Fundamentals of Designing a Data Warehouse

Sensible techniques for developing a data warehousing environment which is relevant, agile, and extensible

Melissa Coates
BI Architect, SentryOne
sentryone.com

MVP Microsoft® Most Valuable Professional

SQL Chick

Blog: sqlchick.com
Twitter: @sqlchick

Presentation content last updated: 2/15/2017

Fundamentals of Designing a Data Warehouse

Agenda

1. Overview of the Need for Data Warehousing
2. DW Design Principles
3. Dimension Design
4. Fact Design
5. When to Use Columnstore or Partitioning
6. DW Tips
7. SSDT 'Database Project' Tips
8. Planning Future Growth of the DW

All syntax shown is from SQL Server 2016.

Screen shots are from SQL Server Data Tools in Visual Studio 2015.

Fundamentals of Designing a Data Warehouse

Out of Scope

- ✓ ETL patterns and techniques
- ✓ Source control
- ✓ Deployment practices
- ✓ Master data management
- ✓ Data quality techniques
- ✓ Semantic layer, OLAP, cubes
- ✓ Front-end reporting
- ✓ Security
- ✓ Tuning & monitoring
- ✓ Automation techniques

Overview of the
Need for Data Warehousing

First Let’s Get This Straight…

Data Warehousing is not dead!

Data warehousing can be “uncool” but it doesn’t have to be if you adopt modern data warehousing concepts & technologies such as:

- ✓ Data lake
- ✓ Hadoop
- ✓ Real-time
- ✓ Large data volume
- ✓ Data virtualization
- ✓ Hybrid & cloud
- ✓ Automation
- ✓ Bimodal environments

Transaction System vs. Data Warehouse

OLTP

Goal:

- ✓ Operational transactions
- ✓ “Writes”

Scope:

One database system

Example Objectives:

- ✓ Process a customer order
- ✓ Generate an invoice

Data Warehouse

Goal:

- ✓ Informational and analytical
- ✓ “Reads”

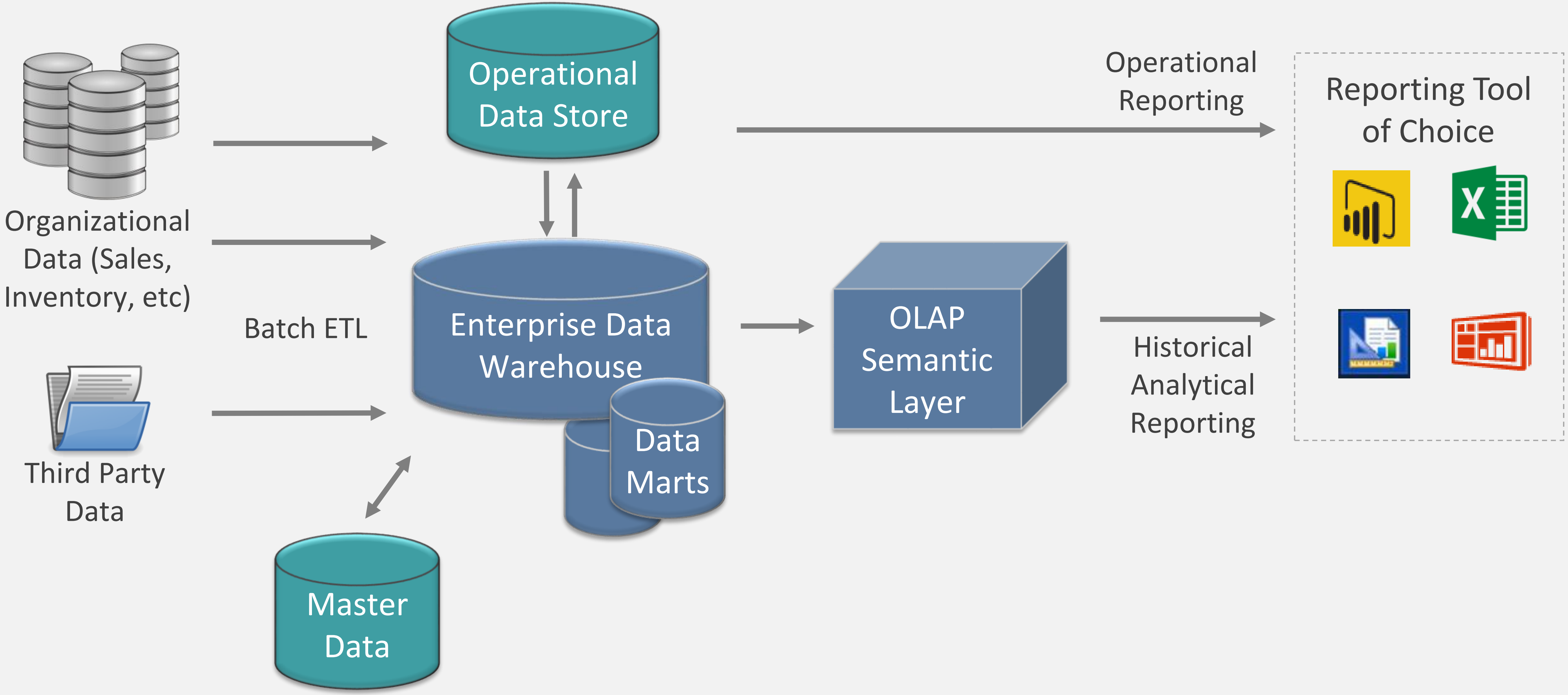
Scope:

Integrate data from multiple systems

Example Objectives:

- ✓ Identify lowest-selling products
- ✓ Analyze margin per customer

DW+BI Systems Used to Be Fairly Straightforward



Organizational Data (Sales, Inventory, etc)

Third Party Data

Batch ETL

Operational Data Store

Enterprise Data Warehouse

Data Marts

Master Data

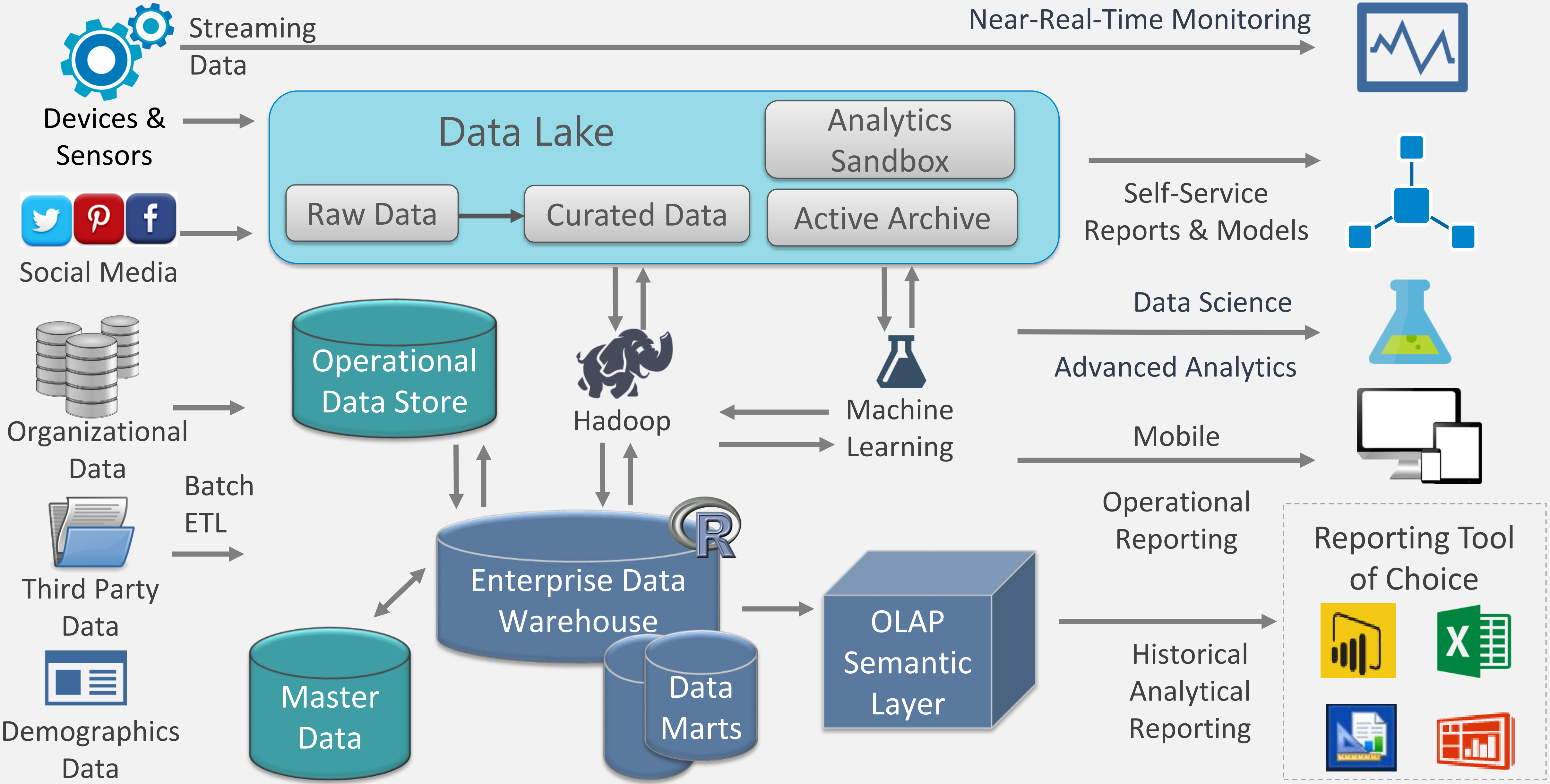
OLAP Semantic Layer

Operational Reporting

Historical Analytical Reporting

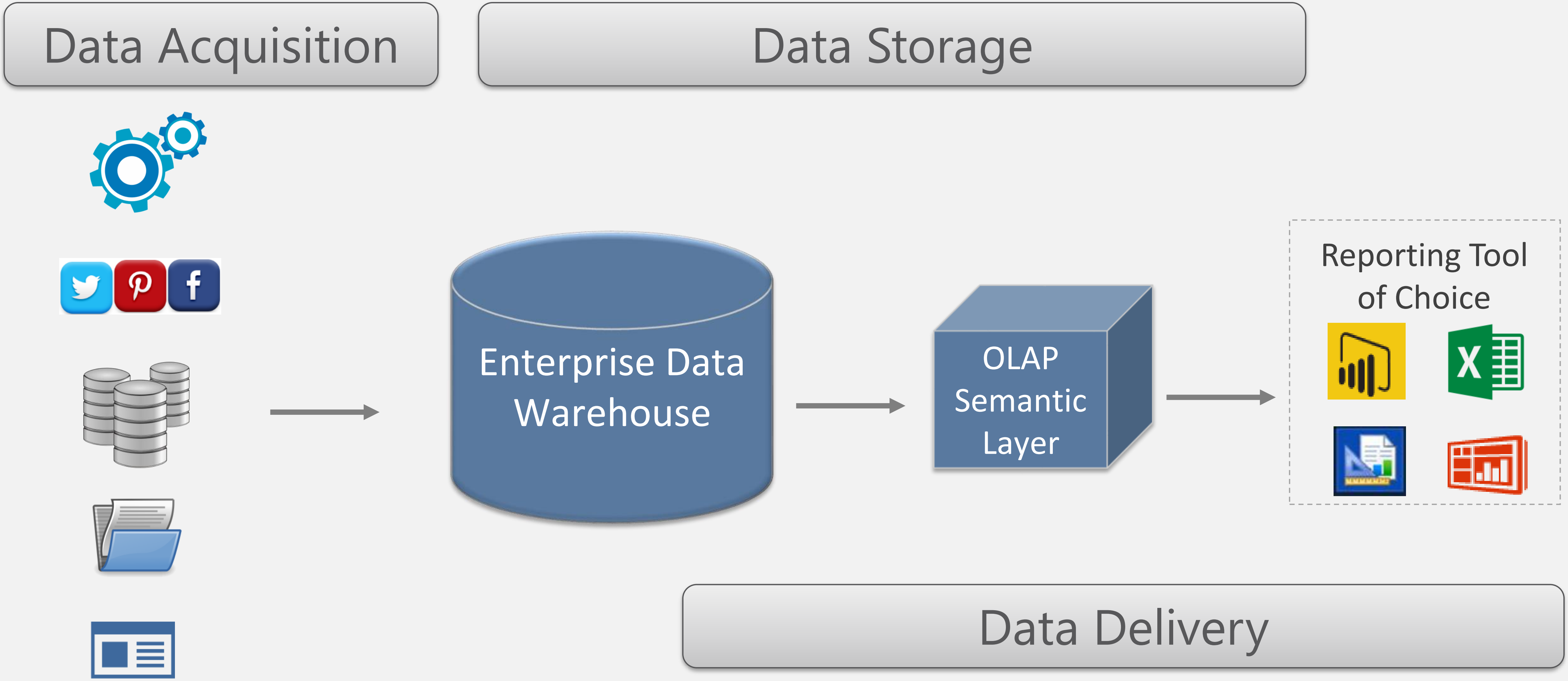
Reporting Tool of Choice

DW+BI Systems Have Grown in Complexity



Data Warehouse
Design Principles

3 Primary Architectural Areas



Data Acquisition

Data Storage

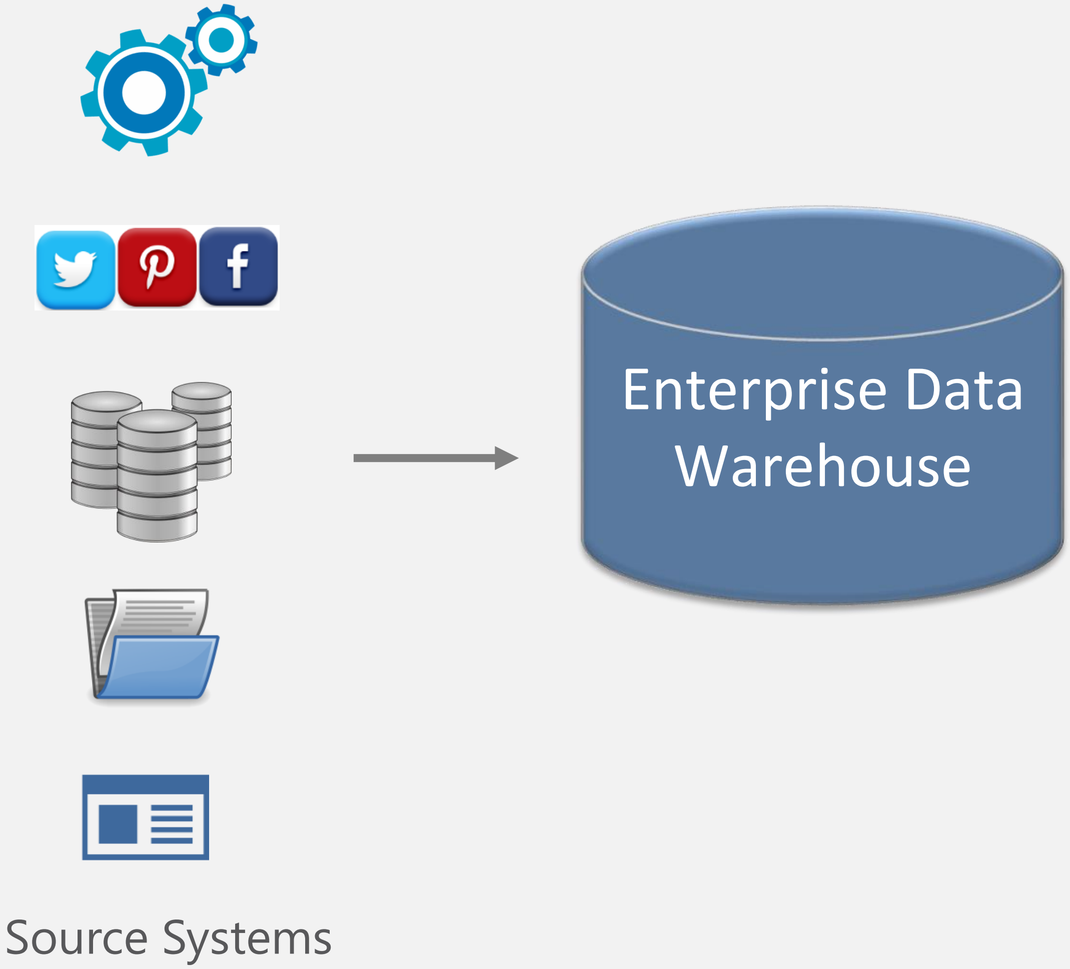
Enterprise Data Warehouse

OLAP Semantic Layer

Reporting Tool of Choice

Data Delivery

Integrate Data from Multiple Sources

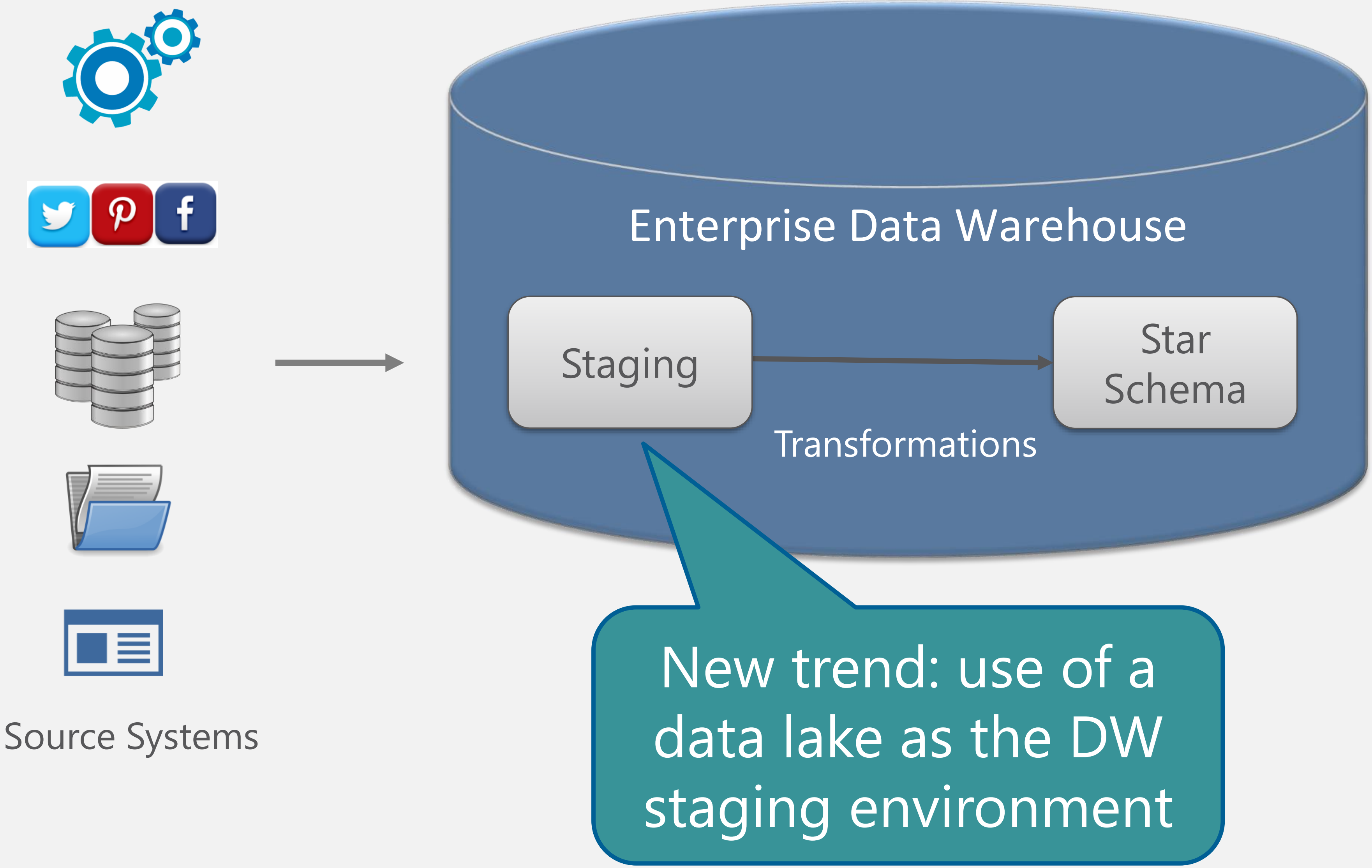


Objective:
Data is inherently more **valuable** once it is integrated.

Example:
Full view of a customer:

- ○ Sales activity +
- ○ Delinquent invoices +
- ○ Support/help requests

Use of Staging Environment

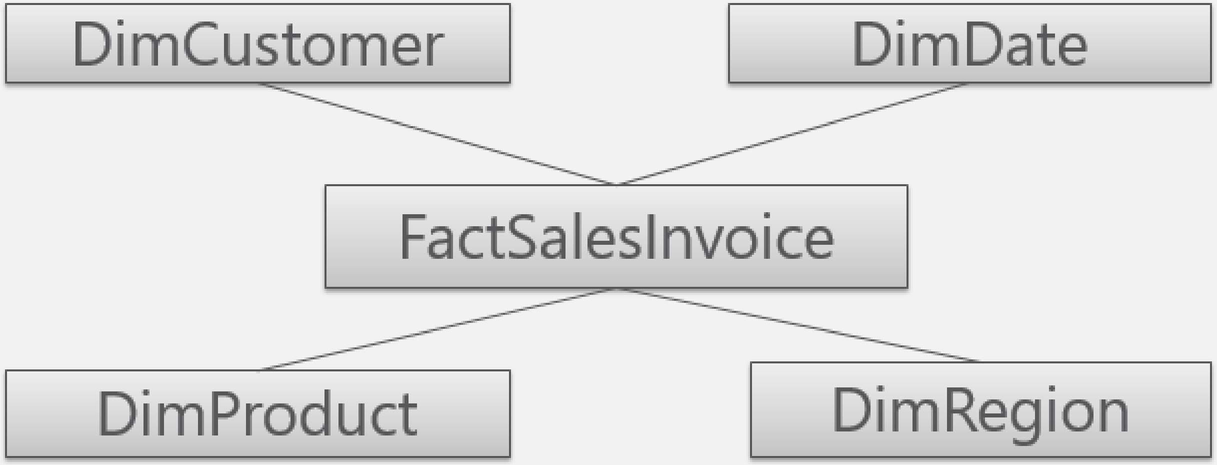


New trend: use of a data lake as the DW staging environment

Staging Objectives:

- ✓ **Reduce load** on source system
- ✓ No changes to source format
- ✓ A “kitchen area”
- ✓ Snapshot of source data for troubleshooting

Usage of a Star Schema



Dimension Table

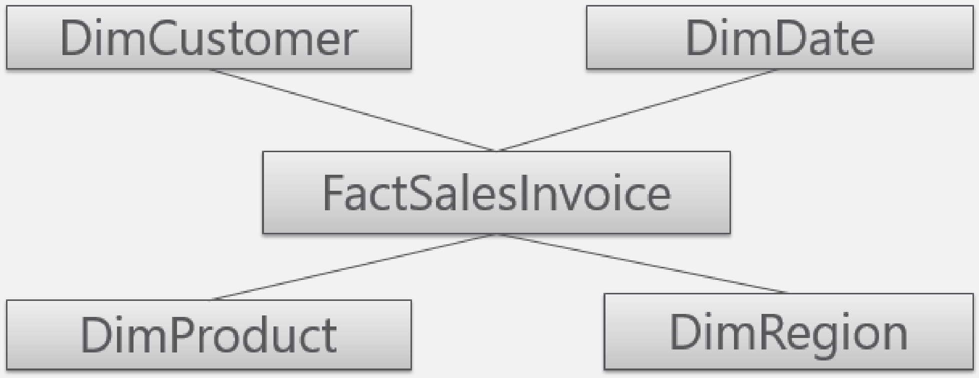
Provides the **descriptive context** – attributes with the who, what, when, why, or how

Fact Table

Fact tables contain the **numeric**, **quantitative** data (aka **measures**)

Benefits of a Star Schema

Optimal for **known** reporting scenarios



Denormalized structure, structured around **business logic**, is good for **performance** & **consistency**

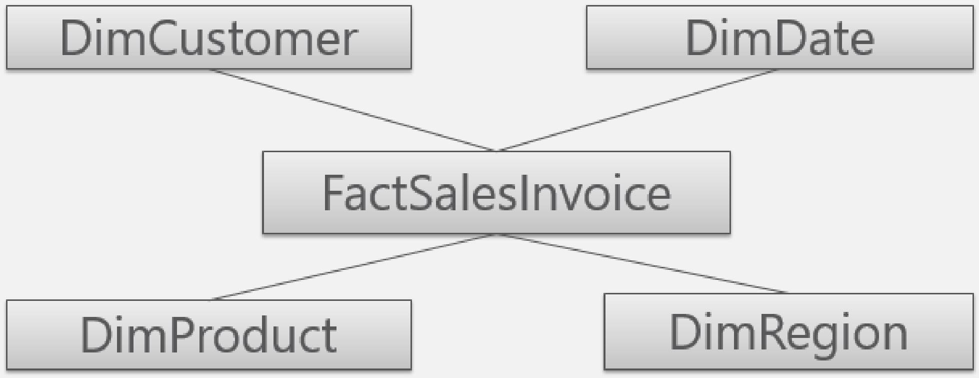
Decoupled from source systems: **surrogate keys** which have no intrinsic meaning

Usability:

- ✓ Stable, predictable environment
- ✓ Less joins, easier navigation
- ✓ Friendly, recognizable names
- ✓ History retention
- ✓ Integrate multiple systems

Challenges of a Star Schema

Requires **up-front analysis** ("schema on write")



Difficult to handle new & **unpredictable or exploratory scenarios**

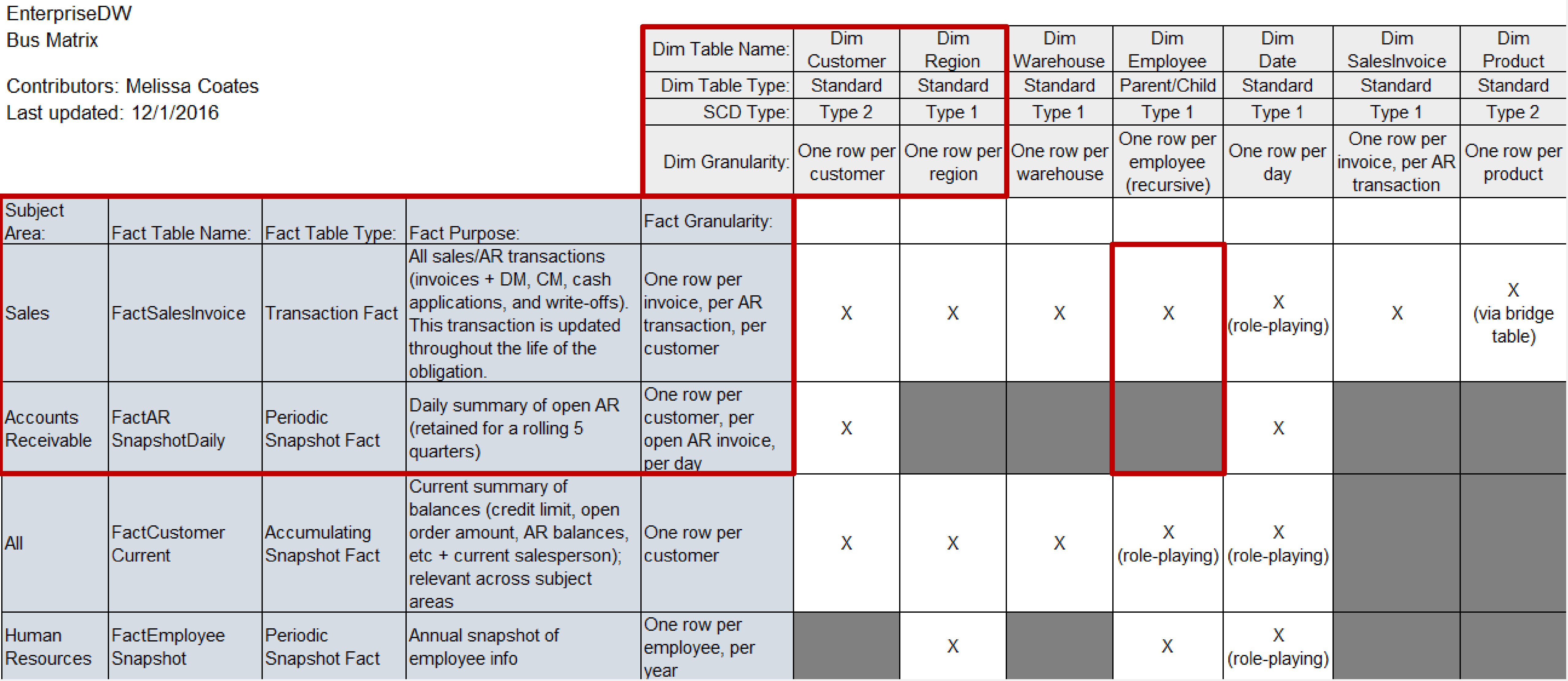
Increasing **volumes of data**

Reducing windows of time for **data loads** (near real-time is challenging)

Data quality issues are often surfaced in the reporting layer

Not practical to contain ***all* of the data** all the time

Declare Grain of Each Table



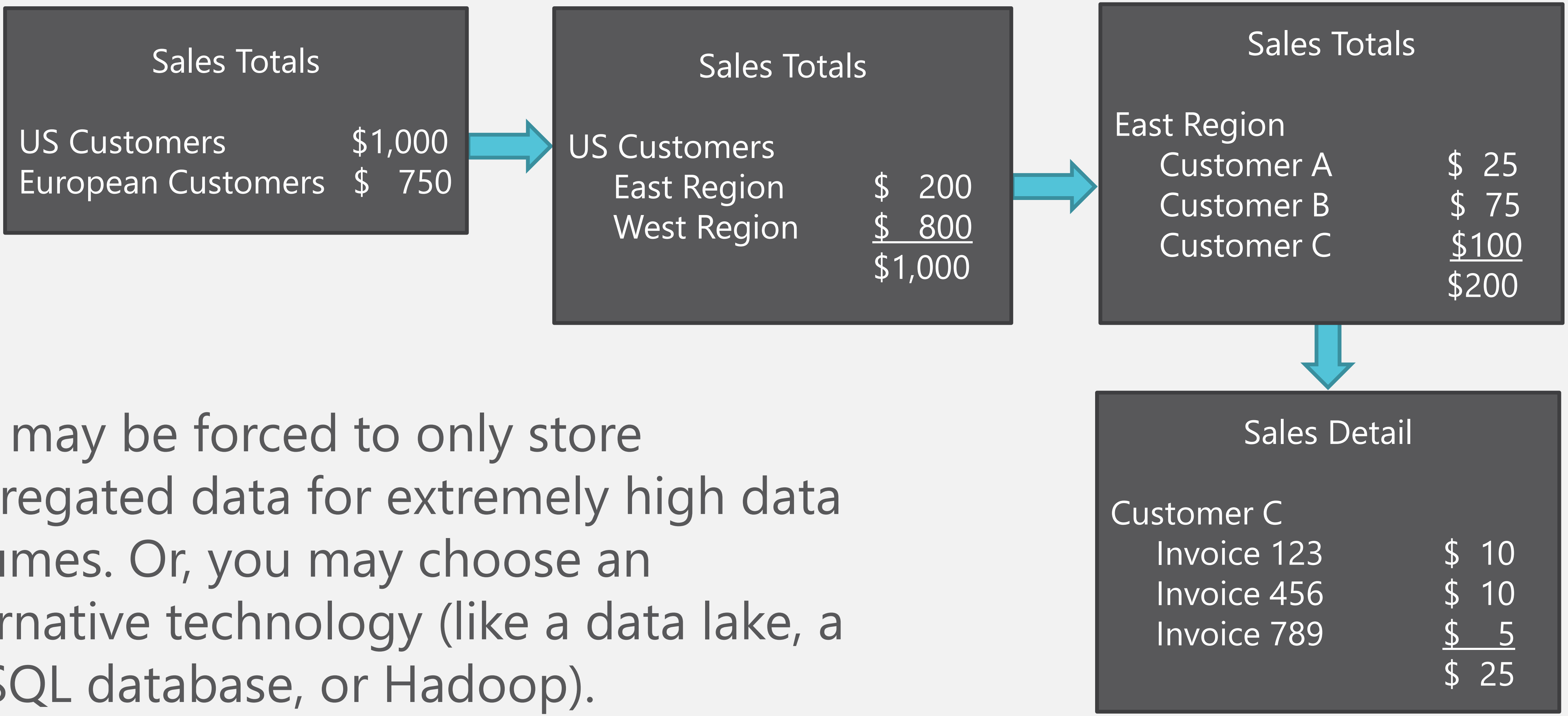
EnterpriseDW
Bus Matrix

Contributors: Melissa Coates
Last updated: 12/1/2016

				Dim Table Name:	Dim Customer	Dim Region	Dim Warehouse	Dim Employee	Dim Date	Dim SalesInvoice	Dim Product
				Dim Table Type:	Standard	Standard	Standard	Parent/Child	Standard	Standard	Standard
				SCD Type:	Type 2	Type 1	Type 1	Type 1	Type 1	Type 1	Type 2
				Dim Granularity:	One row per customer	One row per region	One row per warehouse	One row per employee (recursive)	One row per day	One row per invoice, per AR transaction	One row per product
Subject Area:	Fact Table Name:	Fact Table Type:	Fact Purpose:	Fact Granularity:							
Sales	FactSalesInvoice	Transaction Fact	All sales/AR transactions (invoices + DM, CM, cash applications, and write-offs). This transaction is updated throughout the life of the obligation.	One row per invoice, per AR transaction, per customer	X	X	X	X	X (role-playing)	X	X (via bridge table)
Accounts Receivable	FactAR SnapshotDaily	Periodic Snapshot Fact	Daily summary of open AR (retained for a rolling 5 quarters)	One row per customer, per open AR invoice, per day	X				X		
All	FactCustomer Current	Accumulating Snapshot Fact	Current summary of balances (credit limit, open order amount, AR balances, etc + current salesperson); relevant across subject areas	One row per customer	X	X	X	X (role-playing)	X (role-playing)		
Human Resources	FactEmployee Snapshot	Periodic Snapshot Fact	Annual snapshot of employee info	One row per employee, per year		X		X	X (role-playing)		

Store the Lowest Level Detail You Have

Drill-down behavior:



Sales Totals

US Customers	$1,000
European Customers	$ 750

Sales Totals

US Customers	
East Region	$ 200
West Region	$ 800
	$1,000

Sales Totals

East Region	
Customer A	$ 25
Customer B	$ 75
Customer C	$100
	$200

Sales Detail

Customer C	
Invoice 123	$ 10
Invoice 456	$ 10
Invoice 789	$ 5
	$ 25

You may be forced to only store aggregated data for extremely high data volumes. Or, you may choose an alternative technology (like a data lake, a NoSQL database, or Hadoop).

