



DeepSee Developer Tutorial

Version 2010.1
17 February 2010

DeepSee Developer Tutorial

InterSystems Version 2010.1 17 February 2010

Copyright © 2010 InterSystems Corporation

All rights reserved.

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

About This Book	1
1 Getting Started	3
1.1 Logging into DeepSee	3
1.2 Configuring DeepSee to Use Multiple CPUs	4
1.3 Setting Up the Environment	4
1.4 Generating Data	5
2 Contents of the DeepSee Sample	7
2.1 The Sample Classes	7
2.2 Diagnosis Data	9
2.3 The Extra Data Set	10
2.4 Controlling What Is Generated	10
2.5 Generating Additional Data	11
2.6 Changing or Deleting Data	11
3 Creating a DeepSee Model	13
3.1 Viewing a Class in the Architect	13
3.2 Creating a Basic DeepSee Model	17
3.2.1 Relationship of the DeepSee Model to the Class Definition	21
3.3 Adding a Detail Listing to the Model	22
3.4 Validating the Dimensions	25
3.5 Using a Collection Property	27
3.6 Adding Placeholders for Null Members	29
3.7 Using Ranges, Translations, and Transformations	33
3.8 Adding Measures in the Analyzer	35
3.9 Defining a Filtered Measure	37
3.10 Accessing Other Classes	39
3.11 Model Summary	40
3.11.1 Dimensions	40
3.11.2 Measures	41
3.11.3 Listing Fields	41
3.11.4 Detail Lists	42
3.12 Additional Exercises for the Reader	42
4 Creating Pivot Tables	45
4.1 Creating a Basic Pivot Table	45
4.2 Additional Basic Options	47
4.3 Specifying the Sort Order	52
4.4 Using Ranking and Nesting Options	55
4.5 Using Advanced Layout Options	58
4.6 Summary of Pivot Tables	62
5 Creating KPIs	63
5.1 Creating Simple KPIs	63
5.2 Creating a String-type KPI	64
6 Creating Dashboards	67
6.1 Creating a Simple Dashboard with a Filter	67
6.2 Displaying a KPI in a Speedometer	70

6.3 Adding Multiple Filters	73
6.4 Displaying a KPI in a Label	75
7 Using Multiple Subject Areas Together	77
7.1 Rules for Creating Multiple Subject Areas	77
7.2 Multiple Subject Areas in a Pivot Table	77
7.3 Multiple Subject Areas in a Dashboard	78
7.4 Setting Up the Initial City Rainfall Subject Area	80
7.5 Creating the Linked Pivot Table	80
7.6 Adjusting the City Rainfall Subject Area and Pivot Table	82
7.7 Creating a Dashboard That Uses Multiple Subject Areas	83
8 Performing Real-time Updates	87
8.1 Preparation	87
8.2 Adding and Deleting Patients	87
8.3 Changing Patient Data	88
8.4 Changing Doctor Data	89
Appendix A:Installing the Database for the Sample DeepSee Model	91

About This Book

This book is a tutorial to help developers learn the basic process of creating DeepSee models and then using them to create pivot tables and dashboards. This book contains the following sections:

- [Getting Started](#)
- [Contents of the DeepSee Sample](#)
- [Creating a DeepSee Model](#)
- [Creating Pivot Tables](#)
- [Creating KPIs](#)
- [Creating Dashboards](#)
- [Using Multiple Subject Areas Together](#)
- [Performing Real-time Updates](#)
- [Installing the Database for the Sample DeepSee Model](#)

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 Model Design Guide](#)*, an introductory guide for implementers and business users.
- *[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 on troubleshooting.
- *[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

Getting Started

The SAMPLES namespace includes the DeepSee sample, which consists of the DeepSee.Study.Patient class and related classes. This sample is meant for use as the basis of a DeepSee model. It does not initially contain any data; nor does it include a DeepSee model (dimensions, measures, and so on).

This sample is intended as a flexible starting point for working with DeepSee. You use this sample to generate as much data or as little data as needed, and then you use the DeepSee Architect to create a DeepSee model that explores this data. You can then create DeepSee pivot tables, KPIs, and dashboards based on this model. The sample contains enough complexity to enable you to use the central DeepSee features and to anticipate many typical real-life scenarios.

This book presents hands-on exercises that use this sample, starting with the DeepSee Architect and concluding with the DeepSee Dashboard Designer.

Important: If you have not yet used a DeepSee pivot table, it is suggested that you read the book *Overview of DeepSee*, especially the first chapter.

1.1 Logging into DeepSee

1. Click the InterSystems Launcher.
When you do so, the system displays a menu.
2. Click **DeepSee**.
If you have not yet specified a namespace, the system displays a page that prompts you for a namespace.
Otherwise, the system displays the DeepSee login page.
3. If you are prompted for a namespace, type SAMPLES and then click **Logon to DeepSee**.
The system then displays the DeepSee login page.
4. On the DeepSee login page, enter a DeepSee username and password. For example, you can use the username demo with the password demo.
5. For **Role**, select demo.
6. Click **Login**.

1.2 Configuring DeepSee to Use Multiple CPUs

By default, DeepSee uses only 1 processor when it builds its indices. If your machine has multiple CPUs, it is useful to configure DeepSee to use more processors, so that the rebuild is quicker.

Do the following:

1. Log into DeepSee as described in the previous section.
2. Click **Administrator > Site Configuration**.
3. Click **OLAP** on the left.
4. For **# of Rebuild Process**, specify an integer. Typically you use a number that is the same as or slightly larger than the number of processors.
5. Click **Save**.

1.3 Setting Up the Environment

Before working with the DeepSee sample, you might find it worthwhile to modify the SAMPLES namespace so that the DeepSee model that you create and the DeepSee indices that you generate are stored in separate databases used by this namespace:

Database Name (Example)	Database Purpose	Comments
SampleDSModel	To contain the DeepSee model.	
SampleDSIX	To contain the DeepSee indices.	Pre-expanded. Also, not journaled.

This procedure is not required but is useful for two reasons:

- It enables you to replace your model with a DeepSee model from InterSystems, if desired; see the appendix “[Installing the Database for the Sample DeepSee Model](#).”
- When you later upgrade your Caché installation to a newer release, your work with the DeepSee sample will be retained, even though the SAMPLES database is replaced. (After upgrading, however, it will be necessary to recompile the DeepSee sample classes, as well as regenerate the data and rebuild the DeepSee indices.)

The steps are as follows:

1. First, create the index database (SampleDSIX, for example):
 - a. Access the System Management Portal.
 - b. Click **[Configuration] > [Local Databases]**.
 - c. Click **New**.
 - d. Specify the name of this database, as well as the directory in which to create it.
 - e. Click **Next**.
 - f. For **Journal Globals**, click **No**.
 - g. Click **Finish**.

2. Create the model database (SampleDSModel, for example). Follow steps a through e in the previous step and then click **Finish**.
3. Click **[Configuration] > [Namespaces]**.
4. In the row for the SAMPLES namespace, click **Global Mappings**.
5. Add the following global mappings:

Global	Subscript	Database
BI.*		SampleDSModel
BIidx.*		SampleDSIX
BIlog.*		SampleDSIX
SYS	("BI")	SampleDSModel

For example, to add the first mapping:

- a. Click **New Global Mapping**.
- b. For **Global Database Location**, select the model database (SampleDSModel).
- c. For **Global Name**, type BI.*
- d. Click **Apply**.
- e. Click **Close** to close the dialog box.
- f. Click **Save Changes** to save this change.

Notice that the user interface automatically adds a mapping for SYS so that the result is as follows:

The global mappings for namespace SAMPLES are

	Global	Subscript	Database	
	BI.*		SAMPLEDSMODEL	Edit
	BIidx.*		SAMPLEDSIX	Edit
	BIlog.*		SAMPLEDSIX	Edit
	SYS		SAMPLES	Edit
	SYS	("BI")	SAMPLEDSMODEL	Edit

1.4 Generating Data

To generate data for the DeepSee sample:

1. In the Terminal, switch to the SAMPLES namespace:

```
zn "SAMPLES"
```

2. Execute the following command:

```
do ##class(DeepSee.Study.Utils.Populate).GenerateData()
```

This class method generates 10000 patients, a small sample to start.

Later you can regenerate the data to create a much larger data set if needed. The method also provides options that control which child tables are created; see “[Controlling What Is Generated](#),” in the next chapter.

For an overview of the classes, see the chapter “[Contents of the DeepSee Sample](#).” Also see the class documentation.

2

Contents of the DeepSee Sample

When you create models in DeepSee, it is essential to start by becoming familiar with the data that you are using. This chapter introduces the data used in the rest of this book.

The `DeepSee.Study.Patient` class and related classes are meant to represent a fictitious study of a set of patients, who have assorted allergies and diagnoses. The sample is designed to enable you to try a wide variety of the features of DeepSee and is not meant to contain truly realistic data. It has the following features:

- A method that you use to generate as much or as little data as you need.
- An option for including nulls in the generated data, to simulate real-life gaps in data.
- Built-in structure that you can use DeepSee to detect.
- Collection properties, including relationship properties.
- Date and time properties of various types.
- Simple string and numeric properties.
- Child tables of varying complexity.
- A child table that contains (potentially) multiple rows for any given row in the parent table (and that cannot be accessed from the primary table via cascading dot syntax).
- Structural difficulties that model non-ideal class definitions sometimes found in real-life applications. The purpose of these is to demonstrate how to work around such difficulties when creating a DeepSee model. These difficulties include the following:
 - A detail table that is not linked to the primary table via a property relationship (and thus cannot be accessed from the primary table via cascading dot syntax).
 - An extra table containing an additional set of patients (so that the patients are not all within a single extent).

2.1 The Sample Classes

The sample classes are as follows:

- `DeepSee.Study.Utils.Populate` contains the `GenerateData()` method and other methods to populate the classes.
- `DeepSee.Study.Utils.ForModel` contains utility methods for use in your DeepSee model. These are specific to this sample.

- DeepSee.Study.Patient is the primary class. Each patient has properties like the following:
 - A unique patient ID that identifies the patient within a fictitious epidemiological study.
 - Age and gender. The age and gender combinations of the patients reflect the distribution of ages and genders as recorded in the US 2000 census.
 - Birth date and time.
 - Home city. The city belongs to a parent ZIP code.
 - A primary care physician.
 - A list of allergies. Each allergy is a complex piece of data that consists of an allergen, an allergy severity, and the nature of the reaction to that allergen.

The allergies property is serial, so its data is stored in the DeepSee.Study.Patient table.
 - Several properties that contain the same (or nearly the same) diagnosis data. Some of this data is stored in the DeepSee.Study.Patient table, and some is stored in other tables.

The diagnoses are assigned based on age and gender, from a small set of diagnoses.
 - A test score (for a fictitious test administered to the patients in this study). For this test, the lowest possible value is 50.
 - Group to which the patient belongs (A or B).

DeepSee.Study.Patient inherits from %BI.Adaptor.

- DeepSee.Study.Doctor represents the doctors. Each doctor has properties like the following:
 - Name. The first and last names are stored in separate fields.
 - Main city in which this doctor works. The city belongs to a parent ZIP code.
 - Group to which this doctor belongs (I, II, or III).
 - Average number of patients that this doctor sees per week.
- Some *code tables* — that is, tables that consist of reference information used elsewhere. These tables include a Code property and a Description property. The Code property is used internally (and is used in foreign keys), and the Description property is the corresponding user-friendly string.

These tables are as follows:

- DeepSee.Study.Allergen
- DeepSee.Study.AllergySeverity
- DeepSee.Study.City
- DeepSee.Study.Diagnosis
- DeepSee.Study.NatureOfReaction
- DeepSee.Study.Profession

Each of these tables is set up in the same way. For example, DeepSee.Study.City contains an XData block which contains the city data in a slightly structured form. The **Setup()** method in this class reads that XData block and populates the table.

Tip: You can edit the XData block to include your own data.

When you run the **GenerateData()** method mentioned earlier, it calls these **Setup()** methods.

- `DeepSee.Study.PatientDetails` contains additional data for the patients. It includes the following properties:
 - Patient ID as used in the fictitious epidemiological study.
 - Favorite color.
 - Profession, if any, of the patient. Professions, in turn, belong to different industries.

This class is not linked to `DeepSee.Study.Patient` via a property relationship. Instead, you must use the patient ID to connect `DeepSee.Study.Patient` and `DeepSee.Study.PatientDetails`.

Also, not all patients have records in this table.

- `DeepSee.Study.PatientEncounter` contains a record for each encounter (medical visit) of a patient, and thus can have multiple records for any given patient. Not all patients have records in this table.

This table is not populated by default by the method shown earlier.

- `DeepSee.Study.PatientDiagnosis`, `DeepSee.Study.PatientDiagnosis1`, and `DeepSee.Study.PatientDiagnosis2` store the patient diagnosis data. For information on these variations, see “[Diagnosis Data](#).”
- `DeepSee.Study.PatientSet2` stores an additional set of patients with a slightly different (and smaller) set of fields.

This table is not populated by default by the method shown earlier.

- `DeepSee.Study.CityRainfall` stores rainfall data for the cities, over time. With this class, you can create a separate `DeepSee` model, so that you can experiment with working with models in areas that are semantically unrelated.

This table is not populated by default by the method shown earlier.

2.2 Diagnosis Data

To help you learn different ways to use lists and other collections in `DeepSee`, this sample stores multiple versions of the patients’ diagnoses. The `DeepSee.Study.Patient` class includes the following properties:

```
// Use the main version to see how we handle lists of objects
Property Diagnoses As list Of DeepSee.Study.PatientDiagnosis;

// Use this variation to see how we handle arrays.
Property DiagnosesFromArray As array Of %String;

// Use this variation to see how we handle $LB lists of strings.
Property DiagnosesAsLB As %List;

// Use this variation to see how DeepSee handles pieced strings.
Property DiagnosesAsString As %String;

// Use this variation to see how we handle parent-child relationships
Relationship DiagnosesAsChildren As DeepSee.Study.PatientDiagnosis1
[ Cardinality = child, Inverse = Patient ];

// Use this variation to see how we handle one-many relationships
Relationship DiagnosesAsMany As DeepSee.Study.PatientDiagnosis2
[ Cardinality = many, Inverse = Patient ];
```

A diagnosis consists of a diagnosis code, the corresponding description, the doctor who made the diagnosis, and other data (which we omit for simplicity). The diagnosis properties in `DeepSee.Study.Patient` contain versions of this data as follows:

- The `Diagnoses` property is a list of `DeepSee.Study.PatientDiagnosis`. That class, in turn, includes these properties:

```
Property DiagnosisCode As %String;
Property DiagnosedBy As DeepSee.Study.Doctor;
```

- The `DiagnosesFromArray` property is an array of strings. The strings are the diagnosis codes.

- The `DiagnosesAsLB` property is a list — that is, a classic list such as produced by `$LISTBUILD`. Each list item is a diagnosis code.
- The `DiagnosesAsString` property is a pieced string. It consists of diagnosis codes separated by commas.
- The `DiagnosesAsChildren` property establishes a parent-child relationship between the class `DeepSee.Study.Patient` and the class `DeepSee.Study.PatientDiagnosis1`.

`DeepSee.Study.PatientDiagnosis1` contains the diagnosis data in more detail than the preceding properties. This class has the following properties:

```
Relationship Patient As DeepSee.Study.Patient
[ Cardinality = parent, Inverse = DiagnosesAsChildren ];

Property DiagnosisCode As %String;

Property DiagnosedBy As DeepSee.Study.Doctor;
```

- The `DiagnosesAsMany` property establishes a one-to-many relationship between the class `DeepSee.Study.Patient` and the class `DeepSee.Study.PatientDiagnosis2`.

`DeepSee.Study.PatientDiagnosis2` has the following properties:

```
Relationship Patient As DeepSee.Study.Patient
[ Cardinality = one, Inverse = DiagnosesAsMany ];

Property DiagnosisCode As %String;

Property DiagnosedBy As DeepSee.Study.Doctor;
```

When you generate data, the data generation method ensures that the diagnosis data in these properties is consistent. That is, if a given patient has the diagnosis diabetes, that diagnosis is stored in each of these properties.

2.3 The Extra Data Set

In an ideal scenario, all the unique patients would be stored in a single table. That is, to find the total number of patients, you would have to count the records in only one table. That table serves as the starting point for accessing all data for those patients.

In real life, however, you might need to access an additional set of patients in an unrelated table. (While this is not the best object-oriented design, this scenario can occur if one organization merges with another. Rather than rewriting all the code that both organizations own, it would be simpler to leave the data in its current storage.)

The sample gives you an easy way to practice with such a scenario. The extra set is not generated by default. For information on generating the additional set of patients, see the following two sections.

2.4 Controlling What Is Generated

You can generate data in this sample in a variety of ways. The central method is the `GenerateData()` method of the `DeepSee.Study.Utils.Populate` class, which has the following signature:

```
classmethod GenerateData(patCount As %Integer = 10000,
                        patientsPerDoc As %Numeric = 25,
                        options As %String = "ADT",
                        genNulls As %Boolean = 1)
```

These arguments are as follows:

- *patCount* specifies the number of patients to create.
- *patientsPerDoc* specifies the ratio of patients to doctors and thus determines the number of doctors to create.
- *options* is a string that lets you control which data to generate. Depending on the characters you include in this string, the method behaves as follows:

Character	Action if String Includes This Character	Default
"A"	Generates allergies for the patients.	Enabled
"D"	Generates diagnoses for the patients.	Enabled
"T"	Populates DeepSee.Study.PatientDetails.	Enabled
"E"	Populates DeepSee.Study.PatientEncounter.	Disabled
"R"	Populates DeepSee.Study.CityRainfall.	Disabled
"X"	Creates half the patients in DeepSee.Study.Patient and half in DeepSee.Study.PatientSet2.	Disabled

- *genNulls* lets you control whether to include nulls in the generated data. If *genNulls* is 1, the method generates a random percentage of nulls in specific parts of the data. For example, some percentage of patients will not receive a primary care physician.

2.5 Generating Additional Data

To generate additional data, use the following methods. (You must first use the central **GenerateData()** method [described in the previous section](#).)

