CHAPTER 2: MASTER TABLES AND FORMS

Objectives

The objectives are:

- Using formalized analysis and design methodology.
- Understanding master table and form standards.
- Working with table event triggers.
- Working with the complex data types and their member functions.
- Enabling multi-language functionality.

Introduction

For the solution in this course, several use cases are hinted at in the functional requirements in Chapter 1. For example, the functional requirements state, "Customers can register one or more participants for a seminar." This is a use case, albeit a simple one that will probably require additional information, such as exactly what information is necessary to sign up each participant.

In this chapter the requirements for master tables and forms for the solution will be discussed before moving on to design and implementation phases. The planning documentation in Chapter 1 is used as a reference, particularly for the analysis phase.

A Quick Refresher

Tables

Tables define the structure for and storage of the data upon which a solution relies. Tables are stored in a database – either the Microsoft Dynamics NAV database or SQL server, depending on the installation, but it makes little difference to the application code. (There are performance implications depending on the database being used – this is discussed later in this course.) Master tables define the data-foundation upon which a solution relies. A solution may use additional tables, but non-master tables contain information that is derived from the master tables.

In this chapter a number of master tables will be created to support the data required to implement the solution.

Forms

Master forms are forms that support the display and modification of the data stored by the master tables. There are two types of forms in Microsoft Dynamics NAV: Card forms and List forms. Both types are explicitly tied to master tables. The primary difference is that card forms display one record at a time, and list forms display multiple records at one time.

Triggers

A trigger is a method whose execution is based on system events. Triggers should not be executed explicitly – they are executed automatically, depending on the type of trigger. Triggers serve as a way to define code that is executed automatically whenever specific events occur.

There is one exception: documentation triggers. These triggers contain no code, and are not executed. Instead they are used to author internal documentation.

The two types of executable triggers are *event triggers*, and *function triggers*. Event triggers always have names that start with "On", and are executed when specific events occur. Function triggers are executed whenever a function is executed. All functions within objects support function triggers.

In this chapter, triggers are used primarily to validate information that is entered by the user.

Multi-language Support

Microsoft Dynamics NAV is multi-language enabled. This means that a localized version of Microsoft Dynamics NAV can present itself in different languages. Users can change the language that is used to display texts, and the change is immediate. There is no need to stop and restart Microsoft Dynamics NAV.

Before you start working in a multi-language-enabled database, set the working language to English – United States. Click TOOLS→LANGUAGE and select English – United States.

To enable multi-language functionality when developing solutions, there are a few guidelines to follow. When creating objects in Microsoft Dynamics NAV, the **Name** property of an object must always be in English – United States (ENU), but it should also never be visible to the user. Instead, language-specific names should be provided in the **CaptionML** property.

In this chapter, multi-language support is enabled in some of the labs.

Customers and Participants

As a solution that provides a way to schedule seminars that are attended by participants and paid for by customers, these two entities play a key role. In this section the strategy for implementing these entities will be discussed.

Solution Analysis

Even at this early state, it is clear that representing customers and seminar participants will require master tables and forms, but it is important to resist the temptation to create these items immediately. As will become evident, this is important to customers and seminar participants in particular.

Regarding customers and seminar participants, the functional requirements in Chapter 1 stated:

Seminar participants come from a company that is set up as a customer in Microsoft D, but must be handled separately from the customers. Every customer can register several participants for a seminar but participants cannot be registered unless they are connected with a customer. This is necessary because you want to invoice customers for the participation at seminars.

This means that there must be a one-to-many relationship between customers and seminar participants. This will be important during the design phase.

The functional requirements also mention the need to invoice customers. This implies that a table with only a company name will not suffice. The table used to represent customers will need to be fairly robust, complete with addresses, phone numbers, accounts numbers, and so forth. The exact requirements would require an analysis of all of the solution elements, especially reports. That will not be necessary, however.

