University of Washington

iSchool Info 340

# Module 07 - Data Warehousing and Reporting

In this module, we will look at what a data warehouse is and how it is used for reporting**.**

## Outline

Here is a general outline of what we will be doing this module:

|  |
| --- |
| **Module07: Data Warehousing and Reporting Structures** |
| Session01 Lectures and Labs < 110 mins |
| Data Warehousing and Reporting - 20  Investigate and Identify Data - 10  Lab 1: Investigate and Identify Data – 15  Plan and Create a Data Warehouse Reporting Database – 20  Create the Data Warehouse - 10  Create an ETL Process- 10 |
| Session02 - Lab |
| Lab 2: Working on the Final Milestone 1 - 50 |
| Session03 Lectures and Labs < 110 mins |
| Create Cubes, Tabular Structures, or Non-SQL databases – 40  Creating Reports - 30  Test and Tune the Solution - 10  Approve, Release and Prepare - 10 |

**Note**: Times are only estimates and may change without notice!

# Session01 < 110 mins

In this session, we explore **how reporting databases (OLAP) differ from transaction databases (OLTP).** We will also look at **how developers create reporting solutions** that use OLAP databases.

## Data Warehousing and Reporting

**Transactional** databases focus on you **inserting, updating, and deleting data**. They are all about storing and managing data. **Analytical processing databases are about extracting information from that data by creating reports.**

A **typical pattern** in the industry begins with **simple** reports and **moves** progressively **toward** more **complex** ones over time. Many companies **start by selecting report data directly from OLTP relational tables** **but** **end up** with performance issues as the need to get report data to contend with the need to process transactions. When that happens, each company begins the process of **making changes to enhance reporting performance.**

Here are some common steps in this progression:

1. Use **existing** tables and **ad-hoc** select queries.
2. Upgrade to using **stored procedures**, **functions**, and **views**.
3. Copy the database, as is, to a **dedicated reporting database** and use it for reports.
4. Modify the reporting database to use a **dimensional design** and use it for reports.
5. Copy data to **specialized reporting databases** (cubes, tabular, non-relational/no-SQL).

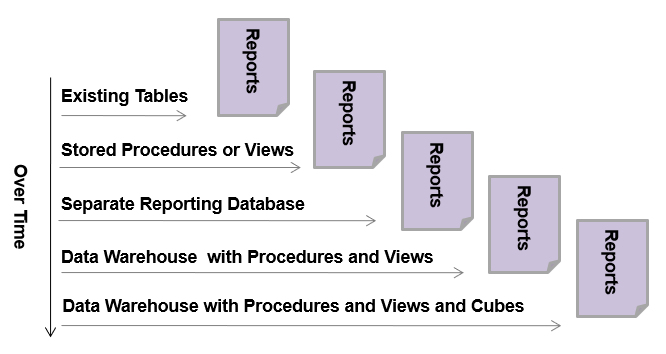


Figure: A common progression of reporting strategies

A **Data Warehouse** (otherwise known as Data Marts, Data Factories, Data Silos, etc.) **is any database that is designed specifically for storing report data**. Over the year these **design**s have materialized into a set of **rules and techniques known as the** "**Dimensional Model**."

### BI Solution Components

The **components** that make up a BI reporting solution **include**:

* A **source** of report data
* A **destination** for report data (DW, Cube, Tabular, Non-SQL)
* An **ETL** process
* A set of **reports**
* **Documentation**

If we use Microsoft's **SQL Server as an example**, the solutions might consist of following servers and objects:

* **SQL Server** to hold **transactional source** data in one or more databases
* **SQL Server** to hold **dimensional destination** reporting data in a data warehouse database
* **Analysis Server** to hold report data in a **cube or tabular database**
* A set of **reports** using Microsoft **Excel**, **SSRS**, or **Power BI**

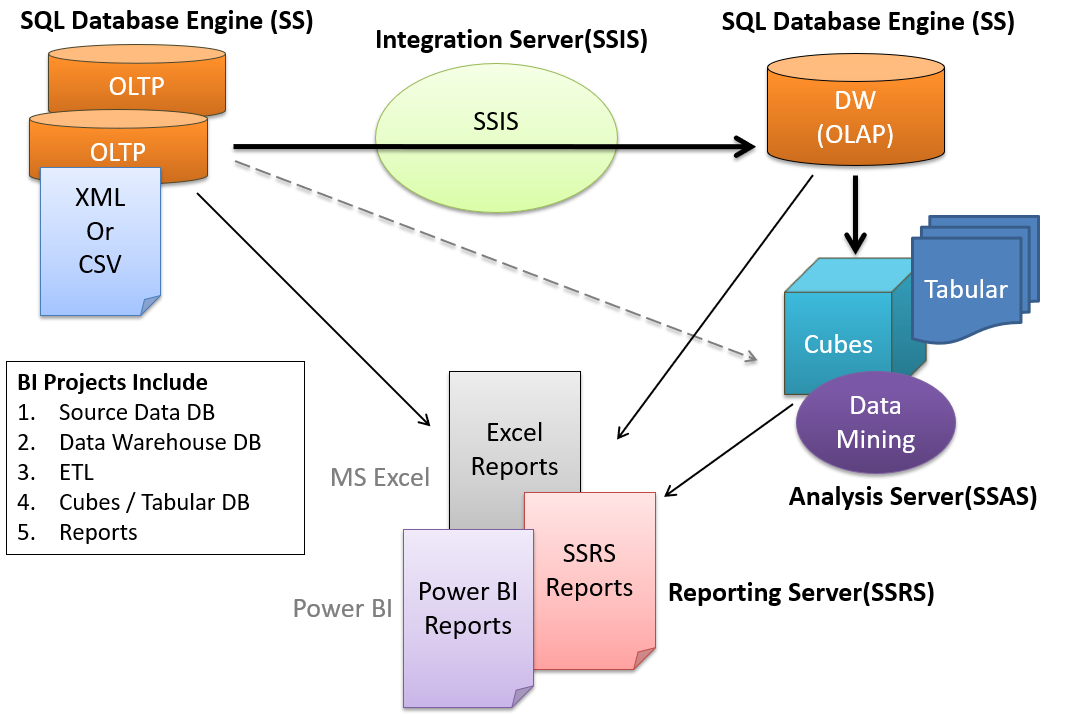


Figure: A BI solution using Microsoft components

### Creating Business Intelligent Reporting Solutions

**A BI Solution is a set of components** (databases, tables, views, stored procedures, ETL scripts, reports, etc.) **that allows data to be turned into useful information**. These components must be designed, created, tested, and ultimately approved to develop a working BI solution.