- To generate additional patients for DeepSee.Study.Patient, use the **AddPatients()** class method of that class.

```
classmethod AddPatients(patCount As %Integer = 100,
                        options As %String = "ADET",
                        genNulls As %Boolean = 1) as %Status
```

- To generate additional patients for DeepSee.Study.PatientSet2, use the **AddPatients()** class method of that class.

```
classmethod AddPatients(patCount As %Integer = 500,
                        options As %String = "AD",
                        genNulls As %Boolean = 1) as %Status
```

- To generate rainfall data in DeepSee.Study.CityRainfall, use the **GenerateData()** class method of that class.

```
classmethod GenerateData() as %Status
```

This data is provided so that you can experiment with multiple, unconnected subject areas. See the chapter [“Using Multiple Subject Areas Together.”](#)

2.6 Changing or Deleting Data

So that you can see how DeepSee handles changes to data, this sample also includes methods to change or delete data in various tables. You must first use the central **GenerateData()** method [described previously](#).

- To delete a percentage of the patients (chosen randomly), use the **DeletePatients()** class method of the class `DeepSee.Study.Patient`.

```
classmethod DeleteSomePatients(percent As %Numeric = 10) as %Status
```

- To change assorted details for a percentage of patients, use the **ChangeSomePatients()** class method of the class `DeepSee.Study.Patient`.

```
classmethod ChangeSomePatients(percent As %Numeric = 20,  
                                rebuild As %Boolean = 0)
```

This method calls the following three methods, which you can also call separately.

- To change the group to which a patient is assigned, for a percentage of patients, use the **ChangePatientGroups()** class method of the class `DeepSee.Study.Patient`.

```
classmethod ChangePatientGroups(percent As %Numeric = 30)
```

- To change the favorite color of a patient, for a percentage of patients, use the **ChangePatientDetails()** class method of the class `DeepSee.Study.PatientDetails`.

```
classmethod ChangePatientDetails(percent As %Numeric = 20,  
                                  rebuild As %Boolean = 0)
```

- To generate additional patient encounters, use the **AddEncounters()** class method of the class `DeepSee.Study.PatientEncounter`.

```
classmethod AddEncounters(percent As %Numeric = 20,  
                           rebuild As %Boolean = 0) as %Status
```

- To change the doctor group and patients per week for doctors, for some percentage of doctors, use the **ChangeSomeDoctors()** class method of the class `DeepSee.Study.Doctor`.

```
classmethod ChangeSomeDoctors(percent As %Numeric = 20  
                               rebuild As %Boolean = 0)
```

Though simple, all these changes are sufficient to demonstrate how to keep DeepSee indices current.

None of these methods affect `DeepSee.Study.PatientSet2`.

3

Creating a DeepSee Model


This tutorial starts with a BI-enabled class and ends with a simple but complete DeepSee model that you can use in the Analyzer. It consists of the following parts:

1. [Viewing a class in the Architect](#)
2. [Creating a basic DeepSee model](#)
3. [Adding a detail listing to the model](#)
4. [Validating the dimensions](#)
5. [Using a collection property](#)
6. [Adding placeholders for null members](#)
7. [Using ranges, translations, and transformations](#)
8. [Defining additional measures in the Analyzer](#)
9. [Defining a filtered measure in the Analyzer](#)
10. [Accessing data in other tables](#)
11. [Summary of the model definition](#)
12. [Self-guided exercises for you to try](#)

Because all this work is within a single namespace, you can open two browser tabs and use one for the Architect and the other for the Analyzer. The tutorial does not describe this technique specifically, but it can be a convenient way to work.

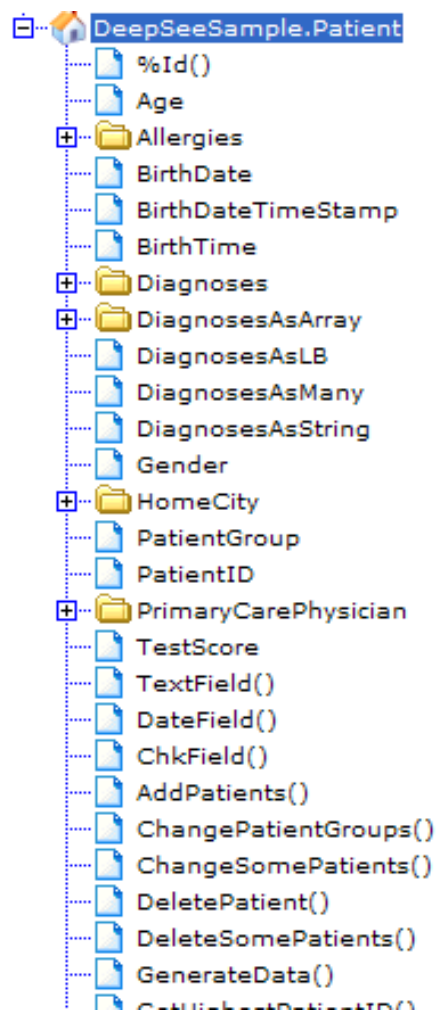
3.1 Viewing a Class in the Architect

In this part of the tutorial, you get acquainted with how the Architect displays a class that is BI-enabled.

1. Log into DeepSee as [described earlier](#).
2. Click **Data Modeler > Architect**.
3. Click the open class button () in the Architect menu bar. By default, DeepSee lists all the classes that inherit from %BI.Adaptor in this namespace.

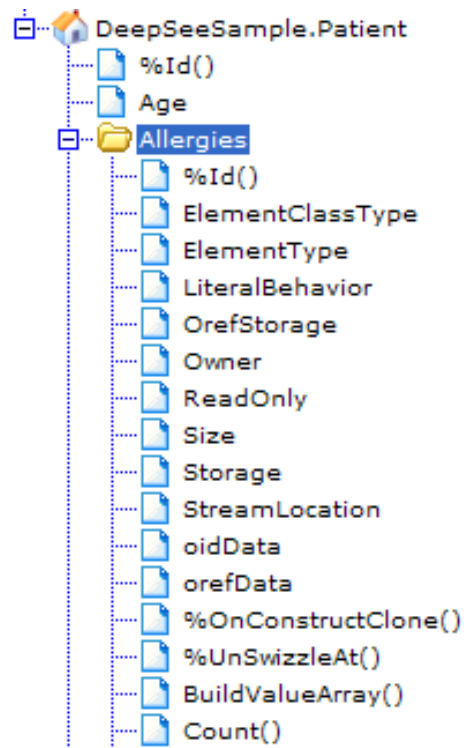
Note: You can open other classes, as well. When you open a class from the Architect, DeepSee adds %BI.Adaptor to the superclass list in that class definition.

4. Click the DeepSee.Study.Patient class and then click **OK**.
5. Click the plus sign (+) beside DeepSee.Study.Patient. DeepSee displays the following:



Notice that some of the properties of this class are shown as folders. Instance methods are shown at the bottom of the list, along with three utilities (**TextField()**, **DateField()**, and **ChkField()**) provided by DeepSee.

6. Click the plus sign (+) beside Allergies. DeepSee displays the following:



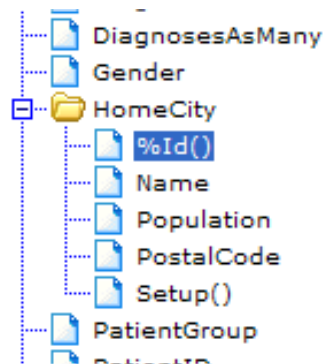
This folder contains the properties and methods that are available for a list, because the Allergies property is defined as a list:

```
Property Allergies As list Of DeepSee.Study.PatientAllergy;
```

DeepSee displays similar results for Diagnoses or DiagnosesFromArray, which are other types of collections.

The Architect does not display the properties and methods of the object used in the collection (in this case DeepSee.Study.PatientAllergy), but you do have access to them in DeepSee.

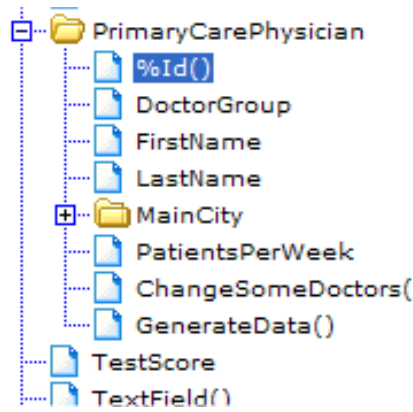
7. Click the plus sign (+) beside HomeCity. DeepSee displays the following:



This folder contains the properties and methods of DeepSee.Study.City, because the HomeCity property is defined as follows:

```
Property HomeCity As DeepSee.Study.City;
```

8. Click the plus sign (+) beside PrimaryCarePhysician. DeepSee displays the following:



This folder contains the properties and methods of DeepSee.Study.Doctor, because the PrimaryCarePhysician property is defined as follows:

```
Property PrimaryCarePhysician As DeepSee.Study.Doctor;
```

9. Expand other folders and examine their contents. Compare what you see to what this section has discussed so far.
10. Open this class (DeepSee.Study.Patient) in the Studio and compare the list of properties you see there to the list shown in the Architect.

Notice that this class includes a property, DiagnosesAsChildren, that is not displayed in the Architect. This property is defined as follows:

```
Relationship DiagnosesAsChildren As DeepSee.Study.PatientDiagnosis1  
[ Cardinality = children, Inverse = Patient ];
```

You can use this property in DeepSee even though the Architect does not display it.

The following rules govern the display of a class in the Architect:

- For a BI-enabled class, all properties are shown except — for a parent-child relationship — the side that is marked `Cardinality = children`.
- This display is recursive; that is, properties of properties are shown.
- If a property is a collection (a list, an array, or a relationship), it is shown as a folder that contains the properties and methods of the collection.
- If a property is of type `%List` (which is the object equivalent of `$LISTBUILD`), it is not shown as a folder.
For example, see the `DiagnosesAsLB` property, which is included in this sample to illustrate this point.
- If a property is a relationship, it is not shown as a folder.
For example, see the `DiagnosesAsMany` property, which is included in this sample to illustrate this point.
- If a class is not accessible from the base class via cascading dot syntax, it is not shown by default.
For example, the Architect does not display `DeepSee.Study.PatientDetails` or `DeepSee.Study.PatientEncounter` by default.
- The Architect does not display properties and methods inherited from superclasses.

All dimensions, measures, and listing fields are based either on a source property or on a source expression (which is a Caché ObjectScript expression) that can use properties of any class.

Important: The Architect provides a useful view of the class properties, which makes it very easy to create DeepSee elements based on those properties. It is important, however, to know that although this view provides a convenient way to access some properties, you can also use a source expression to access any data, including properties and methods of the superclasses. In some cases, you might need to use dynamic SQL. These source expressions are evaluated when the indices are built and thus do not affect your runtime performance. This tutorial demonstrates these points later.

3.2 Creating a Basic DeepSee Model

In this part of the tutorial, you create dimensions, measures, and a subject area, using `DeepSee.Study.Patient` as the base class.

1. Access the Architect and open the class `DeepSee.Study.Patient`.
2. Click the **Dim** tab, expand the folders, and double-click each of the following properties (or drag and drop them to the **Dimensions** table):
 - Age
 - BirthDate
 - Gender
 - PatientGroup
 - TestScore
 - Within HomeCity, Name
 - Within HomeCity, PostalCode
 - Within PrimaryCarePhysician, DoctorGroup
 - Within PrimaryCarePhysician, LastName

As you do this, the Architect defines a dimension based on each of these properties. When you are done, DeepSee displays something like the following (your ID values might be different from this):

Dimensions				
Sort By : ID <input type="button" value="v"/>				
ID	Dimension Name	Property	Data Type	Active
10001	Age	Age	Number	<input checked="" type="checkbox"/>
10002	BirthDate	BirthDate	Date	<input checked="" type="checkbox"/>
10003	Gender	Gender	Values	<input checked="" type="checkbox"/>
10004	PatientGroup	PatientGroup	Values	<input checked="" type="checkbox"/>
10005	TestScore	TestScore	Number	<input checked="" type="checkbox"/>
10006	DoctorGroup	PrimaryCarePhysician.DoctorGroup	Values	<input checked="" type="checkbox"/>
10007	LastName	PrimaryCarePhysician.LastName	Values	<input checked="" type="checkbox"/>
10008	Name	HomeCity.Name	Values	<input checked="" type="checkbox"/>
10009	PostalCode	HomeCity.PostalCode	Values	<input checked="" type="checkbox"/>

The `TestScore` dimension is also a measure, because **Data Type** is **Number**. In fact, this is the process to define measures: add a numeric dimension.

3. Now rename most of these dimensions, as follows:
 - a. Click the dimension in the table.
 - b. The tabs below the table now show details for this dimension.
 - c. Edit **Dimension Name** within the **General** tab.
 - d. Click **Update** to save this change.

The buttons in the lower right apply to the currently selected dimension. If you do not save changes before selecting another dimension, your changes to that dimension are discarded.

Use the following new names:

Old Name	New Name
BirthDate	Birth Date
PatientGroup	Patient Group
Name	Home City
PostalCode	Home ZIP
LastName	Doctor
TestScore	Test Score
DoctorGroup	Doctor Group

Do not forget to click **Update** after you modify a dimension.

Now DeepSee displays something like the following:

Dimensions				
Sort By : <input type="text" value="ID"/>				
ID	Dimension Name	Property	Data Type	Active
10001	Age	Age	Number	<input checked="" type="checkbox"/>
10002	Birth Date	BirthDate	Date	<input checked="" type="checkbox"/>
10003	Gender	Gender	Values	<input checked="" type="checkbox"/>
10004	Patient Group	PatientGroup	Values	<input checked="" type="checkbox"/>
10005	Test Score	TestScore	Number	<input checked="" type="checkbox"/>
10006	Doctor Group	PrimaryCarePhysician.DoctorGroup	Values	<input checked="" type="checkbox"/>
10007	Doctor	PrimaryCarePhysician.LastName	Values	<input checked="" type="checkbox"/>
10008	Home City	HomeCity.Name	Values	<input checked="" type="checkbox"/>
10009	Home ZIP	HomeCity.PostalCode	Values	<input checked="" type="checkbox"/>

4. Double-click Age again in the DeepSee.Study.Patient folder. This step adds this property again to the table on the right.

Then make the following changes:

- Rename the new item to Age Year.
- On the **General** tab, change the type to **Values**.

This step defines the new item as a dimension rather than as a measure.

- Click **Update** to save this change.

5. Next we modify the definition of the Doctor dimension so that it uses both the first and last names. To do this:

- Click Doctor in the **Dimensions** table.
- On the **General** tab, type the following Caché ObjectScript expression into **Complex Code**:

```
%this.PrimaryCarePhysician.FirstName _ " " _ %this.PrimaryCarePhysician.LastName
```

Here, %this represents the current patient. This code is executed when you build the DeepSee indices; DeepSee iterates through the base class and builds the indices for each record in that class. When it does so, it executes any expression in **Complex Code** and uses it to index that dimension.

- Delete the string PrimaryCarePhysician.LastName contained in the second property field. This step is optional but makes the definition less confusing to view.
- Click **Update** to save this change.

The details for this dimension are now as follows:

General	Manual Child Browse	Translation/Replacement	Ranges	Script	SQL
Dimension Name	Doctor		Listing Field :	<input type="checkbox"/>	Count P
Data Type	Values		Transformation Type :	None	
Property :	DeepSee.Study.Patient				
Link Property :		Link To:			
Complex Code :	%this.PrimaryCarePhysician.FirstName_" "%this.PrimaryCarePhysician.LastName				
	Script				

6. Next, we define a subject area so that we can see these dimensions in the Analyzer. To do this:

- a. Click the **Subject** tab.
- b. In **Subject Name**, type the name `Patients`.
- c. On the **Role Access** tab, click the check box next to the demo role.

Subject Details

Subject Name :


Subject Filter

Access All Dim .?
 Active ?

Access	Role	Default Detail Listing	Filter for Detail Listing
<input checked="" type="checkbox"/>	demo		

d. Click **Add**.

7. Compile the base class and rebuild the DeepSee indices. To do so:

- a. Click the process button () in the menu bar.
- b. Click **Rebuild**. This option recompiles and then rebuilds.
- c. Click the X in the upper right to close this dialog box.

Tip: This dialog box is not always automatically refreshed when the compilation or rebuild status changes. To refresh the display and see the current status, click **Refresh**.

8. Verify that you can access this model from the Analyzer. To do so:

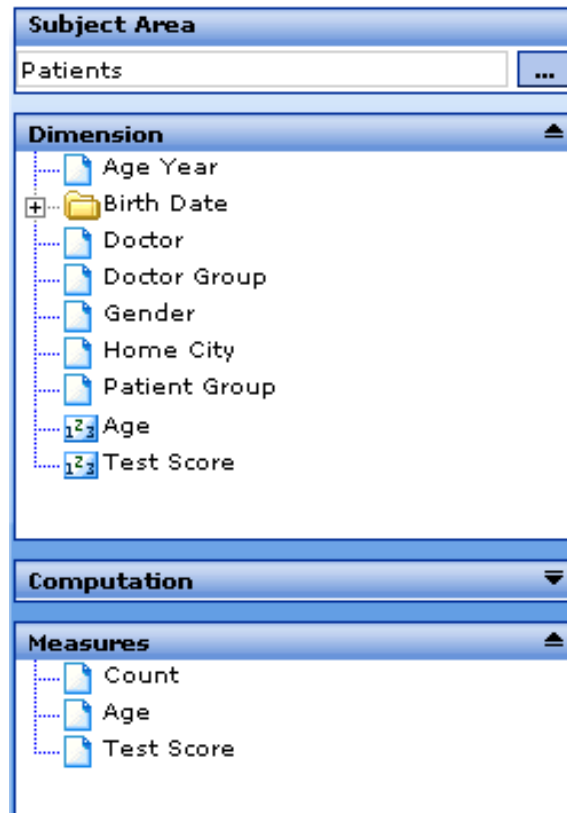
- a. Click **Main > Analyzer**.
- b. Click the browse button (...) in the **Subject Area** panel.



- c. Click `Patients` and then click **OK**.

If `Patients` is not listed, make sure you gave access to this subject area to the demo role.

Now the Analyzer displays the following:



If you do not see these dimensions and measures, make sure that you have rebuilt the DeepSee indices.

