# Kimball Data Warehouse Toolkit

## Ch 16 – Insurance

* We **bring together concepts from nearly all previous chapters to build a DW/BI system for a property and casualty insurance company** in this final case study
* This material depends heavily on ideas from the earlier chapters
* As standard procedure, this chapter launches with background information for a business case.
* While requirements unfold, we’ll draft the EDW bus matrix, much like we would in a real-life requirements analysis effort
* We’ll then design a series of dimensional models by overlaying the core techniques learned thus far
* **Concepts:**
* Requirements-driven approach to dimensional design
* Value chain implications, along with an example bus matrix snippet for an insurance company
* Complementary transaction, periodic snapshot, + accumulating snapshot schemas
* Dimension role playing
* Handling of SCD attributes
* Mini-dimensions for dealing with large, rapidly changing dimension attributes
* Multivalued dimension attributes
* Degenerate dimensions for operational control numbers
* Audit dimensions to track data lineage
* Heterogeneous supertypes + subtypes to handle products with varied attributes + facts
* Junk dimensions for miscellaneous indicators
* Conformed dimensions and facts
* Consolidated fact tables combining metrics from separate business processes
* Fact-less fact tables
* Common mistakes to avoid when designing dimensional models

### Insurance Case Study

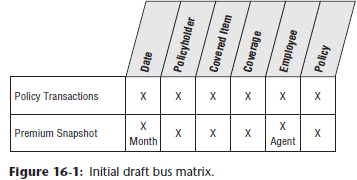
* Imagine working for a large property + casualty insurer that offers auto, homeowner, + personal property insurance
* You conduct extensive interviews w/ business reps + senior management from the claims, field operations, underwriting, finance, + marketing departments
* Based on these interviews, you learn the industry is in a state of flux
* Non-traditional players are leveraging alternative channels, + meanwhile, the industry is consolidating due to globalization, deregulation, + demutualization challenges
* Markets are changing, along with customer needs
* Numerous interviewees tell us **information is becoming an even more important strategic asset**
* Regardless of the functional area, **there is a strong desire to use information more effectively to identify opportunities more quickly + respond most appropriately**
* The good news is that **internal systems + processes already capture the bulk of the data required**
* Most insurance companies generate tons of nitty-gritty operational data
* The **bad news is the data is not *integrated***
* Over the years, political + IT boundaries have encouraged the construction of tall barriers around isolated islands of data
* There are multiple disparate sources for information about the company’s products, customers, + distribution channels
* In the legacy operational systems, the same policyholder may be identified several times separate automobile, home, + personal property applications
* Traditionally, this segmented approach to data was acceptable because the different lines of business functioned largely autonomously + there was little interest in sharing data for cross-selling + collaboration in the past
* ***Now*, within our case study, business management is attempting to better leverage this enormous amount of inconsistent and somewhat redundant data.**
* Besides the inherent issues surrounding data integration, **business users lack the ability to access data easily when needed**
* In an attempt to address this shortcoming, several groups within the case study company rallied their own resources + hired consultants to solve their individual short-term data needs
* In many cases, the **same data was extracted from the same source systems to be accessed by separate organizations without any strategic overall information delivery strategy**
* It didn’t take long to recognize the **negative ramifications associated with separate analytic data repositories because performance results presented at executive meetings differed depending on the data source**
* Management understood this **independent route was not viable as a long-term solution because of the lack of integration, large volumes of redundant data, + difficulty in interpreting + reconciling the results**
* Given the importance of information in this brave new insurance world, management was motivated to deal with the cost implications surrounding the development, support, + analytic inefficiencies of these supposed DW’s that merely proliferated operational **data islands**.
* Senior management chartered the CIO w/ the responsibility + authority to break down the historical data silos to “achieve information nirvana”
* They charged the CIO with the fiduciary responsibility to manage + leverage the organization’s information assets more effectively
* CIO developed an overall vision that wed an enterprise strategy for dealing w/ massive amounts of data w/ a response to the immediate need to become an information-rich organization
* Then, an enterprise DW/BI team was created to begin designing + implementing the vision
* Senior management has been preaching about a transformation to a more customer-centric focus, instead of the traditional product-centric approach, in an effort to gain competitive advantage
* The CIO jumped on that bandwagon as a catalyst for change
* The folks in the trenches have pledged intent to share data rather than squirreling it away for a single purpose
* There is a strong desire for everyone to have a common understanding of the state of the business
* They’re clamoring to get rid of the isolated pockets of data while ensuring they have access to detail + summary data at both the enterprise and line-of-business levels.

#### Insurance Value Chain

* The primary **value chain** of an insurance company is seemingly short and simple
* The **core processes** are to: **issue policies**, **collect premium payments**, + **process claims**
* The **organization is interested in better understanding metrics spawned by each of these events**
* Users want to analyze detailed transactions relating to the formulation of policies, as well as transactions generated by claims processing, + to measure performance over time by coverage, covered item, policyholder, + sales distribution channel characteristics
* Although some users are interested in the enterprise perspective, others want to analyze the heterogeneous nature of the insurance company’s *individual* lines of business.
* Obviously, an insurance company is engaged in many other external processes, such as the investment of premium payments or compensation of contract agents, as well as a host of internally focused activities, such as HR, finance, + purchasing
* For now, **we will focus on the core business related to policies and claims**
* The **insurance value chain begins with a variety of policy transactions**
* Based on your current understanding of the requirements + underlying data, you **opt to handle all transactions impacting a policy as a *single* business process (+ fact table)**
* **If this perspective is too simplistic** to accommodate the metrics, dimensionality, or analytics required, **handle the transaction activities as *separate* fact tables**, such as quoting, rating, + underwriting
* As discussed in Chapter 5: Procurement, there are **trade-off’s between creating separate fact tables for each natural cluster of transaction types vs. lumping the transactions into a single fact table**
* There is also a need to **better understand the premium revenue associated w/ each policy on a monthly basis**
* This will be **key input into the overall profit picture**
* The **insurance business is *very* transaction intensive**, but **the transactions themselves do NOT represent little pieces of revenue**, as is the case with retail or manufacturing sales
* You **cannot merely add up policy transactions to determine the revenue amount**
* The picture is **further complicated in insurance because customers pay in advance for services**
* This same advance-payment model applies to organizations offering magazine subscriptions or extended warranty contracts
* **Premium payments must be spread across multiple periods because the company earns the revenue *over time*** as it provides insurance coverage
* The **complex relationship between policy transactions + revenue measurements often makes it impossible to answer revenue questions by crawling through the individual transactions**
* Not only is **such crawling time-consuming,** but also the **logic required to interpret the effect of different transaction types on revenue can be horrendously complicated**
* The **natural conflict between the detailed transaction view + the snapshot perspective almost *always* requires building both kinds of fact tables in the DW**
* In this case, **the premium snapshot is not merely a summarization of the policy transactions, it is quite a separate thing that comes from a separate source.**