Solution Design

Before moving into the implementation phase and creating new tables and forms, determine if any existing elements can be used instead, saving development and debugging time.

Microsoft Dynamics NAV has many built-in tables and forms, and there are some that will suffice for representing customers and seminar participants. These tables and forms are:

Table	Form
18 Customer	21 Customer Card
	22 Customer List
5050 Contact	5050 Contact Card
	5052 Contact List

The **Customer** tables and forms can be used to represent the customers that enroll seminar participants, and the participants can be represented by the **Contact** tables and forms.

Additionally, there is a one-to-many relationship between the **Customer** and **Contact** tables in Microsoft Dynamics NAV, so this fits the requirements for this solution.

Implementation

In this case, because of the robust analysis and design phases, no implementation is necessary.

Rooms

The next step is to create the tables to manage the seminar rooms in which the seminars are held.

Solution Analysis

The client's functional requirements describe the management of seminar rooms in this way:

Each seminar is held in a seminar room. Some are held in-house and some are held off-site. If a seminar takes place in-house, a room must be assigned. For off-site rooms, the rental rate must be tracked as well.

This description makes it clear that the table and forms used to represent seminar rooms need – at minimum – the location and number of the room, whether it is on or off site, and the rental rate.

Solution Design

The analysis of the room management process shows that you only need one basic table and form. The tracking of prices and costs is the complex part of representing rooms. However, the **Resource** table (156) provides a way of tracking this kind of information for company resources such as employees or machinery. The **Resource** table, along with its built-in behaviors, can be leveraged, eliminating the need to duplicate this functionality in the new **Seminar Room** table. The new table will be associated with both the **Contact** table and the **Resource** table, like this:

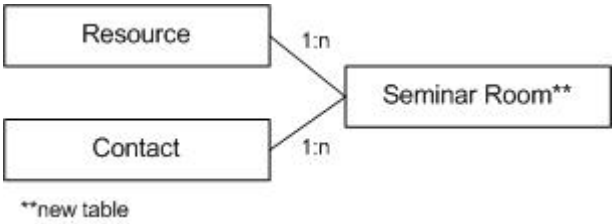

FIGURE 2-1: RELATIONSHIP BETWEEN SEMINAR ROOMS, RESOURCES, AND CONTACTS

Before proceeding, review the forms as they appear when implementation is complete.

The **Seminar Room Card** form (Figure 2-2) displays one room at a time, and can be used to modify the data in the **Seminar Room** table.

FIGURE 2-2: SEMINAR ROOM CARD FORM (123456703): GENERAL TAB

The **Seminar Room Card** form has two tabs: the standard **General** tab, and a **Communication** tab, which looks like this:

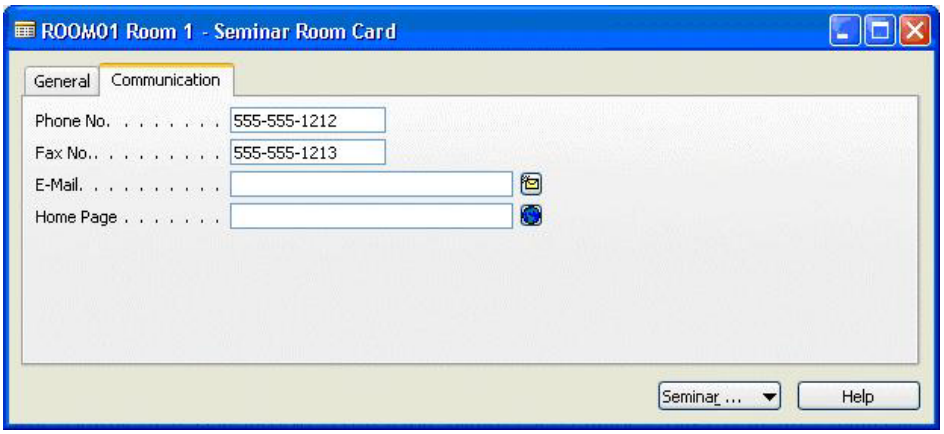

FIGURE 2-3: SEMINAR ROOM CARD FORM (123456703): COMMUNICATION TAB

The **Seminar Room List** form (123456704) allows multiple rooms to be displayed at once. It provides an overview of the seminar rooms, but does not allow the underlying **Seminar Room** table to be modified.