Note: Unlike the other dimensions created here, the `Doctor` dimension can have a very large number of members, depending on the size of your data set. In a real-world implementation, it is unlikely that you would create a dimension at such a low level. This tutorial uses this dimension because it demonstrates a couple of key points.

3.2.1 Relationship of the DeepSee Model to the Class Definition

When you define a DeepSee model, the model definition is stored in globals and the details are visible only via the Architect. When you compile the class, however, Caché uses the model definition and the class definition together and compiles both into the same INT code. Caché uses the INT code to determine how to create the fact table and indices. For details, see [Overview of DeepSee](#).

3.3 Adding a Detail Listing to the Model

In this part of the tutorial, we add a simple detail listing. First we create listing fields and then we use them to create detail listings.

There are two kinds of listing fields:


- A *dimension-type listing field* is a listing field whose definition is owned by a dimension definition. If you redefine or delete the dimension, the listing field is automatically redefined or deleted as well.

The data used by the listing field is stored in a DeepSee global, which means that its performance is fast. This kind of listing field, however, requires more disk space than the other kind.

- An *independent listing field* is a listing field that has an independent definition (not dependent on the definition of any dimension).

For this kind of listing field, DeepSee does not access the data until the user displays the detail listing; then DeepSee directly accesses the source data.

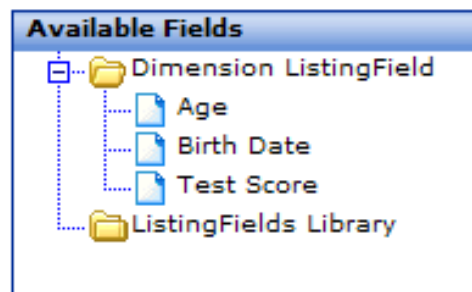
In this part of the tutorial, we will create both kinds of listing fields.

1. At the top of the page, click **Data Modeler > Architect**.
2. Click the open class button () in the Architect menu bar.
3. Click the DeepSee.Study.Patient class and then click **OK**.
4. Click the **Dim** tab.

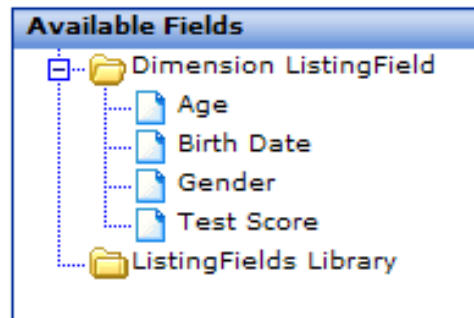
In the **Dimensions** table, notice that the **Listing** check box is selected for the Age, Birth Date, and Test Score dimensions.

By default, when you create a date or numeric dimension, it is also created as a dimension-type listing field.

5. Click the **Detail List** tab and then click the plus sign (+) next to **Dimension Listing Field**. This folder lists all the dimension-type listing fields:



6. Add Gender as another dimension-type listing field. To do so:
 - a. Click the **Dim** tab.
 - b. In the **Dimensions** table, click Gender.
 - c. On the **General** tab, click the **Listing Field** check box.
 - d. Click **Update** to save this change.
7. To see this change, click the **Detail List** tab and then click the plus sign (+) next to **Dimension Listing Field**.



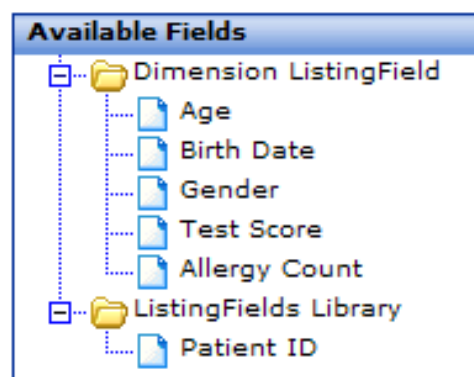
8. Add another listing field, `Patient ID`, as an independent listing field. To do so:
 - a. Click the **List Field Library** tab.
The **Listing Field** table is initially empty.
 - b. Expand the class hierarchy on the left.
 - c. Double-click `PatientID`. A new listing field is immediately added.
 - d. On the **General** tab below the **Listing Field** table, edit the value for **Dimension Name** (which specifies the name for this listing field). Change the name to `Patient ID`.
 - e. Click **Update** in the lower right.

Now the table looks like this:

Listing Field			
Fields Caption	Property	Data Type	Active
Patient ID	PatientID	Text	<input checked="" type="checkbox"/>

9. Now create a detail listing that includes all these listing fields.
 - a. Click the **Detail List** tab.
 - b. In **Listing Name**, type `Basic Details`.
 - c. In **Available Fields**, expand the two folders so that you can see all the available listing fields.

The **Dimension Listing Field** folder lists the dimension-type fields. The **Listing Fields Library** folder lists the independent listing fields.



- d. Double-click each of the listing fields. When you double-click a field, it is added to the area on the right, which lists the contents of this detail listing.

The order of the listing fields here controls the order of the fields in the actual detail listing. However, you can add fields in any order and change the order later. To change the position of a listing field, right-click it and click **Move Up** or **Move Down**.

- e. Click the **Role Access** tab.
- f. On the **Role Access** tab, click the check box next to the demo role.
- g. Click **Add**.

10. Recompile the base class and rebuild the DeepSee indices, as in the previous part of this tutorial.

Tip: You can instead recompile in the Studio and rebuild the indices in the Terminal. To rebuild the indices in the Terminal, change to the SAMPLES namespace and execute the following command:

```
Do ##class(DeepSee.Study.Patient).HiRebuild()
```

11. Verify that you can access this detail listing in the Analyzer. To do so:

- a. Click **Main > Analyzer**.
- b. The `Patients` subject area should be displayed from the last time you used the Analyzer. However, if it is not, click the search button (...) in the **Subject Area** panel. Then click `Patients` and then click **OK**.
- c. Click **Find** in the upper right. The Analyzer then displays a pivot table that counts the records in this subject area. It looks like this:

	Count
All	10,000

- d. Right-click on the white cell and then click **Listing to Screen**.

If this option is not listed, make sure you gave access to this detail listing to the demo role. If correcting that does not fix the problem, make sure that you recompiled and rebuilt.

DeepSee then displays something like the following:

basic details i				
PatientID	Age	Birth Date	Gender	Test Score
SUBJ_100301	2	20 Oct 2007	F	66
SUBJ_100302	4	14 Apr 2005	M	83
SUBJ_100303	74	25 Aug 35	F	63
SUBJ_100304	28	26 May 81	F	
SUBJ_100305	40	09 Jul 69	M	84
SUBJ_100306	66	29 Apr 43	F	82
SUBJ_100307	31	21 Jul 78	F	91
SUBJ_100308	18	13 Jan 91	F	96
SUBJ_100309	70	28 Mar 39	F	86
SUBJ_100310	68	31 Jul 41	M	92
SUBJ_100311	74	13 Oct 35	F	
SUBJ_100312	12	01 Feb 97	M	72

If you do not see a result similar to this, make sure that you recompiled and rebuilt.


3.4 Validating the Dimensions


This part of the tutorial shows you a simple way to validate the dimensions that you have created so far. In this part, we use the Analyzer to determine how many records each dimension accesses.

1. Access the Analyzer and open the `Patients` subject area.
2. Notice the **Filter** pane in the lower left. This pane indicates the number of records currently in the subject area.



The first number (10000) indicates how many patients are in the subject area. The second number (100.00%) indicates whether you have applied any filtering in the Analyzer. The value 100% means that you have not applied any filter.

3. In the toolbar, click the autofind icon ().

The icon changes to look like this: .

Now whenever you change the pivot table, the Analyzer immediately updates the display.

4. In the upper right, click the list for **Grand** and select **Total**.

The Analyzer now displays a simple pivot table that looks like this:

	Count
All	10,000

5. Double-click `Age Year` or drag and drop this dimension to the **Rows** pane.

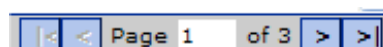
The Analyzer then displays this dimension as rows, as follows; the details are different for you because the data generation is random.

<i>Age Year</i>	Count
0	126
1	141
10	122
11	150
12	144

The pivot table contains more than one page of results.

Note: You can see that the years are sorted as if they are strings. It would be preferable to sort the years numerically. A [later chapter](#) shows how to do this.

6. Click the go to end (`>|`) button in the lower right:

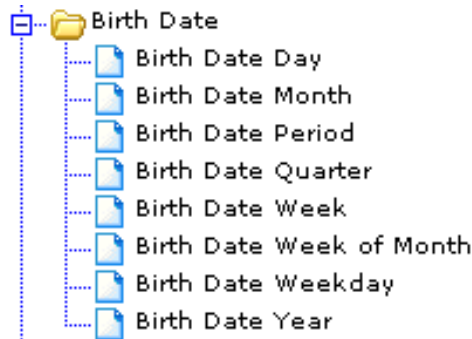


The Analyzer displays the following on the last page of results:

<i>Age Year</i>	Count
Grand Total	10,000

The value for *Grand Total* is the same as the total record count for the subject area. This result is correct.

- Click the X in the upper right of the **Rows** pane. This removes the dimension you used from this pane without otherwise changing the pivot table.
- Click the plus sign (+) next to *Birth Date* in the **Dimension** pane.



- Double-click any dimension in this folder or drag and drop this dimension to the **Rows** pane.

The Analyzer then displays the following:

<i>Birth Date Quarter</i>	Count
Q1	2,572
Q2	2,466
Q3	2,458
Q4	2,504
Grand Total	10,000

The value for *Grand Total* is the same as the total record count for the subject area. This result is correct.

- Click the X in the upper right of the **Rows** pane. This removes the dimension you used from this pane without otherwise changing the pivot table.
- Double-click *Gender* or drag and drop this dimension to the **Rows** pane.

The value for *Grand Total* is the same as the total record count for the subject area. This result is correct.

- Repeat the preceding process for *Home City*, *Home ZIP*, *Patient Group*, and *Doctor*.

You will find that for *Patient Group*, *Doctor Group*, and *Doctor*, the *Grand Total* is smaller than the total record count for the subject area. This indicates that not all patients are assigned to a patient group, not all patients have a recorded primary care physician, and not all doctors are assigned to a doctor group. We address these gaps in a later part of the tutorial.

Ignore the *Age* and *Test Score* dimensions, which are for advanced use.

3.5 Using a Collection Property

In DeepSee.Study.Patient, the Allergies property is a collection. This part of the tutorial shows you some ways to use such properties in DeepSee:

1. Access the Architect and display the DeepSee.Study.Patient class.

Tip: Every time you access the Architect, you must select the class to work with. Instead of switching continually between the Architect and the Analyzer, however, you can open two browser tabs and use one for the Architect and the other for the Analyzer. The tutorial does not describe this technique specifically, but it can be a convenient way to work.

2. Add a measure that contains the number of allergies that a patient has. To do so, we use the Allergies property, as follows:

- a. Click **New** in the lower right.
- b. For **Dimension Name**, use Allergy Count.
- c. For **Data Type**, use **Number**.
- d. For **Complex Code**, use the following expression:

```
%this.Allergies.Count()
```

This expression uses the **Count()** method of the Allergies property.

- e. Click **Add**.

The new dimension is added to the table.

- f. Click the **Active** check box in the row for the new measure.
- g. Click **Update** to save this change.

The screenshot shows the 'General' tab of the Architect interface. The 'Dimension Name' field contains 'Allergy Count'. The 'Data Type' dropdown is set to 'Number'. The 'Property' field contains 'DeepSee.Study.Patient'. The 'Complex Code' field contains '%this.Allergies.Count()'. There is a 'Script' button below the 'Complex Code' field. To the right, the 'Listing Field' checkbox is checked, and the 'Transformation Type' dropdown is set to 'None'.

3. Add a dimension that is based on the allergies that the patients have. This dimension will have one member for each distinct allergy. To add this dimension, use the **Manual Child Browse** option as follows:

- a. Click **New** in the lower right.
- b. For **Dimension Name**, use Allergies.
- c. For **Type**, use **Values**.
- d. Click the **Manual Child Browse** tab.
- e. Type the following into **Loop Coding**:

```
for i=1:1:%this.Allergies.Count() do
```

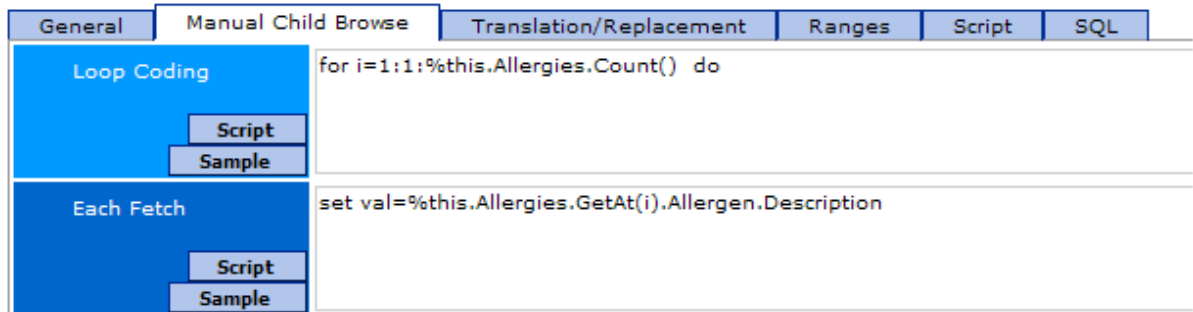
Note: There must be two spaces before **do** and one space after it. There cannot be any line breaks after this.

This syntax controls how to iterate for a given patient, when building the index for this dimension. Specifically, it loops through all the allergies for the patient.

- f. Type the following into **Each Fetch**:

```
set val=%this.Allergies.GetAt(i).Allergen.Description
```

This syntax gets the user-friendly allergen description and places it into the variable *val*. The variable name is hardcoded; that is DeepSee expects you to set this variable.



- g. Click **Add**.

The new dimension is added to the table.

- h. Click the **Active** check box in the row for the new dimension.

- i. Click **Update** to save this change.

- 4. Recompile the base class and rebuild the DeepSee indices, as in the previous part of this tutorial.

- 5. Display the new `Allergies` dimension as rows. Include the total line.

<i>Allergies</i>	Count
additive/coloring agent	466
animal dander	496
ant bites	513
bee stings	511
dairy products	489
dust mites	459
eggs	515
fish	471
mold	489
nil known allergies	509
peanuts	483
pollen	511
shellfish	462
soy	532
tree nuts	459
wheat	502
Grand Total	7,867

Because a patient can have multiple allergies, the grand total can be different from the total number of patients.

Some patients have no allergy data. That is, the `Allergies` property is null. These patients are filtered out of the pivot table when you use the `Allergies` dimension. There are several ways to address this issue; see next section for an example.

The absence of allergy data is not the same as knowing that a patient has no known allergies. Medical information systems often include a way to record that a patient has no known allergies. For example, the system might include an “allergen” called `nil known allergies`. The user asks the patient whether he or she has any allergies, and if the answer is “No,” the user selects the value `nil known allergies`. DeepSee does not assign any special meaning to this string. The dimension treats this “allergen” in the same way as any other allergen.

3.6 Adding Placeholders for Null Members

In an earlier part, we discovered that there are possible problems with the following dimensions:

- Patient Group
- Doctor Group
- Doctor
- Allergies

For all of these, a patient might have a null value for the data used by the dimension. This means that the dimension omits that patient by default. This is correct behavior, but your users might prefer to see a placeholder member in the dimension.

The placeholder member would be used to collect all patients that have a null value for this dimension. The dimension would not omit any patients.

This part of the tutorial shows several ways to address these cases.

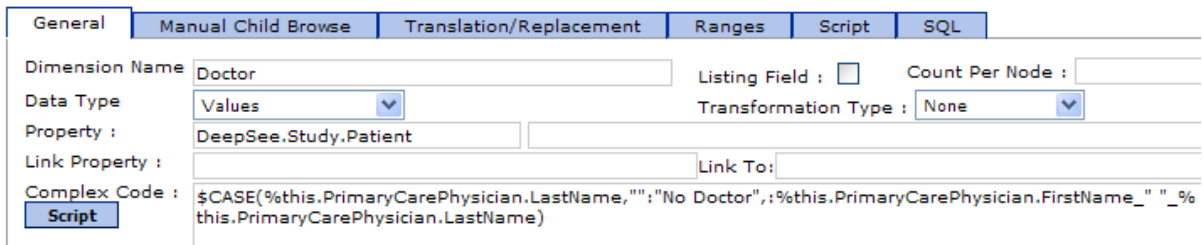
1. Access the Architect and display the `DeepSee.Study.Patient` class.
2. Click the `Patient Group` dimension in the dimension table. Because this dimension is based directly on a source property, we can handle null values for it as follows:
 - a. Click the **Translation/Replacement** tab.
 - b. For **Null Field Replacement**, type `None`.
 - c. Click **Update** to save this change.
3. Repeat the preceding set of steps for the `Doctor Group` dimension.
4. Click the `Doctor` dimension in the dimension table. Because this dimension is based directly on a source expression (that is, using **Complex Code**), that expression must detect and correctly handle null values.

- a. Click the **General** tab.
- b. Edit **Complex Code** as follows:

```
$CASE(%this.PrimaryCarePhysician.LastName,"":"No Doctor",
:%this.PrimaryCarePhysician.FirstName_"_"%this.PrimaryCarePhysician.LastName)
```

Note: Do not include the line break that is shown here. This line break is included only for readability of the documentation.

- c. Click **Update** to save this change.



5. For the `Allergies` dimension, we cannot use either of these techniques, which do not work with **Manual Child Browse**.

First, create a dimension that places patients into two groups: those with recorded allergies and those whose allergies are null:

- a. Create a new dimension as follows:
 - **Dimension Name:** `Allergies Are Null`
 - **Complex Code:**

```
$SELECT(%this.Allergies.Count()=0:"Yes",1:"No")
```

This new dimension gives us an easy way to access the patients that have no recorded data for the `Allergies` property. We will use it as a building block to add a new member to the `Allergies` dimension.

- b. After you add the dimension, click the **Active** check box in the row for the new dimension and click **Update** to save this change.