#### Draft Bus Matrix

* Based on interview findings, along with an understanding of the key source systems, the team begins to draft an **EDW bus matrix with the core policy-centric business processes as rows and core dimensions as columns**



* **2 rows** are defined in the matrix, one corresponding to the **policy transactions**, another for the **monthly premium snapshot**
* The **core dimensions** include **date, policyholder, employee, coverage, covered item, + policy**
* **When drafting the matrix, DON’T attempt to include ALL the dimensions**
* **Instead, try to focus on the CORE common dimensions that are reused in *more than 1* schema**

### Policy Transactions

* Turning attention to the 1st row of the matrix 🡪 focusing on the **transactions** for creating + altering a policy
* Assume the **policy represents a *set* of coverages** sold to the policyholder
* **Coverages can be considered the insurance company’s *products***
* Homeowner coverages include fire, flood, theft, + personal liability; auto coverages include comprehensive, collision damage, uninsured motorist, + personal liability
* In a property + casualty insurance company, coverages apply to a specific covered item, such as a particular house or car
* Both the coverage + covered item are carefully identified in the policy
* A particular covered item usually has several coverages listed in the policy.
* **Agents sell policies to policyholders**
* Before the policy can be created, a **pricing actuary determines the premium rate** that will be charged given the specific coverages, covered items, + qualifications of the policyholder
* An **underwriter**, who takes ultimate responsibility for doing business with the policyholder, **makes the final approval**
* The ***operational* policy transaction system captures** the following **types of transactions**:
* **Create**, **alter**, or **cancel policy** (*with reason*)
* **Create coverage** on a covered item, **alter** coverage, or **cancel** coverage (*with reason*)
* **Rate coverage** or **decline** to rate coverage (*with reason*)
* **Underwrite policy** or **decline** to underwrite policy (*with reason*)
* The **grain of the policy transaction fact table should be 1 row for each individual policy transaction**
* **Each atomic transaction should be embellished with as much context as possible to create a complete dimensional description of the transaction**
* **Dimensions** associated with the policy transaction business process include: **transaction date, effective date, policyholder, employee, coverage, covered item, policy number, + policy transaction type**

#### Dimension Role-Playing

* There are **2 dates associated with each policy transaction**
* The **policy transaction date** = when the transaction was entered into the operational system,
* The **policy transaction effective date** is when the transaction legally takes effect
* These 2 FKs in the fact table should be **uniquely named**
* The **2 *independent* dimensions** associated with these keys are **implemented using a *single* physical date table**
* **Multiple logically distinct tables are then presented to the user *through Views* with unique column names**, as described originally in Chapter 6: Order Management

#### Slowly Changing Dimensions (SCD)

* Insurance companies typically are very interested in tracking changes to dimensions over time
* Can apply **3 *basic* techniques for handling SCD attributes** to the **policyholder dimension** (Ch. 5)
* **Type 1 = simply overwrite the dimension attribute’s prior value**
* Simplest approach to dealing with attribute changes because the **attributes always represent the most current descriptors**
* Ex: Perhaps the business agrees to handle changes to the policyholder’s date of birth as a type 1 change based on the assumption that any changes to this attribute are intended as corrections
* In this manner, ***all* fact table history for this policyholder appears to have always been associated with the updated date of birth**
* **Type 2 SCD change (new dimension *row*) is the most common SCD technique when there’s a requirement for *accurate change tracking over time***
* Because a policyholder’s ZIP is key input to the insurer’s pricing + risk algorithms, users are very interested in tracking ZIP changes, so the type 2 SCD technique is used for this attribute
* In this case, when a ZIP changes, you **create a new policyholder dimension row with a new surrogate key** **+ updated** geographic **attributes**
* **Do NOT go back and revisit the fact table**
* ***Historical* fact table rows**, prior to the ZIP change, **still reflect the *old* surrogate key**
* **Going *forward***, you **use** the policyholder’s ***new* surrogate key, so new fact table rows JOIN to the post-change dimension profile**
* **Although this technique is extremely graceful + powerful, it places more burdens on ETL processing**
* Also, the **number of rows in the dimension table grows with each type 2 SCD change**
* Given there might already be > 1M rows in your policyholder dimension table, you **may opt to use a mini-dimension for tracking** ZIP **changes**
* **Type 3 SCD change (new dimension *column*)**
* Assume each policyholder is classified as belonging to a particular **segment**
* Perhaps non-residential policyholders were historically categorized as either “commercial” or “government” entities
* Going forward, business users want more detailed classifications to differentiate between large multinational, middle market, + small business commercial customers, in addition to nonprofit organizations + governmental agencies
* **For some period of time, users want the ability to analyze results by either the historical or new segment classifications**
* In this case you could use the **type 3 approach to track the change for a period of time** by **adding a column, labeled “Historical” for differentiation**, to **retain the old classifications**
* The **new classification values would populate the segment attribute that has been a permanent fixture on the policyholder dimension**
* This approach, **although not extremely common, allows you to see performance by either the current or historical segment maps**
* **Useful w/ an en masse change**, such as the customer classification realignment
* Obviously, this technique **becomes overly complex if you need to track > 1 version of the historical map, or before-and-after changes for multiple dimension attributes**

#### Mini-Dimensions for Large or Rapidly-Changing Dimensions

* As mentioned earlier, the policyholder dimension qualifies as a **“large” dimension with > 1M rows**
* It is **often important to *accurately* track content values for a subset of attributes**
* Ex: Need an accurate description of some policyholder + covered item attributes at the time the policy was created, as well as at the time of any adjustment or claim
* Chapter 5 🡪 the ***practical* way to track changing attributes in large dimensions is to split the closely-monitored, more rapidly-changing attributes into 1 or more type 4 SCD mini-dimensions *directly linked to the fact table with a separate surrogate key***
* **Use of mini-dimensions has an impact on the efficiency of attribute browsing, because users typically want to browse + constrain on these changeable attributes**
* **If *all* possible combos of the attribute values in the mini-dimension have been created, handling a mini-dimension change simply means placing a different key in the fact table row from a certain point in time forward, + nothing else needs to be changed or added to the database**
* The covered item is the house, car, or other specific insured item
* A covered item *dimension* contains 1 row per actual covered item, + is usually somewhat larger than a policyholder dimension 🡪 another good place to consider deploying a mini-dimension
* You **do NOT want to capture the variable descriptions** of the physical covered objects **as facts because most are textual and are not numeric or continuously valued**
* **Make every effort to put textual attributes into dimension tables because they are the target of textual constraints + the source of report labels**