Dimension Design

Dimension Tables

Dimension tables: provide the **descriptive context** – attributes with the who, what, when, why, or how. They should always include **friendly names & descriptions**.

Dimension tables can contain:

Type of Column in a Dim	Example
Attributes	Customer Name
Non-additive numeric value	Customer Value to Acquisition Cost Ratio
Numeric value used *only* for filtering or grouping (usually accompanied by a "band of ranges")	Customer Satisfaction % Customer Satisfaction Range 90%-100% 80-89% Less than 80%

*Dimension tables should *not* contain aggregatable numeric values (measures).*

Types of Dimension Tables

Most common types of dimensions:

Type of Dim Table	Description
Type 0	Values cannot change (ex: DimDate).
Type 1	Any value which changes is overwritten; no history is preserved.
Type 2 aka Slowly Changing Dimension	Certain important values which change generate a new row which is effective-dated. *(Not all columns should be type 2 - certain columns can be type 1.)*
Type 6	Hybrid of type 1 and 2 which includes a new column for the important values, as well as a new row.

Types 3, 4, 5, and 7 do exist, but are less commonly utilized.

Type 1 Dimension

Original data:

Customer SK	Customer NK	Customer Name	AuditRow UpdateDate
1	ABC	Brian Jones	6-4-2014
2	DEF	Sally Baker	10-1-2015

Change to Customer Name occurs.

Updated data:

Customer SK	Customer NK	Customer Name	AuditRow UpdateDate
1	ABC	Brian Jones	6-4-2014
2	DEF	**Sally Walsh**	**12-2-2016**

Type 2 Dimension

Original data:

Customer SK	Customer NK	Customer Name	AuditRow Effective Date	AuditRow Expired Date	AuditRow IsCurrent
1	ABC	Brian Jones	6-4-2014	12-31-9999	1
2	DEF	Sally Baker	10-1-2015	12-31-9999	1

Change to Customer Name occurs.

Updated data:

Customer SK	Customer NK	Customer Name	AuditRow Effective Date	AuditRow Expired Date	AuditRow IsCurrent
1	ABC	Brian Jones	6-4-2014	12-31-9999	1
2	DEF	Sally Baker	10-1-2015	**12-2-2016**	**0**
3	**DEF**	**Sally Walsh**	**12-3-2016**	**12-31-9999**	**1**

Type 6 Dimension

Original data:

Customer SK	Customer NK	Customer Name	Customer Name Current	AuditRow Effective Date	AuditRow Expired Date	Audit RowIs Current
1	ABC	Brian Jones	Brian Jones	6-4-2014	12-31-9999	1
2	DEF	Sally Baker	Sally Baker	10-1-2015	12-31-9999	1

Change to Customer Name occurs.

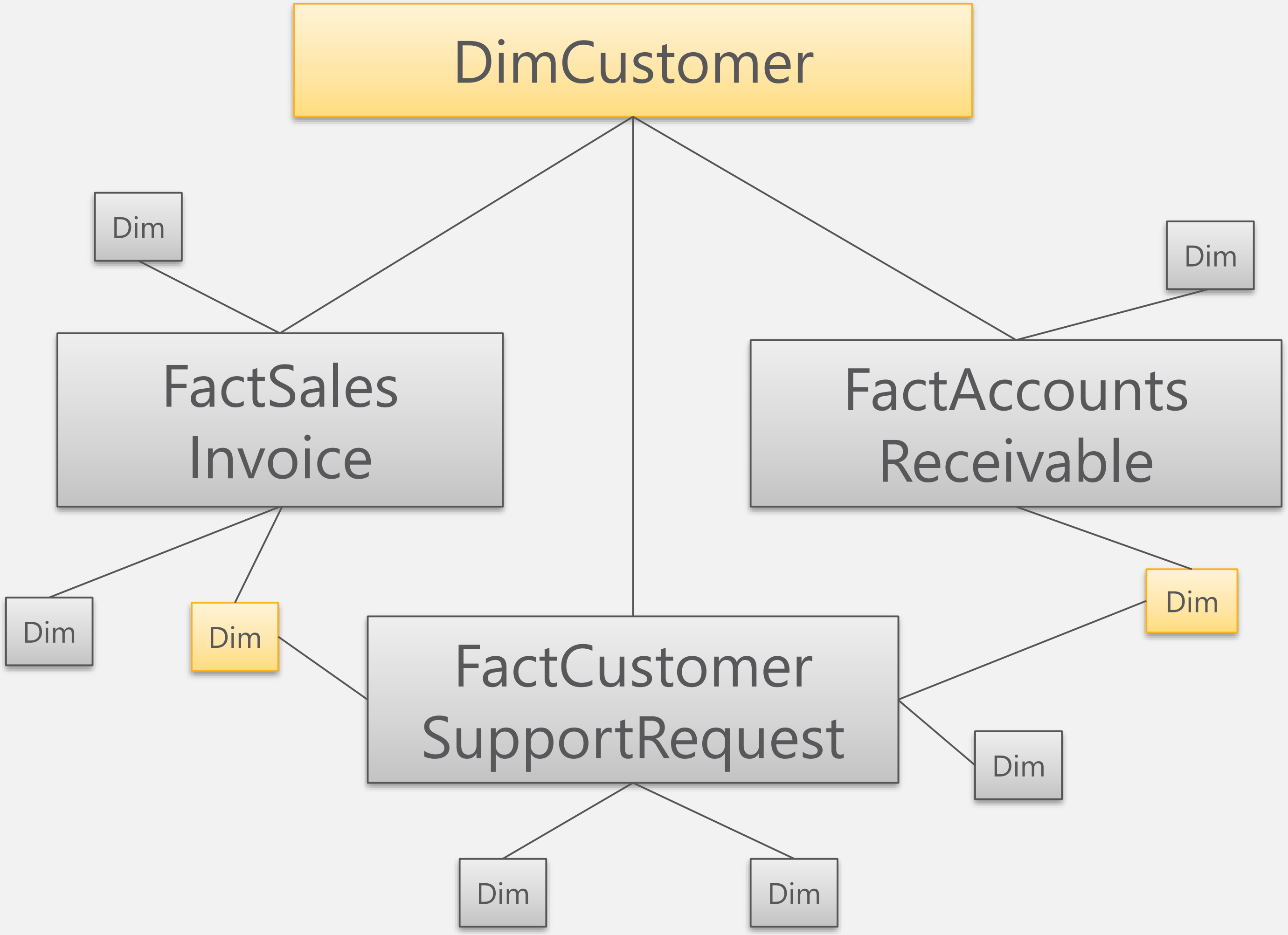
Updated data:

Customer SK	Customer NK	Customer Name	Customer Name Current	Audit Row Effective Date	AuditRow Expired Date	Audit RowIs Current
1	ABC	Brian Jones	Brian Jones	6-4-2014	12-31-9999	1
2	DEF	Sally Baker	**Sally Walsh**	**10-1-2015**	**12-2-2016**	**0**
3	**DEF**	**Sally Walsh**	**Sally Walsh**	**12-3-2016**	**12-31-9999**	**1**

Conformed Dimension

A conformed dimension **reuses the same dimension** across numerous fact tables: critical for unifying data from various sources.

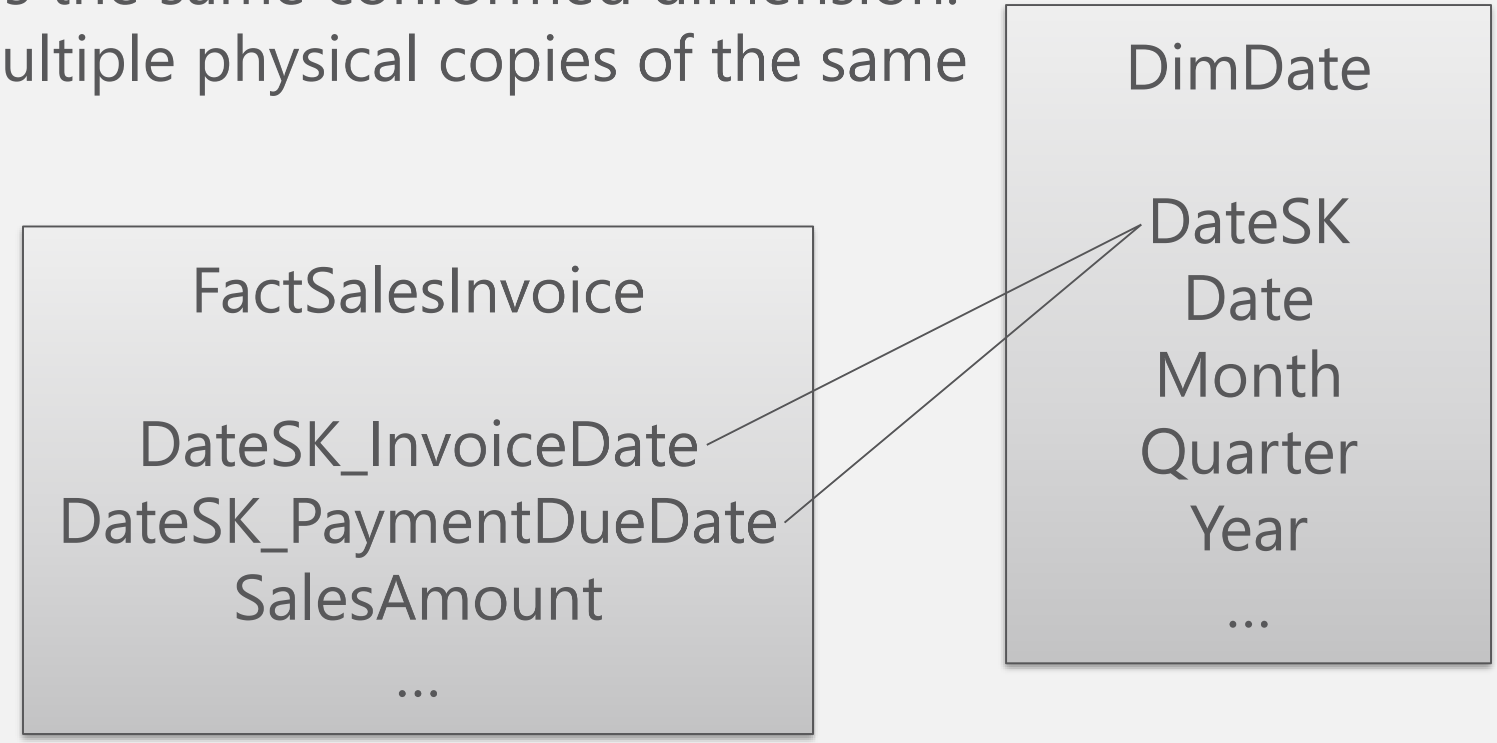
Conformed dimensions provide significant value with '**drill across**' functionality, and provide a **consistent** user experience.



Role-Playing Dimension

A role-playing dimension utilizes the same conformed dimension. Objective is to avoid creating multiple physical copies of the same dimension table.

```
SELECT
    FSI.SalesAmount
   ,InvoiceDate = DtInv.Date
   ,PymtDueDate = DtDue.Date
FROM FactSalesInvoice AS FSI
INNER JOIN DimDate AS DtInv
    ON FSI.DateSK_InvoiceDate = DtInv.DateSK
INNER JOIN DimDate AS DtDue
    ON FSI.DateSK_PaymentDueDate = DtDue.DateSK
```



FactSalesInvoice

DateSK_InvoiceDate
DateSK_PaymentDueDate
SalesAmount
…

DimDate

DateSK
Date
Month
Quarter
Year
…

Hierarchies

Hierarchies are extremely useful for handling rollups, and for drill-down & drill-through behavior.

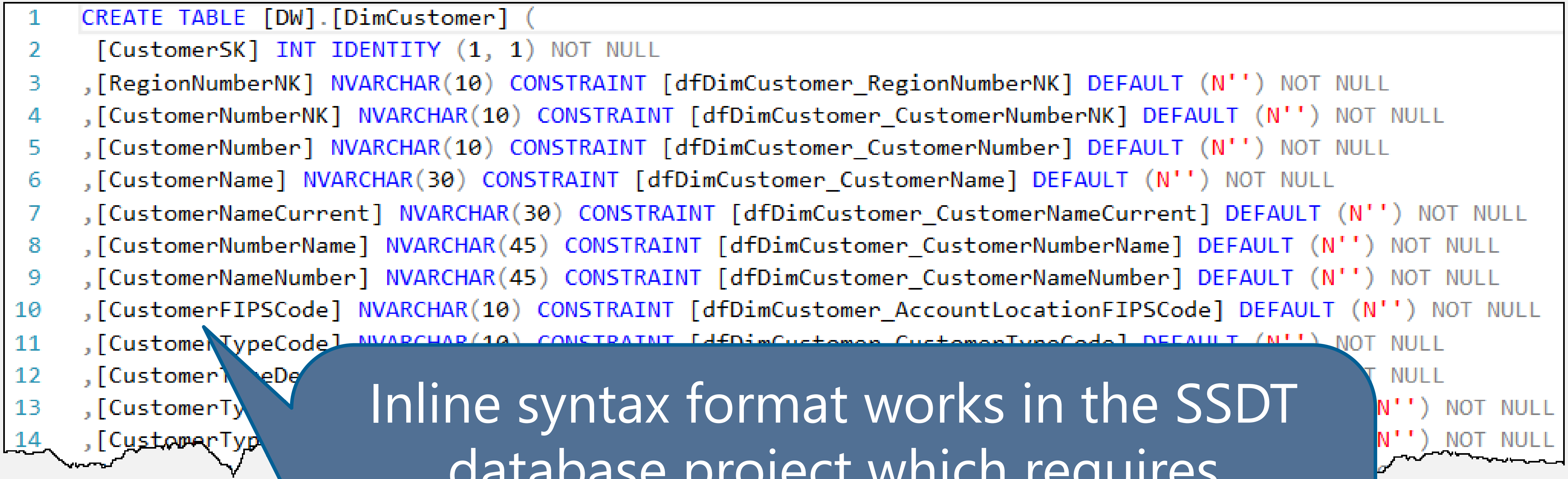
Date Hierarchy

- Year
  - Quarter
    - Month
      - Day

Geography Hierarchy

- Country
  - State or Province
    - City
      - Address

Dimension Design



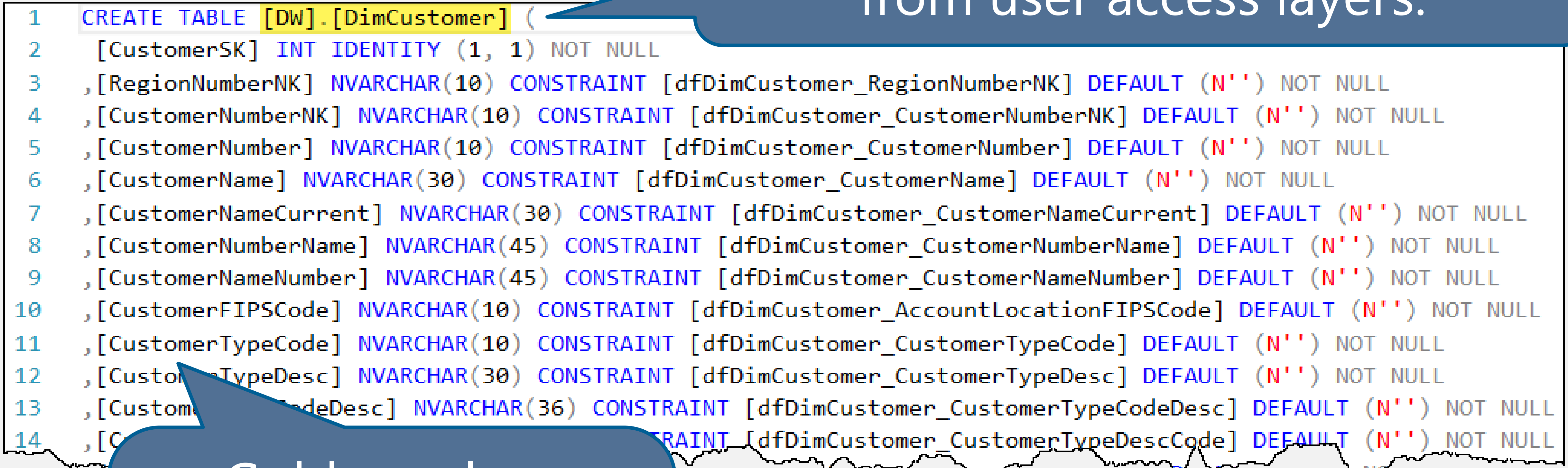
```
CREATE TABLE [DW].[DimCustomer] (
 [CustomerSK] INT IDENTITY (1, 1) NOT NULL
,[RegionNumberNK] NVARCHAR(10) CONSTRAINT [dfDimCustomer_RegionNumberNK] DEFAULT (N'') NOT NULL
,[CustomerNumberNK] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerNumberNK] DEFAULT (N'') NOT NULL
,[CustomerNumber] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerNumber] DEFAULT (N'') NOT NULL
,[CustomerName] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerName] DEFAULT (N'') NOT NULL
,[CustomerNameCurrent] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerNameCurrent] DEFAULT (N'') NOT NULL
,[CustomerNumberName] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNumberName] DEFAULT (N'') NOT NULL
,[CustomerNameNumber] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNameNumber] DEFAULT (N'') NOT NULL
,[CustomerFIPSCode] NVARCHAR(10) CONSTRAINT [dfDimCustomer_AccountLocationFIPSCode] DEFAULT (N'') NOT NULL
,[CustomerTypeCode] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerTypeCode] DEFAULT (N'') NOT NULL
,[Customer  eDe                                                                                T NULL
,[CustomerTy                                                                                N'') NOT NULL
,[CustomerTyp                                                                               N'') NOT NULL
```

Inline syntax format works in the SSDT database project which requires "declarative development."

No alters beneath the create.

Dimension Design

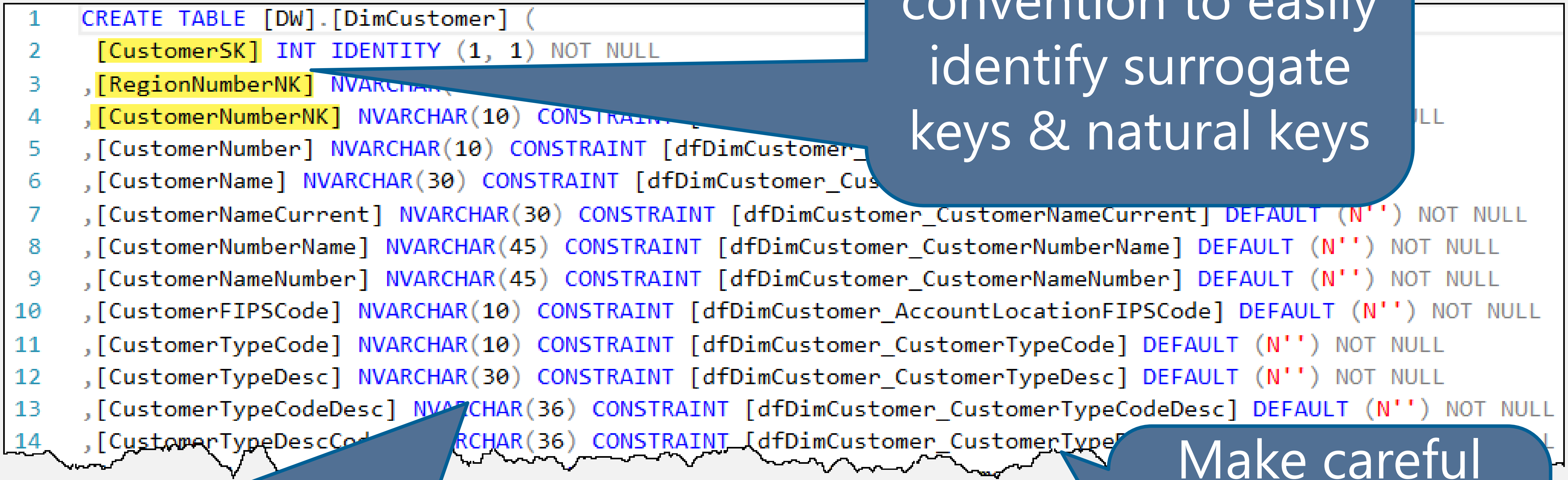
Remove the Dim or Fact prefix from user access layers.



```
CREATE TABLE [DW].[DimCustomer] (
 [CustomerSK] INT IDENTITY (1, 1) NOT NULL
,[RegionNumberNK] NVARCHAR(10) CONSTRAINT [dfDimCustomer_RegionNumberNK] DEFAULT (N'') NOT NULL
,[CustomerNumberNK] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerNumberNK] DEFAULT (N'') NOT NULL
,[CustomerNumber] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerNumber] DEFAULT (N'') NOT NULL
,[CustomerName] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerName] DEFAULT (N'') NOT NULL
,[CustomerNameCurrent] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerNameCurrent] DEFAULT (N'') NOT NULL
,[CustomerNumberName] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNumberName] DEFAULT (N'') NOT NULL
,[CustomerNameNumber] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNameNumber] DEFAULT (N'') NOT NULL
,[CustomerFIPSCode] NVARCHAR(10) CONSTRAINT [dfDimCustomer_AccountLocationFIPSCode] DEFAULT (N'') NOT NULL
,[CustomerTypeCode] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerTypeCode] DEFAULT (N'') NOT NULL
,[Custo[illegible]TypeDesc] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerTypeDesc] DEFAULT (N'') NOT NULL
,[Custom[illegible]deDesc] NVARCHAR(36) CONSTRAINT [dfDimCustomer_CustomerTypeCodeDesc] DEFAULT (N'') NOT NULL
,[C[illegible]RAINT [dfDimCustomer_CustomerTypeDescCode] DEFAULT (N'') NOT NULL
```

Golden rule: a column exists in one and only one place in the DW.

Dimension Design



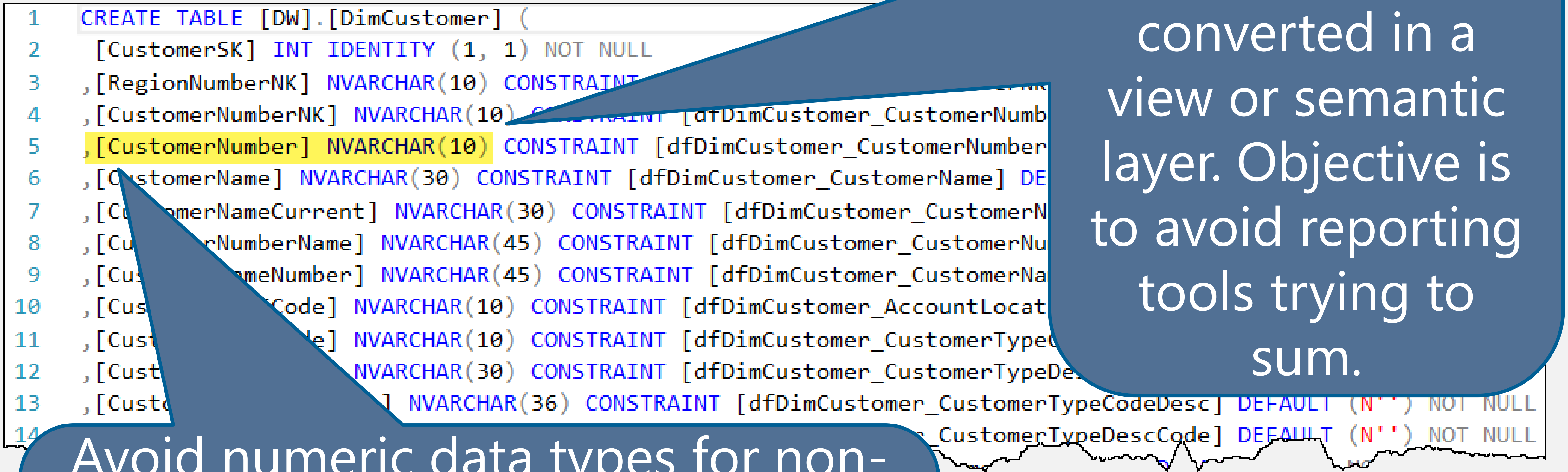
```
CREATE TABLE [DW].[DimCustomer] (
 [CustomerSK] INT IDENTITY (1, 1) NOT NULL
,[RegionNumberNK] NVARCHAR(
,[CustomerNumberNK] NVARCHAR(10) CONSTRAINT [                                                   LL
,[CustomerNumber] NVARCHAR(10) CONSTRAINT [dfDimCustomer_
,[CustomerName] NVARCHAR(30) CONSTRAINT [dfDimCustomer_Cus
,[CustomerNameCurrent] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerNameCurrent] DEFAULT (N'') NOT NULL
,[CustomerNumberName] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNumberName] DEFAULT (N'') NOT NULL
,[CustomerNameNumber] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNameNumber] DEFAULT (N'') NOT NULL
,[CustomerFIPSCode] NVARCHAR(10) CONSTRAINT [dfDimCustomer_AccountLocationFIPSCode] DEFAULT (N'') NOT NULL
,[CustomerTypeCode] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerTypeCode] DEFAULT (N'') NOT NULL
,[CustomerTypeDesc] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerTypeDesc] DEFAULT (N'') NOT NULL
,[CustomerTypeCodeDesc] NVARCHAR(36) CONSTRAINT [dfDimCustomer_CustomerTypeCodeDesc] DEFAULT (N'') NOT NULL
,[CustomerTypeDescCo      RCHAR(36) CONSTRAINT [dfDimCustomer_CustomerType
```

Use a naming convention to easily identify surrogate keys & natural keys

Use the smallest datatypes you can use without risk of overflows

Make careful decisions on the use of varchar vs. nvarchar

Dimension Design

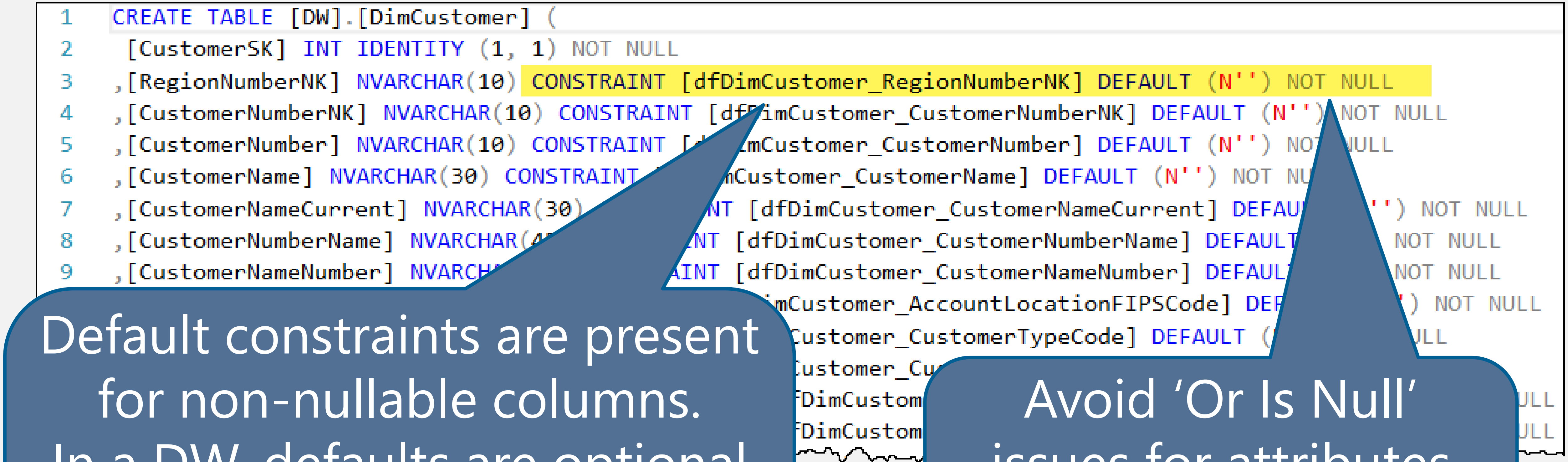


```
CREATE TABLE [DW].[DimCustomer] (
 [CustomerSK] INT IDENTITY (1, 1) NOT NULL
,[RegionNumberNK] NVARCHAR(10) CONSTRAINT
,[CustomerNumberNK] NVARCHAR(10)          [dfDimCustomer_CustomerNumb
,[CustomerNumber] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerNumber
,[CustomerName] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerName] DE
,[Cu    merNameCurrent] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerN
,[Cu    rNumberName] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNu
,[Cus     meNumber] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNa
,[Cus         Code] NVARCHAR(10) CONSTRAINT [dfDimCustomer_AccountLocat
,[Cust          e] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerType
,[Cust             NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerTypeDe
,[Custo           ] NVARCHAR(36) CONSTRAINT [dfDimCustomer_CustomerTypeCodeDesc] DEFAULT (N'') NOT NULL
                                             CustomerTypeDescCode] DEFAULT (N'') NOT NULL
```

Alternatively, could be converted in a view or semantic layer. Objective is to avoid reporting tools trying to sum.