**To create** a reporting solution, you must **understand what component** the solution consists of, **how each component is combined** to create the whole, and ﬁnally, how to recognize when you have achieved your goal.

Here is an outline of **eight common steps for creating a BI reporting solution**.

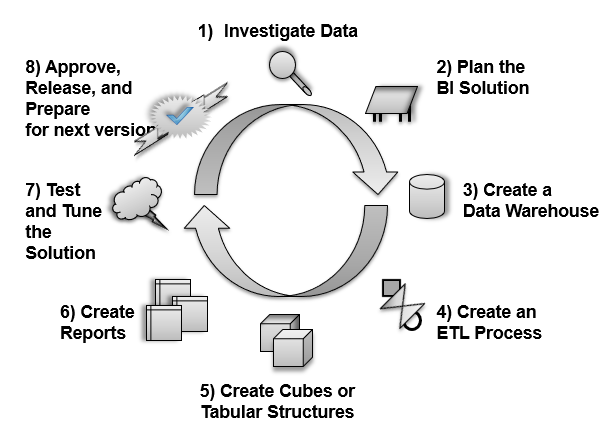


Figure: Eight common steps in creating a BI solution

## Investigate and Identify Data

This step begins with you **determining** **what** type of information is **needed,** **and where** the data for your needs is **located**. Data can be found in many forms, and you may have to use one or more sources to meet your requirements. Some **common data sources include** the following:

* Existing databases
* Spreadsheets
* Text, Log, or XML ﬁles
* Paper documents

Once the data is located, you also need to **decide whether** the **current data is sufﬁcient** for your needs **and** whether you really **need all** the data **or** **just some of the data**. Don't just assume that creating a copy of all the data is better than copying only select datathat would be useful for most reporting needs. Overloading a reporting solution with seldom-used data is the most common way to cripple reporting performance.

As you think about what to include, **prioritize** what **data** must be in **this version** of the solution (there will **likely be other versions you can add data**!) One method of doing this is to use a technique known as "Four Quadrant Prioritizing."

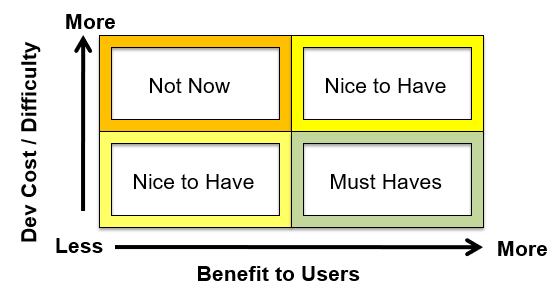


Figure: A Developer's Four Quadrant Prioritizing Matrix

Identify what is the value to your solution versus the difficulty and cost to include in this version. When data provides **substantial benefit** to users, at a **low to medium** development **effort**, you can classify this data is a **Must-Have**.

On the other hand, if data does **not** provide **much benefit** to the users and is **difficult or costly to implement,** then it should be **excluded** from the project **at this time**. It does not mean that in the future it won't be included, it just means that right now is a **Not Now** item.

If data provides a **lot of benefit** to the users but may be **costly or difficult**, then it is something that would be considered "**Nice to Have**," but may not be absolutely necessary to the solution.

The final quadrant is one you must be careful of. This quadrant is where it's **easy to implement** but provides **little to no value** for the users. This option is very attractive because choosing these items will keep you quite busy implementing them and will make you feel like you are really accomplishing something. This particular quadrant is the one that is responsible for "feature creep." "**When in doubt leave it out!**"

### Creating a Metadata Worksheet

You need to **document what your solution** will accomplish, the source and the destination of its data, and the components that will make up the solution. A solution's documentation can be laid out with **common tools such as Microsoft Excel or Microsoft Word**.

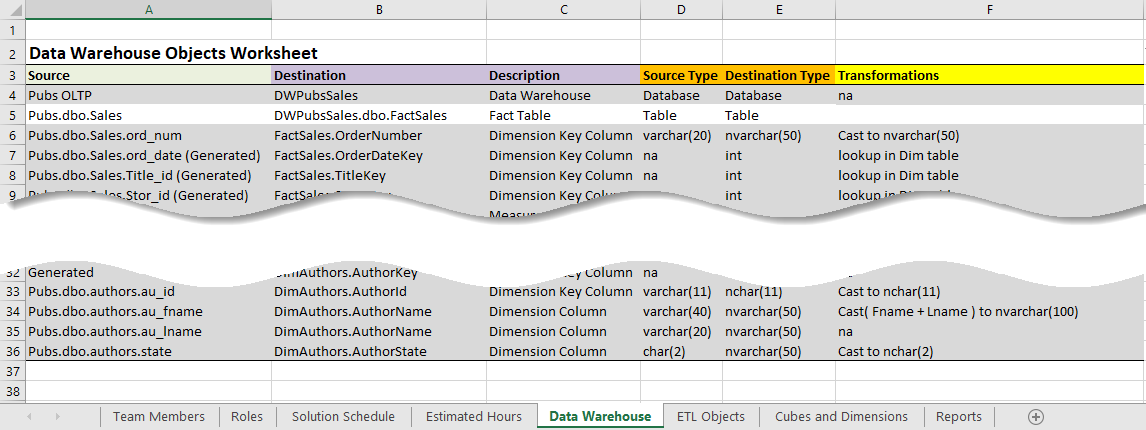


Figure: Using a spreadsheet to document a BI solution

## Lab 1: Investigate and Identify Data

In this lab you will start to create a new BI reporting solution using the Northwind database.