#### Multivalued Dimension Attributes

* Discussed **multivalued dimension attributes** when we associated multiple skills with an employee in Chapter 9: HR Management, in Chapter 10: Financial Services = associated multiple customers with an account, + in Chapter 14: Healthcare = modeled a patient’s multiple diagnoses
* **Multivalued attributes** = **data that needs to represent many-to-many relationships such that dimension members are at a lower grain than related facts**
* *In these cases, a* ***single fact record should relate to multiple dimension******values***
* In this case study, you’ll look at **another multivalued modeling situation**: the **relationship between commercial customers + their industry classifications**
* Each commercial customer may be associated with 1 or more Standard Industry Classification (SIC) or North American Industry Classification System (NAICS) codes
* A large, diversified commercial customer could be represented by a dozen or more classification codes
* Much like w/ diagnosis groups, a **bridge table ties together all the industry classifi cation codes within a group**
* This industry classification **bridge table JOINs directly to either the fact table or the customer dimension as an outrigger**
* **Outrigger = dimension tables JOIN-ed to *other* dimension tables, but just 1 more layer removed from the fact table, rather than being fully normalized snowflakes**
* **Most frequently used when one standard dimension table is referenced in another dimension, such as a hire date attribute in the employee dimension table**
* **Enables you to report fact table metrics by any industry classification**
* If the commercial customer’s industry **breakdown is proportionally identified**, such as 50% agricultural services, 30% dairy products, + 20% oil + gas drilling, **a weighting factor should be included on each bridge table row**
* To handle the case in which **no valid industry code** is associated with a given customer, you **simply create a special bridge table row that represents “Unknown”**

#### Numeric Attributes as Facts or Dimensions

* Coverage dimension 🡪 Large insurance companies have dozens or even 100s of separate coverage products available to sell for a given type of covered item
* Actual appraised value of a specific covered item, like someone’s house, is a ***continuously*-valued numeric quantity that can even vary for a given item over time, so treat it as a legitimate fact**
* **In the dimension table, you could store a more descriptive value range**, such as $250,000 to $299,999 Appraised Value, **for grouping + filtering**
* The basic coverage limit is likely to be more **standardized + not continuously-valued**, like “Replacement Value” or “Up to $250,000”
* In this case, it would **also be treated as a dimension attribute**

#### Degenerate Dimension

* The policy number will be handled as a **degenerate dimension** if you have **extracted all** the policy header **information into other dimensions**
* a) **Dimension “keys” in the fact table that**, however, **do not JOIN to a corresponding dimension table b/c all their interesting attributes have already been placed in *other* analytic dimensions**
* b) **Dimension-*like* attributes stored in a fact table** (essentially **single-attribute dimensions** + thus **don't warrant implementation of an entire dimension table**)
* Most **common with transaction and accumulating snapshot fact tables**
* Obviously want to **avoid creating a policy transaction fact table w/ just a small number of keys while embedding all descriptive details** (including policyholder, dates, + coverages) **in an** **overloaded** policy **dimension**
* In some cases, there may be 1 or 2 attributes that still belong to the policy + *not* to another dimension
* Ex: If the underwriter establishes an overall risk grade for the policy based on the totality of the coverages + covered items, then this risk grade probably belongs in a policy dimension
* *Of course, then the policy number is no longer a degenerate dimension*

#### Low-Cardinality Dimension Tables

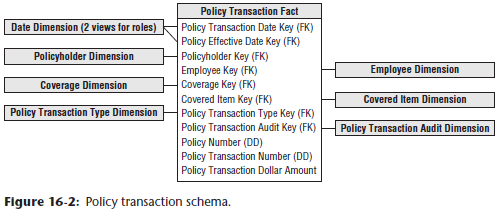
* The policy transaction type dimension is a small dimension for the transaction types listed earlier with reason descriptions, + might contain < 50 rows
* Even though this table is both **narrow in terms of the number of columns + shallow in terms of the number of rows,** the **attributes should still be handled in a dimension** table
* **If textual characteristics are used for query filtering or report labeling, they belong in a dimension**

#### Audit Dimension

* You have the option to **associate ETL process metadata with transaction fact rows** by **including a key that links to an audit dimension row created by the extract process**
* Chapter 6 🡪 **each audit dimension row describes the data lineage of the fact row, including time of the extract, source table, + extract software version**

#### Policy Transaction Fact Table

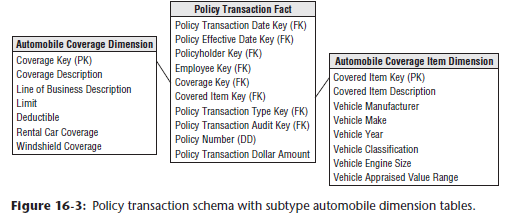
* Policy transaction fact table below illustrates several characteristics of a **classic transaction-grain fact** table



* First, the **fact table consists almost entirely of keys**
* ***Transaction* schemas enable you to analyze behavior in extreme detail**
* As you **descend to *lower* granularity with *atomic* data, the fact table naturally sprouts more dimensionality**
* In this case, the fact table has a single numeric fact, + *interpretation of the fact depends on the corresponding transaction type dimension*
* Because there are different kinds of transactions in the same fact table, in this scenario, you cannot label the fact more specifically

#### Heterogenous Supertype and Subtype Products

* Although there is strong support for an enterprise-wide perspective at the insurance company, business users **don’t want to lose sight of their line-of-business specifics**
* Insurance companies typically are involved in **multiple, very different lines of business**
* Ex: Detailed parameters of homeowners’ coverages differ significantly from auto coverages, + these both differ substantially from personal property coverage, general liability coverage, + other types of insurance.
* Although all coverages can be coded into the generic structures used so far in this chapter, insurance companies want to track numerous *specific* attributes that *make sense only for a particular coverage + covered item*
* Can generalize the initial schema developed above by using the **supertype** and **subtype** technique discussed in Chapter 10
* **Supertype fact table = *intersection* of facts from all account/product/etc. types (along with a supertype dimension table containing the common attributes)**
* **Subtype fact table** **= *systematically*-built separate fact tables (+ associated dimension tables) *for each of the subtypes of a supertype fact table***
* Below shows a schema to handle the specific attributes that describe automobiles + their coverages