Avoid numeric data types for non-aggregatable columns such as Customer Number.
Also useful for retaining leading 0s or for international zip codes.

Dimension Design

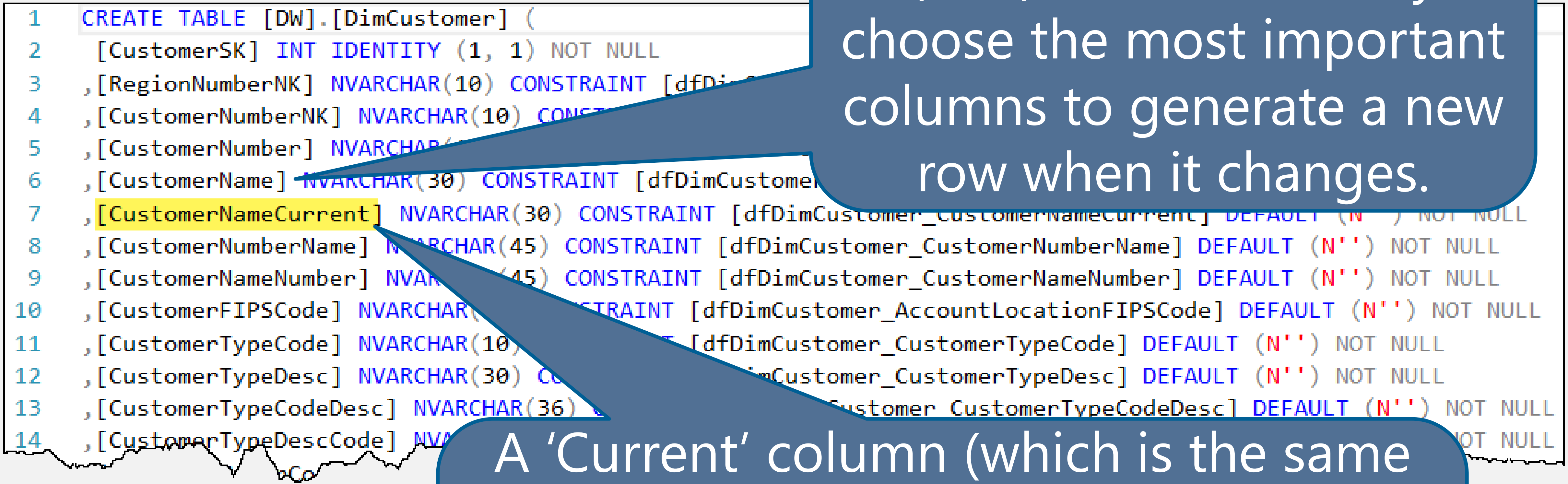


```
CREATE TABLE [DW].[DimCustomer] (
 [CustomerSK] INT IDENTITY (1, 1) NOT NULL
,[RegionNumberNK] NVARCHAR(10) CONSTRAINT [dfDimCustomer_RegionNumberNK] DEFAULT (N'') NOT NULL
,[CustomerNumberNK] NVARCHAR(10) CONSTRAINT [df imCustomer_CustomerNumberNK] DEFAULT (N'') NOT NULL
,[CustomerNumber] NVARCHAR(10) CONSTRAINT [d  mCustomer_CustomerNumber] DEFAULT (N'') NOT NULL
,[CustomerName] NVARCHAR(30) CONSTRAINT    mCustomer_CustomerName] DEFAULT (N'') NOT NU
,[CustomerNameCurrent] NVARCHAR(30)      NT [dfDimCustomer_CustomerNameCurrent] DEFAU    '') NOT NULL
,[CustomerNumberName] NVARCHAR(4        NT [dfDimCustomer_CustomerNumberName] DEFAULT     NOT NULL
,[CustomerNameNumber] NVARCH          AINT [dfDimCustomer_CustomerNameNumber] DEFAUL      NOT NULL
                                         imCustomer_AccountLocationFIPSCode] DEF      ') NOT NULL
                                          Customer_CustomerTypeCode] DEFAULT (        ULL
                                          Customer_Cu
                                          fDimCustom                                   ULL
                                          fDimCustom                                   ULL
```

Default constraints are present for non-nullable columns. In a DW, defaults are optional if ETL strictly controls all data management. *Don't let SQL Server auto-name constraints.

Avoid 'Or Is Null' issues for attributes which are commonly used in predicates.

Dimension Design

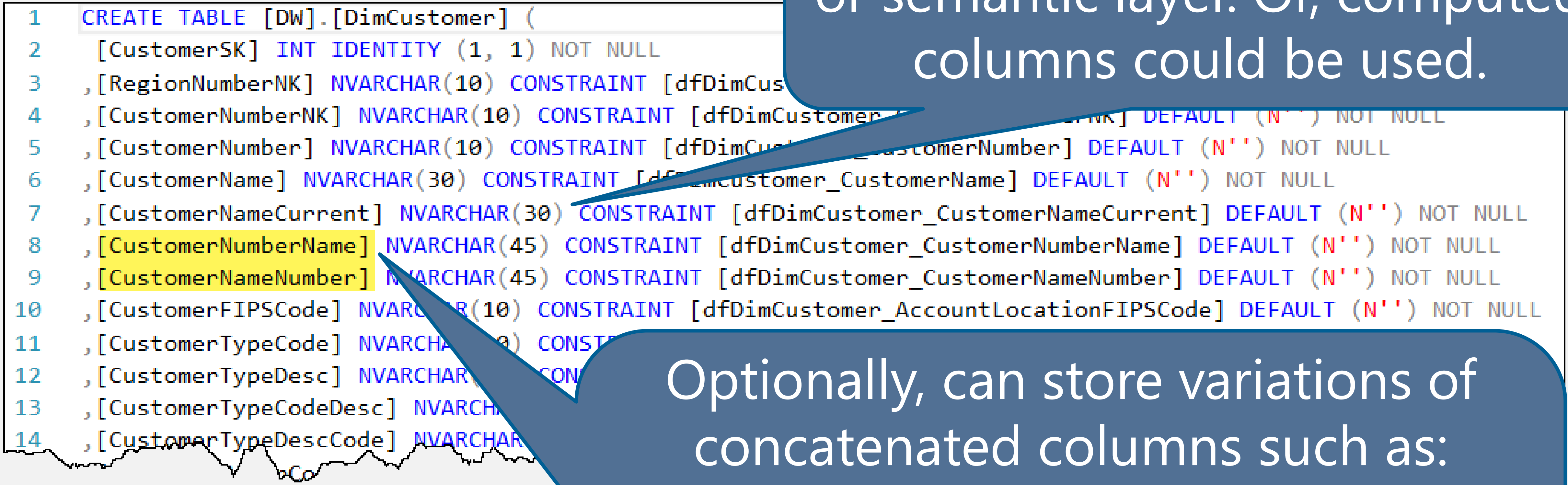


```
CREATE TABLE [DW].[DimCustomer] (
 [CustomerSK] INT IDENTITY (1, 1) NOT NULL
,[RegionNumberNK] NVARCHAR(10) CONSTRAINT [dfDi
,[CustomerNumberNK] NVARCHAR(10) CONST
,[CustomerNumber] NVARCHAR(
,[CustomerName] NVARCHAR(30) CONSTRAINT [dfDimCustome
,[CustomerNameCurrent] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerNameCurrent] DEFAULT (N'') NOT NULL
,[CustomerNumberName] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNumberName] DEFAULT (N'') NOT NULL
,[CustomerNameNumber] NVAR     (45) CONSTRAINT [dfDimCustomer_CustomerNameNumber] DEFAULT (N'') NOT NULL
,[CustomerFIPSCode] NVARCHAR(      TRAINT [dfDimCustomer_AccountLocationFIPSCode] DEFAULT (N'') NOT NULL
,[CustomerTypeCode] NVARCHAR(10       T [dfDimCustomer_CustomerTypeCode] DEFAULT (N'') NOT NULL
,[CustomerTypeDesc] NVARCHAR(30) C          mCustomer_CustomerTypeDesc] DEFAULT (N'') NOT NULL
,[CustomerTypeCodeDesc] NVARCHAR(36)         ustomer_CustomerTypeCodeDesc] DEFAULT (N'') NOT NULL
,[CustomerTypeDescCode] NVA                                                         OT NULL
```

When designing a Type 2 (or 6) dimension, only choose the most important columns to generate a new row when it changes.

A ‘Current’ column (which is the same across all rows in a Type 6 dimension) is helpful for columns commonly used in reporting so all history shows the newest value.

Dimension Design

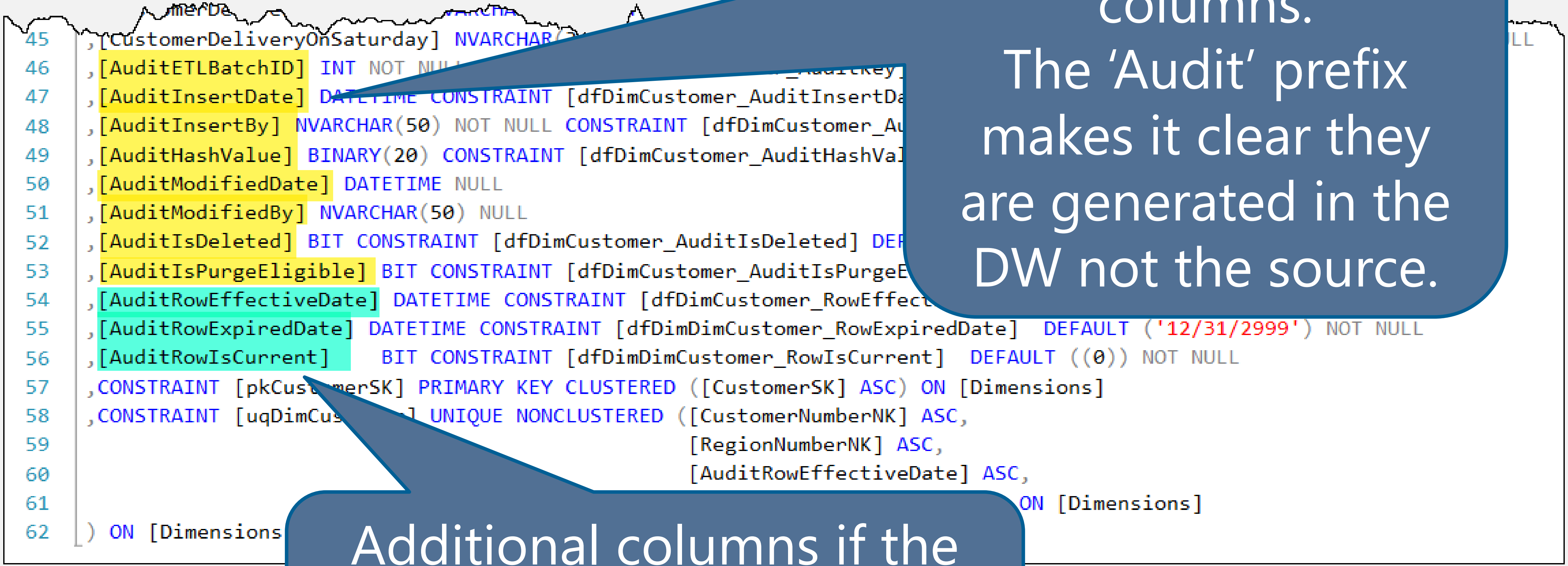


```
CREATE TABLE [DW].[DimCustomer] (
 [CustomerSK] INT IDENTITY (1, 1) NOT NULL
,[RegionNumberNK] NVARCHAR(10) CONSTRAINT [dfDimCus
,[CustomerNumberNK] NVARCHAR(10) CONSTRAINT [dfDimCustomer                rNK] DEFAULT (N'') NOT NULL
,[CustomerNumber] NVARCHAR(10) CONSTRAINT [dfDimCu         stomerNumber] DEFAULT (N'') NOT NULL
,[CustomerName] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerName] DEFAULT (N'') NOT NULL
,[CustomerNameCurrent] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerNameCurrent] DEFAULT (N'') NOT NULL
,[CustomerNumberName] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNumberName] DEFAULT (N'') NOT NULL
,[CustomerNameNumber] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNameNumber] DEFAULT (N'') NOT NULL
,[CustomerFIPSCode] NVARCHAR(10) CONSTRAINT [dfDimCustomer_AccountLocationFIPSCode] DEFAULT (N'') NOT NULL
,[CustomerTypeCode] NVARCHA    0) CONST
,[CustomerTypeDesc] NVARCHAR    CON
,[CustomerTypeCodeDesc] NVARCH
,[CustomerTypeDescCode] NVARCHAR
```

Could also be derived in views or semantic layer. Or, computed columns could be used.

Optionally, can store variations of concatenated columns such as:
Name (Number)
Number - Name
Description (Code)
Code - Description

Dimension Design



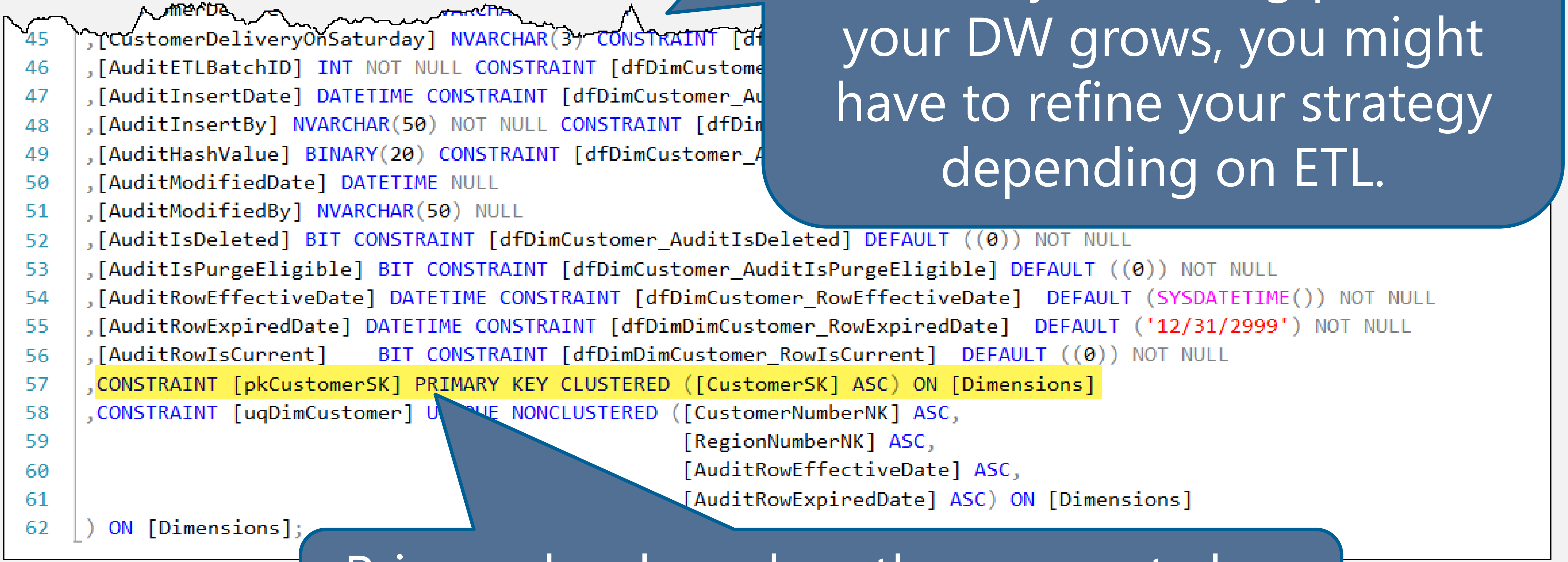
```
,[CustomerDeliveryOnSaturday] NVARCHAR(                                                                                  LL
,[AuditETLBatchID] INT NOT NU                 _AuditKey]
,[AuditInsertDate] DATETIME CONSTRAINT [dfDimCustomer_AuditInsertDa
,[AuditInsertBy] NVARCHAR(50) NOT NULL CONSTRAINT [dfDimCustomer_Au
,[AuditHashValue] BINARY(20) CONSTRAINT [dfDimCustomer_AuditHashVal
,[AuditModifiedDate] DATETIME NULL
,[AuditModifiedBy] NVARCHAR(50) NULL
,[AuditIsDeleted] BIT CONSTRAINT [dfDimCustomer_AuditIsDeleted] DEF
,[AuditIsPurgeEligible] BIT CONSTRAINT [dfDimCustomer_AuditIsPurgeE
,[AuditRowEffectiveDate] DATETIME CONSTRAINT [dfDimCustomer_RowEffect
,[AuditRowExpiredDate] DATETIME CONSTRAINT [dfDimDimCustomer_RowExpiredDate]  DEFAULT ('12/31/2999') NOT NULL
,[AuditRowIsCurrent]    BIT CONSTRAINT [dfDimDimCustomer_RowIsCurrent]  DEFAULT ((0)) NOT NULL
,CONSTRAINT [pkCust   erSK] PRIMARY KEY CLUSTERED ([CustomerSK] ASC) ON [Dimensions]
,CONSTRAINT [uqDimCu    ] UNIQUE NONCLUSTERED ([CustomerNumberNK] ASC,
                                                [RegionNumberNK] ASC,
                                                [AuditRowEffectiveDate] ASC,
                                                                    ON [Dimensions]
) ON [Dimensions
```

Standard audit columns.
The 'Audit' prefix makes it clear they are generated in the DW not the source.

Additional columns if the Type 2 historical change tracking is occurring.

Dimension Design

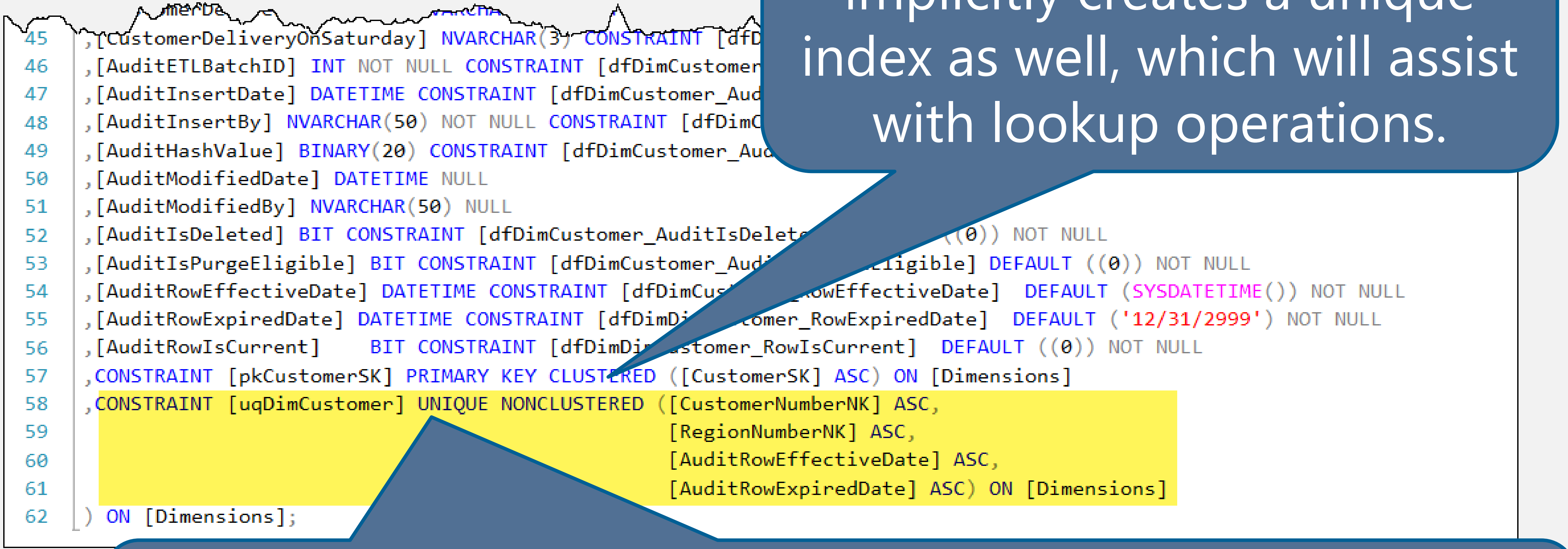
All key & index suggestions are merely a starting point. As your DW grows, you might have to refine your strategy depending on ETL.



```
,[CustomerDeliveryOnSaturday] NVARCHAR(3) CONSTRAINT [df
,[AuditETLBatchID] INT NOT NULL CONSTRAINT [dfDimCustome
,[AuditInsertDate] DATETIME CONSTRAINT [dfDimCustomer_Au
,[AuditInsertBy] NVARCHAR(50) NOT NULL CONSTRAINT [dfDim
,[AuditHashValue] BINARY(20) CONSTRAINT [dfDimCustomer_A
,[AuditModifiedDate] DATETIME NULL
,[AuditModifiedBy] NVARCHAR(50) NULL
,[AuditIsDeleted] BIT CONSTRAINT [dfDimCustomer_AuditIsDeleted] DEFAULT ((0)) NOT NULL
,[AuditIsPurgeEligible] BIT CONSTRAINT [dfDimCustomer_AuditIsPurgeEligible] DEFAULT ((0)) NOT NULL
,[AuditRowEffectiveDate] DATETIME CONSTRAINT [dfDimCustomer_RowEffectiveDate]  DEFAULT (SYSDATETIME()) NOT NULL
,[AuditRowExpiredDate] DATETIME CONSTRAINT [dfDimDimCustomer_RowExpiredDate]  DEFAULT ('12/31/2999') NOT NULL
,[AuditRowIsCurrent]    BIT CONSTRAINT [dfDimDimCustomer_RowIsCurrent]  DEFAULT ((0)) NOT NULL
,CONSTRAINT [pkCustomerSK] PRIMARY KEY CLUSTERED ([CustomerSK] ASC) ON [Dimensions]
,CONSTRAINT [uqDimCustomer] U    UE NONCLUSTERED ([CustomerNumberNK] ASC,
                                                [RegionNumberNK] ASC,
                                                [AuditRowEffectiveDate] ASC,
                                                [AuditRowExpiredDate] ASC) ON [Dimensions]
) ON [Dimensions];
```

Primary key based on the surrogate key. This is also our clustered index.

Dimension Design

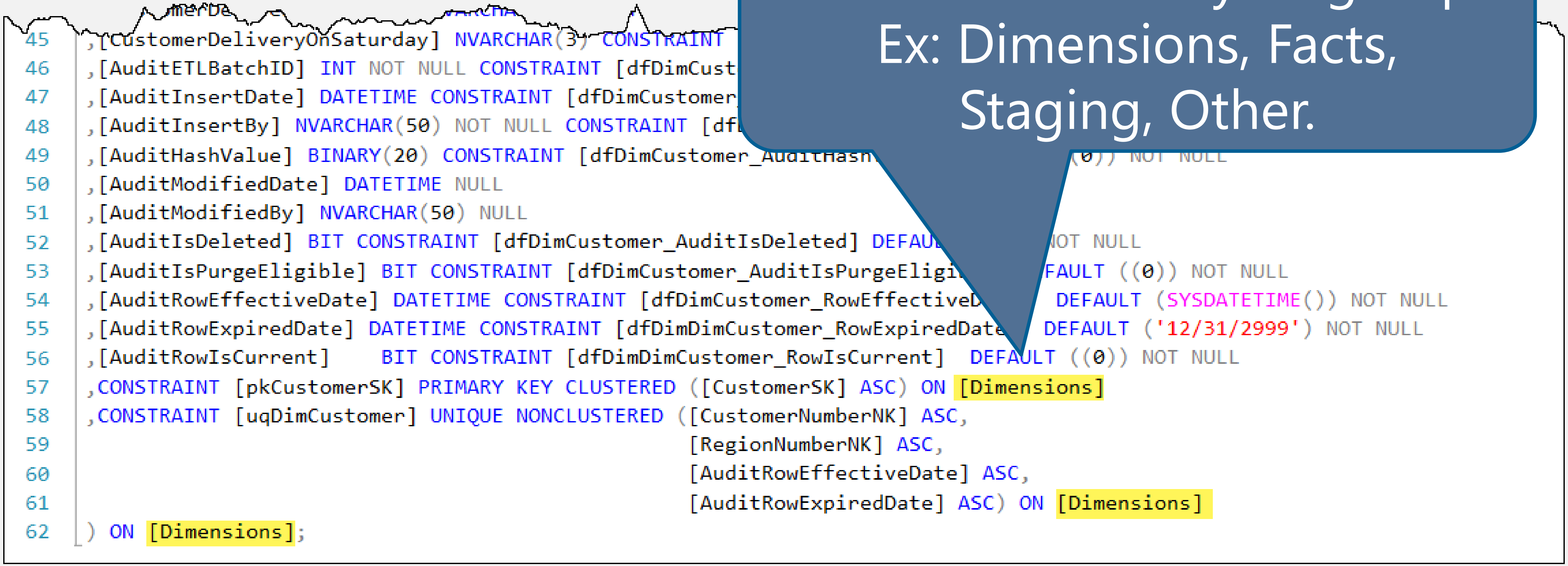


```
,[CustomerDeliveryOnSaturday] NVARCHAR(3) CONSTRAINT [dfD
,[AuditETLBatchID] INT NOT NULL CONSTRAINT [dfDimCustomer
,[AuditInsertDate] DATETIME CONSTRAINT [dfDimCustomer_Aud
,[AuditInsertBy] NVARCHAR(50) NOT NULL CONSTRAINT [dfDimC
,[AuditHashValue] BINARY(20) CONSTRAINT [dfDimCustomer_Aud
,[AuditModifiedDate] DATETIME NULL
,[AuditModifiedBy] NVARCHAR(50) NULL
,[AuditIsDeleted] BIT CONSTRAINT [dfDimCustomer_AuditIsDelete          ((0)) NOT NULL
,[AuditIsPurgeEligible] BIT CONSTRAINT [dfDimCustomer_Aud          ligible] DEFAULT ((0)) NOT NULL
,[AuditRowEffectiveDate] DATETIME CONSTRAINT [dfDimCus          owEffectiveDate]  DEFAULT (SYSDATETIME()) NOT NULL
,[AuditRowExpiredDate] DATETIME CONSTRAINT [dfDimDi          tomer_RowExpiredDate]  DEFAULT ('12/31/2999') NOT NULL
,[AuditRowIsCurrent]    BIT CONSTRAINT [dfDimDi          stomer_RowIsCurrent]  DEFAULT ((0)) NOT NULL
,CONSTRAINT [pkCustomerSK] PRIMARY KEY CLUSTERED ([CustomerSK] ASC) ON [Dimensions]
,CONSTRAINT [uqDimCustomer] UNIQUE NONCLUSTERED ([CustomerNumberNK] ASC,
                                                [RegionNumberNK] ASC,
                                                [AuditRowEffectiveDate] ASC,
                                                [AuditRowExpiredDate] ASC) ON [Dimensions]
) ON [Dimensions];
```

The unique constraint implicitly creates a unique index as well, which will assist with lookup operations.

Unique constraint, based on natural keys, defines the “grain” of the table. It also helps identify data quality issues & is very helpful to the SQL Server query optimizer.

Dimension Design



```
,[CustomerDeliveryOnSaturday] NVARCHAR(3) CONSTRAINT
,[AuditETLBatchID] INT NOT NULL CONSTRAINT [dfDimCust
,[AuditInsertDate] DATETIME CONSTRAINT [dfDimCustomer
,[AuditInsertBy] NVARCHAR(50) NOT NULL CONSTRAINT [df
,[AuditHashValue] BINARY(20) CONSTRAINT [dfDimCustomer_AuditHash              (0)) NOT NULL
,[AuditModifiedDate] DATETIME NULL
,[AuditModifiedBy] NVARCHAR(50) NULL
,[AuditIsDeleted] BIT CONSTRAINT [dfDimCustomer_AuditIsDeleted] DEFAU        NOT NULL
,[AuditIsPurgeEligible] BIT CONSTRAINT [dfDimCustomer_AuditIsPurgeEligi      FAULT ((0)) NOT NULL
,[AuditRowEffectiveDate] DATETIME CONSTRAINT [dfDimCustomer_RowEffectiveD     DEFAULT (SYSDATETIME()) NOT NULL
,[AuditRowExpiredDate] DATETIME CONSTRAINT [dfDimDimCustomer_RowExpiredDate  DEFAULT ('12/31/2999') NOT NULL
,[AuditRowIsCurrent]     BIT CONSTRAINT [dfDimDimCustomer_RowIsCurrent]  DEFAULT ((0)) NOT NULL
,CONSTRAINT [pkCustomerSK] PRIMARY KEY CLUSTERED ([CustomerSK] ASC) ON [Dimensions]
,CONSTRAINT [uqDimCustomer] UNIQUE NONCLUSTERED ([CustomerNumberNK] ASC,
                                                 [RegionNumberNK] ASC,
                                                 [AuditRowEffectiveDate] ASC,
                                                 [AuditRowExpiredDate] ASC) ON [Dimensions]
) ON [Dimensions];
```

Use of non-Primary filegroups.
Ex: Dimensions, Facts,
Staging, Other.

Fact Design

Fact Tables

Fact tables contain the **numeric**, **quantitative** data (aka **measures**).

