



DeepSee Model Design Guide

Version 2011.1

26 May 2011

This book was assembled and formatted in Adobe Page Description Format (PDF) using tools and information from the following sources: Sun Microsystems, RenderX, Inc., Adobe Systems, and the World Wide Web Consortium at www.w3c.org. The primary document development tools were special-purpose XML-processing applications built by InterSystems using Caché and Java.



Caché WEBLINK, Distributed Cache Protocol, M/SQL, M/NET, and M/PACT are registered trademarks of InterSystems Corporation.



InterSystems Jalapeño Technology, Enterprise Cache Protocol, ECP, and InterSystems Zen are trademarks of InterSystems Corporation.

All other brand or product names used herein are trademarks or registered trademarks of their respective companies or organizations.

This document contains trade secret and confidential information which is the property of InterSystems Corporation, One Memorial Drive, Cambridge, MA 02142, or its affiliates, and is furnished for the sole purpose of the operation and maintenance of the products of InterSystems Corporation. No part of this publication is to be used for any other purpose, and this publication is not to be reproduced, copied, disclosed, transmitted, stored in a retrieval system or translated into any human or computer language, in any form, by any means, in whole or in part, without the express prior written consent of InterSystems Corporation.

The copying, use and disposition of this document and the software programs described herein is prohibited except to the limited extent set forth in the standard software license agreement(s) of InterSystems Corporation covering such programs and related documentation. InterSystems Corporation makes no representations and warranties concerning such software programs other than those set forth in such standard software license agreement(s). In addition, the liability of InterSystems Corporation for any losses or damages relating to or arising out of the use of such software programs is limited in the manner set forth in such standard software license agreement(s).

THE FOREGOING IS A GENERAL SUMMARY OF THE RESTRICTIONS AND LIMITATIONS IMPOSED BY INTERSYSTEMS CORPORATION ON THE USE OF, AND LIABILITY ARISING FROM, ITS COMPUTER SOFTWARE. FOR COMPLETE INFORMATION REFERENCE SHOULD BE MADE TO THE STANDARD SOFTWARE LICENSE AGREEMENT(S) OF INTERSYSTEMS CORPORATION, COPIES OF WHICH WILL BE MADE AVAILABLE UPON REQUEST.

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Customer Support

Tel: +1 617 621-0700
Fax: +1 617 374-9391
Email: support@InterSystems.com

Table of Contents

Preface	1
1 Introduction	3
2 Core Principles	5
2.1 Core Principles	5
2.2 How Table Relationships Affect Your Model Options	5
2.2.1 Possible Relationships between Tables	5
2.2.2 How You Can Use Other Tables in DeepSee	6
2.3 Usability Considerations	8
3 Suggested Assessment Process	11
3.1 Collecting the Business Questions	11
3.2 Understanding the Tables and Their Relationships	12
3.3 Determining the Table Sizes	12
3.4 Evaluating a Possible Base Table	12
3.5 Evaluating Other Possible Base Tables	14
3.6 Comparing the Models	14
3.7 Using the Models in Combination	15
4 Case Study: Part 1	17
4.1 Overview of the Tables	17
4.1.1 Customers	18
4.1.2 Brokers	18
4.1.3 Assets	19
4.1.4 Transactions	19
4.2 Transactions As the Base Table	19
4.3 A First Look at the Base Table	20
4.4 Additional, More Complex Measures	21
4.5 Using Related Tables	22
4.5.1 Dimensions Based on Customer Information	22
4.5.2 Dimensions Based on Asset Information	22
4.5.3 Dimensions Based on Broker Information	23
4.5.4 Dimensions to Add	23
4.5.5 Possible Pivot Tables	23
4.6 Additional Dimensions Based on Bins or Ranges	24
4.6.1 Dimensions to Add	24
4.6.2 Possible Pivot Table	24
4.7 Additional Measures Based on Dimension Member Counts	25
4.7.1 Measures to Add	25
4.7.2 Possible Pivot Table	25
4.8 Model Review	25
4.9 The Final Model (Transactions)	26
5 Case Study: Part 2	27
5.1 Customers As the Base Table	27
5.1.1 Using the Fields of the Customers Table	27
5.1.2 Using Related Tables	28
5.1.3 Additional Dimensions from Related Tables	28
5.1.4 Additional Measures from the Transactions Table	29

5.1.5 Additional Measures Based on Dimension Member Counts	30
5.1.6 Model Review	30
5.1.7 The Final Model (Customers)	30
5.2 Comparing the Models	31
5.2.1 Dimensions	31
5.2.2 Measures	31
5.2.3 Comparison Summary	32
5.3 Using the Models in Combination	33
5.4 Final Comments	33
6 Additional Considerations and Options	35
6.1 Null Values	35
6.1.1 Null Values in Dimensions	35
6.1.2 Null Values and Measures	37
6.2 Date and Time Data	37
6.2.1 Date Dimensions	37
6.2.2 Time Dimensions	39
6.2.3 Date Measures	39
6.3 Compound Members	40

List of Tables

Table 5–1: Dimensions in the Transaction and Customer Models 31

Table 5–2: Measures in the Transaction and Customer Models 32

Preface

Important: This book is for DeepSee I. For information on DeepSee II, see [Getting Started with DeepSee II](#), which also lists the other books for DeepSee II.

This book is meant for all users of DeepSee and is intended to give you a solid idea of the models that you can create in DeepSee. These models are the basis for all your DeepSee work. It contains the following sections:

- [Introduction](#)
- [Core Principles](#)
- [Suggested Assessment Process](#)
- [Case Study: Part 1](#)
- [Case Study: Part 2](#)
- [Additional Considerations and Options](#)

For a detailed outline, see the [table of contents](#).

For more information, see the following books:

- [Overview of DeepSee](#), an introductory guide for all users who are interested in learning about DeepSee.
- [DeepSee Developer Tutorial](#), a tutorial for implementers who are creating DeepSee models, pivot tables, and dashboards.
- [Using the DeepSee Connector](#), a guide for implementers who are using the DeepSee Connector to import externally stored data. Note that the DeepSee Connector is available only with Ensemble.
- [Using the DeepSee Architect](#), a guide for implementers who are setting up a DeepSee model for use in the Analyzer.
- [Using the DeepSee Analyzer](#), a guide for implementers and advanced users who want to create pivot tables to embed in applications — or who simply want to explore their data.
- [Using the DeepSee Dashboard Designer](#), a guide for implementers who are using the Dashboard Designer to create dashboards.
- [Expressions and Scripts in DeepSee](#), an implementer guide that describes the syntax and options for all formulas, expressions, and scripts supported in DeepSee. This book also lists all the locations where you can use these expressions and scripts.
- [DeepSee Site Configuration and Maintenance Guide](#), a guide for implementers and system administrators. This book describes how to configure and maintain a DeepSee site. It also includes a chapter that lists common problems and their solutions.
- [DeepSee User Guide](#), a user manual for your end users. This book describes how to work with deployed dashboards and pivot tables.

For general information, see the *InterSystems Documentation Guide*.

1

Introduction

Important: This book is for DeepSee I. For information on DeepSee II, see [Getting Started with DeepSee II](#), which also lists the other books for DeepSee II.

When analysts, business users, and developers first begin to work with DeepSee, they often ask questions like the following:

- How do I choose which table to look at?
- Which fields should I use for dimensions?
- Which fields should I use for measures?
- Can I access data stored in other tables?
- Can I pre-aggregate data for use in DeepSee?

This book is meant for all users of DeepSee and is intended to give you a solid idea of the models that you can create in DeepSee. These models are the basis for all your DeepSee work.

To use this book, it is necessary to understand the basic DeepSee concepts and to be acquainted with how to use pivot tables. However, the book does not refer to syntax or to programming details.

Important: If you have not done so yet, it is suggested that you read the book [Overview of DeepSee](#).

This book emphasizes dimensions and measures much more than listing fields. In many cases, the detail listing is used primarily to identify the records; in these cases, you can simply use a few identifying fields as the listing fields. In other cases, however, the users need specific listing fields to support decision-making or other tasks, and this book cannot anticipate those uses.

2

Core Principles

This chapter describes some core principles to help you create suitable and useful DeepSee models.