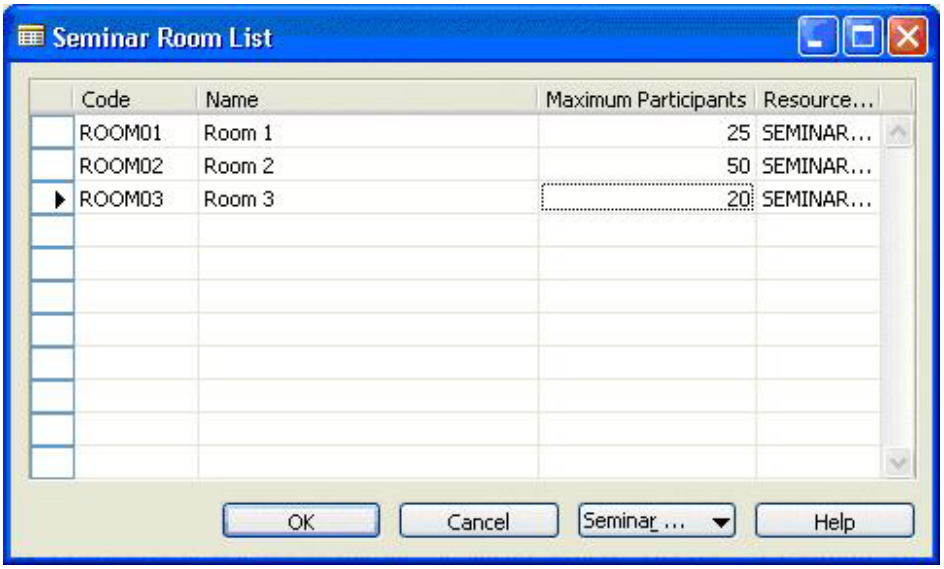

FIGURE 2-4: SEMINAR ROOM LIST (FORM 123456704)

Lab 2.1 - Creating Seminar Room Tables and Forms

Microsoft Dynamics NAV provides a **Comment** table you can use with your tables, which – when used – provides development documentation that can be invaluable both to other developers and to the developer that created the functionality for subsequence upgrades and released. It is also a good idea to use Microsoft Dynamics NAV's extended text functionality. You can learn more about this functionality in the Online Help. Both comments and extended text are necessary in order to adhere to standards.

Follow these steps to complete this lab:

1. In the **Comment Line** table (97) , add the option **Seminar Room** to the options for the field **Table Name**.
2. In the **Extended Text Header** table (279) and the **Extended Text Line** table (280), add the option **Seminar Room** to the options for the field **Table Name**.
3. Create the **Seminar Room** table (123456702) with the following fields:

No.	Field Name	Type	Length	Comment
1	Code	Code	10	Must not be blank.
2	Name	Text	30	
3	Address	Text	30	
4	Address 2	Text	30	
5	City	Text	30	
6	Post Code	Code	20	Relation to the **Post Code** table (225).
7	Country Code	Code	10	Relation to table 9 Country.
8	Phone No.	Text	30	
9	Fax No.	Text	30	
10	Name 2	Text	50	
11	Contact	Text	50	
12	E-Mail	Text	80	
13	Home Page	Text	90	

No.	Field Name	Type	Length	Comment
14	Maximum Participants	Integer		
15	Allocation	Decimal		Must not be editable.
16	Resource No.	Code	20	Relation to table 156 Resource, where Type=Machine.
17	Comment	Boolean		FlowField; checks whether any lines exist on the **Comment Line** table for the **Seminar Room** table and the corresponding seminar room code. Must not be editable.
18	Internal/External	Option		Options: Internal, External
19	Contact No.	Code	20	Relation to the **Contact** table (5050)

– The primary key is **Code**. Set the property to specify form 123456704 as the lookup form for this table.