* **For each line of business** (or coverage type), **subtype dimension tables for both the covered item and associated coverage are created**
* When a BI application needs the specific attributes of a single coverage type, it uses the appropriate subtype dimension tables
* **Notice in this schema 🡪 don’t need separate line-of-business fact tables, because the metrics don’t vary by business**, **BUT** you’d **likely put a View on the supertype fact table to present only rows for a given subtype**
* The **subtype dimension tables are introduced to handle the special line-of-business attributes**
* **No new keys need to be generated,** since, logically, **all we are doing is extending existing dimension rows**

#### Complementary Policy Accumulating Snapshot

* Finally, consider the use of an **accumulating snapshot** to capture the cumulative effect of the transactions
* **Each row summarizes the measurement events occurring at predictable intermediate steps between the beginning and the end of a process**
* **Primary differentiator of accumulating snapshot = revisit + update fact rows as activity occurs**
* Fits most naturally w/ **short-lived processes w/ a definite beginning + end w/ predictable intermediate steps**
* In this scenario, the **grain of the accumulating snapshot fact table likely would be 1 row for each coverage + covered item on a policy**
* Can **envision including policy-centric dates**, such as quoted, rated, underwritten, effective, renewed, + expired
* Likewise, **multiple employee roles could be included on the fact table** for the agent + underwriter
* Many other dimensions discussed earlier would be applicable to this schema, w/ the exception of the transaction type dimension
* The **accumulating snapshot likely would have an expanded fact set**
* Chapter 4: Inventory 🡪 **an accumulating snapshot is effective for representing information about a pipeline process’s key milestones**
* **Captures the cumulative lifespan** of a policy, covered items, + coverages
* However, **does NOT store information about *each and every* transaction that occurred**
* **Unusual transactional events or unexpected outliers from the standard pipeline would likely be masked with an accumulating perspective**
* On the other hand, **an accumulating snapshot, sourced from the transactions, provides a clear picture of the durations or lag times between key process events**

### Premium Periodic Snapshot

* The prior policy transaction schema is useful for answering a wide range of questions
* *However*, the **blizzard of transactions makes it difficult to quickly determine status or financial value** of an in-force policy **at a given point in time**
* Even if all necessary details lie in the transaction data, a **snapshot perspective would require rolling the transactions forward from the beginning of history taking into account complicated business rules for when earned revenue is recognized**
* Not only is this **nearly impractical on a single policy**, but it is **also ridiculous to think about generating summary top line views of key performance metrics in this manner**
* The answer to this dilemma = **Create a *separate* fact table that operates as a companion to the policy transaction table**
* In this case, the business process = the **monthly policy premium periodic snapshot**, w/ a granularity of **one row** per coverage and covered item on a policy ***each month***
* **Periodic snapshot = Each row summarizes measurement events occurring over a standard period (day, week, or month, etc.) + the grain = the *period,* NOT the individual transaction**
* ***Long-lived* processes**, such as bank accounts, are **typically better modeled with periodic snapshot fact tables**

#### Conformed Dimensions

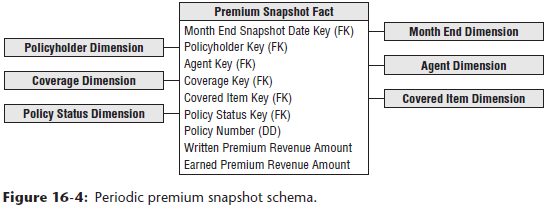
* **When designing the premium periodic snapshot table, strive to reuse as many dimensions from the policy transaction table as possible 🡪 conformed dimensions**
* **Conformed dimensions** **used in *separate* fact tables either must be identical *OR* must represent a shrunken subset of the attributes from the granular dimension**
* Policyholder, covered item, + coverage dimensions would be identical.
* Daily date dimension would be replaced with a **conformed month dimension table**
* Don’t need to track *all* employees involved in policy transactions on a monthly basis, but it may be useful to retain the involved agent, especially because field operations are so focused on ongoing revenue performance analysis
* **Transaction type dimension would NOT be used because it does not apply at the periodic snapshot granularity**
* Instead, **introduce a “status” dimension** so **users can quickly discern the current state of a coverage or policy**, such as new policies or cancellations this month and over time

#### Conformed Facts

* While on the topic of conformity, you **also need to use conformed facts**
* **If the same facts appear in multiple fact tables, such as in the snapshot fact table as well as the consolidated fact table discussed later, then they *must* have consistent definitions + labels**
* **If the facts are *not* identical, they need to be given *different* names.**

#### Pay-in-Advance Facts

* Business management wants to know how much premium revenue was *written* (or sold) each month, as well as how much revenue was *earned*
* Although a policyholder may contract + pay for coverages on covered items for a period of time, the revenue is not *earned* until the service is provided
* **In the case of the insurance company, the revenue from a policy is earned month by month as long as the policyholder doesn’t cancel**
* The **correct calculation of a metric like earned premium would mean fully replicating all the business rules of the operational revenue recognition system within the BI application**
* Typically, the **rules for converting a transaction amount into its monthly revenue impact are complex**, especially with mid-month coverage upgrades + downgrades
* Fortunately, **these metrics can be sourced from a separate operational system**.
* As illustrated below, we include 2 premium revenue metrics in the periodic snapshot fact table to handle the different definitions of written vs. earned premium



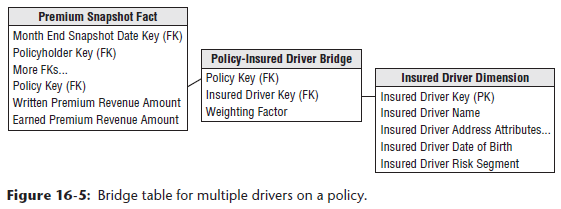
* Simplistically, if an annual policy for a given coverage + covered item was written on January 1 for a cost of $600, the written premium for January would be $600, but the *earned* premium is $50 ($600 divided by 12 months)
* In February, the written premium is - + the earned premium is still $50
* If the policy is canceled on March 31, the earned premium for March is $50, *while the written premium is a negative $450*
* Obviously, at this point the earned revenue stream comes to a crashing halt
* **Pay-in-advance business scenarios typically require the combination of transaction *and* monthly snapshot fact tables to answer questions of transaction frequency + timing, as well as questions of earned income in a given month**
* **Can almost never add enough facts to a snapshot schema to do away with the need for a transaction schema, or vice versa**