2.1 Core Principles

You may encounter additional considerations, but the following list is meant as a starting point:

- Start from the business questions. The questions will guide you to the correct starting point (the base table) for a DeepSee model.
- You can create multiple DeepSee models and use them together. Depending on the business questions, you might need to do so. In some case, there is no single model that answers all the questions.
- Understand the relationships between the base table and its associated tables. These relationships determine how you can use data in the other tables. See the section “[How Table Relationships Affect Your Model Options](#)” for details.
- Find out approximately how many records are in each of the tables you are considering using.
- Consider the usability of the pivot tables and dashboards that would result from your choices. See the section “[Usability Considerations](#)” for details.
- Create only the dimensions that you need. Dimensions take up space, both in the fact table and in the indices. In the case of large base tables, it is particularly undesirable to have dimensions that are not truly necessary.

2.2 How Table Relationships Affect Your Model Options

It is very likely that each of your DeepSee models will use data in multiple tables. It is understand how the relationships among the tables affect your options in DeepSee. Note that the principles demonstrated here would apply equally in any other business intelligence tool.

2.2.1 Possible Relationships between Tables

Two tables can be related in any of the following ways:

- One to one. A row in table A corresponds to only one row in table B, and vice versa. For example, a person can have one spouse.

- One to many. A row in table A corresponds to many rows in table B, but a row in table B corresponds to only one row in table A. For example, an order consists of multiple line items, but any line item belongs to only one order.

A parent-child relationship is a type of one-to-many relationship.

- Many to one, which is the reverse of the preceding.
- Many to many. A row in table A corresponds to many rows in table B, and vice versa. For example, a person can have many doctors, and a doctor can have many patients.

The relationships can be in different forms: explicit via foreign keys or implicit by common understanding. The precise form of the relationship affects how the data is obtained, but does not affect the planning process.

2.2.2 How You Can Use Other Tables in DeepSee

The following table lists the possible relationships between your base table and a secondary table, and it describes how you can use the fields of the secondary table in DeepSee. To simplify the discussion that follows, this table also defines simple terms, shown in *italics*, to describe the secondary table in each of these scenarios.

Description of Secondary Table	Relationship of Base Table to Secondary	Use Fields in Secondary Table as Dimensions?	Use Fields in Secondary Table as Measures?
<i>Table at same level</i>	One to one	Yes.	Yes.
<i>Child table</i>	One to many	Yes, but see the third subsection .	Yes, but see the second subsection .
<i>Parent table</i>	Many to one	Yes.	No; see the first subsection .
<i>Loosely related table</i>	Many to many	Yes, but see the third subsection .	No; see the first subsection .

In principle, you can use fields in the secondary table as listing fields in any circumstance. In many cases, the detail listing is used primarily to identify the records used in any given context. In these cases, you can simply use a few identifying fields as the listing fields — and those fields are in the base table. In other cases, however, the users need specific listing fields to support decision-making or other tasks, and this book cannot anticipate those uses.

2.2.2.1 Measures from Parent Tables or Loosely Related Tables

Do not base a measure on a field in a parent table or a loosely related table. DeepSee would not aggregate that measure in the way that you intend it to. The value on which a measure is based must be one-to-one with the records in the base table.

For example, consider the following two tables:

- Order** — Each row represents an order submitted by a customer. The field `SaleTotal` represents the total monetary value of the order.
- OrderItem** — Each row represents an item in that order. In this table, the field `OrderItemSubtotal` represents the monetary value of this part of the order.

Suppose that we use `OrderItem` as the base table. Also suppose that we define the measure `Sale Total`, based on the parent's `SaleTotal` field. The goal for this measure is to display the total sale amount for all the sales of the selected order items.

Let us consider the contents of the fact table. The following shows an example:

dim A	dim B	...	Sale Total	meas B	meas C	...
nnnnnn	nnnnnn	nnnnnn	279.07	nnnnnn	nnnnnn	nnnnnn
nnnnnn	nnnnnn	nnnnnn	279.07	nnnnnn	nnnnnn	nnnnnn
nnnnnn	nnnnnn	nnnnnn	279.07	nnnnnn	nnnnnn	nnnnnn
nnnnnn	nnnnnn	nnnnnn	279.07	nnnnnn	nnnnnn	nnnnnn
nnnnnn	nnnnnn	nnnnnn	52.14	nnnnnn	nnnnnn	nnnnnn
nnnnnn	nnnnnn	nnnnnn	52.14	nnnnnn	nnnnnn	nnnnnn

The first four rows represent the items in the same order. The next two rows represent the items of another order, and so on.

Suppose that this model has a dimension called `Item Type`. Let us examine what happens when DeepSee retrieves records for all items of type R:

Item Type	dim B	...	Sale Total	meas B	meas C	...
type Q	nnnnnn	nnnnnn	279.07	nnnnnn	nnnnnn	nnnnnn
type R	nnnnnn	nnnnnn	279.07	nnnnnn	nnnnnn	nnnnnn
type R	nnnnnn	nnnnnn	279.07	nnnnnn	nnnnnn	nnnnnn
type S	nnnnnn	nnnnnn	279.07	nnnnnn	nnnnnn	nnnnnn
type R	nnnnnn	nnnnnn	52.14	nnnnnn	nnnnnn	nnnnnn
type T	nnnnnn	nnnnnn	52.14	nnnnnn	nnnnnn	nnnnnn

To compute the value of the `Sale Total` measure for type R, DeepSee adds together the three values shown here: 279.07, 279.07, and 52.14. But this action double-counts one of the orders.

Depending on the case, the `Sale Total` measure might aggregate correctly; that is, it might show the correct total sales figure for the selected order items. But you cannot ensure that the measure does this, because you cannot prevent double-counting as shown as in this example.

2.2.2.2 Measures from Child Tables

Sometimes you find a value in a child table that you would like to use as the basis of a measure. To do so, when you define the measure, you provide an expression that aggregates that value across the relevant rows of the child table.

Consider the following two tables:

- `Customer` — Each row represents a customer.
- `Order` — Each row represents a customer order. The field `SaleTotal` represents the total monetary value of the order.

Suppose that we use `Customer` as the base table, and that we want to create a measure based on the `SaleTotal` field.

Because a customer potentially has multiple orders, there are multiple values for `SaleTotal` for a given customer. To use this field as a measure, we must aggregate those values together. The most likely options are to add the values or to average them, depending on the purpose of this measure.

2.2.2.3 Dimensions from Child Tables or Loosely Related Tables

You can create dimensions based on fields in tables that have any relationship to the base table. If the table is a child table or a loosely related table, however, make sure that the users understand the behavior of the dimension. This section provides an example. It shows a child table, but the reasoning is the same for a loosely related table.

A patient can have multiple allergies. Each allergy has an allergen and a severity. Suppose that the base table is `Patients` and the model includes the `Allergen` and `Allergy Severity` dimensions. We could create a pivot table that looks like this:

	Inactive	Life-threatening	Minor	Moderate	Unable to determine
<i>Allergies</i>	Patient Count	Patient Count	Patient Count	Patient Count	Patient Count
additive/coloring agent	135	128	116	116	147
animal dander	129	139	131	124	126
ant bites	153	146	147	133	146
bee stings	138	130	107	114	115
dairy products	152	160	100	148	132
dust mites	136	127	113	120	125

Upon first seeing this pivot table, the user might think that this pivot table shows correlations between different sets of patient allergies. It does not.

This pivot table, as with all other pivot tables in this subject area, shows sets of patients. For example, the `ant bites` row represents patients who have an allergy to ant bites. The `Minor` column represents patients who have at least one allergy that is marked as minor. There are 147 patients who have an allergy to ant bites and at least one allergy that is marked as minor. This does not mean that there are 147 patients with minor allergies to ant bites.

It is possible to create a pivot table that does show correlations between different sets of patient allergies. To do so, however, you would have to define a model based on the patient allergy, rather than the patient.

2.3 Usability Considerations

You can use DeepSee in many ways, for many purposes, but the central assumption is that the end users are analysts who start by viewing high-level data, from which they drill into lower levels or detail listings, as needed. A pivot table that can be displayed on a single screen is more useful than a pivot table that requires paging down or scrolling horizontally. If a pivot table has, for example, 50000 cells, it is unlikely that the user will look at all those cells.