**NOTE**: This lab is different, in that your will **review the data with your instructor** instead of on your own!

### Step 1: Examine some data

Review the data in the following tables looking for the least number of columns you could use to create a reporting database storing ONLY product sales data. We are looking **only for the "Must Haves!"**

-- 1a) Write code to look at the data

Select \* From Northwind.dbo.Orders as o;

Select \* From Northwind.dbo.[Order Details] as od;

Select \* From Northwind.dbo.Products as p;

Select \* From Northwind.dbo.Categories as c;

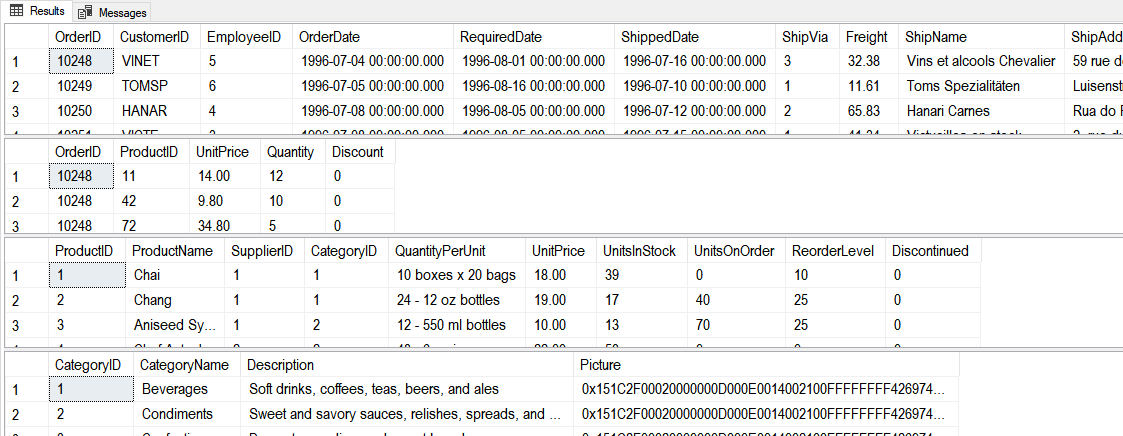


Figure: The results of the previous SQL statements

-- 1b) Write code to select the minimum you want

Select o.OrderID, o.OrderDate

From Northwind.dbo.Orders as o;

Select od.OrderID, od.ProductID, od.Quantity, od.UnitPrice as [OrderedPrice]

From Northwind.dbo.[Order Details] as od;

Select p.ProductID, p.ProductName, p.CategoryID, p.UnitPrice as [StandardPrice]

From Northwind.dbo.Products as p;

Select c.CategoryID, c.CategoryName

From Northwind.dbo.Categories as c;

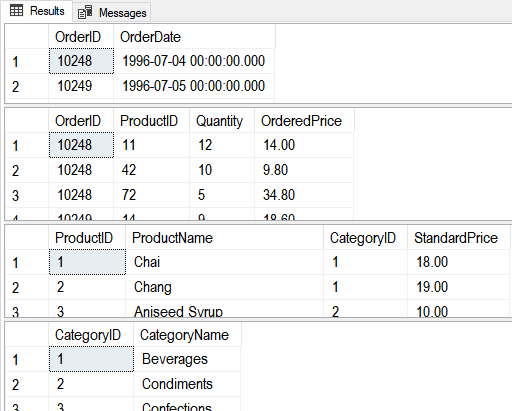


Figure: The results of the previous SQL statements

-- 1c) Write code to identify the data types of the minimum you want

use Northwind;

go

Select TABLE\_NAME, Column\_Name, Data\_Type, CHARACTER\_MAXIMUM\_LENGTH

FROM [INFORMATION\_SCHEMA].Columns

Where table\_name in ('Orders','Order Details','Products','Categories')

And COLUMN\_NAME in ('OrderID', 'OrderDate','Quantity','UnitPrice'

,'ProductID','ProductName','UnitPrice'

,'CategoryID','CategoryName');

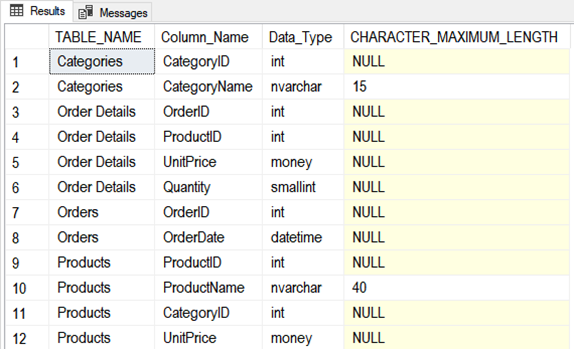


Figure: The results of the previous SQL statement

### Step 2: Start the Solution Document

Review a solutions metadata spreadsheet, by **opening** the **Mod07 Labs BISolutionWorksheets.xlsx** **file** and reviewing the metadata about the chosen source data (the source's name and data type).

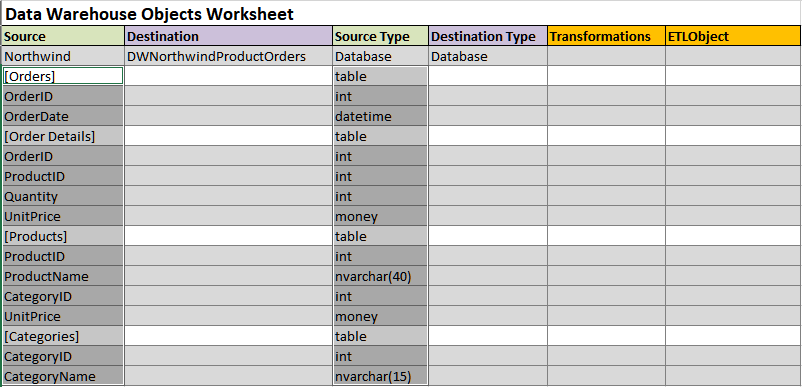


Figure: The spreadsheet for step 2 is completed

### Step 3: Review Your Work

Now, you will review your work with your instructor.

## Plan Data Warehouse Reporting Database

Once you have chosen your source data, you can start to **design a data warehouse OLAP database** to hold a copy of that data in tables **specifically** designed **for reporting performance**.

**OLAP databases** are designed using a **dimensional model**, which **divides data into "measures" and "dimensional attributes**." You **start** your database design by **organizing the measures and dimensional attributes**.

### Measures

Reports are created to provide information about measured facts. These **"measures" describe a process, such as the process of selling of items**. Measure **values**, **such as a quantity items sold** for example, provides **measurable facts about a specific event within that process**. Measurable fact are ones that **can be summed, averaged, or in some way calculated** to provide additional aggregated information about a process.