HINT: *For all fields, set the Caption value to the Name. To save time, use F8 (Copy Previous shortcut).*

4. Use the wizard to create the Seminar Room Card form (123456703) as shown in the previous section. Set the property to specify that this form will be updated when activated.

   – Add a menu button and menu items as follows:

Menu Button	Options	Comment
Seminar Room	List (f5)	Opens the lookup form.
	Comments	Opens the **Comment Sheet** form (124) for the selected entry.
	Extended Texts	Opens the **Extended Text** form (386) for the selected entry.

   – Add a command button next to the **Code** field to provide access to the **Comment Sheet** for the corresponding record.

***HINT**: Copy the command button with the pencil picture from the existing form 21 Customer Card and paste it into your new form. (The local variables for these objects will be copied as well.) Set the RunFormLink property for this button so that only the comments corresponding to the selected Seminar Room are displayed.*

- Add a command button next to the **E-mail** field from which an e-mail can be sent.

***HINT:** Copy the command button from the existing form 21 Customer Card and paste it into the new form.*

- Add a command button next to the **Home Page** field that will open a hyperlink entered into the field.

***HINT:** Copy the command button from the existing form 21 Customer Card and paste it into the new form.*

Use the wizard to create the Seminar Room List form (123456704) as shown in the GUI design. The only fields necessary on this form are Code, Name, Maximum Participants, and Resource No.

- Set the properties to specify that the lines on this form are not editable and that the **Name** field will be "glued" to both sides of the form so that it expands with the form.
- Add a menu button and menu items as follows:

Menu Button	Options	Comments
Seminar Room	Card (shift + f5)	Opens the **Seminar Room Card** form (123456703) for the selected entry. The content of the card should change when the user selects a different entry in the list form.
	Comments	Opens the **Comment Sheet** form (124) for the selected entry.
	Extended Texts	Opens the **Extended Texts** form (386) for the selected entry.

NOTE: *When creating tabular-type forms with the wizard in the Object Designer, fields with a data type of code, decimal and integer are created with a width of 1700 rather than a width of 1650, as is standard. Correct the field widths when creating a tabular form with the wizard.*

The table and forms have been created, but not complete. It is necessary to add some code, mostly for the purposes of data validation.

Lab 2.2 - Adding Code for Seminar Rooms

In this lab you will improve the functionality of the **City** and **Post Code** fields. In Microsoft Dynamics NAV, the standard table **Post Code** links cities and post codes. Therefore, write code so that when a user enters a **City**, the program fills in the corresponding **Post Code** value from the **Post Code** table, and vice versa. The existing table **Post Code** has functions that will help us in doing this.

1. In the **Seminar Room** table, create a global C/AL variable called **PostCode** for the record **Post Code**.
2. Validate the **City** field using the **ValidateCity** function from the **Post Code** table. Use the C/AL Symbol Menu to lookup the parameters for this function and insert the following code in the **City – OnValidate** trigger:

```
PostCode.ValidateCity(City,"Post Code");
```

3. Enter code so that when the user performs a lookup on the **City** field, the program runs the **Post Code** table's **LookUpCity** function. Use the C/AL Symbol Menu to see what parameters must be sent to the function when calling it. When calling the function, set the **ReturnValues** parameter to TRUE.

   – Solution: The **City – OnLookup** trigger code will be:

```
PostCode.LookUpCity(City,"Post Code",TRUE);
```

4. Use the **Post Code** table's **ValidatePostCode** function to validate the value entered by the user in the **Post Code** field.

