DeepSee Developer Tutorial

Version 2017.2
2020-06-25

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

DeepSee Developer Tutorial
Caché Version 2017.2 2020-06-25
Copyright © 2020 InterSystems Corporation
All rights reserved.

InterSystems, InterSystems IRIS, InterSystems Caché, InterSystems Ensemble, and InterSystems HealthShare are registered 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 Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

About This Book

This tutorial is intended 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
- Introduction to Cube Elements
- Creating a Cube
- Expanding the Cube Definition
- Creating Subject Areas
- Creating and Packaging Pivot Tables and Dashboards

For a detailed outline, see the table of contents.

The other developer books for DeepSee are as follows:

- *Getting Started with DeepSee* describes how to get started.
- *DeepSee Implementation Guide* describes how to implement DeepSee, apart from creating the model.
- *Defining DeepSee Models* describes how to define the basic elements used in DeepSee queries: cubes and subject areas. It also describes how to define listing groups.
- *Advanced DeepSee Modeling Guide* describes how to use the more advanced and less common DeepSee modeling features: computed dimensions, unstructured data in cubes, compound cubes, cube relationships, term lists, quality measures, KPIs, plugins, and other special options.
- *Using MDX with DeepSee* introduces MDX and describes how to write MDX queries manually for use with DeepSee cubes.
- *DeepSee MDX Reference* provides reference information on MDX as supported by DeepSee.
- *Tools for Creating DeepSee Web Clients* provides information on the DeepSee JavaScript and REST APIs, which you can use to create web clients for your DeepSee applications.

The following books are for both developers and users:

- *DeepSee End User Guide* describes how to use the DeepSee User Portal and dashboards.
- *Creating DeepSee Dashboards* describes how to create and modify dashboards in DeepSee.
- *Using the DeepSee Analyzer* describes how to create and modify pivot tables, as well as use the Analyzer in general.

Also see the article *Using PMML Models in Caché*.

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

1

Getting Started

The SAMPLES namespace includes two DeepSee samples. One is 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. The DeepSee.Model package includes sample cubes, subject areas, KPIs, pivot tables, and dashboards, for use as reference during this tutorial.

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 test many typical real-life scenarios. This book presents hands-on exercises that use this sample.

Important: DeepSee uses SQL to access data while building the cube, and also when executing detail listings. If your model refers to any class properties that are SQL reserved words, you must enable support for delimited identifiers so that DeepSee can escape the property names. For a list of reserved words, see the "Reserved Words" section in the *Caché SQL Reference*. For information on enabling support for delimited identifiers, see the chapter "Identifiers" in *Using Caché SQL*.

Be sure to consult the online *InterSystems Supported Platforms* document for this release for information on system requirements for DeepSee.

1.1 Getting Started

Most of the tools that you will use are contained in the Management Portal.

To log on:

1. Click the InterSystems Launcher and then click **Management Portal**.

   Depending on your security, you may be prompted to log in with a Caché username and password.

2. Switch to the SAMPLES namespace as follows:

   a. Click **Switch**.

   b. Click SAMPLES.

   c. Click **OK**.

1.2 Regenerating Data

The tutorial uses a larger, slightly more complex set of data than is initially provided in SAMPLES.

To generate data for this tutorial:

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

   ```
   zn "SAMPLES"
   ```

2. Execute the following command:

   ```
   do ##class(DeepSee.Populate).GenerateAll(10000,"ADETR")
   ```

   This class method generates 10000 patients. The "ADETR" string means that the sample will include allergy data (A), diagnosis data (D), encounter data (E), details (T), and the city rainfall data (R).

3. Because we will use SQL queries that run against the tables that we have just populated, it is good practice to run the Tune Table facility on them:

   a. Access the Management Portal and go to the SAMPLES namespace, as described earlier.

   b. Click **System Explorer > SQL**.

   c. Click the **Execute Query** tab.

   d. Click **Actions** and then **Tune All Tables**.

      The system then displays a dialog box where you select a schema and confirm the action.

   e. For Schema, select the DeepSee_Study schema.

   f. Click **Finish**.

   g. Click **Done**.

   The system then runs the Tune Table facility in the background.

2

Introduction to Cube Elements

Before you create your own cube, it is useful to examine a sample cube and see how you can use it. This chapter discusses the following:

- How to access the Patients cube
- Orientation to the Model Contents area
- A typical pivot table
- Measures and levels
- Dimensions and levels
- All members
- Hierarchies
- Properties
- Listings
- Filters and members
- Filters and searchable measures

2.1 Accessing the Patients Cube

1. Access the Management Portal and go to the SAMPLES namespace, as described earlier.
2. Click **Home,DeepSee,Analyzer**.
3. Click the Change Subject Area button  .
4. Click `Patients`.
5. Click **OK**.

The Analyzer page includes three main areas:

- The Model Contents area on the left lists the contents of the cube you selected. You can expand folders and drag and drop items into the Pivot Builder area.

- The Pivot Builder area in the upper right provides options that you use to create pivot tables. This area consists of the **Rows**, **Columns**, **Measures**, and **Filters** boxes.
- The Pivot Preview area in the bottom right displays the pivot table in almost the same way that it will be shown in dashboards.

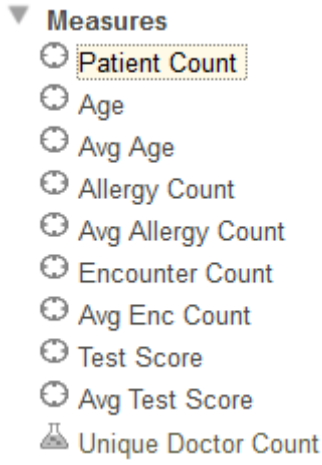
2.2 Orientation to the Model Contents Area

The Model Contents area lists the contents of the cube that you are currently viewing. For this tutorial, select **Dimensions** from the drop-down list; this option displays the measures and dimensions in the given cube.

The top section shows named sets, but this tutorial does not use these. Below that, this area includes the following sections:

2.2.1 Measures

The **Measures** section lists all measures in the cube. For example:



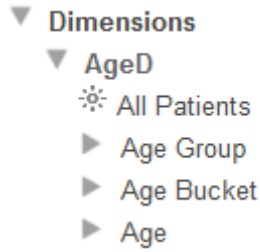
You can have two types of measures, indicated by different icons:

	Standard measures
	Calculated measures, which are defined in terms of other measures

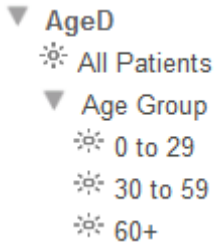
2.2.2 Dimensions

The **Dimensions** section lists the dimensions and the levels, members, and properties that they contain. (It also contains any non-measure calculated members, as well as any sets; this chapter does not discuss these items.)

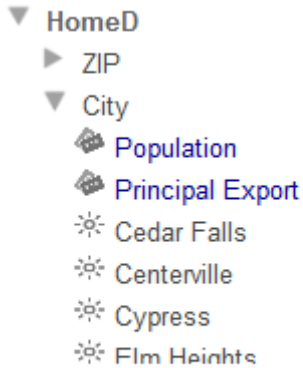
Click the triangle next to any dimension name to expand it. A dimension contains at least one level and may also include a special member known as the *All member*. In the following example, the `AgeD` dimension includes an All member named `All Patients`, as well as the levels `Age Group`, `Age Bucket`, and `Age`.



If you expand a level, the system displays the members of that level. For example:



If a level also includes *properties*, the system shows those properties in blue font, at the start of the list, with a different icon. For example, the `City` level includes the `Population` and `Principal Export` properties:



2.3 Creating a Simple Pivot Table

In this section, you create a simple pivot table that uses levels and measures in a typical way. The goal of this section is to see how levels and measures work and to learn what a member is.

The numbers you see will be different from what is shown here.

1. Expand the `DiagD` dimension in the Model Contents pane.
2. Drag and drop `Diagnoses` to **Rows**.

   Or double-click `Diagnoses`.

   The system displays the following:

Diagnoses	
None	8,425
asthma	703
CHD	323
diabetes	504
osteoporosis	200

3. Drag and drop `Patient Count` to **Measures**.

   Or double-click `Patient Count`.

4. Drag and drop `Avg Age` to **Measures**.

   Or double-click `Avg Age`.

   The system displays the following:

Diagnoses	Patient Count	Avg Age
None	8,425	33.24
asthma	703	34.79
CHD	323	67.49
diabetes	504	57.24
osteoporosis	200	79.46

5. Click **Save**.

   The system displays a dialog box where you specify the pivot table name.

   Save the pivot table and give it a name. When you do so, you are saving the underlying query that retrieves the data, along with the information needed to display it the way you chose. You are not saving the data.

6. For **Folder**, type `Test`
7. For **Pivot Name**, type `Patients by Diagnosis (Patients Cube)`
8. Click **OK**.

It is worthwhile to develop a formal understanding of what we see. Note the following points:

- The base table is Patients, which means that all measures summarize data about patients.
- Apart from the header row, each row of this pivot table displays data for one member of the `Diagnoses` dimension.

  In all cases, a *member* corresponds to a set of records in the fact table. (In most cases, each record in the fact table corresponds to one record in the base table.)

  Therefore, each row in this pivot table displays data for a set of patients with a particular diagnosis.

  Other layouts are possible (as shown later in this book), but in all cases, any data cell in a pivot table is associated with a set of records in the fact table.

- In a typical pivot table, each data cell displays the aggregate value for a measure, aggregated across all records used by that data cell.
- To understand the contents of a given data cell, use the information given by the corresponding labels. For example, consider the cell in the `asthma` row, in the `Patient Count` column. This cell displays the total number of patients who have asthma.

  Similarly, consider the `Avg Age` column for this row. This cell displays the average age of patients who have asthma.

- For different measures, the aggregation can be performed in different ways. For `Patient Count`, DeepSee sums the numbers. For `Avg Age`, DeepSee averages the numbers. Other aggregations are possible.

2.4 Measures and Levels

In this section, we take a closer look at measures and levels.

1. Click **New**.
2. Drag and drop `Count` and `Avg Age`, to the **Measures** area.

   You now see something like this:

	Count	Avg Age
	10,000	35.93

   This simple pivot table shows us the aggregate value for each of these measures, across all the records in the base class. There are 10000 patients and their average age (in this example) is 35.93 years.

3. Compare these values to the values obtained directly from the source table. To do so:
   a. In a separate browser tab or window, access the Management Portal and go to the SAMPLES namespace, as described earlier.
   b. Click **System Explorer > SQL**.
   c. Click the **Execute Query** tab.
   d. Execute the following query:

```
select count(*) as "count",avg(age) as avgage from deepsee_study.patient
```

   You should see the same numbers. For example:

#	count	avgage
1	10000	35.9349
Complete		

   Tip: Leave this browser tab or window open for later use.

4. In the Analyzer, modify the previous pivot table as follows:
   a. Expand `GenD` on the left.
   b. Drag and drop `Gender` to the **Row** area. Now you see something like the following:

Gender	Count	Avg Age
Female	5,161	36.99
Male	4,839	34.81

5. Compare these values to the aggregate values obtained from the source table. To do so:
   a. Access the Management Portal and go to the SAMPLES namespace, as described earlier.
   b. Click **System Explorer > SQL**.

c. Click the **Execute Query** tab.

d. Click **Show History**.

e. Click the query you ran previously.

f. Add the following to the end of the query and then rerun the query:

```
group by gender
```

You should see the same numbers as shown in the pivot table. For example:

#	count	avgage
1	5161	36.99224956403797714
2	4839	34.80719156850588965
Complete		

6. For a final example, make the following change in the Analyzer:

a. Click the X button in the **Rows** pane. This action clears the row definition.

b. Expand `ProfD` and `Profession`.

c. Drag and drop `Electrician` to **Rows**.

The system displays something like this:

Electrician	Count	Avg Age
Electrician	175	39.35

7. Compare these values to the values from the source table. To do so:

a. Access the Management Portal and go to the SAMPLES namespace, as described earlier.

b. Click **System Explorer > SQL**.

c. Click the **Execute Query** tab.

d. Execute the following query:

```
select count(*) as "count",avg(age) as avgage from deepsee_study.patient join
deepsee_study.patientdetails
on deepsee_study.patient.patientid = deepsee_study.patientdetails.patientid
where deepsee_study.patientdetails.profession->profession='Electrician'
```

You should see the same numbers. For example:

#	count	avgage
1	175	39.35428571428571429
Complete		

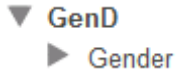
2.5 Dimensions and Levels

In many scenarios, you can use dimensions and levels interchangeably. In this section, we compare them and see the differences.

1. In the Analyzer, click **New**.
2. Drag and drop the GenD definition to the **Rows** area. You should see something like this:

GenD	
Female	5,112
Male	4,888

   The measure shown is Count, which is a count of patients.
3. Click **New**.
4. Expand the GenD dimension. You will see the following in the left area:

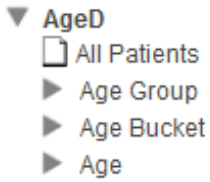


5. Drag and drop the Gender level to the **Rows** area. You should see something like this:

Gender	
Female	5,112
Male	4,888

   In this case, we see the same results except for the caption above the rows.

   In the Patients sample, the names of dimensions are short and end with D, and the name of a level is never identical to the name of the dimension that contains it. This naming convention is not required, and you can use the same name for a level and for the dimension that contains it.
6. Click **New**.
7. Expand the AgeD dimension. You will see the following in the left area:



   This dimension is defined differently from the GenD dimension in two ways:

   - AgeD defines a special member called All Patients, which is an All member. An All member refers to all records of the base class.
   - AgeD defines multiple levels: Age Group, Age Bucket, and Age.
8. Drag and drop the AgeD dimension to the **Rows** area. You should see something like this:

AgeD	
All Patients	10,000
0 to 29	4,250
30 to 59	4,172
60+	1,578

When you drag and drop a dimension for use as rows (or columns), the system displays the All member for that dimension, if any, followed by all the members of the first level defined in that dimension. In this case, the first level is `Age Group`.

2.6 The All Members

An All member refers to all records of the base class. Each dimension can have an All member, but in the Patients cube, only one dimension has an All member.

This part of the tutorial demonstrates how you can use an All member:

1. Click **New**.
2. Expand the `AgeD` dimension.
3. Drag and drop `Age Group` to **Rows**.
4. Drag and drop the measures `Patient Count`, `Avg Age`, and `Avg Test Score` to **Measures**. The system displays something like the following:

Age Group	Patient Count	Avg Age	Avg Test Score
0 to 29	4,250	14.35	74.65
30 to 59	4,172	43.33	74.51
60+	1,578	72.17	75.00

5. Click the Pivot Options button  .
6. In the **Row Options** area, click the **Summary** check box, leave **Sum** selected in the drop-down list, and then click **OK**.

   The system then displays a `Total` line, as follows:

Age Group	Patient Count	Avg Age	Avg Test Score
0 to 29	4,250	14.35	74.65
30 to 59	4,172	43.33	74.51
60+	1,578	72.17	75.00
Total	**10,000**	**130**	**224**

   The `Total` value is appropriate for `Patient Count` but not for the other measures. For `Avg Age` and `Avg Test Score`, it would be more appropriate to display an average value rather than a sum.

7. Click the Pivot Options button  again.
8. In the **Row Options** area, clear the **Summary** check box and then click **OK**.
9. Drag and drop `All Patients` to **Rows**, below `Age Group`. The system then displays the `All Patients` after the members of the `Age Group` level:

Age Group	Patient Count	Avg Age	Avg Test Score
0 to 29	4,250	14.35	74.65
30 to 59	4,172	43.33	74.51
60+	1,578	72.17	75.00
All Patients	10,000	35.56	74.65

The `All Patients` row is a more useful summary line than the `Total` line. It shows the `Patient Count`, `Avg Age`, and `Avg Test Score` measures, each aggregated across all patients.

Note: For `Avg Age` and `Avg Test Score`, in some cases, you might prefer to have an average of the values shown in the pivot table. For example, for `Avg Age`, this summary line adds the ages of all patients and then divides by 10000. You might prefer to add the values of `Avg Age` for the three members shown here and then divide that by three. The All member does not help you do this; instead you would create a calculated member (discussed later in this tutorial).