The number of cells in a pivot table is determined partly by the number of members of the dimensions used for rows and columns. If a given dimension has a huge number of members, the resulting pivot table can easily be too large to be useful as an analysis tool.

There are two interrelated approaches you can use to minimize the number of cells in a pivot table:

- Define and use fairly coarse-grained dimensions. A coarse-grained dimension accesses a larger number of records than a fine-grained dimension. Accordingly, a coarse-grained dimension typically has fewer members than a fine-grained dimension.
- Create a pivot table with a large number of cells, but always use this pivot table in combination with a filter that reduces the number of cells. For example, embed it in different dashboards, in each case with a filter.

Or use this pivot table as a right-click option from another pivot table. Then users can access it from a given row of that pivot table, which filters the table.

Remember also that a pivot table is interactive. Using a pivot table as a starting point, the user can drill in multiple ways for additional detail. Users can also display detail listings to see the lowest-level fields for the records used in a given context. The original pivot table does not have to include all this information.

These considerations result in the following guidelines:

- For a large base table, avoid using a field that contains unique values as a dimension. Such a field is more suitable as a listing field.
- Also avoid using fields that are nearly unique as dimensions, unless it is truly necessary to select and sort by those values.
- Wherever possible, look for coarser groupings of records that can help answer the business questions.

3

Suggested Assessment Process

A typical application includes a large number of tables, some of which contain the transactional data and others of which contain data that is more nearly static. Other tables contain lookup data. You could use any of these tables, theoretically, as the basis of a DeepSee model. And you may or may not need multiple models.

As noted in the previous chapters, when you choose a base table for a DeepSee model, you are determining what the model is analyzing. Ultimately, the question of what to analyze is a business question. That is, you should start by considering the business questions. Very likely, many of these questions are already known, and you might already have reports or other analyses that attempt to answer them. In other cases, you might have questions that have been too difficult to answer in a timely manner.

This book assumes that you have a general idea of where to start — that there is a core set of interrelated tables that you already plan to examine. Even then, however, it can be difficult to know how to proceed effectively. This chapter describes a possible assessment process that you could use to determine how to use your data. Then [chapters 4](#) and [5](#) present a case study.

Developing models is a creative process, so it is useful to recognize a central fact in creative thinking: it is difficult to create and to criticize the creation at the same time. You might find the process easier if you plan for a phase of generating ideas and exploring options, followed by a phase of reviewing, evaluating, discarding, and fine-tuning.

3.1 Collecting the Business Questions

The first step is to collect a list of questions that managers and executives ask about the enterprise. These questions typically have the following forms:

- Which are the most valuable (or least valuable) X?
- Which kinds of X are the most valuable (or least valuable)?
- How many X are late (or early)?
- Which X have the highest (or lowest) Y?
- At what time of day do we have the highest (or lowest) quantity of X?
- Which regions have the greatest (or lowest) number of X?

These questions suggest that the base table should be a table that contains one X per row. When you have a list of questions like the preceding, you have a strong indication of the table or tables to examine.

3.2 Understanding the Tables and Their Relationships

The next task is to make sure you understand the nature of the relationships among the tables that you plan to examine.

See the section “[How Table Relationships Affect Your Model Options](#)” in the previous chapter.

3.3 Determining the Table Sizes

It is important to determine how many records are in each of the tables that you might use. Understanding the volume is key to guiding you towards choosing dimensions that aggregate at a high-enough level to be useful.

3.4 Evaluating a Possible Base Table

You can start with any table. The tables with the most records are often a good starting point. In this step, you make several passes, looking at the table from different points of view.

1. First look at each field of the table and consider how it could be used in DeepSee (if at all). Consider whether the field could be used as a dimension, a measure, or a listing field (or some combination of uses). For each field, ask the following basic questions:

- How many distinct values are possible for this field?
- Does this field uniquely identify a record in this table, or does it do so in conjunction with other fields?

In either case, if you define a dimension on this field, the dimension will have a very large number of members, and there will be a huge number of rows or columns in any pivot table that uses it.

Such a field is more suitable for use as a listing field. A low-level dimension is not generally recommended.

- Is the field value unique? Such fields are not generally useful as dimensions.
- Is the field numeric, and if so, can it be treated meaningfully as a number?

Identifiers are often numeric but are treated as strings. You would not add IDs together.

If the field is a number, consider using it as a measure. Also consider the best way to aggregate values for this field. By default, DeepSee sums the numbers, but other options are available: average, minimum, maximum, mean, and median.

(You might also consider using the field as a dimension; this is discussed later in this chapter.)

- Does the field contain string values, particularly string values from a fixed set (for example, status values or city names)?

If so, it might be useful as a dimension, if it does not have too many distinct values. A dimension with large numbers of members does not provide a high-level view.

- Does the field contain date, date/time, or time values? If so, it might be useful as a dimension.

When you use a date or date/time value as a dimension, DeepSee automatically generates a set of variations that use different parts of the date value. For example, the Year variation uses only the year part of the date, the Quarter variation uses only the quarter part of the date, and so on. You can use these dimensions individually or in combination.

(You might also consider using the field in a measure; this is discussed later in this chapter.)

When you use a time value as a dimension, you can have a dimension member for a member for every hour of the day, or you can define your own time ranges.

For a more detailed look at date and time data in DeepSee, see the chapter “[Date and Time Data](#).”

- How accurate is the data in this field? If it is inaccurate in many cases, the field is harder to use for analysis.
- Does the field contain free text? If so, the field might not contain data that is reliable enough for your analytical needs.
- Does the field point (directly or indirectly) to another table that represents a coarser granularity? If so, you might want to ignore this field and instead use fields in the other table. See step 3.
- Does the field contain a list of data elements? If so, the field is essentially an additional table, with finer granularity than the base table. See step 3.

In many cases, multiple fields are related to each other, and you may not need to use them all as dimensions. This is true of geographical data, where a country contains postal codes, which contain cities, and so on, and each of these elements is stored in a separate field. For an initial look at the data, you might meet your needs by using only the highest-level of these fields (country). Later, if you determine that you want to see a more detailed breakout by geography, you can add dimensions for postal code, city, and so on.

2. Next try to find whether it is possible to compute meaningful, useful values from combinations of the fields. Consider the following additional questions:
 - Does the base table contain fields that can be combined with other fields to obtain a useful value? This question applies particularly to numeric fields. For example, it is common practice to record the number of units and the unit price in separate fields.
 - Would it be useful to define additional measures that aggregate in different ways? DeepSee aggregates by summing, by default, but you can also aggregate by obtaining the average, median, minimum, or maximum value.
 - Does the base table contain non-numeric values that can be aggregated? This question applies particularly to date fields. You can aggregate dates by selecting the earliest or the last date.
3. Look at other tables that have a relationship with the base table. See the section “[How Table Relationships Affect Your Model Options](#)” in the previous chapter.

Determine which fields to use for dimensions, measures, or (less often) listing fields.

4. Next look through all the tables for fields or values that could usefully be placed into bins, to be used as dimensions. You can define custom bins for any type of value, especially numeric values, date values, and time values.
5. After you have chosen all the dimensions, consider whether you want to create measures that count dimension members. For any dimension, you can create a measure that indicates the number of distinct members that are used in any context. For example, `Customer` is a dimension in the `Transactions` model, discussed in the case study later in this book. Any set of transactions involves some number of customers. The measure `Customer Count` gives us that number.
6. Review the model elements and consider whether all are needed. Each dimension is indexed separately and thus consumes disk space. Depending on how a measure is defined, it might also be indexed.

For each dimension, consider the following questions:

- Is it useful to select items by this dimension? For example, if the dimension is at a low level, the pivot table will have a comparatively large number of rows or columns and will thus not serve as a summary.
- If there are related dimensions, do you need all of them?

For example, in the case study later in this book, we have the dimensions `Broker Office` and `Broker Country`. If a given country has only one or two broker offices, you might want to eliminate one of these dimensions, because there is little difference in how they behave.

- If this is a date-type dimension, do you have another date-type dimension that is more useful? For example, your database might store beginning and ending dates. In some contexts, you might decide that only the ending date (or only the beginning date) is useful.
- Will it be clear to users what this dimension does?

