by both the development team and the functionality either approved or discarded by the client.

In this step the development team must also address any and all remaining bugs that may be present in the DWH.

Once all sprints are complete and all bugs have been addressed, the project is considered complete.

Data Warehouse Implementation
This chapter serve as step-by-step instructions on how to create a DWH with TX DWA by using some of the previously discussed concepts. When you finish all the tasks and exercises in this guide, you should be able to do “real” work using TX DWA. It is obviously not feasible to cover every situation and technique and it is also important to note that “best practices” depends on business requirements. However, this guide should allow a developer to accomplish at least the basic tasks and hopefully provide enough foundation to start working towards meeting more complex requirements.

The business case will be based on a fictional company named Adventure Works. You will assume the role as a developer that is hired to build a DWH for Adventure Works. The case will attempt to incorporate elements from agile methodologies to make the case more realistic.

Prerequisites
In order to work through the steps, you will need:

- The AdventureWorks2012 sample database: Appendix 1: Download and Install Sample Data
- The TemplateDWH database: Appendix 2: Create TemplateDWH Script

Adventure Works DWH Project

Background
The Adventure Works corporations sell bicycles and related items retail directly to individual customers and also to resellers. The Adventure Works sales order table contains Internet sales to individuals as well as reseller sales. Internet sales is only available for direct to individuals, i.e. retail, and includes direct shipping to customer addresses.

Sales to resellers are based on purchase orders for bicycles and accessories. Purchase orders are fulfilled and goods are sent to the resellers. The billing address for a reseller may be different than the shipping address. All orders have a due date which management tracks rigorously.

The SalesOrderHeader table does not have a separate foreign key for individual customers and resellers. Instead, it only has a CustomerID. The CustomerID ca be for either an individual customer or a reseller. We will learn how to handle this later on.
Since the Adventure Works transaction processing database design is very complex it is very difficult to do the kind of reporting and management needs. Even when reports are developed, they often put a heavy load on the database that slows down the Internet sales processing.

**Adventure Works Transaction Processing Database Design for Sales**

As mentioned earlier, the Adventure Works transaction processing database is very complex. There are some design decisions that are not intuitive. Some of these complexities include:

1. SalesOrderHeader and SalesOrderDetail contain orders for both internet customers (Person) and resellers (Stores)
2. SalesOrderHeader contains only a CustomerID. There is no internet customer id nor a store customer id. You cannot tell from the CustomerID whether the customer is a person or a store.
3. You can infer whether the CustomerID is for a person or a store based on the OnlineOrderFlag. If OnlineOrderFlag is True, you can tell that the order is for a person, but there is no way to tell which person
4. The Customer table only contains keys for people customers, store customers, store contacts sales territory where the customer is located and the AccountNumber assigned to the customer by the accounting system. Just looking at the name of this table, you would never guess that store contact and AccountNumber will be part of the table.
5. Most non-transaction tables have the column “Name”. These columns should have a prefix to “Name” so that a person will know what kind of name it is. Examples include: ProductName, ProductCategoryName, PersonName, etc.

Fortunately, our analysts have studied this and many of these design decisions will be revealed as we go through the Agile development process. The diagram for the Adventure Works sales subject area is listed below.

As you can see, the sales business area database design is pretty complicated. In each sprint, we will begin denormalizing this design into a much more intuitive multidimensional design.

**Adventure Works Transaction Processing Sales Database Design Diagram**

Here is the transaction processing database design for the AW Sales business process.
Sales Sprint 1 User Story
As the Sales Director, I need to understand:
- What internet sales customers are buying what products?
- What are my top-selling products?
- Who are my top customers?
- Does seasonality have a major effect on product sales?
- And many more questions...

Sprint 1 DWH Model
The data warehouse database design to be used in Sprint one is shown below. Notice that it has only four tables. Three of these tables are denormalizations of the tables in the AdventureWorks database. There’s a fourth table which does not exist in the AdventureWorks database. This is a date dimension hierarchy that allows us to easily choose days, weeks, months, quarters, etc., without having to code SQL date functions. This can be a huge time saver.

In addition, you see three database views: DimOrderDate, DimDueDate, and DimShipDate. They can be many types of dates that the data warehouse. Having a single date hierarchy table with multiple views for the various roles that the date dimension can play as a modern data warehouse best practice.
Data Warehouse Architecture

A data warehouse architecture diagram shows the various components of a data warehouse where software operations will be done. It usually shows a sequence of operations from left to right or bottom of the top.

The data warehouse architecture diagram below shows the main components of the data warehouse to be: (1) the sources for the data warehouse; (2) the staging area; (3) the data warehouse; and (4) the presentation area.

A sequence of operations is:

1. The data from the source systems is copied into the staging area
2. The data is transformed in the staging area (denormalized, lookups done, etc.)
3. The transformed data is loaded from the staging area into the data warehouse database
4. The data warehouse data is consumed either through conformed data marts or OLAP databases

When using TX DWA, you should do all of the transformation in the staging area. This is much more efficient than putting logic in the load from staging to the data warehouse database.

We will cover the DWH architecture in more detail when we address each architecture component.
Let’s take a moment to get quick overview of the TX DWA interface and some helpful concepts and commands.

The Interface
Although a new project won’t have all of the parts pictured in the illustration, here’s what a project in progress will look like when first opened:
5. The upper left contains the main menu and shortcuts for opening, saving, and creating new projects.

6. At the top of the window are the tool ribbons. These can be toggled between Project tools and general tools. We will not be using these ribbons much at first – many of the commands are for more advanced features.

7. The Object Menu is a context-sensitive menu that will list common commands depending on what TX DWA object you have selected. For instance, if you select a table, it will present options to modify that table.

8. The lower section of the window contains the workspace. What the workspace displays depends on which tab is selected. Most TX DWA work occurs in the Data tab, although we will cover the uses of some of the other tabs as well.

9. In the data tab, we can see the TX DWA object tree. Each item that is visible here represents a node in this tree. These nodes represent a TX object or object group.

10. The workspace footer contains data on which project is open, and which server and project repository are in use. The circle in the lower left is green when the project is saved, and red when the project contains unsaved changes.

Navigating the Object Tree

Object groups and TX objects can have other objects grouped under them. When this happens, the object will be displayed with a “…” to its left. To show what is contained under an object or group, we can expand it by double clicking on the object, or clicking on the “…” symbol. Using this, we can see that the Data Warehouse contains multiple tables, and that a table contains multiple fields and a set of indexes:

These objects can be collapsed again by clicking on the “…” symbol, or double clicking the object.

Objects can also be opened or closed with keyboard shortcuts. The left and right arrow keys (← and →) will expand or collapse an object. By selecting an object and pressing CTRL + ← or →, the object and all of the objects it contains will all be opened or closed all at once.

An object can also be opened in a new window by selecting it and pressing CTRL + W. This can allow you to keep it front and center even while navigating away from it in the tree:
Creating a Project

A TX DWA Project contains all of the necessary parts of a Data Warehouse solution. This includes data source connections, a staging area for ETL, a data warehouse to contain report-ready data, and a cube to aid in report building. The Staging area, the Data Warehouse, and the Cube all have their own databases in SQL Server. Other TX DWA data is automatically stored in the project repository.

During this step, we can specify the way certain parts of the project will behave. When we select options at this step, the changes will be applied to all relevant objects. This values can be changed later, but will not be applied retroactively. To save time, it’s important that these configuration options are correct from the beginning.

Task: Create the Project

1. Open the TX application Menu by clicking the X button
2. Click New Project
3. Name project "AdventureWorksProject"
4. Check “Team Development”
5. Check "Show System Control Fields"
6. Click OK

Note that there are no “correct” settings. These settings will simply make the rest of the training easier to do.
The Data Warehouse

The Data Warehouse (DWH) is the central part of a BI solution. It is the destination for all data from Staging, and serves as the foundation of any additional reporting layer, data mart, or cube. It’s a good idea to create objects in the DWH first. You need to have DWH design before you start working on the Staging Area. In general, you create the data structure for the DWH and do all of the transformation in the Staging Area. Ideally, the DWH data structures are copied from the staging area and little or no manipulation is done on the DWH tables.

The DWH provides a database that contains integrated data from several source systems. This data has been transformed to a common design so that you don’t have to know the data structure of each source system. It also keeps history that enables historical analysis as well as predictive analysis. This is essential for any organization to increase its ability to compete.

Creating the Data Warehouse does two things. First, it creates a representation of the Data Warehouse in TX DWA. It also creates a database in SQL Server to hold tables, views, procedures, and the other SQL objects that make up the structure of the DWH.

Task: Create the Data Warehouse
1. Right click “Data Warehouses
2. Click “Add Data Warehouse”. The Data Warehouse dialogue box will appear.

3. For “Name”, enter “AdventureDWH”. This is the friendly name for the Data Warehouse that will appear in the TX DWA user interface.

4. In the SQL Server section, enter the server name. In this case, use “.” or “localhost”.

5. Under “Database”, enter “AW2012DWH”. This will be the name of the SQL Server database that stores DWH data and supporting objects.

6. Click “Create”. TX DWA will automatically create the new SQL Server database.

7. Click OK. The dialogue box will close.

8. Expand the Data Warehouses node. This will show a number of different objects that can be created in the Data Warehouse. Most can’t be expanded, because no objects of that type exist yet.

Create DWH Table Structure

In this section, we will create a two tables that will serve as the beginning of the Data Warehouse. One table will be a sales transaction fact table, the other will be a dimension that contains information on products. These two tables will allow an end user to analyze their sales by a variety of product attributes.