   – Solution: The **Post Code – OnValidate** trigger code will be:

```
PostCode.ValidatePostCode(City,"Post Code");
```

5. Enter code so that when the user performs a lookup on the **Post Code** field, the program runs the **Post Code** table's **LookUpPostCode** function. When calling the function, set the **ReturnValues** parameter to TRUE.

   – Solution: The **Post Code – OnLookup** trigger code will be:

```
PostCode.LookUpPostCode(City,"Post Code",TRUE);
```

The next task is to improve the functionality of the **Resource No.** and **Contact No.** fields by retrieving information. In the following steps, you want to create code so that when a user enters a **Resource No.** or a **Contact No.**, if the **Name** field is empty, it will be filled with the **Name** from the corresponding **Resource** or **Contact** record.

6. Create two global record variables, one for the **Resource** table and one for the **Contact** table.
7. Enter code so that when validating the **Resource No.** entered by the user, if the **Name** field in the Seminar Room table is empty, the program looks up the **Name** field in the corresponding Resource record and assigns it to the **Name** field in the Seminar Room table.
   - Solution: Enter this code in the **Resource No. –OnValidate** trigger:

```
IF Resource.GET("Resource No.") AND (Name = '') THEN
  Name := Resource.Name;
```

8. Enter code so that when validating the **Contact No.** entered by the user, if the **Name** field in the **Seminar Room** table is empty, the program looks up the **Name** field in the corresponding Contact record and assigns it to the **Name** field in the Seminar Room table.
   - Solution: Enter this code in the **Contact No. – OnValidate** trigger:

```
IF Cont.GET("Contact No.") AND (Name = '') THEN
  Name := Cont.Name;
```

9. Create two global record variables, one for the **Comment Line** and one for the **Extended Text Header** tables.
10. Enter code in the appropriate table trigger so that when a record is deleted, any corresponding records in the **Comment Line** table are deleted as well.

HINT: *Use the DELETEALL function to delete the records.*

Solution: Enter this code in the **OnDelete** trigger:

```
CommentLine.SETRANGE("Table Name",CommentLine."Table
Name"::"Seminar Room");
CommentLine.SETRANGE("No.",Code);
CommentLine.DELETEALL;
```

11. Enter code in the appropriate table trigger so that when a record is deleted, any corresponding records in the **Extended Text Header** table are deleted as well.

HINT: *Use the DELETEALL function again, but this time, make sure that the parameter is set to TRUE so that the code in the delete trigger of the Extended Text Header fires. The reason you need to set the parameter to TRUE for Extended Text Header, where you did not for Comment Line, is that the OnDelete trigger code for Extended Text Header includes code to delete the associated Extended Text Line records.*

Solution: Add this code to the **OnDelete** trigger:

```
ExtTextHeader.SETRANGE("Table Name",ExtTextHeader."Table
Name"::"Seminar Room");
ExtTextHeader.SETRANGE("No.",Code);
ExtTextHeader.DELETEALL(TRUE);
```

Add code to the triggers in the forms to ensure that the form works properly.

12. In the **Seminar Room Card** form (123456703), enter code in the appropriate form trigger so that after the program has retrieved the record for the form, the program removes the filter on the **Code** field for the table.

Solution: Add this code to the **Form – OnAfterGetRecord** trigger:

```
SETRANGE(Code);
```

13. In the **Seminar Room Card** form, in the C/AL code for the **E-mail** command button, delete the existing code and enter code in the appropriate trigger so that when the user clicks the button, the program creates a new mail message.

HINT: *A local C/AL variable for the Mail codeunit already exists in this trigger. Use the OpenNewMessage function from this codeunit.*

- Solution: Add this code to the E-mail command button – **OnPush** trigger:

```
Mail.OpenNewMessage("E-Mail");
```

Instructors

Analysis

The functional requirements in Chapter 1 describe the instructor functionality this way:

Each seminar is taught by an instructor, who is a CRONUS employee. To make use of our existing resource information, each instructor must be set up as a Resource in Navision.

