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

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

<table>
<thead>
<tr>
<th></th>
<th>OLTP</th>
<th>Data Warehouse</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal:</strong></td>
<td>✓ Operational transactions</td>
<td>✓ Informational and analytical</td>
</tr>
<tr>
<td></td>
<td>✓ “Writes”</td>
<td>✓ “Reads”</td>
</tr>
<tr>
<td><strong>Scope:</strong></td>
<td>One database system</td>
<td>Integrate data from multiple systems</td>
</tr>
<tr>
<td><strong>Example Objectives:</strong></td>
<td>✓ Process a customer order</td>
<td>✓ Identify lowest-selling products</td>
</tr>
<tr>
<td></td>
<td>✓ Generate an invoice</td>
<td>✓ Analyze margin per customer</td>
</tr>
</tbody>
</table>
DW+BI Systems Used to Be Fairly Straightforward

- Organizational Data (Sales, Inventory, etc)
- Third Party Data
- Master Data
- Operational Data Store
- Enterprise Data Warehouse
- Batch ETL
- Data Marts
- OLAP Semantic Layer
- Operational Reporting
- Historical Analytical Reporting

Reporting Tool of Choice

- [Image of Reporting Tool]
DW+BI Systems Have Grown in Complexity

Data Lake
- Raw Data
- Curated Data
- Active Archive

Operational Data Store

Enterprise Data Warehouse
- Master Data
- Data Marts

OLAP Semantic Layer

Analytics Sandbox

Near-Real-Time Monitoring

Streaming Data

Devices & Sensors
- Social Media

Organizational Data

Third Party Data

Demographics Data

Batch ETL

Hadoop

Machine Learning

Advanced Analytics

Data Science

Self-Service Reports & Models

Mobile

Operational Reporting

Historical Analytical Reporting

Reporting Tool of Choice

Operational Reporting

Self-Service Reports & Models
Data Warehouse Design Principles
3 Primary Architectural Areas

- Data Acquisition
- Enterprise Data Warehouse
- OLAP Semantic Layer
- Data Storage
- 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

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

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

Source Systems
Usage of a Star Schema

**Dimension Table**

- **DimCustomer**
- **DimDate**
- **DimProduct**
- **DimRegion**

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

**Fact Table**

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

Usage of a Star Schema
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
<table>
<thead>
<tr>
<th>Subject Area</th>
<th>Fact Table Name</th>
<th>Fact Table Type</th>
<th>Fact Purpose</th>
<th>Dim Table Name</th>
<th>Dim Table Type</th>
<th>Dim Region Type</th>
<th>Dim Warehouse Type</th>
<th>Dim Employee Type</th>
<th>Dim Date Type</th>
<th>Dim SalesInvoice Type</th>
<th>Dim Product Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sales</td>
<td>FactSalesInvoice</td>
<td>Transaction Fact</td>
<td>All sales/AR transactions (invoices + DM, CM, cash applications, and write-offs) This transaction is updated throughout the life of the obligation.</td>
<td>Dim Customer</td>
<td>Standard</td>
<td>Type 1</td>
<td>Type 1</td>
<td>Parent/Child</td>
<td>Standard</td>
<td>Standard</td>
<td>Standard</td>
</tr>
<tr>
<td>Accounts Receivable</td>
<td>FactAR</td>
<td>Periodic Fact</td>
<td>Daily summary of open AR (retained for a rolling 5 quarters)</td>
<td>Dim Customer</td>
<td>Standard</td>
<td>Type 1</td>
<td>Type 1</td>
<td>Parent/Child</td>
<td>Standard</td>
<td>Standard</td>
<td>Standard</td>
</tr>
<tr>
<td>All</td>
<td>FactCustomer</td>
<td>Accumulating Fact</td>
<td>Current summary of balances (credit limit, open order amount, AR balances, etc + current salesperson); relevant across subject areas.</td>
<td>Dim Customer</td>
<td>Standard</td>
<td>Type 1</td>
<td>Type 1</td>
<td>Parent/Child</td>
<td>Standard</td>
<td>Standard</td>
<td>Standard</td>
</tr>
<tr>
<td>Human Resources</td>
<td>FactEmployee</td>
<td>Periodic Fact</td>
<td>Annual snapshot of employee info</td>
<td>Dim Employee</td>
<td>Parent/Child</td>
<td>Type 1</td>
<td>Type 1</td>
<td>Parent/Child</td>
<td>Standard</td>
<td>Standard</td>
<td>Standard</td>
</tr>
</tbody>
</table>