For each measure, consider the following questions:

- Is this measure really useful?
- Does this measure represent a quantity that is already understood by your users?
- How does this measure behave in different contexts? Is it useful in all contexts? If not, should it be replaced with a measure that is aggregated by averaging? Or should it be eliminated?

7. Identify any measures that would be useful as KPIs (either alone or in combination with each other). For each KPI, consider the following questions:

- How do you plan to use this KPI? That is, will you apply filters to it?

By default, a KPI is aggregated across all the records of the subject area. When you include a KPI on a dashboard, however, you can apply filters to it, so that it is filtered in the same way as other elements displayed on the dashboard.

- What values does this KPI have at different aggregation levels? Are those values instantly meaningful to the users?
- Is it necessary to instead define a KPI that is normalized by a reference value?

For example, the KPI `Sales` is not normalized. This KPI is hard to interpret at lower levels; users probably know good values for this KPI for the entire organization, but probably cannot recognize good values for this KPI at the item level. Depending upon the users, this KPI might not be suitable except at the highest level.

The KPI `Sales Percent Change` is normalized and can be interpreted more easily at all levels.

If needed, reexamine the tables again for elements that could be used to construct additional KPIs.

3.5 Evaluating Other Possible Base Tables

After performing the preceding exercise with one table, repeat it with each of the related tables, starting with the most closely related tables and working outward. As you do this, you place a different entity into focus and you consider different questions.

As you examine any table for use as a model basis, always remember that the objects counted in this model are rows in this table.

3.6 Comparing the Models

After planning the separate models, review them side by side to check for consistency, identify any gaps, and remove unneeded elements. You might also need to rename some elements (particularly measures) to clarify their behavior.

Then look at each model and list what it can do that the other models cannot do.

3.7 Using the Models in Combination

You can use multiple DeepSee models together, even though they use different units of analysis. In particular:

- You can define a pivot table that links multiple pivot tables together. These pivot tables can be based on different models.
- You can define dashboards that display pivot tables and KPIs from different models. The dashboard can include filters that apply to these elements, even though the elements come from different models.

In these situations, the key to getting the correct behavior is to ensure that you define dimensions in a consistent way in all the models, wherever applicable. Specifically, the dimension name and the member names must be the same in all the models that contain them. (This rule does not present a problem for your pivot tables; you can customize dimension and member names in the pivot tables themselves as needed.)

4

Case Study: Part 1

This chapter and the following chapter are a case study that examines the core tables in an application that manages customer transactions. Customers buy and sell assets, and each transaction is recorded, along with the number of units sold or bought and the unit price at the time of the transaction. The case study examines how we might use these tables in DeepSee.

4.1 Overview of the Tables

This section introduces the central tables of this example application.

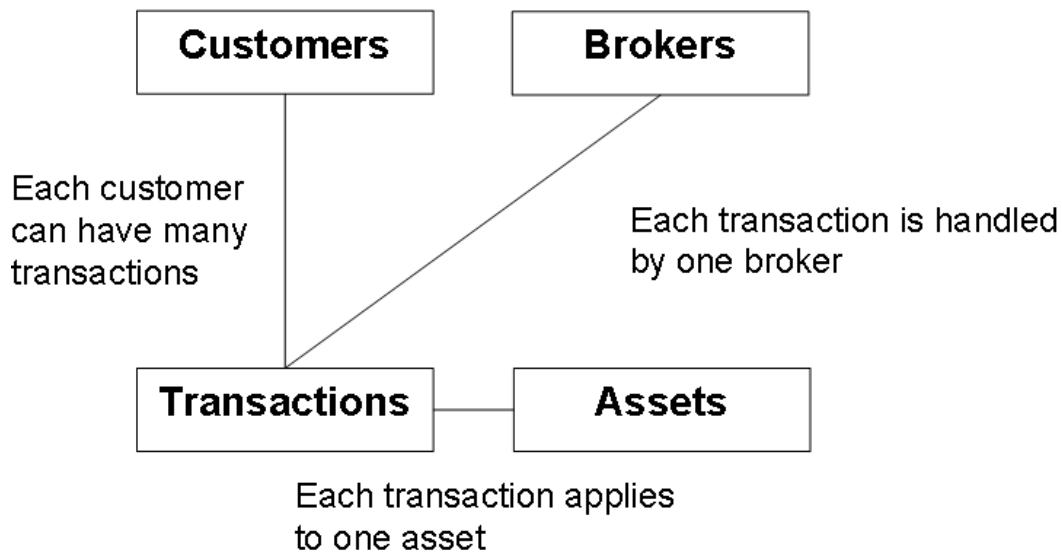
The `Customers` table contains the customer information: customer IDs, contact information, and so on. In this example, there are hundreds of thousands of customers.

The `Brokers` table contains the brokers; each broker works out of a specific office, and the offices are in different countries. There are many dozens of brokers.

The `Assets` table lists the assets that the customers buy and sell. There are different types of assets. The table contains hundreds of assets.

Finally, the `Transactions` table lists the transactions made by the customers. Each transaction applies to a specific asset, is of type Buy or Sell, and records the number of units traded, as well as the unit price. This table also records the broker who handled each transaction. There are millions of transactions in this table.

The following figure shows these tables and their connections:



The following subsections describe the fields in these tables. Except where noted, these fields contain strings.

4.1.1 Customers

The `Customers` table has the following fields:

- `CustomerID` — Unique identifier for the customer in this system.
- `Name` — Name of the customer.
- `BirthYear` — Birth year of the customer. This field is numeric.
- `StreetAddress` — Street address of the customer.
- `City` — City of the customer.
- `PostalCode` — Postal code of the customer.
- `Country` — Country of the customer.
- `PhoneNumber` — Phone number of the customer.
- `SecondaryPhoneNumber` — Secondary phone number of the customer. Any additional contact information is written to the `Comments` field.
- `CustomerClass` — Class into which this customer has been placed by the broker who most recently updated the customer's record. The brokers sometimes use this to flag the more important customers. There are three customer classes, A, B, and C.
- `HouseholdIncome` — Household income of the customer. This field is numeric.
- `Comments` — A free-text field that contains any additional information entered by the brokers.

4.1.2 Brokers

The `Brokers` table has the following fields:

- `BrokerID` — Unique identifier for the broker in this system.
- `Name` — Name of the broker.
- `OfficeID` — Unique identifier for the office from which this broker works.

- `OfficeName` — Name for the office.
- `StreetAddress` — Street address of the office.
- `PostalCode` — Postal code of the office.
- `City` — City of the office.
- `PostalCode` — Postal code of the office.
- `Country` — Country of the office.

4.1.3 Assets

The `Assets` table has the following fields:

- `AssetID` — Unique identifier for the asset in this system.
- `Name` — Name of the asset.
- `Vendor` — Vendor who provides this asset. There are only a few vendors.
- `Type` — Type of this asset. There are only a few asset types.

4.1.4 Transactions

The `Transactions` table has the following fields:

- `TransactionID` — Unique identifier for this transaction.
- `Comments` — A free-text field that contains comments entered by the broker.
- `Customer` — Customer who performed this transaction.
- `Asset` — Asset that was bought or sold in this transaction.
- `Type` — Either Buy or Sell.
- `UnitPrice` — The price per unit for the asset, as paid in this transaction. This field is numeric.
- `UnitCount` — The number of units traded in this transaction. This field is numeric.
- `Status` — Current status of the transaction, either Pending or Completed.
- `StartTimeStamp` — Date and time when this transaction was initiated. This field is a date/time field.
- `EndTimeStamp` — Date and time when this transaction was completed. This field is a date/time field. If the transaction is pending, this field is empty.
- `Broker` — Broker who managed this transaction.
- `BrokerFee` — Commission paid to this broker for this transaction. This field is numeric.

4.2 Transactions As the Base Table

There are four tables, which gives us four possible starting places. For the purposes of simplicity, this case study will examine only two of these tables as starting places.

The `Transactions` table has the largest number of records and would thus be interesting to use as the base table of a DeepSee model. This rest of this chapter explores what DeepSee can do if we use this table as the base table.