10. Click the X button in the **Rows** pane. This action clears the row definition.
11. Expand the `DiagD` dimension.
12. Drag and drop `Diagnoses` to the **Rows** pane.
13. Drag and drop `All Patients` to **Rows**, below `Diagnoses`. You then see something like the following:

Diagnoses	Patient Count	Avg Age	Avg Test Score
None	8,440	32.71	74.65
asthma	713	35.17	74.44
CHD	303	69.33	74.92
diabetes	525	58.73	74.45
osteoporosis	198	79.32	75.03
All Patients	10,000	35.56	74.65

As you can see, you can use the generically named `All Patient` member with dimensions other than `Age`, the dimension in which it happens to be defined.

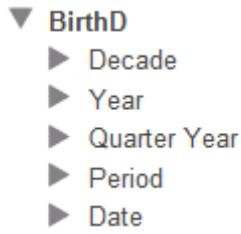
2.7 Hierarchies

A dimension contains one or more hierarchies, each of which can contain multiple levels. The Model Contents area lists the levels in the order specified by the hierarchy, but (to save space) does not display the hierarchy names for this cube.

Users can take advantage of hierarchies to drill to lower levels. This part of the tutorial demonstrates how this works.

1. Click **New**.
2. Expand the `BirthD` dimension in the Model Contents pane.

   The system displays the following:



3. Drag and drop `Decade` to **Rows**.

   Or double-click `Decade`.

   The system displays something like the following:

Decade	
1910s	81
1920s	203
1930s	528
1940s	688
1950s	1,068
1960s	1,447
1970s	1,555
1980s	1,395
1990s	1,478
2000s	1,422
2010s	135

   The measure shown is Count, which is a count of patients.

4. Double-click the `1950s` row (or any other row with a comparatively large number of patients). Click anywhere to the right of the **<<** symbols.

   The system then displays the patients born in that decade, grouped by year (the next lowest level in the hierarchy), as follows:

Decade:1950s	
« 1950	78
« 1951	96
« 1952	98
« 1953	108
« 1954	107
« 1955	100
« 1956	134
« 1957	116
« 1958	107
« 1959	124

   This double-click behavior is available within pivot tables displayed on dashboards (not just within the Analyzer).

5. Double-click a row again. The system displays the patients born in that year, grouped by year and quarter:

Year:1954	
« Q1 1954	28
« Q2 1954	25
« Q3 1954	35
« Q4 1954	19

6. Double-click a row again. The system displays the patients born in that year and quarter, grouped by year and month:

Quarter Year:Q2 1954	
« Apr-1954	8
« May-1954	6
« Jun-1954	11

7. Double-click a row again. The system displays the patients born in that year and month, grouped by actual date:

Period:May-1954	
« May 1 1954	1
« May 15 1954	2
« May 20 1954	1
« May 21 1954	1
« May 22 1954	1

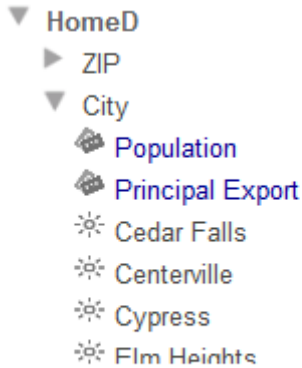
8. Click the **<<** symbols repeatedly to return to the original state of the pivot table.

2.8 Properties

A level can have properties, which you can display in pivot tables.

1. Click **New**.
2. Expand the `HomeD` dimension in the Model Contents pane.
3. Expand the `City` level.

   The system displays the following:



4. Drag and drop `City` to **Rows**.

   The system displays something like the following:

City	
Cedar Falls	1,054
Centerville	1,136
Cypress	1,076
Elm Heights	1,179
Juniper	1,121
Magnolia	1,171
Pine	1,101
Redwood	1,110
Spruce	1,052

   The measure shown is Count, which is a count of patients.

5. Drag and drop `Population` to **Columns**.
6. Drag and drop `Principal Export` to **Columns**.

   The system displays the following:

City	Population	Principal Export
Cedar Falls	90,000	iron
Centerville	49,000	video games
Cypress	3,000	gravel
Elm Heights	33,194	lettuce
Juniper	10,333	wheat
Magnolia	4,503	bundt cake
Pine	15,060	spaghetti
Redwood	29,192	peaches
Spruce	5,900	mud

7. Click the X button in the **Rows** pane.

8. Drag and drop ZIP to **Rows**.

   The system displays something like the following:

ZIP	Population	Principal Export
32006		
32007		
34577		
36711		
38928		

   These properties do not have values for this level.

In pivot tables, properties are different from measures in several ways:

- Properties can have string values.
- Properties have values only for the level in which they are defined.

Depending on how a cube is defined, properties can also affect the sorting and the member names of the level to which they belong. There are examples later in this tutorial.

2.9 Listings

This part of the tutorial demonstrates listings, which display selected records from the lowest-level data for the selected cell or cells. To see how these work, we will first create a pivot table that uses a very small number of records. Then when we display the listing, we will be able to compare it easily to the aggregate value of the cell from which we started.

1. Click **New**.
2. Drag and drop Patient Count and Avg Test Score to **Measures**.
3. Expand the AgeD dimension in the Model Contents pane.
4. Expand the Age level.
5. Drag and drop the member 0 to **Columns**. This member refers to all patients who are less than 1 year old.

   Note that you must click the member name rather than the icon to its left.

   The system displays something like the following:

	0	
	Patient Count	Avg Test Score
	143	54.45

6. Drag and drop the member 1 to **Columns**, below the member 0.

   The system displays something like the following:

	0		1	
	Patient Count	Avg Test Score	Patient Count	Avg Test Score
	143	54.45	147	60.88

7. Expand the `BirthTD` dimension.
8. Drag and drop the `Birth Time` level to **Rows**.

   The system displays something like the following:

	0		1	
Birth Time	Patient Count	Avg Test Score	Patient Count	Avg Test Score
12am	5	68	7	69.86
1am	4	70.50	8	48.13
2am	9	59.44	8	64.63
3am	9	76	9	75.33
4am	9	61.33	7	48.14
5am	5	54.60	5	71
6am	6	65.67	11	56.55

9. Click a cell. For example, click the `Patient Count` cell in the `12am` row, below `0`.
10. Click the Display Listing button  .

    The system considers the selected context, which in this case is patients under 1 year old, who were born between midnight and 1 am. The system then executes an SQL query against the source data. This query includes selected fields for these patients, as follows:

#	PatientID	Age	Gender	Home City	Test Score
1	SUBJ_101358	0	F	Spruce	80
2	SUBJ_102580	0	F	Pine	64
3	SUBJ_107116	0	F	Magnolia	59
4	SUBJ_102201	0	M	Centerville	67
5	SUBJ_102666	0	M	Elm Heights	70

11. Count the number of rows displayed. This equals the `Patient Count` value in the row you started from.
12. Click the Display Table button  to redisplay the pivot table in its original state.

    By default, the Patients cube uses a listing called `Patient details`, which includes the fields `PatientID`, `Age`, `Gender`, and others, as you just saw. You can display other listings as well.

13. Click the Pivot Options button  to display options for this pivot table.

    The system displays a dialog box.

14. For the **Listing** drop-down list, click `Doctor details` and then click **OK**.

    The `Doctor details` listing displays information about the primary care physicians for the selected patients.

15. Click the same cell that you clicked earlier and then click the Display Listing button  .

    Now the system displays something like the following:

#	PatientID	Doctor Last Name	Doctor First Name	Doctor Group
1	SUBJ_101358			
2	SUBJ_102201	Massias	Ralph	I
3	SUBJ_102580	Klein	Gertrude	III
4	SUBJ_102666	Yakulis	Bill	I
5	SUBJ_107116	Geoffrion	Debra	III

2.10 Filters and Members

In a typical pivot table, you use members as rows, as columns, or both, as seen earlier in this chapter. Another common use for members is to enable you to filter the data.

1. In the Analyzer, click **New**.
2. Expand `ColorD` and `Favorite Color`.
3. Drag and drop `Favorite Color` to **Rows**.

   The system displays something like the following:

Favorite Color	
None	2,409
Blue	1,247
Green	1,269
Orange	1,267
Purple	1,259
Red	1,282
Yellow	1,267

   This pivot table displays the members of the `Favorite Color` as rows. The measure shown is Count, which is a count of patients.

4. Drag and drop `Red` to **Filters**.

   The Analyzer now shows only one member of the `Favorite Color` level. You see something like this:

Favorite Color	
Red	1,282

   Make a note of the total number of patients.

5. Click the X button in the **Rows** box.
6. Expand `AgeD`.
7. Drag and drop `Age Group` to **Rows**.

   The Analyzer now displays something like this:

Age Group	
0 to 29	533
30 to 59	535
60+	214

8. Click the Pivot Options button  .
9. In the **Row Options** area, click the **Summary** check box, leave **Sum** selected in the drop-down list, and then click **OK**.

   The Analyzer now displays something like this:

Age Group	
0 to 29	533
30 to 59	535
60+	214
Total	1,282

   The `Total` line displays the sum of the numbers in the column. Notice that the total here is the same as shown earlier.

You can use any member as a filter for any pivot table, no matter what the pivot table uses for rows (or for columns). In all cases, the system retrieves only the records associated with the given member.

You can use multiple members as filters, and you can combine filters. For details, see *Using the DeepSee Analyzer*.

2.11 Filters and Searchable Measures

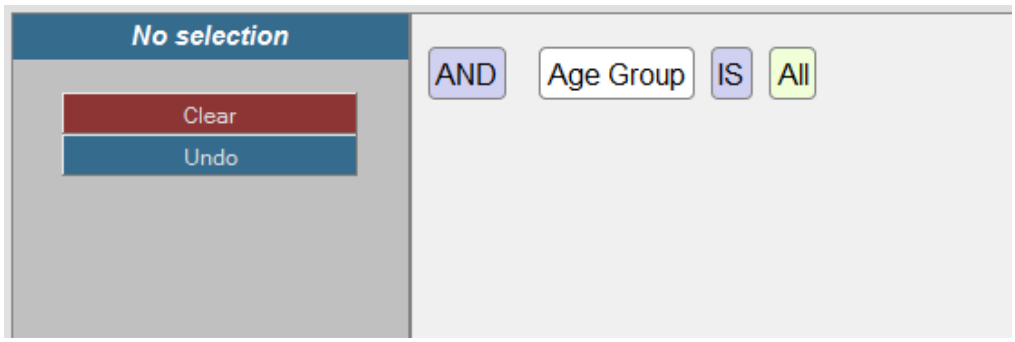
In DeepSee, you can define searchable measures. With such a measure, you can apply a filter that considers the values at the level of the source record itself.

1. Click **New**.

   The system displays the count of all patients:

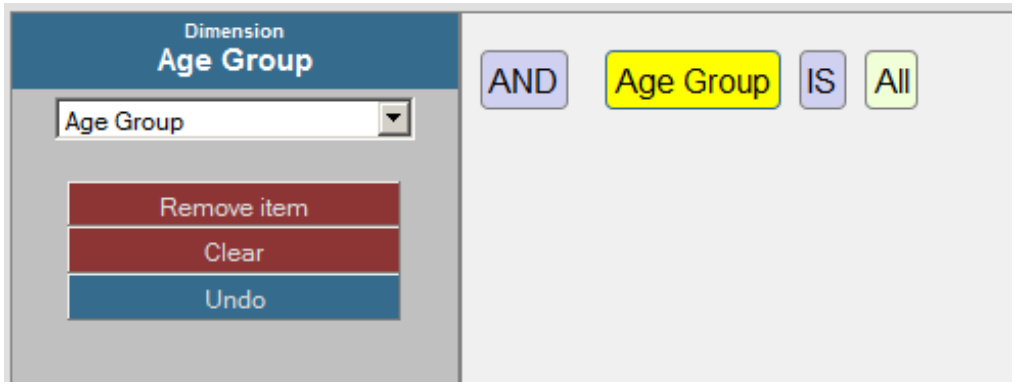
	All
Count	10,000

2. Click the Advanced Options button  in the **Filters** box.
3. Click **Add Condition**. Then you see this:

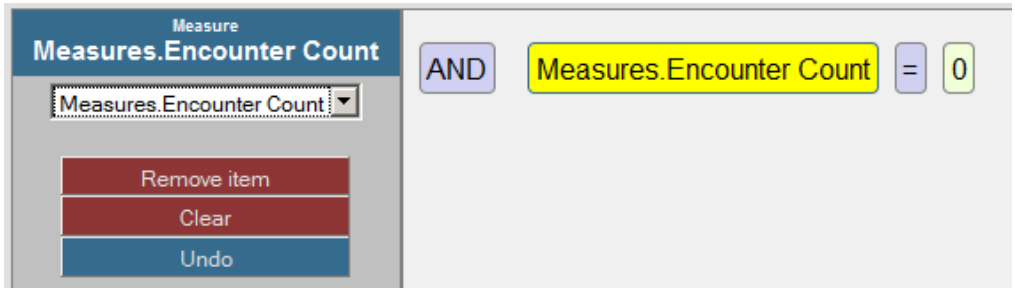


4. Click `Age Group`, which enables you to edit this part of the expression.

   The dialog box now looks something like this:



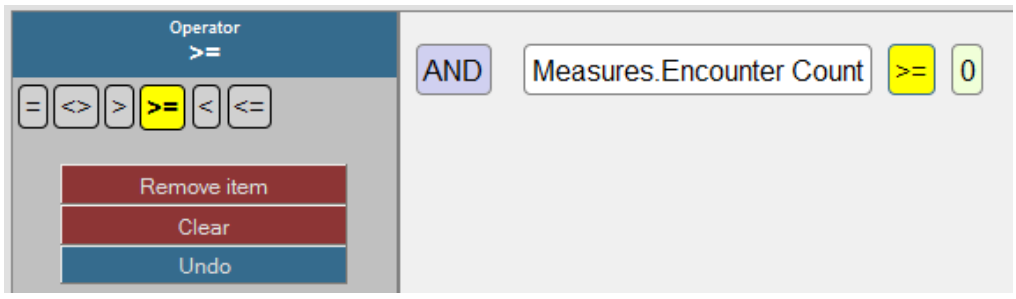
5. Click the drop-down list on the left, scroll down, and click `Measures.Encounter Count`. As soon as you do, the expression is updated. For example:



6. Click the = operator, which enables you to edit this part of the expression.

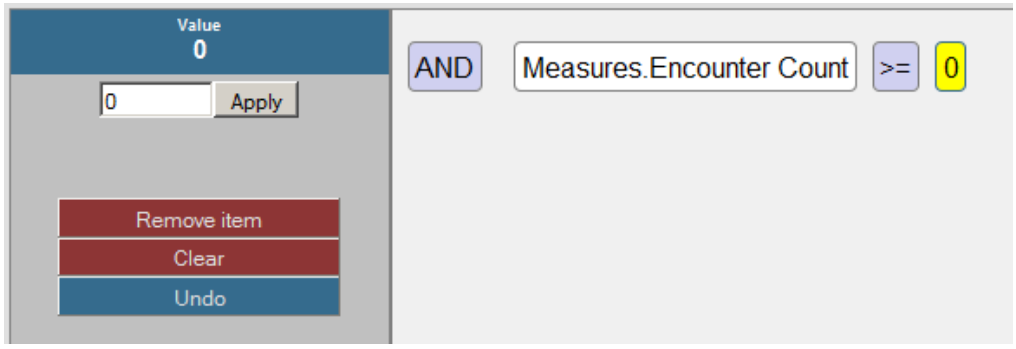
   The dialog box now looks something like this:

7. Click the >= operator. As soon as you do, the expression is updated. For example:



8. Click `0`, which enables you to edit this part of the expression.

The dialog box now looks something like this:



9. Type `10` into the field and click **Apply**.
10. Click **OK**.

    The system then displays the total count of all patients who have at least ten encounters:

	All
Count	7,944

    Now let us see the effect of adding a level to the pivot table.

11. Expand the `AgeD` dimension in the Model Contents pane.
12. Drag and drop `Age Group` to **Rows**.

    The system displays something like the following:

Age Group	
0 to 29	3,217
30 to 59	3,353
60+	1,374

13. Click the Pivot Options button  .
14. In the **Row Options** area, click the **Summary** check box, leave **Sum** selected in the drop-down list, and then click **OK**.
15. Click **OK**.

    The Analyzer now displays something like this:

Age Group	
0 to 29	3,217
30 to 59	3,353
60+	1,374
Total	7,944

    The `Total` line displays the sum of the numbers in the column. Notice that the total here is the same as shown earlier.