With this information, define how instructors will be managed in the module.

Design

According to the analysis of the management of instructor information, you need to create one table for managing the instructors.

Managing instructors is done the same way as managing the seminar rooms. Define them as resources to track their assignments, costs, and prices.

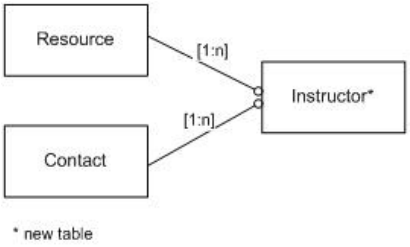

FIGURE 2-5: RELATIONSHIP BETWEEN INSTRUCTORS, RESOURCES, AND CONTACTS

Review the forms as they appear when complete, starting with the simplest form: The Instructors form:

The Instructors form (Figure 2-6) enables the entry of instructor information.

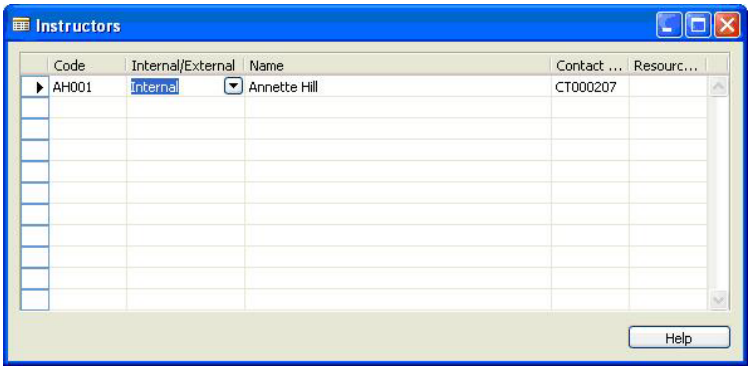

FIGURE 2-6: INSTRUCTORS FORM (123456705)

Implementation

Now you are ready to implement the instructor functionality.

Lab 2.3 - Creating Intructor Tables and Forms

Follow these steps to complete this lab:

1. Create the **Instructor** table (123456703) with the following fields:

No.	Field Name	Type	Length	Comment
1	Code	Code	10	Must not be blank.
2	Name	Text	30	
3	Internal/External	Option		Options: Internal, External
4	Resource No.	Code	20	Relation to table 156 Resource, where Type=Person.
5	Contact No.	Code	20	Relation to the **Contact** table (5050)

The primary key is the Code field. Set the property to specify the Instructors form (123456705) as the lookup form for this table.

NOTE: *Specify the DataCaptionFields for the table and a Caption for each field in the table.*

2. In the **Instructor** table, enter code so that when the program validates the value in the **Resource No.** field, if the **Name** field in the **Instructor** record is empty, the program fills it with the **Name** field from the corresponding **Resource** record. Do the same for the **Contact No.** field.

HINT: *You wrote very similar code in the previous lab.*

3. Create the **Instructors** form (123456705) as shown in the GUI design.

Seminars

Analysis

The functional requirements in Chapter 1 describe the seminar functionality this way:

The CRONUS training department holds several different seminars. Each seminar has a fixed duration, and a minimum and maximum number of participants. In some cases seminars can be overbooked, depending on the capacity of the assigned room. Each seminar can be cancelled if there are not enough participants. The price of each seminar is fixed. You want to take advantage of the current Job functionality in Microsoft Dynamics NAV and define each seminar as a job. When a seminar is completed, the seminar should be posted as a job, with additional seminar-specific information.

From this description it is possible to create a use case:

When defining a new seminar, seminar managers describe the course details, including information such as the seminar name, duration, price, maximum and minimum number of participants, and a job code.

As in the analysis of the seminar management process, you need two tables to manage seminar information. The first table is simply a setup table where the user can define seminar numbering. There is a table to track the seminar profile information.