Typically **one fact table per distinct business process.**
Exception: "consolidated" facts (aka "merged" facts) such as actual vs. forecast which require the same granularity and are frequently analyzed together.

Fact tables can contain:

Type of Column in a Fact	Example
Measures	Sales Amount
Foreign keys to dimension table	3392 (meaningless integer surrogate key)
Degenerate dimension	Order Number

Types of Fact Tables

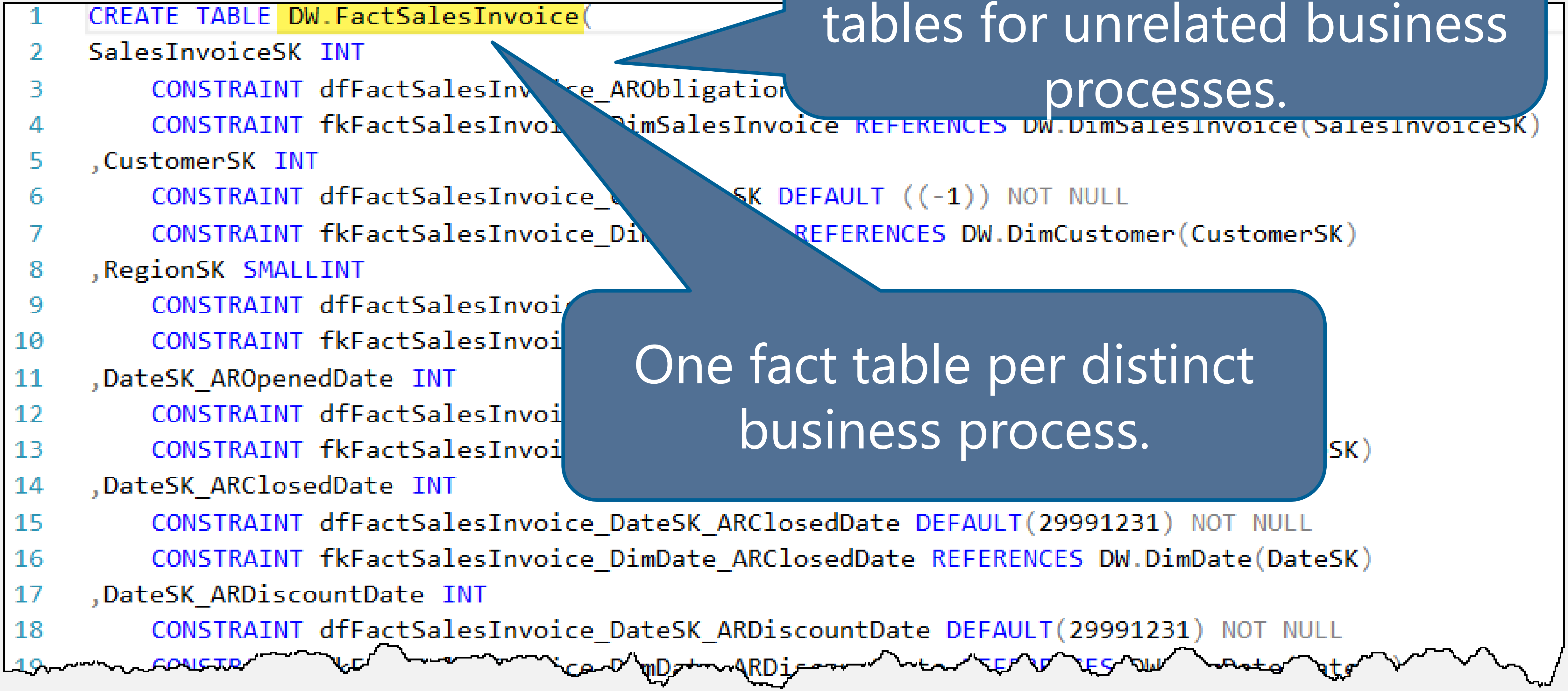
Most common types of facts:

Type of Fact Table	Description	Example
Transaction Fact	An event at a point in time	FactSalesInvoice
Periodic Snapshot Fact	Summary at a point in time	FactARBalanceDaily
Accumulating Snapshot Fact	Summary across the lifetime of an event	FactStudentApplication
Timespan Tracking Fact	Effective-dated rows	FactCapitalAssetBalance

Other facts:

Type of Fact Table	Description	Example
Factless Fact Table	Recording when an event did not occur	FactPromotionNoSales
Aggregate Facts	Rollups, usually to improve reporting speed	FactSalesInvoiceSummary

Fact Design

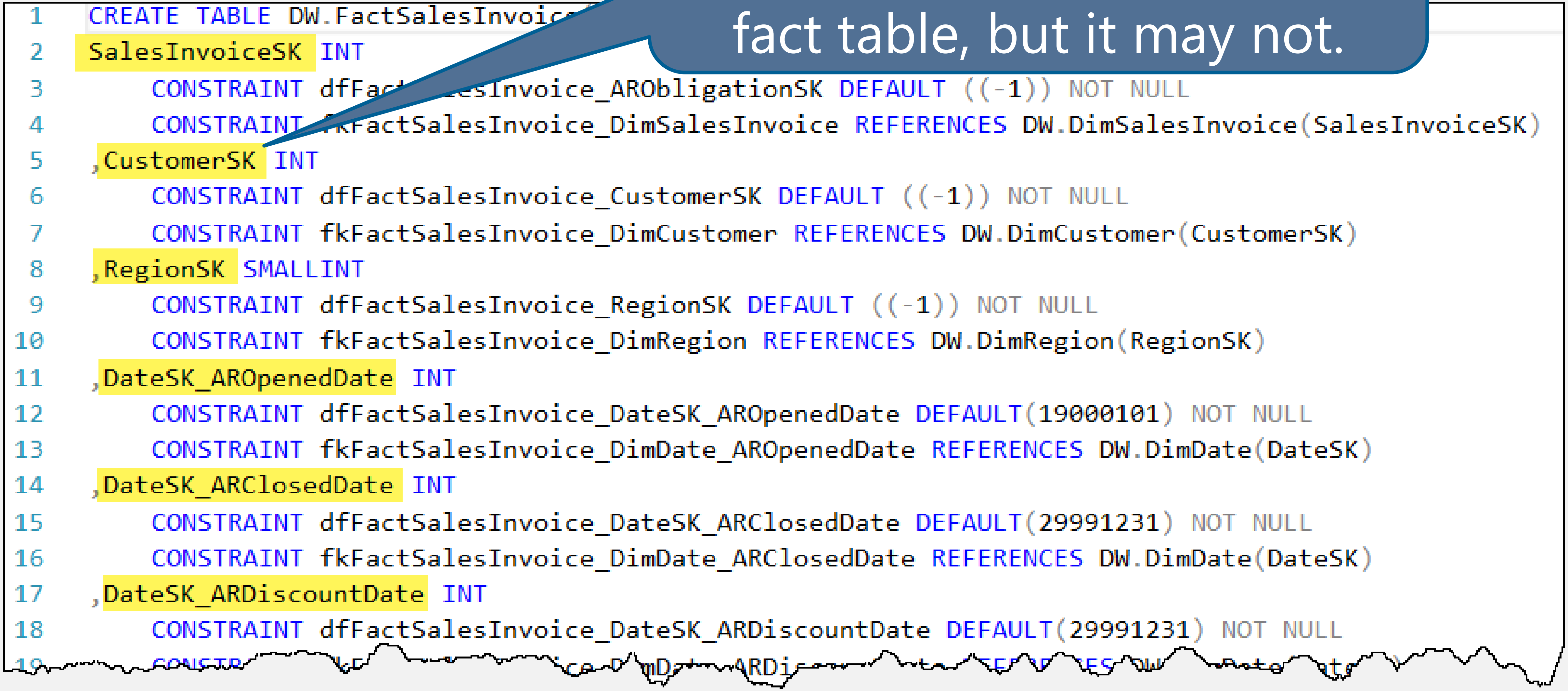


```
CREATE TABLE DW.FactSalesInvoice(
SalesInvoiceSK INT
    CONSTRAINT dfFactSalesInv     ce_ARObligation
    CONSTRAINT fkFactSalesInvoi     DimSalesInvoice REFERENCES DW.DimSalesInvoice(SalesInvoiceSK)
,CustomerSK INT
    CONSTRAINT dfFactSalesInvoice_C          SK DEFAULT ((-1)) NOT NULL
    CONSTRAINT fkFactSalesInvoice_Di          REFERENCES DW.DimCustomer(CustomerSK)
,RegionSK SMALLINT
    CONSTRAINT dfFactSalesInvoi
    CONSTRAINT fkFactSalesInvoi
,DateSK_AROpenedDate INT
    CONSTRAINT dfFactSalesInvoi
    CONSTRAINT fkFactSalesInvoi                                                  SK)
,DateSK_ARClosedDate INT
    CONSTRAINT dfFactSalesInvoice_DateSK_ARClosedDate DEFAULT(29991231) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimDate_ARClosedDate REFERENCES DW.DimDate(DateSK)
,DateSK_ARDiscountDate INT
    CONSTRAINT dfFactSalesInvoice_DateSK_ARDiscountDate DEFAULT(29991231) NOT NULL
```

Even if all of the SKs are the same, avoid combining fact tables for unrelated business processes.

One fact table per distinct business process.

Fact Design

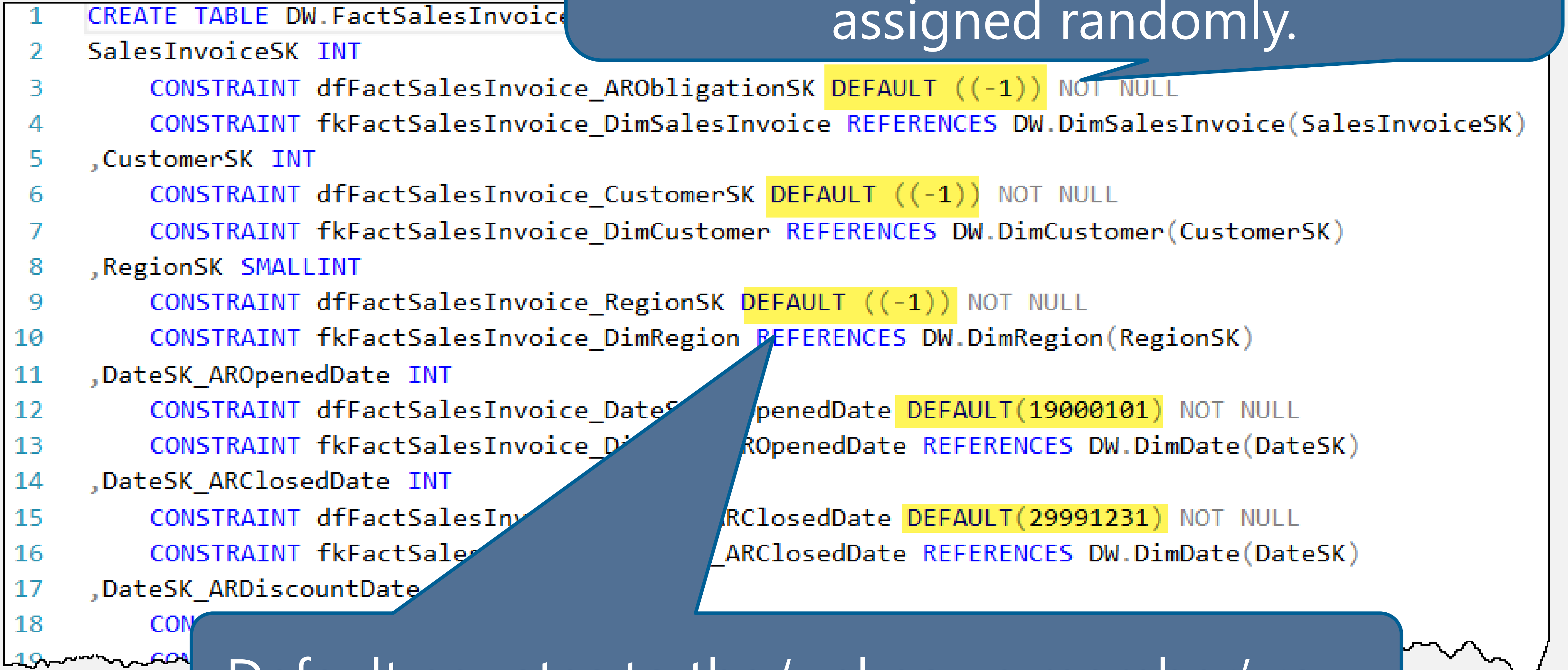


```
CREATE TABLE DW.FactSalesInvoic
SalesInvoiceSK INT
    CONSTRAINT dfFactSalesInvoice_ARObligationSK DEFAULT ((-1)) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimSalesInvoice REFERENCES DW.DimSalesInvoice(SalesInvoiceSK)
,CustomerSK INT
    CONSTRAINT dfFactSalesInvoice_CustomerSK DEFAULT ((-1)) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimCustomer REFERENCES DW.DimCustomer(CustomerSK)
,RegionSK SMALLINT
    CONSTRAINT dfFactSalesInvoice_RegionSK DEFAULT ((-1)) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimRegion REFERENCES DW.DimRegion(RegionSK)
,DateSK_AROpenedDate INT
    CONSTRAINT dfFactSalesInvoice_DateSK_AROpenedDate DEFAULT(19000101) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimDate_AROpenedDate REFERENCES DW.DimDate(DateSK)
,DateSK_ARClosedDate INT
    CONSTRAINT dfFactSalesInvoice_DateSK_ARClosedDate DEFAULT(29991231) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimDate_ARClosedDate REFERENCES DW.DimDate(DateSK)
,DateSK_ARDiscountDate INT
    CONSTRAINT dfFactSalesInvoice_DateSK_ARDiscountDate DEFAULT(29991231) NOT NULL
```

The combination of SKs might dictate the grain of the fact table, but it may not.

Fact Design

Some data modelers prefer the unknown member row to have its key assigned randomly.

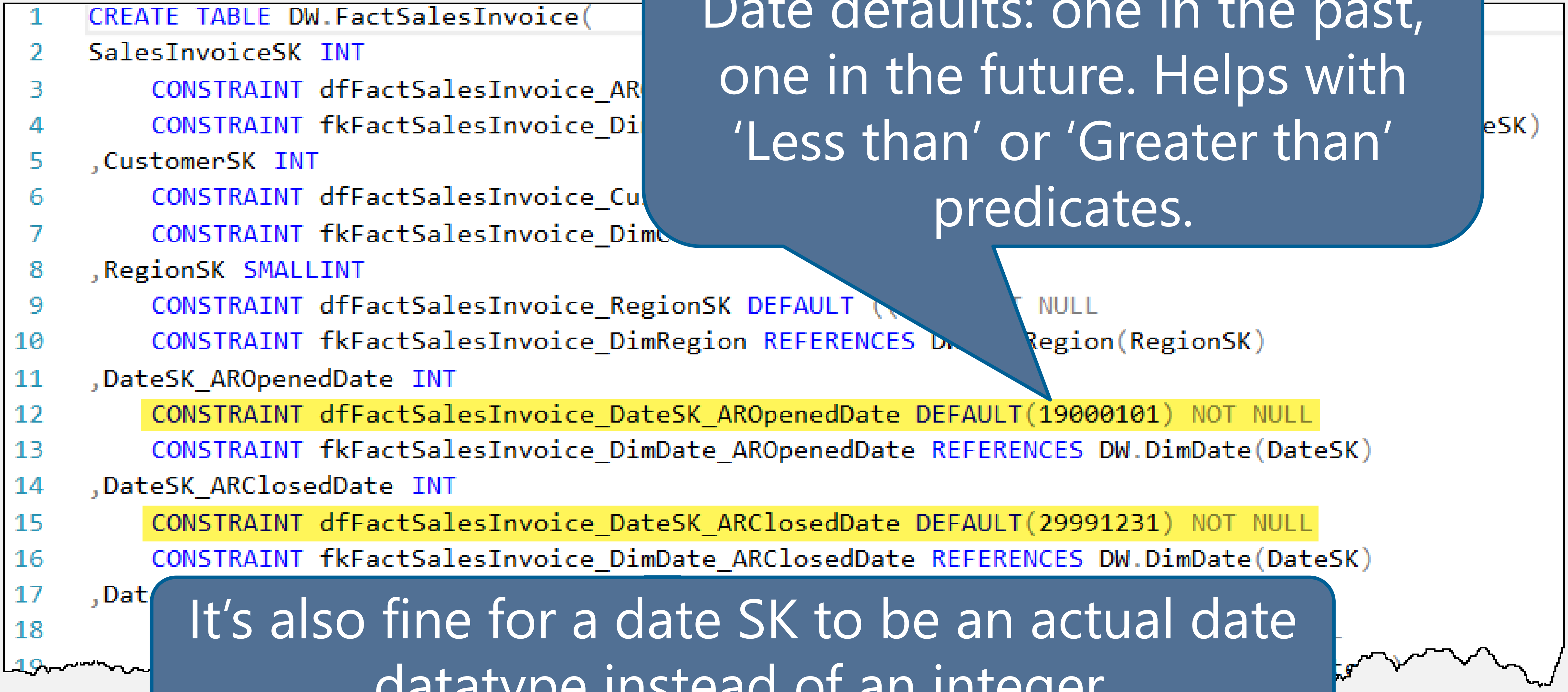


```
CREATE TABLE DW.FactSalesInvoic
SalesInvoiceSK INT
    CONSTRAINT dfFactSalesInvoice_ARObligationSK DEFAULT ((-1)) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimSalesInvoice REFERENCES DW.DimSalesInvoice(SalesInvoiceSK)
,CustomerSK INT
    CONSTRAINT dfFactSalesInvoice_CustomerSK DEFAULT ((-1)) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimCustomer REFERENCES DW.DimCustomer(CustomerSK)
,RegionSK SMALLINT
    CONSTRAINT dfFactSalesInvoice_RegionSK DEFAULT ((-1)) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimRegion REFERENCES DW.DimRegion(RegionSK)
,DateSK_AROpenedDate INT
    CONSTRAINT dfFactSalesInvoice_DateS       penedDate DEFAULT(19000101) NOT NULL
    CONSTRAINT fkFactSalesInvoice_Di          ROpenedDate REFERENCES DW.DimDate(DateSK)
,DateSK_ARClosedDate INT
    CONSTRAINT dfFactSalesIn                  RClosedDate DEFAULT(29991231) NOT NULL
    CONSTRAINT fkFactSales                    _ARClosedDate REFERENCES DW.DimDate(DateSK)
,DateSK_ARDiscountDate
    CON
    CON
```

Default equates to the ‘unknown member’ row.

Fact Design

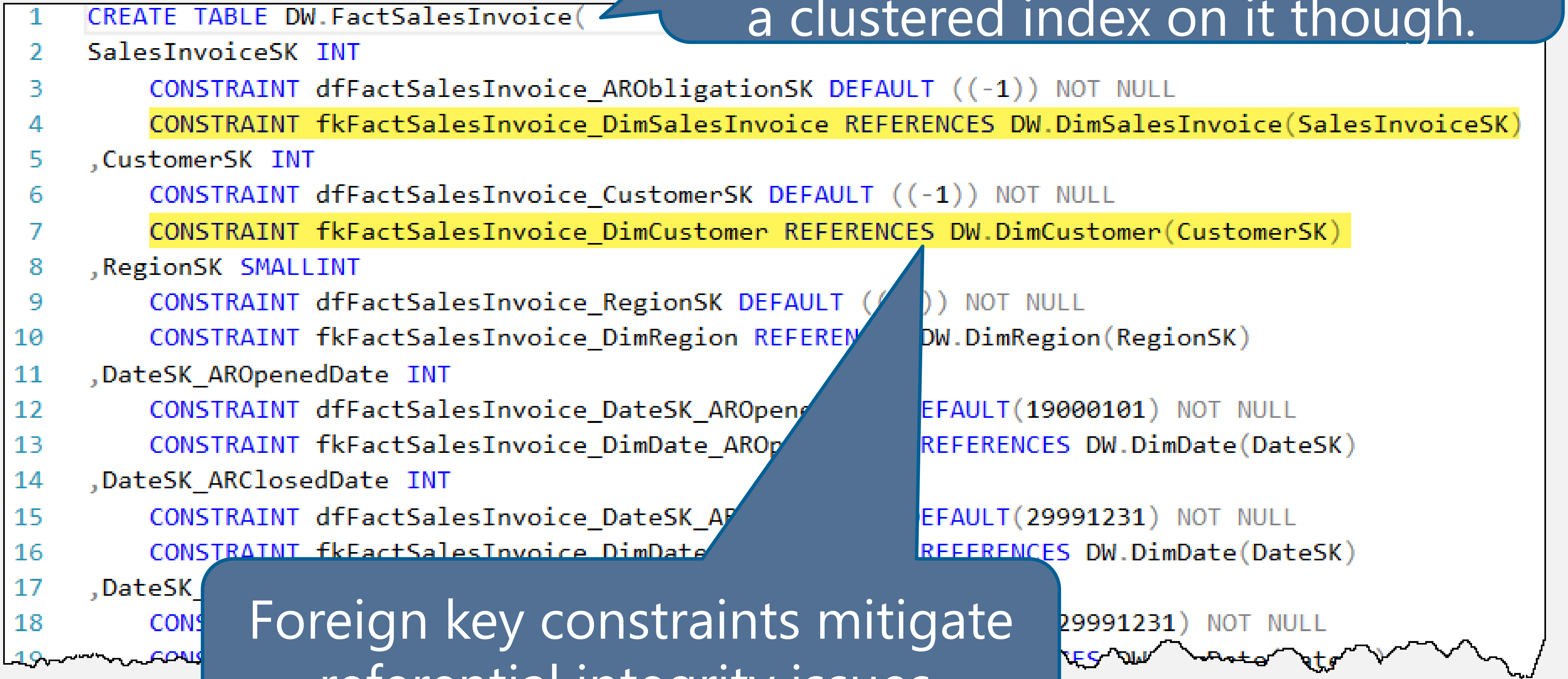
Optionally can use two types of Date defaults: one in the past, one in the future. Helps with 'Less than' or 'Greater than' predicates.



```
CREATE TABLE DW.FactSalesInvoice(
SalesInvoiceSK INT
    CONSTRAINT dfFactSalesInvoice_AR
    CONSTRAINT fkFactSalesInvoice_Di                                           eSK)
,CustomerSK INT
    CONSTRAINT dfFactSalesInvoice_Cu
    CONSTRAINT fkFactSalesInvoice_Dim
,RegionSK SMALLINT
    CONSTRAINT dfFactSalesInvoice_RegionSK DEFAULT ((       T NULL
    CONSTRAINT fkFactSalesInvoice_DimRegion REFERENCES D      Region(RegionSK)
,DateSK_AROpenedDate INT
    CONSTRAINT dfFactSalesInvoice_DateSK_AROpenedDate DEFAULT(19000101) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimDate_AROpenedDate REFERENCES DW.DimDate(DateSK)
,DateSK_ARClosedDate INT
    CONSTRAINT dfFactSalesInvoice_DateSK_ARClosedDate DEFAULT(29991231) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimDate_ARClosedDate REFERENCES DW.DimDate(DateSK)
,Dat
```

It's also fine for a date SK to be an actual date datatype instead of an integer.

Fact Design

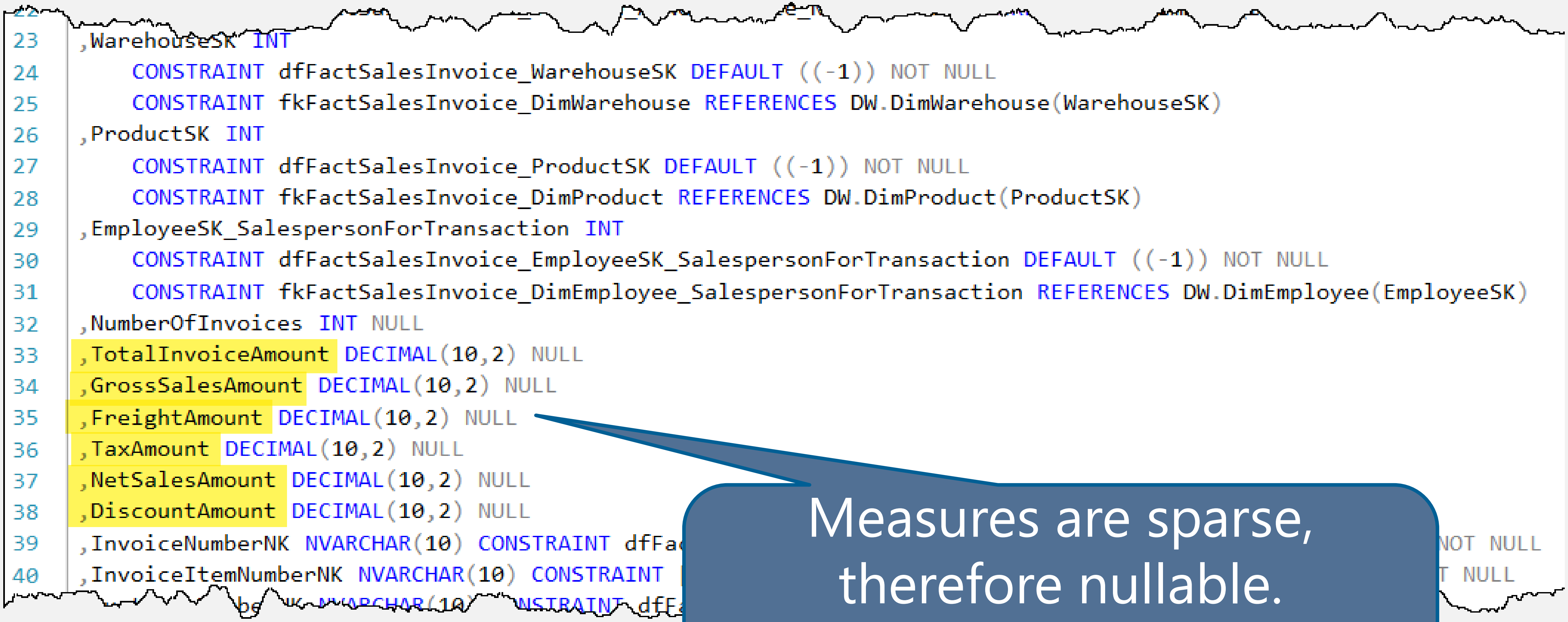


```
CREATE TABLE DW.FactSalesInvoice(
SalesInvoiceSK INT
    CONSTRAINT dfFactSalesInvoice_ARObligationSK DEFAULT ((-1)) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimSalesInvoice REFERENCES DW.DimSalesInvoice(SalesInvoiceSK)
,CustomerSK INT
    CONSTRAINT dfFactSalesInvoice_CustomerSK DEFAULT ((-1)) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimCustomer REFERENCES DW.DimCustomer(CustomerSK)
,RegionSK SMALLINT
    CONSTRAINT dfFactSalesInvoice_RegionSK DEFAULT ((    )) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimRegion REFEREN      DW.DimRegion(RegionSK)
,DateSK_AROpenedDate INT
    CONSTRAINT dfFactSalesInvoice_DateSK_AROpene      EFAULT(19000101) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimDate_AROp        REFERENCES DW.DimDate(DateSK)
,DateSK_ARClosedDate INT
    CONSTRAINT dfFactSalesInvoice_DateSK_AR           EFAULT(29991231) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimDate             REFERENCES DW.DimDate(DateSK)
,DateSK_
    CONS                                              29991231) NOT NULL
```

Having a PK in a fact is personal preference. Usually you don’t want a clustered index on it though.

Foreign key constraints mitigate referential integrity issues.

Fact Design

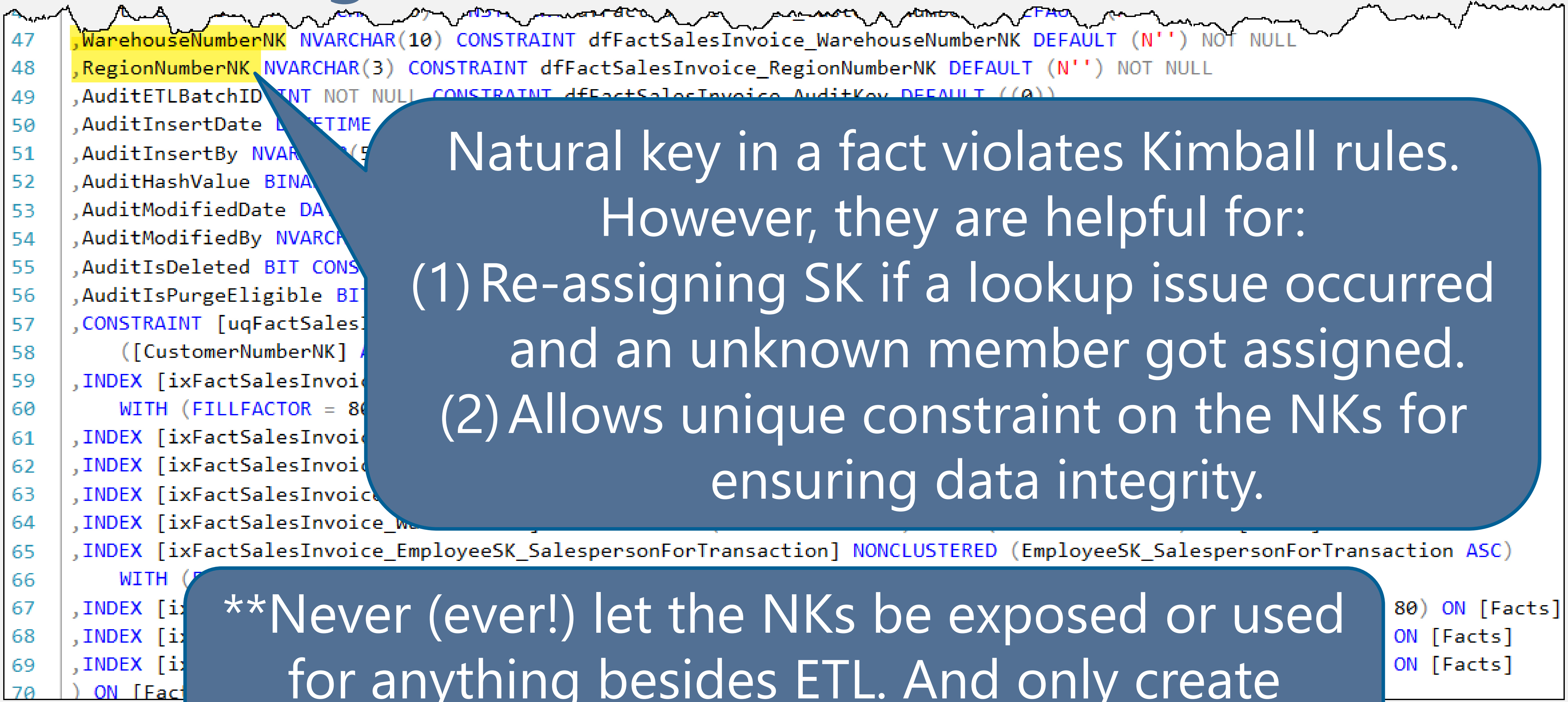


```
,WarehouseSK INT
    CONSTRAINT dfFactSalesInvoice_WarehouseSK DEFAULT ((-1)) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimWarehouse REFERENCES DW.DimWarehouse(WarehouseSK)
,ProductSK INT
    CONSTRAINT dfFactSalesInvoice_ProductSK DEFAULT ((-1)) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimProduct REFERENCES DW.DimProduct(ProductSK)
,EmployeeSK_SalespersonForTransaction INT
    CONSTRAINT dfFactSalesInvoice_EmployeeSK_SalespersonForTransaction DEFAULT ((-1)) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimEmployee_SalespersonForTransaction REFERENCES DW.DimEmployee(EmployeeSK)
,NumberOfInvoices INT NULL
,TotalInvoiceAmount DECIMAL(10,2) NULL
,GrossSalesAmount DECIMAL(10,2) NULL
,FreightAmount DECIMAL(10,2) NULL
,TaxAmount DECIMAL(10,2) NULL
,NetSalesAmount DECIMAL(10,2) NULL
,DiscountAmount DECIMAL(10,2) NULL
,InvoiceNumberNK NVARCHAR(10) CONSTRAINT dfFa                                                    NOT NULL
,InvoiceItemNumberNK NVARCHAR(10) CONSTRAINT                                                    T NULL
```

Measures are sparse, therefore nullable.

