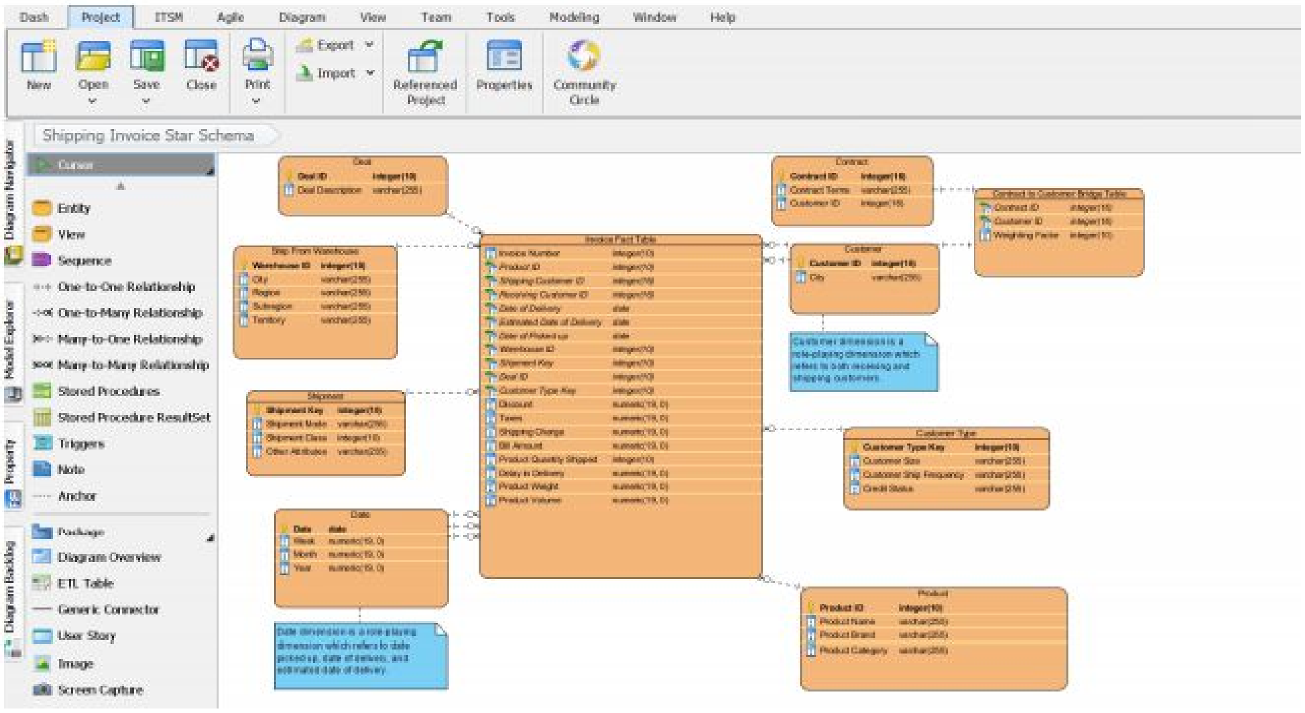
Victor Xu
MSIS 543, Final

Part A: Star Schema Exercise - Data Warehouse Modeling

Important notes:

- The following diagram is designed using Visual Paradigm.
- All descriptions are selectively quoted from the 543 Final instructions. I have asked my professor for approval.