**Dim Granularity:**
- One row per customer
- One row per region
- One row per warehouse
- One row per employee (recursive)
- One row per invoice, per AR transaction
- One row per product

**Fact Granularity:**
- X (role-playing)
- (via bridge table)
Store the Lowest Level Detail You Have

Drill-down behavior:

<table>
<thead>
<tr>
<th>Sales Totals</th>
<th>US Customers</th>
<th>$1,000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>European Customers</td>
<td>$ 750</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sales Totals</th>
<th>US Customers</th>
<th>East Region</th>
<th>$ 200</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>West Region</td>
<td>$ 800</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$1,000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sales Totals</th>
<th>East Region</th>
<th>Customer A</th>
<th>$ 25</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Customer B</td>
<td>$ 75</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Customer C</td>
<td>$100</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$200</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sales Detail</th>
<th>Customer C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Invoice 123</td>
</tr>
<tr>
<td></td>
<td>Invoice 456</td>
</tr>
<tr>
<td></td>
<td>Invoice 789</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Type of Column in a Dim</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>Customer Name</td>
</tr>
<tr>
<td>Non-additive numeric value</td>
<td>Customer Value to Acquisition Cost Ratio</td>
</tr>
</tbody>
</table>
| 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:

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

*Types 3, 4, 5, and 7 do exist, but are less commonly utilized.*
## Type 1 Dimension

### Original data:

<table>
<thead>
<tr>
<th>Customer SK</th>
<th>Customer NK</th>
<th>Customer Name</th>
<th>AuditRow UpdateDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABC</td>
<td>Brian Jones</td>
<td>6-4-2014</td>
</tr>
<tr>
<td>2</td>
<td>DEF</td>
<td>Sally Baker</td>
<td>10-1-2015</td>
</tr>
</tbody>
</table>

### Change to Customer Name occurs.

<table>
<thead>
<tr>
<th>Customer SK</th>
<th>Customer NK</th>
<th>Customer Name</th>
<th>AuditRow UpdateDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABC</td>
<td>Brian Jones</td>
<td>6-4-2014</td>
</tr>
<tr>
<td>2</td>
<td>DEF</td>
<td><strong>Sally Walsh</strong></td>
<td><strong>12-2-2016</strong></td>
</tr>
</tbody>
</table>
## Type 2 Dimension

### Original data:

<table>
<thead>
<tr>
<th>Customer SK</th>
<th>Customer NK</th>
<th>Customer Name</th>
<th>AuditRow Effective Date</th>
<th>AuditRow Expired Date</th>
<th>AuditRow IsCurrent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABC</td>
<td>Brian Jones</td>
<td>6-4-2014</td>
<td>12-31-9999</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>DEF</td>
<td>Sally Baker</td>
<td>10-1-2015</td>
<td>12-31-9999</td>
<td>1</td>
</tr>
</tbody>
</table>

### Change to Customer Name occurs.

### Updated data:

<table>
<thead>
<tr>
<th>Customer SK</th>
<th>Customer NK</th>
<th>Customer Name</th>
<th>AuditRow Effective Date</th>
<th>AuditRow Expired Date</th>
<th>AuditRow IsCurrent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABC</td>
<td>Brian Jones</td>
<td>6-4-2014</td>
<td>12-31-9999</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>DEF</td>
<td>Sally Baker</td>
<td>10-1-2015</td>
<td>12-2-2016</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>DEF</td>
<td>Sally Walsh</td>
<td>12-3-2016</td>
<td>12-31-9999</td>
<td>1</td>
</tr>
</tbody>
</table>
### Type 6 Dimension

**Original data:**

<table>
<thead>
<tr>
<th>Customer SK</th>
<th>Customer NK</th>
<th>Customer Name</th>
<th>Customer Name Current</th>
<th>Audit Row Effective Date</th>
<th>Audit Row Expired Date</th>
<th>Audit Row Is Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABC</td>
<td>Brian Jones</td>
<td>Brian Jones</td>
<td>6-4-2014</td>
<td>12-31-9999</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>DEF</td>
<td>Sally Baker</td>
<td>Sally Baker</td>
<td>10-1-2015</td>
<td>12-31-9999</td>
<td>1</td>
</tr>
</tbody>
</table>