3

Creating a Cube

In this chapter, we create a simple cube. This chapter discusses the following topics:

- Creating a basic cube
- Adding levels and measures
- Examining the cube
- Refining the cube
- Adding a listing
- Looking at the fact and level tables

3.1 Creating a Basic Cube

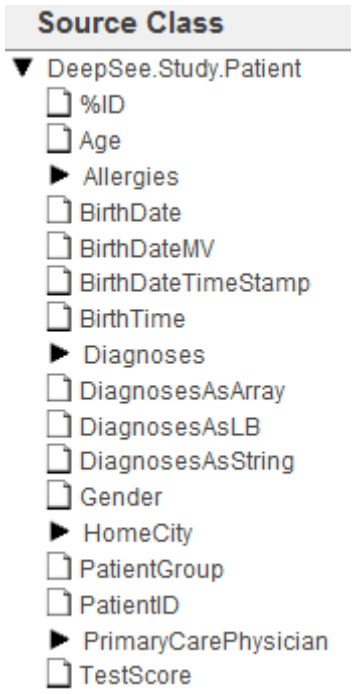
1. Access the Management Portal and go to the SAMPLES namespace, as described earlier.
2. Click **Home,DeepSee,Architect**.
3. Click **New**.

   The system displays a dialog box.
4. In this dialog box, specify the following:
   - **Definition Type: Cube** — Select this.
   - **Cube Name** — `Tutorial`
   - **Class Name for the Cube** — `Tutorial.Cube`
   - **Source Class** — Click the **Browse** button, select `DeepSee.Study.Patient`, and click **OK**.
5. Click **OK**.

   The system creates the cube class, which you can also view and modify in Studio.
6. Click the bold top row in the middle area (labeled `Tutorial`). This selects the cube so that you can edit its details on the right.
7. In the Details Pane, type `None` into **Null replacement string**.
8. Click **Save** and then click **OK**.

The system updates the cube class.

The Class Viewer, which is the left area, now displays this:



Important: The Class Viewer provide a useful view of the class properties (apart from relationship properties) of your base class, 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. These source expressions are evaluated when the cube is built and thus do not affect your runtime performance. This tutorial demonstrates these points later.

3.2 Adding Levels and Measures

In this part of the tutorial, you add some levels and measures to the `Tutorial` cube.

1. Drag and drop the following items from the Class Viewer (the left area) to the **Measures** heading in the Model Viewer (the middle area):
   - `Age`
   - `TestScore`

   This creates measures named `Age` and `TestScore`, based on the class properties with those names.
2. Make the following changes to the `TestScore` measure:
   a. Click the measure name, below the **Measures** heading.
   b. In the Details pane (the right area), change **Name** to `Test Score`
   c. Click **Searchable**.
3. Create the `Avg Age` measure as follows:

a. Drag and drop the Age property again from the Class Viewer to the **Measures** heading in the Model Viewer.

   This step creates a new measure named Age1.

b. Click the measure name in the Model Viewer and then edit the following details in the Details Pane:

   - For **Aggregate**, choose **AVG**.
   - For **Name**, specify Avg Age.
   - For **Format String**, specify #.##

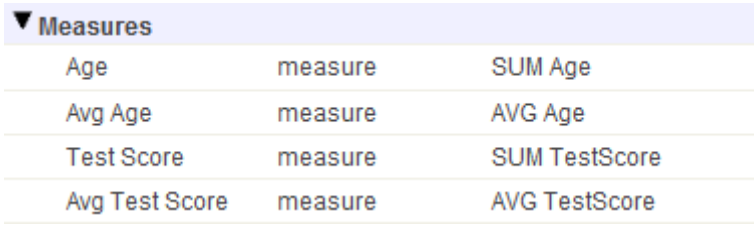
4. Create the Avg Test Score measure as follows:

   a. Drag and drop the TestScore property again from the Class Viewer to the **Measures** heading in the Model Viewer. This step creates a new measure named TestScore1.

   b. Click the measure name in the Model Viewer and then edit the following details in the Details Pane:

      - For **Aggregate**, choose **AVG**.
      - For **Name**, specify Avg Test Score.
      - For **Format String**, specify #.##

   Now you should have four measures:



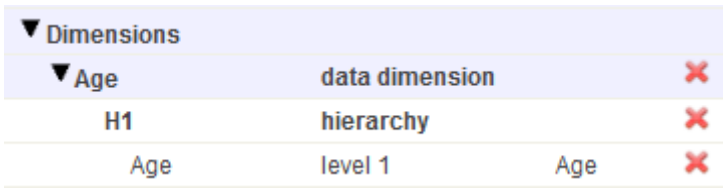
5. Click **Save** and then click **OK**.

   The system updates the cube class.

6. Add a dimension, hierarchy, and level based on the Age property, as follows:

   a. Drag and drop the Age property to the **Dimensions** heading.

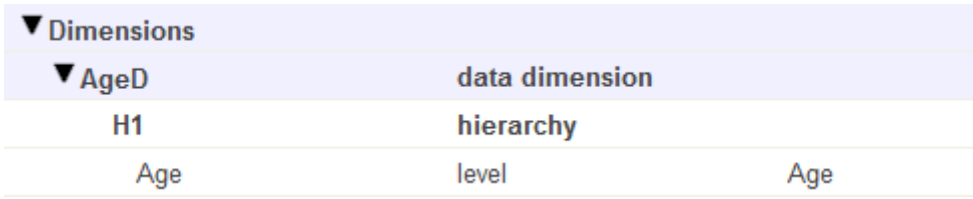
      The Architect immediately creates a dimension, hierarchy, and level, and the Model Viewer now displays the following:



   b. Click the first Age item, which is labeled **data dimension**.

   c. In the right area, edit **Name** to be AgeD.

      The Model Viewer now displays the following:



Depend on how you plan to use DeepSee, users might never see the names of the dimensions. In this tutorial, we follow the convention used in the Patients sample, which assumes that we will not use a dimension as rows or columns in pivot tables (we will instead use levels as rows or columns).

d. Select the option **Enable the All level for this dimension**.

e. Edit **Caption for All member** to be `All Patients`.

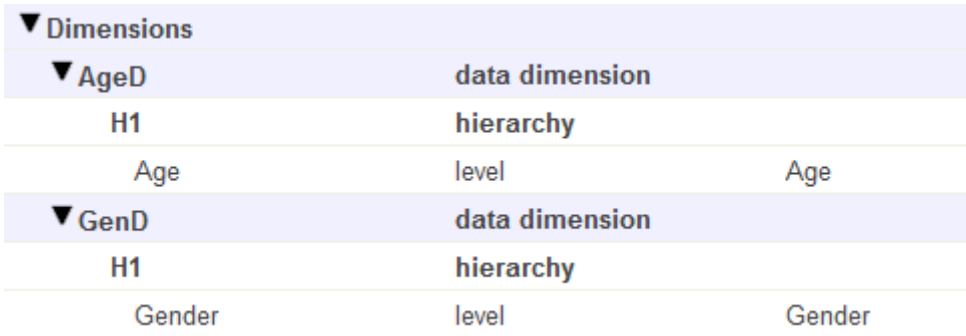
f. Edit **Display name for All member** to be `All Patients`.

7. Save the cube definition in the same way that you did earlier.

8. Add a dimension, hierarchy, and level based on the `Gender` property. Repeat the previous steps with the following differences:

   - Drag and drop the `Gender` property.
   - Rename the dimension to `GenD`.
   - Do not select the option **Enable the All level for this dimension**.

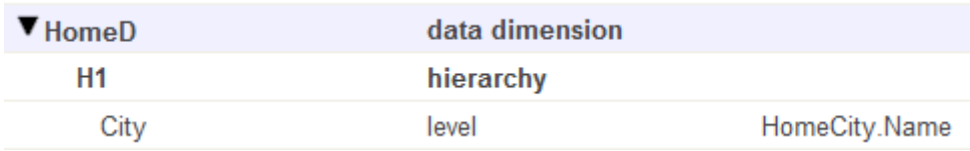
   The Model Viewer now displays the following:



9. Add a dimension, hierarchy, and level based on the `HomeCity` property. Repeat the previous steps with the following differences:

   - Expand the `HomeCity` property and then drag and drop the `Name` property within this folder to **Dimensions**
   - Rename the dimension to `HomeD`
   - Rename the level to `City`
   - Do not select the option **Enable the All level for this dimension**.

   For this new dimension, hierarchy, and level, the Model Viewer now displays the following:



Note that in this case, the **Property** option uses Caché dot syntax.

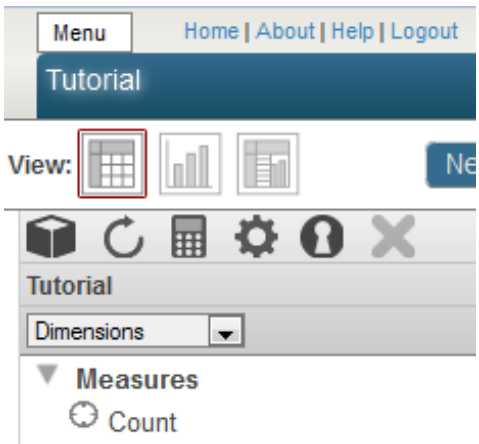
10. Add properties to the `City` level:
    a. Expand `HomeCity` on the left (in the Class Viewer area).
    b. Drag `Population` and drop it onto the `City` level in the middle area.
    c. Drag `PrincipalExport` and drop it onto the `City` level in the middle area.
    d. Select the new `PrincipalExport` property and rename it to `Principal Export`.
11. Add a dimension, hierarchy, and level based on the `PrimaryCarePhysician` property. To do this:
    a. Click **Add Element**.
    b. For **Enter New Element Name**, type `DocD`.
    c. Click **Data Dimension**.
    d. Click **OK**.
    e. Click `New_Level1` in the Model Viewer area.
    f. Change **Name** to `Doctor`.
    g. Type the following ObjectScript expression into **Expression**:

    ```
    %source.PrimaryCarePhysician.LastName_", "_%source.PrimaryCarePhysician.FirstName
    ```

    The variable *%source* refers to the current record. DeepSee evaluates this expression when it builds the cube.

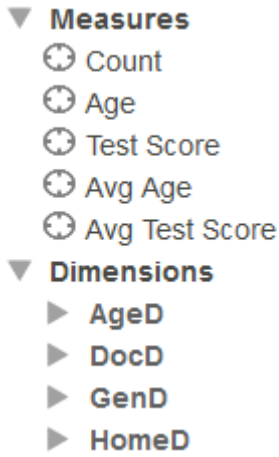
    You could instead use the drag-and-drop procedure as you did earlier, and then edit the definitions.
12. Save the cube definition in the same way that you did earlier.
13. Compile and build the cube. To do so:
    a. Click **Compile**, which starts the compilation and displays progress in a dialog box.
    b. When the system is finished compiling, click **Done**.
    c. Click **Build** and then click **Build**.
    d. When the system is finished building the cube and its indices, click **Done**.
14. In a separate browser tab or window, open the Analyzer.

    Check the upper left area, which displays the title of the currently selected cube or subject area. You should see the following:



If the title is not `Tutorial`, then click the Change button (), click `Tutorial`, and click **OK**.

The left area of the Analyzer displays the current contents of this cube as follows:



If you do not see this, make sure that you have generated data for the sample and that you have compiled and built the cube.

3.3 Examining the Initial Cube

In this section, we examine the cube to see if there is anything we want to change.

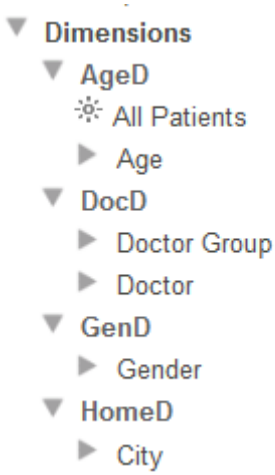
To examine the cube, we will create simple pivot tables by dragging and dropping cube elements from the left area to the Pivot Builder area, which consists of the **Rows** area and the three boxes to its right.

The first thing to notice is that the Analyzer displays a measure (`Count`) that we did not define. This measure is provided automatically, and it counts the records in the base class.

Do the following to get acquainted with the new cube:

1. Click the triangle next to each dimension name in the left area.

   When you are done, you should see the following:



2. Drag and drop the `Age` level to the **Rows** area. You should see something like this:

Age	
0	135
1	149
10	145
11	145
12	154
13	154

Notice that the members of this level as sorted as strings. For this level, it would be better to sort the members numerically, so we will have to make an adjustment to this level.

3. Drag and drop the `Doctor` level to the **Rows** area, placing it directly on `Age`. (In this action, you replace `Age` with `Doctor`.) Now you should see something like this:

Doctor	
,	1,462
Adam, Susan	24
Adams, Elmo	26
Ahmed, Belinda	18
Ahmed, Edward	25

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 to demonstrate a couple of key points.

The doctor name `,` refers to patients who do not have a recorded primary care physician (for these patients, both the last name and first name are null for the `PrimaryCarePhysician` field). We will change this when we redefine this level in the next part of the tutorial.

Whenever you create a level that is based on an identifier, it is important to consider whether that identifier is unique. In many cases (product names, procedure codes, department names, and so on), the identifier is unique. However, it is not safe to assume that names of people are unique. Because we have based this level directly on the doctors' names, DeepSee combines any doctors that have the same name.

For example, some patients could have a doctor named Agnes Djakovic, represented as row 17 in the `Doctor` table, and some other patients could have a doctor with the same name, but who is represented as row 380 in the same table. The `Doctor` level would have a member named `Agnes Djakovic` that combines those patients.

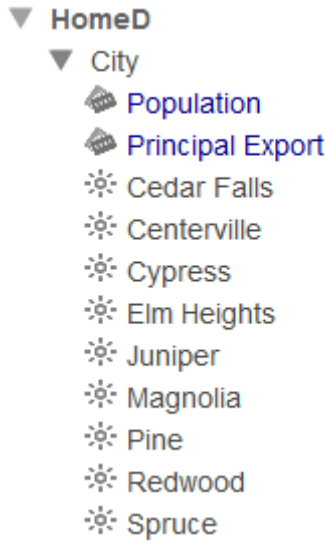
In a later part of the tutorial, we will use a more robust approach.

4. Drag and drop the `Gender` level to the **Rows** area, placing it directly on top of `Doctor`. This action replaces the `Doctor` level with the `Gender` level. Now you should see something like this:

Gender	
Female	5,112
Male	4,888

We will not need to make any changes to this level.

5. Expand the `City` level on the left. You should see this:



6. Drag and drop the `City` level to the **Rows** area, placing it directly on top of `Gender`. Now you should see something like this:

City	
Cedar Falls	1,054
Centerville	1,136
Cypress	1,076
Elm Heights	1,179
Juniper	1,121
Magnolia	1,171
Pine	1,101
Redwood	1,110
Spruce	1,052

7. Drag and drop the `Population` and `Principal Export` properties to **Columns**. You should see this:

City	Population	Principal Export
Cedar Falls	90,000	iron
Centerville	49,000	video games
Cypress	3,000	gravel
Elm Heights	33,194	lettuce
Juniper	10,333	wheat
Magnolia	4,503	bundt cake
Pine	15,060	spaghetti
Redwood	29,192	peaches
Spruce	5,900	mud

We will not need to make any changes to this level.

3.4 Refining the Cube

In this part of the tutorial, we will make the following changes to the cube:

- Change how the members of Age are sorted.
- Ensure that the Doctor level does not combine doctors who have the same name
- Ensure that the Doctor level has a member named None (the cube default replacement string) rather than ,

1. Access the Architect, which shows the cube definition you last looked at.
2. First, redefine the Age level so that its members are sorted numerically. To do so:
   a. Click the Age level.
   b. Click **Add Element**.
   c. For **Enter** , type AgeSort
   d. Click **Property**.
   e. Click **OK**.

      The system adds the property and selects it in the Architect.
   f. In the Details Pane, select **Expression** and enter the following:

      ```
      $CASE($LENGTH(%source.Age),2:%source.Age,:"0"_%source.Age)
      ```

      This expression adds leading zeros to the age, so that string sorting causes the ages to be sorted correctly. The first age is 01, the second is 02, and so on. (The highest age in this sample is 99 years, so no age has more than two characters.)
   g. For **Sort members by property value**, select **asc**.

      This option causes the system to use the values of this property to control how the members are sorted.
   h. Save the cube.

   Note: The Patients sample uses a different approach, and both approaches are valid.
3. Redefine the Doctor level again so that it cannot combine doctors who have the same name. To do so:
   a. Click the Doctor level.
   b. Select the value in the **Expression** field and copy it to Notepad or other temporary location.
   c. Select **Property** and enter PrimaryCarePhysician

      Now the Doctor level is based on the bare PrimaryCarePhysician property, which is an OREF and is unique for each doctor.