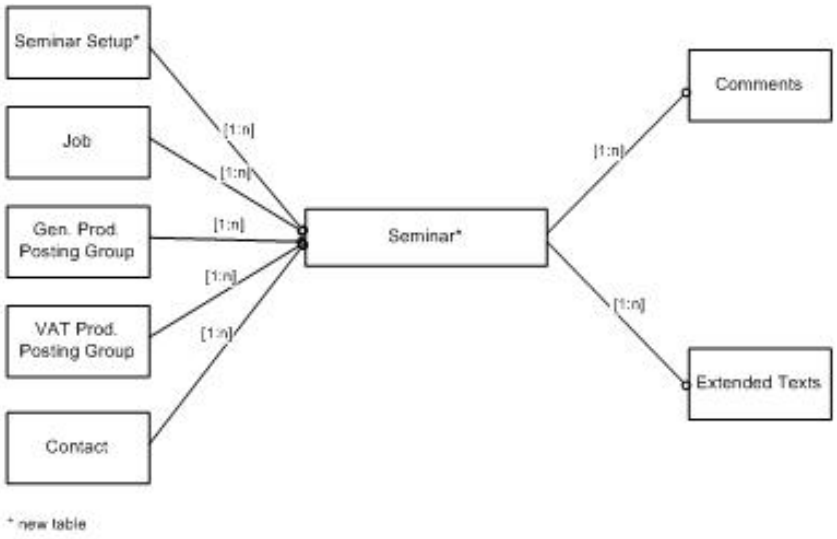

FIGURE 2-7: RELATIONSHIP BETWEEN SEMINARS AND OTHER ENTITIES

This description provides the information necessary to design and implement the seminar functionality.

Design

With the design complete, Review at how the forms appear when implementation is complete.

The **Seminar Setup** form (Figure 2-8) for allows the entry of setup information for the seminar.

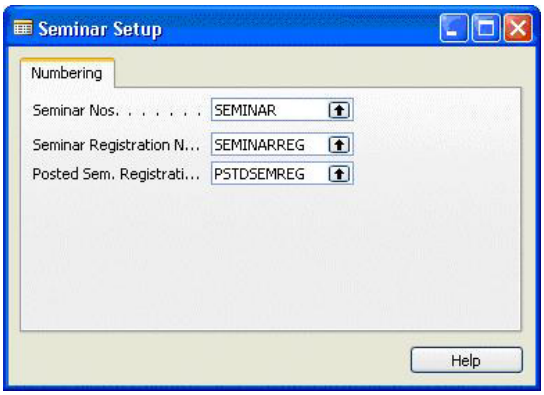

FIGURE 2-8: SEMINAR SETUP (FORM 123456702)

The **Seminar Card** form, shown in Figure 2-9, allows the entry of seminar details.