If we use `Transactions` as the base table in a DeepSee model, the default measure is the count of transactions. The pivot tables will display aggregate data for transactions, grouped and sorted by properties that derive directly or indirectly from transactions. The following shows a pivot table that we will be able to create:

	Buy	Sell
<i>Completed Date Period</i>	Transaction Count	Transaction Count
2009-06	nnnnnnnn	nnnnnnnn
2009-05	nnnnnnnn	nnnnnnnn
2009-04	nnnnnnnn	nnnnnnnn
...	nnnnnnnn	nnnnnnnn

This pivot table groups the transactions into sets defined by completion date period (used as rows) and the type (Buy or Sell, used as columns). Each data cell in this pivot table displays an aggregate value that applies to a set of transactions. In this case, the aggregate value displayed is the default measure, which is `Transaction Count`.

4.3 A First Look at the Base Table

In our first look, we examine the fields of the `Transactions` table and ask the basic questions listed in step 1 of “[Evaluating a Table](#).” Ignore other tables for now.

The `Customer` field is interesting. It seems that it would be nice to be able to group transactions by customer. However, there are hundreds of thousands of customers, so we will ignore this field. Later when we examine other tables, we will look for ways to group transactions by customer at a coarser granularity (that is, according to some shared customer attribute rather than directly by customer).

Similarly, we will ignore the `Asset` field; there are too many different assets.

We make the following preliminary decisions about the DeepSee model. Unless otherwise noted, the element name will be the same as the field name.

- Use as dimensions:
 - `Type`
 - `Status` — Because the word “status” could be misinterpreted, we will call this dimension `Transaction Status`.
 - `StartTimeStamp` — We will ignore the time part of the time stamp. We will create a date-type measure and call it `Initiated Date`. DeepSee will create a set of variations that include `Initiated Date Year`, `Initiated Date Quarter`, and so on. A complete list is in the section “[Date and Time Data](#).”
 - `EndTimeStamp` — We will ignore the time part of the time stamp. We will create a date-type measure and call it `Completed Date`. DeepSee automatically creates a set of variations as it does for `Initiated Date`.
 - `Broker` — There are dozens of brokers, so this might not be a useful dimension, but we’ll create it for now.
- Use as measures:
 - `Transaction Count` — This is the default measure created by DeepSee.
 - `UnitCount` — We will call this measure `Unit Count`. This gives us the number of units bought or sold.
 - `BrokerFee` — We will call this measure `Broker Fee`.

- Use as listing fields:
 - TransactionID — We will call this listing field Transaction ID.
 - Customer
 - Asset
 - Type
 - Status
 - StartTimeStamp — We will call this listing field Initiated Date.
 - EndTimeStamp — We will call this listing field Completed Date.
- Ignore the other fields for now.

To make this model more concrete, let us consider a possible pivot table that we could create from it:

		Buy		Sell	
<i>Completed Date Year</i>	<i>Completed Date Quarter</i>	<i>Transaction Count</i>	<i>Unit Count</i>	<i>Transaction Count</i>	<i>Unit Count</i>
2009	Q2	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
	Q1	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
2008	Q4	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
	Q3	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
	Q2	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
	Q1	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
2007	Q4	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
	Q3	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
	Q2	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
	Q1	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
...		nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn

This pivot table shows the number of transactions and of units traded, by year and quarter when completed.

4.4 Additional, More Complex Measures

In the previous section, we considered numeric measures that we could create, based directly on fields in the base class. The next step is to consider the additional questions listed in step 2 of “[Evaluating a Table](#).” We review the Transactions table again and add the following measures:

- Average Broker Fee — This measure uses the BrokerFee field, but aggregates by averaging rather than summing.
- Net Transaction Value — This measure is defined by the following logic:
 - If Type is Buy: UnitPrice * UnitCount
 - If Type is Sell: -1 * UnitPrice * UnitCount
- Average Transaction Value — This measure is defined using the preceding logic but is aggregated by averaging rather than summing.
- Average Transaction Duration — This measure computes the difference between the initiation and the completion of a transaction. It is aggregated by averaging.

For additional possible measures, see “[Date Measures](#),” later in this book.

Now we can create pivot tables like the following:

Broker	2009			...
	Average Transaction Value	Average Broker Fee	Average Transaction Duration	
Broker A	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
Broker B	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
Broker C	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
...	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn

Each row in this pivot table shows the transactions for one broker. For example, the cell in the 2009>Average Transaction Value column shows the average value, per transaction, for that broker in 2009.

4.5 Using Related Tables

After considering the information available in the base table, consider the fields in related tables, as described in step 3 of “[Evaluating a Table](#).”

There are three tables related to the base table: `Customers`, `Brokers`, and `Assets`. These tables are all parent tables to `Transactions`, so we will examine the fields in them for possible use as dimensions. Fields in these tables would not be suitable as measures.

4.5.1 Dimensions Based on Customer Information

For each transaction, there is a customer with additional data in the `Customers` table. We can access the following fields and use them as dimensions:

- `Country` — The `Country` field might be useful as a dimension. It would enable us to group transactions by the customers’ home countries.

If geographical breakdowns are particularly interesting, we could also use `PostalCode`, `City`, and `PostalCode` as dimensions. We will ignore these for this example. It is sensible to start with the coarsest-grained dimension and add finer-grained dimensions later if needed.

- `CustomerClass` — We will call this dimension `Customer Class`.

The fields `BirthYear` and `HouseholdIncome` are numeric and are thus continuous or nearly so. This means that if we use them as dimensions, we could potentially have an undesirably large number of members. [Later](#) we will consider another way to use these fields as dimensions.

The `StreetAddress`, `PhoneNumber`, and `Secondary` fields are fairly low level; it is unlikely that these would be useful as dimensions in any model.

4.5.2 Dimensions Based on Asset Information

For each transaction, there is an asset with additional data in the `Assets` table.

In the `Assets` table, both `Vendor` and `Type` would be useful for grouping the transactions. We can use these fields as dimensions. Because the word “type” is fairly generic, we call this dimension `Asset Type`.

4.5.3 Dimensions Based on Broker Information

For each transaction, there is a broker with additional data in the `Brokers` table.

In the `Brokers` table, the `Office Name` field would be useful for grouping, so we use this as the `Broker Office` dimension.

We can also add dimensions based on the broker’s office. For this example, let us add the `Broker Country` dimension.

If geographical breakdowns are particularly interesting, we could also use the offices’ city and postal code as dimensions. We will ignore these fields for this example.

4.5.4 Dimensions to Add

Based on the preceding discussion, we add the following dimensions:

- `Country`
- `Customer Class`
- `Vendor`
- `Asset Type`
- `Broker Office`
- `Broker Country`

4.5.5 Possible Pivot Tables

Now we can create pivot tables like the following:

	Vendor A		Vendor B		...
<i>Asset Type</i>	Unit Count	Net Transaction Value	Unit Count	Net Transaction Value	
Type A	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
Type B	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
Type C	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
...	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn

This pivot table looks at the transactions broken out by asset vendor and asset type. It shows the number of units traded and the net value of the trades, across all transactions in the given cell. By default, this aggregation is performed across all dates, but you could add a further breakout by date or you could filter the pivot table to show only a particular span of dates.

For example, we could rearrange the previous pivot table and add the year as follows:

Asset Type	Vendor	2009		2008		...
		Unit Count	Net Transaction Value	Unit Count	Net Transaction Value	
Type A	Vendor A	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
	Vendor B	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
	...	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
Type B	Vendor A	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
	Vendor B	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
	...	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
Type C	Vendor A	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
	Vendor B	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
	...	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn

4.6 Additional Dimensions Based on Bins or Ranges

In this step, we look at all the related tables for any values that could be placed into bins.

We can define custom bins for any type of value, especially numeric values, date values, and time values.

4.6.1 Dimensions to Add

It would be useful to categorize transactions by customer income, but you would not necessarily want to see the exact incomes. So we add a dimension called `Customer Household Income`, with the following members:

- Less than \$100000
- \$100000 to \$250000
- \$250000 to \$500000
- More than \$500000

Similarly, we add a dimension called `Birth Year`, which consists of ten-year bins.