      This ensures that the level does not combine different doctors who happen to have the same name.

      This step also ensures that the value is null for patients with no doctor; this means that the cube default null replacement string is used for that member of this level.
   d. While the Doctor level is selected, click **Add Element**.
   e. For **Enter New Element Name**, type Doctor Name
   f. Click **Property**.

g. Click **OK**.

The system adds the property and selects it in the Architect.

h. In the Details Pane, select **Expression** and enter the following:

```
%source.PrimaryCarePhysician.LastName_", "_%source.PrimaryCarePhysician.FirstName
```

i. Select **Use as member names**.

This option causes the system to use the value of this property as the name for each member.

j. For **Sort members by property value**, select **asc**.

This option causes the system to sort the members in ascending order by the value of this property.

4. Compile the cube.

When you do so, the Architect saves the cube.

5. Build the cube.

6. Go to the Analyzer and click the **DeepSee > Analyzer** link to refresh with the most current model.

7. Double-check the changes. You should see the following:

- When you drag and drop `Age` to **Rows**, you see the members sorted in numeric order:

Age	
0	135
1	149
2	144
3	137
4	124
5	147

- When you drag and drop `Doctor` to **Rows**, you see the `None` member:

Doctor	
None	1,505
Adam, Dan	13
Adam, Danielle	21
Adam, Keith	24
Adam, Olga	31
Adam, Peter	23
Adam, Uma	21

Depending on the generated data, you might also see duplicate doctor names. For example:

Lee, Amanda	19
Lee, Bill	16
Lee, Chris	19
Lee, Chris	24

3.5 Adding a Listing to the Cube

A listing enables the users to see selected fields from the lowest-level data, which is useful in many scenarios. This information can help users identify outlier records or any records where follow-up activity might be needed.

1. First, let us examine the available fields in the Patients table.
   a. Access the Management Portal and go to the SAMPLES namespace, as described earlier.

      (If this is open on another browser tab, switch to that tab.)

   b. Click **System Explorer > SQL**.

   c. Click the **Execute Query** tab.

   d. Execute the following query:

      ```
      select * from deepsee_study.patient
      ```

      This displays the first 1000 patients and shows the available fields.

   e. Now try a query like the following:

      ```
      select patientid, age,testscore,homecity->name as "City",
      primarycarephysician->lastname as "Doctor" from DeepSee_Study.Patient
      ```

   f. Copy the query to Notepad or to any other convenient temporary location.

   Leave this browser tab or window open for later use.

2. Add a listing that uses the fields in the query we just ran:
   a. Access the Architect.

      (If this is open on another browser tab, switch to that tab.)

   b. Click **Add Element**.

   c. For **Enter New Element Name**, type `Sample Listing`.

   d. Click **Listing**.

   e. Click **OK**.

      The system adds the listing.

   f. In the Details pane, copy the list of fields from the earlier saved query to the **Field list** area. Specifically, paste this:

      ```
      patientid, age,testscore,homecity->name as "City", primarycarephysician->lastname as "Doctor"
      ```

      The system uses this list of fields and builds the SQL query.

   g. Compile the cube.

      When you do so, the Architect saves the cube.

      You do not need to rebuild the cube.

3. Verify that you can access this listing in the Analyzer. To do so:
   a. Access the Analyzer.

      (If this is open on another browser tab, switch to that tab and click the **DeepSee > Analyzer** link to refresh with the most current model.)

b. Optionally create a simple pivot table.

c. Click a cell in the pivot table preview area.

d. Click the Display Listing button  .

The system displays something like the following:

#	PatientID	Age	TestScore	City	Doctor
1	SUBJ_100315	93	67	Cypress	Zevon
2	SUBJ_100325	36	75	Magnolia	Burroughs
3	SUBJ_100341	9	50	Spruce	Hammel
4	SUBJ_100356	8		Redwood	Novello
5	SUBJ_100385	32	70	Cedar Falls	Townsend
6	SUBJ_100403	11	51	Cedar Falls	Olsen
7	SUBJ_100426	29	72	Magnolia	Quine
8	SUBJ_100467	73		Pine	Quilty

Note: The system displays the first 1000 records by default. You can change this within the Analyzer.

If you instead get a message that listings are not supported, make sure that you saved and recompiled the cube.

4. Modify the listing to sort the records in a different way:

a. Access the Architect again.

b. Click the listing in the Model Contents area.

c. In the Details pane, enter the following into **Order By**:

```
age,homecity->name
```

d. Compile the cube.

When you do so, the Architect saves the cube.

5. Verify that the listing is now sorted by age, and then by city within age.

Display a listing as before. You should see something like this:

#	PatientID	Age	TestScore	City	Doctor
1	SUBJ_107966	0	99	Cedar Falls	Vonnegut
2	SUBJ_110142	0	88	Cedar Falls	Alton
3	SUBJ_109388	0	88	Cypress	
4	SUBJ_106292	0	56	Juniper	Young
5	SUBJ_104172	1	61	Centerville	Frith
6	SUBJ_102058	1		Cypress	Vivaldi
7	SUBJ_103519	1	79	Elm Heights	Koenig
8	SUBJ_104709	1	66	Elm Heights	Ingersol

Scroll down to verify that patients are sorted by city within age.

3.6 Looking at the Fact and Level Tables

If you are responsible for creating cube definitions, it is useful to understand how DeepSee uses the cube definition to build the tables that DeepSee uses directly: the *fact table* and *level tables*. In this section we examine these tables.

1. Access the Management Portal and go to the SAMPLES namespace, as described earlier.
2. Click **System Explorer > SQL**.
3. Click the **Execute Query** tab.
4. Execute the following SQL query, which runs against the base table used by your cube:

   ```
   select top 1 age,gender,homecity->name,primarycarephysician->lastname,
   primarycarephysician->firstname, testscore from DeepSee_Study.patient
   ```

   Make a note of the details:

#	Age	Gender	Name	LastName	FirstName	TestScore
1	13	F	Magnolia	Quince	Marvin	88
Complete						

5. In the left area, navigate to the table `Tutorial_Cube.Fact`.
6. Click Open Table.

   The system displays something like the following:

#	ID	%partition	%sourceId	DxAge	DxGender	DxNameViaHomeCity	DxPrimaryCarePhysician	MxAgeN	MxTestScoreN
1	1	1	1	1	1	1	1	13.00	88.00
2	2	1	2	2	1	2	2	27.00	
3	3	1	3	3	2	3	3	22.00	71.00
4	4	1	4	4	2	4	4	10.00	69.00
5	5	1	5	5	1	4	5	77.00	69.00
6	6	1	6	6	2	5	6	59.00	87.00
7	7	1	7	7	1	1	4	17.00	
8	8	1	8	8	1	6	7	39.00	
9	9	1	9	9	1	6	8	84.00	87.00
10	10	1	10	10	1	2	9	11.00	
11	11	1	11	11	1	7	10	25.00	
12	12	1	12	12	1	3	11	2.00	52.00
13	13	1	13	13	2	1	12	8.00	
14	14	1	14	14	1	3	13	19.00	

   This table is generated when you compile a cube and is populated when you build the cube. The fact table has one row (called a *fact*) for each record that it used from the source table. In this case, each fact corresponds to one patient.

   The first row in this table corresponds to the first row in the base table (who is 13 years old and who has a test score of 88).

7. Note the following points:

   - The `%sourceId` field indicates the ID of source record on which a fact was based.
   - Each field with a name that starts `Dx` corresponds to a level that you defined. The fact table stores integers in these fields, which refer to records in the level tables.
   - Each field with a name that starts `Mx` corresponds to a measure that you defined. The fact table stores numbers (rather than integers) in these fields, because that is the default type for measures.
   - For some facts, the value of the `MxTestScore` field is null.

8. Click **Close window**.
9. Navigate to the table `Tutorial_Cube.StarGender`.
10. Click Open Table. The system displays something like this:

#	ID	DxGender
1	1	Female
2	2	Male
Complete		

This table contains the names of the members of the `Gender` level. The `DxGender` field of the fact table refers to the rows in this table.

In your case, you might see `Male` before `Female`.

In this case, the `Female` member is first, because the first patient processed by the system is female.

When the system populates these tables, it iterates through the records in the base table. For each record, the system looks at the definition of each level, determines a value, adds that value (if needed) to the corresponding level table, and writes a lookup value into the level field of the fact table.

11. Click **Close window**.
12. Navigate to the table `Tutorial_Cube.StarAge`. The system displays something like the following:

#	ID	Dx781900468	DxAge
1	1	13	13
2	2	27	27
3	3	22	22
4	4	10	10
5	5	77	77
6	6	59	59
7	7	17	17
8	8	39	39
9	9	84	84
10	10	11	11
11	11	25	25
12	12	02	2
13	13	08	8
14	14	19	19

The `Age` level is defined by the `Age` field of the base class; that value is shown in the `DxAge` column. This level has a level property that is used to define the sort order for the level members; that value is shown in the `Dx781900468` column.

The first record in this level table corresponds to the age of 13 years, the first patient processed by the system in this example.

13. Click **Close window**.
14. Navigate to the table `Tutorial_Cube.StarNameViaHomeCity`. The system displays something like the following:

#	ID	Dx1438697606	DxNameViaHomeCity	DxPopulationViaHomeCity
1	1	bundt cake	Magnolia	4503
2	2	lettuce	Elm Heights	33194
3	3	spaghetti	Pine	15060
4	4	wheat	Juniper	10333
5	5	video games	Centerville	49000
6	6	mud	Spruce	5900
7	7	gravel	Cypress	3000
8	8	peaches	Redwood	29192
9	9	iron	Cedar Falls	90000
Complete				

The `City` level is defined by the `HomeCity->Name` field in the base class; that value is shown in the `DxNameViaHomeCity` column. This level has two level properties that are shown in the other columns.

The first record in this table is `Magnolia`, the home city of the first patient in the base table.

15. Click **Close window**.
16. Navigate to the table `Tutorial_Cube.StarPrimaryCarePhysician`. The system displays something like the following:

#	ID	Dx582175229	DxPrimaryCarePhysician
1	1	Quince, Marvin	232
2	2	Lopez, Kim	41
3	3	Drabek, Wolfgang	13
4	4	,	<null>
5	5	Cerri, Elvira	159
6	6	Sorenson, Hannah	12
7	7	Houseman, Janice	283

The `Doctor` level is defined by the `PrimaryCarePhysician` field in the base class, which is a reference (OREF) to an instance of the `DeepSee.Study.Doctor` class. The system converts the OREF to an integer and writes it into the `DxPrimaryCarePhysician` column.

For this level, the member names are defined by a level property that concatenates the last name and first name, with a comma between them. The value of this level property is stored in the `Dx582175229` column.

The first doctor in this table is `Quince, Marvin`, the primary care physician of the first patient in the base table.

The name of the null doctor is a comma, but this name is never shown; instead, for this member, the system uses the null replacement string that you specified.

Tip: To make the field names in these tables more useful, you can specify the option **Field name in fact table** for the levels and measures that you define. Note that this option does not apply to time levels (discussed in the next chapter), which have special internal handling.

4

Expanding the Cube Definition

In the previous chapter, we created and tested a simple cube. In this chapter, we expand that cube to use more parts of the Patient data and try more DeepSee features. This chapter discusses the following topics:

- Adding a level to a hierarchy
- Adding time levels
- Using collection properties
- Using replacements
- Accessing data in other tables

4.1 Adding a Level to a Hierarchy

So far, each dimension we have created has contained one hierarchy with one level. In this section, we add a level to the hierarchy in the `HomeD` dimension.

1. In the Architect, add a level to the `HomeD` dimension as follows:
   a. In the Class Viewer, expand `HomeCity`.
   b. Drag `PostalCode` and drop it onto the `H1` hierarchy within the `HomeD` dimension.

      This step adds the new level `PostalCode` after the `City` level.
   c. Click `PostalCode`.
   d. In the Details pane, change **Name** to `ZIP Code`.
2. Compile the cube.

   When you do so, the Architect saves the cube.
3. Build the cube.
4. Access the Analyzer.

   (If this is open on another browser tab, switch to that tab and click the **DeepSee > Analyzer** link to refresh with the most current model.)
5. Expand the `HomeD` dimension in the left. You should see the following:

6. Display the `ZIP Code` levels as rows. You should see something like this:

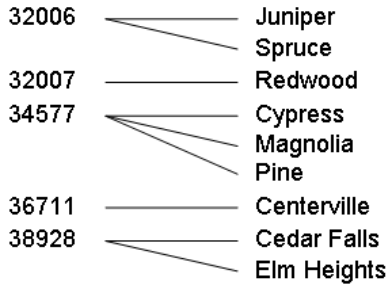
ZIP Code	
32006	1,051
32006	1,122
32007	1,159
34577	1,075
34577	1,138
34577	1,183
36711	1,106
38928	1,090
38928	1,076

Notice that some members have the same name. It is sometimes correct to have multiple members with the same name. In this case, however, it is an error, because ZIP codes are unique.

There are only two ways in which a level can have multiple members with the same name:

- The level name is based on a level property, which is not unique. (For an example, see the `Doctor` level that we defined in the previous chapter.)
- The level has a parent level. When DeepSee creates members of a level, it considers not only the source property or expression; it also considers the parent member.

In reality, there is a many-to-many relationship between ZIP codes and cities, so that neither is the parent of the other. In the Patients sample, ZIP codes contain small cities as follows:



When we added the `ZIP Code` level, we placed it after the `City` level, which means that `City` is the parent of `ZIP Code`. This affected how the system generated members for `ZIP Code`. For example, the system assumed that the ZIP code `32006` of the city `Juniper` was not the same as the ZIP code `32006` of the city `Spruce`.

7. Go back to the Architect and correct the `HomeD` dimension.

   a. Click the `ZIP Code` level.

   b. Click the up arrow button.

   c. Compile the cube.

      When you do so, the Architect saves the cube.

d. Build the cube.

8. Access the Analyzer.

   (If this is open on another browser tab, switch to that tab and click the **DeepSee > Analyzer** link to refresh with the most current model.)

9. Expand the `HomeD` dimension in the left. You should see the following, which is now correct:

10. Display the `ZIP Code` levels as rows. Now you should see something like this, which is correct:

ZIP Code	
32006	2,173
32007	1,159
34577	3,396
36711	1,106
38928	2,166

11. Double-click the row `34577`. The system now displays the cities within this ZIP code.

ZIP Code:34577	
« Cypress	1,183
« Magnolia	1,075
« Pine	1,138

12. Optionally do the following to see how this change has affected the fact and level tables.

    a. Access the Management Portal and go to the SAMPLES namespace, as described earlier.

    b. Click **System Explorer > SQL**.

    c. In the left area, navigate to the table `Tutorial_Cube.Fact`.

       Notice that this table now has the field `DxPostalCodeViaHomeCity` in addition to `DxNameViaHomeCity`. That is, the fact table stores a value for each level, even the levels are related.

    d. In the left area, navigate to and open the table StarNameViaHomeCity.

       The system displays something like the following:

#	ID	Dx1438697606	DxNameViaHomeCity	DxPopulationViaHomeCity	DxPostalCodeViaHomeCity
1	1	bundt cake	Magnolia	4503	1
2	2	lettuce	Elm Heights	33194	2
3	3	spaghetti	Pine	15060	1
4	4	wheat	Juniper	10333	3
5	5	video games	Centerville	49000	4
6	6	mud	Spruce	5900	3
7	7	gravel	Cypress	3000	1
8	8	peaches	Redwood	29192	5
9	9	iron	Cedar Falls	90000	2
Complete					

Notice that now the table stores, for each city, the ZIP code to which that city belongs.

e. Close this table and navigate to the table `Tutorial_Cube.StarPostalCodeViaHomeCity`.

The system displays something like the following:

#	ID	DxPostalCodeViaHomeCity
1	1	34577
2	2	38928
3	3	32006
4	4	36711
5	5	32007
Complete		

This level table is like the other level tables: one row for each level member.

4.2 Adding Time Levels

In this part of the tutorial, we add time levels to the cube.

The Patients class includes the patient's birth date in several forms (so that you can try different formats with DeepSee):

```
Property BirthDate As %Date;
Property BirthDateTimeStamp As %TimeStamp;
Property BirthDateMV As %MV.Date;
```

DeepSee has built-in support for all three of these formats, as well as for **$HOROLOG** format and others (for details, see *Defining DeepSee Models*).