0s are not stored except in a factless fact table.

Fact Design



```
,WarehouseNumberNK NVARCHAR(10) CONSTRAINT dfFactSalesInvoice_WarehouseNumberNK DEFAULT (N'') NOT NULL
,RegionNumberNK NVARCHAR(3) CONSTRAINT dfFactSalesInvoice_RegionNumberNK DEFAULT (N'') NOT NULL
,AuditETLBatchID INT NOT NULL CONSTRAINT dfFactSalesInvoice_AuditKey DEFAULT ((0))
,AuditInsertDate D  ETIME
,AuditInsertBy NVAR   (5
,AuditHashValue BINA
,AuditModifiedDate DA
,AuditModifiedBy NVARCH
,AuditIsDeleted BIT CONS
,AuditIsPurgeEligible BI
,CONSTRAINT [uqFactSalesI
    ([CustomerNumberNK] A
,INDEX [ixFactSalesInvoi
    WITH (FILLFACTOR = 80
,INDEX [ixFactSalesInvoi
,INDEX [ixFactSalesInvoi
,INDEX [ixFactSalesInvoic
,INDEX [ixFactSalesInvoice_Wa
,INDEX [ixFactSalesInvoice_EmployeeSK_SalespersonForTransaction] NONCLUSTERED (EmployeeSK_SalespersonForTransaction ASC)
    WITH (F
,INDEX [i                                                                         80) ON [Facts]
,INDEX [i                                                                         ON [Facts]
,INDEX [i                                                                         ON [Facts]
) ON [Fact
```

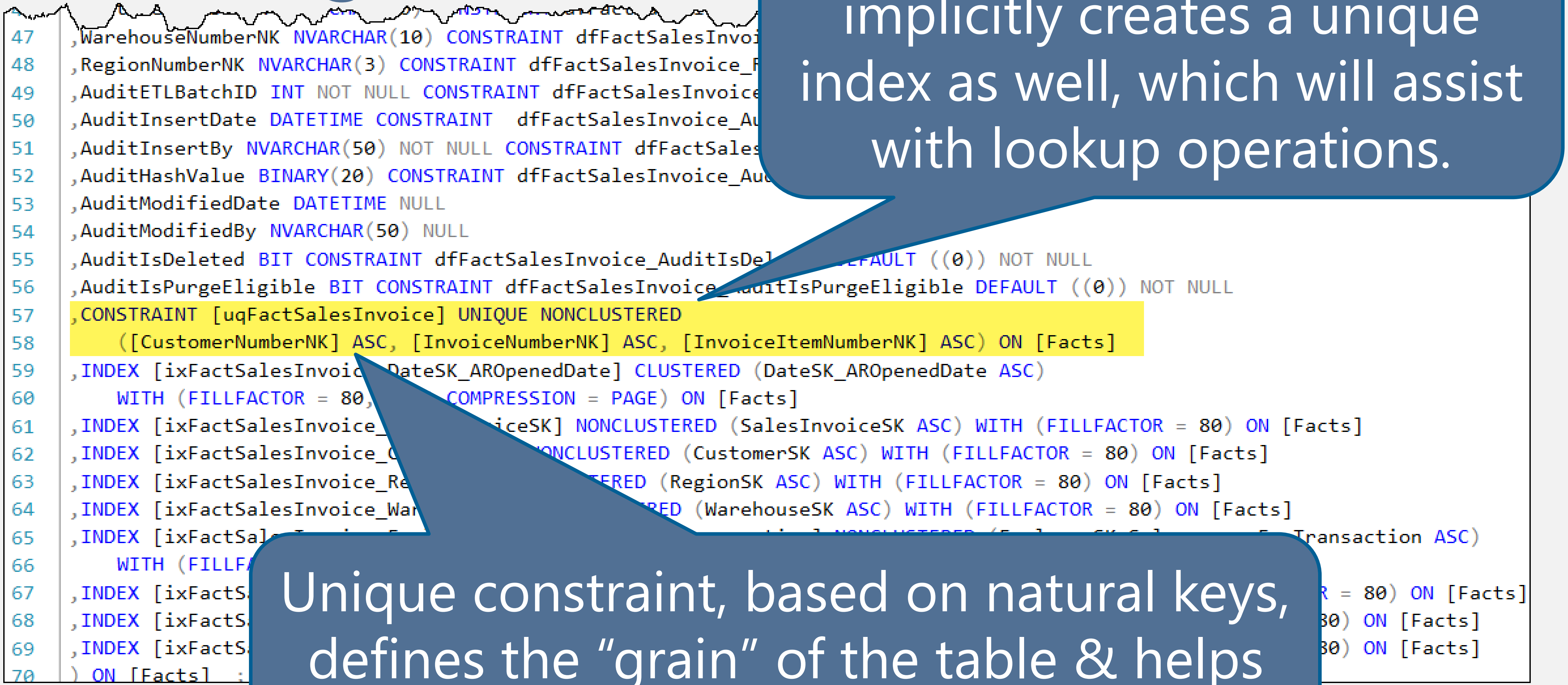
Natural key in a fact violates Kimball rules. However, they are helpful for:

(1) Re-assigning SK if a lookup issue occurred and an unknown member got assigned.

(2) Allows unique constraint on the NKs for ensuring data integrity.

Never (ever!) let the NKs be exposed or used for anything besides ETL. And only create minimum # of NKs to identify the row.

Fact Design

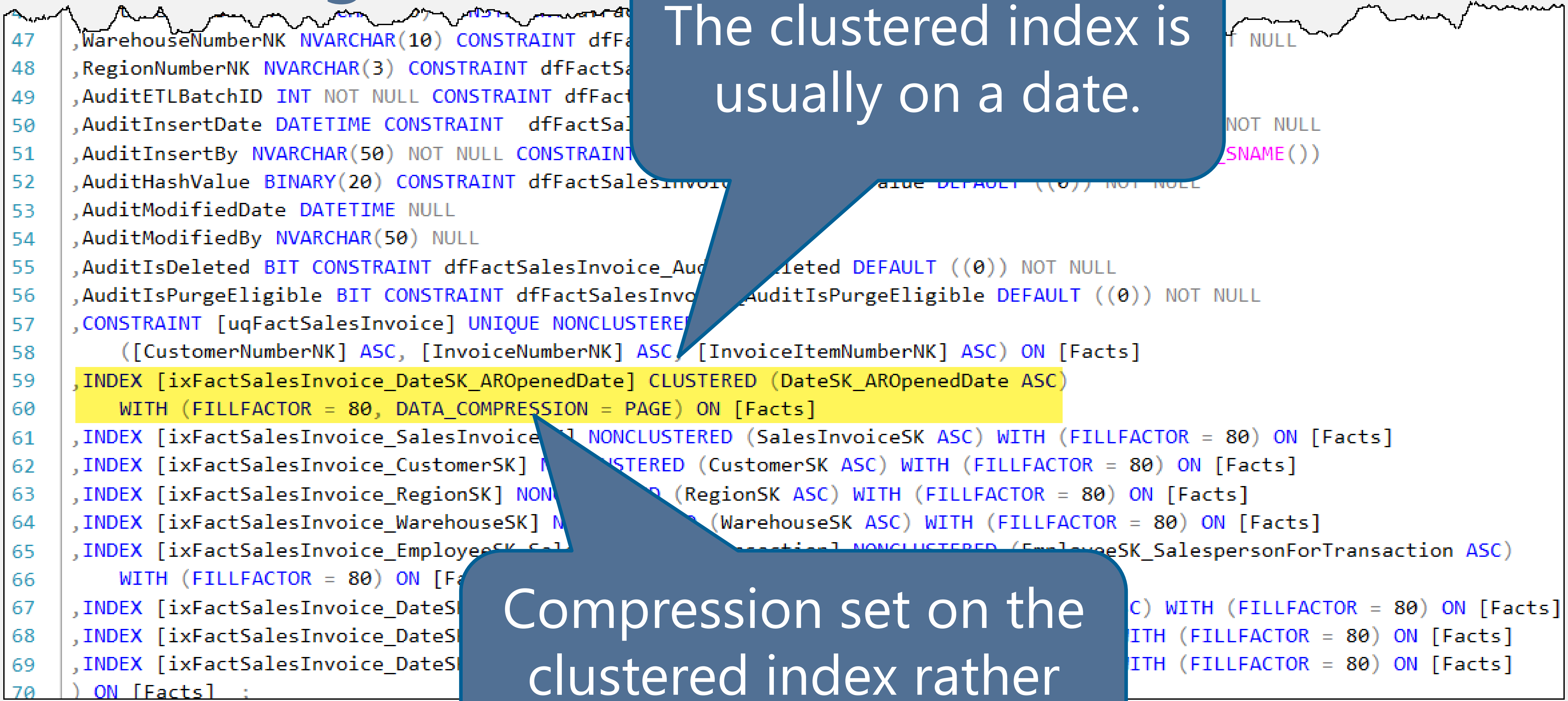


```
,WarehouseNumberNK NVARCHAR(10) CONSTRAINT dfFactSalesInvoi
,RegionNumberNK NVARCHAR(3) CONSTRAINT dfFactSalesInvoice_F
,AuditETLBatchID INT NOT NULL CONSTRAINT dfFactSalesInvoice
,AuditInsertDate DATETIME CONSTRAINT  dfFactSalesInvoice_Au
,AuditInsertBy NVARCHAR(50) NOT NULL CONSTRAINT dfFactSales
,AuditHashValue BINARY(20) CONSTRAINT dfFactSalesInvoice_Au
,AuditModifiedDate DATETIME NULL
,AuditModifiedBy NVARCHAR(50) NULL
,AuditIsDeleted BIT CONSTRAINT dfFactSalesInvoice_AuditIsDel      EFAULT ((0)) NOT NULL
,AuditIsPurgeEligible BIT CONSTRAINT dfFactSalesInvoice_AuditIsPurgeEligible DEFAULT ((0)) NOT NULL
,CONSTRAINT [uqFactSalesInvoice] UNIQUE NONCLUSTERED
    ([CustomerNumberNK] ASC, [InvoiceNumberNK] ASC, [InvoiceItemNumberNK] ASC) ON [Facts]
,INDEX [ixFactSalesInvoic  DateSK_AROpenedDate] CLUSTERED (DateSK_AROpenedDate ASC)
    WITH (FILLFACTOR = 80,      COMPRESSION = PAGE) ON [Facts]
,INDEX [ixFactSalesInvoice_      iceSK] NONCLUSTERED (SalesInvoiceSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
,INDEX [ixFactSalesInvoice_C      NCLUSTERED (CustomerSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
,INDEX [ixFactSalesInvoice_Re      TERED (RegionSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
,INDEX [ixFactSalesInvoice_War      RED (WarehouseSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
,INDEX [ixFactSal      Transaction ASC)
    WITH (FILLF
,INDEX [ixFactS      R = 80) ON [Facts]
,INDEX [ixFactS      80) ON [Facts]
,INDEX [ixFactS      80) ON [Facts]
) ON [Facts] ;
```

The unique constraint implicitly creates a unique index as well, which will assist with lookup operations.

Unique constraint, based on natural keys, defines the “grain” of the table & helps identify data quality issues.

Fact Design

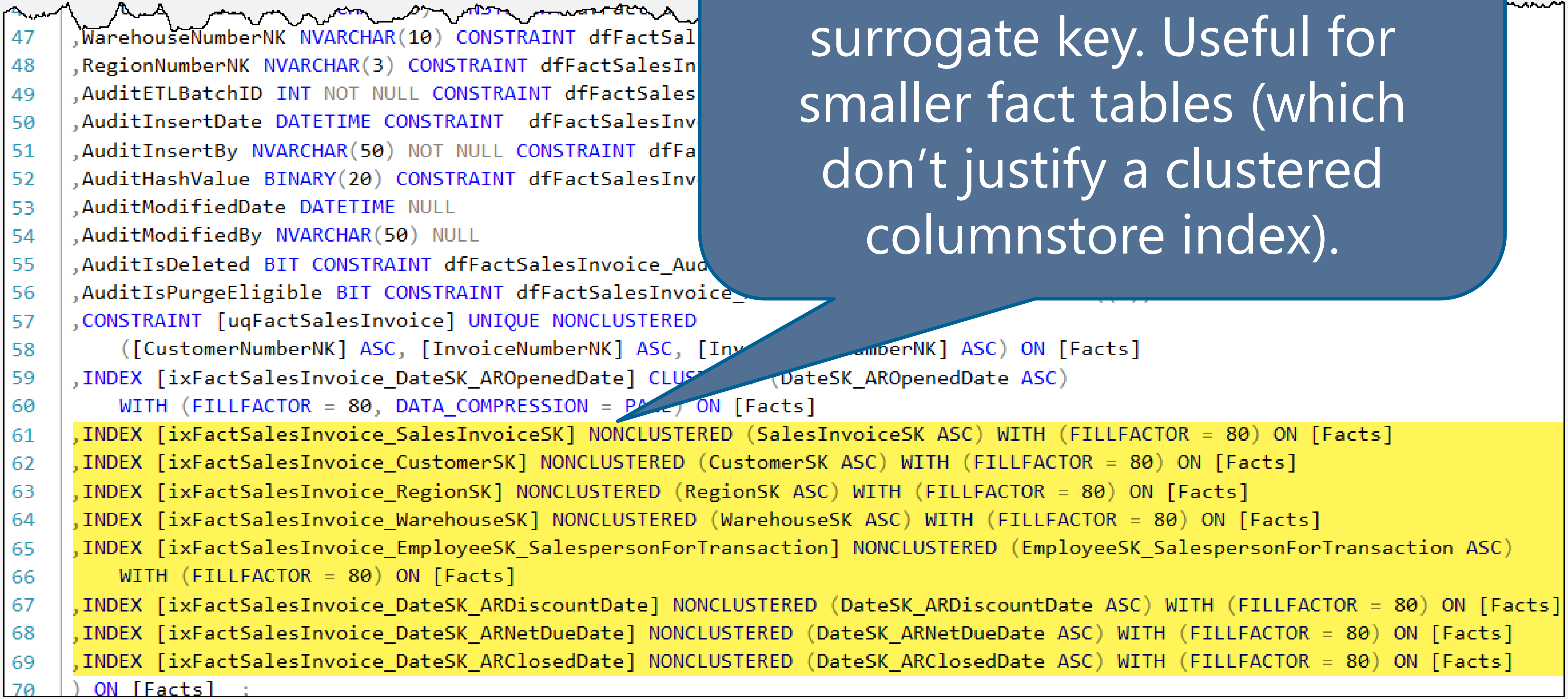


```
,WarehouseNumberNK NVARCHAR(10) CONSTRAINT dfFa                                    T NULL
,RegionNumberNK NVARCHAR(3) CONSTRAINT dfFactSa
,AuditETLBatchID INT NOT NULL CONSTRAINT dfFact
,AuditInsertDate DATETIME CONSTRAINT  dfFactSal                                   NOT NULL
,AuditInsertBy NVARCHAR(50) NOT NULL CONSTRAINT                                   SNAME())
,AuditHashValue BINARY(20) CONSTRAINT dfFactSalesInvoic     alue DEFAULT ((0)) NOT NULL
,AuditModifiedDate DATETIME NULL
,AuditModifiedBy NVARCHAR(50) NULL
,AuditIsDeleted BIT CONSTRAINT dfFactSalesInvoice_Aud     leted DEFAULT ((0)) NOT NULL
,AuditIsPurgeEligible BIT CONSTRAINT dfFactSalesInvo   AuditIsPurgeEligible DEFAULT ((0)) NOT NULL
,CONSTRAINT [uqFactSalesInvoice] UNIQUE NONCLUSTERED
    ([CustomerNumberNK] ASC, [InvoiceNumberNK] ASC, [InvoiceItemNumberNK] ASC) ON [Facts]
,INDEX [ixFactSalesInvoice_DateSK_AROpenedDate] CLUSTERED (DateSK_AROpenedDate ASC)
    WITH (FILLFACTOR = 80, DATA_COMPRESSION = PAGE) ON [Facts]
,INDEX [ixFactSalesInvoice_SalesInvoice  ] NONCLUSTERED (SalesInvoiceSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
,INDEX [ixFactSalesInvoice_CustomerSK] N     STERED (CustomerSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
,INDEX [ixFactSalesInvoice_RegionSK] NON       D (RegionSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
,INDEX [ixFactSalesInvoice_WarehouseSK] N        (WarehouseSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
,INDEX [ixFactSalesInvoice_Employee           saction] NONCLUSTERED (EmployeeSK_SalespersonForTransaction ASC)
    WITH (FILLFACTOR = 80) ON [Fa
,INDEX [ixFactSalesInvoice_DateS                                  C) WITH (FILLFACTOR = 80) ON [Facts]
,INDEX [ixFactSalesInvoice_DateS                                  ITH (FILLFACTOR = 80) ON [Facts]
,INDEX [ixFactSalesInvoice_DateS                                  ITH (FILLFACTOR = 80) ON [Facts]
) ON [Facts] ;
```

The clustered index is usually on a date.

Compression set on the clustered index rather than the table.

Fact Design



```
,WarehouseNumberNK NVARCHAR(10) CONSTRAINT dfFactSal
,RegionNumberNK NVARCHAR(3) CONSTRAINT dfFactSalesIn
,AuditETLBatchID INT NOT NULL CONSTRAINT dfFactSales
,AuditInsertDate DATETIME CONSTRAINT  dfFactSalesInv
,AuditInsertBy NVARCHAR(50) NOT NULL CONSTRAINT dfFa
,AuditHashValue BINARY(20) CONSTRAINT dfFactSalesInv
,AuditModifiedDate DATETIME NULL
,AuditModifiedBy NVARCHAR(50) NULL
,AuditIsDeleted BIT CONSTRAINT dfFactSalesInvoice_Aud
,AuditIsPurgeEligible BIT CONSTRAINT dfFactSalesInvoice_
,CONSTRAINT [uqFactSalesInvoice] UNIQUE NONCLUSTERED
    ([CustomerNumberNK] ASC, [InvoiceNumberNK] ASC, [Inv          mberNK] ASC) ON [Facts]
,INDEX [ixFactSalesInvoice_DateSK_AROpenedDate] CLUS          (DateSK_AROpenedDate ASC)
    WITH (FILLFACTOR = 80, DATA_COMPRESSION = PA    ) ON [Facts]
,INDEX [ixFactSalesInvoice_SalesInvoiceSK] NONCLUSTERED (SalesInvoiceSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
,INDEX [ixFactSalesInvoice_CustomerSK] NONCLUSTERED (CustomerSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
,INDEX [ixFactSalesInvoice_RegionSK] NONCLUSTERED (RegionSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
,INDEX [ixFactSalesInvoice_WarehouseSK] NONCLUSTERED (WarehouseSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
,INDEX [ixFactSalesInvoice_EmployeeSK_SalespersonForTransaction] NONCLUSTERED (EmployeeSK_SalespersonForTransaction ASC)
    WITH (FILLFACTOR = 80) ON [Facts]
,INDEX [ixFactSalesInvoice_DateSK_ARDiscountDate] NONCLUSTERED (DateSK_ARDiscountDate ASC) WITH (FILLFACTOR = 80) ON [Facts]
,INDEX [ixFactSalesInvoice_DateSK_ARNetDueDate] NONCLUSTERED (DateSK_ARNetDueDate ASC) WITH (FILLFACTOR = 80) ON [Facts]
,INDEX [ixFactSalesInvoice_DateSK_ARClosedDate] NONCLUSTERED (DateSK_ARClosedDate ASC) WITH (FILLFACTOR = 80) ON [Facts]
) ON [Facts]  ;
```

Nonclustered index on each surrogate key. Useful for smaller fact tables (which don’t justify a clustered columnstore index).

When to Use
Columnstore Indexes or Partitioning

Handling Larger Fact Tables

Clustered Columnstore Index

Useful for:

- ✓ Reducing data storage due to compression of redundant values
- ✓ Improving query times for large datasets
- ✓ Improving query times due to reduced I/O (ex: column elimination)

Table Partitioning

Useful for:

- ✓ Improving data load times due to partition switching
- ✓ Flexibility for maintenance on larger tables
- ✓ Improving query performance *(possibly)* due parallelism & partition elimination behavior

Clustered Columnstore Index

Simplified & conceptual

Rowstore:

Page

CounterName	Disk	DateMeasurementTaken	TimeMeasurementTaken	Measurement
Avg. Disk sec/Read	G:\	1/30/2017	4:48:41 PM	0.01818
Avg. Disk sec/Read	L:\	1/30/2017	4:48:41 PM	0.00385
Avg. Disk sec/Read	T:\	1/30/2017	4:48:41 PM	0.00780
Avg. Disk Bytes/Read	G:\	1/30/2017	4:48:41 PM	53120.73782
Avg. Disk Bytes/Read	L:\	1/30/2017	4:48:41 PM	42362.51095
Avg. Disk Bytes/Read	T:\	1/30/2017	4:48:41 PM	47951.40657

Columnstore:

Page

CounterName
Avg. Disk sec/Read
Avg. Disk Bytes/Read

Page

Disk
G:\
L:\
T:\

Page

DateMeasurementTaken
1/30/2017

Page

TimeMeasurementTaken
4:48:41 PM

Page

Measurement
0.01818
0.00385
0.00780
53120.73782
42362.51095
47951.40657

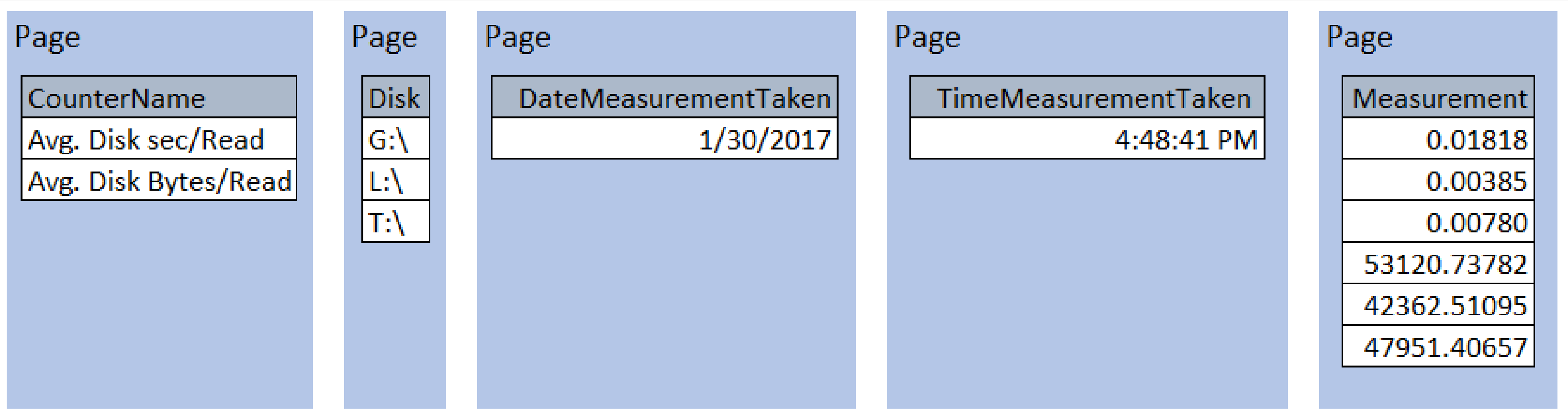
Reduced storage for low cardinality columns

Clustered Columnstore Index

Simplified & conceptual

CCI most suitable for:

- ✓ Tables over 1 million rows



Page

CounterName
Avg. Disk sec/Read
Avg. Disk Bytes/Read

Page

Disk
G:\
L:\
T:\

Page

DateMeasurementTaken
1/30/2017

Page

TimeMeasurementTaken
4:48:41 PM

Page

Measurement
0.01818
0.00385
0.00780
53120.73782
42362.51095
47951.40657

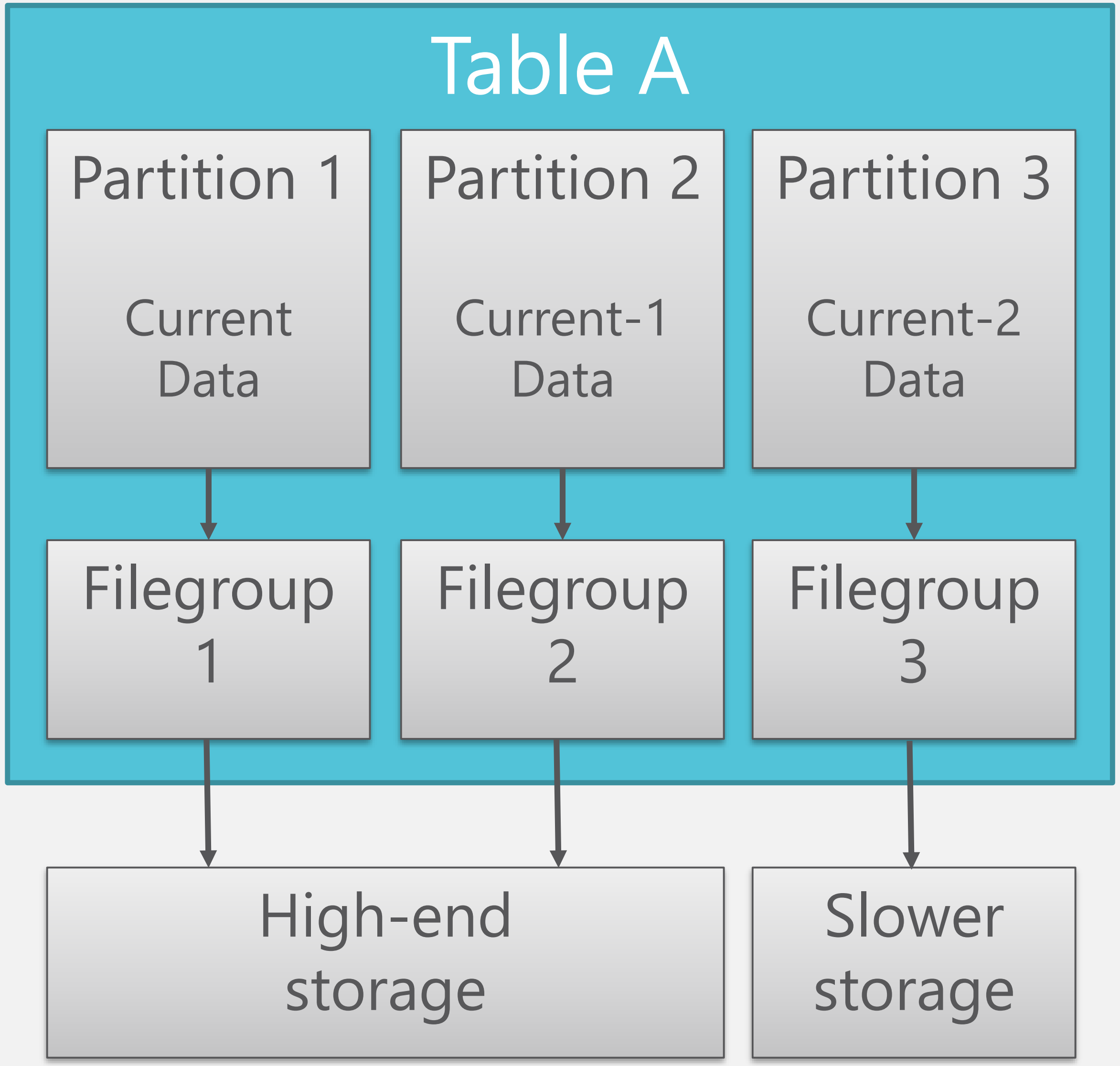
- ✓ Data structured in a denormalized star schema format (DW not OLTP)
- ✓ Support for analytical query workload which scans a large number of rows, and retrieves few columns
- ✓ Data which is not frequently updated (‘cold’ data not ‘hot’)
- ✓ Can selectively be used on insert-oriented workloads (ex: IoT)

(A nonclustered columnstore index targets analytical queries on an OLTP rather than a data warehouse.)