Dash
Project
ITSM
Agile
Diagram
View
Team
Tools
Modeling
Window
Help
New
Open
Save
Close
Print
Export
Import
Referenced Project
Properties
Community Circle
Shipping Invoice Star Schema
Cursor
Entity
View
Sequence
One-to-One Relationship
One-to-Many Relationship
Many-to-One Relationship
Many-to-Many Relationship
Stored Procedures
Stored Procedure ResultSet
Triggers
Note
Anchor
Package
Diagram Overview
ETL Table
Generic Connector
User Story
Image
Screen Capture
Diagram Navigator
Model Explorer
Property
Diagram Backlog
Deal
Deal ID integer(10)
Deal Description varchar(255)
Ship From Warehouse
Warehouse ID integer(10)
City varchar(255)
Region varchar(255)
Subregion varchar(255)
Territory varchar(255)
Shipment
Shipment Key integer(10)
Shipment Mode varchar(255)
Shipment Class integer(10)
Other Attributes varchar(255)
Date
Date date
Week numeric(19, 0)
Month numeric(19, 0)
Year numeric(19, 0)
Date dimension is a role-playing dimension which refers to date picked up, date of delivery, and estimated date of delivery.
Invoice Fact Table
Invoice Number integer(10)
Product ID integer(10)
Shipping Customer ID integer(10)
Receiving Customer ID integer(10)
Date of Delivery date
Estimated Date of Delivery date
Date of Picked up date
Warehouse ID integer(10)
Shipment Key integer(10)
Deal ID integer(10)
Customer Type Key integer(10)
Discount numeric(19, 0)
Taxes numeric(19, 0)
Shipping Charge numeric(19, 0)
Bill Amount numeric(19, 0)
Product Quantity Shipped integer(10)
Delay in Delivery numeric(19, 0)
Product Weight numeric(19, 0)
Product Volume numeric(19, 0)
Contract
Contract ID integer(10)
Contract Terms varchar(255)
Customer ID integer(10)
Contract to Customer Bridge Table
Contract ID integer(10)
Customer ID integer(10)
Weighting Factor integer(10)
Customer
Customer ID integer(10)
City varchar(255)
Customer dimension is a role-playing dimension which refers to both receiving and shipping customers.
Customer Type
Customer Type Key integer(10)
Customer Size varchar(255)
Customer Ship Frequency varchar(255)
Credit Status varchar(255)
Product
Product ID integer(10)
Product Name varchar(255)
Product Brand varchar(255)
Product Category varchar(255)

The descriptions of the Star Schema are:

- A shipping company serves a limited group of customers.
- A shipment invoice contains the following information:
    - Invoice Number; Date Picked Up; Date of Delivery; Ship From Customer; Ship To Customer; Ship From Warehouse; Ship From City; Ship To City; Shipment Mode (like air/sea/truck/train); Shipment Class (codes like 1,2,3 which translate to express, expedited, standard); Contract ID; Total Shipping Charges; Deal ID (refers to a discount deal in effect); Discount Amount; Taxes; Total Billed Amount; Total Ship Weight; Total Ship Volume
- Each Line in Invoice contains: Product ID, Product Name, Quantity, Ship Weight, Ship Volume, Charge

Assumptions:

- Quantity is number of items
- Discount and Taxes can be allocated per line
- Number of contracts is relatively small. Each customer may have multiple contracts
- Number of Deals is relatively small. Some shipments may have no deal applicable

Requirement:

- Design a dimensional model, with invoice line as the grain of the fact table.

Victor Xu
MSIS 543, Final

Fact Table (shipment invoice line)
Invoice Number (PK, DD)
Invoice_Product (FK)
Deal ID (FK)

Date Picked Up Key (FK)
Date of Delivery Key (FK)
[*both access Date Dimension*]

*Customer ship from Key (FK)
Ship from Key (FK) - inside this dim table = correlated warehouse and city info
Shipment_Info (FK) - inside this table = correlated modes and class info
Ship to Key (FK)
*Customer ship to Key (FK)
[*both access Customer Dimension*]
Contract ID (FK)

Discount Dollar Amount
Taxes Dollar Amount
Total Ship Weight
Total Ship Volume
Total Billed Amount

[Additional attributes not necessary in this question but in real life may include:]
Customer satisfaction key (FK)
Fixed cost
Variable cost
Distribution cost
Contribution dollar amount
Shipment line item on time count
Shipment line item complete count
Shipment line item damage free count