The class also includes the patient's birth time, as part of the `BirthDateTimeStamp` property or as the following property:

```
Property BirthTime As %Time;
```

The most flexible property is `BirthDateTimeStamp`, because it contains both the birth date and the birth time, so we will use that as the basis for the time levels.

1. Access the Architect and display the `Tutorial` cube.
2. Click **Add Element**.
3. For **Enter New Element Name**, type `BirthD`.
4. Click **Time Dimension**.
5. Click **OK**.

   The system creates a dimension, hierarchy, and level.
6. Make the following change to the dimension:
   - Click the search button next to **Property**, click `BirthDateTimeStamp`, and click **OK**.
7. Make the following changes to the level:
   - Rename the level to `Year`.
   - For **Extract value with function**, select **Year**.

This option means that this level is based only the patients' birth years.

8. Add another level as follows:
   a. Click the hierarchy `H1` in this dimension.
   b. Click **Add Element**.
   c. For **Enter New Element Name**, type `Month Year`.
   d. Click **Level**.
   e. Click **OK**.

      The system creates a new level in the hierarchy `H1`, after the existing `Year` level.

9. For the `Month Year` level, make the following change:
   - For **Extract value with function**, select **MonthYear**.

     This option means that this level is based on the combined birth year and month.

10. Add another hierarchy and level to the `BirthD` dimension, as follows:
    a. Click the dimension name.
    b. Click **Add Element**.
    c. For **Enter New Element Name**, type `H2`.
    d. Click **Hierarchy**.
    e. Click **OK**.

       The system creates a new hierarchy and level.

    f. For the new level, make the following changes:
       - Rename the level to `Time`.
       - For **Extract value with function**, select **HourNumber**.

         This option means that this level is based on the time of day the patient was born.

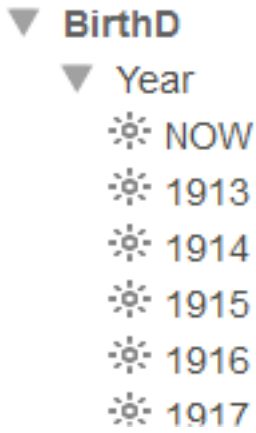
11. Compile the cube.

    When you do so, the Architect saves the cube.

12. Build the cube.
13. Access the Analyzer.

    (If this is open on another browser tab, switch to that tab and click the **DeepSee > Analyzer** link to refresh with the most current model.)

14. Try the new levels. You should see the following:
    - When you expand `Year` in the left area, you see this:



NOW is a special member that refers to the current year (in this context).

- The `Month Year` level also has a NOW member, which refers to the current year and month.
- When you use `Year` as rows, you should see something like this:

Year	
1911	8
1912	4
1913	6
1914	14
1915	4
1916	10
1917	8

- When you use `Month Year` as rows, you should see something like this:

Month Year	
Feb-1911	2
Jul-1911	3
Oct-1911	1
Nov-1911	1
Dec-1911	1
Mar-1912	1
Jun-1912	1

- When you use `Time` as rows, you should see something like this:

Time	
12am	438
1am	421
2am	425
3am	419
4am	414
5am	410
6am	411

The system does not generate tables for time levels, which have special internal handling.

4.3 Using a Collection Property

You can create levels based on collection properties. Specifically, the system can directly use either a list of the type returned by **$LIST**, %List, or a character-delimited list. If a collection property stores data in some other way, it is necessary to extract the necessary data and create one of the supported types of lists.

The `DeepSee.Study.Patient` class has several collection properties, including `Allergies` and `DiagnosesAsLB`. The `DiagnosesAsLB` property is defined as follows:

```
Property DiagnosesAsLB As %List;
```

The `Allergies` property is defined as follows:

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

This part of the tutorial shows you how to create levels and measures that use these properties:

1. Access the Architect and display the `Tutorial` cube.
2. Add a dimension, hierarchy, and level that uses the `DiagnosesAsLB` property, as follows:
   a. Click **Add Element**.
   b. For **Enter New Element Name**, type `DiagD`.
   c. Click **Data Dimension**.
   d. Click **OK**.

      The system creates a dimension, hierarchy, and level.
   e. Rename the level to `Diagnoses`.
   f. While the level is selected, click the search button for **Property**, select the `DiagnosesAsLB` property, and click **OK**.
   g. For **Source value is a list of type**, click **$LIST**. This type refers to data that has the format returned by the **$LIST** function or that has the type %List.
   h. Save the cube class.
3. In the Architect, add a dimension, hierarchy, and level as before, with the following changes:
   - The dimension name should be `AllerD`.
   - The level name should be `Allergies`.
   - Do not specify a value for **Property**.

     There is no property that we can use directly. It will be necessary to extract the list of allergies via an expression.
   - Specify the following value for **Expression**:

     ```
     ##class(Tutorial.Cube).GetAllergies(%source.%ID)
     ```

     The system evaluates this expression once for each row in the fact table, when it builds the cube.

     The variable *%source* refers to the current record. This expression gets the ID of the patient, invokes the utility method (which we have not yet written), and returns a list of allergies for the patient.

- Remember to select **$LIST** for **Source value is a list of type**.

Then save your cube class.

The next step will be to write this utility method.

4. Open Studio and access the SAMPLES namespace.
5. Open your cube class, `Tutorial.Cube`.
6. Add a method named `GetAllergies()`, as follows:

```
ClassMethod GetAllergies(ID As %Numeric) As %List
{
    Set allergies=##class(DeepSee.Study.Patient).%OpenId(ID,0).Allergies
    If (allergies.Count()=0) {Quit $LISTFROMSTRING("")}
    Set list=""
    For i=1:1:allergies.Count() {
        Set $LI(list,i)=allergies.GetAt(i).Allergen.Description
        }
    Quit list
}
```

Given the ID of a patient, this method returns a list of allergies of that patient, in the format expected by the level we created.

The second argument of **%OpenId()** specifies the level of concurrency locking to use. Because we only need to read data from the object, we specify this value as 0, which establishes no concurrency locking and thus runs more quickly.

7. Save and compile your cube class in Studio.
8. Add a measure that contains the number of allergies that a patient has. To do so, we use the `Allergies` property, as follows:
   a. Return to the Architect.
   b. Click **Add Element**.
   c. For **Enter New Element Name**, type `Avg Allergy Count`.
   d. Click **Measure**.
   e. Click **OK**.

      The new measure is added to the table.
   f. Click the measure in the Model Contents area.
   g. For **Aggregate**, click **AVG**.
   h. For **Expression**, enter the following:

      ```
      ##class(Tutorial.Cube).GetAllergyCount(%source.%ID)
      ```

      We will have to write this method later.
   i. Save the cube class in the Architect.
   j. Because you have edited the class in Studio, the Architect displays a dialog box that asks whether you want to override the stored definition. Click **OK**. The Architect overrides only the parts of the class definition that you can edit in the Architect; that is, it does not override any methods you have added to the class.
   k. In Studio, add the following method to your cube class:

      ```
      ClassMethod GetAllergyCount(ID As %Numeric) As %Numeric
      {
           Set allergies=##class(DeepSee.Study.Patient).%OpenId(ID,0).Allergies
           Quit allergies.Count()
      }
      ```

l. Save and compile the cube class in Studio.

9. Rebuild the DeepSee cube.

   To do this, you can return to the Architect and rebuild the same way that you did before.

   Or you can open a Terminal window and enter the following command in the SAMPLES namespace:

   ```
   do ##class(%DeepSee.Utils).%BuildCube("tutorial")
   ```

   Notice that the method uses the logical name of the cube (rather than the class name). Also notice that the cube name is not case-sensitive.

10. Access the Analyzer.

    (If this is open on another browser tab, switch to that tab and click the **DeepSee > Analyzer** link to refresh with the most current model.)

11. Display the `Diagnoses` level as rows. You should see the following:

Diagnoses	
None	8,499
asthma	647
CHD	318
diabetes	506
osteoporosis	176

    In your data, you might also see the epilepsy diagnosis, which is more rare.

    You might instead see something like the following:

Diagnoses	
CHD	247
CHD osteoporosis	25
asthma	618
asthma CHD	16
asthma CHD osteoporosis	2

    This occurs if you do not specify the appropriate type for **Source value is a list of type**.

12. Click **New**.

13. Display the new `Allergies` level as rows, and display the `Count` and `Avg Allergy Count` measures. You should see something like the following:

Allergies	Count	Avg Allergy
None	3,879	0
additive/coloring agent	431	1.78
animal dander	454	1.73
ant bites	433	1.77
bee stings	453	1.75
dairy products	402	1.70
dust mites	447	1.66
eggs	393	1.75
fish	459	1.75
mold	421	1.79
nil known allergies	1,485	1
peanuts	431	1.72
pollen	468	1.73
shellfish	430	1.75
soy	450	1.71
tree nuts	442	1.65

The `nil known allergies` member represents the patients who have no known allergies. Some medical information systems use the following technique to record the fact that a patient has no known allergies:

- The system includes a special "allergen" called `nil known allergies`.
- A user of the system 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.

The null member (called `None`) represents the patients whose `Allergies` property is null. Because it is incorrect to assume that these patients have no allergies, the name of this member is misleading. A better name would be `No Data Available`.

Notice that the `Avg Allergy Count` measure is 0 for patients who belong to the null member. The `Avg Allergy Count` measure should be null for these patients.

Also notice that the `Avg Allergy Count` measure is 1 for patients with no known allergies. This is because the `Allergies` property does include the special `nil known allergies` allergen. The `Avg Allergy Count` measure should be 0 for these patients.

Later in this section, we will correct the name of the null member and adjust our logic for the `Avg Allergy Count` measure.

14. Return to the Architect.
15. Click the `Allergies` level.
16. For **Null replacement string**, specify `No Data Available`.
17. Save the cube class.
18. In Studio, edit the method `GetAllergyCount()` as follows:

```
ClassMethod GetAllergyCount(ID As %Numeric)
{
    Set allergies=##class(DeepSee.Study.Patient).%OpenId(ID,0).Allergies
    //check to see if patient has any recorded allergy data
    //if not, count is null

    If allergies.Count()=0 {
        Set allcount=""
                }
                 //check to see if patient has "Nil known allergies"
                 //in this case, the patient has one "allergen" whose code is 000
                Elseif ((allergies.Count()=1) && (allergies.GetAt(1).Allergen.Code="000")) {
                        Set allcount=0
                        }
                Else {
         Set allcount=allergies.Count()
         }

     Quit allcount
}
```

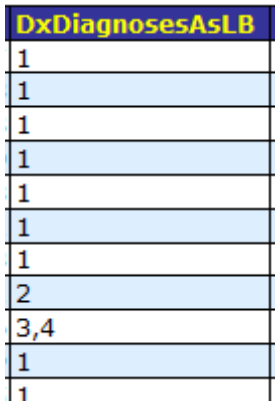
19. Save the cube class.
20. Compile the cube class in Studio or in the Architect.
21. Build the cube in the Architect.
22. Access the Analyzer.

    (If this is open on another browser tab, switch to that tab and click the **DeepSee > Analyzer** link to refresh with the most current model.)
23. Display the `Allergies` as rows, and display the `Count` and `Avg Allergy Count` measures. Now you should see something like the following:

Allergies	Count	Avg Allergy
No Data Available	3,879	
additive/coloring agent	431	1.78
animal dander	454	1.73
ant bites	433	1.77
bee stings	453	1.75
dairy products	402	1.70
dust mites	447	1.66
eggs	393	1.75
fish	459	1.75
mold	421	1.79
nil known allergies	1,485	0
peanuts	431	1.72
pollen	468	1.73
shellfish	430	1.75
soy	450	1.71
tree nuts	442	1.66

24. Optionally do the following to see how list-based levels are represented in the fact and level tables.
    a. Access the Management Portal and go to the SAMPLES namespace, as described earlier.
    b. Click **System Explorer > SQL**.
    c. In the left area, navigate to and open the table `Tutorial_Cube.Fact` and scroll to the field `DxDiagnosesAsLB`.

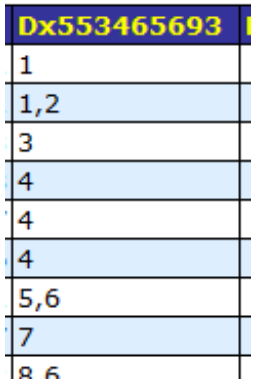
The system displays something like the following:



DxDiagnosesAsLB
1
1
1
1
1
1
1
2
3,4
1
1

This field contains the diagnoses for the patients. Notice that it contains multiple values in some cases.

The table also displays the allergies level, perhaps like this:

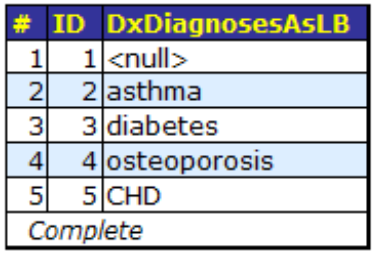


Dx553465693
1
1,2
3
4
4
4
5,6
7
8,6

The name of this field is less obvious, because it is generated, because the level itself is based on an expression.

Because this is another list-based level, it contains multiple values in some cases.

d. Now navigate to and open the table `Tutorial_Cube.StarDiagnosesAsLB`.



#	ID	DxDiagnosesAsLB
1	1	<null>
2	2	asthma
3	3	diabetes
4	4	osteoporosis
5	5	CHD
Complete		

This level table is like the other level tables: one row for each level member.

The level table for allergies is similar: one row for each level member.

The method we used for `Avg Allergy Count` was fairly simple. Consider the following method:

```
ClassMethod GetScore(ID As %Numeric) As %String
{
    //get customer rating data & call duration from source record
    set call=##class(MyPackage.MyClass).%OpenId(ID,0)
    set professionalism=call.Professionalism
    set knowledge=call.Knowledge
    set speed=call.OpenDuration

    If ...
        //logic to check for nulls and combine these values into weighted overall score
    Quit score
}
```

You could use a method like this to define a measure that indicates an overall score.

4.4 Defining Replacements

In this part of the tutorial, we use options that transform the original values for levels into other values. Here we will use the `Age` property of the patient. We will define levels that place patients into buckets that are larger than one year.

The `Age Group` level will have the following members:

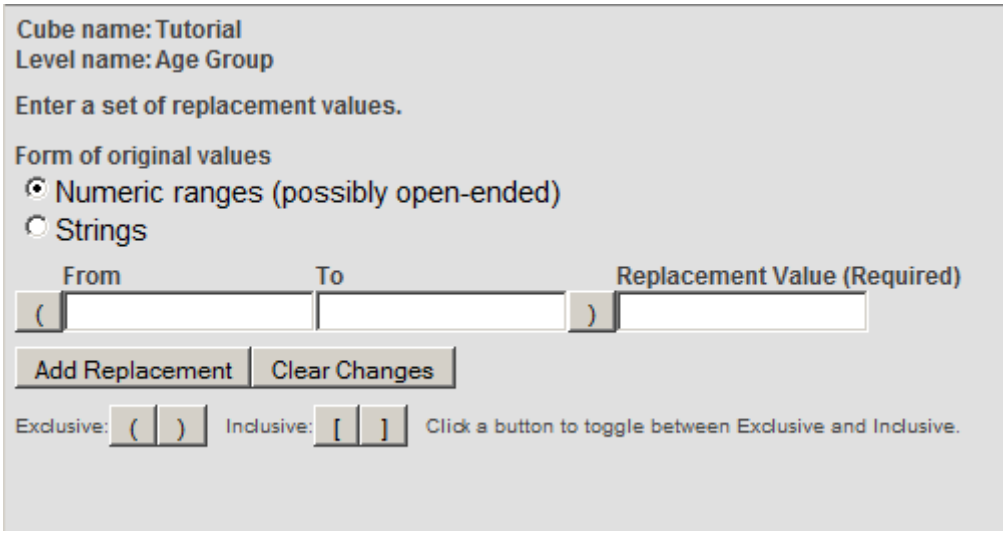
- The `0 to 29` member consists of the patients who are less than 30 years old.
- The `30 to 59` member consists of the patients who are between 30 and 59, inclusive.
- The `60+` member consists of the patients who are older than 60 years.

Similarly, the `Age Bucket` level will have the members `0 to 9`, `10 to 19`, and so on.