**Change to Customer Name occurs.**

**Updated data:**

<table>
<thead>
<tr>
<th>Customer SK</th>
<th>Customer NK</th>
<th>Customer Name</th>
<th>Customer Name Current</th>
<th>Audit Row Effective Date</th>
<th>Audit Row Expired Date</th>
<th>Audit Row Is Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABC</td>
<td>Brian Jones</td>
<td>Brian Jones</td>
<td>6-4-2014</td>
<td>12-31-9999</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>DEF</td>
<td>Sally Baker</td>
<td>Sally Walsh</td>
<td>10-1-2015</td>
<td>12-2-2016</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>DEF</td>
<td>Sally Walsh</td>
<td>Sally Walsh</td>
<td>12-3-2016</td>
<td>12-31-9999</td>
<td>1</td>
</tr>
</tbody>
</table>
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
```
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,
  [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,
  [CustomerType] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerType] DEFAULT ('N''') NOT NULL
)
```

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

No alters beneath the create.
Dimension Design

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

Remove the Dim or Fact prefix from user access layers.
Dimension Design

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

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

Alternatively, could be converted in a view or semantic layer. Objective is to avoid reporting tools trying to sum.
### 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(10) CONSTRAINT [dfDimCustomer_CustomerNumberName] DEFAULT (N'') NOT NULL  
| , [CustomerNameNumber] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerNameNumber] DEFAULT (N'') NOT NULL  
| , [Customer_CustomerTypeCode] DEFAULT (N'') NOT NULL  
| , [DimCustomer_AccountLocationFIPSCode] DEFAULT (N'') NOT NULL  
| , [DimCustomer_CustomerTypeCode] DEFAULT (N'') NOT NULL  
| , [DimCustomer_CustomerNameCurrent] DEFAULT (N'') NOT NULL  
| )

**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**

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.

```sql
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,
    [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_CustomerNameName] DEFAULT (N'') NOT NULL,
    [CustomerNumberName] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNameNumber] DEFAULT (N'') NOT NULL,
    [CustomerFIPSCode] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomAccountLocationFIPSCode] 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
)
```
Dimension Design

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

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

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

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

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

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:

<table>
<thead>
<tr>
<th>Type of Column in a Fact</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measures</td>
<td>Sales Amount</td>
</tr>
<tr>
<td>Foreign keys to dimension table</td>
<td>3392 (meaningless integer surrogate key)</td>
</tr>
<tr>
<td>Degenerate dimension</td>
<td>Order Number</td>
</tr>
</tbody>
</table>
## Types of Fact Tables

Most common types of facts:

<table>
<thead>
<tr>
<th>Type of Fact Table</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction Fact</td>
<td>An event at a point in time</td>
<td>FactSalesInvoice</td>
</tr>
<tr>
<td>Periodic Snapshot Fact</td>
<td>Summary at a point in time</td>
<td>FactARBalanceDaily</td>
</tr>
<tr>
<td>Accumulating Snapshot Fact</td>
<td>Summary across the lifetime of an event</td>
<td>FactStudentApplication</td>
</tr>
<tr>
<td>Timespan Tracking Fact</td>
<td>Effective-dated rows</td>
<td>FactCapitalAssetBalance</td>
</tr>
</tbody>
</table>

Other facts:

<table>
<thead>
<tr>
<th>Type of Fact Table</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factless Fact Table</td>
<td>Recording when an event did not occur</td>
<td>FactPromotionNoSales</td>
</tr>
<tr>
<td>Aggregate Facts</td>
<td>Rollups, usually to improve reporting speed</td>
<td>FactSalesInvoiceSummary</td>
</tr>
</tbody>
</table>
Fact Design

- One fact table per distinct business process.

```
CREATE TABLE DW.FactSalesInvoice (SalesInvoiceSK INT,
CONSTRAINT dfFactSalesInvoice_AROblication
CONSTRAINT fkFactSalesInvoice_DimSalesInvoice REFERENCES DW.DimSalesInvoice(SalesInvoiceSK),
CustomerSK INT,
CONSTRAINT dfFactSalesInvoice_SK DEFAULT ((-1)) NOT NULL,
CONSTRAINT fkFactSalesInvoice_DimCustomer REFERENCES DW.DimCustomer(CustomerSK),
RegionSK SMALLINT,
CONSTRAINT dfFactSalesInvoice_DimRegion REFERENCES DW.DimRegion(RegionSK),
DateSK_AROpenedDate INT,
CONSTRAINT dfFactSalesInvoice_DateSK_ARClosedDate DEFAULT (29991231) NOT NULL,
CONSTRAINT fkFactSalesInvoice_DimDate_ARClosedDate 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,
CONSTRAINT fkFactSalesInvoice_DimDate_ARDiscountDate REFERENCES DW.DimDate(DateSK))
```
The combination of SKs might dictate the grain of the fact table, but it may not.
Some data modelers prefer the unknown member row to have its key assigned randomly.