TX DWA has several methods for creating the Data Warehouse tables. In this section, we will cover two methods: importing a table from an external data source, and creating a table manually. A third method, creating a table based on a Staging table, will be covered in Mapping Staging Data to the Data Warehouse.

In addition, you will learn how to use pre-existing tables as a predefined external source for DWH tables. Your Data Architect may have created dimension tables that are to be used in all data warehouse projects. The TemplateDWH database simulates this scenario. Make sure that you restore the TemplateDWH database into your SQL Server instance.
Creating Tables in the Data Warehouse

TX DWA can bring in tables into the Data Warehouse directly from an external source. This is useful for incorporating a pre-existing data warehouse into a new TX DWA deployment, or for just copying the structure of a pre-existing database template. This training includes the template. If you have not yet installed it through SQL Server Management Studio, please see Appendix 2: Create TemplateDWH Script.

Add External SQL Connection

**TASK: ADDING AN EXTERNAL SQL CONNECTION**

1. In the data warehouses node, Right Click on the Data Warehouse name you just created.
2. Under Advanced, click on Add External SQL Connection. The “Add External SQL Connection” dialogue box will appear.
3. Under “Name”, enter “TemplateDWH”. This is the friendly name for the external connection that will appear in the TX DWA user interface.
4. In the SQL Server section, enter the server name. In this case, use “.” or “localhost”.
5. Under “Database”, enter “DWH_TEMPLATE_DB” or choose “DWH_TEMPLATE_DB” from the dropdown menu. This should be the name of the SQL Server database that we created earlier to store the template data.
6. Click OK. The dialogue box will close.
7. A new node in the object tree will appear at the bottom of the data warehouse called “External SQL Connections”.

**TASK: READING OBJECTS FROM AN EXTERNAL CONNECTION**

1. Expand the External SQL Connections node to view the new connection you just created (Click the “+” button)
2. Right click on “TemplateDWH” node
3. Click “Read objects from Data Source”
4. The “Read objects from Data Source” window will appear as the system works. When this process finishes, a new window will open on the right side of the workspace.
TASK: IMPORTING TABLES FROM AN EXTERNAL CONNECTION

At this point, the external connection to the Data Warehouse should be selected, and a new window containing a short list of objects should be open on the right side of the workspace. These are all tables and views present in the external source.

We need to bring these tables into the Data Warehouse.

1. Double-click on DimCustomer to view the fields in that table.
2. Click the box next to CustomerKey. A check-mark will appear next to the DimCustomer table name, indicating that this table has been brought into the Data Warehouse with all of the selected fields. A “*” will also appear next to the Tables node in AdventureDWH, indicating there is now an object in that group.
3. Check the rest of the fields in DimCustomer
4. Double-click on DimProduct to view its fields.
5. Check the box next to DimProduct to automatically select all of its fields.

In practice, we will always want to select one field at a time from a data source, and only choose fields we are absolutely sure we need. In this case, DimProduct has been pre-formatted for this exercise, so we can bring in all of its fields at once. Do this by checking the box to the left of the table name. Notice that all of the fields are also automatically selected.

Once you have selected the tables and columns you want to bring into the Data Warehouse from the external source, expand the Tables node under AdventureDWH. You should be able to see the tables you selected listed there.

Currently, these table names should be in red. Red names in TX DWA indicate that a change has been made to an object, but that the change has not yet been fully committed.
TASK: CONVERT EXTERNAL SOURCE TABLES TO CUSTOM TABLES

The tables we’ve brought in from the external source currently appear with grey icons. This indicates that they are not only copying the format of the table in the external source, but are going to read data from it as well. Since we are using the external source as a template, not as a fully-functioning external section of our data warehouse, we need to convert these tables to custom tables. This will retain their format, but remove the behind-the-scenes copy that TX DWA has put in place.

1. Right click on TemplateDWH_dbo_DimProduct.
2. Select “Edit Table”.
3. Under Advanced, click “Convert table to Custom table”.
4. A dialogue box will appear and prompt you to confirm your choice. Click “Yes” to continue.
5. Click “Yes” to continue. Note that the table icons turn from grey to blue. This indicates that the table is now an ordinary Data Warehouse table.
6. Repeat steps 1-4 for TemplateDWH_dbo_DimCustomer.

TASK: RENAME TABLES

Now that we have successfully imported the tables, we no longer need to know that they originally came from a data warehouse template. To make the tables easier to work with moving forward, we will remove the source prefix and give them a name that’s easier to read and type.

1. Right click on TemplateDWH_dbo_DimCustomer.
2. Select “Edit Table”.
3. A dialogue box will appear with a single text field called “Name”.
4. In the text field, rename the table to “DimCustomer”.
5. Repeat steps 1-4, and change the name of TemplateDWH_dbo_DimCustomer to just “DimCustomer”.
6. Repeat steps 1–4 and change the name of “TemplateDWH_dbo_<Table Name>” to just “Dim<Table Name>” or “Fact<Table Name>” for any other imported tables from TemplateDWH.
**TASK: MANUALLY CREATE TABLES**

It’s very common for new requirements to come arrive at short notice. In addition, we have to be able to accommodate design requirements that are not part of a given template or pre-constructed data warehouse. The primary way this is done is through manually creating tables.

**CREATE A NEW TABLE**

1. Right click on “Tables” under AdventureDWH and select “Add Table”.
2. The “Add Data Warehouse Table” dialogue box will appear. Enter the name of the new table: “FactInternetSales”.
3. If we needed to add more than one table, we would press +1, which would create the current table and re-open the “Add Data Warehouse Table” dialogue box. In this case, we do not need to add another table, so we will instead hit “OK”.

We will not add any fields to this table as we will show how to add fields to it later.

**Deploying**

In TX DWA, the changes you make will not automatically be reflected in the structure and content of the majority of the objects that you create. These changes have to be committed in an additional step. To commit changes to an object’s structure to SQL Server, that object must be deployed. To populate or re-populate an object with data, the object must be executed. Deployment will also save the project, but this can be done independently of deployment as well.

An object and all of its sub-objects can be deployed, executed, or both, as applicable, from its right click menu. The right click menu of major named objects, like the name of your project, data warehouse, or staging area will give you additional options for deployment and execution that can be very helpful. We’ll cover both options below.

**Deploy Data Warehouse Tables**

Since there is currently nothing feeding data to our new Data Warehouse tables, there is no need to execute them. For now, we will just deploy them.
Task: Deploy a Single Table

1. Right click on the DimProduct table and select “Deploy”.
2. The Deploy dialogue box will appear.
3. Click “Review Tasks & expand Tasks to see all of the deployment tasks
4. Click “Start”
5. Click “+” signs to see deployment steps
6. When Version Details box opens, Click “OK”
7. The deployment process starts. When it is finished, click “Close”
8. Products name in Data warehouse is now black instead of red

When deployment is complete, there will be three available options in the lower right: Start, to restart the deployment, View Log, to view the status of each step of the deployment, and Close, to leave the deployment window. View Log is very helpful if the deployment fails, as you can see on which step deployment failed and view the associated error message. Since our deployment was successful, click “Close”.

Note the name of the DimProduct table in the Data Warehouse is now in black instead of red. This means that the structure of the version you see in TX DWA and the committed version in SQL Server are the same.

Task: Deploy Multiple Tables

Taking the time to deploy each table individually can be prohibitively slow, especially in large, complex projects. Fortunately, TX DWA has additional options for deploying more than one table at a time.
1. Right click on the name of the data warehouse, AdventureDWH, and mouse down to “Deploy”.
2. The “Deploy and/or Execute” popup appears. The default options are already set to deploy only those tables which are new or have been modified.
3. Click “Start” to begin deployment.
4. The Version Details dialogue box will appear. Press OK to continue.
5. The Version Details dialogue box will close and the progress bar in the Deploy Modified Objects dialogue box will fill as TX DWA completes each deployment task. Note that the deployed tables will turn black to indicate that they have been successfully deployed.
6. Click “Close” to close the Deploy Modified Objects dialogue box.

**TASK: ENABLE PHYSICAL VALID TABLE**

DWH tables that are manually created or brought into the DWH from an external source need to have their table settings adjusted. These steps will help improve performance of the data warehouse.

1. Right click DimCustomer
2. Select “Table Settings F4”
3. Popup box appears
4. Click “Performance” tab at top
5. Check “Enable physical table”
6. OK
7. Repeat for all dimensions
8. Deploy
The Staging Area

The staging area is very important to the overall DWH architecture. It is a landing zone for consolidating data from multiple source systems. It is also the place where data is standardized from these multiple systems.

The staging area allows independent scheduling of data retrieval from the source system. Business logic can identify and handle “invalid” data as well as data validity rules.

In short, the staging area is where all of the operations to transform the data from source systems into an integrated form to copy (load) into the data warehouse take place.

Creating a Business Unit

A Business Unit is a TX DWA object that contains two primary sub-objects: a Staging area and a group of data source connections. These data source connections allow TX DWA to quickly copy in data from operational source systems. Once the data is brought into the Staging area through the data source connections, we perform the tasks that we learned about in the DWH Design Concepts such as flattening normalized data, flattening super types and subtypes. The transformed data is then copied into the Data Warehouse, where it can be pulled into reports or Cubes.

TX DWA allows multiple business units. This is very convenient when an organization has many different databases possibly from different countries that have different transformation rules.