1. Access the Architect.
2. Add another level to the `AgeD` dimension as follows. To do so:
   a. Click the `Age` level. This ensures that the new level, which is less granular, will be added before the `Age` level.
   b. Click **Add Element**.
   c. For **Enter New Element Name**, type `Age Group`.
   d. Click **Level**.
   e. Click **OK**.
3. Redefine the new `Age Group` level to have a range expression, as follows:
   a. Click the new `Age Group` level.
   b. For **Field name in fact table**, specify `DxAgeGroup`

      This will make it easier for us to see how the level definition affects the generated tables.
   c. For **Property**, type `Age`.
   d. Click the search button next to **Range Expression**.

      The system displays a dialog box where you specify a set of replacements. Originally, this dialog box looks like this:



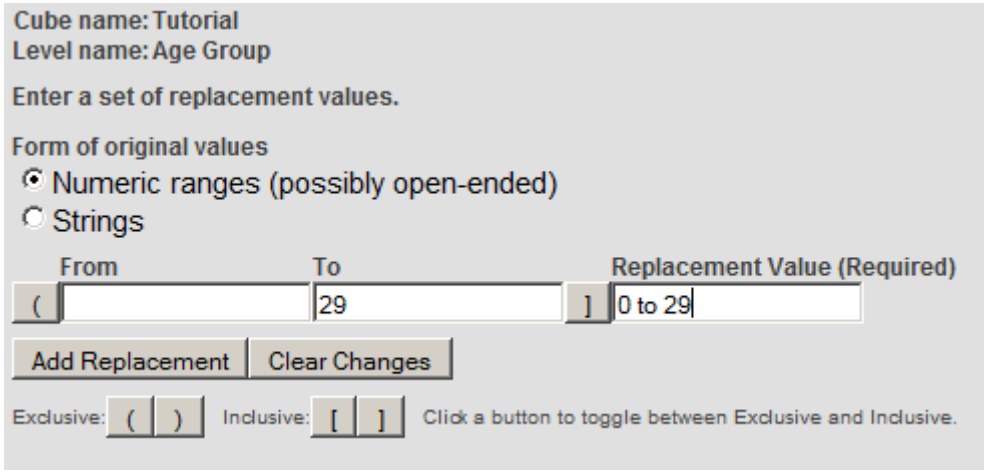
For numeric data, for each replacement, you specify a range of original values, as well as a new value to use instead.

e. Type `29` into **To**.

The button to the right of **To** is initially as follows: 

f. Click this button to change it to this: 

g. Type `0 to 29` into **Replacement Value**. The result is as follows:



It does not matter which button is next to **From**, because no value is specified for the lower end of this range.

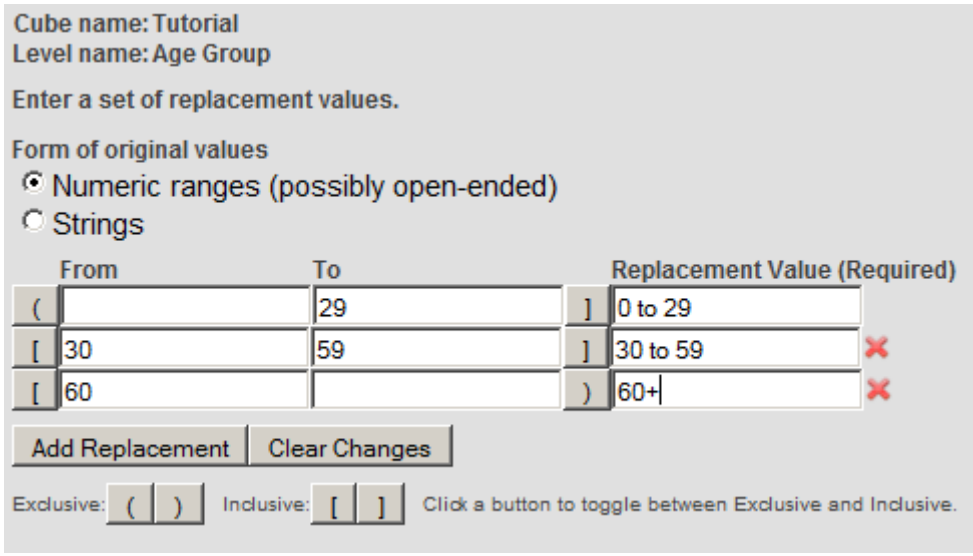
h. Click **Add Replacement**.

i. In the new row, click the toggle buttons next to **From** and **To**.

j. Type `30` into **From** and type `59` into **To**.

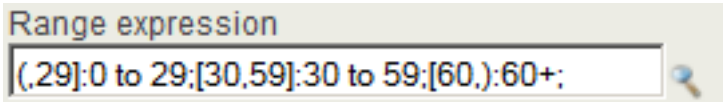
k. Type `30 to 59` into **Replacement Value**.

l. Click **Add Replacement** and add the final row so that the result is as follows:



m. Click **OK**.

The system closes the dialog box and displays a value in the **Range expression** field as follows:



This value shows the syntax that DeepSee uses internally to represent the replacements that you specified.

4. Save the cube.

For the `Age Bucket` level, we could use the same technique. Instead, however, we will use an alternative: a source expression that converts an age in years into a string that corresponds to the appropriate ten-year bucket.

5. In Studio, open the class `DeepSee.Model.PatientsCube`.

6. Look at the definition of the method `GetAgeBucket()`, which is as follows:

```
ClassMethod GetAgeBucket(age As %Numeric) As %String
{
    If (age="") {Set return=""}
    ElseIf (age<10) {Set return="0 to 9"}
    ElseIf (age<20) {Set return="10 to 19"}
    ElseIf (age<30) {Set return="20 to 29"}
    ElseIf (age<40) {Set return="30 to 39"}
    ElseIf (age<50) {Set return="40 to 49"}
    ElseIf (age<60) {Set return="50 to 59"}
    ElseIf (age<70) {Set return="60 to 69"}
    ElseIf (age<80) {Set return="70 to 79"}
    ElseIf (age>=80) {Set return="80+"}
    Else {Set return=""}
    Quit return
}
```

Notice that the input to this method is just a number, rather than a patient identifier.

7. In the Architect, add another level to `AgeD` as follows:

a. Click the `Age` level. This ensures that the new level, which is less granular, will be added before the `Age` level.

b. Click **Add Element**.

c. For **Enter New Element Name**, type `Age Bucket`.

d. Click **Level**.

e. Click **OK**.

The new level is added just before `Age`, but after `Age Group`.

f. For **Field name in fact table**, specify `DxAgeBucket`

This will make it easier for us to see how the level definition affects the generated tables.

g. For **Expression**, enter the following:

```
##class(DeepSee.Model.PatientsCube).GetAgeBucket(%source.Age)
```

Note: In practice, you are more likely to include utility methods in a central location such as the cube class that uses them (rather than some other cube as in this case). One point of this exercise is to demonstrate that you can invoke any class method that is accessible in this namespace. Similarly, you can invoke any routine or system function.

8. Save the cube.

Because you have edited the class in Studio, the Architect displays a dialog box that asks whether you want to override the stored definition. Click **OK**. The Architect overrides only the parts of the class definition that you can edit in the Architect; that is, it does not override any methods you have added to the class.

9. Compile the cube.
10. Rebuild the cube.
11. Access the Analyzer.

(If this is open on another browser tab, switch to that tab and click the **DeepSee > Analyzer** link to refresh with the most current model.)

12. Display the new `Age Group` level as rows. You should now see something like the following:

Age Group	
0 to 29	4,260
30 to 59	4,130
60+	1,610

13. Display the new `Age Bucket` level as rows. You should now see something like the following:

Age Bucket	
0 to 9	1,413
10 to 19	1,452
20 to 29	1,395
30 to 39	1,548
40 to 49	1,498
50 to 59	1,084
60 to 69	749
70 to 79	553
80+	308

14. Examine one of the new level tables to understand what the system has done:

a. Access the Management Portal and go to the SAMPLES namespace, as described earlier.

b. Click **System Explorer > SQL**.

c. In the left area, navigate to and open the table `Tutorial_Cube.Fact`.

This table now has three fields to store the values for the levels of the `AgeD` hierarchy:

DxAge	DxAgeBucket	DxAgeGroup
1	1	1
2	2	1
3	2	1
4	1	1
5	3	2
6	4	3

d. Navigate to and open the table `Tutorial_Cube.DxAgeGroup`.

The system displays something like the following:

#	ID	DxAgeGroup
1	1	0 to 29
2	2	60+
3	3	30 to 59
Complete		

The system used your range expression to create this data.

e. Open the table `Tutorial_Cube.DxAgeBucket`.

The system displays something like the following:

#	ID	DxAgeBucket	DxAgeGroup
1	1	10 to 19	1
2	2	20 to 29	1
3	3	70 to 79	2
4	4	50 to 59	3
5	5	30 to 39	3
6	6	80+	2
7	7	0 to 9	1
8	8	40 to 49	3
9	9	60 to 69	2
Complete			

Because this level is not at the top of the hierarchy, it contains a reference, for each element, to the its parent member in the `Age Group` level; see the `DxAgeGroup` column.

The system used the `GetAgeBucket()` method to create this data.

These two levels are defined in an equivalent fashion. That is, using the **Range Expression** option is equivalent to executing your own method to provide a conversion. A method can include logic that is much more complex than simple replacements. Consider the following method:

```
ClassMethod GetClassification(ID As %Numeric) As %String
{
    //get customer rating data & call duration from source record
    set customer=##class(MyPackage.MyClass).%OpenId(ID,0)
    set detail1=customer.Detail1
    set detail2=customer.Detail2
    set detail3=customer.Detail3
    ...

    If ...
        //logic to use these details and return a string, either "A", "B", or "C"
    Quit classification
}
```

You could use a method like this to populate a level that groups customers based on an algorithm that uses multiple pieces of information about the customers.

4.5 Accessing Other Classes

The DeepSee Architect provides easy access to most of the properties within the base 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 use data in the `DeepSee.Study.PatientDetails` class as levels in our cube.

The `DeepSee.Study.Patient` and `DeepSee.Study.PatientDetails` classes are not connected by a class property and do not have any formal connection. Instead, both tables have a `PatientID` property, which connects them by convention. That is, to find information for a given patient, you must find the records that have the same `PatientID` in these two tables.

In this exercise, we examine the data in `DeepSee.Study.PatientDetails`, try various SQL queries, and wrap a query in a method for use in defining a level. If you are more adept with SQL, you might want to skip some of the earlier steps.

1. Access the Management Portal and go to the SAMPLES namespace, as described earlier.
2. Click **System Explorer > SQL**.
3. Click the **Execute Query** tab.
4. Execute the following query:

   ```
   SELECT PatientID FROM DeepSee_Study.Patient
   ```

5. Make a note of one of the `PatientID` values, for future reference.
6. Execute the following query:

   ```
   SELECT * FROM DeepSee_Study.PatientDetails WHERE PatientID='SUBJ_100301'
   ```

   The system displays something like the following:

#	ID	FavoriteColor	PatientID	Profession
1	1	Blue	SUBJ_100301	
Complete				

7. Execute the following query:

   ```
   SELECT FavoriteColor FROM DeepSee_Study.PatientDetails WHERE PatientID='SUBJ_100301'
   ```

   The system displays something like the following:

#	FavoriteColor
1	Blue
Complete	

This query returns one value, the string `Blue`.

Now we need to write a class method that runs a similar query and returns the value obtained by the query.

This method will contain a query wrapped in `&sql()`. We need to make the following changes to the query:

- Instead of `FavoriteColor`, we must use `FavoriteColor INTO :ReturnValue` so that the returned value is written to a host variable named `ReturnValue`.
- Instead of using `'SUBJ_100301'`, we must pass in the `PatientID` field of the base class.

After executing the embedded SQL, the method should check the variable `SQLCODE`, which is 0 only for a successful query. The query would be unsuccessful if no record was found. In such a case, it would be appropriate to return an empty string.

8. In Studio, add the following method to your cube class, `Tutorial.Cube`:

```
ClassMethod GetFavoriteColor(patientID As %String) As %String
{
 &sql(SELECT FavoriteColor INTO :ReturnValue FROM DeepSee_Study.PatientDetails WHERE
PatientID=:patientID)
 If (SQLCODE'=0) {
  Set ReturnValue=""
  }
 Quit ReturnValue
}
```

Note: There is an index on the `PatientID` field in `DeepSee.Study.PatientDetails`. This enables the query to run more quickly than it would otherwise.

If an application does include tables that can be related most easily through SQL queries, as in this example, it probably already has indices on the relevant fields. Whenever you write a method like this, however, you should make sure that the appropriate indices exist.

9. Save and compile the class.
10. In the Terminal, test the method as follows:

```
SAMPLES>write ##class(Tutorial.Cube).GetFavoriteColor("SUBJ_100301")
Blue
```

11. Access the Architect.
12. Create a new dimension, hierarchy, and level, as follows:
    a. Click **Add Element**.
    b. For **Enter New Element Name**, type `ColorD`.
    c. Click **Data Dimension**.
    d. Click **OK**.

    The system creates a dimension, hierarchy, and level.

    e. Rename the level to `Favorite Color`.
    f. For **Field name in fact table**, specify `DxFavColor`

    This will make it easier for us to see how the level definition affects the generated tables.

g. For the level, type the following into **Expression**:

```
##class(Tutorial.Cube).GetFavoriteColor(%source.PatientID)
```

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

13. Save the cube.

Because you have edited the class in Studio, the Architect displays a dialog box that asks whether you want to override the stored definition. Click **OK**. The Architect overrides only the parts of the class definition that you can edit in the Architect; that is, it does not override any methods you have added to the class.

14. Compile the cube.

15. Rebuild the cube.

The system executes your method and its embedded SQL once for each record in the base table.

16. Open the Analyzer and display the new level as rows. Now you should see something like the following:

Favorite Color	
None	2,380
Blue	1,273
Green	1,268
Orange	1,278
Purple	1,308
Red	1,249
Yellow	1,244

17. Optionally open the level table for this level:

a. Access the Management Portal and go to the SAMPLES namespace, as described earlier.

b. Click **System Explorer > SQL**.

c. In the left area, navigate to and open the table `Tutorial_Cube.DxFavColor`.

The system displays something like the following:

#	ID	DxFavColor
1	1	Blue
2	2	Red
3	3	Green
4	4	Yellow
5	5	<null>
6	6	Orange
7	7	Purple
Complete		

5

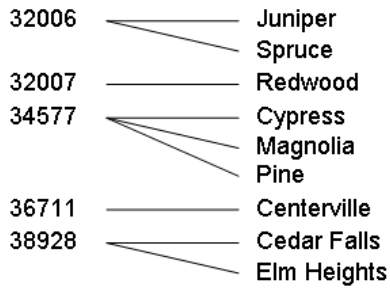
Creating Subject Areas

A subject area is a subcube with optional overrides to names of items. You define a subject area to enable users to focus on smaller sets of data, for security reasons or other reasons. This chapter discusses the following topics:

- Introduction
- Creating the subject areas
- Examining the subject areas
- A look at common filter expressions

5.1 Introduction

In this tutorial, we create two subject areas that divide the patient data by ZIP code. In the Patients sample, ZIP codes contain small cities as follows:



We will create the following subject areas:

Subject Area Name	Contents
`Patient Set A`	Patients who live in ZIP codes 32006, 32007, or 36711
`Patient Set B`	Patients who live in ZIP codes 34577 or 38928

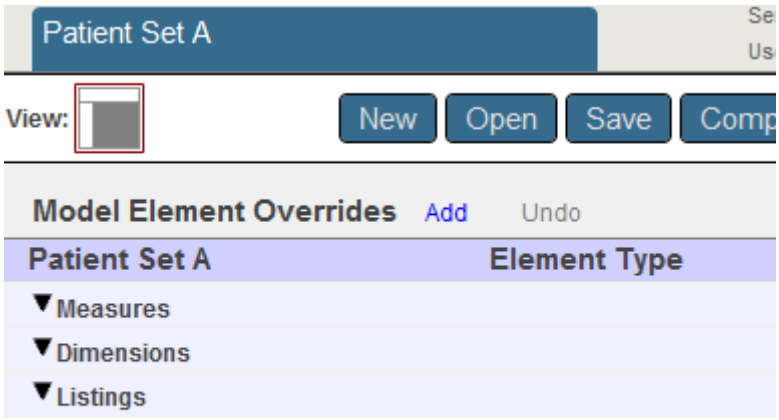
5.2 Creating the Subject Areas

To create the subject areas, do the following:

1. In the Architect, click **New**.
2. Click **Subject Area**.
3. For **Subject Area Name**, type `Patient Set A`
4. For **Class name for the Subject Area**, type `Tutorial.SubjectA`
5. For **Base Cube**, click **Browse** and select `Tutorial`
6. Click **OK**.