#### Heterogenous Supertypes and Subtypes Revisited

* We are again confronted with the **need to look at the snapshot data with more specific line-of-business attributes, + grapple with snapshot facts that vary by line of business**
* Because the **custom facts for each line are incompatible with each other, most of the fact row would be filled with NULLs if you include *all* the line-of-business facts on every row**
* In this scenario, **the answer = separate the monthly snapshot fact table physically by line of business**
* **End up with the single supertype monthly snapshot schema + a series of subtype snapshots, one for each line of business or coverage type**
* **Each of the subtype snapshot fact tables = a copy of a *segment* of the supertype fact table for *just those coverage keys + covered item keys belonging to a particular line of business***
* We include the supertype facts as a convenience so analyses within a coverage type can use both the supertype + custom subtype facts without accessing 2 large fact tables

#### Multivalued Dimensions Revisited

* Auto insurance provides another opportunity to discuss multivalued dimensions.
* Often multiple insured drivers are associated with a policy
* **Can construct a bridge table, illustrated below, to capture the relationship between the insured drivers + policy**



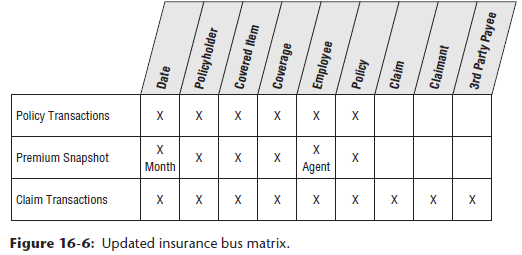
* In this case the insurance company can assign realistic **weighting factors** based on each driver’s share of the total premium cost
* **Because these relationships may change over time, you can add effective + expiration dates to the bridge table**
* Before you know it, you end up with a **fact-less fact table** **to capture the evolving relationships** between a policy, policy holder, covered item, + insured driver over time

### More Insurance Case Study Background

* Unfortunately, the insurance business has a downside
* We learn from interviewees that there’s more to life than collecting premium revenue payments
* **The main costs in this industry result from claim losses**
* After a policy is in effect, then a claim can be made against a specific coverage + covered item
* A **claimant**, who may be the policyholder or a new party not previously known to the insurance company, makes the claim
* When the insurance company opens a new claim, a **reserve** is usually established, which is a preliminary estimate of the insurance company’s eventual liability for the claim
* As further information becomes known, this **reserve can be adjusted**
* Before the insurance company pays any claim, there is usually an investigative phase where they send out an adjuster to examine the covered item + interview the claimant, policyholder, or other individuals involved
* **The investigative phase produces a stream of task transactions**
* In complex claims, various outside experts may be required to pass judgment on the claim + the extent of the damage.
* In most cases, after the investigative phase, the insurance company issues a number of payments, many of which go to 3rd parties such as doctors, lawyers, or auto body shop operators, + some payments may go directly to the claimant
* **It is important to clearly identify the employee responsible for every payment made against an open claim**
* The insurance company may take possession of the covered item after replacing it for the policyholder or claimant
* If the item has any remaining value, **salvage payments** received by the insurance company are a credit against the claim accounting
* **Eventually, the payments are completed and the claim is closed**
* If nothing unusual happens, **this is the end of the transaction stream generated by the claim**
* *However*, in some cases, further claim payments or claimant lawsuits **may force a claim to be reopened**
* **An important measure for an insurance company is how often + under what circumstances claims are reopened**
* In addition to analyzing the detailed claims processing transactions, the insurance company *also* wants to **understand what happens over the life of a claim**
* Ex: The time lag between claim open date + 1st payment date is an important measure of claims processing efficiency

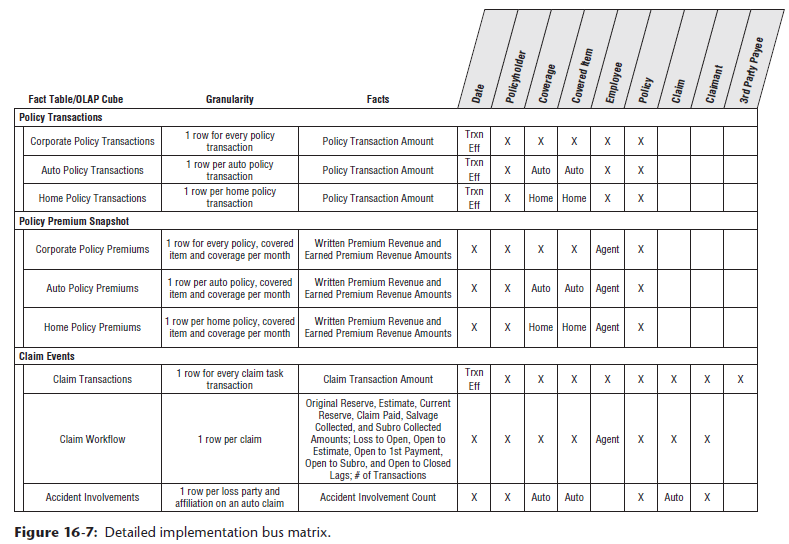
#### Updated Insurance Bus Matrix

* With a better understanding of the claims side of the business, the draft bus matrix from earlier needs to be revisited
* Based on the new requirements, you **add *another* row to the matrix to accommodate claim transactions**
* **Many of the dimensions identified earlier** in the project will be **reused**, but you **also add new columns to the matrix for claim, claimant, + third-party payee**