Create a Business Unit and Staging Area

This task is comprised of three parts: creating the Business Unit itself, creating the staging database, and connecting to an external data source.
**TASK: CREATE THE BUSINESS UNIT**

1. In the Project Node, right click Business Units
2. Click “Add Business Unit”. The Add Business Unit dialogue box will appear.
3. Enter “AdventureWorksBU” in the Name field
4. Click “OK”. The Staging Database dialogue box will automatically appear.

Instructions for filling out the Staging Database dialogue box follow in the next section.

**TASK: CREATE A STAGING DATABASE**

A Business Unit must have an associated Staging database. As a result, once you create a Business Unit, it will automatically prompt you to create a Staging database as well.

1. Under “Name”, enter “AdventureStage”. This is the friendly name for the Staging Database that will appear in the TX DWA user interface.
2. In the SQL Server section, enter the server name. In this case, use “.” or “localhost”.
3. Under “Database”, enter “AW2012Stage”. This will be the name of the SQL Server database that stores DWH data and supporting objects.
4. Click “Create”. TX DWA will automatically create the new SQL Server database.
5. Click “OK”. The dialogue box will close.
6. Expand the Business Units node. This will show a number of different objects that can be created in the Staging Area, as well as the Data Sources node.

**Add A DATA SOURCE**

**Task: Connect to an External Data Source**

We now need to connect to the external data source. This will allow us to bring operational data into the Staging area. Once we copy over this data, we can begin to perform the ETL tasks necessary to prepare the data for the Data Warehouse.

1. Right click on the Data Sources node in the AdventureWorksBU
2. Select “Data Sources” in the pop-up menu, then select “Add SQL Server data source”

3. Under “Name”, enter “AW2012Source”. This is the friendly name for the external connection that will appear in the TX DWA user interface.
4. In the SQL Server section, enter the server name. In this case, use “.” or “localhost”.
5. Under “Database”, enter “AdventureWorks2012” or choose “AdventureWorks2012” from the dropdown menu. This should be the name of the SQL Server database that we created earlier to store the sample operational database.
6. Click OK. The dialogue box will close.
7. A “…” symbol will appear next to Data Sources. Double-click on Data Sources or click on the “…” symbol to view the new data source connection.

Task: Reading Objects from the Data Source

1. If you haven’t already, expand the Data Sources node to view the new connection you just created (Click the “…” button)
2. Select the connection “AW2012Source”. An empty window will appear on the right side of the work space
3. Right click on the connection and select “Read objects from Data Source”
4. Select “Read objects from data source”
The “Read objects from Data Source” window will appear as the system works. When this process finishes, the window on the right will display the tables and views present in the source.

Create Staging Table Structure
In this section, we will create a four tables that will serve as the beginning of the Staging area. These tables will form the foundation of DimProduct table in the Data Warehouse.

TX DWA has two methods for creating Staging tables. In this section, we will cover one method: importing a table from a Data Source. The other method, creating a custom table, is generally reserved for advanced ETL tasks.

Importing Tables from a Data Source
At this point, the Data Source “AW2012Source” should be selected, and a new window containing a list of objects should be open on the right side of the workspace. These are all tables and views present in the data source.

We need to bring some of these tables into the Staging area. To keep staging from getting cluttered, for now, let’s just import all of these tables related to Product into the staging area.
TASK: IMPORT PRODUC T INTO STAGING

1. Due to the large number of objects in the source, searching will help make finding the right tables easier. Enter “Product” in the search bar and press Enter or hit the >> icon to the right of the search bar.

2. Double-click on Product to view the fields in that table.

3. Click the box next to ProductID. A check-mark will appear next to the Product table name, indicating that this table has been brought into the staging database with all of the selected fields. A “✓” will also appear next to the Tables node in AdventureStage, indicating there is now an object in that group.

4. Check Color
5. Check Name
6. Check ProductID
7. Check ProductModelID
8. Check ProductSubcategoryID
9. Close Product to free up room in the data source window. Do this by clicking the “×” next to the table name, or double-clicking on Product.

10. Double-click on ProductCategory to view its fields.

11. In practice, we will always want to select one field at a time from a data source, and only choose fields we are absolutely sure we need. To expedite this exercise, we will simply bring in all fields from a number of tables, including ProductCategory. Do this by checking the box to the left of the table name. Notice that all of the fields are also automatically selected.

12. Check tables ProductModel and ProductSubcategory as well.
13. To view the full list of tables again, click “Alphabetical view”.
14. Deploy the tables in the Staging area as demonstrated in the Deploy Tables section.

Once you have selected and deployed the tables and columns you want to bring into Staging from the external source, expand the Tables node under AdventureStage. You should be able to see the tables you selected listed there. Note that before deployment, the table names should be red. After deployment, the table names should be black. Black names in TX DWA indicate that all changes to an object have been deployed.

Data Source Tables vs Staging Tables
If you open Data Sources, you will see tables that look very much like the ones present in the Staging area, shown here side by side:
Although both use blue table icons, the two objects are very different! The tables under Data Sources are views of the original tables in the operational data source. The present the data exactly as it exists in the source, but show only the columns you have selected. The tables in Staging are local copies based on those views. The copies in staging are the ones that we can change and manipulate as needed using TX DWA.

There are a couple of important uses for the tables under Data Sources which we will cover later, but for the most part it is a good idea to leave the Data Source node closed during development in order avoid confusion.

Denormalize “Product” Dimension

In the Create Staging Table Structure section we brought all the data we need to populate DimProduct in from our data source. However, this data is currently split between four different tables. We will need to go through some intermediate steps to join the tables together and add the necessary attributes to the main product table.

TX DWA will allow you to insert data directly from the involved tables in staging into DimProduct. However, this will not get us the denormalized (flattened) results we need. Instead we would create new lines for each combination of referenced tables because TX DWA doesn’t know how the data in these tables are related. The solution is to get all of the data properly related in one table, then copy that table into the Data Warehouse. Providing the end user with a copy, hides all of the necessary ETL work, and allows them to use access clean, simple copy of the data. The process of taking data from many tables and moving into a single table is called Denormalization, and is covered in Part 1 of this guide.

In the case of Product, ProductCategory, ProductSubcategory, and ProductModel, we would want to pull all of the fields into the Product table. Each Product is related to a Model, a Category, and a Subcategory directly or through another table i.e. indirectly.
To perform these copies, we will first arrange the tables in the proper order in Staging. Then, we must create direct and indirect relations to relate ProductCategory, ProductSubcategory, and Product Model to Product. Finally, we will perform the copies themselves using the TX DWA Lookup feature. Lookups will allow us to create new columns in Product that will hold data copied in from the other three tables.

**Task: Arrange Product tables in the Correct Order**

In order for TX to perform the lookups properly, you must ensure the target table (i.e. Product) is physically lower in the list than the source tables. In other words, in the denormalization process, you should always be dragging attributes DOWN into the destination table.

TX DWA allows you to change the order of most objects using simple drag-and-drop functionality.

1. After bringing in the Product tables from the data source and deploying them, your Staging area should look like this:
2. Left-click and hold on AW2012Source_Prod_ction_Product. Your cursor will turn into a black circle-and-line “no symbol”. This indicates that you cannot move the table onto itself.
3. Drag the AW2012Source_Prod_ction_Product table down to the bottom of the Tables area in Staging. You must drag it below AW2012Source_Prod_ction_ProductSubcategory, but above the Data Mapping node. If the cursor turns into the “no symbol”, you have moved the cursor too far down.
4. Release the mouse button. AW2012Source_Prod_ction_Product should move down to the bottom of the list.
5. The tables should look like this:
6. Deploy these tables

**Creating Relations**

To perform a lookup, you should first specify how the two tables relate to each other. In most cases, all fields in two related tables are related in a similar way. In these cases, it’s best practice to relate the tables before doing the lookups. This is has several benefits.
Using relations makes creating the lookups easier. Lookups require the user to specify joins, or how the data in the lookup column is related to the data in the target table. Creating relations eliminates the need to specify those lookup conditions. This is because the lookup will use the relationship between the two tables as joins by default if no other joins are specified. This benefit is especially useful when creating multiple lookups.

Changing relations is much easier than changing lookup conditions. If you have multiple lookups from the same table that rely on the table relations instead of individualized conditions, you only have to change the table relation to get the lookups to use new criteria. The alternative is having to change each lookup one at a time.

Lastly, even if you only have one lookup field, creating a table relation makes the interaction between the two tables clear and explicit.

Note that there are cases where relations should not be made between tables to perform lookups. Some of those cases will be covered later in the guide.

Denormalize Product Hierarchy Tables

In this section we will define the relationships between the primary keys in ProductSubcategory, ProductCategory and ProductModel to the corresponding foreign keys in Product. Relationships tell the TX application what fields to use as join conditions between tables.

Task: Denormalize ProductCategory into ProductSubcategory – Direct Relation

There are two main kinds of relations – direct and indirect. In direct relations, two tables either share a primary key, or have a primary key/foreign key relation. In AdventureWorks, ProductCategory and ProductSubcategory are directly related. We can determine this by noting that there is a foreign key from ProductCategory in ProductSubcategory.

We will learn how to relate these tables together and to denormalize ProductSubcategory by adding the field ProductCategory.Name to ProductSubcategory.

1. Expand ProductCategory and ProductSubcategory.
2. Identify the table ProductCategory and find the field ProductCategoryID.
3. Relate the two tables by left-clicking and dragging ProductCategoryID from ProductCategory (the source table) onto ProductCategoryID in the ProductSubcategory table (target table).
   a. A dialogue box will pop up asking "Do you want to create relation?"
   b. Click Yes.
4. If done correctly you will see a new “relation” node created at the bottom of the ProductSubcategory tree.
   a. A dialogue box will pop up asking "Do you want to create relation?"
   b. Click No.
6. A new field “Name2” appears in ProductSubcategory. Right click on Name2 and select “Edit Conditional Lookup”.
7. Change the name to “ProductCategoryName”. If done correctly, ProductSubcategory should look like this.