6. In the next step, we use this new dimension (`Allergies Are Null`) to add a placeholder member to the `Allergies` dimension. (Yes, a dimension can use other dimensions.)
 - a. Click the **Compound Member** tab.
 - b. Click **Add**.
 - c. Enter the following details:
 - **Dimensions:** `Allergies` (This is the name of the dimension to which we are adding a member.)
 - **Condition Name:** `No Information Available` (This is the name of the member that we are adding.)
 - **Dynamic:** Clear this check box so that this dimension is indexed along with the others.
 - d. In the **Filter** area, add the following filter expression:

`[Allergies Are Null = Yes]`

On the **Compound Member** tab, the member definition now looks as follows:

Dimensions :	<input type="text" value="Allergies"/>	<input type="checkbox"/> Dynamic ?
Condition Name :	<input type="text" value="No Information Available"/>	
Filter	<input type="text" value="[Allergies Are Null = Yes]"/>	Clear Edit

- e. Click **Update** to save this change.
7. Recompile the base class and rebuild the DeepSee indices, as in the previous part of this tutorial.
 8. Return to the Analyzer and display the `Patient Group` dimension as rows as we did previously. You should now see something like the following:

<i>Patient Group</i>	<i>Count</i>
A	4,046
B	3,969
None	1,985
Grand Total	10,000

9. Display the `Doctor` dimension as we did previously. Page down as needed. You should now see something like the following:

<i>Doctor</i>	Count
Neil O'Donnell	19
Neil Ragon	21
Neil Zevon	29
Nellie Frost	29
Nellie Hanson	16
Nellie Ihringer	19
Nellie Olsen	35
Nellie Zucherro	24
No Doctor	1,546
Norbert Avery	28
Norbert Goncharuk	17

Also, the grand total should now be 10000.

10. Display the Allergies dimension as rows. Include the grand total. Now you should see something like the following:

<i>Allergies</i>	Count
No Information Available	4,365
additive/coloring agent	466
animal dander	496
ant bites	513
bee stings	511
dairy products	489
dust mites	459
eggs	515
fish	471
mold	489
nil known allergies	509
peanuts	483
pollen	511
shellfish	462
soy	532
tree nuts	459
wheat	502
Grand Total	12,232

Notice that the new member is included. Also notice that the grand total is greater than the number of patients, which is expected now that this dimension is using all the patients.

3.7 Using Ranges, Translations, and Transformations

In this part of the tutorial, we use options that transform the original values for dimensions into other values:

1. Access the Architect and display the DeepSee.Study.Patient class.
2. Add another dimension that is based on the patient age but that uses ranges. The new dimension places patients into ten-year buckets. To do so:
 - a. Double-click Age again in the DeepSee.Study.Patient folder. This step adds this property again to the table on the right.
 - b. Rename the new item to Age Bucket.
 - c. On the **General** tab, change the type to **Values**.
 - d. Click the **Ranges** tab.
 - e. Right-click and then select **Add Line**.
 - f. Type data into the new line as follows:

Caption	From	inclusive	To	inclusive
0-9		<input type="checkbox"/>	9	<input checked="" type="checkbox"/>

- g. Add more lines, one to define each bucket. For the last bucket, use the caption of 80+
- h. Click **Update** to save this change.

Caption	From	inclusive	To	inclusive
0-9		<input type="checkbox"/>	9	<input checked="" type="checkbox"/>
10-19	10	<input checked="" type="checkbox"/>	19	<input checked="" type="checkbox"/>
20-29	20	<input checked="" type="checkbox"/>	29	<input checked="" type="checkbox"/>
30-39	30	<input checked="" type="checkbox"/>	39	<input checked="" type="checkbox"/>

3. Add a dimension, Age Group, that is similar to the preceding dimension but that places patients into larger groups as follows:

Caption	From	inclusive	To	inclusive
0-29		<input type="checkbox"/>	29	<input checked="" type="checkbox"/>
30-59	30	<input checked="" type="checkbox"/>	59	<input checked="" type="checkbox"/>
60+	60	<input checked="" type="checkbox"/>		<input type="checkbox"/>

4. Redefine the Gender dimension to use a transformation. This dimension is based on the following property:

```
Property Gender As %String(DISPLAYLIST = ",Female,Male", VALUELIST = ",F,M");
```

Currently the members of this dimension are `F` and `M`, which are the values stored in the database (that is, the values given by the `VALUELIST` parameter). We can redefine this dimension so that the members are `Female` and `Male`, that is, the values given by the `DISPLAYLIST` parameter. To do so:

- a. Click `Gender` in the dimension table.
- b. On the **General** tab, click **Transformation Type** and then click **External Value**. (**External Value** is a predefined transformation provided by DeepSee.)
- c. Click **Update** to save this change.

Note: The values given by the `VALUELIST` and `DISPLAYLIST` parameters are sometimes referred to as the *internal value* and the *external value*, respectively.

5. Redefine the `Patient Group` dimension to use a translation, as follows:

- a. Click `Patient Group` in the dimension table.
- b. Click the **Translation/Replacement** tab.
- c. Right-click and select **Add Line**.
- d. In the new, empty row, type `A` into **Original Text**.
- e. Type `Group A` into **Translate To**.
- f. Add another line to translate `B` into `Group B`.
- g. Click **Update** to save this change.

General		Manual Child Browse		Translation/Replacement	
Original Text				Translate To	
A				Group A	
B				Group B	

6. Recompile the base class and rebuild the DeepSee indices, as in the previous part of this tutorial.
7. Return to the Analyzer and display the new `Age Bucket` dimension as rows. Include the total line. You should now see something like the following:

<i>Age Bucket</i>	Count
0-9	1,349
10-19	1,433
20-29	1,374
30-39	1,576
40-49	1,485
50-59	1,103
60-69	697
70-79	629
80+	354
Grand Total	10,000

If your grand total is not the same as the record count in your base class, be sure that you defined the ranges correctly.

- Display the new `Age Group` dimension as rows. Include the total line.

You should now see something like the following:

<i>Age Group</i>	Count
0-29	4,156
30-59	4,164
60+	1,680
Grand Total	10,000

- Display the `Gender` dimension as rows:


<i>Gender</i>	Count
Female	5,217
Male	4,783

- Display the `Patient Group` dimension as rows:

<i>Patient Group</i>	Count
Group A	4,046
Group B	3,969
None	1,985

3.8 Adding Measures in the Analyzer

You define the base measures in the Architect, but you can define additional measures in the Analyzer. In particular, this is how to define measures that are aggregated in ways other than by addition. In this part of the tutorial, we examine the measures defined so far and we refine them.

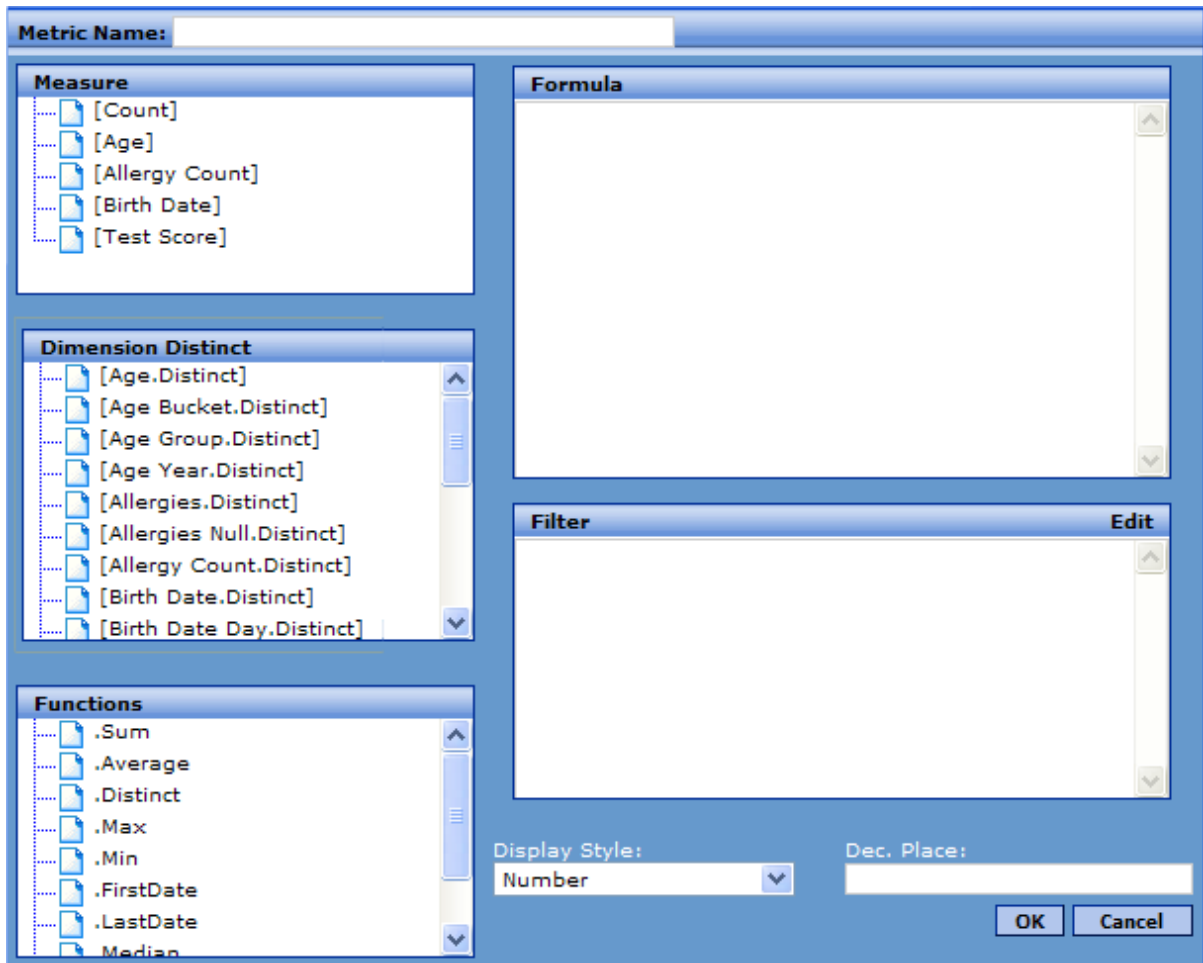
- In the Analyzer, click the new button () in the toolbar.
- In the **Measures** panel, double-click `Count`, `Age`, `Allergy Count`, and `Test Score` and then click **Find**. The Analyzer then displays the cumulative values for these four measures, across the entire subject area:

	Count	Age	Allergy Count	Test Score
All	10,000	363,434.00	8,055.00	592,066.00

The `Count` measure shows the count of records in the subject area; this measure name could be too generic for the users.

The other measures are all aggregated by addition. For example, `Age` is the cumulative age for all patients in the subject area. It would probably be more useful to have additional measures that show average values.

3. Right-click in the **Measures** panel on the left and click **Add Measure**. DeepSee then displays a dialog box where you can define a new measure.



4. Create a new `Patient Count` measure as follows:

- a. For **Metric Name**, use `Patient Count`.
- b. Drag and drop `[Count]` from **Measure** to **Formula**.

Within a measure formula, the syntax `[measure name]` represents the value of a base measure. The formula can use Caché ObjectScript syntax, in addition to DeepSee functions (discussed next).

- c. Click **OK**.

5. Create the new `Average Age` measure as follows:

- a. Right-click in the **Measures** panel and click **Add Measure**.
- b. For **Metric Name**, use `Average Age`.
- c. Drag and drop `[Age]` from **Measure** to **Formula**.
- d. Drag and drop `.Average` from **Functions** to **Formula**.

The formula now is as follows:

```
[Age.Average]
```

- e. For **Dec. Place**, enter 2.
 - f. Click **OK**.
6. Create the additional measures `Average Allergy Count` and `Average Test Score`.

When you drag and drop a measure into a pivot table, DeepSee copies the current definition of the measure into that pivot table. This means that if you redefine the measure in the **Measures** panel (perhaps because you made an error), the pivot table is not affected. This feature can cause a little extra work during development but does keep the pivot table definitions stable.

7. Display the new measures, in the same way that we displayed the base measures:

	Patient Count	Average Age	Average Allergy Count	Average Test Score
All	10,000	36.34	0.81	59.21

By default, DeepSee measures consider all records, including nulls. A null value is treated as zero. So the `Average Test Score` measure might need revision, because the lowest possible score on this test is 50. Some patients do have a null value for the `TestScore` property, and the null scores are treated as 0. So the average test score is lower than it would be if we excluded those patients. In the next section, we redefine the measure `Average Test Score` to correct that problem.

3.9 Defining a Filtered Measure

In this part of this tutorial, we redefine the measure `Average Test Score` so that it filters out the patients that have null test scores.

1. Use the Architect to create a dimension that places the patients into two groups: those with null test scores and those with non-null test scores:

- **Dimension Name:** `Tested?`
- **Data Type:** `Values`
- **Complex Code:**

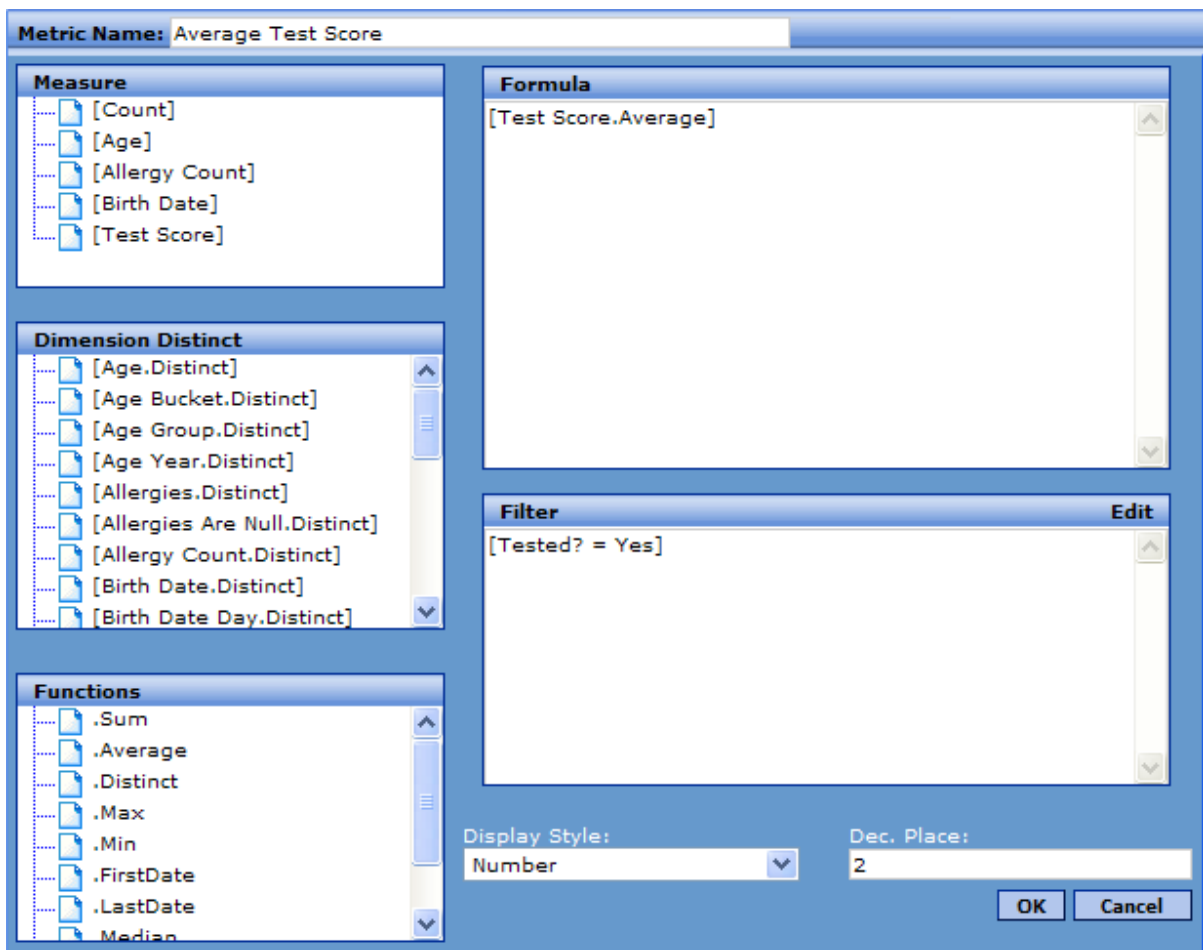
```
$SELECT(%this.TestScore="" : "No", 1 : "Yes")
```


2. After you add the dimension, click the **Active** check box in the row for the new dimension and click **Update** to save this change.
3. Recompile the base class and rebuild the DeepSee indices.
4. Go to the Analyzer.
5. Redefine the `Average Test Score` as follows:

- a. Right-click in the **Measures** panel and click **Edit Measure**.
- b. Click **Edit** in the **Filter** area. Enter the following filter expression:

```
[Tested? = Yes]
```

- c. Click **OK**.



6. In the Analyzer, click the new button () in the toolbar.
7. Double-click the Tested? dimension to display this dimension as rows.
8. Double-click the Test Score, Average Test Score, and Patient Count measures and then click **Find**.

<i>Tested?</i>	<i>Test Score</i>	<i>Average Test Score</i>	<i>Patient Count</i>
No	0.00		2,019
Yes	592,110.00	74.19	7,981

The No row shows data for patients that do not have a test score, and the Yes row shows data for patients that do have a test score.

In the No row, Test Score is 0, as expected. Also Average Test Score is null, which is correct.

3.10 Accessing Other Classes

The DeepSee Architect provides easy access to most of the properties within the BI-enabled class, but we can use other properties, as well, including properties of classes that you can access only via SQL. In this part of the tutorial, we access the data in the `DeepSee.Study.PatientDetails` class from our base class.

1. In the Studio, open the class `DeepSee.Study.Utils.ForModel`.
2. Look at the definition of the **GetFavoriteColor()** method. This method does the following:
 - a. Uses `%this.PatientID` as input. The variable `%this` represents an instance of your base class.
 - b. Uses this input to determine which record or records to access of the other class. The method does this by using an SQL query.
 - c. Accesses the patient's record in `DeepSee.Study.PatientDetails` and gets the `FavoriteColor` field.
 - d. Returns the favorite color (if available) or the string `No Data Available` if not.