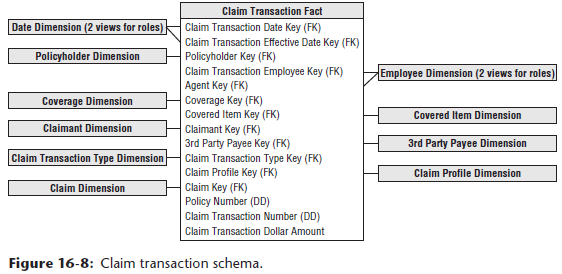
#### Detailed Implementation Bus Matrix

* DW/BI teams sometimes struggle w/ the level of detail captured in an EDW bus matrix.
* In the planning phase of an architected DW/BI project, it makes sense to stick w/ rather high-level business processes (or sources)
* *Multiple* fact tables at *different levels of granularity* may result from each of these business process rows
* **In a subsequent implementation phase, you can take a subset of the matrix to a lower level of detail by reflecting all fact tables/OLAP cubes resulting from the process as separate matrix rows**
* At this point, **the matrix can be enhanced by adding columns to reflect the granularity + metrics associated with each fact table or cube**
* Below illustrates a more detailed implementation bus matrix



### Claim Transactions

* The **operational claim processing system generates a slew of transactions**, including the following **transaction task types**:
* Open claim, reopen claim, close claim
* Set reserve, reset reserve, close reserve
* Set salvage estimate, receive salvage payment
* Adjuster inspection, adjuster interview
* Open lawsuit, close lawsuit
* Make payment, receive payment
* Subrogate claim
* When updating the simpler bus matrix above, you determine that this schema uses a number of dimensions developed for the policy world
* You again have **2 roleplaying dates associated with the claim transactions**
* **Unique column labels should distinguish the claim transaction + effective dates from those associated with policy transactions**
* The employee = the **employee involved in the transactional task**
* As mentioned in the business case, this is **particularly interesting for payment authorization transactions**
* The claim transaction type dimension would include the transaction types + groupings just listed
* As shown in below, there are **several new dimensions in the claim transaction fact table**



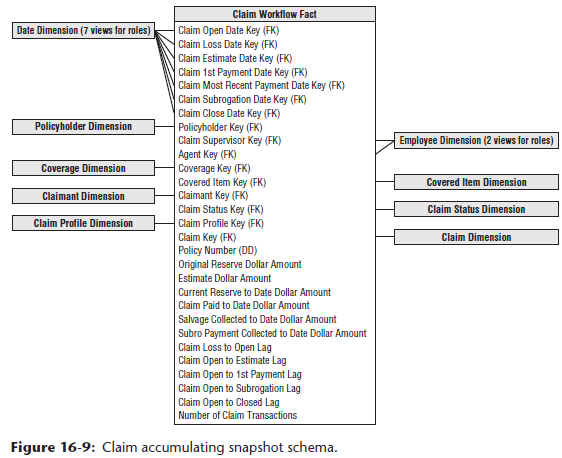
* The **claimant is the party making the claim**, typically an individual., + the **3rd-party payee** may be either an individual or commercial entity
* Both the claimant + payee dimensions usually are **dirty dimensions** because of the **difficulty of reliably identifying them across claims**
* Unscrupulous potential payees may go out of their way not to identify themselves in a way that would easily tie them to other claims in the insurance company’s system

#### Transaction vs. Profiling Junk Dimensions

* Beyond the reused dimensions from the policy-centric schemas + the new claim-centric dimensions just listed, **there are a large number of indicators + descriptions related to a claim**
* Designers are sometimes tempted to **dump all these descriptive attributes into a claim dimension**
* This approach **makes sense for high-cardinality descriptors**, such as the specific address where the loss occurred or a narrative describing the event
* ***However*, in general, avoid creating dimensions with the same number of rows as the fact table**
* Chapter 6 🡪 **low-cardinality codified data**, like method used to report the loss or an indicator denoting whether the claim resulted from a catastrophic event, **are better handled in a junk dimension**
* **Junk dimension = combines together a number of miscellaneous, low-cardinality ﬂags and indicators, rather than making separate dimensions for each ﬂag and attribute**
* Does NOT need to be the Cartesian product of all the attributes' possible values, but **should only contain the combination of values that ACTUALLY occur in the source data**
* In this case, the **junk dimension would more appropriately be referred to as the claim *profile* dimension**, w/ **1** row per unique combination of profile attributes
* Grouping or filtering on the profile attributes would yield faster query responses than if they were alternatively handled as claim dimension attributes

### Claim Accumulating Snapshot

* **Even with a robust transaction schema, there’s a whole class of urgent business questions that can’t be answered using only transaction detail**
* It is difficult to derive claim-to-date performance measures by traversing through every detailed claim task transaction from the beginning of the claim’s history + appropriately applying the transactions
* **On a periodic basis**, perhaps at the close of each day, you can **roll forward all transactions to update an accumulating claim snapshot incrementally**
* **Granularity = 1 row per claim**, + the row is **created once when the claim is opened + then is updated throughout the life of a claim until it is finally closed**
* **Many of the dimensions are reusable** (conformed dimensions) as illustrated below



* Should **include more dates in this fact table to track the key claim milestones + deliver time lags**
* These lags may be the raw difference between 2 dates, or they may be calculated in a more sophisticated way by accounting for only workdays in the calculations
* A **status dimension** is added to quickly identify all open, closed, or reopened claims, for example
* **Transaction-specific dimensions** such as employee, payee, and claim transaction type **are suppressed, whereas the list of additive, numeric measures has been expanded**

#### Accumulating Snapshot for Complex Workflows

* **Accumulating snapshot fact tables are typically appropriate for predictable workflows w/ well-established milestones, + usually have 5-10 key milestone dates representing the pipeline’s start, completion, + key events in between**
* ***However*, sometimes workflows are less predictable 🡪 still have a definite start + end date, but the milestones in between are numerous + less stable**
* Some occurrences may skip over some intermediate milestones, but **there’s no reliable pattern**
* **In this situation, 1st task = identify the key dates that link to role-playing date dimensions**
* **These dates represent the most important milestones**
* The **start + end dates for the process would certainly qualify**, + in addition, **consider other commonly occurring critical milestones**
* These dates (and their associated dimensions) will be used extensively for BI application filtering
* ***However*, if the number of additional milestones is both voluminous *and* unpredictable, they can’t all be handled as additional date FKs in the fact table**
* Typically, **business users are more interested in the lags between these milestones, rather than filtering or grouping on the dates themselves**
* If there were 20 total potential milestone events, there’d be 190 potential lag durations: event A-to-B, A-to-C, … (19 possible lags from event A), B-to-C, … (18 possible lags from event B), etc.
* **Instead of physically storing 190 lag metrics, you can get away with just storing 19 of them + then calculate the others**
* Because every pipeline occurrence starts by passing through milestone A, the workflow begin date, you could store all 19 lags from the anchor event A + then calculate the other variations
* Ex: To get the lag from B-to-C, take the A-to-C lag value + subtract the A-to-B lag
* If there happens to be a NULL for one of the lags involved in a calculation, then the result also needs to be NULL because one of the events never occurred
* But such a NULL result is handled gracefully if you are counting or averaging that lag across a number of claim rows