Dimension Tables:
Date Dimension: Key, Date, other attributes

Customer Dimension: cust_key, name, street address, city, state, zip, phone, other attributes
Potential outrigger table: attributes for customer size, customer ship frequency, credit status (low cardinality, depends on how often this information is queried)

Victor Xu
MSIS 543, Final

Invoice_Product Dimension: (invoice number, product ID), product name, category, quantity, individual product ship weight, individual product ship volume, individual product charge, other attributes
(1 to many relationship with fact table)

Ship-from Dimension: key, warehouse name, street, city, state, contact person, facility type, subregion, region, territory, other attributes
Contains one row for each manufacturer warehouse or shipping location

Ship-to Dimension: key, warehouse name, street, city, state, contact person, facility type, subregion, region, territory, other attributes
Could consider merging ship from and ship to tables (i.e. larger ship_id table, perhaps w. outrigger for storing overlapping information…depends on pragmatics of querying)

Shipment_Info Dimension: ID, shipment mode, shipment class, total weight, total volume, total charges, other attributes
Descriptions for a particular shipment mode + class for estimated delivery duration

Contract Dimension: ID, contract duration start/end, Contract Terms, Policies, handling errors/delays in delivery, other attributes
Assuming that each customer may have multiple contracts, but a unique contract would only apply to 1 customer

Deal Dimension: ID, deal terms, discount rates, other attributes

Question: At the moment we cannot track information about deals in existence on any given date with the fact table of invoice line items. We would need to join the FT and deal dimensional table (since only the deal ID is available immediately in the FT). When we need detailed attributes like existence by date, we would need to query something like:

```
SELECT interested_deal_attributes
FROM Deal
INNER JOIN Invoice
ON Deal.Deal_ID = Invoice.Deal_ID
WHERE Date_of_Delivery = givenDate
```

Another option might be creating a mini-dimension with just this required information (subset of attributes from deal dimension table) for faster querying, depending on business pragmatics.

Victor Xu
MSIS 543, Final

Part B:

1. State what each of the following is **briefly**. Provide an example for each.

a. MiniDimension

Stores a subset of attributes which change frequently (or are often queried together in combination) from a larger dimension, and becomes connected to the fact table as a separate dimension. Minidimensions consist of correlated clumps of attributes and support consistent day-to-day analytic reporting (business pragmatics).

Example: A large customer dimension includes attributes including age, number of children, income level. If these particular attributes are changed or analyzed frequently, we could have them removed to create a demographics minidimension (with a new FK added to the fact table) for better efficiency. Attributes not as frequently queried are left in the original large cust. Dim.

b. Junk Dimension

Stores miscellaneous attributes (i.e. low cardinality indicators, flag codes, text) that are unrelated to any particular dimension, and hence stored in a separate table for convenience. These attributes are not accessed frequently.

Example: A hospital's DW has a gender dimension and marital status dimension. In the fact table, there are currently 2 keys referring to these dimensions (if kept as single dimensions, would be limited to single attributes). To eliminate these small dimensions, a junk dimension cross joins all possible attributes (gender, marital status) into a single dimension, with just a single key in the fact table.

c. Degenerate Dimension

A dimension key (in the fact table) that does not join to a corresponding dimension table, and has no attributes. Common when the grain of a FT represents a single transaction item/line item

Example: Operational control #s: Order numbers, invoice numbers, bill-of-lading numbers
These can be maintained in the fact table instead of in separate dimensions, as we have already extracted info (attributes) from other dimensions. A more specific example: PK of the retail sales FT consists of a degenerate POS transaction number and product key (assuming the POS system rolls up all sales for a given product within a POS shopping cart into a single line item).

d. MultiValued Dimension

A dimension with more than 1 value per fact row, compared to the classic dimensional schema where each dimension attached to the FT has a single value. Bridge tables are often used.