Relating ProductCategory and ProductSubcategory together and bringing ProductCategory.Name into ProductSubcategory eliminated the indirect relationship between ProductCategory and Product. Now, the field we needed from productCategory (Name) is contained in ProductSubcategory. The next thing that we have to do is to denormalize the data that is now in ProductSubcategory into Product.

Task: Denormalize ProductSubcategory into Product – Direct Relation

Now that the ProductCategory data that we need is denormalized into ProductSubcategory, denormalize ProductSubcategory into Product. Please follow the steps below.

1. Expand Product and ProductSubcategory. Note that Product has foreign keys for ProductSubcategory.
2. Identify the table ProductSubcategory and find the field ProductSubcategoryID.
3. Relate the two tables by left-clicking and dragging ProductSubcategoryID from ProductSubcategory (the source table) onto ProductSubcategoryID in the Product table (target table).
   a. A dialogue box will pop up asking "Do you want to create relation?"
   b. Click Yes.
4. If done correctly you will see a new “relation” node created at the bottom inside the Product table object.
5. Drag ProductCategory.CategoryName onto Product table name.
   a. A dialogue box will pop up asking "Do you want to create relation?"
   b. Click No.
   a. A dialogue box will pop up asking "Do you want to create relation?"
   b. Click No.
7. A new field “Name2” appears in ProductSubcategory.
8. Change the name to “ProductSubcategoryName”. If done correctly, Product should look like this.

Note: For tables with many fields, it is more convenient to open a window (Ctl + w) rather than expanding the tables in the tree as we did above.
Task: Exercise

1. Denormalize ProductModel into Product
2. If done correctly, the final staging Product table should look like this:

Executing

While deploying a table commits any changes made to it and saves the project, execution is needed to populate the table with data.

Executing a table will first pull data in from a source. In Staging, that source is one or more operational Data Sources. This copy is handled automatically by TX DWA. In the Data Warehouse, that source is usually a table or view in Staging, and must be explicitly configured by the user.

Once data is copied into a table, it will undergo any changes that are specified by the TX DWA table setup. These changes can include transformations and lookups, or data quality checks like the enforcement of primary key constraints. If no changes are specified, the execution will be a simple copy of the original data.

Deployment and execution are often done together, so that the changes made to a table can be reflected in its data. Additionally, deploying a normal table will remove all of the existing data, leaving it empty. The table must then be executed to repopulate it.

Execution without deployment is reserved for reloading data only and no structural changes to the object(s) has been made.

Deploy & Execute Tables

1. Right click your staging database
2. Click Deploy & Execute
3. The Deploy and/or Execute window will appear
4. Click Start
5. The Save Version Details Dialog will appear
6. Click OK
7. The Deploy and Execute window will appear. When it is finished, click the “Close” button

Once you have deployed & executed the tables, you can preview the Product table and see the new fields.

1. Right click the Product table name
2. Click Preview Table. Add four zeros to the “Select Top:” field & click “Apply Top” button

Note that most products do not have a Category, Subcategory and Model values. These are parts. Products like bikes and clothing do have these values.
Denormalize Person Table
This section will cover denormalizing the Customer table in staging area.

The AdventureWorks transaction system has complex data structures. It has a BusinessEntity supertype which has subtypes (roles) of Person, Vendor and Store (Store = Reseller). Person can be an Employee, Salesperson, Contact or Customer. The diagram to the right shows the tables that are important to the BusinessEntity supertype.

Even though BusinessEntity is the supertype of Person, Vendor and Store, we will not be using it because it only contains the BusinessEntityID key and no other fields of interest to us.

Our user stories specify that we need a Customer dimension for Internet Sales. However, we can’t just use Person as it is.

Analysis has shown that the SalesOrderHeader table in AdventureWorks does not have A PersonID or a BusinessEntityID as a foreign key for the Customer. Instead, it has a Customer ID.

One would think that this is the same as the BusinessEntityID in the Person table, but this is not the case. If you look at the Customer table, you will see that it has a CustomerID, a Person ID and a Store ID. It contains these fields because both people and stores can be customers. The SalesOrderHeader table does not have separate fields for Person Customers and Store Customers (Resellers). It has a single field called Customer ID. This can either be a Person or a Store. So, we’re going to have to denormalize the CustomerID field from the Customer table into the Person table.

Notice also, that the person table does not have an email address. We will need to denormalize the email address into the Customer table.

Strategy for Relating Business Entity Subtypes Together
In this sprint, we only want to include customers where Person.Person Type = ‘IN’.

Background
- We will not use BusinessEntity in these sprints
- Person.PersonType contains information about the various types of Person
- We are only interested in the ‘IN’ (individual retail customer)
- Customer table does not have a BusinessEntityID
Strategy
- We will denormalize all needed fields from any Person child table or related table into Person.
- We know that Sprint 2 will need another person role in addition to Customer
- We will learn how to handle this situation

Task: Select Needed Tables for Customer
1. Click on the data source you want to import data from
2. The objects list will open on the right side
3. Double click Person and check BusinessEntityID, FirstName, LastName, MiddleName, PersonType and Title. Select all fields from Customer & EmailAddress
4. Expand the Tables node in Staging
5. Reorder the tables: Customer, EmailAddress & Person.
6. Your Staging area should now look like the image shown here.

Task: Relate Customer and Email Address to Person
In this task, put the Person table into a separate window. This makes it easier to work with multiple tables at a time.
1. Select Person table name and CTL + w
2. Drag Customer.PersonID onto Person.BusinessEntityID
3. Drag EmailAddress.BusinessEntityID onto Person.BusinessEntityID
4. The Person table tree should show relationships
The EmailAddress and Customer tables are now related to Person. The next step is to bring the fields that are needed from EmailAddress and Customer into Person

Task: Bring new fields into person from EmailAddress and Customer Tables
1. Drag EmailAddress.EmailAddress onto Person
2. Drag Customer.CustomerID onto Person
3. Person should look like this
4. Deploy & execute tables
5. Preview Person table
Task: Create New Field in Person Staging Table

We would like to have a single field in which to display a person’s name rather than have to include two field names in a query or report. To do this, we must add a new field to the Person staging table. It is possible to do this in the DWH section, but it is best practice to do all transformations in the staging area.

1. Right click Person staging table
2. Click “Add Custom Field”
3. Type “DisplayName” into “Field Name” field in “Add Field” popup
4. Expand Person table
5. Person table should look like this

Task: Apply Transformation to Staging Person.DisplayName

Transformations are a way to add code to a field in order to accomplish a specific task or to create value and a custom field based on the contents of one or more separate fields. The example that we will use is to concatenate First Name and Last Name in the Person staging table into the “DisplayName” field that we just created.

1. Right click “DisplayName” and select “Field Transformations”. The Field Transformation box opens on the right
2. Select “Display Name” on the top and make sure the Operator field at the bottom contains "Custom”
3. Click "Add” and the "Transformation Custom SQL” window opens
4. Drag [First Name] and [LastName] from the list on the left into the window
5. Separate them with + ' ' + to concatenate FirstName and LastName together.
6. Expand “Display Name” and see transformation
7. Click OK
8. Deploy & execute
9. Preview Person table

Denormalize “SalesOrder” Fact Table
In this section we will be implementing the concepts of Denormalized Header & Detail Facts. Many times, operational database fact tables will be normalized into multiple tables just like the previous dimensions. The prime example of this is the order header and order detail tables. Just like with a physical sales order you have the header details that apply to the entire order, such as the customer, order date, ship date, ship to address, freight charges, or taxes etc. Then you have the line item details such as the product name, price, and quantity.

Although the tables are of a different grain, best practice is to move the Order Header lines down to the Order Detail records. The SalesOrderHeader contains fields that are for the order as a whole instead of relating to a single SalesOrderDetail line. In our example, these fields are Freight and TaxAmount. So, we can’t just move these fields into every SalesOrderDetail lines. If we did that, every SalesOrderDetail line would have the full amount of Freight and TaxAmount. This is obviously not correct.

We could also allocate the Freight and TaxAmount to every detail line. However, it is difficult to get business rules on how to do this.

A solution is to create a separate SalesOrderDetail line for all of the fields in the header that contain amounts for the order as a whole. This works as well as the normalized structure in the source system. But once these amounts are moved into the SalesOrderDetail, how would we know which amounts are which?

The following sections will start by explaining how to denormalize fields from the SalesOrderHeader that are appropriate to be in the SalesOrderDetail. Then we will explain how to incorporate the TaxAmount and Fright charges into the SalesOrderDetail table. These facts are for the order as a whole and not for the detail records. This will require creating
“artificial” ProductIDs for TaxAmount and Freight. This will be clear after we denormalize the appropriate fields from the SalesOrderHeader into the SalesOrderDetail.

**Denormalizing SalesOrderDetail Table**

Just like we denormalized the Person table, we have to denormalize the Sales Order Detail table as well.