The method is as follows:

```
ClassMethod GetFavoriteColor(patientID As %String) As %String
{
  Set ReturnValue="No Data Available"
  Set myquery="SELECT FavoriteColor AS ReturnValue FROM DeepSee_Study.PatientDetails
WHERE PatientDetails.PatientID=?"
  Set rset=##class(%ResultSet).%New( "%DynamicQuery:SQL" )
  Set status=rset.Prepare(myquery)
  If $$$ISERR(status) {Do $System.Status.DisplayError(status)}
  Set status=rset.Execute(patientID)
  If $$$ISERR(status) {Do $System.Status.DisplayError(status)}
  While rset.Next() {
    Set ReturnValue=rset.Data("ReturnValue")
  }
  Quit ReturnValue
}
```

Do not include the line break shown here (where the `myquery` variable is set); this is included only for display reasons.

Note: There is a normal (non-DeepSee) index on the `PatientID` field in `DeepSee.Study.PatientDetails`. This enables the query to run more quickly than it would otherwise.

If your application does include tables that can be related most easily through SQL queries, as in this example, you probably already have indices on the relevant fields. If not, you should add them.

3. Access the Architect and open the class `DeepSee.Study.Patient`.
4. Create a new dimension that invokes this method, as follows:
 - a. Set **Dimension Name** as `Favorite Color`.
 - b. Specify **Complex Code** as follows:

```
##class(DeepSee.Study.Utils.ForModel).GetFavoriteColor(%this.PatientID)
```

This code is executed when you build the indices; see the notes about performance in the previous step.

5. After you add the dimension, click the **Active** check box in the row for the new dimension and click **Update** to save this change.
6. Recompile the base class and rebuild the DeepSee indices.
7. Open the Analyzer and display the new dimension as rows. Include the grand total. Now you should see something like the following:

<i>Favorite Color</i>	<i>Count</i>
Blue	1,253
Green	1,220
No Data Available	2,387
Orange	1,236
Purple	1,315
Red	1,329
Yellow	1,260

3.11 Model Summary

For reference, this section summarizes the model defined in this tutorial.

3.11.1 Dimensions

The model includes the following dimensions:

- **Age Bucket** — Groups patients into ten-year age buckets. For example, a patient who is 17 years old is in the 10–19 bucket.
Each patient belongs to exactly one member of this dimension.
- **Age Group** — Groups patients into three buckets by age. For example, a patient who is 17 years old is in the 0–29 bucket.
Each patient belongs to exactly one member of this dimension.
- **Age Year** — Groups patients into one-year age buckets.
Each patient belongs to exactly one member of this dimension.
- **Allergies** — Groups patients by allergy. A given patient can have any number of allergies and thus can belong to any number of the members of this dimension.
This dimension includes the member `No Information Available`, which contains all patients with null allergy data.
- **Allergies Are Null** — Groups patients into two members. The member `Yes` contains all patients that have null allergy data; the member `No` contains all the other patients. This dimension is used by the `Allergies` dimension.
- **Birth Date** — Groups patients by birth date. This is a standard DeepSee date-type dimension. Each patient belongs to exactly one member of each variation of this dimension. There are no nulls in this data.
- **Doctor** — Groups patients by primary care physician.
This dimension includes the member `No Doctor`, which applies to any patient with no recorded primary care physician.
Each patient belongs to exactly one member of this dimension.

Note: Unlike the other dimensions, the `Doctor` dimension can have a very large number of members, depending on the size of your data set. In a real-world implementation, it is unlikely that you would create a dimension at such a low level. This tutorial uses this dimension because it provides a useful demonstration of **Complex Code**.

- `Doctor Group` — Groups patients by the assigned group of the primary care physician.
This dimension includes the member `None`, which applies to any doctor who was not assigned to a doctor group.
Each patient belongs to exactly one member of this dimension.
- `Favorite Color` — Groups patients by favorite color.
This dimension includes the member `No Data Available`, which applies to any patient with no recorded favorite color.
Each patient belongs to exactly one member of this dimension.
- `Gender` — Groups patients by gender. Each patient belongs to exactly one member of this dimension. There are no nulls in this data.
- `Home City` — Groups patients by home city. Each patient belongs to exactly one member of this dimension. There are no nulls in this data.
- `Home ZIP` — Groups patients by home ZIP code. Each patient belongs to exactly one member of this dimension. There are no nulls in this data.
- `Patient Group` — Groups patients by the assigned patient groups.
This dimension includes the member `None`, which applies to any patient who was not assigned to a patient group.
Each patient belongs to exactly one member of this dimension.
- `Tested?` — Groups patients into two members. The member `No` contains all patients that have a null test score; the member `Yes` contains all the other patients. This dimension is used by the `Average Test Score` measure.

3.11.2 Measures

The model contains the following measures defined in the Architect:

- `Count` — The default measure, this displays the number of patients. Also see `Patient Count`.
- `Age` — Displays the cumulative age of the patients.
- `Allergy Count` — Displays the cumulative count of allergies of the patients.
- `Test Score` — Displays the cumulative test score of all patients.

The model contains the following measures defined in the Analyzer:

- `Average Age` — Displays the average age of the patients.
- `Average Allergy Count` — Displays the average allergy count, per patient, considering all patients.
- `Average Test Score` — Displays the average test score, per patient, considering only the patients that have non-null test scores.
- `Patient Count` — Displays the number of patients.

3.11.3 Listing Fields

The model contains the following dimension-type listing fields:

- `Allergy Count` — Displays the number of allergies that the patient has.
- `Age` — Displays the age of the patient.
- `Birth Date` — Displays the birth date of the patient.
- `Gender` — Displays the gender of the patient.
- `Test Score` — Displays the test score of the patient.

The model also contains one independent listing field:

- `PatientID` — Displays the ID of the patient.

3.11.4 Detail Lists

The model contains one detail list, named `Basic Details`, which contains all the listing fields. It is available to all roles.

3.12 Additional Exercises for the Reader

The preceding sections are meant to help you become familiar with the Architect and common tasks you perform there. To continue the learning experience, create the following elements. Some of these are variations of the elements you have already created; these variations are suggested to give you experience with different forms of data. Some elements are new.

Element Type	Element Name	Comments
Dimension	<code>Allergy Severities</code>	Use the same technique that we used for allergies.
Dimension	<code>Birth Date TimeStamp</code>	Use the <code>BirthDateTimeStamp</code> property.
Dimension	<code>Birth Time</code>	Display the birth time in hour buckets. Use the <code>BirthTime</code> property.
Dimension	<code>Birth Time from TimeStamp</code>	Display the birth time in hour buckets. Use the <code>BirthDateTimeStamp</code> property.
Dimension	<code>Diagnoses</code>	Use the <code>Diagnoses</code> property.
Dimension	<code>Diagnoses As Array</code>	Use the <code>DiagnosesAsArray</code> property.
Dimension	<code>Diagnoses As Children</code>	Use the <code>DiagnosesAsChildren</code> property.
Dimension	<code>Diagnoses as LB</code>	Use the <code>DiagnosesAsLB</code> property.
Dimension	<code>Diagnoses as Many</code>	Use the <code>DiagnosesAsMany</code> property.
Dimension	<code>Industry</code>	Use the <code>Industry</code> property in <code>DeepSee.Study.PatientDetails</code> .
Dimension	<code>Profession</code>	Use the <code>Profession</code> property in <code>DeepSee.Study.PatientDetails</code> .
Measure	<code>Minimum Test Score</code>	Use the <code>TestScore</code> property. Display the lowest test score for any set of patients. Note that the minimum possible score for this test is 50.
Measure	<code>Maximum Test Score</code>	Use the <code>TestScore</code> property. Display the highest test score for any set of patients.

Element Type	Element Name	Comments
Measure	Encounter Count	Display the number of encounters for any set of patients.
Measure	Average Encounter Count	Display the average encounter count per patient, for any set of patients.

Note that by default, the sample does not include encounter data. To add encounter data, execute the following command in the Terminal:

```
Do ##class(DeepSee.Study.PatientEncounter).AddEncounters()
```


4

Creating Pivot Tables


This chapter presents a tutorial in multiple parts:

1. [Creating a basic pivot table](#)
2. [Using additional basic options](#)
3. [Specifying the sort order of members in a pivot table](#)
4. [Using options for ranking and nesting](#)
5. [Using advanced layout options](#)

This tutorial uses the data from the [previous chapter](#). However, you can adapt the steps as needed for your own data model.

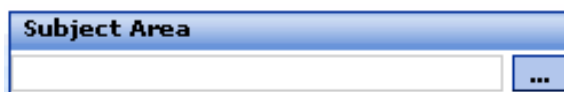
4.1 Creating a Basic Pivot Table

It is quick and easy to create pivot tables with the Analyzer, but there are many shortcuts and many variations. This section demonstrates the basic steps.

1. Log into DeepSee as [described earlier](#).
2. Click **Main > Analyzer**.
3. Click the new button () to create a new pivot table.
4. Select a *subject area*. The subject areas have already been defined; their purpose is to partition data for use by different user roles.

To select a subject area:


- a. Click the browse button (...) in the **Subject Area** panel.




The system lists the subject areas that are available to you, according to the role you used when you logged in.

- b. Click **Patients** and then click **OK**. The subject area is then shown in the **Subject Area** panel, and its contents are now available within the Analyzer, in the panels on the left. These panels list the available dimensions, computations, measures, and other elements in this subject area.

The `Patients` subject area contains information about patients; it includes information such as their allergies, primary care physicians, home city and ZIP code, favorite color, and so on. For an overview of these model elements, see “[Model Summary](#)” in the preceding chapter.

5. In the toolbar, click the autofind icon (.

The icon changes to look like this: .

Tip: If you do not enable autofind, you must click **Find** to display the pivot table, each time you redefine the pivot table.

6. Now specify the *measures* to display in the pivot table. Measures are data that can be displayed in the body or main area of the pivot table.

- a. Drag `Patient Count` from the **Measures** panel and drop it in the **Metrics** panel.

As soon as you do, the pivot table is updated as follows:

	Patient Count
All	10,000

- b. Drag and drop `Average Age` as well. When you do, the pivot table is updated as follows:

	Patient Count	Average Age
All	10,000	36.34

Tip: Alternatively, you can double-click each measure, one at a time.


7. Specify how to break down the measures, which are currently aggregated across the entire subject area. For this example, drag `Home City` from the **Dimension** panel and drop it in the **Row** panel. When you do, the pivot table is updated as follows:

<i>Home City</i>	Patient Count	Average Age
Cedar Falls	1,127	36.24
Centerville	1,081	37.15
Cypress	1,185	35.63
Elm Heights	1,088	35.41
Juniper	1,104	37.30
Magnolia	1,073	36.54
Pine	1,115	36.27
Redwood	1,136	36.28
Spruce	1,091	36.34

Each city is shown in a separate row. A dimension contains *members*, so the cities are members of the `Home City` dimension. The pivot table shows separate data for each member of this dimension.




8. Save the pivot table and give it a name. Here, we are saving the underlying query that retrieves the data, along with the information needed to display it the way you chose. We are not saving the data itself.

To save the pivot table:

- a. Click the save button () and then click **Save Pivot**.
- b. For **Name**, type `Count` and `Age by City`.
- c. For **Notes**, optionally type any notes.
- d. Click the browse button (...) next to **Folder** and select a folder such as **Default**. Then click **OK**.
- e. Click **OK** to accept the name.
- f. Click **OK** on the confirmation dialog box.

You have successfully created a reusable pivot table that you can include in your applications. Later, when you create dashboards, you can insert this pivot table, place it, and resize it as needed. When the pivot table runs, it retrieves the current data as instructed by the query.

9. It is worthwhile to develop a formal understanding of what you are viewing. Compare the pivot table with the options you used.

Row*	Column	Metrics*
 Home City		 Patient Count  Average Age

Note the following points:

- The green cells display labels, and the white (and off-white) cells display data.
- The measures (`Patient Count` and `Average Age`) are displayed as columns.


To understand the contents of a given data cell, use the information given by the corresponding labels. For example, consider the cell in the `Cedar Falls` row, in the `Patient Count` column. This cell displays the total number of patients whose home city is Cedar Falls.



Similarly, consider the `Average Age` column for the `Cedar Falls` row. This cell displays the average age of patients whose home city is Cedar Falls.

- Data cells display aggregate data, and the aggregation can be performed in different ways. For `Patient Count`, DeepSee sums the numbers. For `Average Age`, DeepSee averages the numbers. Other aggregations are possible.

4.2 Additional Basic Options

In this part of the tutorial, we explore additional basic options:

1. In the Analyzer, click the new button () in the toolbar.
2. If autofind is not enabled, enable it.

Icon	Meaning
	Autofind is enabled.
	Autofind is disabled.

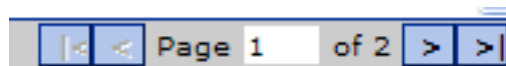
3. Drag `Patient Count` from the **Measures** panel and drop it in the **Metrics** panel.
4. Click the plus sign next to `Birth Date`. This expands the folder to show the birth date variations (for any date-type dimension, the **Dimension** panel displays a collapsible folder containing multiple variants).



5. Drag `Birth Date Year` to the **Row** panel. DeepSee updates the pivot table as follows:

<i>Birth Date Year</i>	Patient Count
1910	15
1911	15
1912	14
1913	8
1914	3
1915	10
1916	10
1917	10
1918	16
1919	10
1920	12

6. Notice that this pivot table spans multiple pages. The lower right of the page shows buttons that you can use to move from page to page:



7. Now drag Gender to the **Column** panel. DeepSee updates the pivot table as follows:

	Female	Male
<i>Birth Date Year</i>	Patient Count	Patient Count
1910	15	
1911	15	
1912	14	
1913	8	
1914	3	
1915	10	
1916	10	
1917	6	4
1918	6	10
1919	6	4
1920	8	5


8. Compare this pivot table to the previous one. Because we added Gender to the columns, the patient count is now broken out and grouped by the patient gender. That is, the `Female > Patient Count` column shows the count of female patients, and the `Male > Patient Count` column shows the count of male patients. As before, each row shows the number of patients born in a given year.

Also notice that the numbers in this pivot table agree (as they should) with the numbers in the previous pivot table. For example, the previous pivot table indicated that 15 patients were born in 1910. *This* pivot table tells us that all 15 of those are female.

9. Save this pivot table, using the name `Patient Counts by Birth Year and Gender`.
10. Let us create another variation. Right-click in the **Row** panel or the **Column** panel and then click **Swap between Row and Column**. As soon as you do, the pivot options are changed, and the pivot table is immediately updated.

	1910	1911	1912	1913	1914	1915	1916
<i>Gender</i>	Patient Count	Patient Count	Patient Count	Patient Count	Patient Count	Patient Count	Patient Count
Female	15	15	14	8	3	10	
Male							

This pivot table displays the same data as the previous one, but has the dimensions reversed. Notice that the measures are still displayed as columns.

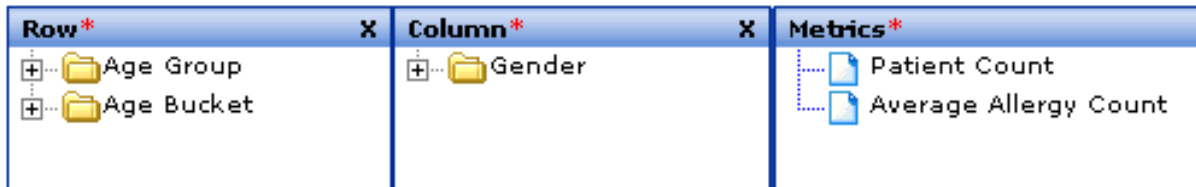
11. Let us create a pivot table that uses multiple dimensions as a row or column:
- Click the new button () in the toolbar.
 - Drag `Age Group` to the **Row** panel.
 - Drag `Age Bucket` to the **Row** panel.

- d. Drag Gender to the **Column** panel.
- e. Drag Patient Count and Average Allergy Count to the **Metrics** panel.

The pivot table looks like the following:

		Female		Male	
<i>Age Group</i>	<i>Age Bucket</i>	Patient Count	Average Allergy Count	Patient Count	Average Allergy Count
0-29	0-9	661	0.79	688	0.79
	10-19	705	0.85	728	0.80
	20-29	692	0.83	682	0.82
30-59	30-39	800	0.76	776	0.82
	40-49	728	0.81	757	0.76
	50-59	568	0.77	535	0.80
60+	60-69	389	0.89	308	0.84
	70-79	368	0.77	261	0.77
	80+	235	0.86	119	0.80

12. Compare this pivot table with the options that define it.



Notice the following:

- The age groups and age buckets are shown as rows.
- Because the Age Group dimension is listed first, the age groups are the outer row dimension.
- Because Gender is used as columns, this pivot table contains an outer column for each gender.
- The measures are shown as data columns, within each gender.


You can include any number of dimensions in the **Row** panel and any number of dimensions in the **Column** panel. The results are analogous to this example.

- 13. Save this pivot table, using the name Age Groups and Buckets.
- 14. Let us consider how the *order* of dimensions affects a pivot table. In the **Row** panel, right-click Age Bucket and click **Move Up**. As soon as you do, the pivot table is updated as follows:

		Female		Male	
<i>Age Bucket</i>	<i>Age Group</i>	Patient Count	Average Allergy Count	Patient Count	Average Allergy Count
0-9	0-29	661	0.79	688	0.79
10-19	0-29	705	0.85	728	0.80
20-29	0-29	692	0.83	682	0.82
30-39	30-59	800	0.76	776	0.82
40-49	30-59	728	0.81	757	0.76
50-59	30-59	568	0.77	535	0.80
60-69	60+	389	0.89	308	0.84
70-79	60+	368	0.77	261	0.77
80+	60+	235	0.86	119	0.80

15. Notice that this pivot table shows the same numbers as the previous, but the rows are now organized differently. Because *Age Bucket* is now the first dimension used in **Row**, it is the outermost grouping. *Age Group* is the next dimension and is included within *Age Bucket*.

16. Now we create a pivot table that includes summary lines (totals and subtotals):

- a. Click the new button () in the toolbar.
- b. Drag *Age Group* to the **Row** panel.
- c. Drag *Age Bucket* to the **Row** panel.
- d. Drag *Gender* to the **Column** panel.
- e. Drag *Patient Count* to the **Metrics** panel.