#### Timespan Accumulating Snapshot

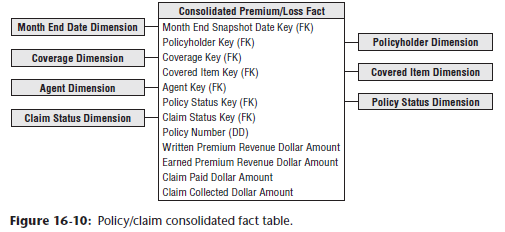
* **An accumulating snapshot does a great job presenting a workflow’s *current* state, but it obliterates the intermediate states**
* Ex: A claim can move in + out of various states such as opened, denied, closed, disputed, opened again, + closed again
* The **claim transaction fact table will have separate rows for each of these events, but as discussed earlier, *it doesn’t accumulate metrics across transactions*,** + trying to re-create the evolution of a workflow from these transactional events would be a nightmare
* Meanwhile, **a classic accumulating snapshot doesn’t allow you to re-create the claim workflow at any arbitrary date in the past**
* Alternatively, you **could add effective + expiration dates to the accumulating snapshot**
* **Instead of destructively updating each row as changes occur, add a new row that preserves the state of a claim for a span of time**
* **Similar to a type 2 SCD**, the fact row includes the following additional columns:
* Snapshot start date
* Snapshot end date (updated when a new row for a given claim is added)
* Snapshot current flag (updated when a new row is added)
* **Most users are only interested in the current view provided by a classic accumulating snapshot, + you can meet their needs by defining a View that filters the *historical* snapshot rows based on the current flag**
* The minority of users + reports who need to look at the pipeline as of any arbitrary date in the past can do so by filtering on the snapshot start + end dates.
* **The timespan accumulating snapshot fact table is more complicated to maintain than a standard accumulating snapshot, but the logic is similar**
* **Where the *classic accumulating* snapshot updates a row, the *timespan* snapshot updates the administrative columns on the row formerly known as current, + inserts a new row**

#### Periodic Instead of Accumulating Snapshot

* **In cases where a claim is not so short-lived**, such as with long-term disability or bodily injury claims that have a multiyear life span, you **may represent the snapshot as a periodic snapshot rather than an accumulating snapshot**
* **Grain of the periodic snapshot = 1 row for every active claim at a regular snapshot interval**, such as monthly
* The **facts would represent numeric, additive facts that occurred during the period** such as amount claimed, amount paid, + change in reserve

### Policy/Claim Consolidated Periodic Snapshot

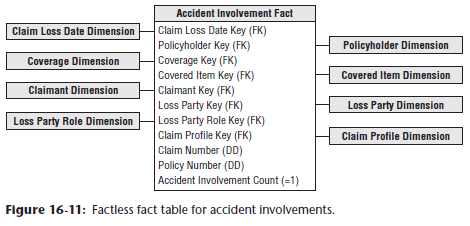
* With the fact tables designed thus far, you can deliver a robust perspective of the policy + claim transactions, in addition to snapshots from both processes
* *However,* **business users are also interested in profit metrics**
* Although premium revenue + claim loss financial metrics could be derived by separately querying 2 fact tables + then combining the results set, you opt to **go the next step in the spirit of ease of use + performance for this common drill-across requirement**
* Can construct *another* fact table that brings together the premium revenue + claim loss metrics:



* This table has a **reduced set of dimensions corresponding to the lowest level of granularity common to both processes**
* Chapter 7: Accounting 🡪 this is a **consolidated fact table** because it **combines data from multiple business processes**
* It is **best to develop consolidated fact tables after the base metrics have been delivered in separate atomic dimensional models**

### Fact-less Accident Events

* We earlier described **fact-less fact tables as the collision of keys at a point in space + time**
* In the case of an auto insurer, you can record literal collisions using a fact-less fact table
* In this situation, **the fact table registers the many-to-many correlations** between loss parties + loss items, or in laymen’s terms, all correlations **between the people + vehicles involved in an accident**
* 2 new dimensions appear in the fact-less fact table below



* **Loss party** captures the individuals involved in the accident, whereas **loss party role** identifies them as passengers, witnesses, legal representation, or some other capacity
* As in Chapter 3: Retail Sales, we **include a fact that is always valued at 1 to facilitate counting and aggregation**
* This fact-less fact table can represent complex accidents involving many individuals + vehicles because **the number of involved parties with various roles is open-ended.**
* **When there is > 1 claimant or loss party associated w/ an accident, can optionally treat these dimensions as multivalued dimensions using claimant group + loss party group bridge tables**
* This **has the advantage that the grain of the fact table is preserved = 1 record per accident claim**
* Either schema variation could answer questions such as, “How many bodily injury claims did you handle where ABC Legal Partners represented the claimant and EZ-Dent-B-Gone body shop performed the repair?”

### Common Dimensional Modeling Mistakes to Avoid

* Now, we focus on NOT-to-do’s, by elaborating on dimensional modeling techniques that should be avoided **in reverse order of importance**
* **Be aware, however, that even the less important mistakes can seriously compromise a DW/BI system**

#### Mistake 10: Place Text Attributes in a Fact Table

* **The process of creating a dimensional model is always a kind of triage**
* **The numeric measurements delivered from an operational business process source belong in the fact table**
* **Descriptive textual attributes comprising the *context* of the measurements go in dimension tables**
* In nearly every case, **if an attribute is used for constraining + grouping, it belongs in a dimension table**
* Finally, make **a field-by-field decision about leftover codes + pseudo-numeric items**, placing them in the **fact table if they’re more like measurements + used in calculations** or in a **dimension table if they are more like descriptions used for filtering + labeling**
* **Don’t leave “true text”, *especially* comment fields, in the fact table**
* **Get these text attributes** off the main runway of the DW + **into dimension tables**

#### Mistake 9: Limit Verbose Descriptors to Save Space

* You might think you are being a conservative designer by keeping the size of the dimensions under control
* **However, in virtually every DW, dimension tables are geometrically smaller than the fact tables**
* Having a 100 MB product dimension table is insignificant if the fact table is 100 or 1000X as large
* **Job as designers of easy-to-use dimensional models = supply *as much* verbose descriptive context in each dimension as possible**
* Make sure **every code is augmented with readable descriptive text**
* Remember the **textual attributes in dimension tables** provide the **browsing, constraining, or filtering parameters in BI applications**, as well as the **content for row + column headers in reports**