**TASK: DENORMALIZE SALESORDERDETAIL TABLE**

1. Select SalesOrderHeader & SalesOrderDetail from AW2012Source
2. Select the fields from SalesOrderDetail shown at right in blue
3. Select the fields from SalesOrderHeader shown at right shown in red plus Freight and TaxAmt
4. Order tables so that SalesOrderDetail is below SalesOrderHeader
5. Relate the tables together by dragging SalesOrderHeader.SaleOrderID onto SalesOrderDetail.SalesOrderID
6. Drag Fields from Header to Detail
7. When done, should look like figure on right
8. Right click “OnLineOrderFlag”
9. Expand until see green icon
10. Right click and select Edit Lookup
11. Change operator to “Top”
12. Deploy & Execute
13. Preview SalesOrderDetail table

**Task: Relate Staging Product Table to SalesOrderDetail**

We must make a relationship between Product and SalesOrderDetail so that we can show the correct product in queries. We must also create a foreign key that can be used in the data warehouse to join between DimProduct and FactInternetSales. Use this pattern to relate any necessary dimension tables to a fact table and create a data warehouse key. To accomplish this, do the following steps.

**TASK: RELATE PRODUCT TO SALESORDERDETAIL**

1. Drag Product.ProductID onto the SalesOrderDetail.ProductID and answer “Yes” to “Do you want to create a relation?”
2. Drag Product.DW_Id onto SalesOrderDetail name. Choose “Do not use a relation as join”
3. Deploy and Execute
4. Preview SalesOrderDetail table

**TASK: SET UP SALESORDERDETAIL CUSTOMERKEY USING AN EXPLICIT JOIN**

As described earlier, there is a need to relate both an Internet customer and a salesperson to the SalesOrderDetail table. These are both roles of the Person table. Because the same Person table plays multiple roles in the SalesOrderDetail table, we cannot relate Person to the SalesOrderDetail table the way that we related Product.ProductID to the SalesOrderDetail table. We would get incorrect results. Instead, we have to make an explicit join for an individual
customer. We do this by joining the Person table to the Sales Order Header table on a CustomerKey. We need this customer key for the data warehouse fact table in order to join the appropriate customer to the data warehouse fact table. The steps to do this are:

1. Expand Person.System Fields
2. Drag Person.DW_Id onto SalesOrderDetail name
3. Rename SalesOrderDetail.DW_Id2 to CustomerKey and change the CustomerKey data type from bigint to int
4. Right click the “Join” under CustomerKey
5. Right click “Add Join”
6. Select “CustomerID” for both “Join Column” and “Value” drop downs
7. Click OK
8. Expand the “Join” node and see join definition
9. Deploy & Execute

Bring SalesOrderHeader Fields into Denormalized SalesOrderDetail Table
Just as in the previous examples, we must relate the source table (SalesOrderHeader) to the destination table (SalesOrderDetail) and drag the needed fields from the source table to the destination table.

Moving Facts from Sales Header to Sales Line
There are often measures in the header that are part of calculating the total for an order that do not appear in the sales line. To resolve this, we must identify the measures, create new product types for them in the product table in Staging, and then copy the values from the header into the transaction table as new transaction lines. In Adventure Works 2012, these measures are Tax Amount and Freight.

Identify Header Level Measures to Be Moved to Detail table
In this section, we will identify the measures in the SalesOrderHeader table we need to allocate down to the transaction line. Remember that we must create “artificial” Products for the Header measures in order to use them in the SalesOrderDetail table.

We can document this by making a comment using the Description feature. This feature can be used by TX DWA developers to leave notes for themselves or others. Most TX DWA objects can be commented on using the Describe feature.

TASK: INSERT COMMENT
1. Expand the SalesOrderHeader table
2. Scroll through the values to identify which Header fields are measures.
3. Right click the Freight field name in the tree.
4. Click Description.
Write a description for the field. For example, “Freight amount for entire order. Must be assigned a ProductID and properly allocated into the Sales Detail table.” The fields with the descriptions turn bold.

Click OK.

Do this for all Header Measures to be moved to the SalesOrderDetail table.

For our example, the fields we only need Freight, and TaxAmt.

Now the table field names that have custom descriptors will be in bold as shown here.

Insert Artificial Items in Product Table Manually

Now that you have a clear list of the Header Measures, you can create their associated “Products” in the Product table.

**TASK: INSERT “ARTIFICIAL” PRODUCT IDS FOR HEADER FACTS**

1. Right click on Product table in staging. Select Advanced. This will open a sub-menu.
2. Under Advanced, click “Custom Data”. This function allows us to enter new “artificial Products” into the table through TX without altering the source table.
3. Scroll right until you find the “Name” field.
4. Enter the name of the first measure header identified in the last step. In this case “Freight”.
5. Create a ProductID for this measure. This product ID must not overlap with a source system ProductID. Since all of the product IDs in Adventure Works are positive numbers, we will use a sequence of negative numbers beginning with “-10” continuing with -11, -12, -13, -14 etc. In this case, enter “-11” for the ProductID.
6. Repeat these steps for each measure that will be moved from the SalesOrderHeader to SalesOrderDetail. In this case, TaxAmt with a ProductID of of -11.
7. Click OK.

Insert Header Facts as Rows in the SalesOrderDetail Table

Now that we have identified the Order Header Measures and created ProductIDs for them in the Product table, we need to insert these measures into the SalesOrderDetail Table. In TX, this is done using a feature called a Table Insert. This feature allows us to copy rows and/or static values from one table into another. We must create a Table Insert for each of the previously identified Header measures.
**TASK: CREATE TABLE INSERT & MAP EXISTING FIELDS**

1. Right click SalesOrderDetail table
2. Under Advanced, click “Add Table Insert”
3. Type “Insert Freight” in Name
4. Select “Insert from table” from dropdown & choose SalesOrderHeader & click “OK”
5. Next to LineTotal click Mapping dropdown & select Freight
6. Next to OrderQty click Mapping dropdown & select “Fixed Value”
7. Enter 1 in the Fixed Value field
8. Next to ProductID click Mapping dropdown & select “Fixed Value”
9. Enter ProductID that you assigned for Freight (-10)
10. Next to SalesOrderDetailID click Mapping dropdown & select “Fixed Value”
11. Enter -10 in the Fixed Value field
12. Next to SalesOrderID click Mapping dropdown & select “SalesOrderID”
13. Next to UnitPrice click Mapping dropdown & select Freight
14. Next to UnitPriceDiscount click Mapping dropdown & select “Fixed Value”
15. Enter 0 in the Fixed Value field
16. Click OK
17. A “Table Inserts” folder will appear under SalesOrderDetail
18. Expand & you will see Insert Freight Values
19. Right click on “Table Inserts” folder, click “Add Table Insert” and do the same steps for TaxAmt using a fixed value of -11 for TaxAmt.

**TASK: DEPLOY & EXECUTE STAGING TABLES**

To populate the table with the new data from your table inserts you must Deploy & Execute the tables. After you do this:

1. Right click the SalesOrderDetail table & select “Preview Table”
2. Click the “Query Tool” button above the “Close” Button to bring up the Query Tool
3. Enter 4 additional zeros in the “Max no. of rows” field.
4. Type in “Where ProductID < 0” at the end of the Select statement, click the “Execute” button & you will see all of the Freight and TaxAmt entries.
Map Staging Fields into DWH

Once we have a working copy of a dimension table or fact table in Staging, we will copy the necessary fields from the Staging version of the table up into the Data Warehouse. From there it can be queried by the client or used in a cube.

We can either manually map these fields one at a time, or use TX Smart Sync feature to quickly map certain fields. If you drag a Staging table onto a Warehouse table, you will be offered three options: Smart Sync with table <TableName>, Synchronize with table <TableName>, and Synchronize with table <TableName> (Existing fields only). Smart sync and sync existing fields only options are functionally the same at this point - if a field has the same name in both Staging and in the Warehouse, TX will create the copies between the tables automatically.

“Synchronize with table <TableName>” will create copies between the fields that share a name in both Staging and the DWH, just like the other options. In addition, it will create fields that exist in Staging but not the Warehouse to the Warehouse table, and automatically create copies for these new fields as well.

Make sure all staging and DWH tables are deployed and executed.

Task: Map Staging Tables to the Data Warehouse Tables

We have already imported DimProduct and DimCustomer into the DWH from an external source. When we deployed them, we only created the data structures in the data warehouse database. So far, there’s been no data transferred to these data warehouse tables. Now that we have done the denormalizations and other necessary transformations in the staging area, we can now do the necessary steps to load the data from the staging area into the data warehouse tables.

There is more than one way to do this. We will explore different ways to transfer data from the staging area to the data warehouse tables in the section.

Task: Map Fields Using Synchronization

1. Put Staging Product table in new window (CTRL-W)
2. Drag Staging Product table name onto DimProduct. A menu pops up
3. Click “Synchronize with table DimProduct (Only existing fields)”. All DimProduct fields will have a plus beside it except ProductKey and ProductName
4. Drag missing fields from Product table onto DimProduct
5. Drag Product.Name to DimProduct.ProductName
6. Drag Staging Product.DW_Id system field onto DimProduct.ProductKey. This creates a surrogate key for the data warehouse. A “Data Movement” options dialog will open. Make sure that “From DW_Id” is mapped to “To ProductKey” and click OK.
7. All of the fields in DimProduct should now have a plus sign beside it. Expand all of the fields in DimProduct.
8. DimProduct should look like this.
9. Follow these steps for the remaining denormalized dimension tables you created in staging.

**EXERCISE:** CREATE PRIMARY KEY FOR DIMCUSTOMER

Note: DimCustomer table layout found in Appendix 4.

**TASK:** MAP SALESORDERDETAIL TO DWH WITH SYNCHRONIZATION

Since FactInternetSales in the DWH does not have any columns defined for it, use the

Currently, there is no “real” fact table in the DWH. There is only the “FactInternetSales” table name with no data elements. The task below will show how to create DWH table fields directly from a staging table.