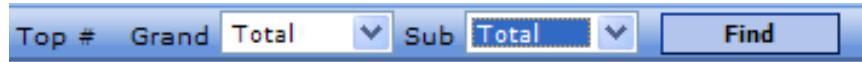
The pivot table now looks like this:

		Female	Male
<i>Age Group</i>	<i>Age Bucket</i>	Patient Count	Patient Count
0-29	0-9	661	688
	10-19	705	728
	20-29	692	682
30-59	30-39	800	776
	40-49	728	757
	50-59	568	535
60+	60-69	389	308
	70-79	368	261
	80+	235	119

- f. In the upper right, click the drop-down list for **Grand** and then click **Total**.



- g. Click the drop-down list for **Sub** and then click **Total**.



Now the pivot table includes both subtotals and the grand total, and it looks like the following:

		Female	Male
<i>Age Group</i>	<i>Age Bucket</i>	Patient Count	Patient Count
0-29	0-9	661	688
	10-19	705	728
	20-29	692	682
	Sub Total	2,058	2,098
30-59	30-39	800	776
	40-49	728	757
	50-59	568	535
	Sub Total	2,096	2,068
60+	60-69	389	308
	70-79	368	261
	80+	235	119
	Sub Total	992	688
Grand Total		5,146	4,854

17. Save this pivot table, using the name `Example with Summaries`.

In this tutorial, we enabled the autofind option so that the pivot table was redisplayed each time we made a design change. With a large data set, however, it can be impractical to constantly redisplay the pivot table. Thus the autofind option is disabled by default.

4.3 Specifying the Sort Order

One of the most common requirements is to control the order in which the members of a dimension are sorted. For any dimension, the default sort order is ascending sort, treating the member names as strings.

This section shows a couple of ways to control the sort order. Note that the sort order affects both the pivot table and the corresponding chart (not discussed in this tutorial).

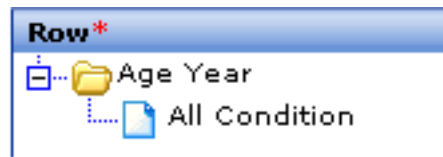
1. Click the new button () in the toolbar.

2. If autofind is not enabled, enable it, as described in the preceding section.
3. Drag `Patient Count` to the **Metrics** panel.
4. Drag `Age Year` to the **Row** panel.

The pivot table now looks like this:

<i>Age Year</i>	Patient Count
0	129
1	141
10	149
11	159
12	169
13	137

5. Click the plus sign (+) next to the `Age Year` dimension in the **Row** panel.



6. Right-click **All Condition** and then select **Edit Condition**.

DeepSee then displays a dialog box where you can customize this dimension, as it appears in this pivot table.

7. For **Sorting**, select **Numeric Ascending**. Do not make any other changes in this dialog box.
8. Click **OK**.

The pivot table now looks as follows:

<i>Age Year</i>	Count
0	129
1	141
2	150
3	140
4	140

The change you made applies only to the current pivot table.

9. Optionally save this pivot table.

Now we look at another pivot table, and we specify sorting in a different way.

10. Click the new button () in the toolbar.

11. Drag `Patient Count` to the **Metrics** panel.
12. Drag `Favorite Color` to the **Row** panel.

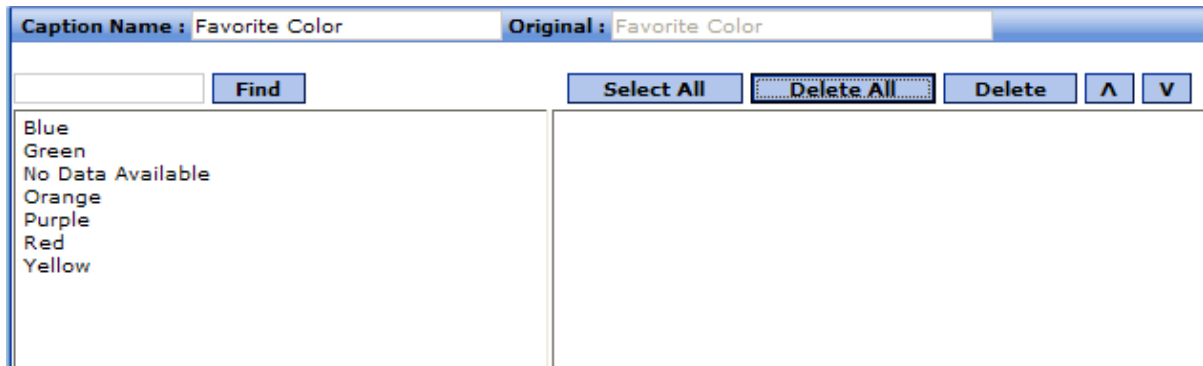
The pivot table now looks like this:

<i>Favorite Color</i>	Patient Count
Blue	1,253
Green	1,220
No Data Available	2,387
Orange	1,236
Purple	1,315
Red	1,329
Yellow	1,260

You might prefer to see the member `No Data Available` at the top or the bottom.

13. Click the plus sign (+) next to the `Favorite Color` dimension in the **Row** panel.
14. Right-click **All Condition** and then select **Edit Condition**.

DeepSee then displays a dialog box where you can customize this dimension, as it appears in this pivot table. The upper area of this dialog box includes a list of the members on the left:



15. Click **Select All**, which moves all members from the left list to the empty list on the right.
16. In the right list, click the member `No Data Available` and then click the up arrow button repeatedly to move this member to the start of the list.

Ignore the **Sorting** option that is shown beneath this list; it has no effect when you select members.

17. Click **OK**.

The change you made applies only to the current pivot table.


Now the pivot table appears as follows:

<i>Favorite Color</i>	Patient Count
No Data Available	2,387
Blue	1,253
Green	1,220
Orange	1,236
Purple	1,315
Red	1,329
Yellow	1,260

18. Optionally save this pivot table.

4.4 Using Ranking and Nesting Options

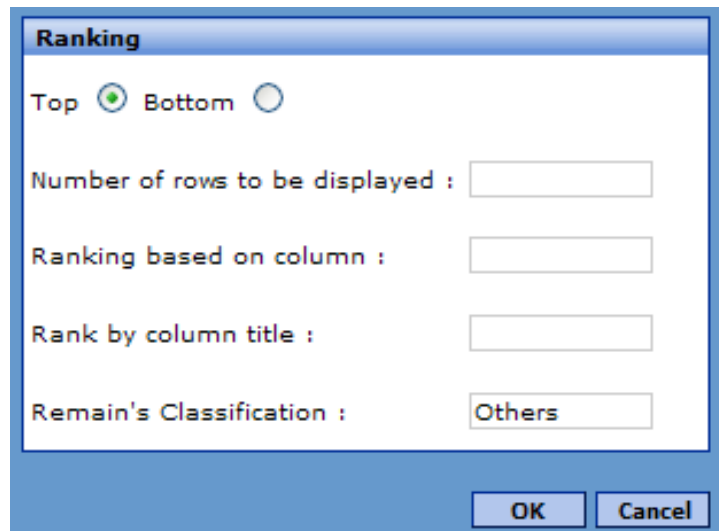
In many cases, you are interested in the details of only the top-ranked or bottom-ranked members. For example, you might be interested only in the top-selling products or the departments with the slowest response times. This section of the tutorial shows how you can easily perform such ranking.

1. Click the new button () in the toolbar.
2. If autofind is not enabled, enable it, as described in the preceding section.
3. Drag `Patient Count` to the **Metrics** panel.
4. Drag `Allergies` to the **Row** panel.

The pivot table now looks like this:

<i>Allergies</i>	Patient Count
No Information Available	4,432
additive/coloring agent	492
animal dander	478
ant bites	514
bee stings	483
dairy products	482
dust mites	497


5. In the upper right, click the **Rank : Top #** link. DeepSee then displays the following dialog box:




6. For **Number of rows to be displayed**, type 4.
7. For **Ranking based on column**, type 1.
8. Click **OK**.

The pivot table now looks like this:

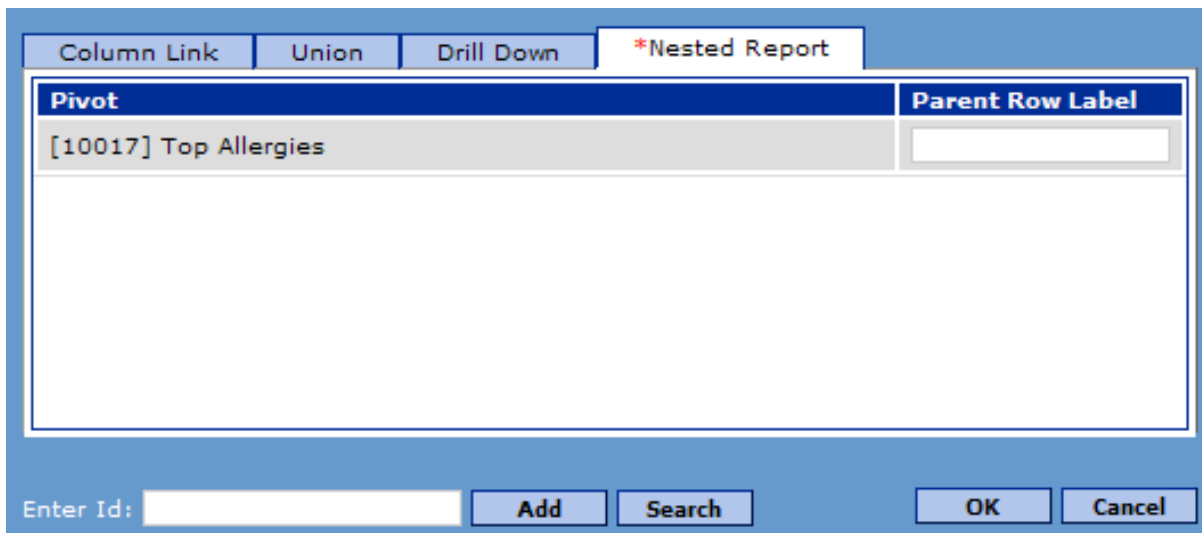
<i>Allergies</i>	Patient Count
No Information Available	4,432
ant bites	514
dust mites	497
peanuts	494
Others	6,101

9. Save this pivot table, using the name Top Allergies.
Now we will use this pivot table within another pivot table to create a more complex layout.
10. Click the new button () in the toolbar.
11. Drag and drop Age Group to the **Row** panel.

<i>Age Group</i>	Count
0-29	4,255
30-59	4,185
60+	1,560

12. Click the link report button () in the toolbar.
DeepSee displays a dialog box that has multiple tabs.

13. Click the **Nested Report** tab.
14. Right-click in the empty area of this tab and then click **Add Line**.
DeepSee displays another dialog box, containing a list of folders.
15. Click the plus sign (+) next to the **Default** folder to expand it.
16. Click the **Top Allergies** pivot table and then click **OK** to select it.




17. Click **OK** to close the dialog box.
18. Save this pivot table, using the name **Top Allergies by Age Group**.
19. Click the open pivot table button (📁), click the **Top Allergies by Age Group** pivot table, and then click **OK**.
DeepSee then displays the pivot table as follows:

Top Allergies by Age Group		
Age Group	Allergies	Patient Count
0-29	No Information Available	1,884
	ant bites	229
	dust mites	219
	additive/coloring agent	214
	Others	2,568
30-59	No Information Available	1,844
	nil known allergies	213
	eggs	211
	tree nuts	207
	Others	2,551
60+	No Information Available	704
	peanuts	91
	fish	85
	bee stings	81
	Others	937

4.5 Using Advanced Layout Options

In this part of the tutorial, we improve the previous pivot table by removing the `No Information Available` member from the age groups and placing it after them instead.

1. First, we redefine the `Top Allergies` pivot table to filter out null allergies:
 - a. Click the open pivot table button () , click the `Top Allergies` pivot table, and then click **OK**.
 - b. Click **Edit** in the **Filter** panel in the lower left.
 - c. Double-click `Allergies Are Null` in the **Dimension** list.
 - d. Double-click the `No` member.

This generates a filter expression as follows:

```
[Allergies Are Null = No]
```

- e. Click **OK**.
- f. In the upper right, click the **Rank : Top 4** link.
- g. For **Number of rows to be displayed**, change the value to 3.
- h. Click **OK**.

- i. Save your changes.

This pivot table now looks like the following:

Top Allergies	
<i>Allergies</i>	<i>Patient Count</i>
ant bites	514
dust mites	497
peanuts	494
Others	6,101


2. Now reopen the Top Allergies by Age Group pivot table. Now it looks like this:

Top Allergies by Age Group		
<i>Age Group</i>	<i>Allergies</i>	<i>Patient Count</i>
0-29	ant bites	229
	dust mites	219
	additive/coloring agent	214
	Others	2,568
30-59	nil known allergies	213
	eggs	211
	tree nuts	207
	Others	2,551
60+	peanuts	91
	fish	85
	bee stings	81
	Others	937

Now we want to add a row that looks like this:

All Ages	No Information Available	4,432
----------	--------------------------	-------

To add this row, we will define another pivot table and append to the end of the first one.

3. Define a new pivot table as follows:
 - a. Click the new button () in the toolbar.
 - b. Drag and drop Age Group to the **Row** panel.

- c. Drag and drop *Allergies* to the **Row** panel.
- d. Click the plus sign (+) next to *Age Group*.
- e. Right-click **All Condition** and then click **Edit Condition**.
- f. In **Compound Name**, in the bottom area of the dialog box, specify *All Ages*.

In this step and the next step, we define a custom member of this dimension. This member will select all data.

- g. For **Compound Query**, type the following filter expression, which uses a wildcard:

[Age Group ?= *]

- h. Click **Add Compound**.

This step places the new custom member in the list in the upper right, which means that DeepSee will display only this member.

- i. Click **OK**.

Now this pivot table looks as follows:

<i>Age Group</i>	<i>Allergies</i>	Count
All Ages	No Information Available	4,432
	additive/coloring agent	492
	animal dander	478
	ant bites	514
	bee stings	483
	dairy products	482
	dust mites	497
	eggs	463
	fish	471
	mold	449
	nil known allergies	470
	peanuts	494
	pollen	489
	shellfish	460
	soy	442
	tree nuts	469
wheat	453	

Now we need to filter out the records that have non-null allergy data.

- j. Click **Edit** in the **Filter** panel in the lower left.
- k. Double-click *Allergies Are Null* in the **Dimension** list.
- l. Double-click the *Yes* member.

This generates a filter expression as follows:

[Allergies Are Null = Yes]


- m. Click **OK**.
- n. Save this pivot table, using the name All Age Groups - No Allergy Data

This new pivot table looks as follows:

<i>Age Group</i>	<i>Allergies</i>	<i>Count</i>
All Ages	No Information Available	4,432

4. Now reopen the Top Allergies by Age Group pivot table, which currently looks like this:

Top Allergies by Age Group		
<i>Age Group</i>	<i>Allergies</i>	<i>Patient Count</i>
0-29	ant bites	229
	dust mites	219
	additive/coloring agent	214
	Others	2,568
30-59	nil known allergies	213
	eggs	211
	tree nuts	207
	Others	2,551
60+	peanuts	91
	fish	85
	bee stings	81
	Others	937

5. Click the link report button () in the toolbar.
DeepSee displays a dialog box that has multiple tabs.
6. Click the **Union** tab.
7. Right-click in the empty area of this tab and then click **Add Line**.
DeepSee displays another dialog box, containing a list of folders.
8. Click the plus sign (+) next to the **Default** folder to expand it.
9. Click the All Age Groups - No Allergy Data pivot table and click **OK**.
10. Click **Match Column** and then click **OK**.
11. Click **Find**.

The pivot table now looks like this:

Top Allergies by Age Group		
Age Group	Allergies	Patient Count
0-29	ant bites	229
	dust mites	219
	additive/coloring agent	214
	Others	2,568
30-59	nil known allergies	213
	eggs	211
	tree nuts	207
	Others	2,551
60+	peanuts	91
	fish	85
	bee stings	81
	Others	937
All Ages	No Information Available	4,432

12. Save the pivot table.

4.6 Summary of Pivot Tables

In this part of the tutorial, we created the following pivot tables:

- Count and Age by City
- Patient Counts by Birth Year and Gender
- Example with Summaries
- Top Allergies
- All Age Groups - No Allergy Data
- Top Allergies by Age Group

5

Creating KPIs

A key performance indicator (KPI) is an aggregate value that you can display on its own in a speedometer or label on a dashboard. KPIs are based directly on measures or on formulas that combine measures.

This part of the tutorial shows you how to create KPIs. It includes the following sections:

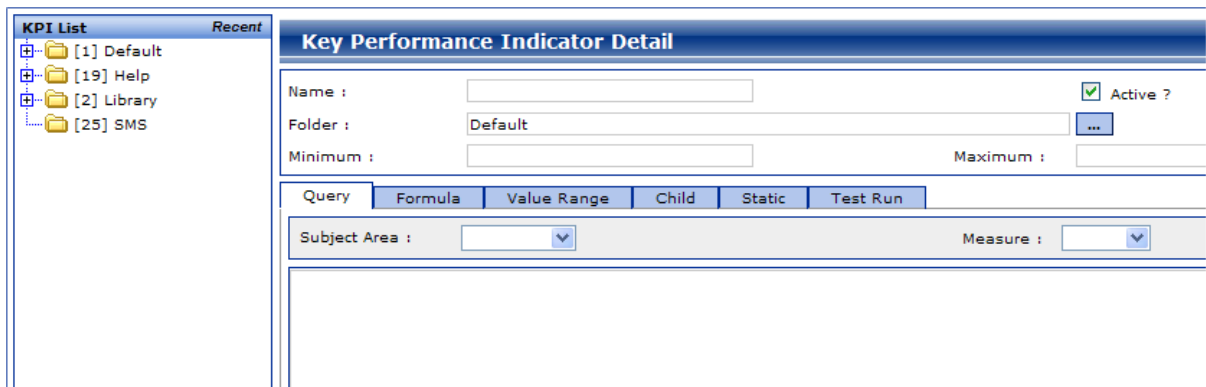
1. [Creating simple KPIs](#)
2. [Creating a string KPI](#)

In the [next part of the tutorial](#), we use these KPIs on a dashboard.