Example: A patient receiving a healthcare treatment may have multiple concurrent diagnoses. The multivalued dimension is attached to the FT via a group dimension key to a bridge table, with one row for each concurrent diagnosis in a group.

Another example (textbook): Bank account can have one, two, or more individual account holders. Including customer as an account attribute violates the granularity of the dimension table. And including customer as an additional dimension in the fact table violates the granularity of the FT, since more than one individual can be associated with any given account.

2. *You created an Employee dimension for a company with 50,000 employees, and identified Team Assignment as an attribute that changes every six months. Management wants to track history of Team assignment. How will you handle change of this attribute?*

I would use SCD type 2 to track history of Team Assignment, that is, inserting new rows to capture the assignment changes (using surrogate keys). This technique is powerful because the new dimension row automatically partitions history in the fact table, and is gracefully extensible (not a re-design of the tables). I believe this method is appropriate for now, because Kimball & Ross mentions type 2 becomes problematic when dimension tables already exceed a million rows. Since we are basically adding 50,000 rows every 6 months, we can use this method to store 10 years' worth of data before reaching this threshold.

In another project, you find a similar situation, with team assignment changing every two weeks in a very large employee dimension. Management wants to track history of team assignment here as well. How will you handle change of this attribute?

After initially considering hybrid approaches, I realized that with the fast (and large) changes happening every 2 weeks, we're not really dealing with 'slowly' changing dimension anymore (so the previous SCD techniques aren't optimal...it's a fast changing dimension). My approach would be breaking off the 'Team Assignment' attribute from the Employee dimension (along with any other attributes changed at this frequency) into a separate minidimension. The fact table would then have two foreign keys—one for the primary Employee dimension table and another for the rapidly changing 'Team Assignment' attribute. These dimension tables would be associated with one another every time we put a row in the fact table.

Victor Xu
MSIS 543, Final

3. A.*Give an example of what could be a mini-dimension that spins off from one of the dimensions in this diagram (make suitable assumptions). Identify which dimension you take the mini-dimension out from, what kind of information may be part of the mini-dimension, and the reasoning behind spinning off the mini-dimension.*

One example would be 'Policyholder Dimension', as we may want to make mini-dimension(s) containing attributes like policyholder's ZIP code (also for number of children, income level, etc.). The ZIP code would change as the policyholder moves, and is closely correlated to the insurer's pricing and risk algorithms for these customers. The company wants to track these changes, instead of just overwriting records. We may assume that a national insurance company will have well over 1 million rows in the large policyholder dimension table. If true, there would be significant burden on the data staging (as the number of rows grow) to support tracking of these fast changing attributes. So splitting into a minidimension directly linked to the fact table with a separate surrogate key will improve the efficiency of attribute browsing (since the company needs to frequently query + update these attributes). Whether ZIP code is the only attribute in a mini-dimension (versus combining other attributes like income level) depends on how frequently these attributes are queried together (correlation/business pragmatics), and whether updating occurs at similar periods. Other examples may be found in the covered item dimension.

B. *There are different kinds of employees who handle the different transaction types, for example, these could be sales rep, agent, auditor. Currently they are all part of the single Employee dimension. Let us say that individual employee types have some attributes unique to each, while they continue to share the common attributes currently in the Employee dimension. If we also wanted to include the custom employee type attributes into the schema, how will we do that?*
Instinctively I think this would be a good example of a permissible outrigger, that is, creating another table (connected to the Employee dimension). If this question implies the common attributes should continue to stay in the Employee dimension, then we'd place the custom employee type attributes into the outrigger (for more detailed querying). This is assuming that these custom employee type attributes have low-cardinality and perhaps aren't queried super frequently. (If the implication is false, we might switch the contents of the outrigger and Employee dimension tables around, depending on what reduces redundancy of repeated information).

But these decisions are ultimately driven by business pragmatics. If somehow these custom employee type attributes become super frequently analyzed in the future, we might instead consider placing them into a mini-dimension connected to the FT for operationally faster quieres.