1. Open the SalesOrderDetail table in a new window.
2. Drag the SalesOrderDetail table name onto the FactInternetSales table name.
3. A pop-up menu will appear and offer the three options discussed above.
4. Select option 2 “Synchronize with table FactInternetSales”.
5. FactInternetSale now has the same fields as SalesOrderDetail.
6. Expand the fields to view the copies.
7. Edit FactInternetSales.ProductKey and change the data type from bigint to int.
8. Edit FactInternetSales.CustomerKey and change the data type from bigint to int.

When using the Synchronize tool, it’s always a good idea to check to make sure the copies have been mapped as expected. Your screen should look like the screen shot.

**TASK:** DEPLOY & EXECUTE THE PROJECT

Once the fields have all been mapped you must Deploy & Execute the entire project to ensure all the data is populated correctly.

1. Right Click the Project name.
2. Under Deploy & Execute, Click Deploy & Execute.
3. Preview DimCustomer and write down the number of rows. Remember to add 4 zeros to the “Select Top:” field & click the “Apply Top” button.
Create DWH Table Loading Conditions

Currently, all staging Person records will be loaded into the DimCustomer table. Since we only want to have internet sales customers loaded into DimCustomer, we must add a “Data Selection Rule” to the DimCustomer table in order to load only internet sales customers.

**TASK: ADD DATA SELECTION RULE TO DIMCUSTOMER**

1. Right click on DimCustomer and click "Add Data Selection Rule". The “Add Data Selection Rule” dialog box opens on the right.
2. Select PersonType in the list at the top.
4. Type IN (no quotes) in the Value field.
5. Click the "Add" button.
6. Expand DimCustomer and notice a condition “Customer Type Equal IN”.
7. Deploy and execute DimCustomer.
8. Preview the DimCustomer table. Remember to add 4 zeros to the “Select Top:” field & click the “Apply Top” button. Note the number of rows now.

Now, only people who are internet sales customers will be loaded into DimCustomer.

Exercise

Limit the transfer of records from SalesOrderDetail table to only those where the OnlineOrderFlag is true.

Date Dimension

Most data warehouse consumers need to see reports that are based on regular and fiscal calendar components such as week, month, quarter, year, etc. Historically, getting data at these granularities involved using complex SQL date functions. Some of these are DATEPART, DATEFROMPART, DAY, MONTH, YEAR. Each of these require at least one parameter and writing SQL code.

To make reporting by date components easier, virtually all data warehouses utilize a date dimension. The date dimension is a hierarchy table that explicitly lists the various date components. A typical date dimension is shown on the right of this page. As you can see, most of the date components are listed.

The DisplayDate field shows the date in a standard format such as “2015-12-23”. Traditionally, someone had to write some code to populate all of these date components. TX DWA automatically generates a date dimension that has a few more fields than the example shown in the DimDate table on the right. These additional fields include FirstDayOfMonth, LastDayOfMonth, FirstDayOfQuarter, LastDayOfQuarter, FirstDayOfYear, LastDayOfYear, etc.

The task below shows how to create a date dimension in TX DWA.
TASK: CREATE THE DATE DIMENSION

1. Right click DWH “Tables” node
2. Click “Add Date Table”
3. Name: “DimDate”
4. Set start date to 2000-01-01
5. Enter 5000 in “Days ahead” or whatever future time in days that you prefer
6. Click “OK”
7. Deploy & execute

Creating Views

Database views offer a lot of value in the data warehouse. For example, they can simplify complex queries, they can limit data access to specific users and they can provide an extra security layer as well as many other benefits.

Creating views is very easy in TX. All you have to do is select the table in either the Data Warehouse section or the Staging Section. It is common to have many types of date fields (OrderDate, DueDate, ShipDate, etc.).

Dim Date is an example of a "role playing dimension" because it represents many types of dates.

Let’s see how to create a view for the order date field. After this, as an exercise, you will create views for DueDate and ShipDate.

Create View for OrderDate

1. In the Data Warehouses section, expand Tables
2. Drag table DimDate name down onto views in Data Warehouses section. (“View Name” window pops up)
3. Change the name in the “View Name” window to “DimOrderDate”
4. The Edit Custom View window opens. You can:
   a. Get rid of any fields in the table that you don’t need
   b. Rename a field by typing “AS whatever”
   c. Write a where clause at the bottom of the script in the Edit Custom View window
   d. In the where clause, the field names that you use must look exactly like the field name in the view script including brackets. The best way is to copy the name from the script and paste to insert
5. Click OK
6. A pop-up “Remove Custom Properties?” May appear. If it does, say YES
7. Deploy View
8. Right click the view name
9. Click “Read Views”. You must do this for view to show fields
10. If rename a field in view, right click view and choose “Synchronize View Fields”
Exercise: Create views for DueDate and ShipDate

Enable History Tracking through Slowly Changing Dimensions
In this section we will be implementing the concept of Surrogate Keys vs. Natural Keys discussed on page 7 and Slowly Changing Dimensions.

Task: Assign Dimension Primary Keys
In this step you will configure TX to be able to identify what dimension attributes in staging are Primary Keys. First, identify the dimension table’s natural key. It is typically one of the first attributes with a name similar to the table name and includes the words "ID" or "number" in the attribute name. In this example, it will be the ProductID in the Product Table and the BusinessEntityID in the Person Table.

Task: Create Primary Key
1. Expand Staging Product table
2. Right click on ProductID & Click “Include in Primary Key”

Task: Modify Staging Product Table Settings
TX DWA allows you to control table behavior. You accomplish this through table settings. Here’s how you change table settings in the Product table.

1. Right click Staging Product table
2. Click “Table Settings” or F4
3. Click “History Settings Tab”
4. Check “Enable History” – Red “X” appears
5. Click “Data Extraction Tab”
6. Uncheck “Truncate valid table before data cleansing”
7. Click OK
8. An “H” will appear on the table
Select History Fields

1. Expand the new staging history table. A yellow icon called “History Settings” will appear at the bottom of the list of fields.
2. Select History Settings. The history settings options will appear at the right of the workspace.
3. The Natural Key is already selected.
4. Select check box(es) next to all Type II slowly changing dimension fields. In this case, there is only one, “Color”.

Set Up Staging Transaction Table to Recognize SCD II

In addition to enabling dimension tables for SCD II, must also enable Fact table to recognize SCD II new records.

1. Expand SalesOrderDetail
2. Expand ProductKey
3. Expand Lookup Fields
4. Expand DW_Id
5. Locate Join node
6. Right click on Join node & select “Add Join”.
7. “Add Join” window opens, select ProductId in “Join Column” and in the “Value” drop down & OK.
8. Add Join for “SCD To DateTime >= OrderDate”.
9. Add Join for “SCD From DateTime < OrderDate” this.
10. Deploy & Execute the Project.

Any dimension where the DW_Id is used to create a primary key in a fact table must be a history table. Otherwise, if the table is rebuilt in the source or additions made other than being appended to the table, the source ID field and the DW_Id will be out of sync. This causes incorrect lookup values.
Configure Incremental load

As data volumes grow, it takes longer and longer to load the data if you doing a full load every time. Eventually, it may take so long to do a full load that users are blocked from using the data warehouse. This is remedied by setting large tables to be incrementally loaded. Incremental loading is a technique to reduce the amount data (and therefore, the time) it takes to load data into the data warehouse from source systems. This can reduce the ETL time enormously.

Incremental loading is easily done in TX DWA. In order to implement incremental loading, we must configure the tables in Staging and in the Source System. The sections below describe how to do this using SalesOrderDetail as an example.

Set up Staging Tables

Create Staging Table Incremental Primary Key

1. In order to configure the staging area tables, we must first create a primary key for the incremental load.
2. Expand SalesOrderDetail
3. Right click on SalesOrderDetailID
4. Click “Include in Primary Key”
5. Repeat for SalesOrderID

Next, we must modify the table settings

Modify Source Table Settings

1. Right click SalesOrderDetail
2. Click “Table Settings” or F4
3. Click “Data Extraction Tab”
4. Check “Enable source-based incremental load” – Error Message will show up
5. Uncheck “Truncate valid table before data cleansing”
6. Click OK

Finally, we must create the source system incremental selection rule.

Source Incremental Selection Rule

1. Under Data Sources Node, right click the source table for SalesOrderDetail
2. Right click on the SalesOrderDetail source table and select “Add Incremental Selection Rule”.
3. A list of incremental values in the source table will appear on the right of the workspace. Check the box next to ModifiedDate.

TIMEXTENDER
4. The table is now set up for source-to-Staging incremental loading. Expand the table to see incremental selection rule.

Stage to DWH Incremental Settings
We need to add a SalesOrderDetail system field to the DWH Fact table

1. Open SalesOrderDetail in a new window and expand System Fields
2. Drag DW_TimeStamp from SalesOrderDetail onto the table name of FactInternetSales
3. Expand FactInternetSales and rename “SalesOrderDetail DW_TimeStamp” to “StagingTimeStamp”

Set Up Staging Table for Incremental Load
Make FactInternetSales incremental by setting up a primary key and modifying the Table Settings like we did for SalesOrderDetail in Staging.