For **example**, if we look at the sales table in the Pubs database, you can see that the **Qty column** can be used as a measure. The sales quantity can be summed or averaged to provide more sales information.

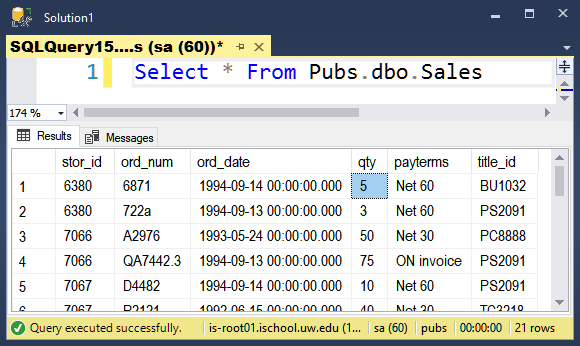


Figure: Data in the Pubs Sales tables

### Dimensions

Unlike the sales quantity in our example, the **Store IDs** (stor\_id) data is **not measurable** (even though they are using numeric values.) Instead, these values **are descriptive** in that they describe the store a given quantity sold by that store. **Descriptive values are known as "dimensional attributes,"** not measured data. The **Stor\_ID, Stor\_Name, City, and State** columns of the pubs.dbo.stores tables would **all** be **examples of dimensional attributes**.

### Star and Snowflake designs

**While the measures** are stored **in a Fact table,** and the **dimensional data** are stored **in** one or more **Dimension tables**. **Dimensional Keys link** the Fact and Dimension tables via a **Primary Key to Foreign Key relationship**.

You can **choose between two options for creating dimensional tables. These options are known as the** **star versus snowﬂake dimensional designs.**

**Star and snowflake** designs **refer to the pattern** formed **between dimension tables and** **a fact table**. (I know! Neither looks much like their names!)

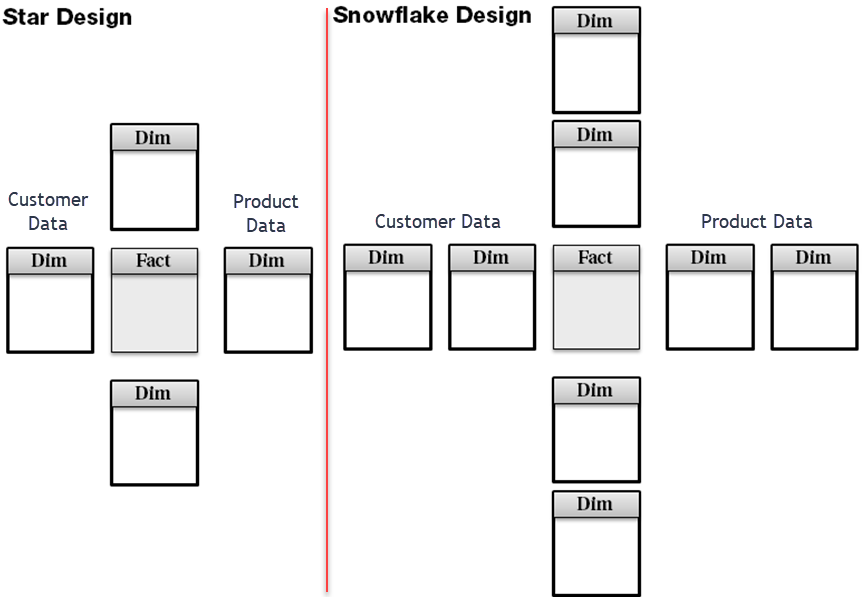


Figure: Star and Snowflake table patterns

A **better name** for these designs would have been "**single-table** **dimensions" (for star)** and "**multi-table** **dimensions" (for snowflake), since the number of tables** used to hold the dimensional attributes **on a given subject determines a star or snowflake design**.

When **all** the **dimensional attributes** **for a single subject, such as products or customers, are stored in one table**, it is known as a **Star** design.

Dimension containing **multiple tables** per dimension **are** a snowflake designs. **Snowflakes** form a circle of **two or more** tiers of **linked** **dimensional tables about a single subject** around the fact table of a data mart. Whether it forms a circle of three tiers, four tiers, or hundreds of tiers, it still is a snowflake design.

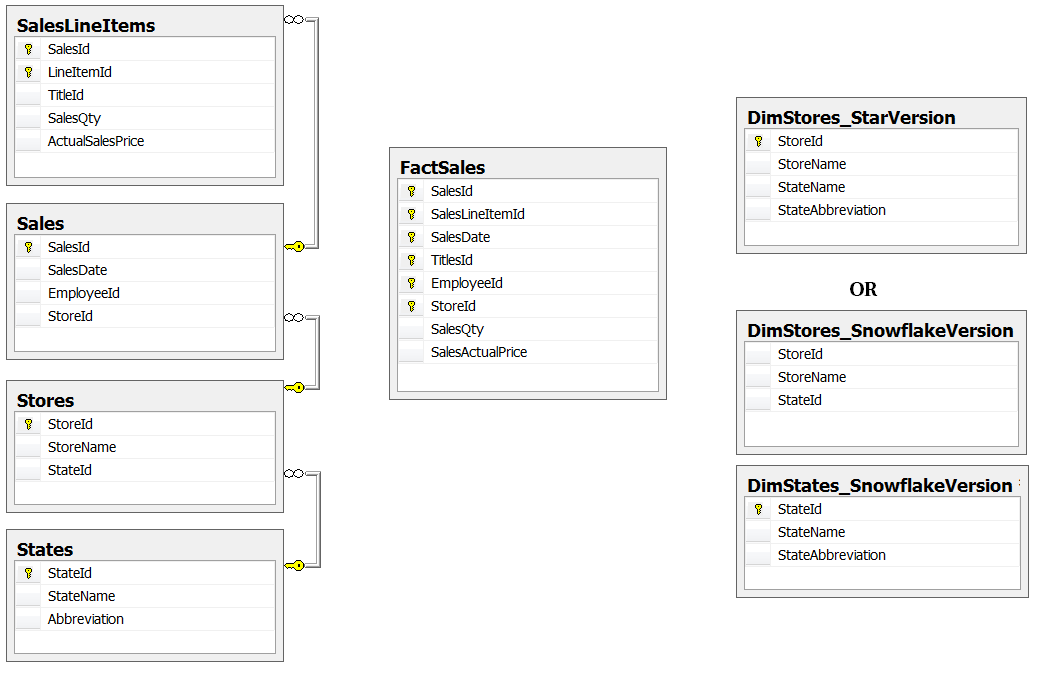


Figure: OLTP tables alongside of their Snowflake or Star dimensional counterparts

**In most cases**, just collapse a set of tables into a **single, "star dimension" table**. This is **more convenient** to build reports from then a snowflake design since fewer tables are needed to extract report data.