   The system creates the subject area and saves the class.

   You should see the following:



   In the Architect, there is no user interface for defining a filter. Instead it is necessary to type a suitable filter expression or to copy and paste one from the Analyzer.

7. In a separate browser tab or window, access the Analyzer and then do the following:

   a. Expand `HomeD`.

   b. Drop `ZIP Code` to the **Filters** box. This adds a filter box directly above the pivot table.

   c. In that filter box, click the search button and then select `32006`, `32007`, and `36711`.

      Click the check mark.

      This action filters the pivot table.

      Important: Do not drag and drop `32006`, `32007`, and `36711` separately to the **Filters** box. Instead drag the level as described and then select the members.

   d. Click the Query Text button  .

      The system then displays a dialog box that shows the MDX query that the Analyzer is using:

```
SELECT  FROM [Patients]
%FILTER %OR({[HOMED].[H1].[ZIP Code].&[32006],[HOMED].[H1].[ZIP Code].&[32007],[HOMED].[H1].[ZIP
 Code].&[36711]})
```

   e. Copy the text after `%FILTER` to the system clipboard.

   f. Click **OK**.

8. In the Architect, click the line labeled `Patient Set A`.
9. In the Detail Pane, paste the copied text into **Filter**.

```
%OR({[HOMED].[H1].[ZIP Code].&[32006],[HOMED].[H1].[ZIP Code].&[32007],[HOMED].[H1].[ZIP
Code].&[36711]})
```

10. Click **Save** and then click **OK**.
11. Compile the subject area.
12. For the second subject area, repeat the preceding steps, with the following changes:
    - For **Subject Area Name**, type `Patient Set B`
    - For **Class name for the Subject Area**, type `Tutorial.SubjectB`
    - Repeat the preceding steps with the other two ZIP codes. So, for **Filter**, use the following:

      ```
      %OR({[HOMED].[H1].[ZIP Code].&[34577],[HOMED].[H1].[ZIP Code].&[38928]})
      ```

5.3 Examining the Subject Areas

Now we examine the subject areas that we have created. The numbers you see will be different from those shown here.

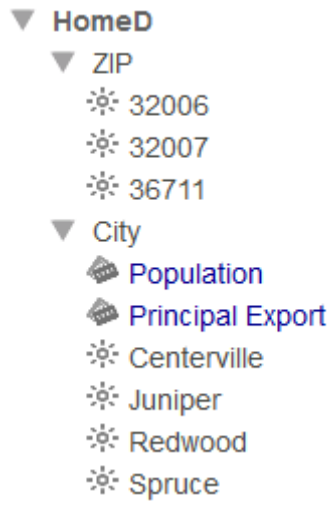
1. In the Analyzer, click the Change button  .
2. Click `Patient Set A`.
3. Click **OK**.

   The Analyzer then displays the contents of the selected subject area.

   Notice that the total record count is not as high as it is for your base cube:

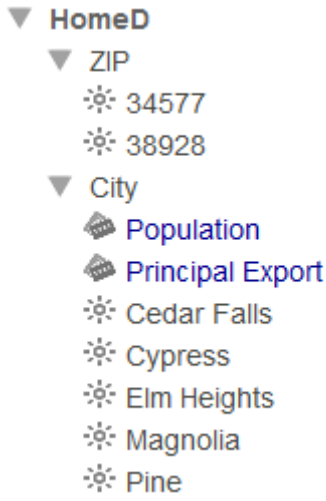
	All
Count	4,490

4. In the Model Contents area, expand the `HomeD` dimension, `ZIP Code` level, and `City` level. You should see the following:



5. Repeat the preceding steps for `Patient Set B`.

   When you expand the `HomeD` dimension, `ZIP Code` level, and `City` level. You should see the following:



5.4 Common Filter Expressions

In this section, we experiment with common filters in the Analyzer and see their effect on the generated queries.

1. In the Analyzer, open the `Tutorial` cube.

   The Analyzer refers to both cubes and subject areas as subject areas. The formal distinction between them is relevant only when you are creating them.

2. Click **New**.

   The Analyzer displays Count (a count of the records):

	All
Count	10,000

Before we add a filter, let us see how the query is currently defined, so that we have a basis of comparison.

3. Click the Query Text button  .

   The system then displays a dialog box that shows the MDX query that the Analyzer is using:

   ```
   SELECT FROM [TUTORIAL]
   ```

4. Click **OK**.
5. Expand `ColorD` and `Favorite Color`.
6. Drag and drop `Orange` to **Filters**.

   The Analyzer now uses only patients whose favorite color is orange. It looks something like this:

	All
Count	1,303

7. Click the Query Text button  .

   The system then displays the following query:

   ```
   SELECT FROM [TUTORIAL] %FILTER [COLORD].[H1].[FAVORITE COLOR].&[ORANGE]
   ```

   The `%FILTER` keyword restricts the query. The fragment after `%FILTER` is a filter expression:

   ```
   [COLORD].[H1].[FAVORITE COLOR].&[ORANGE]
   ```

   This filter expression is a member expression, because it refers to a member (the `Orange` member of the `Favorite Color` level). A member is a set of records, and a member expression refers to that set of records.

   Notice that this expression uses the dimension, hierarchy, and level names. The `&[ORANGE]` fragment refers to the key of the `Orange` member. The Analyzer uses keys rather than names, but you can use either if the member names are unique.

8. Click **OK**.
9. Add another color to the filter. To do so:
   a. Click the X next to `Orange` in **Filters**.

      This removes that filter.

   b. Drag and drop `Favorite Color` to **Filters**. This adds a filter box directly above the pivot table.
   c. In that filter box, click the search button and then select `Orange` and `Purple`.
   d. Click the check mark.

      This action filters the pivot table.

      Important: Do not drag and drop `Orange` and `Purple` separately to the **Filters** box. Instead drag the level as described and then select the members.

      The Analyzer now looks something like this:

	All
Count	2,558

The system now uses only patients whose favorite color is orange or whose favorite color is purple. (Notice that the count is higher than it was for orange alone.)

10. Display the query text again. Now you should see the following:

```
SELECT FROM [TUTORIAL]
%FILTER %OR({[COLORD].[H1].[FAVORITE COLOR].&[ORANGE],[COLORD].[H1].[FAVORITE COLOR].&[PURPLE]})
```

    In this case, the filter expression is as follows:

```
%OR({[COLORD].[H1].[FAVORITE COLOR].&[ORANGE],[COLORD].[H1].[FAVORITE COLOR].&[PURPLE]})
```

    The %OR function is an InterSystems optimization; the argument to this function is a set.

    The set is enclosed by curly braces `{}` and consists of a comma-separated list of elements. In this case, the set contains two member expressions. A set expression refers to all the records indicated by the elements of the set. In this case, the set refers to all patients whose favorite color is orange and all patients whose favorite color is purple.

11. Click **OK**.

12. Use the filter drop-down list and clear the check box next to `Purple`.

    Now the Analyzer uses only patients whose favorite color is orange.

13. Expand `AllerD` and `Allergies`.

14. Drag and drop `mold` to **Filters**, beneath `Orange`.

    The Analyzer now looks something like this:

	All
Count	52

    Notice that the count is lower than we saw using just `Orange` alone. This pivot table displays only patients whose favorite color is orange and who are allergic to mold.

15. Display the query text again. Now you should see the following:

```
SELECT FROM [TUTORIAL]
%FILTER NONEMPTYCROSSJOIN([ALLERD].[H1].[ALLERGIES].&[MOLD],[COLORD].[H1].[FAVORITE COLOR].&[ORANGE])
```

    In this case, the filter expression is as follows:

```
NONEMPTYCROSSJOIN([ALLERD].[H1].[ALLERGIES].&[MOLD],[COLORD].[H1].[FAVORITE COLOR].&[ORANGE])
```

    The MDX function NONEMPTYCROSSJOIN combines two members and returns the resulting *tuple*. The tuple accesses only the records that belong to both of the given members.

Now you have seen the three most common filter expressions:

member expression

When you use a member expression as a filter, the system accesses only the records that belong to this member.

You can write a member expression as follows:

```
[dimension name].[hierarchy name].[level name].&[member key]
```

Or:

```
[dimension name].[hierarchy name].[level name].[member name]
```

Where:

- *dimension name* is a dimension name.
- *hierarchy name* is a hierarchy name. You can omit the hierarchy name. If you do, the query uses the first level with the given name, as defined in this dimension.
- *level name* is the name of a level within that hierarchy. You can omit the level name. If you do, the query uses the first member with the given name, as defined within this dimension.
- *member key* is the key of a member within the given level. This is often the same as the member name.
- *member name* is the name of a member within the given level.

set expression

When you use a set of members as a filter, the system accesses the records that belong to any of the given members. That is, the members are combined with logical OR.

You can write a set expression that refers to members as follows:

```
{member_expression,member_expression,member_expression...}
```

Where *member_expression* is a member expression.

tuple expression

When you use a tuple as a filter, the system accesses the records that belong to all of the given members. That is, the members are combined with logical AND.

You can write a tuple expression as follows:

```
NONEMPTYCROSSJOIN(member_expression,member_expression)
```

Or:

```
(member_expression,member_expression)
```

For additional variations, see *Using MDX with DeepSee* and *DeepSee MDX Reference*.

6

Creating and Packaging Pivot Tables and Dashboards

After creating one or more cubes, you typically create and package a set of initial pivot tables and dashboards, and users typically create new pivot tables and dashboards as needed.

This chapter briefly leads you through the process of creating pivot tables and dashboards. It consists of the following steps:

1. Creating pivot tables
2. Creating a dashboard
3. Exporting and packaging these items

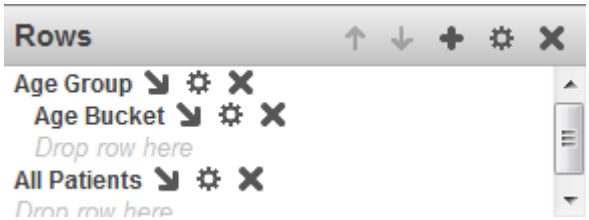
For much more information on creating pivot tables and dashboards, see *Using the DeepSee Analyzer* and *Creating DeepSee Dashboards*.

6.1 Creating Pivot Tables

Earlier in this tutorial, we created a pivot table that uses the Patients cube. Now let us create pivot tables that use your new cube.

1. Access the Analyzer.
2. Navigate to the `Tutorial` cube.
3. Expand the `AgeD` dimension in the Model Contents pane.
4. Drag and drop `Age Group` to **Rows**.
5. Drag `Age Bucket` to **Rows** and drop it onto the Breakout Here button  .
6. Drag and drop `Count` to **Columns**.
7. Drag and drop `All Patients` to **Rows**, at the bottom.

   The **Rows** box now looks like this:



The pivot table is as follows:

Age Group		Count
0 to 29	0 to 9	1,459
	10 to 19	1,428
	20 to 29	1,335
30 to 59	30 to 39	1,524
	40 to 49	1,520
	50 to 59	1,090
60+	60 to 69	688
	70 to 79	604
	80+	352
All Patients		10,000

8. Click **Save**.

   The system displays a dialog box where you specify the pivot table name.

   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.

9. For **Folder**, type `Tutorial`
10. For **Pivot Name**, type `Patients by Age Group`
11. Click **OK**.
12. Create another pivot table as follows:
    - For **Rows**, use `Diagnoses`
    - For **Columns**, use `Count` and `Avg Age`
    - For **Folder**, select or type `Tutorial`
    - For **Pivot Name**, type `Patients by Diagnosis`

6.2 Creating a Dashboard

In this section we create a simple dashboard.

1. Click the **DeepSee** link at the top of the page.
2. Click **Home,DeepSee,User Portal**.

   The system then displays the User Portal, which lists the existing public dashboards and pivot tables.

3. Click **Menu > New Dashboard**.

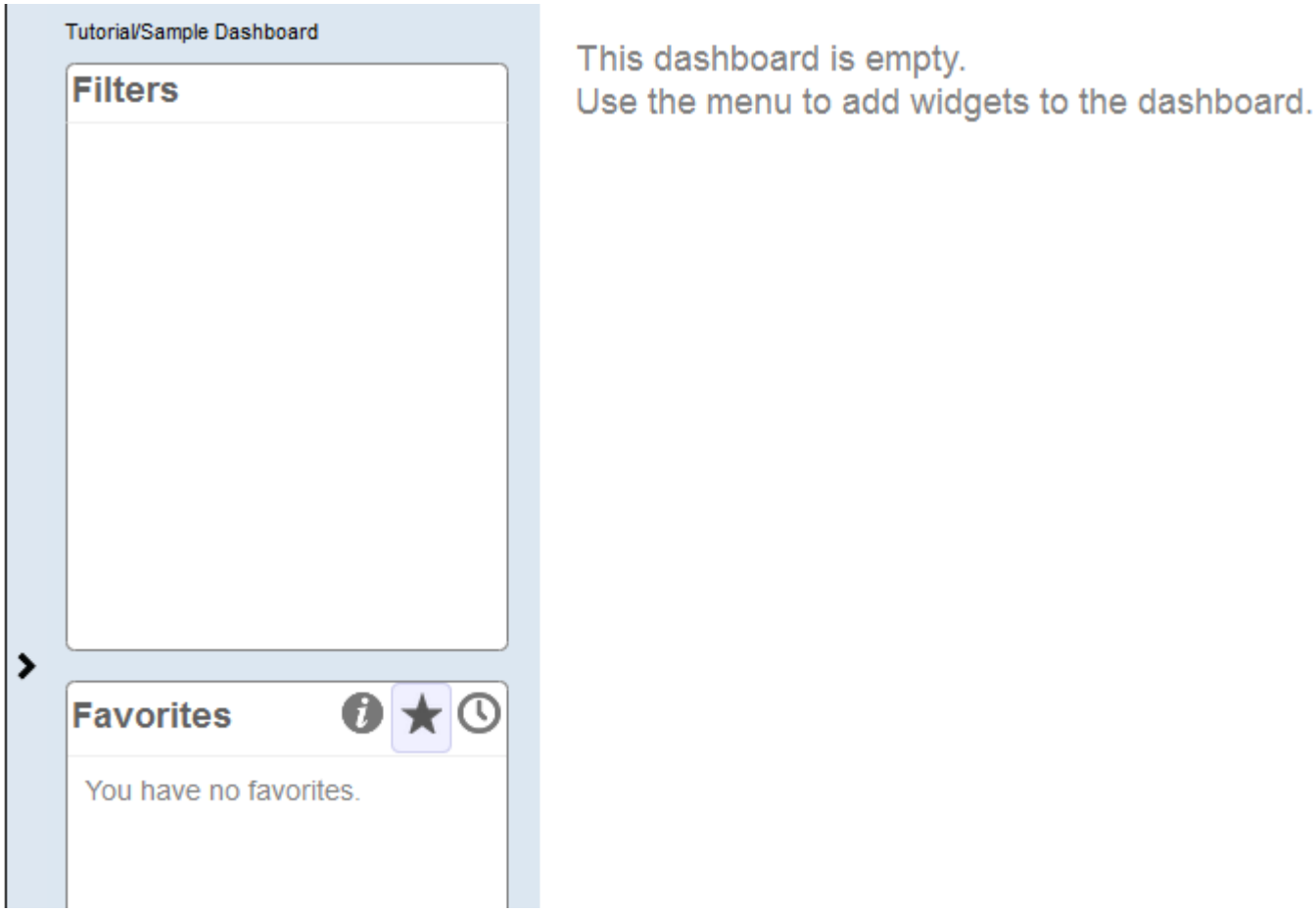
   The system displays a dialog box that prompts you for basic information about the new dashboard.