4.6.2 Possible Pivot Table

This pivot table shows the transaction count, the number of units traded, and the average transaction value, broken out by customer household income.

Household Income	Transaction Count	Unit Count	Average Transaction Value
Less than \$100,000	nnnnnnnn	nnnnnnnn	nnnnnnnn
\$100,000 to \$249,000	nnnnnnnn	nnnnnnnn	nnnnnnnn
\$250,000 to \$499,000	nnnnnnnn	nnnnnnnn	nnnnnnnn
More than \$500,000	nnnnnnnn	nnnnnnnn	nnnnnnnn

This pivot table looks at all dates, but you could add a further breakout by date or you could filter the pivot table to show only a particular span of dates.

4.7 Additional Measures Based on Dimension Member Counts

After you have chosen all the dimensions, it is worthwhile to consider whether you want to create measures that count dimension members. For any dimension, you can create a measure that indicates the number of distinct members that are used in any context. For example, `Customer` is a dimension. Any set of transactions involves some number of unique customers. The measure `Customer Count` gives us that number.

4.7.1 Measures to Add

We will add the following measures:

- `Customer Count` — Counts the customers involved in a set of transactions.
- `Broker Count` — Counts the brokers involved in a set of transactions.
- `Asset Type Count` — Counts the asset types bought or sold in a set of transactions.

4.7.2 Possible Pivot Table

Now we can create pivot tables like the following:

<i>Customer Country</i>	<i>Customer Count</i>	<i>Net Transaction Value</i>
Country A	<i>nnnnnnnn</i>	<i>nnnnnnnn</i>
Country B	<i>nnnnnnnn</i>	<i>nnnnnnnn</i>
Country C	<i>nnnnnnnn</i>	<i>nnnnnnnn</i>
...	<i>nnnnnnnn</i>	<i>nnnnnnnn</i>

This pivot table tells us for each country, the number of customers based in that country and the net value of their transactions.

4.8 Model Review

After selecting the model elements, it is worthwhile to review the set, as described in the step 6 of “[Evaluating a Table](#).” In this step, we consider whether all the elements are useful and needed. During this review, we conclude the following:

- The dimensions `Broker Office` and `Broker Country` are related, but a given country has only one or two broker offices, so there is little difference in how these dimensions behave. For our example, we remove the `Broker Country` dimension.
- The `Broker Count` measure counts the brokers involved in a set of transactions. It is not clear if this is really useful, so we remove this measure.
- The `Asset Type Count` measure counts the asset types bought or sold in a set of transactions. For our example, our users are not accustomed to considering how many asset types are involved, so we remove this measure.

4.9 The Final Model (Transactions)

In summary, when we use transactions as the base table, we can create a model that consists of the following:

- Dimensions:
 - Type — Groups the transactions by type: Buy or Sell.
 - Transaction Status — Groups the transactions by transaction status.
 - Completed Date — Groups the transactions by the date on which they were completed.
 - Initiated Date — Groups the transactions by the date on which they were started.
 - Customer Country — Groups the transactions by the customer's home country.
 - Customer Class — Groups the transactions by customer class.
 - Asset Type — Groups the transactions by asset type.
 - Vendor — Groups the transactions by asset vendor.
 - Broker — Groups the transactions by the broker who performed them.
 - Broker Office — Groups the transactions by the offices of the brokers who performed them.
 - Customer Household Income — Groups the transactions by customer household income bracket.
 - Birth Year — Groups the transactions by the customer's birth year.
- Measures:
 - Transaction Count — Counts the transactions.
 - Unit Count — Counts the units traded in a set of transactions.
 - Broker Fee — Shows the total amount of the broker fees for a set of transactions.
 - Average Broker Fee — Shows the average amount, per transaction, of the broker fees for a set of transactions.
 - Net Transaction Value — Shows the net value of the transactions, from the point of view of the asset vendors.
 - Average Transaction Value — Shows the average value of the transactions.
 - Customer Count — Counts the distinct customers involved in a set of transactions.
 - Average Transaction Duration — Shows the average length of time between the start and the completion of a transaction.
- Listing fields:
 - Transaction ID
 - Customer
 - Asset
 - Type
 - Status
 - Initiated Date
 - Completed Date

5

Case Study: Part 2

This chapter explores what DeepSee can do if we use customers as the base class, instead of transactions as in the previous chapter. In this scenario, we will see many of the same dimensions and measures. The measures will behave differently because this model uses a different base table.

5.1 Customers As the Base Table

If we use `Customers` as the base table in a DeepSee model, the pivot tables will display aggregate data for customers, grouped and sorted by properties that derive directly or indirectly from customers. In this model, the default measure is `Customer Count`.

5.1.1 Using the Fields of the Customers Table

Let us examine the fields of `Customers` and make the following preliminary decisions about this DeepSee model. Unless otherwise noted, the element name will be the same as the field name.

- Use as dimensions:
 - `BirthYear` — We will use this to define a dimension called `Birth Year`, which will consist of ten-year bins.
 - `Country`
 - `CustomerClass` — We will call this dimension `Class`.
 - `HouseholdIncome` — We will call this dimension `Household Income`, and this dimension will consist of bins as in the example in “[Dimensions Based on Binning Numeric Data](#).”
- Use as measures:
 - `HouseholdIncome` — We will define two measures based on this field: `Total Household Income` and `Average Household Income`.
- Use as listing fields:
 - `CustomerID` — We will call this listing field `Customer ID`.
 - `Name`
 - `Country`
- Ignore the other fields for now.

Now we can create pivot tables like the following:

<i>Customer Class</i>	<i>Average Household Income</i>
Class A	nnnnnnnn
Class B	nnnnnnnn
Class C	nnnnnnnn

5.1.2 Using Related Tables

After considering the information available in the `Customers` table, it is worthwhile to consider fields in the related tables:

- `Transactions` — This is the most closely related table, and it is a child table to the `Customers` table. Fields in this table might be suitable as dimensions.

Also, if we aggregate data across all the transactions for a customer, we can use that value as the basis of a measure.

- `Asset` — Each transaction of a customer has an asset. This table is loosely related to `Customers`; that is, there is a many-to-many relationship between the tables. In this model, we can use asset data as dimensions but not as measures.
- `Brokers` — Each transaction of a customer is performed by a broker. This table is loosely related to `Customers`. In this model, we can use broker data as dimensions but not as measures.

The following sections discuss the details.

5.1.3 Additional Dimensions from Related Tables

For each customer, we can access all the transactions for that customer, from the `Transactions` table. We can use transaction-related data as dimensions. We might add the following dimensions:

- `Broker` — Groups customers according to the brokers who have performed their transactions.
- `Type` — Groups the customers by the types of transactions they have had.
- `Completed Date` — Groups the customers by the completion dates of their transactions.
- `Initiated Date` — Groups the customers by the initiation dates of their transactions.

We could add other dimensions based on transactions, but it is less clear they would be useful.

We can also access details for the assets that a customer has traded. These details are in the `Assets` table. We can add the following dimensions:

- `Asset Type` — Groups customers according to the asset types that they have bought or sold.
- `Vendor` — Groups customers according to the vendors whose assets they have bought or sold.

Finally, we can access details for the broker of the transaction. These details are in the `Brokers` table. We can add the following dimension:

- `Broker Office` — Groups customers according to the offices of the brokers who have performed their transactions.

Now we can create pivot tables like the following:

<i>Asset Type</i>	<i>Customer Count</i>	<i>Average Household Income</i>	<i>...</i>
Asset Type A	nnnnnnnn	nnnnnnnn	nnnnnnnn
Asset Type B	nnnnnnnn	nnnnnnnn	nnnnnnnn
Asset Type C	nnnnnnnn	nnnnnnnn	nnnnnnnn
...	nnnnnnnn	nnnnnnnn	nnnnnnnn

This pivot table shows customers grouped by the types of assets that they buy and sell. For each asset type, the pivot table shows the number of customers who buy and sell that asset type and their average household income.