#### Mistake 8: Split Hierarchies into Multiple Dimensions

* A **hierarchy = a cascaded series of many-to-one relationships**
* Ex: Many products roll up to a single brand, + many brands roll up to a single category
* **If a dimension is expressed at the *lowest* level of granularity, such as product, then all higher levels of the hierarchy can be expressed as unique values in the product row**
* Business users understand hierarchies, + your job is to **present hierarchies in the most natural and efficient manner in the eyes of the *users*, NOT in the eyes of a data modeler who has focused his entire career on designing 3NF entity-relationship models for transaction processing systems**
* **A fixed-depth hierarchy belongs together in a *single* physical flat dimension table, *UNLESS* data volumes or velocity of change dictate otherwise**
* **Resist urges to snowflake a hierarchy by generating a set of progressively smaller subdimension tables**
* Finally, **if more than 1 rollup exists simultaneously for a dimension, in most cases it’s perfectly reasonable to include multiple hierarchies in the same dimension *as long as the dimension has been defined at the lowest possible grain (+ the hierarchies are uniquely labeled)***

#### Mistake 7: Ignore the Need to Track Dimension Changes

* Contrary to popular belief, **business users often want to understand the impact of changes on at least a subset of the dimension tables’ attributes**
* It is unlikely users will settle for dimension tables with attributes that always reflect the current state of the world
* **3 basic techniques (types 1 (overwrite), 2 (new dimension row), and 3 (new dimension column)) track slowly moving attribute changes, + *don’t rely on type 1 exclusively***
* Likewise, **if a group of attributes changes rapidly, you can *split* a dimension to capture the more volatile attributes in a mini-dimension**

#### Mistake 6: Solve All Performance Problems with More Hardware

* **Aggregates, or derived summary tables, are a cost-effective way to improve query performance**
* **Most BI tool vendors have explicit support for the use of aggregates**
* **Adding expensive hardware should be done as part of a *balanced program* that includes building aggregates, partitioning, creating indices, choosing query-efficient DBMS software, increasing real memory size, increasing CPU speed, + adding parallelism at the hardware level**

#### Mistake 5: Use Operational Keys to JOIN Dimensions and Facts

* Novice designers are sometimes too literal minded when designing dimension tables’ PKs that connect to the fact tables’ FKs
* It is counterproductive to declare a suite of dimension attributes as the dimension table key + then use them *all* as the basis of the physical JOIN to the fact table
* This includes the unfortunate practice of declaring the dimension key to be the operational key, along with an effective date
* All types of ugly problems will eventually arise
* **The dimension’s operational or intelligent key should be *replaced* with a simple integer surrogate key sequentially numbered from 1 to *N*, where *N* = total number of rows in the dimension table**
* **The date dimension is the sole exception to this rule**.

#### Mistake 4: Neglect to Declare and Comply with the Fact Grain

* **ALL dimensional designs should begin by articulating the business process that generates the numeric performance measurements**
* Second, **the *exact* granularity of that data MUST be specified**
* Building fact tables at the **most atomic, granular level will gracefully resist the ad hoc attack**
* Third, **surround these measurements with dimensions that are *true to that grain***
* **Staying true to the grain is a crucial step in the design of a dimensional model**
* **A subtle but serious design error is to add helpful facts to a fact table**, such as rows that describe totals for an extended timespan or a large geographic area (ex: YTD metrics)
* Although these **extra facts are well known at the time of the individual measurement** + would seem to make some BI applications simpler, they **cause havoc because all the automatic summations across dimensions overcount these higher-level facts, producing incorrect results**
* **Each different measurement grain demands its *own* fact table**

#### Mistake 3: Use a Report to Design the Dimensional Model

* A **dimensional model** has NOTHING to do with an intended report!
* Rather, it is **a model of a *measurement process***
* Again: **Numeric measurements form the basis of fact tables, + the dimensions appropriate for a given fact table are the *context* that describes the circumstances of the measurements**
* **A dimensional model is based solidly on the physics of a measurement process + is quite independent of how a user chooses to define a report**
* Ex: A project team confessed it had built several hundred fact tables to deliver order management data to its business users, + it turned out each fact table had been constructed to address a specific report request, such that the *same* data was extracted many, many times to populate all these *slightly* different fact tables
* Now, the team was struggling to update the databases w/in the nightly batch window
* Rather than designing a quagmire of report-centric schemas, the team should have **focused on the measurement processes**
* The **users’ requirements could have been handled with a well-designed schema for the atomic data *along with* a handful (NOT hundreds) of performance-enhancing aggregations**

#### Mistake 2: Expect Users to Query Normalized Atomic Data

* **The *lowest level* data is always the most dimensional + should be the foundation of a dimensional design**
* **Data that has been aggregated in *any* way has been deprived of some of its dimensions**
* You **can’t build a dimensional model with aggregated data and then expect users + their BI tools to seamlessly drill down to 3NF data for the atomic details**
* **Normalized models may be helpful for preparing the data in the ETL kitchen, but they should NEVER be used for presenting the data to business users**

#### Mistake 1: Fail to Conform Facts and Dimensions

* These 2 separate mistakes are both **so dangerous to a successful DW/BI design**
* It would be a shame to get this far + then build isolated data repository stovepipes
* **If you have a numeric measured fact, such as revenue, in 2+ databases sourced from different underlying systems, you need to take special care to ensure the technical definitions of these facts *exactly match***
* **If the definitions do NOT exactly match, then they shouldn’t *both* be referred to as revenue**
* ***This* is conforming the facts**
* **Single most important design technique in dimensional modeling = conforming *dimensions***
* **If 2+ fact tables are associated with the *same* dimension, you must be *fanatical* about making these dimensions identical (or carefully chosen subsets of each other)**
* **When you conform dimensions across fact tables, you can drill across separate data sources because the constraints + row headers mean the same thing and match at the data level**
* **Conformed dimensions are the secret sauce needed for building *distributed* DW/BI environments, adding unexpected new data sources to an existing DW, + making multiple incompatible technologies function together harmoniously**
* **Conformed dimensions also allow teams to be more agile because they’re not re-creating the wheel repeatedly, which translates into a faster delivery of value to the business community**