Partitioned Table

Useful for:

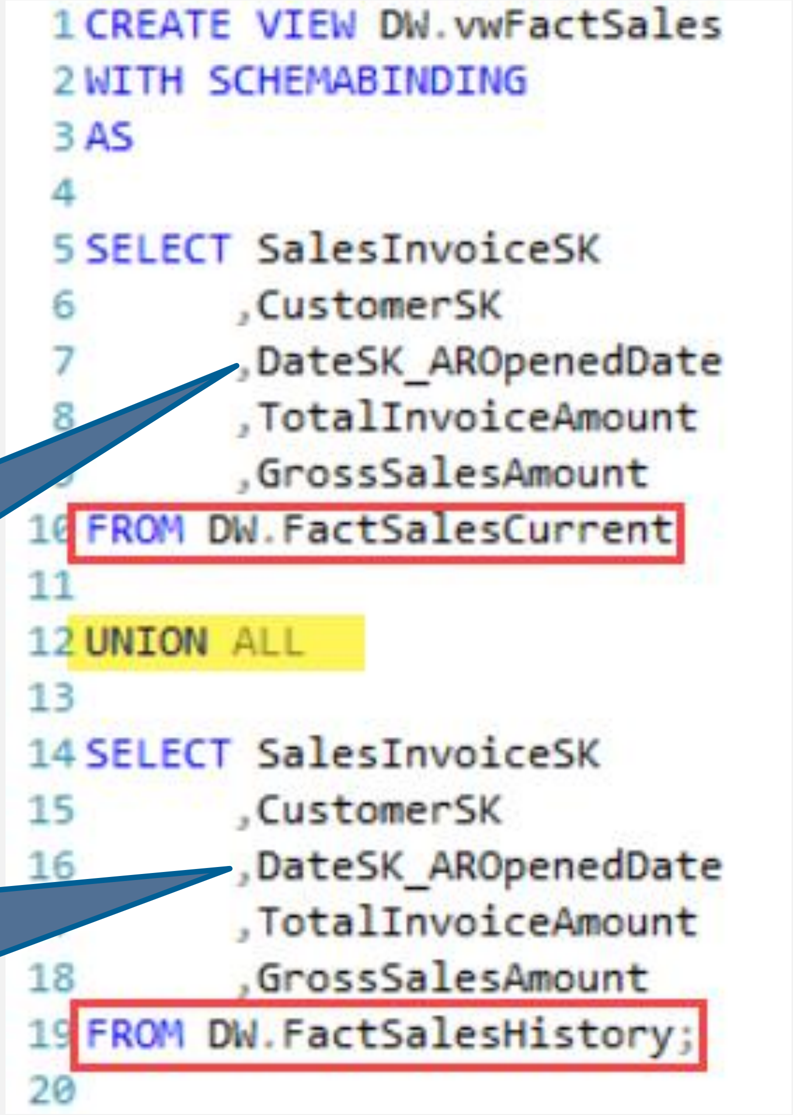
- ✓ Speeding up ETL processes
  - ✓ Large datasets (50GB+)
  - ✓ Small maintenance windows
  - ✓ Use of a sliding window
- ✓ Storage of partitions on separate drives (filegroups)
  - ✓ Older (cold) data on cheaper storage
  - ✓ Historical data on read-only filegroup
- ✓ Speeding up queries *(possibly)*
  - ✓ Partition elimination
  - ✓ Parallelism



Partitioned View

Useful for:

- ✓ Query performance (similar to partitioned table)
- ✓ Sharing of a single table (“partition”) across multiple views
- ✓ Displaying info from > 1 database or server (via a linked server)



```sql
CREATE VIEW DW.vwFactSales
WITH SCHEMABINDING
AS

SELECT SalesInvoiceSK
        ,CustomerSK
        ,DateSK_AROpenedDate
        ,TotalInvoiceAmount
        ,GrossSalesAmount
FROM DW.FactSalesCurrent

UNION ALL

SELECT SalesInvoiceSK
        ,CustomerSK
        ,DateSK_AROpenedDate
        ,TotalInvoiceAmount
        ,GrossSalesAmount
FROM DW.FactSalesHistory;
```

Requires “Check” constraints on the underlying tables (usually on a date column)

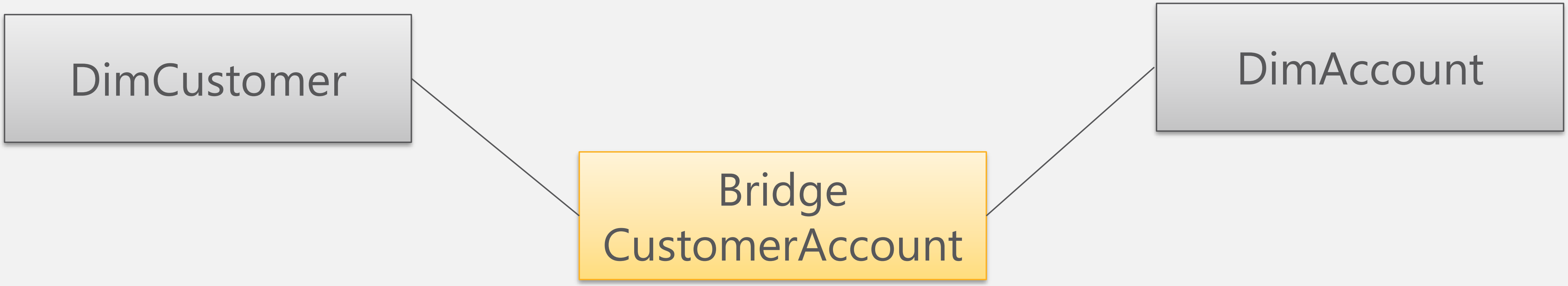
Data Warehouse Tips

Handling Many-to-Many Scenarios

Classic many-to-many scenarios:

✓ A sales order is for many products, and a product is on many sales orders

✓ A customer has multiple bank accounts, and a bank account belongs to multiple customers



Ways to Track History in a DW

Most common options for tracking history:

1. Slowly changing dimension
2. Fact snapshot tables
3. Timestamp tracking fact

New option in SQL Server 2016:

4. Temporal data tables → *Not a full replacement for slowly changing dimensions, but definitely useful for auditing*

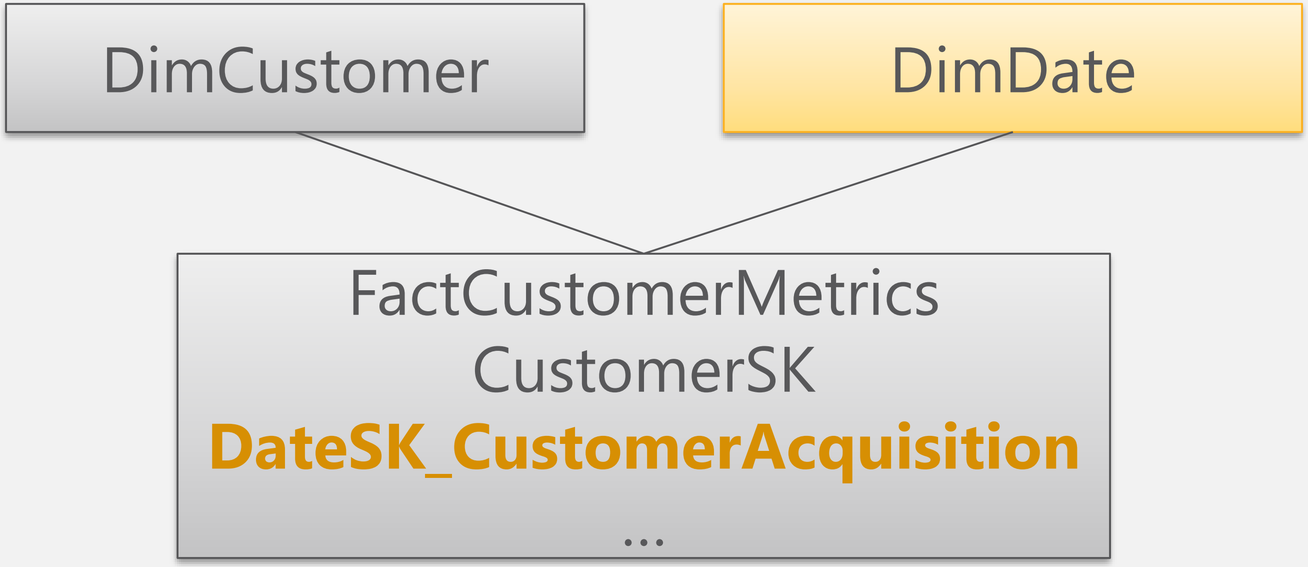
“Smart Dates” vs. “Dumb Dates” in a DW

A “dumb date” is just an attribute:

DimCustomer

CustomerSK
CustomerNK
CustomerAcquisitionDate
…

A “smart date” relates to a full-fledged Date dimension which allows significant time analysis capabilities:



DimCustomer

DimDate

FactCustomerMetrics
CustomerSK
DateSK_CustomerAcquisition
…

Handling of Nulls in Dimensions

Remember the NOT NULL and default constraints

Rule of thumb is to **avoid nulls** in attribute columns.

What happens with this:

```
SELECT CustomerType WHERE CustomerType <> ‘Retail’
```

Too easy to forget:

```
SELECT CustomerType WHERE CustomerType <> ‘Retail’
                              OR CustomerType IS NULL
```

Handling of Nulls in Facts

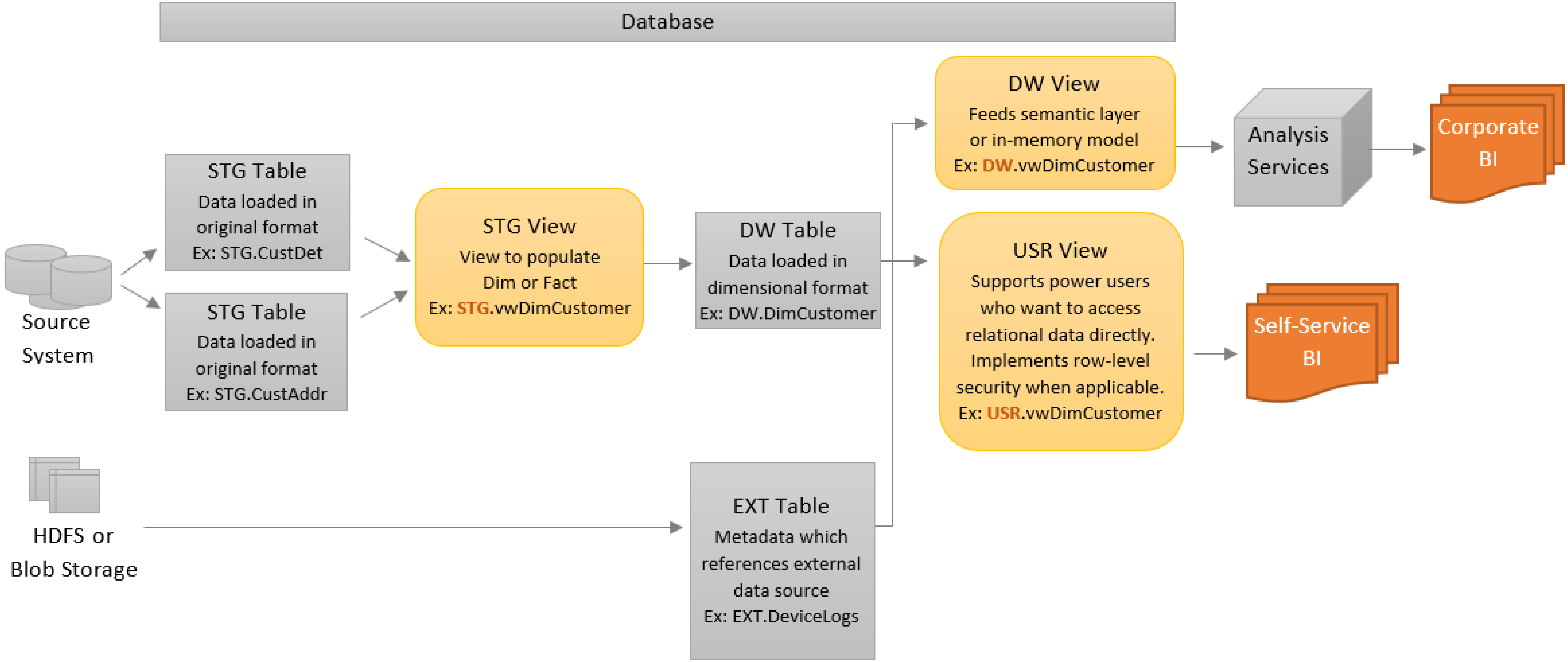
Best practice is to **avoid nulls in foreign keys**. (However, nulls are ok for a measure.)

By using an ‘unknown member’ relationship to the dimension, you can:

- ✓ Safely do inner joins
- ✓ Allow the fact record to be inserted & meet referential integrity
- ✓ Allow the fact record to be inserted which avoids understating measurement amounts

  Ex: Just because one key is unknown, such as an EmployeeSK for who rang up the sale, should the sale not be counted?

Views Customized for Different Purposes


Database
Source System
HDFS or Blob Storage
STG Table
Data loaded in original format
Ex: STG.CustDet
STG Table
Data loaded in original format
Ex: STG.CustAddr
STG View
View to populate Dim or Fact
Ex: STG.vwDimCustomer
DW Table
Data loaded in dimensional format
Ex: DW.DimCustomer
EXT Table
Metadata which references external data source
Ex: EXT.DeviceLogs
DW View
Feeds semantic layer or in-memory model
Ex: DW.vwDimCustomer
USR View
Supports power users who want to access relational data directly. Implements row-level security when applicable.
Ex: USR.vwDimCustomer
Analysis Services
Corporate BI
Self-Service BI

Recap of Important DW Design Principles

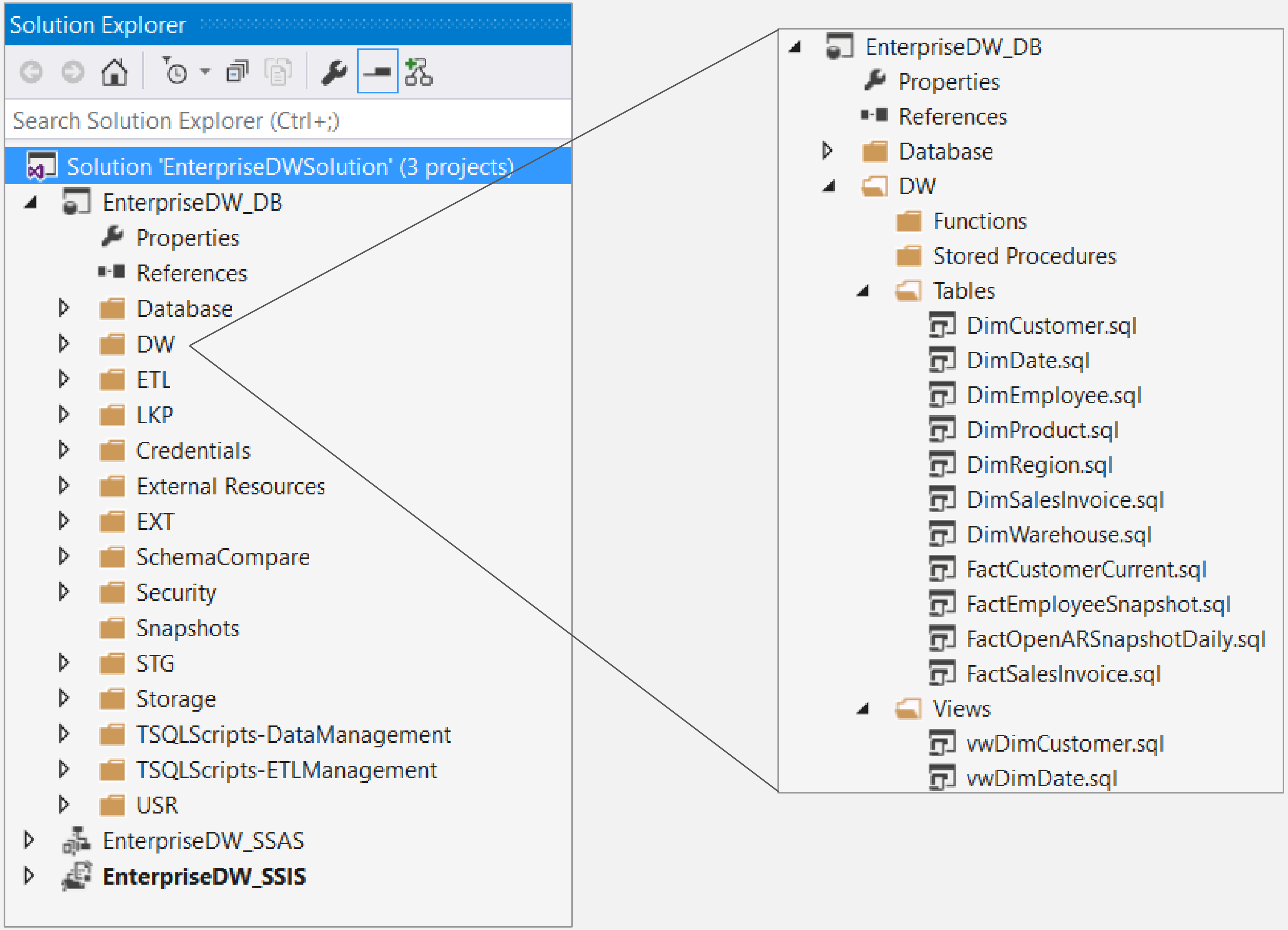
✓Staging as a “kitchen” area
✓Integrate data from multiple systems to increase its value
✓Denormalize the data into a star schema
✓A column exists in one and only one place in the star schema
✓Avoid snowflake design most of the time
✓Use surrogate keys which are independent from source systems
✓Use conformed dimensions
✓Know the grain of every table
✓Have a strategy for handling changes, and for storage of history
✓Store the lowest level of detail that you can
✓Use an ‘unknown member’ to avoid understating facts
✓Transform the data, but don’t “fix” it in the DW
✓Structure your dimensional model around business processes

Recap of Important DW Design Principles

- ✓Design facts around a single business event
- ✓Always use friendly names & descriptions
- ✓Use an explicit date dimension in a “role-playing” way
- ✓Utilize bridge tables to handle many-to-many scenarios
- ✓Plan for complexities such as:
    - ✓Header/line data
    - ✓Semi-additive facts
    - ✓Multiple currencies
    - ✓Multiple units of measure
    - ✓Alternate hierarchies and calculations per business units
    - ✓Allocation of measures in a snowflake design
    - ✓Reporting of what didn’t occur (factless facts)
    - ✓Dimensional only analysis

SSDT “Database Project” Tips

Database Project Format



Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'EnterpriseDWSolution' (3 projects)
  - EnterpriseDW_DB
    - Properties
    - References
    - Database
    - DW
    - ETL
    - LKP
    - Credentials
    - External Resources
    - EXT
    - SchemaCompare
    - Security
    - Snapshots
    - STG
    - Storage
    - TSQLScripts-DataManagement
    - TSQLScripts-ETLManagement
    - USR
  - EnterpriseDW_SSAS
  - **EnterpriseDW_SSIS**

- EnterpriseDW_DB
  - Properties
  - References
  - Database
  - DW
    - Functions
    - Stored Procedures
    - Tables
      - DimCustomer.sql
      - DimDate.sql
      - DimEmployee.sql
      - DimProduct.sql
      - DimRegion.sql
      - DimSalesInvoice.sql
      - DimWarehouse.sql
      - FactCustomerCurrent.sql
      - FactEmployeeSnapshot.sql
      - FactOpenARSnapshotDaily.sql
      - FactSalesInvoice.sql
    - Views
      - vwDimCustomer.sql
      - vwDimDate.sql

This project is organized by:

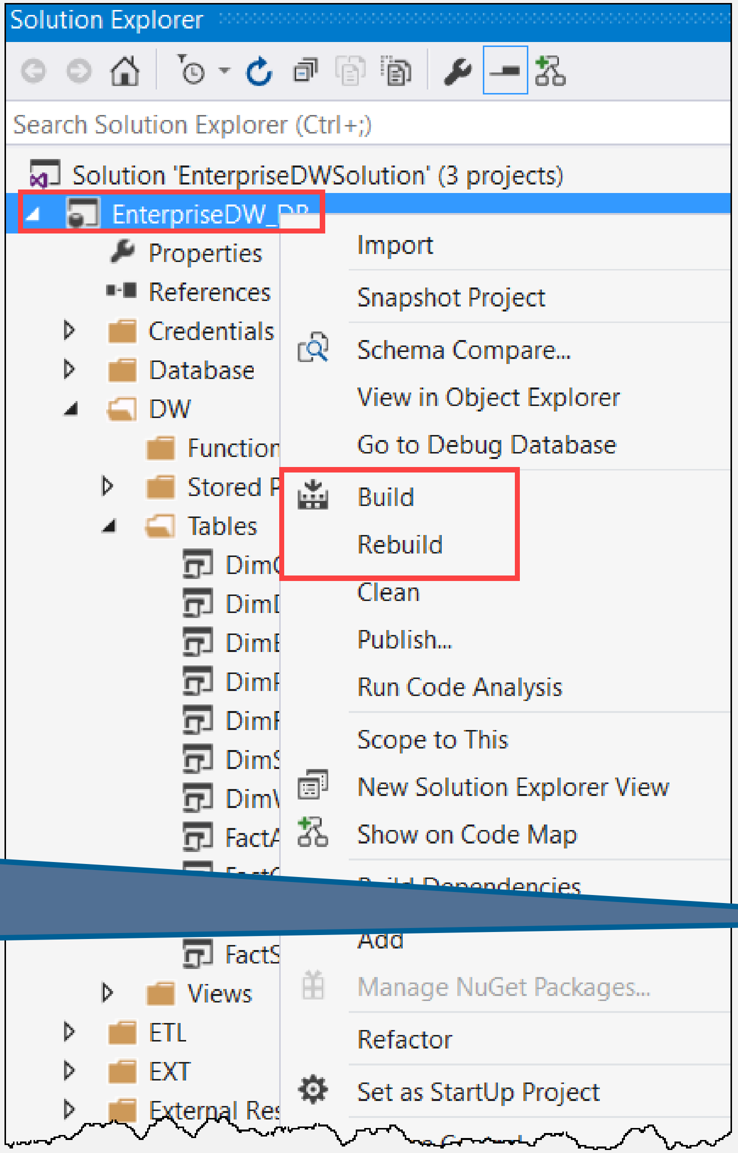
1 – Schema
(or Category)

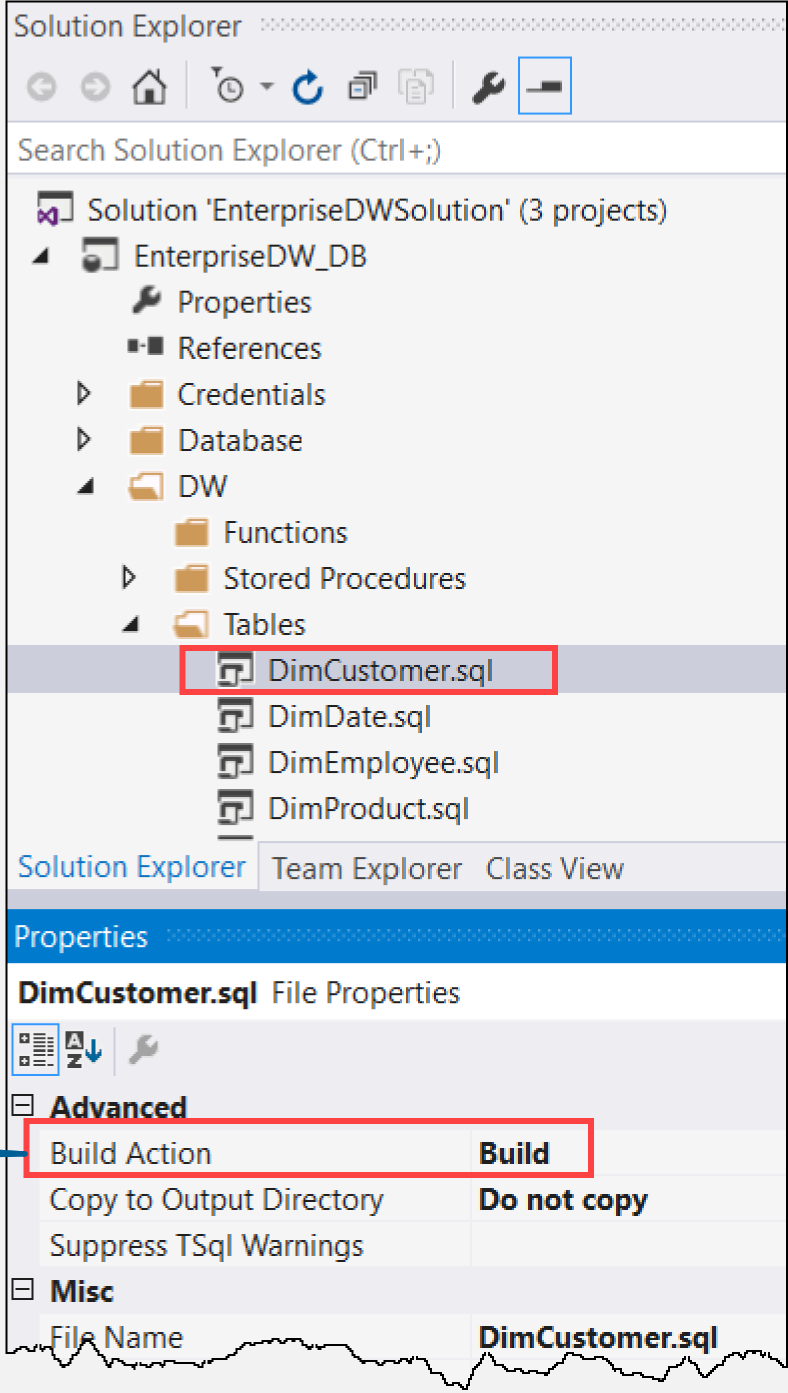
2 – Object Type

3 – Object

Building the Database Project

Build frequently to verify no errors or missing references





DimCustomer.sql File Properties

Advanced

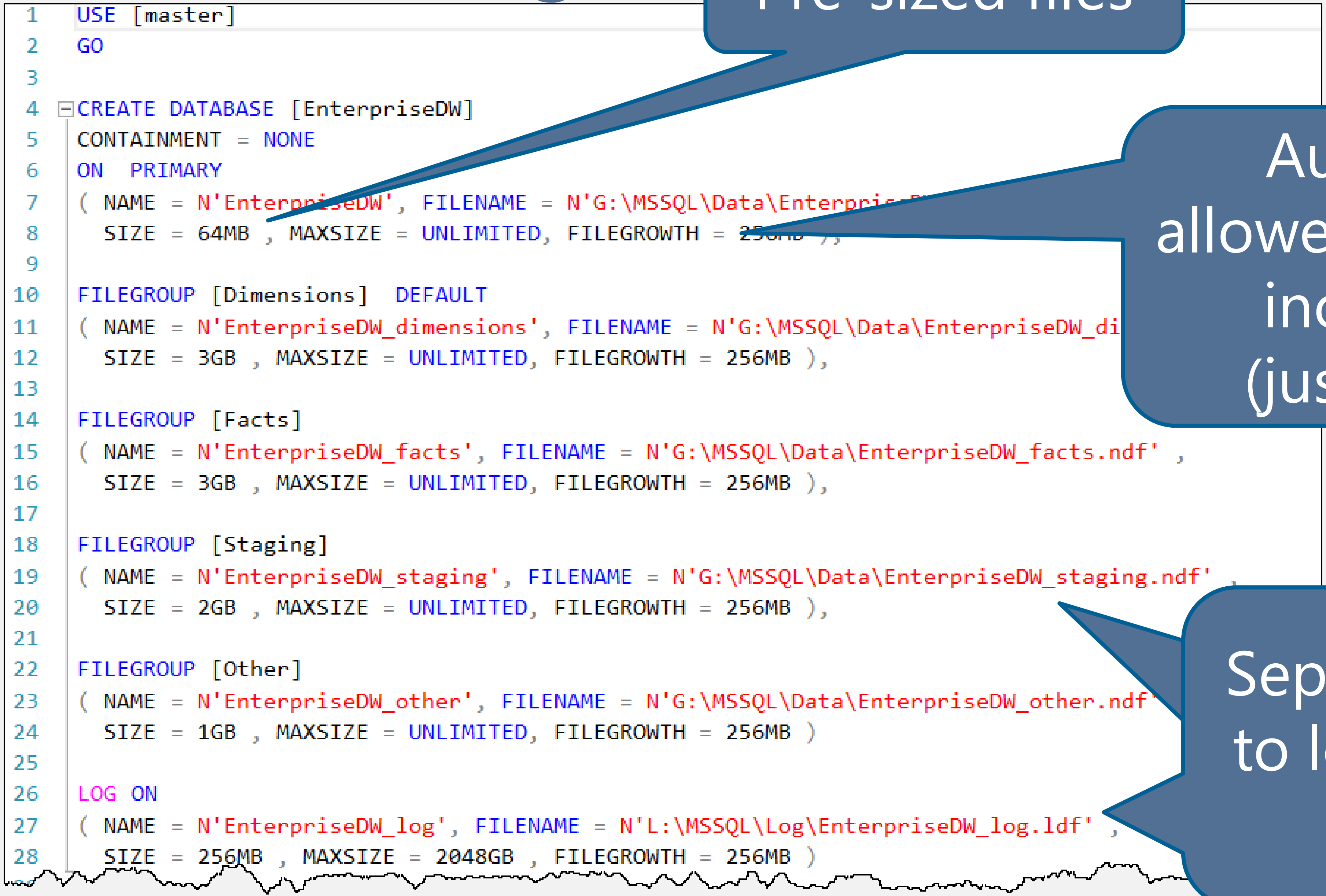
Build Action	**Build**
Copy to Output Directory	**Do not copy**
Suppress TSql Warnings	

Misc

File Name	**DimCustomer.sql**

Nearly all objects should be set to Build

Database Design



```
USE [master]
GO

CREATE DATABASE [EnterpriseDW]
CONTAINMENT = NONE
ON  PRIMARY
( NAME = N'EnterpriseDW', FILENAME = N'G:\MSSQL\Data\Enterpri
  SIZE = 64MB , MAXSIZE = UNLIMITED, FILEGROWTH = 256MB ),

FILEGROUP [Dimensions]  DEFAULT
( NAME = N'EnterpriseDW_dimensions', FILENAME = N'G:\MSSQL\Data\EnterpriseDW_di
  SIZE = 3GB , MAXSIZE = UNLIMITED, FILEGROWTH = 256MB ),

FILEGROUP [Facts]
( NAME = N'EnterpriseDW_facts', FILENAME = N'G:\MSSQL\Data\EnterpriseDW_facts.ndf' ,
  SIZE = 3GB , MAXSIZE = UNLIMITED, FILEGROWTH = 256MB ),

FILEGROUP [Staging]
( NAME = N'EnterpriseDW_staging', FILENAME = N'G:\MSSQL\Data\EnterpriseDW_staging.ndf' ,
  SIZE = 2GB , MAXSIZE = UNLIMITED, FILEGROWTH = 256MB ),

FILEGROUP [Other]
( NAME = N'EnterpriseDW_other', FILENAME = N'G:\MSSQL\Data\EnterpriseDW_other.ndf'
  SIZE = 1GB , MAXSIZE = UNLIMITED, FILEGROWTH = 256MB )

LOG ON
( NAME = N'EnterpriseDW_log', FILENAME = N'L:\MSSQL\Log\EnterpriseDW_log.ldf' ,
  SIZE = 256MB , MAXSIZE = 2048GB , FILEGROWTH = 256MB )
```