5.1.4 Additional Measures from the Transactions Table

A customer potentially has multiple transactions, and we have already identified some useful measures based on the transaction data. We can use the same data in this model. For each customer, we will combine the customer's transactions and define the following measures:

- **Total Transaction Value** — The sum of the values of the all transactions for a given customer.
- **Average Transaction Value** — The average of Total Transaction Value per customer. For a given customer, this is identical to Total Transaction Value.
- **Transaction Count** — The number of transactions recorded for a given customer.
- **Average Transaction Count** — The average of Transaction Count per customer. For a given customer, this is identical to Transaction Count.
- **Unit Count** — Total number of units that this customer has bought or sold.
- **Broker Fee** — Total of all broker fees paid by this customer.
- **Average Broker Fee** — The average of Broker Fee per customer. For a given customer, this is identical to Broker Fee.

In all these cases, we are aggregating the selected value across all transactions of a given customer. This example represents a use of pre-aggregated data within a DeepSee model.

We could also create measures that show the earliest and latest transaction dates for a customer, as well as the average duration of all transactions of a customer. We omit these measures in this exercise for simplicity.

Now we can create pivot tables like the following:

		<i>Asset Type A</i>	<i>Asset Type B</i>	<i>Asset Type C</i>	<i>...</i>
<i>Customer Class</i>	<i>Household Income</i>	<i>Average Transaction Value</i>	<i>Average Transaction Value</i>	<i>Average Transaction Value</i>	
Class A	\$250,000 to \$499,000	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
	More than \$500,000	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
Class B	\$100,000 to \$249,000	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
	\$250,000 to \$499,000	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
Class C	Less than \$100,000	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn
	\$100,000 to \$249,000	nnnnnnnn	nnnnnnnn	nnnnnnnn	nnnnnnnn

5.1.5 Additional Measures Based on Dimension Member Counts

After you have chosen all the dimensions, it is worthwhile to consider whether you want to create measures that count dimension members. In this example, no such measure seems immediately useful.

5.1.6 Model Review

As we did with transactions, we review the model elements that we have chosen so far to evaluate whether we need all of them.

In this phase, we decide that we are more interested in the customers' countries than in any smaller geographic breakdown, so we remove the `PostalCode` dimension.

5.1.7 The Final Model (Customers)

In summary, when we use customers as the base table, we can create a model that consists of the following:

- Dimensions:
 - `Birth Year` — Groups the customers by birth year, in ten-year buckets.
 - `Country` — Groups the customers by state or province.
 - `Class` — Groups the customers by class.
 - `Household Income` — Groups the customers by household income.
 - `Broker` — Groups customers according to the brokers who have performed their transactions.
 - `Asset Type` — Groups customers according to the asset types that they have bought or sold.
 - `Vendor` — Groups customers according to the vendors whose assets they have bought or sold.
 - `Broker Office` — Groups customers according to the offices of the brokers who have performed their transactions.
- Measures:
 - `Customer Count` — Counts the customers.
 - `Total Household Income` — Shows the total household income for any set of customers.
 - `Average Household Income` — Shows the average household income for any set of customers.
 - `Total Transaction Value` — The sum of the values of the all transactions for any set of customers.
 - `Average Transaction Value` — The average of `Total Transaction Value` per customer, for any set of customers.
 - `Transaction Count` — The number of transactions recorded for any set of customer.
 - `Average Transaction Count` — The average of `Transaction Count` per customer, for any set of customers.
 - `Unit Count` — Total number of units bought or sold by any set of customers.
 - `Broker Fee` — Total of all broker fees paid by any set of customers.
 - `Average Broker Fee` — The average of `Broker Fee` per customer, for any set of customers.
- Listing fields:

- Customer ID
- Name
- Country

5.2 Comparing the Models

Now that we have planned two models, let us compare them to each other.

5.2.1 Dimensions

First, the following table shows the dimensions defined in the models. Dimensions that behave similarly are shown side by side; for clarity, we will rename dimensions in such cases to use the same name, as listed in the Notes column.

Table 5–1: Dimensions in the Transaction and Customer Models

Transaction	Customer	Notes
Customer Country	Country	In both models, use the name Customer Country.
Customer Class	Class	In both models, use the name Customer Class.
Customer Household Income	Household Income	In both models, use the name Household Income.
Asset Type	Asset Type	
Vendor	Vendor	
Type		
Transaction Status		
Initiated Date		
Completed Date		
Broker	Broker	
Broker Office	Broker Office	Rename both of these dimensions to Office.
Birth Year	Birth Year	

Remember that each dimension in the Transactions model selects transactions, and each dimension in the Customers model selects customers. For example, in the Transactions model, broker office A selects all transactions handled by broker office A. In the Customers model, broker office A selects all customers who have had a least one transaction handled by broker office A.

5.2.2 Measures

The measures in the two models are shown in the following table. Measures that behave the same are shown side by side; for clarity, we will rename measures in such cases to use the same name, as listed in the Notes column.

For measures that perform averages, we will append (Customers) or (Transactions) to the end of the measure name. This convention ensures that users know the context in which this average is computed.

Table 5–2: Measures in the Transaction and Customer Models

Transaction	Customer	Proposed New Name
Transaction Count	Transaction Count	
	Average Transaction Count	Average Transaction Count (Customers)
Unit Count	Unit Count	
Broker Fee	Broker Fee	
Average Broker Fee		Average Broker Fee (Transactions)
	Average Broker Fee	Average Broker Fee (Customers).
Net Transaction Value	Total Transaction Value	Total Transaction Value
Average Transaction Value		Average Transaction Value (Transactions)
	Average Transaction Value	Average Transaction Value (Customers)
Earliest Transaction Date		
Latest Transaction Date		
Average Transaction Duration		Average Transaction Duration (Transactions)
Customer Count	Customer Count	
	Total Household Income	
	Average Household Income	

5.2.3 Comparison Summary

The Transactions model has the following features that are not present in the Customers model:

- Ability to create sets of transactions, which can be correlated and compared
- Ability to show averages per transaction

The Customers model has the following features that are not present in the Transactions model:

- Ability to create sets of customers, which can be correlated and compared
- Ability to show averages per customer

Both models have some dimensions based on transaction properties. The Transactions model has more of such dimensions, including `Initiated Date` and `Completed Date`. We could add these dimensions to the Customers model, but it is less clear if they are useful in that model; see the discussion in “[Dimensions from Child Tables or Loosely Related Tables.](#)”

5.3 Using the Models in Combination

In DeepSee, you can create pivot tables and dashboards that combine elements from multiple models. (For details, see the [DeepSee Developer Tutorial](#).)

We could use the Transactions model and the Customers model together to create pivot tables like this:

<i>Asset Type</i>	<i>Transaction Count</i>	<i>Average Transaction Value (Transactions)</i>	<i>Average Transaction Value (Customers)</i>
Type A	nnnnnnnn	nnnnnnnn	nnnnnnnn
Type B	nnnnnnnn	nnnnnnnn	nnnnnnnn
Type C	nnnnnnnn	nnnnnnnn	nnnnnnnn
...	nnnnnnnn	nnnnnnnn	nnnnnnnn

This pivot table has a row for each asset type that shows the following:

- Number of transactions in which an asset of this type was bought or sold
- Average value, per transaction, of those transactions. For a given asset type, if there were 100 transactions, their cumulative value is divided by 100.
- Average value, per customer, of those transactions. For a given asset type, if there were 2500 customers, their cumulative value is divided by 2500.

5.4 Final Comments

In the case study shown here, you would need a DeepSee model built on Transactions as the base table in the following cases:

- If you needed to be able to select transactions by dates or other transaction-specific dimension
- If you needed to be able to correlate transactions in different ways
- If you needed per-transaction averages

If you do not need any of those features, you would create a model based on Customers. Because this table has many fewer rows than Transactions, less disk space would be needed for the DeepSee indices.

6

Additional Considerations and Options

This chapter discusses additional issues to consider when planning your DeepSee models.

- [Null values](#)
- [Dates and time data](#)
- [Compound members, which are built on existing members](#)

6.1 Null Values

It is important to be aware of how null values can affect DeepSee pivot tables, so that you can choose how to handle them in your models. For these questions, there are no right and wrong answers. The answers you choose depend upon your business needs and your users' needs.