5.1 Creating Simple KPIs

In this part of the tutorial, we create a couple of simple KPIs:

1. Log into DeepSee as described [earlier in this book](#).
2. Click **Main > KPI Setup**. DeepSee displays a page like the following:



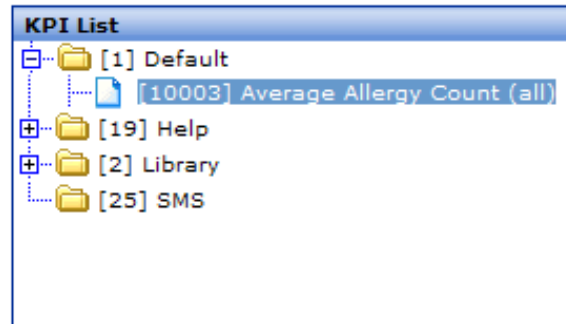
The screenshot shows the 'Key Performance Indicator Detail' form. On the left is a 'KPI List' sidebar with a 'Recent' tab and a tree view containing folders: [1] Default, [19] Help, [2] Library, and [25] SMS. The main form has the following fields and controls:

- Name :** A text input field.
- Folder :** A text input field containing 'Default' and a dropdown arrow.
- Minimum :** A text input field.
- Maximum :** A text input field.
- Active ?** A checked checkbox.
- Query** section with tabs: **Formula**, Value Range, Child, Static, Test Run.
- Subject Area :** A dropdown menu.
- Measure :** A dropdown menu.

3. Add a KPI as follows:
 - a. For **Name**, type Average Allergy Count.
 - b. For **Subject Area**, select the Patients subject area.
 - c. For **Measure**, select the Average Allergy Count measure.
 - d. Click **Add**.

- e. Click **OK**.

DeepSee creates the KPI. In the left area of the page, DeepSee displays the new KPI within the **Default** folder, as follows:



The number shown in square brackets is the ID of the KPI; make a note of this for future use.

4. Click **New**.
5. Add another KPI as follows:
 - a. For **Name**, type `Average Age`.
 - b. For **Folder**, click the browse button (...) and then click the **Default** folder.
 - c. For **Subject Area**, select the `Patients` subject area.
 - d. For **Measure**, select the `Average Age` measure.
 - e. Click **Add**.
 - f. Click **OK**.
 - g. Make a note of the ID of the KPI that you just added.

5.2 Creating a String-type KPI

Measures and KPIs can contain string or date values, instead of numeric values. A string KPI can be a useful form of aggregate data.

For example, suppose that we are interested in the average age of the patients, and we want to convert that number to a string and display that in a label on the dashboard.

1. First, we must create a suitable measure. To do so:
 - a. Access the Analyzer.
 - b. In the **Measures** panel, right-click and select **Add Measure**.
 - c. For **Metric Name**, type `Average Age (string)`.
 - d. For **Formula**, type the following:


```
$SELECT([Age.Average]<21:"Youths",[Age.Average]<55:"Adults",1:"Seniors")
```
 - e. For **Display Style**, select **String**.
 - f. Click **OK**.

2. Test this measure in the Analyzer to ensure that it does display strings. To do so, create a pivot table that uses the measure. For example:

<i>Age Group</i>	Average Age (string)
0-29	Youths
30-59	Adults
60+	Seniors

3. Click **Main > KPI Setup**.
4. Add a KPI as follows:
 - a. For **Name**, type *Average Age (string)*.
 - b. For **Subject Area**, select the *Patients* subject area.
 - c. For **Measure**, select the *Average Age (string)* measure.
 - d. Click **Add**.
 - e. Click **OK**.
 - f. Make a note of the ID of the KPI that you just added.

6

Creating Dashboards

This part of the tutorial shows you how to create dashboards. It includes the following sections:

1. [Creating a simple dashboard with a filter](#)
2. [Adding a speedometer that displays a KPI](#)
3. [Creating a dashboard with multiple filters](#)
4. [Adding a label that displays a string KPI](#)

6.1 Creating a Simple Dashboard with a Filter

In the first part of the tutorial, we create a simple dashboard that contains a pivot table and a drop-down list (called a combo box) that filters it.


1. Log into DeepSee as described [earlier in this book](#).
2. Click **Main > Open Dashboard** and then click **New**.

DeepSee displays a dialog box where you specify the basic properties of the new dashboard.

3. In this dialog box, specify the following basic information:
 - For **Board Name**, type the name of this dashboard: `Dashboard Test 1`.
 - For **Folder**, click the browse button (...) and select the **Default** folder.
4. Click **OK**.

DeepSee then displays the dashboard in edit mode.


5. Add a pivot table to this dashboard as follows:

- a. Click the add pivot table button ().
- b. In the **List Of Pivot** section, expand folders as needed and click the `Count and Age by City` pivot table
- c. Click **OK**.

The pivot table is added in the upper left corner of the dashboard.

- d. Drag and drop the pivot table to the desired location.
- e. Resize the pivot table as needed.

6. Add a combo box to this dashboard as follows:


- a. Click the add combo box button ()
- b. For **Subject Area**, click the browse button (...), click the `Patients` subject area, and click **OK**. The dimensions in this subject area are then displayed within **Dimensions**.
- c. For **Dimensions**, double-click the `Home ZIP` dimension.

DeepSee updates the right area as follows:

Selected	
Subject Area	Patients
Dimension	Home ZIP

- d. Click **OK**.
The combo box is added in the upper left corner of the dashboard.
- e. Drag and drop the combo box to the desired location.
- f. Resize the combo box as needed.

7. Add a label as follows:

- a. Click the add label button ()
- b. For **Normal Display**, type `Patient ZIP Code:`
- c. Click **Font**.
- d. For **Font Style**, click **Bold**.
- e. For **Font Size**, click 12.
- f. Click **OK** to accept the font changes.
- g. Click **OK**.

The label is added in the upper left corner of the dashboard.

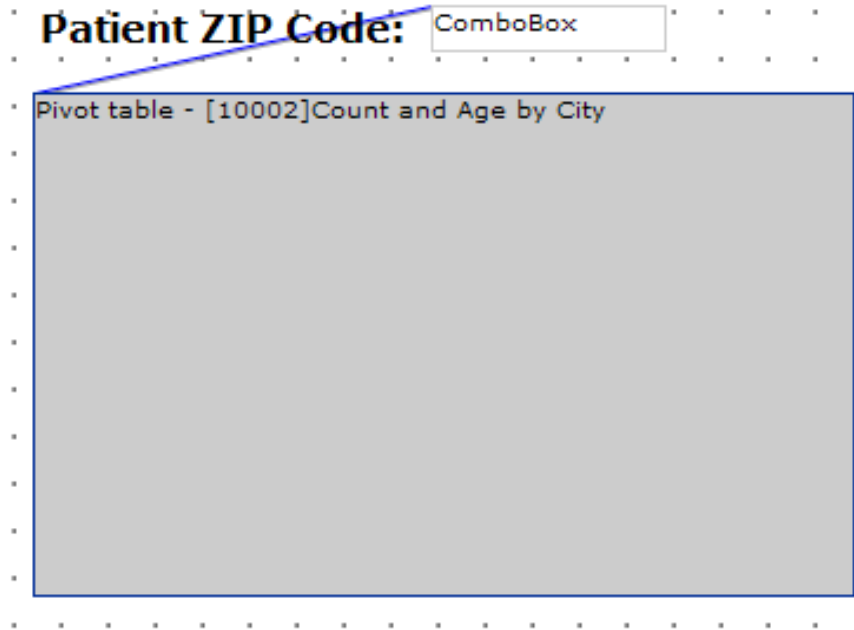
- h. Drag and drop the label to the desired location.
- i. Resize the label as needed.

8. Create a filter link between the combo box and the pivot table, as follows:

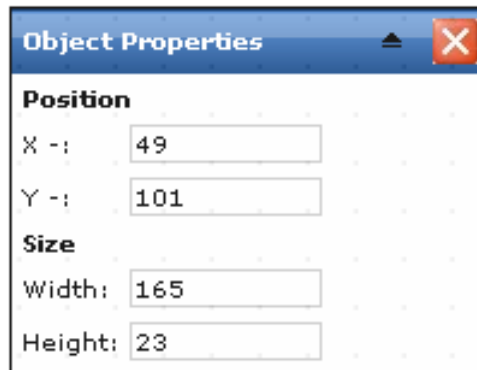
- a. Right-click the combo box and then click **Set as Filter**.
- b. Right-click the pivot table and then click **Apply Filter to this object**. When you do so, a blue line is drawn between the two elements.


9. Create a refresh link between the combo box and the pivot table, as follows:


- a. Right-click the combo box and then click **Set as Refresh Trigger**.
- b. Right-click the pivot table and then click **Set as Refresh Target**. When you do so, a gray line is drawn between the two elements. The line is very close to the blue line and might be difficult to see.



10. Look for the **Object Properties** dialog box, which looks like this:



If this dialog box is not currently displayed, click the show object properties box () icon in the right side of the toolbar.

11. Click the label and make a note of its position.
12. Click the combo box. Then edit its y-position to be the same as the label.
13. Click the pivot table. Then edit its x-position to be the same as the label.
14. Click the save dashboard icon () . Then click **OK**.
15. Click the close button (X) in the upper right of the Designer.
DeepSee displays the new dashboard in view mode.
16. Refresh the browser window.

DeepSee displays something like this:

Patient ZIP Code:

Home City	Patient Count	Average Age
Cedar Falls	1,127	36.24
Centerville	1,081	37.15
Cypress	1,185	35.63
Elm Heights	1,088	35.41
Juniper	1,104	37.30
Magnolia	1,073	36.54
Pine	1,115	36.27


Test the dashboard by selecting a ZIP code from the combo box. The pivot table should be filtered and refreshed immediately to show only data associated with that member:

Patient ZIP Code:


Home City	Patient Count	Average Age
Cypress	1,185	35.63
Magnolia	1,073	36.54
Pine	1,115	36.27

6.2 Displaying a KPI in a Speedometer

In this part of the tutorial, we edit the previous dashboard to display a KPI in a speedometer.

1. Right-click and then click **Edit Dashboard**.
2. Add a speedometer to this dashboard as follows:
 - a. Click the add speedometer button ()
 - b. For **Caption**, type Average Allergy Count.
 - c. For **KPI Override Setting**, click the browse button (...), expand folders as needed, and click the Average Allergy Count KPI.
 - d. Click **OK**.

The speedometer is added in the upper left corner of the dashboard.

- e. Drag and drop the speedometer to the desired location.
 - f. Resize the speedometer as needed.
3. Create a filter link between the combo box and the speedometer, as follows:
 - a. Right-click the combo box and then click **Set as Filter**.
 - b. Right-click the speedometer and then click **Apply Filter to this object**. When you do so, a blue line is drawn between the two elements.
 4. Create a refresh link between the combo box and the speedometer, as follows:
 - a. Right-click the combo box and then click **Set as Refresh Trigger**.
 - b. Right-click the speedometer and then click **Set as Refresh Target**. When you do so, a gray line is drawn between the two elements.
 5. Rearrange the other dashboard elements if needed.
 6. Click the save dashboard icon () . Then click **OK**.
 7. Click the close button (X) in the upper right of the Designer.

DeepSee displays the new dashboard in view mode.
 8. Refresh the browser window.

Now your dashboard might look like this:



Patient ZIP Code:

Home City	Patient Count	Average Age
Cedar Falls	1,185	35.82
Centerville	1,146	36.65
Cypress	1,072	35.56
Elm Heights	1,076	37.70
Juniper	1,088	35.16
Magnolia	1,135	34.98
Pine	1,089	36.74
Redwood	1,102	35.10
Spruce	1,107	35.89

When you choose an item from the combo box, the pivot table and the speedometer should both be automatically filtered and refreshed. For example:



Patient ZIP Code:

Count and Age by City


< Level 0 of 0 > [red dot] [info] [list] [bar chart]

Home City	Patient Count	Average Age
Juniper	1,088	35.16
Spruce	1,107	35.89

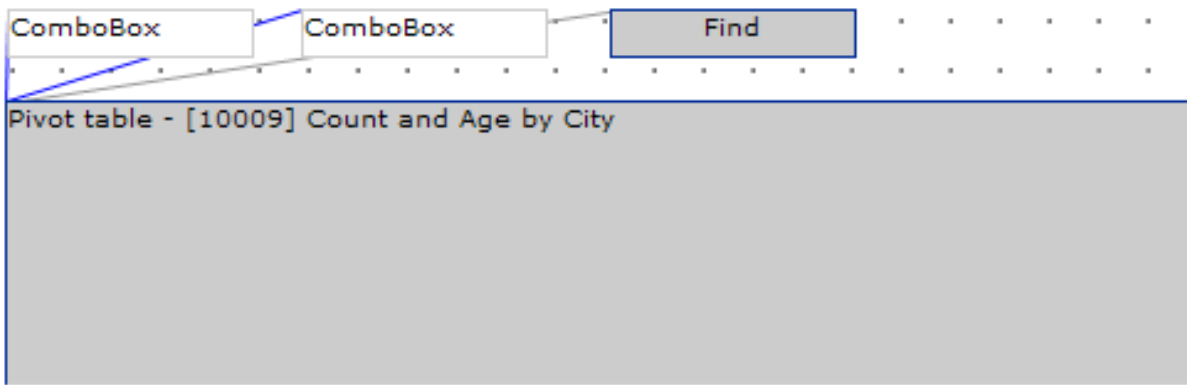
6.3 Adding Multiple Filters


In this part of the tutorial, we create a slightly more complex dashboard that contains multiple combo boxes. Instead of including refresh links between these combo boxes and the pivot table, we add a button and use that as the refresh trigger.

1. Right-click and then click **New Dashboard > Blank**.
2. In the dialog box, specify the following basic information:
 - For **Board Name**, type the name of this dashboard: `Dashboard Test 2`.
 - For **Folder**, click the browse button (...) and select the **Default** folder.
3. Click **OK**.
DeepSee then displays the dashboard in edit mode.
4. Add the `Count and Age by City` pivot table to this dashboard as you did in the previous part of the tutorial.
5. Add a combo box to this dashboard, as you did in the previous part of the tutorial. For this combo box, use the dimension `Age Group`.
6. Add a second combo box as follows:
 - a. Right-click the first combo box and then click **Duplicate**. The copy is placed directly on top of the original.
 - b. Drag and drop the copy.
 - c. Double-click the copy, which displays a dialog box of its properties.
 - d. In this dialog box, double-click a different dimension. In this case, use the dimension `Favorite Color`.

- e. Click **OK**.
7. For each combo box, add a filter link between that combo box and the pivot table, as follows:
 - a. Right-click the combo box and then click **Set as Filter**.
 - b. Right-click the pivot table and then click **Apply Filter to this object**. When you do so, a blue line is drawn between the two elements.
8. Optionally add labels above or next to the combo boxes.
9. Add a button to this dashboard as follows:
 - a. Click the add button icon ().
 - b. For **Normal Display**, type Find
 - c. Click **OK**.
The button is added in the upper left corner of the dashboard.
 - d. Drag and drop the button to the desired location.
10. Create a refresh link between the button and the pivot table, as follows:
 - a. Right-click the button and then click **Set as Refresh Trigger**.
 - b. Right-click the pivot table and then click **Set as Refresh Target**. When you do so, a gray line is drawn between the two elements.

The result looks like the following:



11. Click the save dashboard icon (). Then click **OK**.
12. Click the close button (X) in the upper right of the Designer.
DeepSee displays the new dashboard in view mode.
13. Refresh the browser window.

Depending upon your data, your data model, and your pivot table, DeepSee displays something like this:

▼

▼


Find

Count and Age by City		
Home City	Patient Count	Average Age
Cedar Falls	1,185	35.82
Centerville	1,146	36.65
Cypress	1,072	35.56
Elm Heights	1,076	37.70
Juniper	1,088	35.16
Magnolia	1,135	34.98
Pine	1,089	36.74
Redwood	1,102	35.10
Spruce	1,107	35.89

Test the new dashboard by clicking options from the drop-down lists. Notice that the pivot table is not refreshed until you click **Find**.

6.4 Displaying a KPI in a Label


In this part of the tutorial, we edit the previous dashboard to display a string KPI in a label.

1. Right-click and then click **Edit Dashboard**.
2. Click the add label button ().
DeepSee displays a dialog box in which you specify the details.

3. For **Normal Display**, type the following:

```
$$KPI(10003)
```

For 10003, substitute the ID of the *Average Age (string)* KPI that you [created earlier](#).

4. Click **OK**.
The label is added in the upper left corner of the dashboard.
5. Add filter links between each combo box and the new label.
6. Add a refresh link between the button and the new label.
7. Click the save dashboard icon (). Then click **OK**.
8. Click the close button (X) in the upper right of the Designer.
DeepSee displays the new dashboard in view mode.

9. Refresh the browser window.

Now your dashboard might look like this:

Adults

Count and Age by City < Level 0 of 0 > ... i

Home City	Patient Count	Average Age
Cedar Falls	1,185	35.82
Centerville	1,146	36.65
Cypress	1,072	35.56
Elm Heights	1,076	37.70
Juniper	1,088	35.16
Magnolia	1,135	34.98
Pine	1,089	36.74
Redwood	1,102	35.10
Spruce	1,107	35.89

When you select items from the combo boxes and then click **Find**, DeepSee updates both the pivot table and the label. For example:

Seniors

Count and Age by City < Level 0 of 0 >

Home City	Patient Count	Average Age
Cedar Falls	27	69.33
Centerville	27	71.63
Cypress	33	71.45

7

Using Multiple Subject Areas Together

For simplicity, this book has assumed so far that you are creating and using only a single subject area. However, it is often convenient or even necessary to create and use multiple subject areas in combination. This chapter discusses the following topics:

- [Rules to follow when you create multiple subject areas to use together](#)
- [What it means to use multiple subject areas in a pivot table](#)
- [What it means to use multiple subject areas in a dashboard](#)
- [How to set up the secondary subject area for use in this part of the tutorial](#)
- [How to create an initial version of the linked pivot table shown in this chapter](#)
- [How to create a corrected version of the linked pivot table](#)
- [How to create the dashboard shown in this chapter](#)

7.1 Rules for Creating Multiple Subject Areas

To define multiple subject areas that you can use together as described in this chapter, do the following:

- Identify all the dimensions that are common across the subject areas. Typically, dimensions based on time and location are common, even for unrelated subject areas.