Many data warehouses end up with a **hybrid** with **some** of the dimension tables designed in the **stars** design pattern and **other** dimension tables designed in the **snowflake** design pattern. When possible use the **star design for simplicity**. If you come across a circumstance where you need to **reduce redundancy**, change the design to a **snowflake**. In the end both are simply tools, so choose the tool that is appropriate for your needs.

## Create an Data Warehouse Reporting Database

When the planning is done, it's time to create the data warehouse. There are several techniques to accomplish this, including SQL code, a table designer, and a diagramming tool that will let you visually create the data warehouse objects. Of course, the required SQL code should be easy for you at by this module. Here is an example of a very simple version of the Pubs data warehouse.

Use [master];

go

If Exists(Select Name from SysDatabases Where Name = 'DWPubsSales')

Begin

Alter Database DWPubsSales Set Single\_user With Rollback Immediate;

Drop Database DWPubsSales;

End

go

Create Database DWPubsSales;

go

Use DWPubsSales;

go

/\*\*\*\*\*\* Create the Dimension Tables \*\*\*\*\*\*/

Create Table [dbo].[DimStores](

[StoreId] [nchar](4) NOT NULL Primary Key,

[StoreName] [nvarchar](100) NOT NULL,

[StoreState] [nChar](2) NOT NULL);

go

Create Table [dbo].[DimTitles](

[TitleId] [nvarchar](6) Primary Key NOT NULL,

[TitleName] [nvarchar](100) NOT NULL,

[TitleType] [nvarchar](100) NOT NULL,

[TitlePrice] Money NOT NULL,

[PublisherID] int NOT NULL,

[PublisherName] [nvarchar](100) NOT NULL,

[PublishedDate] [date] NOT NULL);

go

/\*\*\*\*\*\* Create the Fact Tables \*\*\*\*\*\*/

Create Table [dbo].[FactSales](

[OrderNumber] [nvarchar](100) NOT NULL,

[OrderDate] [date] NOT NULL,

[TitleID] [nvarchar](6) NOT NULL,

[StoreID] [nchar](4) NOT NULL,

[SalesQuantity] [int] NOT NULL, -- Measure

Constraint [PK\_FactSales] Primary Key

([OrderNumber] ASC, [OrderDate] ASC, [TitleID] ASC, [StoreID] ASC));

go

/\*\*\*\*\*\* Add Foreign Keys \*\*\*\*\*\*/

Alter Table [dbo].[FactSales] With Check Add Constraint [FK\_FactSales\_DimStores]

Foreign Key ([StoreID]) References [dbo].[DimStores] ([StoreID]);

go

Alter Table [dbo].[FactSales] With Check Add Constraint [FK\_FactSales\_DimTitles]

Foreign Key ([TitleID]) References [dbo].[DimTitles] ([TitleID]);

go

## Create an ETL Process

**E**xtracting the **data** from its **original** location, **t**ransforming **the data to be consistent with your new data** warehouse design, and **l**oading **the data into** the new **data warehouse** location, is known as **ETL processing**. You create an ETL process using a **combination** **of** programming, such as **SQL, Java, Python, C#**, etc., and/or **specialized tools,** such as SQL Server's Integration Server (**SSIS**).

Often, you **use SQL code to do the bulk of your ETL work**, since it is specifically designed to manipulate large amounts of data. Here is an Example of some simplistic ETL code (In real life this code would be much more complex).

-- Step 1) Code used to Clear tables (Will be used with SSIS Execute SQL Tasks)

Use DWPubsSales;

--1b) Clear all tables data warehouse tables

Delete From dbo.FactSales;

Delete From dbo.DimTitles;

Delete From dbo.DimStores;

Go

-- Step 2) Code used to fill tables (Will be used with SSIS Data Flow Tasks)

Insert into DimStores

Select

[StoreId] = Cast(stor\_id as nChar(4))

, [StoreName] = Cast(stor\_name as nVarchar(100))

, [StoreState] = Cast(state as nChar(2))

From pubs.dbo.stores;

Go

Insert into DimTitles

Select

[TitleId] = t.title\_id

, [TitleName] = Cast(t.title as nvarchar(100))

, [TitleType] = Cast(IIF([type] = 'UNDECIDED', 'undecided', [type] ) as nvarchar(100))

, [TitlePrice] = [price]

, [PublisherID] = p.pub\_id

, [PublisherName] = p.pub\_name

, [PublishedDate] = T.pubdate

From [Pubs].[dbo].[Titles] as T

Join [Pubs].[dbo].[Publishers] as P

On T.[pub\_id] = P.[pub\_Id]

WHERE T.price is NOT NULL;

Go

Insert into FactSales

Select

[OrderNumber] = Cast(ord\_num as nVarchar(50))

, [OrderDateKey] = Cast(ord\_date as Date)

, [TitleID] = title\_id

, [StoreID] = stor\_id

, [SalesQuantity] = qty

From Pubs.dbo.Sales as s;

--Verify the tables are filled

Go

Select \* From DimStores;

Select \* From DimTitles;

Select \* From FactSales;

Go

## Creating the ETL Stored Procedures

You would next create stored procedures that to hold your ETL code like this:

Create Procedure pETLDimStores

/\* Author: RRoot

\*\* Desc: Processes DimStores ETL

\*\* Change Log: When,Who,What

\*\* 2017-01-01,RRoot,Created Sproc.