6.1.1 Null Values in Dimensions

When DeepSee builds the indices, it ignores any null values by default. For example, suppose that your base table is `PatientEncounter` and that you have a dimension based on the encounter type. If a given encounter does not have a recorded value for encounter type, DeepSee ignores this encounter, when determining how to categorize it for this dimension. The fact table looks like this:

Encounter Type	dim B	...	meas A	meas B	...
inpatient	nnnnnn	nnnnnn	nnnnnn	nnnnnn	nnnnnn
inpatient	nnnnnn	nnnnnn	nnnnnn	nnnnnn	nnnnnn
	nnnnnn	nnnnnn	nnnnnn	nnnnnn	nnnnnn
outpatient	nnnnnn	nnnnnn	nnnnnn	nnnnnn	nnnnnn
outpatient	nnnnnn	nnnnnn	nnnnnn	nnnnnn	nnnnnn
outpatient	nnnnnn	nnnnnn	nnnnnn	nnnnnn	nnnnnn
inpatient	nnnnnn	nnnnnn	nnnnnn	nnnnnn	nnnnnn
inpatient	nnnnnn	nnnnnn	nnnnnn	nnnnnn	nnnnnn
inpatient	nnnnnn	nnnnnn	nnnnnn	nnnnnn	nnnnnn
inpatient	nnnnnn	nnnnnn	nnnnnn	nnnnnn	nnnnnn

The fact table does include the record, but there is no value for the `Encounter Type` dimension, and DeepSee ignores this record when it uses this dimension.

Suppose that dimension B is the facility where the encounter occurred. The facilities are A and B.

Encounter Type	Facility	...	meas A	meas B	...
inpatient	A	nnnnnn	nnnnnn	nnnnnn	nnnnnn
inpatient	A	nnnnnn	nnnnnn	nnnnnn	nnnnnn
	A	nnnnnn	nnnnnn	nnnnnn	nnnnnn
outpatient	B	nnnnnn	nnnnnn	nnnnnn	nnnnnn
outpatient	B	nnnnnn	nnnnnn	nnnnnn	nnnnnn
outpatient	B	nnnnnn	nnnnnn	nnnnnn	nnnnnn
inpatient	B	nnnnnn	nnnnnn	nnnnnn	nnnnnn
inpatient	A	nnnnnn	nnnnnn	nnnnnn	nnnnnn
inpatient	A	nnnnnn	nnnnnn	nnnnnn	nnnnnn
inpatient	A	nnnnnn	nnnnnn	nnnnnn	nnnnnn

Suppose that you create a pivot table that uses the `Facility` dimension alone as rows. The row for facility A represents the six records visible here.

Suppose that you add the `Encounter Type` dimension to the same pivot table, perhaps as columns. When you do so, DeepSee filters out any encounters that have no value for `Encounter Type`. This behavior is correct, although the users might not expect it. The data in the pivot table, as always, represents the selected records exactly as it should. There is no place to show the typeless encounter and thus it is not included.

Typically, you create a placeholder. This is an additional member of the dimension whose purpose is to group such records. For example, for the `Encounter Type` dimension, we could add the member `Unknown`. This member is then treated in the same way as any other member.

The implementation details depend upon the source of data for the dimension. For the purpose of planning your model, all that you need to do is to identify whether to create these placeholders and what names to use. Each dimension can have its own null placeholder.

6.1.2 Null Values and Measures

As noted in the previous section, when DeepSee builds the indices, it ignores any null values by default. For a measure, this means that the fact table contains no value for that measure, for the given record.

[illegible]

The DeepSee aggregation functions treat this null value as zero. For example, in the case shown here, suppose that measure C is aggregated by averaging the values. DeepSee adds the three values and then divides by four, which is the number of selected records.

For some types of aggregation, depending on your data, it might be suitable to assume that null means zero.

In other cases, however, this assumption is incorrect. In such cases, you define the measure so that it is filtered to use only the records where the measure value is not null. In effect, a filtered measure uses only a subset of the data and has a null value for the remainder of the data.

6.2 Date and Time Data

Date and time data is special in that there are many units of measure for them. This chapter starts by showing the options that DeepSee provides by default, as well as explaining some ways you can use the options, as well as your additional options.

6.2.1 Date Dimensions

With two key exceptions, a date dimension is just like any other dimension:

- When you create a date dimension using the standard DeepSee date type, DeepSee provides a set of variations. Each of these is a normal dimension and obeys all the same rules that other dimensions obey. You can use them alone or with other dimensions, in any combination.
- The base dimension is continuous and cannot be used in pivot tables. It can, however, be used in a filter (and remember that filters can be used nearly anywhere in DeepSee). This allows you to select a continuous range of dates.

As an example, if you define the Birth Date dimension, the following dimensions will be available to you:

- Birth Date
- Birth Date Year
- Birth Date Period

- Birth Date Quarter
- Birth Date Month
- Birth Date Week
- Birth Date Week of Month
- Birth Date Day
- Birth Date Weekday

If you use the Birth Date Year alone or with another date dimension, the resulting pivot table shows an overall timeline. For example:

<i>Birth Date Year</i>	<i>Birth Date Quarter</i>	Patient Count
2009	Q4	43
	Q3	33
	Q2	42
	Q1	26
2008	Q4	36
	Q3	28
	Q2	37
	Q1	32

The same is true for the Birth Date Period dimension.

The other date dimensions, however, use only a selected part of the date, and they ignore the overall timeline. For example, if you use Birth Date Quarter alone, you see this:

<i>Birth Date Quarter</i>	Patient Count
Q4	2,516
Q3	2,560
Q2	2,483
Q1	2,441

The Q4 row shows data for all patients born in Q4 of any year, across the entire set of patients in the base table. You could filter this pivot table to a given year, of course.

If you need to aggregate records by date in another manner, you can do so. The implementation details depend upon the data type and your needs. For the purpose of planning your model, you need only to identify each additional type of date unit.

6.2.2 Time Dimensions

By default, DeepSee provides one way to handle time data: creating hour-long buckets as follows:

Birth Time (Hourly Ranges)	
<i>Birth Time Hour Range</i>	Count
00:00 - 00:59	103
01:00 - 01:59	118
02:00 - 02:59	112
03:00 - 03:59	93
04:00 - 04:59	96
05:00 - 05:59	102

If you need to aggregate records in time using a larger or smaller bin, you can do so. For example:

Birth Time (Ranges)	
<i>Birth Time</i>	Count
before 8 am	849
8 am to 4 pm	815
4 pm to midnight	836

The implementation details depend upon the data type and your needs. For the purpose of planning your model, you need only to identify each additional type of time unit.

6.2.3 Date Measures

You can define a measure that uses a date value. Every measure must include a way to be aggregated, and the easiest way to aggregate a set of dates is to choose the first or last date in the set.

- `Earliest Transaction Date` — This measure is based on the `EndTimeStamp` field.
- `Latest Transaction Date` — This measure is based on the `EndTimeStamp` field.

Now we can create pivot tables like the following:

<i>Broker</i>	Latest Transaction Date	Net Transaction Value
Broker A	nnnnnnnn	nnnnnnnn
Broker B	nnnnnnnn	nnnnnnnn
Broker C	nnnnnnnn	nnnnnnnn
...	nnnnnnnn	nnnnnnnn

You can also create numeric measures that compute the difference between two dates. For example, you can define a measure that computes the age of something. If you are computing the age relative to the current date, however, note that the value of this measure would change constantly, which would require rebuilding the indices constantly.

In the cases where analysts consider age (for example, patient ages), it is common to compute the ages as of a certain date and to then treat the ages as static.

6.3 Compound Members

DeepSee provides the ability to create new members (*compound members*) based on existing members. A compound member is defined by a filter expression, and there is no limit to the complexity of that expression. For example, you can define compound members that do the following:

- Select all transactions between dates X and Y
- Select all transactions for customers with last names that start with R.
- Select all customers in this given range of birth years.
- Select all transactions except for those relating to this asset type.
- Select all customers who have worked with broker C.
- Select all transactions between dates X and Y that apply to customers in this income bracket
- Select all customers of class D who have traded assets of type T.
- Select all transactions of a given status that involve assets of vendor V.

When you define a compound member, you assign it to a dimension, which is either an existing dimension or a new dimension.

The compound members can be used in the same way as any other members, which means that the analyst working with your DeepSee model can drag and drop it into pivot tables, without having to create a complex query.