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.

It’s also fine for a date SK to be an actual date datatype instead of an integer.
Foreign key constraints mitigate referential integrity issues.

Having a PK in a fact is personal preference. Usually you don’t want a clustered index on it though.
Measures are sparse, therefore nullable. 0s are not stored except in a factless fact table.
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

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.
The clustered index is usually on a date.

Compression set on the clustered index rather than the table.
Fact Design

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

Rowstore:

<table>
<thead>
<tr>
<th>CounterName</th>
<th>Disk</th>
<th>DateMeasurementTaken</th>
<th>TimeMeasurementTaken</th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Disk sec/Read</td>
<td>G:\</td>
<td>1/30/2017</td>
<td>4:48:41 PM</td>
<td>0.01818</td>
</tr>
<tr>
<td>Avg. Disk sec/Read</td>
<td>L:\</td>
<td>1/30/2017</td>
<td>4:48:41 PM</td>
<td>0.00385</td>
</tr>
<tr>
<td>Avg. Disk sec/Read</td>
<td>T:\</td>
<td>1/30/2017</td>
<td>4:48:41 PM</td>
<td>0.00780</td>
</tr>
<tr>
<td>Avg. Disk Bytes/Read</td>
<td>G:\</td>
<td>1/30/2017</td>
<td>4:48:41 PM</td>
<td>53120.73782</td>
</tr>
<tr>
<td>Avg. Disk Bytes/Read</td>
<td>L:\</td>
<td>1/30/2017</td>
<td>4:48:41 PM</td>
<td>42362.51095</td>
</tr>
<tr>
<td>Avg. Disk Bytes/Read</td>
<td>T:\</td>
<td>1/30/2017</td>
<td>4:48:41 PM</td>
<td>47951.40657</td>
</tr>
</tbody>
</table>

Columnstore:

Reduced storage for low cardinality columns
CCI most suitable for:
- Tables over 1 million rows
- 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)

```
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:

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

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

- **Source System**
  - STG Table: Data loaded in original format
    - Ex: STG.CustDet
  - STG Table: Data loaded in original format
    - Ex: STG.wvDimCustomer

- **Database**
  - STG View: View to populate Dim or Fact
    - Ex: STG.wvDimCustomer
  - DW Table: Data loaded in dimensional format
    - Ex: DW.DimCustomer
  - EXT Table: Metadata which references external data source
    - Ex: EXT.DeviceLogs

- **Analysis Services**
  - DW View: Feeds semantic layer or in-memory model
    - Ex: DW.wvDimCustomer
  - USR View: Supports power users who want to access relational data directly. Implements row-level security when applicable.
    - Ex: USR.wvDimCustomer

- **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

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.

Nearly all objects should be set to Build.
Database Design

```sql
USE [master]
GO

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

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

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

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

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

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

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

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.

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

Build action = none since this is DML
Schema Compare

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

Saved settings
Schema Compare Options

General

Object Types

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

- Partition Schemes
- Permissions
- Primary Keys
- Roles
- 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
Option to generate error during build
Schema Compare

Generates a script to use for deployment

Usually don’t want to let the target update directly
Data Compare

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

- Streaming Data
  - Devices & Sensors
  - Social Media
  - Organizational Data
  - Third Party Data
  - Demographics Data

- Data Lake
  - Raw Data
  - Curated Data
  - Active Archive

- Operational Data Store
  - Hadoop
  - Batch ETL

- Enterprise Data Warehouse
  - Master Data
  - Data Marts

- OLAP Semantic Layer
  - Analytics Sandbox
  - Active Archive

- Near-Real-Time Monitoring
  - Self-Service Reports & Models
  - Data Science
  - Advanced Analytics
  - Mobile

- Machine Learning

- Reporting Tool of Choice
  - Operational Reporting
  - Historical Analytical Reporting
  - Mobile Self-Service Reports & Models
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:

Updated in a later iteration:
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

Reusability Downstream  Speed of Change Implemented

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

**Recommended Resources**

- **Read First**
  - The Data Warehouse Toolkit
  - The Complete Reference

- **Read Second**
  - Star Schema
  - Agile Data Warehouse Design
  - Agile Analytics
  - Agile Data Warehousing for the Enterprise
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