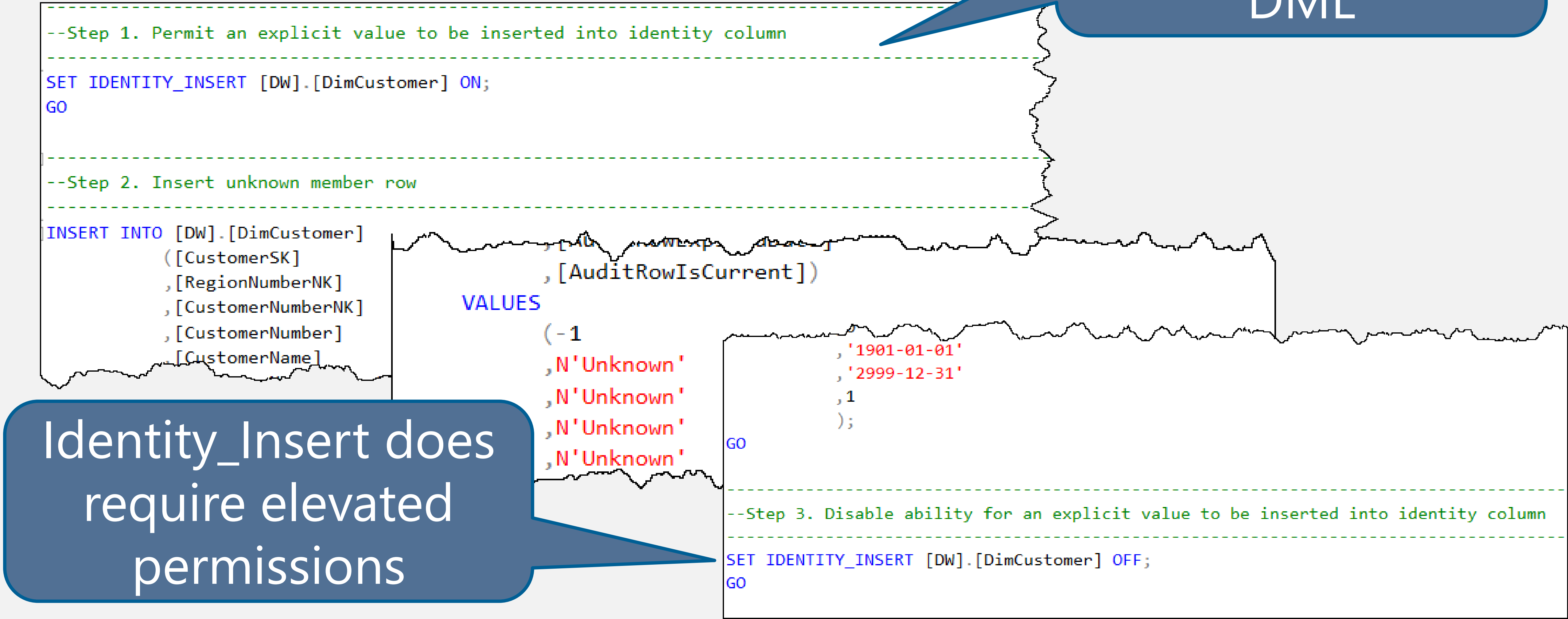
Pre-sized files

Auto-grow allowed in sizeable increments (just in case)

Separate disks to locate data & log

Unknown Member Row

The SK reference in a fact table if the real value is unknown or does not exist.



```
--------------------------------------------------------------------------------
--Step 1. Permit an explicit value to be inserted into identity column
--------------------------------------------------------------------------------
SET IDENTITY_INSERT [DW].[DimCustomer] ON;
GO


--------------------------------------------------------------------------------
--Step 2. Insert unknown member row
--------------------------------------------------------------------------------
INSERT INTO [DW].[DimCustomer]
         ([CustomerSK]
         ,[RegionNumberNK]
         ,[CustomerNumberNK]
         ,[CustomerNumber]
         ,[CustomerName]
```

```
         ,[AuditRowIsCurrent])
   VALUES
         (-1
         ,N'Unknown'
         ,N'Unknown'
         ,N'Unknown'
         ,N'Unknown'
```

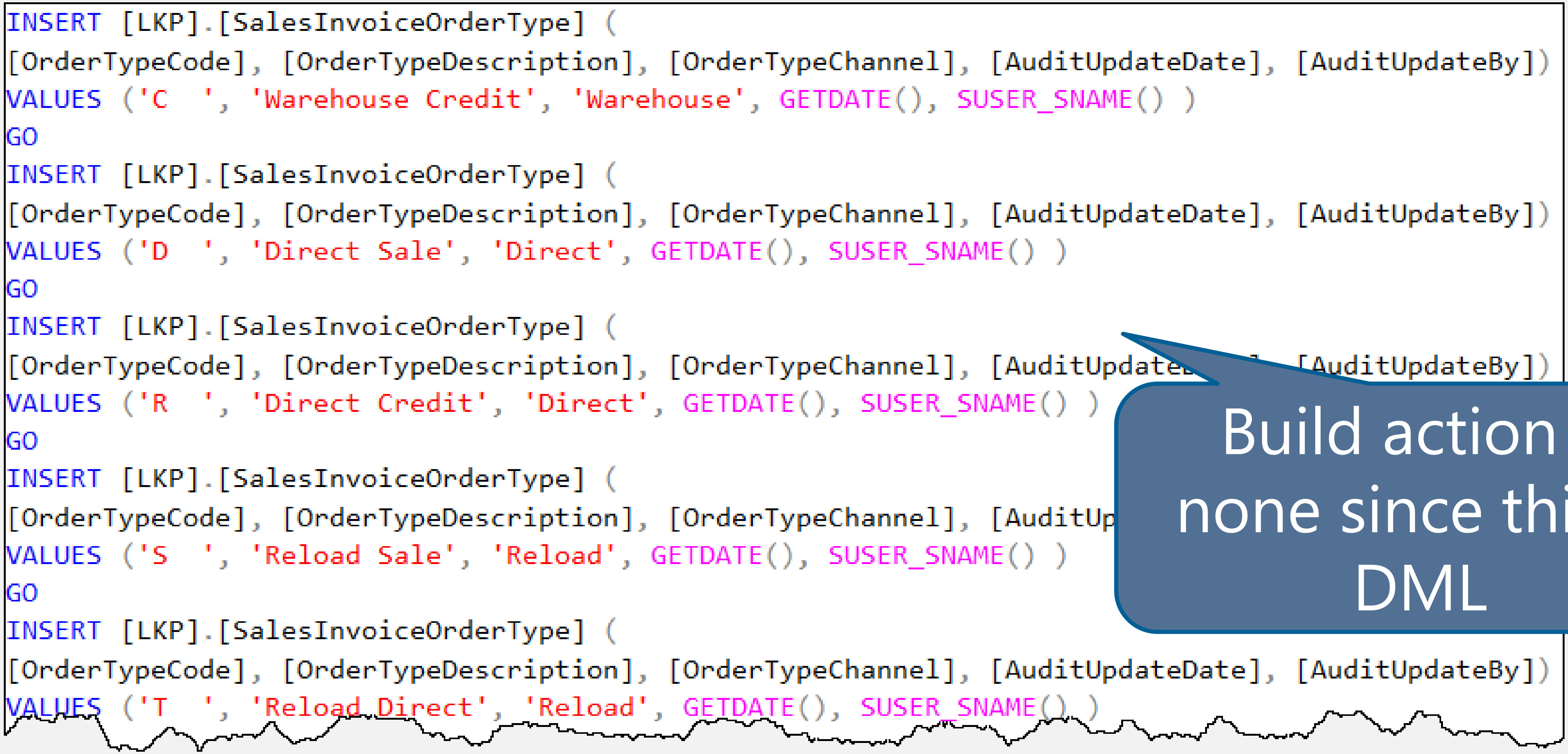
```
         ,'1901-01-01'
         ,'2999-12-31'
         ,1
         );
GO

--------------------------------------------------------------------------------
--Step 3. Disable ability for an explicit value to be inserted into identity column
--------------------------------------------------------------------------------
SET IDENTITY_INSERT [DW].[DimCustomer] OFF;
GO
```

Build action = none since this is DML

Identity_Insert does require elevated permissions

Manually Maintained Data

Maintain a DML script in a Lookup (LKP) table instead of hard-coding in the ETL.

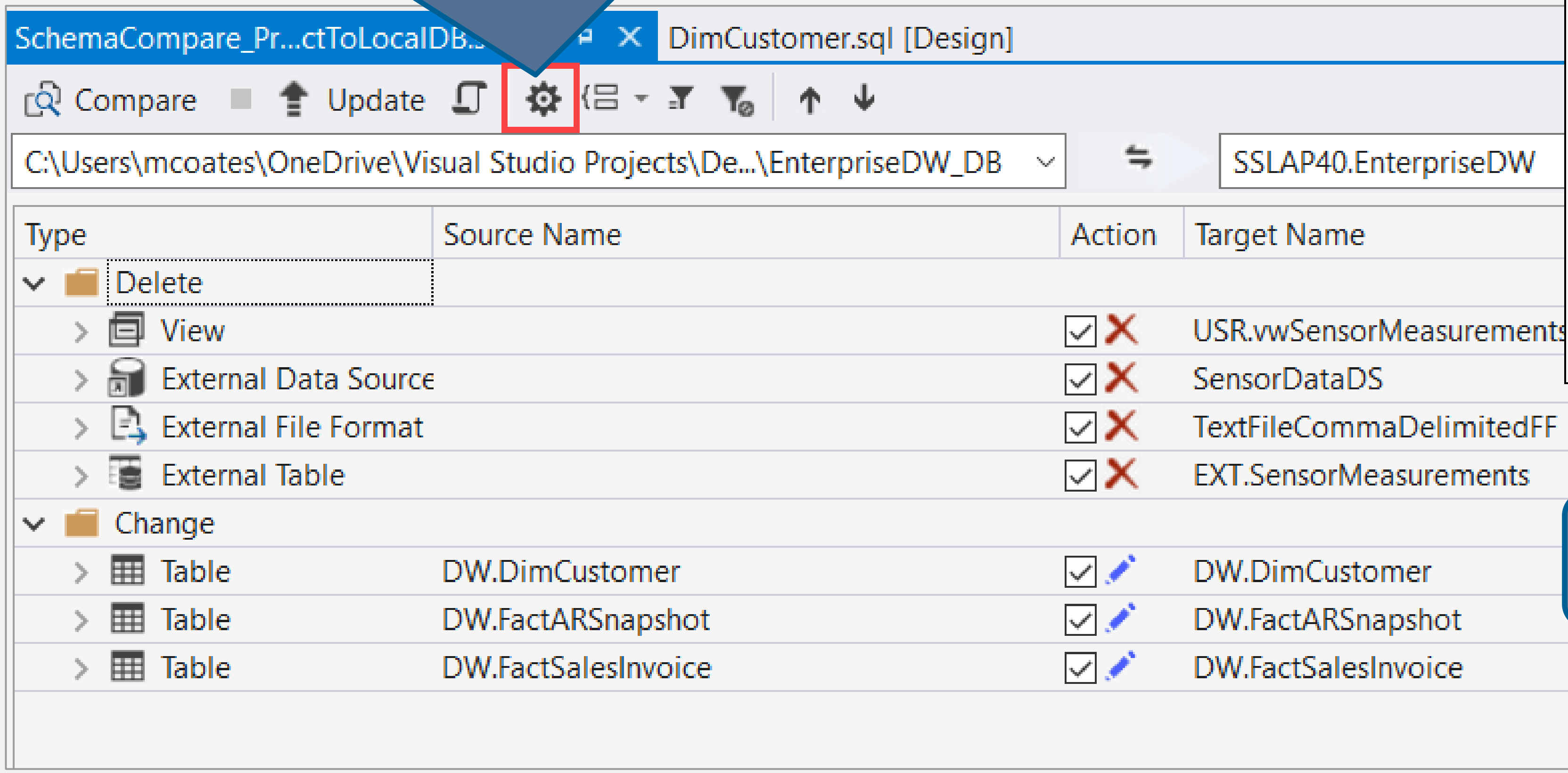


```
INSERT [LKP].[SalesInvoiceOrderType] (
[OrderTypeCode], [OrderTypeDescription], [OrderTypeChannel], [AuditUpdateDate], [AuditUpdateBy])
VALUES ('C  ', 'Warehouse Credit', 'Warehouse', GETDATE(), SUSER_SNAME() )
GO
INSERT [LKP].[SalesInvoiceOrderType] (
[OrderTypeCode], [OrderTypeDescription], [OrderTypeChannel], [AuditUpdateDate], [AuditUpdateBy])
VALUES ('D  ', 'Direct Sale', 'Direct', GETDATE(), SUSER_SNAME() )
GO
INSERT [LKP].[SalesInvoiceOrderType] (
[OrderTypeCode], [OrderTypeDescription], [OrderTypeChannel], [AuditUpdate       [AuditUpdateBy])
VALUES ('R  ', 'Direct Credit', 'Direct', GETDATE(), SUSER_SNAME() )
GO
INSERT [LKP].[SalesInvoiceOrderType] (
[OrderTypeCode], [OrderTypeDescription], [OrderTypeChannel], [AuditUp
VALUES ('S  ', 'Reload Sale', 'Reload', GETDATE(), SUSER_SNAME() )
GO
INSERT [LKP].[SalesInvoiceOrderType] (
[OrderTypeCode], [OrderTypeDescription], [OrderTypeChannel], [AuditUpdateDate], [AuditUpdateBy])
VALUES ('T  ', 'Reload Direct', 'Reload', GETDATE(), SUSER_SNAME() )
```

Build action = none since this is DML

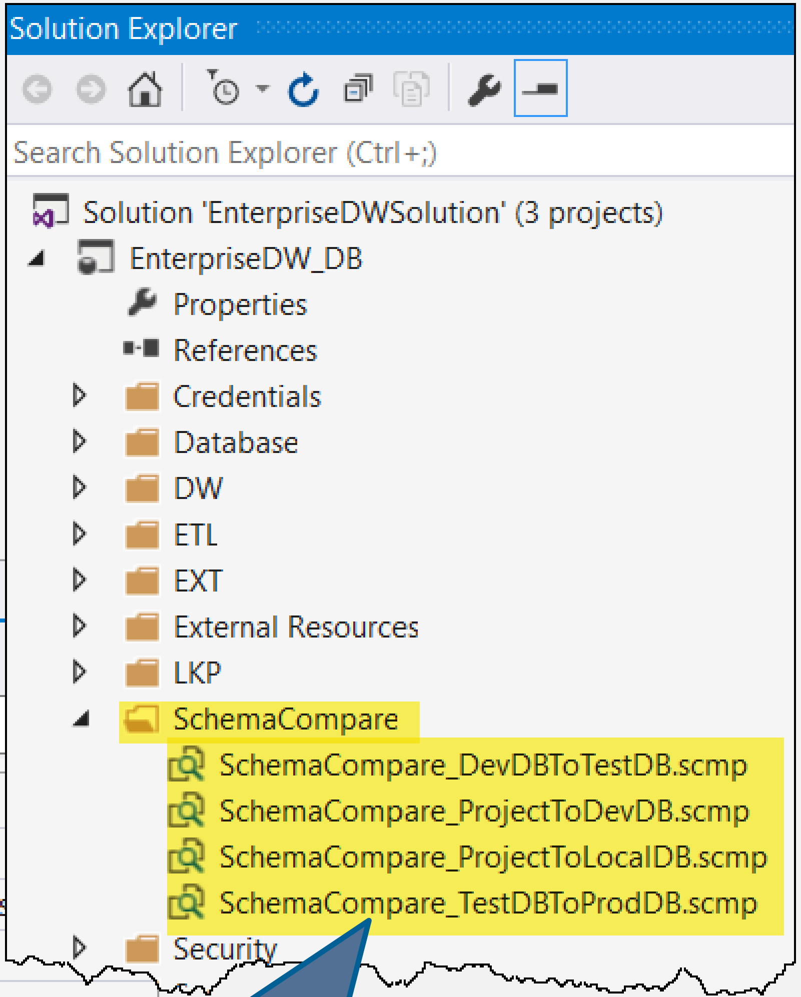
Schema Compare

Settings to exclude permissions, users, etc + options to ignore



C:\Users\mcoates\OneDrive\Visual Studio Projects\De...\EnterpriseDW_DB ⇆ SSLAP40.EnterpriseDW

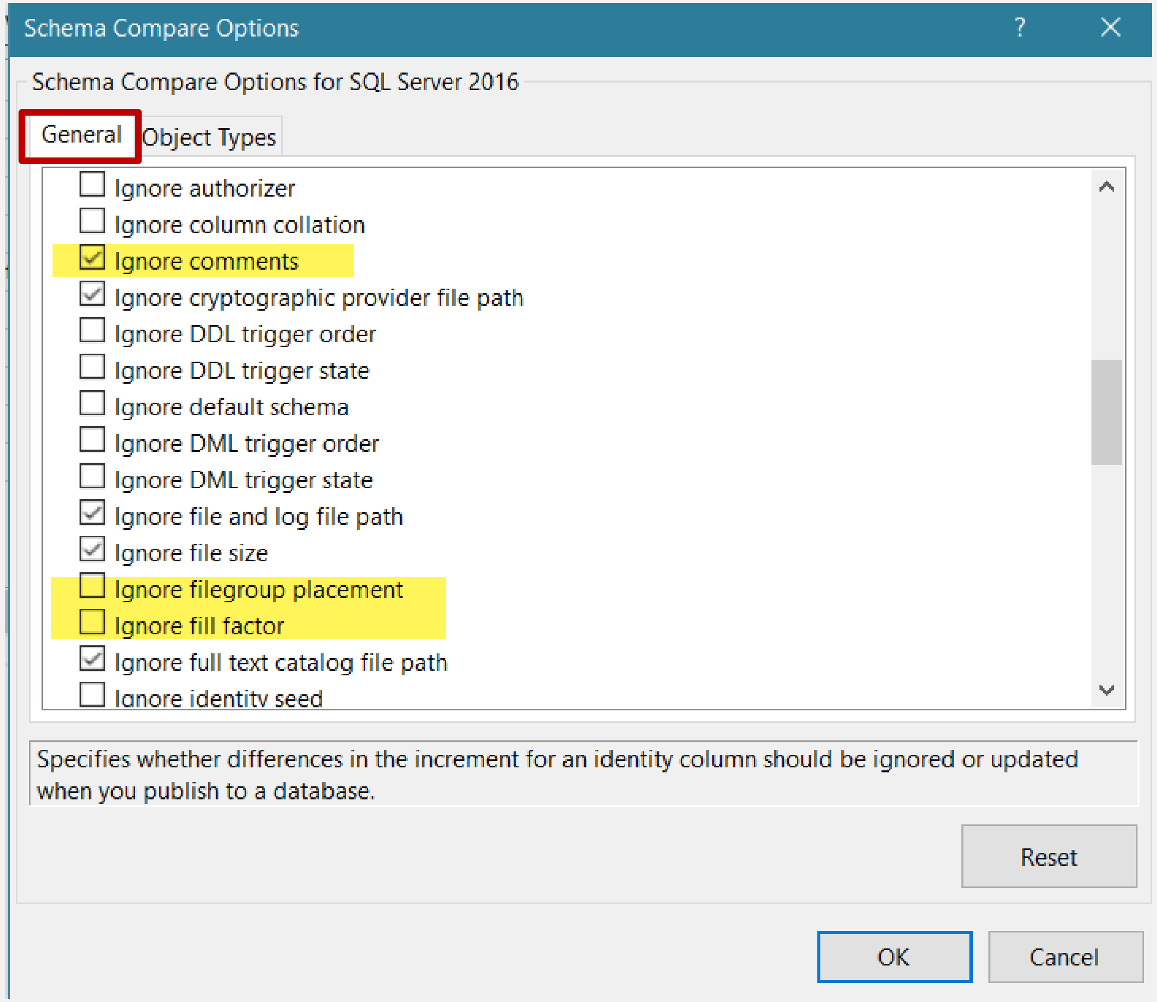
Type	Source Name	Action	Target Name
Delete			
View		☑ ✗	USR.vwSensorMeasurements
External Data Source		☑ ✗	SensorDataDS
External File Format		☑ ✗	TextFileCommaDelimitedFF
External Table		☑ ✗	EXT.SensorMeasurements
Change			
Table	DW.DimCustomer	☑ ✎	DW.DimCustomer
Table	DW.FactARSnapshot	☑ ✎	DW.FactARSnapshot
Table	DW.FactSalesInvoice	☑ ✎	DW.FactSalesInvoice



- Solution 'EnterpriseDWSolution' (3 projects)
  - EnterpriseDW_DB
    - Properties
    - References
    - Credentials
    - Database
    - DW
    - ETL
    - EXT
    - External Resources
    - LKP
    - SchemaCompare
      - SchemaCompare_DevDBToTestDB.scmp
      - SchemaCompare_ProjectToDevDB.scmp
      - SchemaCompare_ProjectToLocalDB.scmp
      - SchemaCompare_TestDBToProdDB.scmp
    - Security

Saved settings

Schema Compare Options



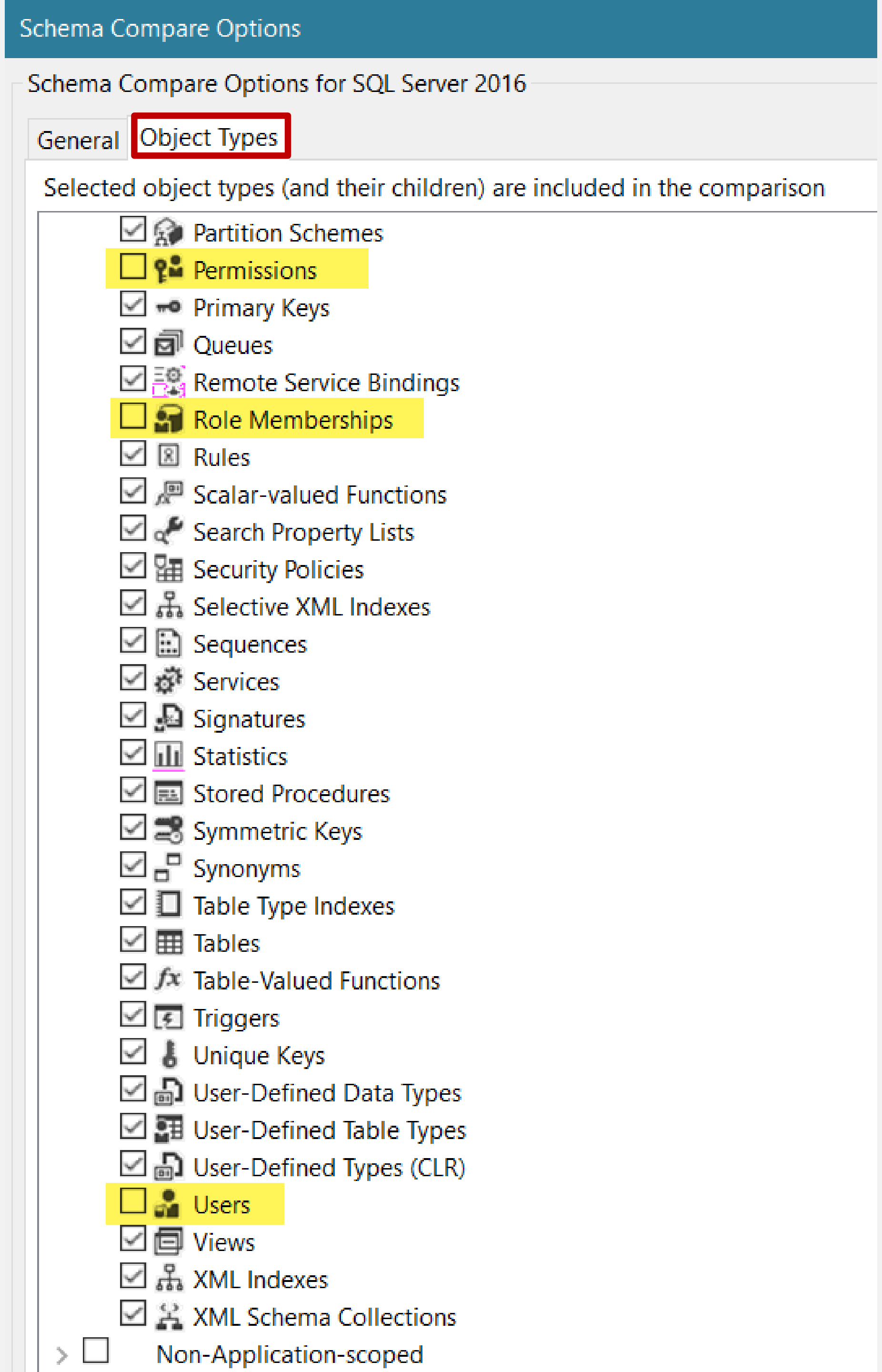
Schema Compare Options

Schema Compare Options for SQL Server 2016

General | Object Types

☐ Ignore authorizer
☐ Ignore column collation
☑ Ignore comments
☑ Ignore cryptographic provider file path
☐ Ignore DDL trigger order
☐ Ignore DDL trigger state
☐ Ignore default schema
☐ Ignore DML trigger order
☐ Ignore DML trigger state
☑ Ignore file and log file path
☑ Ignore file size
☐ Ignore filegroup placement
☐ Ignore fill factor
☑ Ignore full text catalog file path
☐ Ignore identity seed

Specifies whether differences in the increment for an identity column should be ignored or updated when you publish to a database.



Schema Compare Options

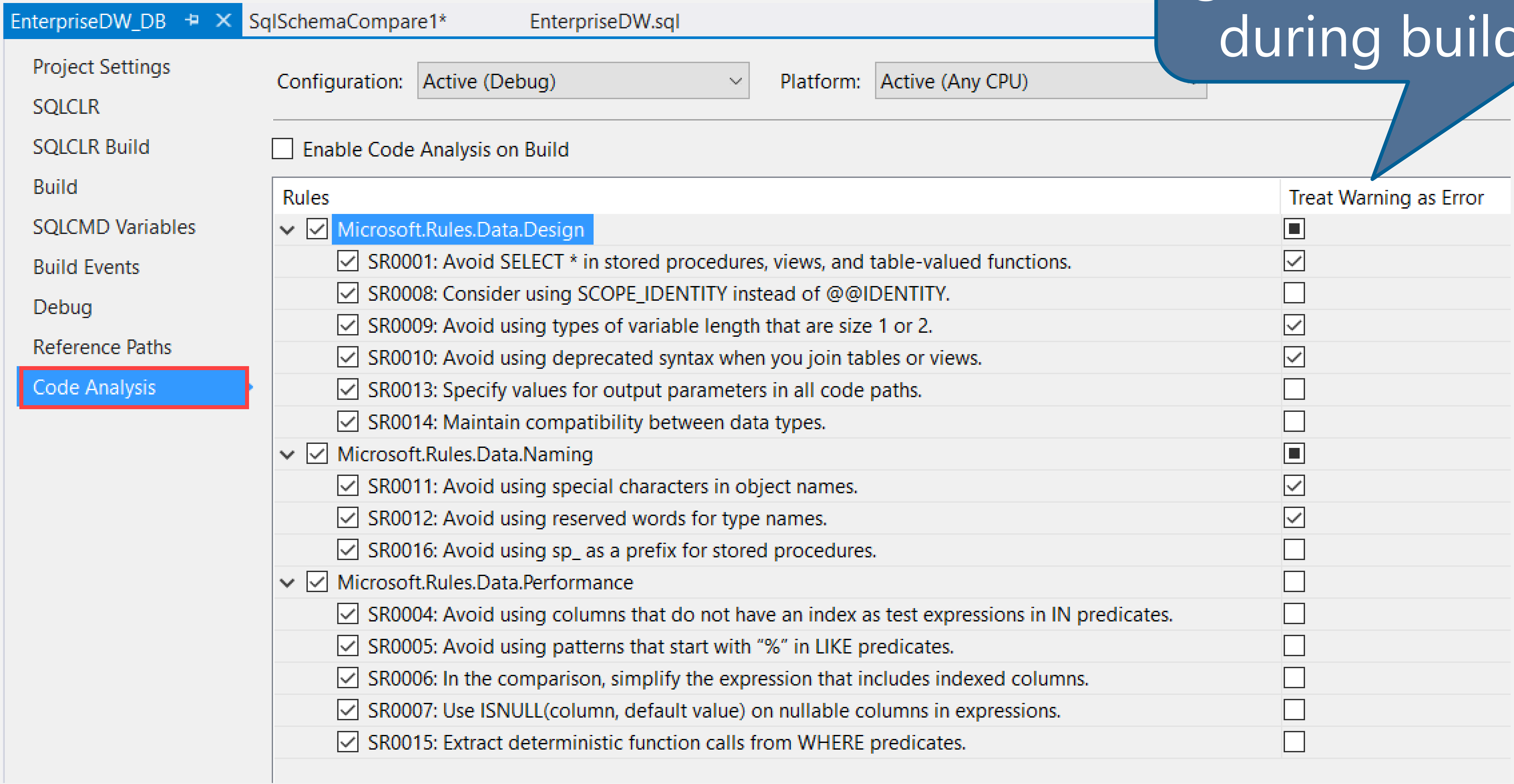
Schema Compare Options for SQL Server 2016

General | Object Types

Selected object types (and their children) are included in the comparison

☑ Partition Schemes
☐ Permissions
☑ Primary Keys
☑ Queues
☑ Remote Service Bindings
☐ Role Memberships
☑ Rules
☑ Scalar-valued Functions
☑ Search Property Lists
☑ Security Policies
☑ Selective XML Indexes
☑ Sequences
☑ Services
☑ Signatures
☑ Statistics
☑ Stored Procedures
☑ Symmetric Keys
☑ Synonyms
☑ Table Type Indexes
☑ Tables
☑ Table-Valued Functions
☑ Triggers
☑ Unique Keys
☑ User-Defined Data Types
☑ User-Defined Table Types
☑ User-Defined Types (CLR)
☐ Users
☑ Views
☑ XML Indexes
☑ XML Schema Collections
› ☐ Non-Application-scoped

Project Properties



Configuration: Active (Debug)   Platform: Active (Any CPU)