4. For **Folder**, type or select `Tutorial`

5. For **Dashboard Name**, type `Sample Dashboard`

6. Click **OK**.

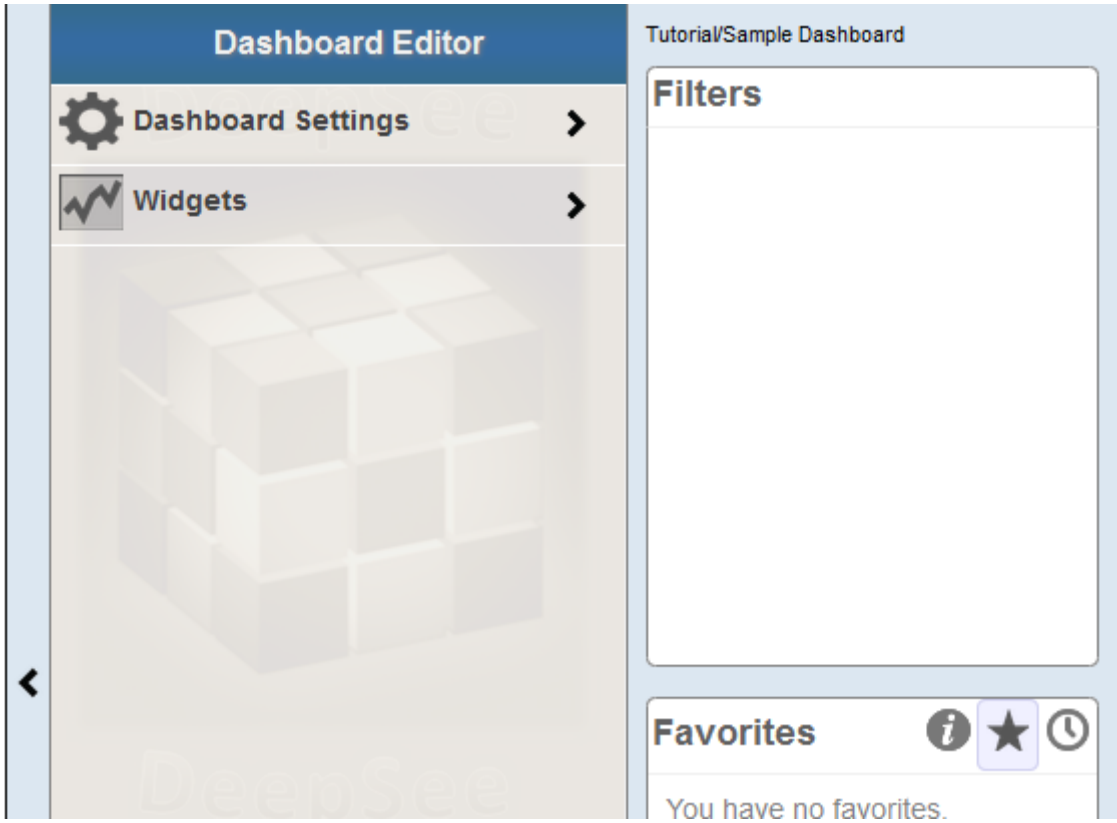
   The system creates, saves, and displays the dashboard, which is initially empty.



   Notice the > button on the left side of the dashboard.

7. Click the > button.

   This step expands the Dashboard Editor, as follows:



8. Click **Widgets**.

   This step displays a list of any widgets that this dashboard currently contains (none in this case).

9. Click the + button.

   This step displays a dialog box where you can choose some initial options.

10. In the dialog box, click **Pivots and Charts**, if this is not already expanded. The list expands to show a list of choices.

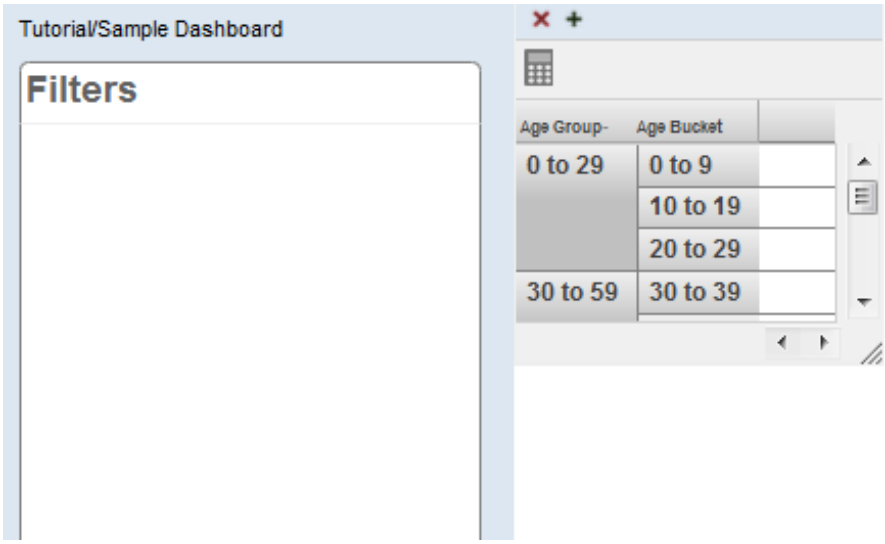
11. In this list, click **Table**.

12. Click the Search button next to **Data source**.

13. Click `Tutorial` and then click `Patients by Age Group`

14. Click **OK** to add this widget.

    The system displays the dashboard like this:



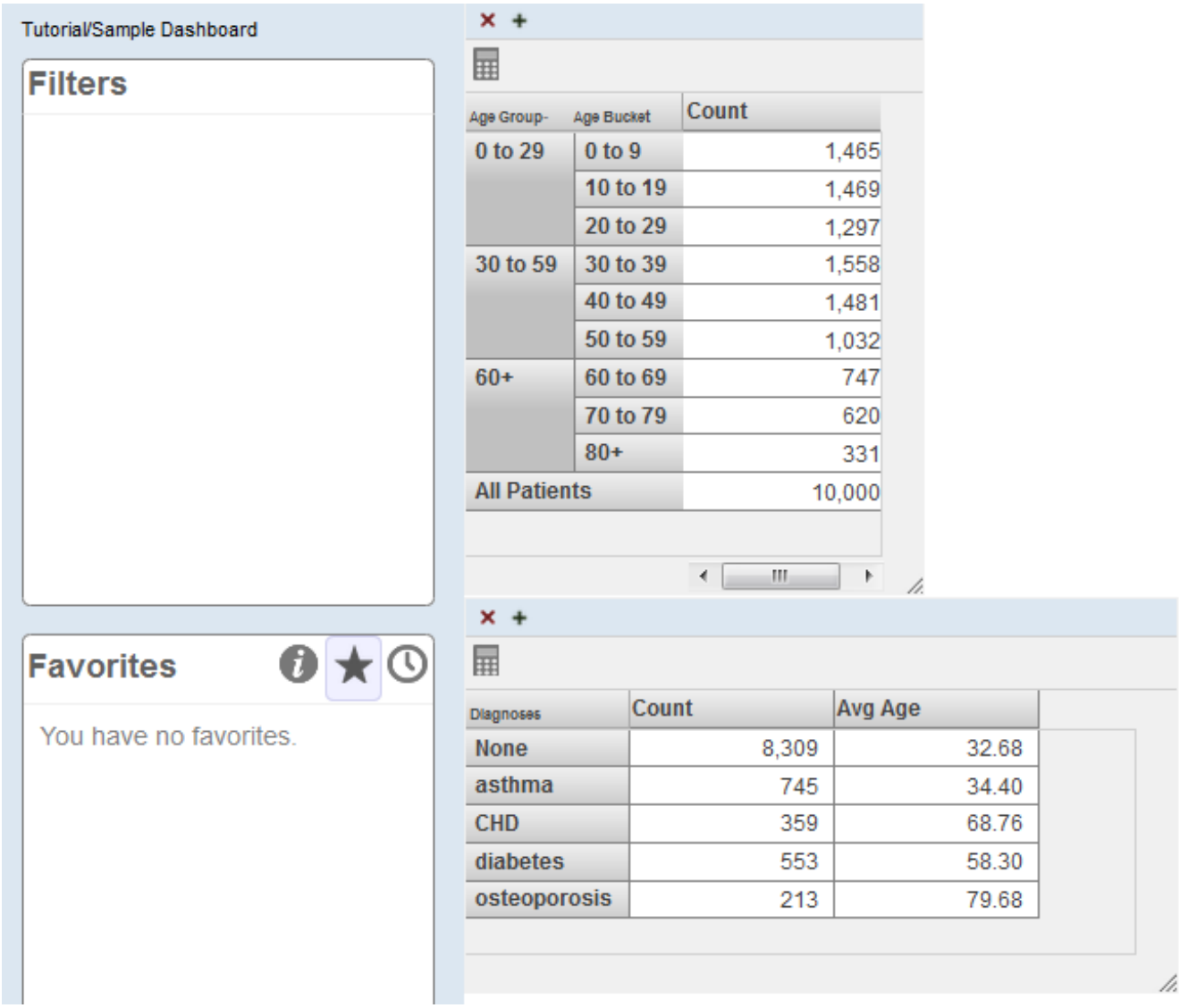
15. Repeat the preceding steps to add the other pivot table that you created in the previous section.

    The newly added widget is placed in the same default position (in the upper left) and therefore covers the previously added widget.

16. Put the cursor into the title bar of the newly added widget and then drag this widget and drop it below the other widget.

    You might want to leave space below the upper widget so that you can resize it.

17. Use the resize control in the lower right corner of each widget to resize it so that all rows are visible without scrolling.



18. Click **Menu > Save**.

    The system saves the dashboard immediately.

19. Do the following to add some filter controls to this dashboard:

    a. Open the Dashboard Editor.

    b. Click **Widgets**.

    c. Click the `Patients by Age Group` widget.

    d. Click **Controls**.

    The Dashboard Editor displays a list (currently empty) of any controls defined on the selected widget.

    e. Click the plus sign button above the list.

    The system displays a dialog box where you specify the control.

    f. For **Location**, select **Dashboard**.

    g. For **Target**, type *

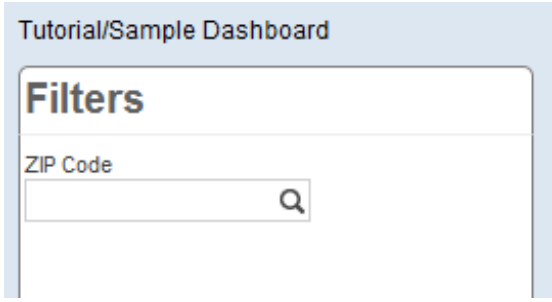
    h. For **Action**, select **Apply Filter**.

i. For **Filter**, select `ZIP Code`

j. For **Control Label or Icon**, type `ZIP Code`

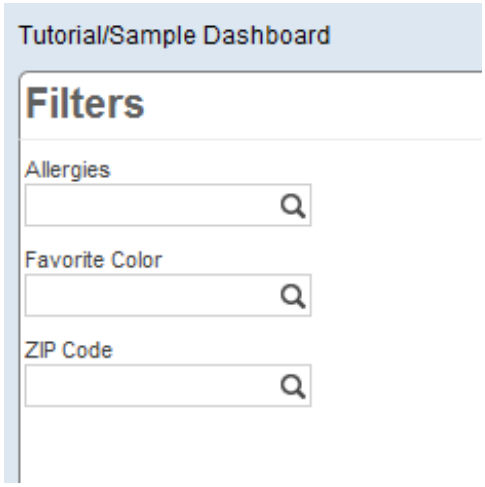
k. Click **OK** to add the control.

Now the upper left corner of the dashboard is as follows:



20. Repeat the preceding steps to add the `Allergies` and `Favorite Color` filters.

    Now the upper left corner of the dashboard is as follows:



21. Reconfigure these filter controls so that each one has a default value. To reconfigure the `ZIP Code` filter:

    a. Open the Dashboard Editor.

    b. Access the definition of the `Patients by Age Group` widget.

    c. Click **Controls**.

    d. Click the `ZIP Code` control.

    e. For **Default Value**, type `32007` and click the check mark.

    f. Click **Done**.

    For `Allergies`, use the default value `soy`

    For `Favorite Color`, use the default value `blue`

    As you make these changes, the system automatically saves them.

22. Test the dashboard and verify the following:

    - If you use the browser's refresh button, each filter should show the correct default value.

- Each filter should affect both widgets.

6.3 Exporting and Packaging the Pivot Tables and Dashboards

1. Click **Menu > Management Portal**.

   If you do not see this option, you are currently logged in as a user without direct access to the DeepSee development tools. In this case, log in again to DeepSee as described earlier in this book.

2. Click **DeepSee > Admin > Folder Manager**.

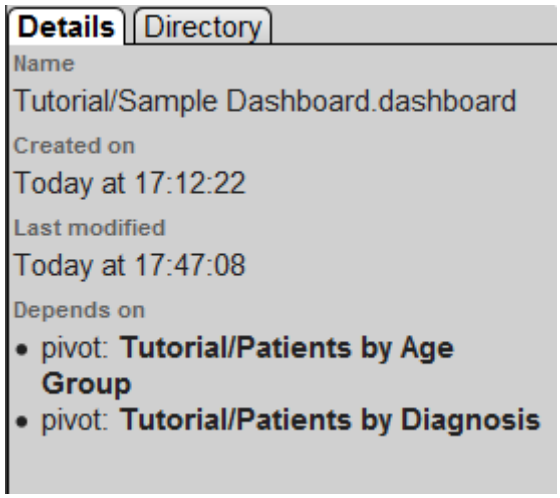
   The system then displays the Folder Manager.

3. Scroll to the `Tutorial` folder, which should look something like this:

	Tutorial	Folder	
☐	Patients by Age Group	Pivot	_SYSTEM
☐	Patients by Diagnosis	Pivot	_SYSTEM
☐	Sample Dashboard	Dashboard	_SYSTEM

   The column on the right indicates the Caché user who created these items; you may see different values there.

4. Click the check box to the left of `Sample Dashboard`. The left area of the page now lists the pivot tables used by this dashboard:

   Both of these pivot tables are in the same folder as the dashboard itself, which is not required.

5. Click the check box to the left of the two newer pivot tables as well.

6. For **Directory**, type the name of an existing directory to which you have write permission.

7. Click **Export**.

   The system then writes three files to that directory.

You could use these files as a package for the items you have defined, but in the next set of steps, we will use a more convenient approach.

8. In Studio, create a new class called `Tutorial.DashboardsAndPivots`. The class should extend %DeepSee.UserLibrary.Container.

9. In the new class, add an XData block named `Contents`, as follows:

```
XData Contents [ XMLNamespace = "http://www.intersystems.com/deepsee/library" ]
{
<items>
</items>
}
```

10. Save the new class.

11. Copy the dashboard and pivot table definitions from your exported files into this XData block:

    a. Use a text editor to open the dashboard file (`Tutorial-Sample_Dashboard.xml`), which looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<dashboard xmlns="http://www.intersystems.com/deepsee/library" folderName="Tutorial" name="Sample
 Dashboard" ...
...
</dashboard>
```

    b. Copy the contents of this file, starting with the second line and ending at the end of the file. That is, do not copy the first line, which is the XML declaration.

    c. Paste the copied text into the XData block between `<items>` and `</items>`. Now you should see this:

```
XData Contents [ XMLNamespace = "http://www.intersystems.com/deepsee/library" ]
{
<items>
   <dashboard xmlns="http://www.intersystems.com/deepsee/library" folderName="Tutorial"
name="Sample Dashboard" ...
   ...
   </dashboard>
</items>
}
```

    d. Use a text editor to open one of the pivot table files, which has a similar structure (with `<pivot>` instead of `<dashboard>`).

    e. Copy the contents of this file, , starting with the second line and ending at the end of the file.

    f. Paste the copied text into the XData block between `</dashboard>` and `</items>`. Now you should see something like this:

```
XData Contents [ XMLNamespace = "http://www.intersystems.com/deepsee/library" ]
{
<items>
   <dashboard xmlns="http://www.intersystems.com/deepsee/library" folderName="Tutorial"
name="Sample Dashboard" ...
   ...
   </dashboard>
  <pivot xmlns="http://www.intersystems.com/deepsee/library" folderName="Tutorial" name="Patients
 by Age Group" ...
   ...
   </pivot>
</items>
}
```

    Note: You could instead insert the copied text between `<items>` and `<dashboard>`. The order of the items has no effect on anything.

    g. Repeat the preceding steps with the other pivot table file.

    h. Save the class definition.

12. Return to the Folder Manager and refresh the page.

    This clears any selections you may have made.

13. Click the check box next to each of the three items in the `Tutorial` folder.
14. Click **Delete** and then click **OK** to confirm.
15. In Studio, compile the class you just created.

    When you do this, the system imports the dashboard and pivot table definitions contained in that class.

16. Return to the Folder Manager and refresh the page. Notice that the tutorial dashboard and pivot tables are available again.

You can include as many dashboards and pivot tables as needed in a container class like this, and you can have multiple container classes. There are no requirements to organize the dashboards and pivot tables in any particular way in these container classes. For example, you do not need to place the pivot tables in the same container class as the dashboards that use them. There is also no requirement to create these container classes at all.