\*/

As

Begin

Declare @RC int = 0;

Begin Try

Begin Transaction

-- ETL Transaction Code --

-- Step 1) Clear Table

Delete From DimStores;

-- Step 2) Fill Table

Insert into DimStores

Select

[StoreId] = Cast(stor\_id as nChar(4))

, [StoreName] = Cast(stor\_name as nVarchar(100))

, [StoreState] = Cast(state as nChar(2))

From pubs.dbo.stores;

Commit Transaction

Set @RC = +1

End Try

Begin Catch

Rollback Transaction

Print Error\_Message();

Set @RC = -1;

End Catch

Return @RC;

End

go

----Testing Code:

Declare @Status int;

Exec @Status = pETLDimStores;

Select @Status;

Select \* From DimStores;

go

## ETL Applications

If I were **using** a ETL application like **Microsoft's SSIS**, I would **use similar code** to configure an SSIS Package file, but using SQL code **inside** of **Execute SQL Tasks** and **Data Flow Tasks**.

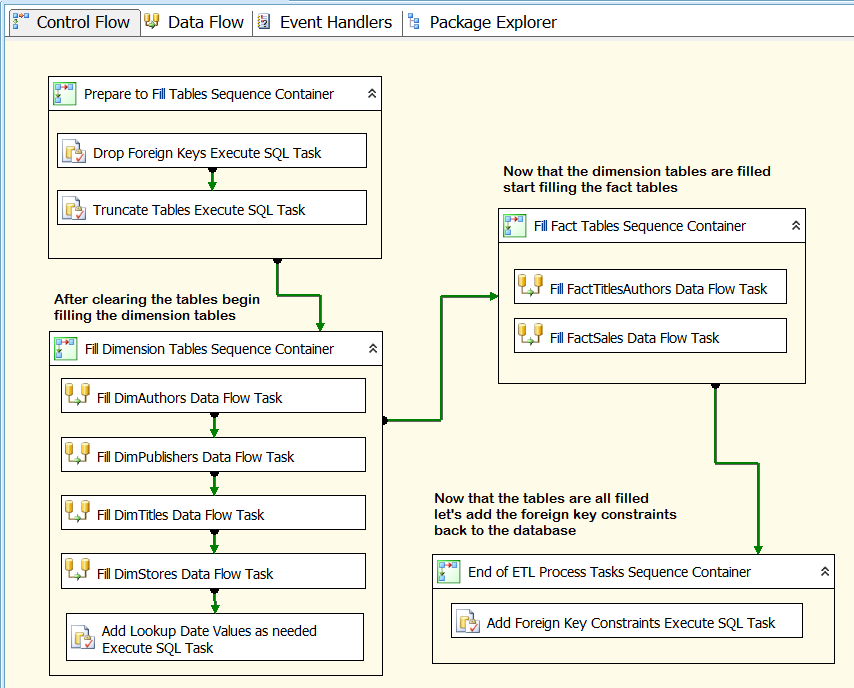


Figure: An ETL SSIS package

"Integration Services includes a rich **set** **of built-in tasks and transformations**; **tools for constructing packages**; and the Integration Services service for running and managing packages. You **can use** the graphical Integration Services tools to create solutions **without writing** a single line of **code**; **or** you can program the **extensive** Integration Services object model to create packages programmatically and **code custom tasks** and other package objects." (<https://docs.microsoft.com/en-us/sql/integration-services/sql-server-integration-services>, 2017)

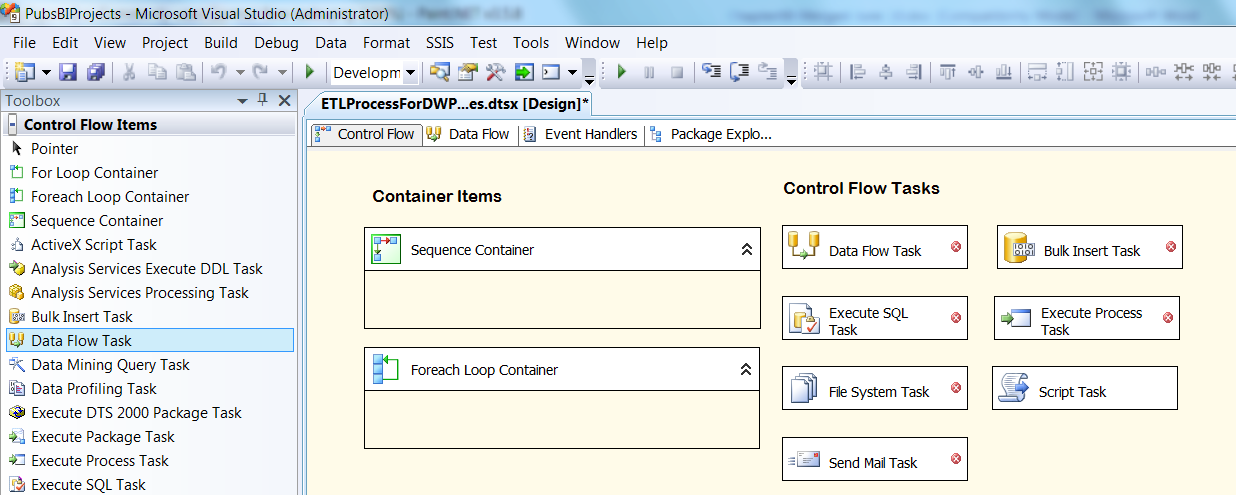


Figure: Commonly used Task in SSIS

# Session02 Lectures and Labs < 50 mins

In this session you will continue to work on the final. You will work on your own for the whole 50 minutes of this lab but may ask questions whenever you would like help.

# Session03 Lectures and Labs < 110 mins

In this session, you will learn more about creating a BI solution.

## Creating Cubes, Tabular Structures, or Non-SQL databases

Microsoft SQL Server includes **two additional** **database options**, both are a part of SQL Server Analysis Services (SSAS). Both the SSAS cube databases and the SSAS tabular databases provide **increased reporting performance**. You can think of these **cubes and tabular structures as a set of many report tables combined into a single reporting object**. **Cube** databases store data on a hard-drive, like SQL Servers relational databases, but in a **file structure optimized** for faster data reads. **Tabular** structures, store their data **in memory** and in a **highly compressed** format. Reporting applications pulling data from a in memory structure will be faster than off a hard drive, even with file optimization, but are **limited by the amount of RAM** the Server has access to.

**Non-SQL** databases **often** store their data on **several hard drives** at once which **spreads out the workload** involved with locating and retrieving report data. This can greatly include **performance when working with BIG DATA sets**, and has become a **popular** option. Like, cubes and tabular, these databases are not designed using the relational or dimension model, providing the benefit of a **less constrained, and perhaps less organized**, storage of data.