□ Enable Code Analysis on Build

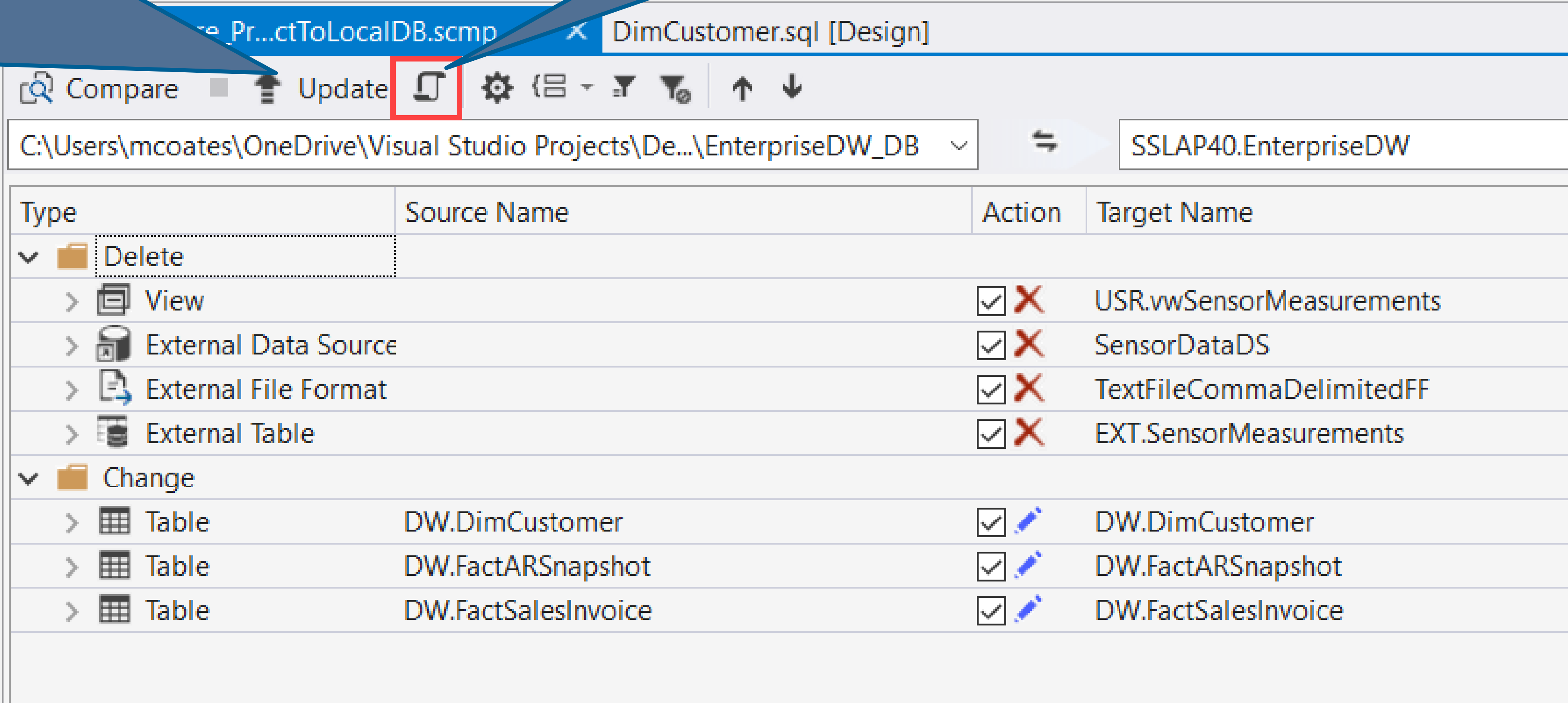
Rules	Treat Warning as Error
☑ Microsoft.Rules.Data.Design	■
☑ SR0001: Avoid SELECT * in stored procedures, views, and table-valued functions.	☑
☑ SR0008: Consider using SCOPE_IDENTITY instead of @@IDENTITY.	□
☑ SR0009: Avoid using types of variable length that are size 1 or 2.	☑
☑ SR0010: Avoid using deprecated syntax when you join tables or views.	☑
☑ SR0013: Specify values for output parameters in all code paths.	□
☑ SR0014: Maintain compatibility between data types.	□
☑ Microsoft.Rules.Data.Naming	■
☑ SR0011: Avoid using special characters in object names.	☑
☑ SR0012: Avoid using reserved words for type names.	☑
☑ SR0016: Avoid using sp_ as a prefix for stored procedures.	□
☑ Microsoft.Rules.Data.Performance	□
☑ SR0004: Avoid using columns that do not have an index as test expressions in IN predicates.	□
☑ SR0005: Avoid using patterns that start with “%” in LIKE predicates.	□
☑ SR0006: In the comparison, simplify the expression that includes indexed columns.	□
☑ SR0007: Use ISNULL(column, default value) on nullable columns in expressions.	□
☑ SR0015: Extract deterministic function calls from WHERE predicates.	□

Option to generate error during build

Schema Compare

Usually <u>don't</u> want to let the target update directly

Generates a script to use for deployment



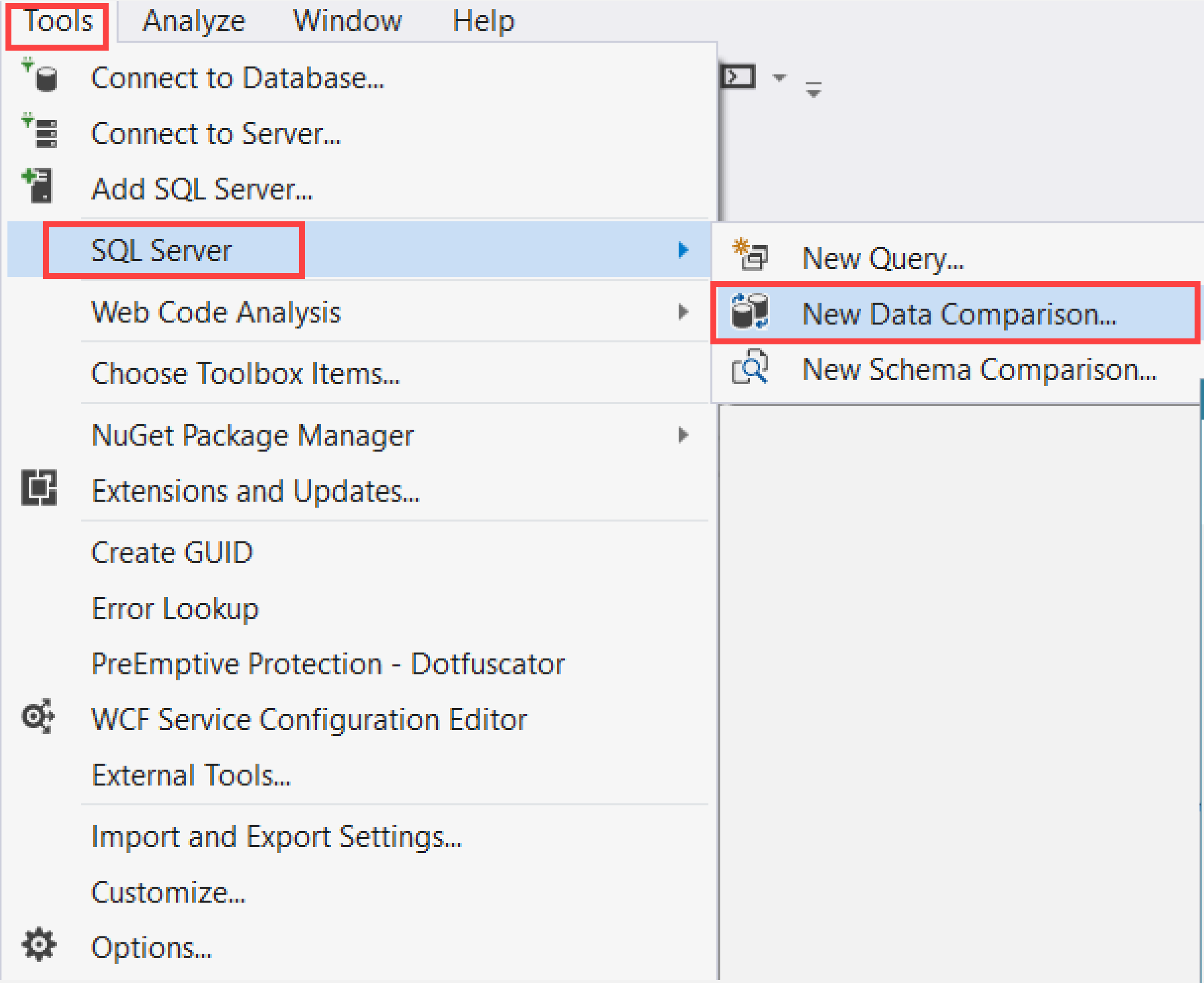
C:\Users\mcoates\OneDrive\Visual Studio Projects\De...\EnterpriseDW_DB

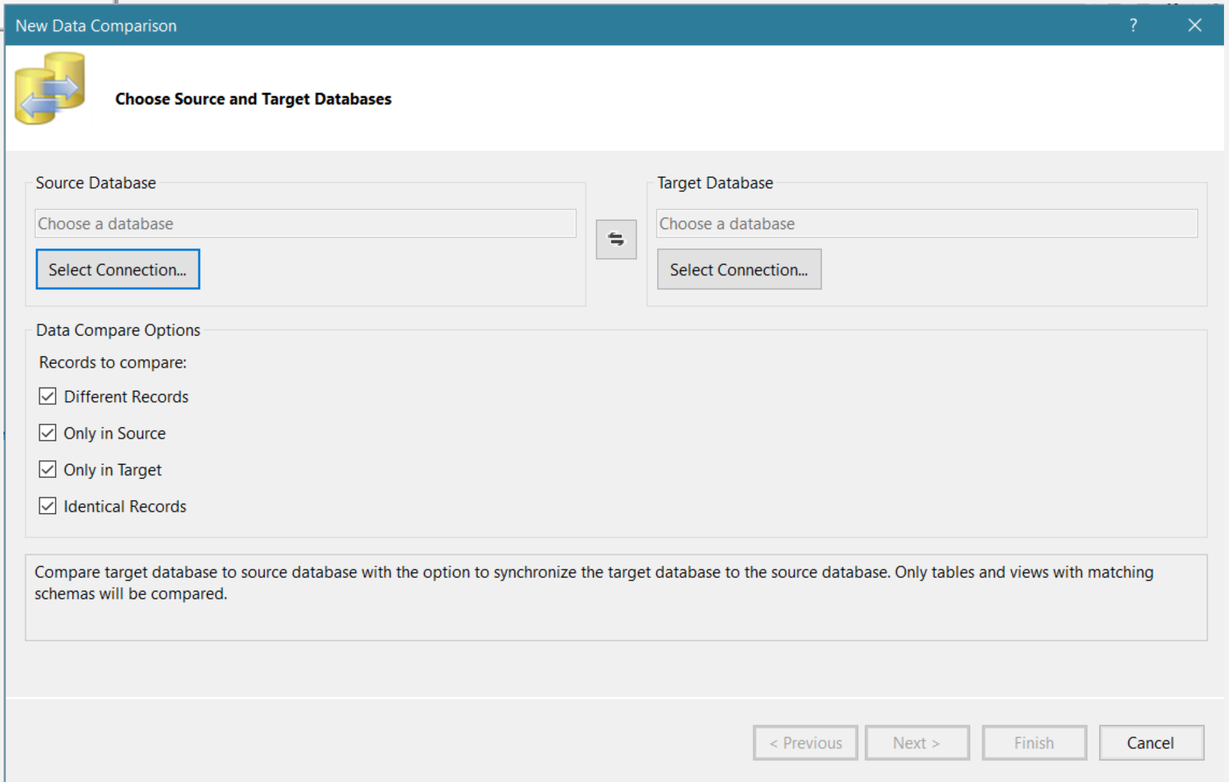
SSLAP40.EnterpriseDW

Type	Source Name	Action	Target Name
Delete			
View		☑ ✗	USR.vwSensorMeasurements
External Data Source		☑ ✗	SensorDataDS
External File Format		☑ ✗	TextFileCommaDelimitedFF
External Table		☑ ✗	EXT.SensorMeasurements
Change			
Table	DW.DimCustomer	☑ ✎	DW.DimCustomer
Table	DW.FactARSnapshot	☑ ✎	DW.FactARSnapshot
Table	DW.FactSalesInvoice	☑ ✎	DW.FactSalesInvoice

Data Compare

Basic functionality to compare data between two tables -- schema must match.





Choose Source and Target Databases

Source Database

Choose a database

Target Database

Choose a database

Data Compare Options

Records to compare:

☑ Different Records

☑ Only in Source

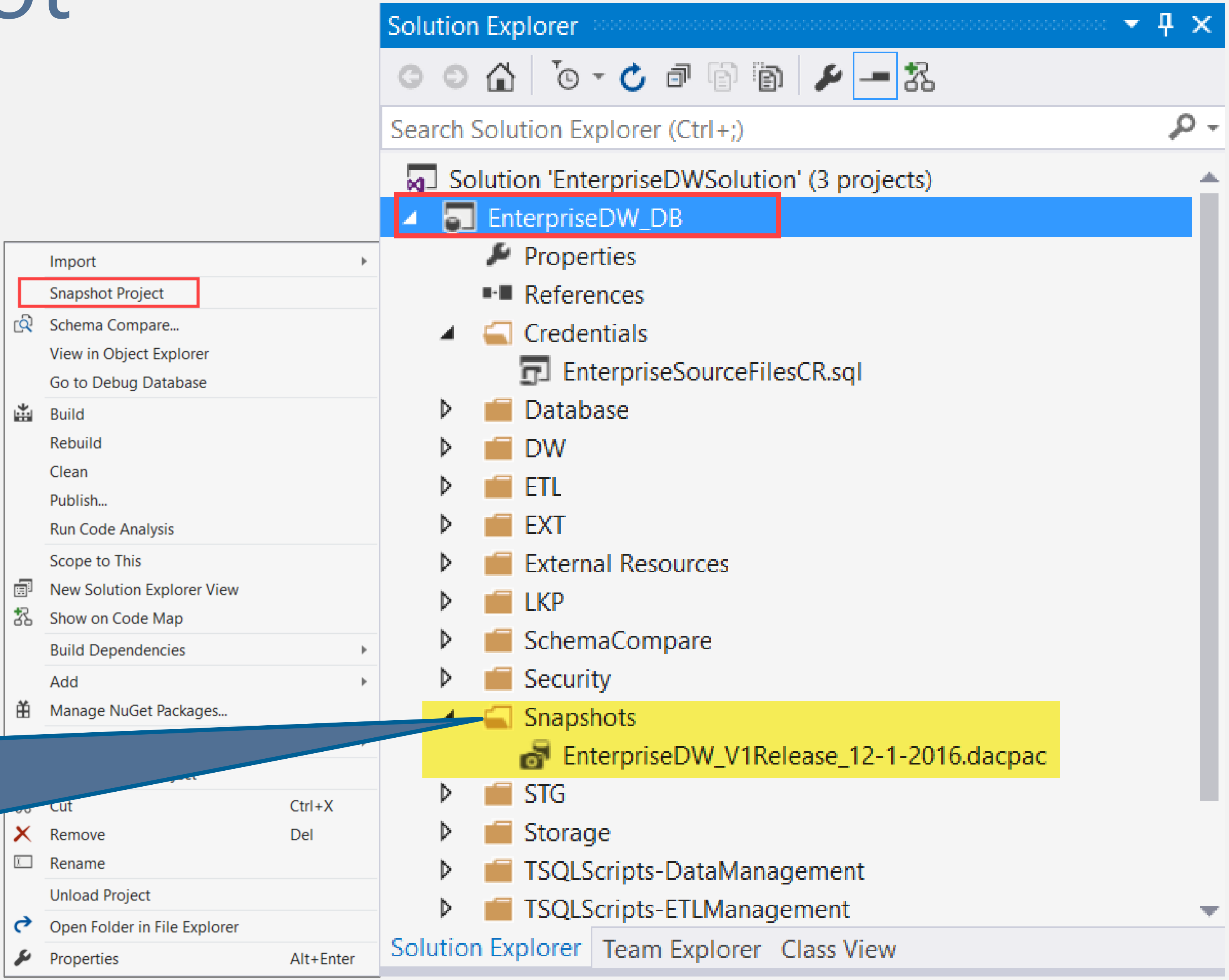
☑ Only in Target

☑ Identical Records

Compare target database to source database with the option to synchronize the target database to the source database. Only tables and views with matching schemas will be compared.

Project Snapshot

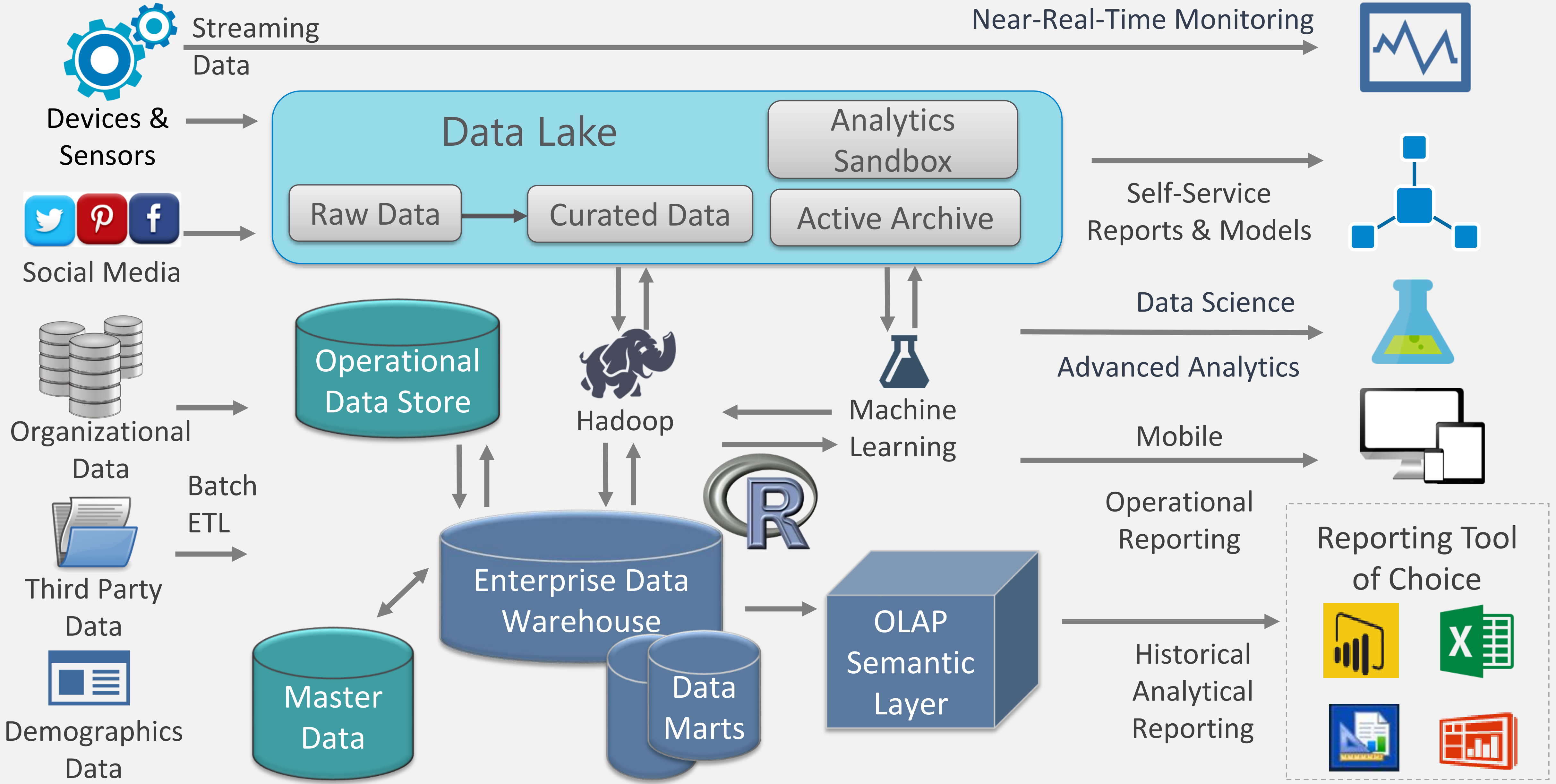
Snapshot of the database schema at a point in time (ex: major release points).



Store the .dacpac file in the project if desired

Planning Future Growth of the Data Warehouse

Modern /DW/BI/Analytics Systems



Growing your DW/BI/Analytics Environment

Cloud & Hybrid Platforms

Modern DW Multi-Platform Architecture

Advanced Analytics

Real-Time Reporting

Self-Service BI

Agile, Nimble Solutions

Achieving Extensibility in a DW

Design with change in mind. Ex: Create a lookup table with code/descriptions, or implement in a view, rather than hard-coding in ETL.

Plan for a **hybrid** environment with **multiple architectures**.

Introduce **conformed dimensions first** whenever possible.

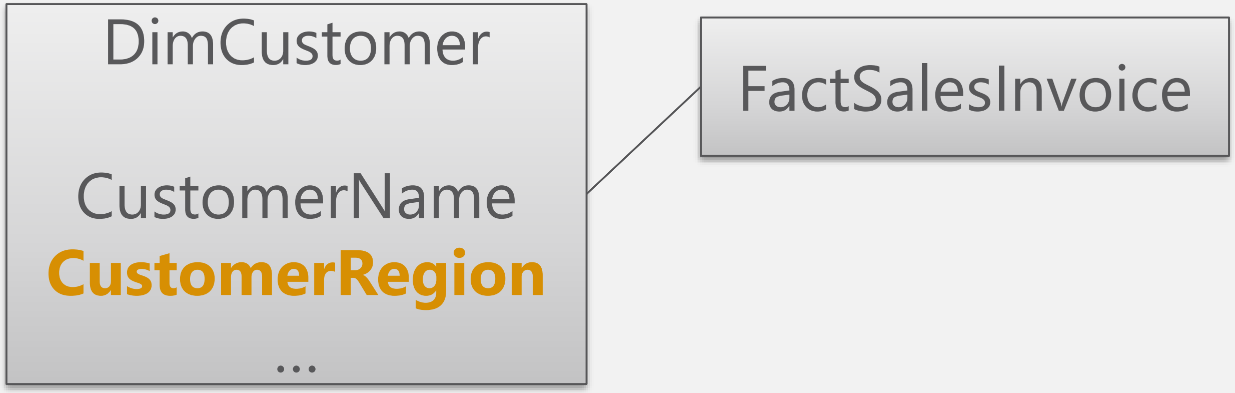
Try to **avoid isolated "stovepipe" implementations** unless the isolation is absolutely intended.

Conduct **active prototyping sessions** with business users to flush out requirements. A data modeling tool like Power BI works well for this.

Achieving Extensibility in a DW

Be prepared to do some **refactoring** along the way. Ex: converting an attribute to be a conformed dimension.

First implementation:



DimCustomer

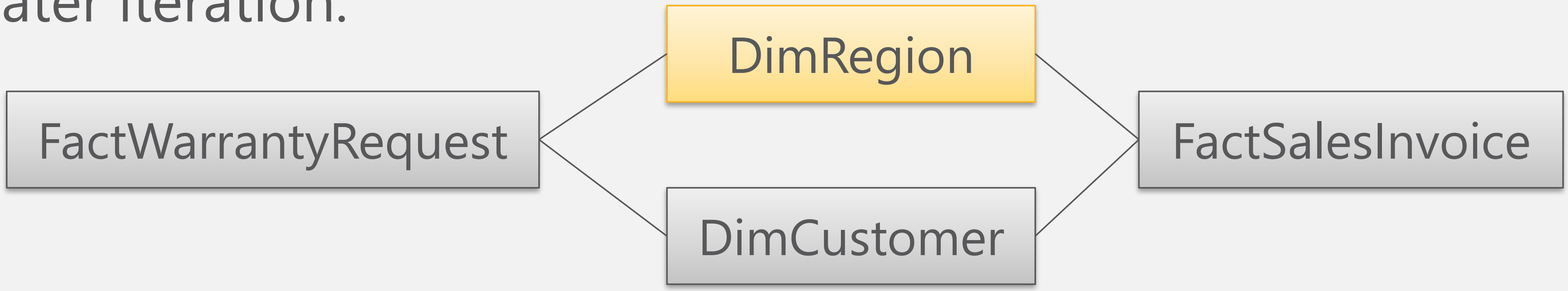
CustomerName

CustomerRegion

…

FactSalesInvoice

Updated in a later iteration:



FactWarrantyRequest

DimRegion

DimCustomer

FactSalesInvoice

Achieving Extensibility in a DW

Introducing new measures:

- Can be a new column in a fact table as long as it’s the same grain & the same business process

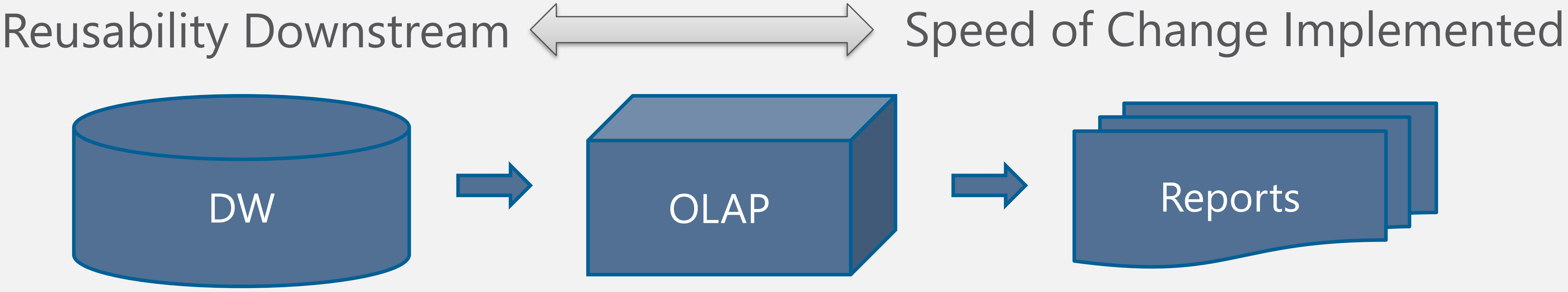
Introducing new attributes:

- Can be a new column in a dimension, or
- Can be via a new foreign key in a fact table as long as it doesn’t affect the grain

Agility for the things that usually require the most time investment:

- Data modeling
- ETL processes
- Data quality

Achieving Extensibility in a DW



Consider using an ***OLAP cube or in-memory model*** (like Analysis Services) for:

- Summary data (as opposed to summary tables in your DW)
- Year-to-Date type of calculations
- Year-over-Year type of calculations
- Aggregate level calculations (as opposed to row-by-row calculations)

Modern DW: Important Concepts to Know

Polygot Persistence

Using the most effective data storage technology to handle different data storage needs

Lambda Architecture

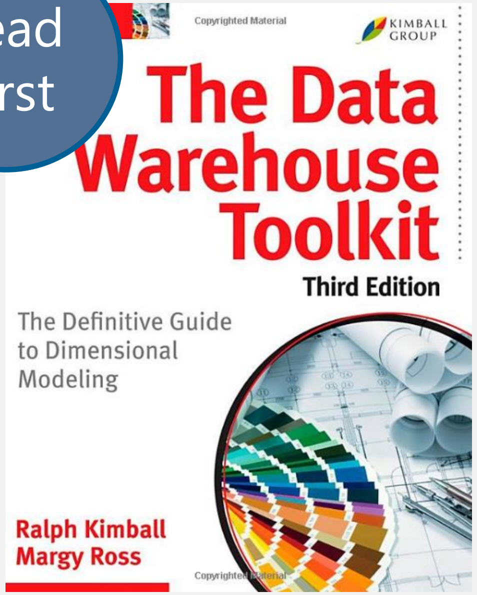
Data processing architecture which supports large amounts of data via a speed layer, batch layer, and serving layer

Schema on Read

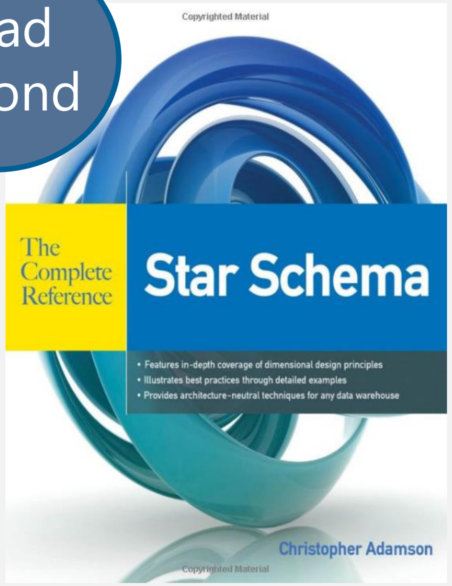
Data structure is applied at query time rather than when the data is initially stored

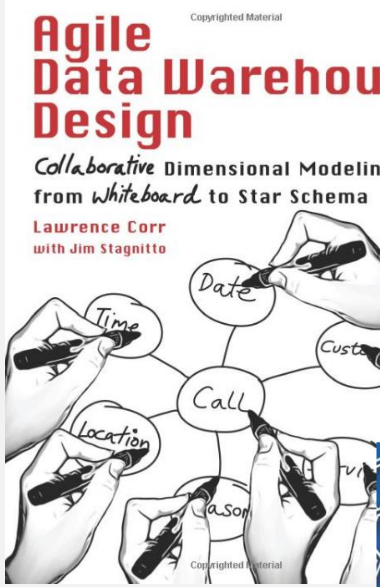
Recommended Resources

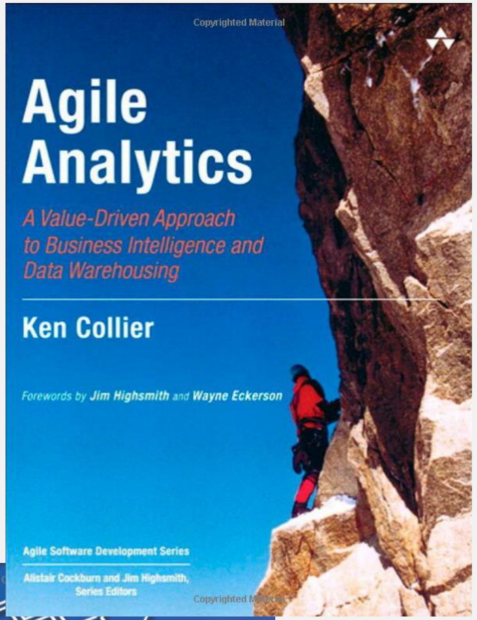
Read First

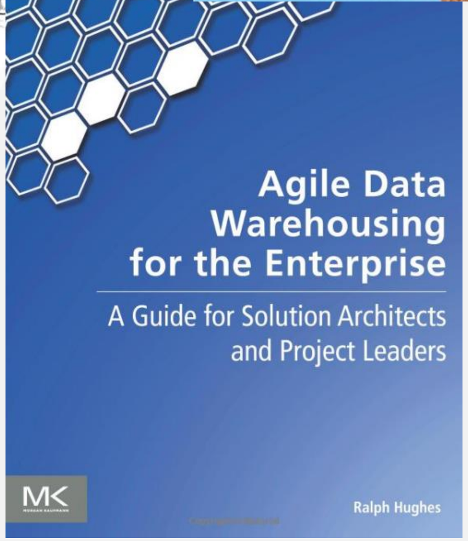

Copyrighted Material
KIMBALL GROUP
The Data Warehouse Toolkit
Third Edition
The Definitive Guide to Dimensional Modeling
Ralph Kimball
Margy Ross

Read Second


Copyrighted Material
The Complete Reference
Star Schema
• Features in-depth coverage of dimensional design principles
• Illustrates best practices through detailed examples
• Provides architecture-neutral techniques for any data warehouse
Christopher Adamson


Copyrighted Material
Agile Data Warehouse Design
Collaborative Dimensional Modeling, from Whiteboard to Star Schema
Lawrence Corr
with Jim Stagnitto
Date
Time
Call
Location


Copyrighted Material
Agile Analytics
A Value-Driven Approach to Business Intelligence and Data Warehousing
Ken Collier
Forewords by Jim Highsmith and Wayne Eckerson
Agile Software Development Series
Alistair Cockburn and Jim Highsmith, Series Editors


Agile Data Warehousing for the Enterprise
A Guide for Solution Architects and Project Leaders
MK
Ralph Hughes

Thank You for Attending

To download a copy of this presentation:

SQLChick.com “Presentations & Downloads” page

Melissa Coates

BI Architect, SentryOne

sentryone.com

Blog: sqlchick.com

Twitter: @sqlchick