1. Make FactInternetSales incremental by setting up a Primary Key and modifying Table Settings like we did for SalesOrderDetail in Staging
2. Right click FactInternetSales table and select Add Incremental Selection Rule
3. Check the box next to StagingTimeStamp
4. The table is now formatted for a Staging-to-Data Warehouse incremental load
Deploying and Executing Incremental Tables

1. Deploy and execute the SalesOrderDetail
2. Click “Start” to load the table
3. Repeat for FactInternetSales
4. If you see this message, all it means is that there is more than one table not deployed in the incremental process and that **both** have to be deployed & executed so that the incremental load process will work properly.
5. After successfully deploying & executing, the tables will have an “I” imbedded

DWH Exercise

So that you can have a better understanding of the things that you have learned so far, you need to do this exercise.

User Story

As the Sales Director, I need to understand:

- Who are my top salespeople?
- Who are the top salespeople by product?
- Who are our top resellers by product?
- Who are our top customers?
- What is the year-over-year change in sales by salesperson, by product?
- And many more sales questions.

What we know

- SalesOrderDetail has sales for internet customers and resellers
- FactInternetSales contains only internet sales orders
- Only resellers sales have a sales person assigned

To do:

- Add OrderDate and ShipDate to SalesOrderDetail
- Add a Reseller Dimension
- Add SalesPerson Dimension (Remember how we set up SalesOrderDetail.CustomerKey earlier)
- Sales person = Person.PersonType = SP
- Remove data selection rule on FactInternetSales
- Rename FactInternetSales to FactSalesTransaction
• Create separate views for FactInternetSales and FactResellerSales
Cubes Implementation
This chapter serve as step-by-step instructions on how to create a Cube in TX DWA. We will continue from the last chapter using the bike company AdventureWorks as our business case.

Prerequisites
In order to work through the steps, you will need:

6. The AdventureDWH data warehouse created in: Data Warehouse Implementation

Online Analytical Processing (OLAP)
An Online Analytical Processing (OLAP) database (also called a Cube) is a multidimensional data structure. Cubes were developed for the purpose of business intelligence by reducing query response time. The OLAP engine does this by pre-calculating aggregates during processing, so a query is not calculating the results but simply extracting it. This provides the end user with almost instantaneous results. As an example, imagine the difference in performance between running a simple SUM() function across several million rows in Excel and to only reference a single cell. That is what makes OLAP such a powerful reporting tool as it enables analysis across much larger data sets.

The source data for Cubes is pulled from the fact tables and dimensions in the data warehouse. The aggregates are based on the intersections between dimensions and facts in the star schema.

Cubes are part of the presentation area as shown in the architecture diagram below.

Terminology
OLAP has its own terminology. Before diving into how to build a cube, it is a good idea to learn the terminology. This will ensure that there is no misunderstanding when we walk through the steps of creating an OLAP solution.
**Attribute**
An attribute is a field in a dimension. Dimensions usually have several attributes.

**Cube**
A cube is a special type of database (OLAP). It is designed to process multi-dimensional designs.

**Measure**
A measure is a numeric value that lacks context by itself. Examples: 1,562.59; 9,001; etc.

**Measure Group**
A measure group is a collection of related measures.

**Dimension**
A dimension is a data structure that contains descriptive data that gives context to measures. Examples include Products, Customers and Time. Dimensions are made up of attributes. A dimension may also include a hierarchy that is made up of “Levels”.

**Level**
A level is a property of a hierarchy and represents a particular position in the hierarchy.

**Hierarchy**
A hierarchy sets relationships between levels. It is used to provide a seamless drill path for dimension levels. For example, the Time hierarchy may consist of years, quarters, months and days.

### Create an OLAP Server

The first thing that we must do is to create an OLAP server. This sets up an Analysis Services OLAP database so that you can create cubes. The steps to create an OLAP server are below.

1. Click on the Cube tab above the work space.
2. Right-click on “OLAP Servers” and click “Add OLAP Server.” A dialogue box will appear.
3. In the dialogue box, “AdventureCubes” in the Name field. This is the friendly name for the cube server that will appear in the TX DWA interface.
4. In Server Name, enter “.” or “localhost.”
5. In Database, enter AW2012OLAP.
6. Click OK.
7. A dialogue box will appear asking to open the data warehouse in a new window. Select “yes.”

Now that our server has been set up, we can add cubes to it.
Create a Cube
As mentioned above, cubes are multidimensional structures that contain dimensions and measures. You can have many cubes in an OLAP server. Like a relational database, each of these cubes can be set to incremental loading.

Security can also be set on OLAP cubes. This is different from the security that is set for the relational database. Security will be covered in a future version of this document.

The steps to create a cube are as follows:

1. Under AdventureCubes, right-click on Cubes and select “Add Cube”
   A dialogue box will appear.
2. Under Name, enter “Sales”
3. From the list of tables, check the box next to FactInternetSales
4. Click OK
5. The Sales cube will appear under AdventureCubes

Create Measures
The first thing to do after the cube has been created is to create the measures. Measures are the facts that users are interested in. Measures can be aggregations (sums), counts, averages, calculations and many other types. Measures are based on an underlying fact table or view in the data warehouse.

There are three types of measures and an optional “business function” measure developed by TimeXtender.

The types of measures are:
• Standard measures – These correspond to columns in the DW fact table. These are also referred to as “basic” measures
• Derived measures – These are pre-calculated measures that are calculated when the cube is being processed. It only has to be calculated once
• Calculated measures – These are calculated at run-time
• Business function measures – These are reusable, parameterized “wrappers” around MDX statement(s) used as a calculated measure in a cube. Business function measures can standardize frequently used calculations to ensure that these calculations are always done the same and not different between cubes or across different projects. It also eliminates broken MDX caused by renaming of a dimension or measure

Create a Standard Measure
1. Expand the Sales cube. Right-click on Measures and select Add Standard Measure. The Add Measure window will appear
2. Expand the Sales cube. Right-click on Measures and select Add Standard Measure. The Add Measure window will appear.
3. Under Name, enter “Total”
4. Under Field, select LineTotal
5. From Format String, select #,0.00;(#,0.00)
6. Click OK
7. Expand Measures. The “Total” measure should be visible.
8. Repeat for GrossLineTotal
Create a Business Function Measure

We have created a business function measure to demonstrate how to reuse a common calculation. As we go through the steps, the details about the parameters will be explained.

1. Right-click on Measures, click Add Business Function then click Discount Percentage Calculation. The Business Function window will open.
2. Name the measure Discount Percentage.
3. For Associated Measure group, select FactSalesTransaction.
4. Select the format %0.00 from the format dropdown menu.
5. Note that the Formula radio button is selected. This is the one selected automatically when you finish the Business Function Measure and shows the Parameter names.
6. For the parameter NetSales, select Total from the dropdown menu. For Gross Sales, select GrossLineTotal.
7. Click the MDX radio button. Note that the names in the formula change to the names in the Values column in the Parameters section. Reselect the Formula radio button.
8. Click OK.
Create a Calculated Measure
A calculated measure is a measure that you define by entering a formula.

1. Right-click on Measures and select Add Calculated Measure
   The Add Calculated Measure window will open
2. Name the measure Discount Amount
3. Select FactInternetSales from the Associated Measure Group dropdown
4. Select the format string #,0.00
5. Expand the measures node at the right of the window and drag GrossLineTotal and Total into the Expression box to create the following statement: [Measures].[GrossLineTotal] – [Measures].[Total]
6. Click OK

Create OLAP Dimensions
After the measures have been created, the next thing to do is to create OLAP dimensions. These dimensions can be used by any of the cubes in an OLAP database. They are sourced from the relational data warehouse. If you have not done so, press the Data tab next to the Cubes tab, right-click the AdventureDWH node and press CTRL-W to create a window for the data warehouse and expand it. Click the Cubes node and we are back in the OLAP section. Now you will be able to drag the data warehouse tables into the OLAP server. No manual entry of all of the fields is required.
Create Product Dimension

1. From the open Data Warehouse window, drag DimProduct onto the Dimensions node under Adventure Cubes. The Add Dimension dialogue box will appear.

2. Under Name, enter DimProduct and click “OK”.
3. Once you leave the Add Dimension dialogue box, a new dialogue box called Add Quick Levels will appear. Select all of the fields except the System fields (the ones that start with DW_)
4. Click OK
5. A new dialogue box with a dropdown menu will prompt you to select a dimension key level. Choose “Product Key.”
6. Click OK. All dialogue boxes will close.
7. Expand the Dimensions node. DimProduct should now be a dimension in the OLAP server. Expand DimProduct to view its fields. They key field should have a small gold key icon next to it.
8. Repeat for DimCustomer
Create Date Dimension

Drag DimDate from the Data Warehouse window onto the AdventureCubes Dimensions node. DimDate will immediately appear under Dimensions. This is because TX DWA knows that DimDate is a date table and configures it automatically.

A date dimension cannot function in a cube without defining a date hierarchy. TX DWA automatically does this for us. The “Calendar” node is automatically created by TX DWA.

Take a close look at the “Calendar” hierarchy. This is a good example of an OLAP dimensional hierarchy. The dimension levels organized so that the level above always contains the level below. This organization helps the system know how to relate this dimension to other dimensions, and how to aggregate or “rollup” measure data related to the hierarchy. For example, an individual Date rose up to a Month, a Month rose up into a Quarter and a Quarter rose up into a Year.

Even though we can assume that DimProduct should have a rollup of ProductName to ProductModel to ProductSubcategory to Product Category, TX DWA will not know that unless we explicitly tell it to. We are going to do that next.