**Note**: We will talk about Non-SQL databases and Big Data in a later module.

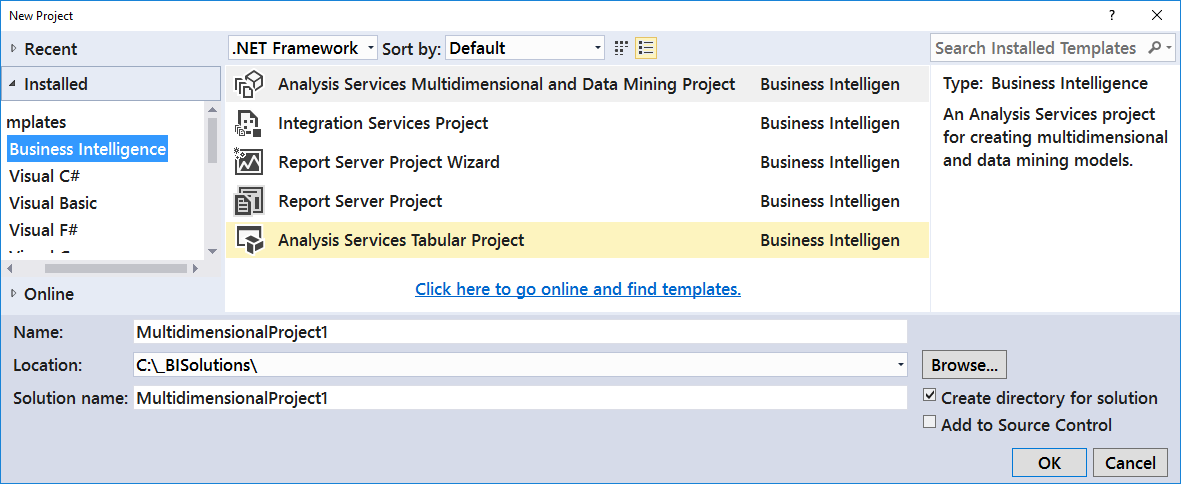


Figure: Creating a SSAS projects in Visual Studio

## Creating Reports

Once you have your data loaded into a data warehouse, cube, tabular structure, or non-SQL database, you need to create **preliminary reports**. Later, users of your solution, will create their own reports, but before you let them do so, these ﬁrst reports are **used for testing and evaluating the accuracy and performance** of your BI reporting.

Microsoft technologies to create your BI reports, including **Excel, SQL Server Reporting Services (SSRS), and Power BI**, but there are **many** other companies that make **reporting software**, such as Tableau.

**Note: We take a deeper look at reporting in Module08**

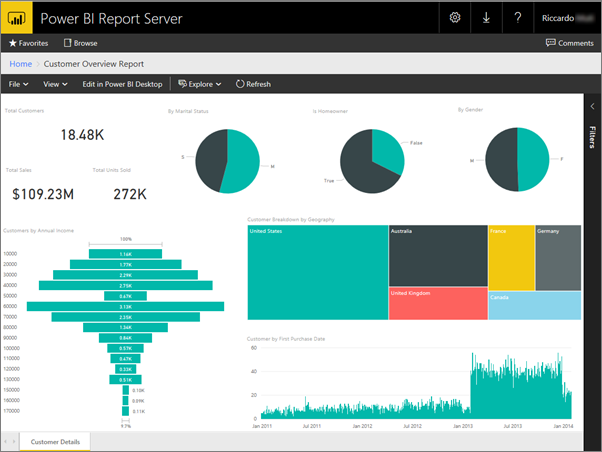


Figure: A Power BI Report (<https://powerbi.microsoft.com/en-us/documentation/reportserver-quickstart-powerbi-report>)

## Test and Tune the Solution

You need to **test** reports for **accuracy, visual consistency, and performance**. The **most important** of the three **is accuracy**. If the reports are slow or do not look professional, it is indeed cause for concern, but if your reports are inaccurate, your entire BI reporting solution will fail!

In **MS SQL Server**, people have been using **SQL Profiler** to test and turn SQL code being executed.

<https://docs.microsoft.com/en-us/sql/tools/sql-server-profiler/sql-server-profiler>

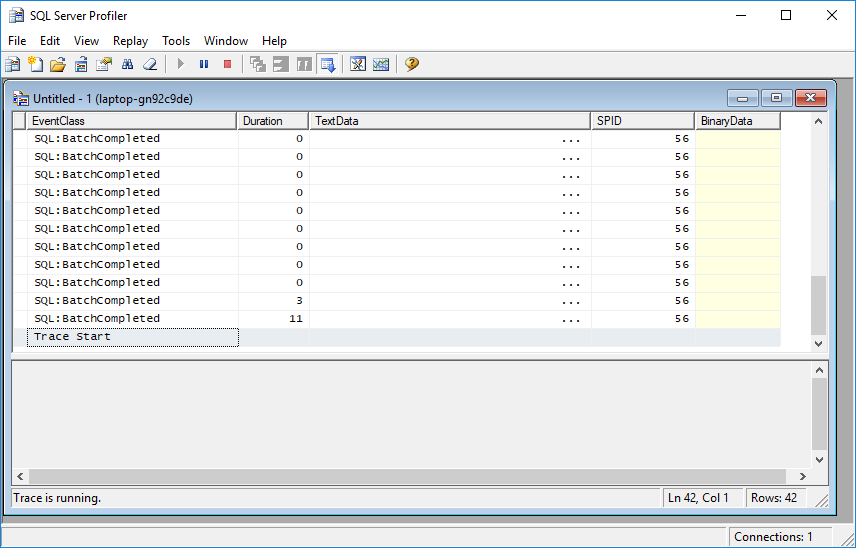


Figure: SQL Profiler user interface

While, this will continue for a while, MS is **replacing Profiler with Extended Events**.

"XE will replace the SQL Profiler in the future versions. By the moment, SQL Server includes Profiler and XE.