Other dimensions might or might not be common depending on whether the subject areas are related to each other in any meaningful sense. For example, suppose that one subject area represents transactions and another represents the customers who own the transactions. These two subject areas might have common dimensions such as customer, customer class, and so on.

- When you define those dimensions, ensure that the dimension names and sets of member names are the same in all relevant subject areas. (The underlying details of **Complex Code**, transformation options, and so on do not matter. All that matters is that the DeepSee indices work the same way, even if they are created differently.)

7.2 Multiple Subject Areas in a Pivot Table

You can use the **Link** options in the Analyzer to link pivot tables that are defined in different subject areas. For example:

Patient Count and Rainfall by Year		
Date Year	Inches Of Rain	Patient Count
1900	25.74	
1901	13.84	
1902	12.39	
1903	17.20	
1904	16.82	
1905	19.65	
1906	23.82	
1907	16.60	
1908	22.78	
1909	18.19	
1910	17.47	3
1911	17.06	1

This pivot table is a horizontal link of two other pivot tables. The right pivot table (used as the right data column) is defined in the `Patients` subject area and shows the number of patients born in each year. The left pivot table (used as the left data column) is defined in the `City Rainfall` subject area and shows the number of inches of rain that fell in that year.

In order for the resulting pivot table to respond correctly to filters, you must follow the rules listed [earlier in this chapter](#).

7.3 Multiple Subject Areas in a Dashboard

With DeepSee, your overall goal is to create dashboards. A dashboard usually includes one or more data elements (which can be pivot tables, charts, detail listings, or KPIs displayed in dashboards or labels), as well as one or more filters that apply to those data elements.

Within a dashboard, the data elements can belong to different subject areas. If you follow the rules listed [earlier in this chapter](#), the dashboard can include filters that use the dimensions that are common to subject areas. This means that you can create a dashboard like the following:

Year: City:

Patient Count by Home City and Birth Year									
	Cedar Falls	Centerville	Cypress	Elm Heights	Juniper	Magnolia	Pine	Redwood	Spruce
Birth Date Year	Patient Count	Patient Count	Patient Count	Patient Count	Patient Count	Patient Count	Patient Count	Patient Count	Patient Count
1910	3	1		1	3	2		1	1
1911	2	2	1	2	1	1	2		1
1912	3					1			

Rainfall by City and Year									
	Cedar Falls	Centerville	Cypress	Elm Heights	Juniper	Magnolia	Pine	Redwood	Spruce
Year	Inches Of Rain	Inches Of Rain	Inches Of Rain	Inches Of Rain	Inches Of Rain	Inches Of Rain	Inches Of Rain	Inches Of Rain	Inches Of Rain
1900	22.41	19.40	21.81	17.91	25.74	19.77	20.54	14.57	17.78
1901	17.44	18.60	20.04	23.14	13.84	23.44	20.47	22.56	18.88
1902	17.49	19.92	20.26	22.63	12.39	19.04	19.26	25.15	18.98
1903	16.61	23.26	24.37	18.11	17.20	14.65	17.54	24.05	24.07

If the user selects a city and a year from the drop-down lists, the dashboard is filtered as follows:

Year: City:

Patient Count by Home City and Birth Year	
	Elm Heights
Birth Date Year	Patient Count
1910	1

Rainfall by City and Year	
	Elm Heights
Year	Inches Of Rain
1910	17.16

7.4 Setting Up the Initial City Rainfall Subject Area

The rest of this chapter shows you how to define the simple pivot tables and dashboards shown in the previous sections. It also demonstrates the rules given earlier in this chapter.

We start by creating the rainfall data and defining an initial version of the `City Rainfall` subject area:

1. Generate the city rainfall data, by entering the following command in the Terminal, in the `SAMPLES` namespace:

```
do ##class(DeepSee.Study.CityRainfall).GenerateData()
```

Note: This method assumes that the cities have already been created (which occurs when you generate patients). Unless you have already generated patients as described earlier in this book, this method does not generate any data.

2. Access the DeepSee Architect and open the class `DeepSee.Study.CityRainfall`.
3. Define the following elements:

Element Type	Element Name	Details
Dimension	<code>City</code>	Use the <code>City.Name</code> property.
Dimension	<code>Date</code>	Use the <code>MonthAndYear</code> property.
Measure	<code>Inches Of Rain</code>	Use the <code>InchesOfRain</code> property.

4. Define a subject area named `City Rainfall`.
5. Compile the class `DeepSee.Study.CityRainfall` and build the DeepSee indices.

7.5 Creating the Linked Pivot Table


In this section, we create an initial version of linked pivot table:

1. In the Analyzer, go to the `Patients` subject area.
2. Define and save a pivot table as follows:
 - **Row** — `Birth Date Year`
 - **Metrics** — `Patient Count`
 - Name of pivot table — `Patient Count by Year`

Patient Count by Year	
<i>Birth Date Year</i>	Patient Count
1910	12
1911	12
1912	4
1913	3
1914	11

3. In the Analyzer, go to the `City Rainfall` subject area.
4. Define and save a pivot table as follows:
 - **Row** — `Date Year`
 - **Metrics** — `Inches Of Rain`

<i>Date Year</i>	Inches Of Rain
1900	179.93
1901	178.41
1902	175.12
1903	179.86
1904	176.61

5. Then do the following:
 - a. Click the link button () in the toolbar.
 - b. On the **Column Link** tab, right-click and select **Add Line**.
DeepSee then displays a dialog box in which you can select another pivot table.
 - c. Select the pivot table `Patient Count by Year` and click **OK**. The new row is added to the table on the **Column Link** tab.
 - d. Click the **Include Parent Query** check box.
 - e. Click **OK**.
 - f. Use the **Save As** option to save the resulting pivot table with the name `Patient Count and Rainfall by Year`.
6. Click **Find**.
DeepSee displays something like the following:

Patient Count and Rainfall by Year		
<i>Date Year</i>	<i>Inches Of Rain</i>	<i>Patient Count</i>
1900	179.93	
1901	178.41	
1902	175.12	
1903	179.86	
1904	176.61	
1905	177.21	
1906	174.47	
1907	172.40	
1908	178.97	
1909	186.92	
1910	168.95	12
1911	178.61	12

The pivot table looks correct. Next we examine whether it can be filtered correctly.

- Click the **Edit** option in the **Filter** pane and apply a filter like the following:

```
[City = Juniper]
```

- Click **Find**. When you do so, the right data column is removed. This occurs because the filter you have applied removes this column. The `Patients` subject area does not recognize the `City` dimension and thus the patient data is filtered out.

In the next section we correct this problem.

7.6 Adjusting the City Rainfall Subject Area and Pivot Table

As noted earlier in this chapter, two subject areas can respond to the same filter only if the dimension name and the member names are the same in both subject areas.

We could change the dimension names in the `Patients` subject area to be more general (`City` instead of `Home City`, for example). By now, however, you may have created many pivot tables using the current names, and it would be undesirable to have to change those. So instead we will rename the dimensions in the `City Rainfall` subject area:

- In the Architect, access the `City Rainfall` subject area.
- Make the following changes in this subject area:
 - Rename the `City` dimension to `Home City`.
 - Rename the `Date` dimension to `Birth Date`.

3. Compile the class `DeepSee.Study.CityRainfall` and build the `DeepSee` indices.
4. In the Analyzer, open the `Patient Count` and `Rainfall by Year` pivot table.
5. Make the following changes to this pivot table:
 - a. Redefine this pivot table to use `Birth Date Year` instead of `Date Year` for rows.
 - b. While the pivot table is still displayed, click the plus sign next to `Birth Date` in the **Row** pane.
 - c. Right-click **All Condition** and then click **Edit Condition**.
 - d. Edit **Caption Name** to be `Date Year`.
 - e. Click **OK**.
 - f. Save the pivot table.
6. Open the pivot table `Patient Count` and `Rainfall by Year`.
7. Try again to filter the pivot table using the new dimension name:


```
[Home City = Juniper]
```

You should see that both columns are filtered. For example:

Patient Count and Rainfall by Year		
<i>Date Year</i>	<i>Inches Of Rain</i>	<i>Patient Count</i>
1900	25.74	
1901	13.84	
1902	12.39	
1903	17.20	
1904	16.82	
1905	19.65	
1906	23.82	
1907	16.60	
1908	22.78	
1909	18.19	
1910	17.47	3
1911	17.06	1

7.7 Creating a Dashboard That Uses Multiple Subject Areas

In the last part of this tutorial, we create the dashboard shown earlier in this chapter:

1. In the Analyzer, go to the `Patients` subject area.
 2. Define and save a pivot table as follows:
 - **Row** — `Birth Date Year`
 - **Column** — `Home City`
 - **Metrics** — `Patient Count`
 - Name of pivot table — `Patient Count by Home City and Birth Year`
 3. Go to the `City Rainfall` subject area.
 4. Define and save a pivot table as follows:
 - **Row** — `Birth Date Year`
 - **Column** — `Home City`
 - **Metrics** — `Inches Of Rain`
 - Name of pivot table — `Rainfall by City and Year`
 5. Change the caption in the second pivot table as follows:
 - a. Click the plus sign next to `Birth Date Year` in the **Row** pane.
 - b. Right-click **All Condition** and then click **Edit Condition**.
 - c. Edit **Caption Name** to be `Year`.
 - d. Click **OK**.
 6. Save the `Rainfall by City and Year` pivot table again.
 7. Access the Dashboard Designer.
 8. Create a new dashboard.
 9. To the new dashboard, add two pivot tables (`Patient Count by Home City and Birth Year` and `Rainfall by City and Year`).
 10. Add a combo box as follows:
 - a. Click the add combo box button ()
 - b. For **Subject Area**, click the browse button (...), click the `City Rainfall` subject area, and click **OK**.
 - c. For **Dimensions**, double-click the `Birth Date Year` dimension.
 - d. Click **OK**.

The combo box is added in the upper left corner of the dashboard.
 - e. Drag and drop the combo box to the desired location.
 - f. Resize the combo box as needed.
- The `Birth Date Year` dimension of the `City Rainfall` contains more dates than the same dimension in the other subject area. Because we would like to see all possible dates, this is the dimension we use.
11. Add another combo box. This one should use the `Home City` dimension. You can use either subject area to get this dimension.
 12. Add refresh triggers and filter links from these two combo boxes to both of the pivot tables.

13. Add labels.

8

Performing Real-time Updates

The sample enables you to see that DeepSee can display live data. It includes multiple methods that update various data elements. This chapter describes how to see these real-time updates.

8.1 Preparation

To prepare, do the following:

1. Enable incremental updates for this DeepSee installation. To do so, go to **Administrator > Site Configuration** and select the option **ETL > Incremental Index Update**. Then click **Save**.
2. Recompile `DeepSee.Study.Patient`.
3. Make sure that the DeepSee indices are current. If they are not, run **HiRebuild()** in `DeepSee.Study.Patient`.
4. In the Analyzer, create the following simple pivot tables:

Pivot Table Name	Row	Metrics	Other Options
Counts		Patient Count and Encounter Count (if you have created this measure)	
Patient Groups	Patient Group	Patient Count	Select Grand Total
Favorite Colors	Favorite Color	Patient Count	Select Grand Total
Doctors	Doctor Group	Patient Count	Select Grand Total

8.2 Adding and Deleting Patients

To show that the DeepSee indices are updated when you add and delete patients:

1. Display the `Counts` pivot table.
2. In the Terminal, execute the following command to add some patients:

```
do ##class(DeepSee.Study.Patient).AddPatients()
```

The output (in the Terminal) indicates the current patient count.

3. Rerun the pivot table to see the change in the patient count.
4. In the Terminal, execute the following command to delete some patients:

```
do ##class(DeepSee.Study.Patient).DeleteSomePatients()
```

The output (in the Terminal) indicates the current patient count.

Note: Deleting patients is significantly slower compared to the other changes documented in this chapter.

5. Rerun the pivot table to see the change in the patient count, as well as the current patient count.

If you have enabled incremental updates, then when you add, update, or delete an object from `DeepSee.Study.Patient`, the **%OnAfterSaveMethod()** (inherited from `%Bl.Adaptor`) updates the DeepSee indices automatically. The methods **AddPatients()**, and **DeleteSomePatients()** both take advantage of this fact.

8.3 Changing Patient Data

In `DeepSee.Study.Patient`, you can use the method **ChangeSomePatients()** to change assorted properties for a random percentage of patients. This method makes changes as follows:

- The patient is assigned to a different patient group.
- The favorite color of a patient is changed.
- Additional patient encounters are added.

These three changes occur to different randomly selected subsets of patients. For example, a patient's favorite color might be changed, but that patient might not receive any new patient encounters.

To demonstrate that DeepSee recognizes these real-time changes:

1. Display the `Patient Groups` or the `Favorite Colors` pivot table.
2. In the Terminal, execute the following command:

```
do ##class(DeepSee.Study.Patient).ChangeSomePatients(,1)
```

The first argument is the percentage of patients to change; the default is 20 percent.

The second argument is the *rebuild* argument. If this equals 1, the method will update the DeepSee indices for the affected patients. DeepSee performs this update only if you have enabled incremental updates as described earlier.

3. Rerun the pivot table. You should see changes in the number of patients within each patient group and with each favorite color. There should be, however, no overall change in the patient count.

To demonstrate that DeepSee recognizes real-time additions to the encounters, use the `Counts` pivot table. In this case, you should also see an increase in the total encounter count. There should be, however, no overall change in the patient count.

The **ChangeSomePatients()** calls the following methods:

- **ChangePatientGroups()** in `DeepSee.Study.Patient`. This method causes Caché to run **%OnAfterSaveMethod()** and update the indices for the affected patients.
- **ChangePatientDetails()** in `DeepSee.Study.PatientDetails`. This method changes data in `DeepSee.Study.PatientDetails`.

Because `DeepSee.Study.Patient` does not see this class, this method does not cause Caché to run `%OnAfterSaveMethod()`.

Instead, if `rebuild` is 1, the method determines the IDs of the affected patients and calls `zzBuildOne()` for those IDs, updating the DeepSee indices. The relevant section of code is as follows:

```
Set patID=patdetails.PatientID

Set myquery="SELECT ID FROM DeepSee_Study.Patient WHERE PatientID=?"
Set rset=##class(%ResultSet).%New("%DynamicQuery:SQL")
Set status=rset.Prepare(myquery)
If $$$ISERR(status) {Do $System.Status.DisplayError(status) Quit}
Set status=rset.Execute(patID)
If $$$ISERR(status) {Do $System.Status.DisplayError(status) Quit}
While rset.Next() {
    Set id=rset.Data("ID")
    Do ##class(DeepSee.Study.Patient).zzBuildOne(id)
}
```

- `AddEncounters()` in `DeepSee.Study.PatientEncounter`. This method adds records to `DeepSee.Study.PatientEncounter`.

Because `DeepSee.Study.Patient` does not see this class, this method does not cause Caché to run `%OnAfterSaveMethod()`.

Instead, if `rebuild` is 1, the method determines the IDs of the affected patients and calls `zzBuildOne()` for those IDs, updating the DeepSee indices. The technique is similar to the one shown in the previous item.

8.4 Changing Doctor Data

In `DeepSee.Study.Doctor`, you can use the method `ChangeSomeDoctors()` to randomly change the doctor group and patients per week of some doctors.

1. Display the `Doctors` pivot table.
2. In the Terminal, execute the following command:

```
do ##class(DeepSee.Study.Doctor).ChangeSomeDoctors(,1)
```

The first argument is the percentage of doctors to change; the default is 20 percent.

The second argument is the `rebuild` argument. If this equals 1, the method will update the DeepSee indices for the affected patients. DeepSee performs this update only if you have enabled incremental updates as described earlier.

3. Rerun the pivot table. You should see a change in the number of patients with doctors in each group, but no overall change in the patient count.

This method determines the IDs of the affected patients and calls `zzBuildOne()` for those IDs, updating the DeepSee indices. The technique is similar to the one shown previously.

A

Installing the Database for the Sample DeepSee Model

If you reconfigured the SAMPLES namespace as described in “[Setting Up the Environment](#),” the DeepSee model that you have created is in a separate database. You can replace that database with a model database provided by InterSystems.

You might do this if you want to bypass the exercises and simply examine a working DeepSee model.

To obtain and install this database:

1. Access the InterSystems ftp site: [ftp.intersystems.com](ftp://ftp.intersystems.com).
2. Go to the directory `pub/deepsee/`.
3. Download the file `DeepSee.Study.Model.zip`.
4. Uncompress this file, which includes `CACHE.DAT` and `readme.txt`.
5. If your operating system is big-endian, use the `cvendian` tool to convert the downloaded `CACHE.DAT` to a big-endian database. See the article *Using cvendian to Convert Between Big-endian and Little-endian Systems*.

The downloaded `CACHE.DAT` was created on a Windows machine and is little-endian.

6. Stop Caché.
7. Find the directory that contains your model database file (the file used by the database `SampleDSModel`, if you used that name).
8. Rename your `CACHE.DAT` file, for example, to `SaveCACHE.DAT`.
9. Move the new `CACHE.DAT` into that directory.
10. Restart Caché.
11. In the Terminal, switch to the SAMPLES namespace and then enter the following command:

```
do setPath^%bi.SMr.Setup(1)
```

This command updates the path options. (The settings used in the downloaded model database are unlikely to be appropriate for your machine.) For information, see the *DeepSee Site Configuration and Maintenance Guide*.

12. In DeepSee, adjust the **# of Rebuild Processes?** option as appropriate for your machine. See the section “[Configuring DeepSee to Use Multiple CPUs](#),” earlier in this book.

In the downloaded model database, this option is set to 2.

13. Recompile the class `DeepSee.Study.Patient` and then rebuild the DeepSee indices.

This model database is meant for use with the DeepSee.Study sample. You can use this only if your SAMPLES namespace is configured as described in “[Setting Up the Environment.](#)”