Create a Hierarchy

1. Right-click on DimProduct and select Add Hierarchy. The Hierarchy window will open.
2. Name the hierarchy Product Hierarchy
3. Select, in order, the levels ProductCategoryName, ProductSubcategoryName, ProductModelName, and ProductName.
4. Click OK.
5. The new hierarchy will appear at the bottom of the dimension levels list.
Adding Dimensions to a Cube
Now that we have defined dimensions for the OLAP database, we can now attach them to cubes. All that we have to do is to drag the dimensions onto the cube Dimensions node.

Add Product Dimension to the Cube
Ensure that the Sales cube is expanded so that you can see the Sales cube dimensions node. Drag DimProduct onto the Sales cube dimensions node.

Dim Product will appear on the Sales cube dimension node followed by “show dimension”. This means that the dimension is not related to any measure the cube.

Relating the Product Dimension to a Measure

We must now relate Dim Product to a measure table. We do this by:

1. Right clicking on DimProduct in the Sales cube and select Dimension Relations.
2. Under Dimension Relations, choose either all fact tables or FactInternetSales. Since the cube only draws on one fact table at the moment, the result will be the same.

The Dimension Relations dialogue window opens up. The Dimension Relations dialogue window consists of three columns. From right to left, there is a list of dimension levels, their key values, and dropdown menus to their right.

Note that for most dimensions, key column values and dimension levels are the same. TX DWA date dimensions are an exception to this.
The key value for the dimension is indicated by bold text and followed by the parenthetical (KEY). All dimension levels should have a red circle around a white exclamation mark. This is a warning that the fields are not related to a measure yet.

3. Find the Key level for DimProduct, in this case, ProductKey.

4. Open the dropdown menu for ProductKey to the right. This menu will display all of the fields in the FactInternetSales table.

5. Select ProductKey from the dropdown menu to relate DimProduct to FactInternetSales on ProductKey.

6. Click OK. The dialogue box will close, and DimProduct will no longer be classified as a shell dimension.

7. **Repeat this process for DimCustomer.**

---

**Roleplaying Dimensions in the Cube**

As was mentioned in the first part of this document, some dimensions play more than one role. The date dimension is a good example of this. In the relational database, we created separate views for each kind of date dimension we needed for the FactInternetSales table. The views that we created were DimDueDate, DimOrderDate and DimShipDate. We will need to do the same thing for the Sales cube. We will create three views of the date dimension by:

1. Add DimDate to the Sales cube by dragging in from the AdventureCube Dimensions list onto the Dimensions node under the Sales cube.
2. Right-click the new copy of DimDate and select Edit Cube Dimension. A dialogue box will appear.
3. Under Name, rename the dimension DimDueDate.
4. Click OK. The dialogue box will close. Note that the base table for the roleplaying dimension DimDueDate appears in parenthesis after the dimension name.
5. Right click DimDueDate and Select Dimension Relations and then click FactInternetSales. The Dimension Relations dialogue box will appear.
6. Relate DimDueDate to FactInternetSales on Date = DueDate
7. Click OK
8. Repeat this process to create roleplaying date dimensions for DimOrderDate and DimShipDate
9. Deploy & execute
The Finished Cube
The finished cube should contain the following:

1. One OLAP Server, AdventureCubes
2. One Cube, Sales
   a. Two Standard Measures, Total & GrossLineTotal
   b. One Business Function Measure, Discount Percentage Calculation
   c. One Calculated Measure, Discount Amount
   d. Two ordinary dimensions, DimCustomer and DimProduct
   e. Three roleplaying date dimensions, DimDueDate, DimOrderDate and DimShipDate
3. Three base OLAP dimensions, DimCustomer, DimProduct, and DimDate

Exploring the Cube
TX DWA has a function that allows us to browse the cube without leaving the TX DWA interface

1. Right-click the Sales cube
2. Select Browse Cube. The cube browser will open
Cube Browser Window
The cube browser window is shown on the right. It has three main parts:

1. On the left is the display area, that will show the cube browsing result
2. On the top right is a list of cubes, with their associated measures, levels, and hierarchies
3. On the bottom right is the area that measures and levels can be dragged into the build a view of the cube

4. Find Total under Sales cube Measures
5. Drag Total into the Measures box in the lower right
6. The total sales will appear in the display area

Without context, the measure by itself is of limited use. Let’s add a dimension by:
7. Find ProductCategoryName in DimProduct.
8. Drag ProductCategoryName into the Rows box in the lower right.
9. DimProduct will appear as a row heading. Click the plus next to DimProduct to see the level you’ve chosen.
10. The display area now shows total sales broken down by product category.

Let’s add a date dimension so that we can see product category trends over time.

11. Find DimOrderDate and open it. Notice that fields like Year or Week are not shown; instead it shows the Calendar hierarchy. Using this hierarchy to view the cube will automatically configure date how we want it.
12. Drag DimOrderDate.Calendar into columns.
13. In the display area, expand All DimDate by clicking on the plus sign. Resize your window as needed.
14. You are now viewing total sales by product category over time.

Try adding Customer.Title to rows under ProductCategoryName. This will allow you to see all of the previous data by customer demographic where available.
Once you've done this, you will have a three dimensional cube! Note that dimension levels and hierarchies can be dragged between the row and column boxes to “pivot” the cube and show a new way of viewing the data.

A Three Dimensional Cube
Here is the three dimensional cube. With just this simple browsing interface, business people can get a lot of good information.
Appendix 1: Download and Install Sample Data

The AdventureWorks2012 database can be downloaded for free online from the following link: [http://msftdbprodsamples.codeplex.com/downloads/get/478214](http://msftdbprodsamples.codeplex.com/downloads/get/478214). When you download be sure you select the 2012 data version, NOT the DW or the LT version of the database.

Follow these steps:

1. Download a database for your SQL Server version
2. Unzip the database (mdf) file and log (ldf) file
3. From Microsoft SQL Server Management Studio (SSMS), connect to a SQL Server instance
4. Right click Databases
5. Click Attach
6. Click the Add button
7. Locate the AdventureWorks database mdf file. For instance, AdventureWorks2012_Data.mdf
8. Click the OK button on the Locate Database Files dialog window
9. Click the OK button on the Attach Databases dialog window to attach the database
   a. Note: You may need to unlock the .mdf file in file properties in Windows Explorer because Windows automatically restricts sources downloaded from the Internet
Appendix 2: Create TemplateDWH Script

Follow these steps:

1. Open a new query in SQL Server Management Studio
2. Copy and paste the SQL code below into the open query window
3. Hit F5 or press Execute

```
--Create Database
CREATE DATABASE [TemplateDWH]
GO
USE [TemplateDWH]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO

--Create DimCustomer
CREATE TABLE [dbo].[DimCustomer](
    [CustomerKey] [int] NOT NULL,
    [Title] [nvarchar](8) NULL,
    [FirstName] [nvarchar](50) NULL,
    [MiddleName] [nvarchar](50) NULL,
    [LastName] [nvarchar](50) NULL,
    [BirthDate] [date] NULL,
    [MaritalStatus] [nchar](1) NULL,
    [Gender] [nchar](1) NULL,
    [EmailAddress] [nvarchar](50) NULL,
    [AddressLine1] [nvarchar](120) NULL,
    [AddressLine2] [nvarchar](120) NULL,
    [Phone] [nvarchar](20) NULL) ON [PRIMARY]
GO

--Create DimEmployee
CREATE TABLE [dbo].[DimEmployee](
    [EmployeeKey] [int] NOT NULL,
    [FirstName] [nvarchar](50) NOT NULL,
    [LastName] [nvarchar](50) NOT NULL,
    [MiddleName] [nvarchar](50) NULL,
    [Title] [nvarchar](50) NULL,
    [HireDate] [date] NULL,
    [BirthDate] [date] NULL,
    [LoginID] [nvarchar](256) NULL,
    [EmailAddress] [nvarchar](50) NULL,
    [Phone] [nvarchar](25) NULL,
    [MaritalStatus] [nchar](1) NULL,
    [Gender] [nchar](1) NULL,
    [DepartmentName] [nchar](50) NULL,
    [Status] [nvarchar](50) NULL) ON [PRIMARY]
GO

--Create DimProduct
CREATE TABLE [dbo].[DimProduct](
    [ProductKey] [int] NOT NULL,
    [EnglishProductName] [nvarchar](50) NOT NULL,
    [StandardCost] [money] NULL,
    [FinishedGoodsFlag] [bit] NOT NULL,
    [Color] [nvarchar](15) NOT NULL,
    [ListPrice] [money] NULL,
    [ProductLine] [nchar](2) NULL,
    [DealerPrice] [money] NULL,
    [Class] [nchar](2) NULL,
    [Style] [nchar](2) NULL,
    [ModelName] [nvarchar](50) NULL,
    [EnglishDescription] [nvarchar](400) NULL,
    [Status] [nvarchar](7) NULL) ON [PRIMARY]
GO

SET ANSI_PADDING OFF
```
Appendix 3: Import Business Function Library

Follow these steps:

1. Open a browser and go to:

2. Download the “Business Function Library [V.].XML”

3. In TX DWA, navigate to the Tools tab in the ribbon

4. Click on Import Business Functions in the Business Functions and SQL Snippets section

5. Find the downloaded file and check the Select All option
Appendix 4: Completed Sprint 2

Your solution should look similar to screenshot below, if you have followed the guide.
Completed Sprint 2 Staging Tables
Completed Sprint 2 DW Tables & Views
Additional Resources

**TimeXtender**
TX TWA Trial: [http://go.timextender.com/tx-dwa-trial](http://go.timextender.com/tx-dwa-trial)
Support site: [https://support.timextender.com](https://support.timextender.com)

**Agile Methodologies**