The XEs is a feature included in SQL Server 2008. It is a lighter option that consumes less resources than the Profiler. It also can monitor more events than the Profiler. For example, you can monitor Azure, Column Store Events, InMemory OLTP, AlwaysOn Events. In fact, Profiler is not adding new events since the SQL 2008. All the new features are available to be monitored only in XE and not in Profiler." (<https://www.sqlshack.com/is-this-the-end-of-sql-profiler/> , 2017)

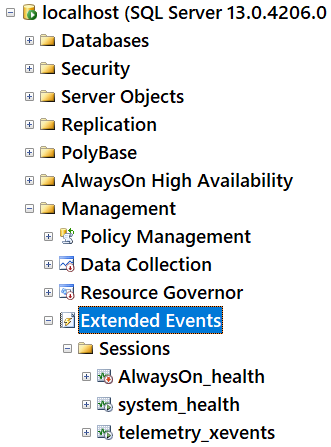


Figure: The Extend Events node in SSMS Object Explorer

Tip: For more information on Extended Events see: <https://docs.microsoft.com/en-us/sql/relational-databases/extended-events/quick-start-extended-events-in-sql-server>

## Approve, Release and Prepare

At the end of the solution development cycle, you need **to package and deploy your documents, scripts, databases, and reports**. You also need to create user documentation, as well as train your users to use your newly developed BI solution.

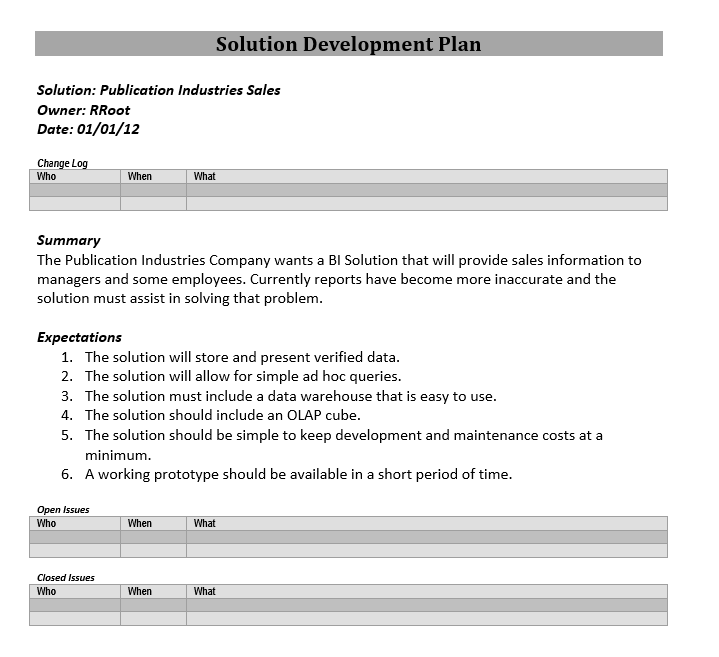


Figure: A formal solution development document

### Starting Again

Here again is an outline of **eight common steps for creating a BI reporting solution**. Each of these steps **should be a bit more clear as to how they are important** in developing a BI reporting solution.

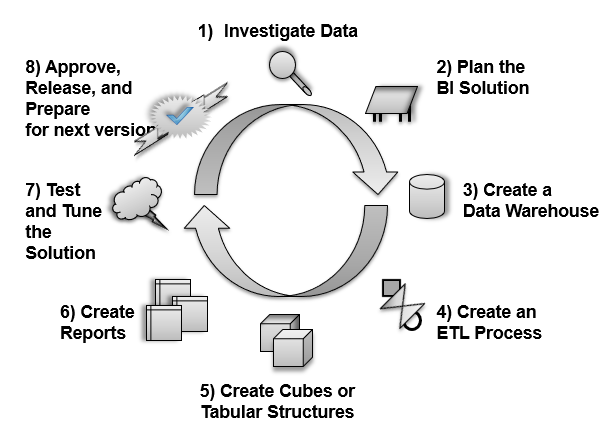


Figure: Eight common steps in creating a BI solution

Here are the examples of how **the components that come with Microsoft SQL Server to support BI reporting solutions**.

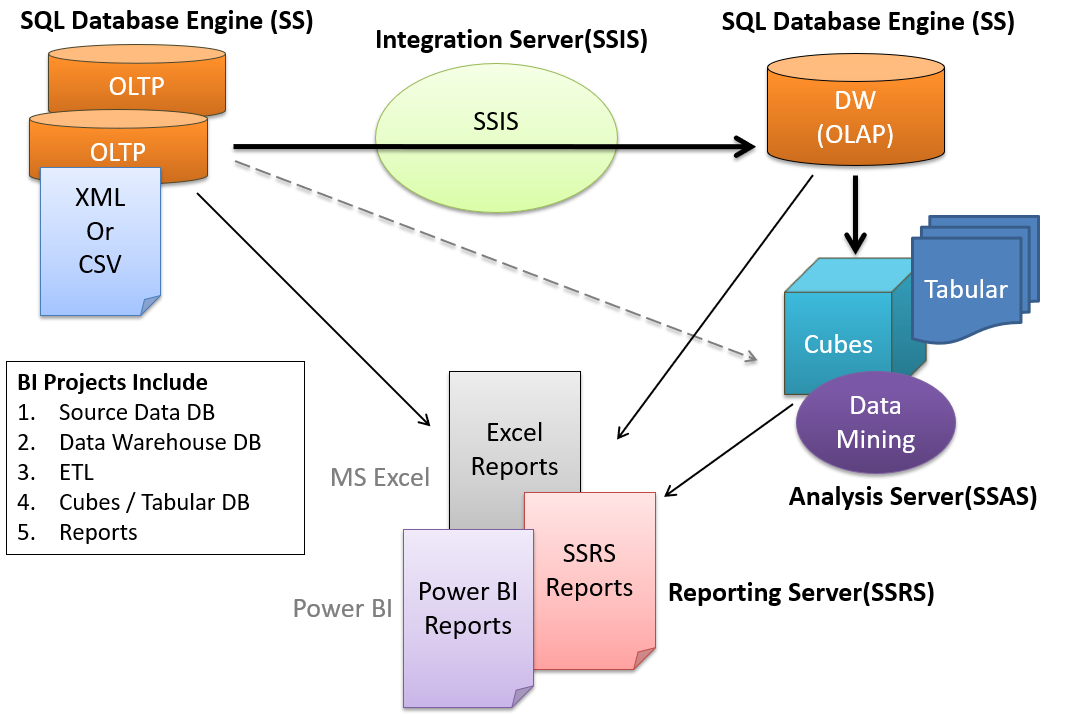


Figure: A BI solution using Microsoft components