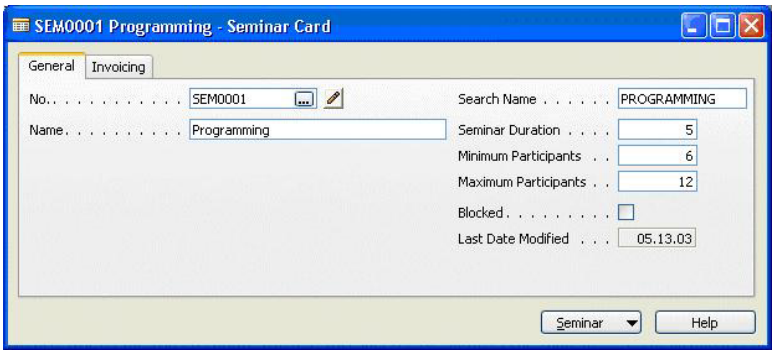

FIGURE 2-9: SEMINAR CARD (FORM 123456700): GENERAL TAB

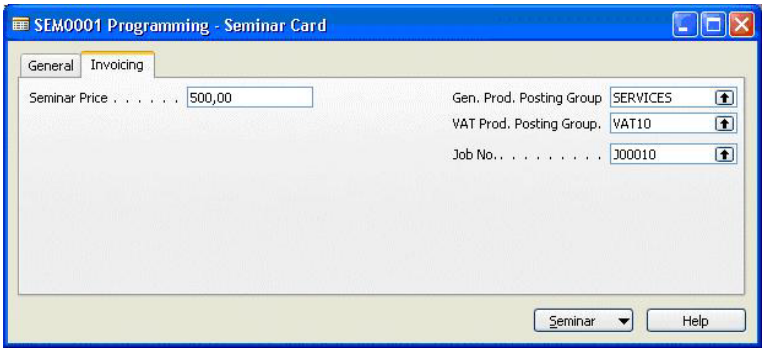

FIGURE 2-10: SEMINAR CARD (FORM 123456700): INVOICING TAB

The **Seminar List** form (Figure 2-11) provides an overview of the seminars, but does not allow the underlying Seminar table data to be modified.

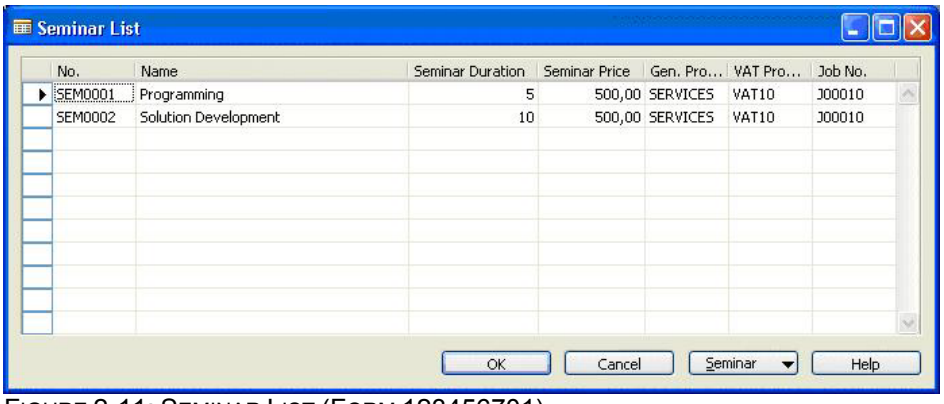

FIGURE 2-11: SEMINAR LIST (FORM 123456701)

Lab 2.4 - Creating Seminar Tables and Forms

To create the seminar master files and forms, do the following:

1. In the Comment Line table (97), add the option **Seminar** to the options for the field **Table Name**.
2. In the **Extended Text Header** table 279 and the Extended Text Line table (280), add the option **Seminar** to the options for the field **Table Name**.
3. **Create Table 123456701 Seminar Setup** with the following fields:

No.	Field Name	Type	Length	Comment
1	Primary Key	Code	10	
2	Seminar Nos.	Code	10	Relation to 308 No. Series table.
3	Seminar Registration Nos.	Code	10	Relation to 308 No. Series table.
4	Posted Sem. Registration Nos.	Code	10	Relation to 308 No. Series table.

   – Per Microsoft Dynamics NAV standards for setup tables, the key for this table is the **Primary Key** field, which is left blank.

4. Create the **Seminar** table (123456700) with the following fields:

No.	Field Name	Type	Length	Comment
1	No.	Code	20	An alternate search field is the **Search Name** field.
2	Name	Text	50	
3	Seminar Duration	Decimal		Decimal places 0:1
4	Minimum Participants	Integer		
5	Maximum Participants	Integer		
6	Search Name	Code	30	

No.	Field Name	Type	Length	Comment
7	Blocked	Boolean		
8	Last Date Modified	Date		Must not be editable.
9	Comment	Boolean		FlowField; CalcFormula checks whether lines exist in the Comment Line table for the seminar. Must not be editable.
10	Job No.	Code	20	Relation to table 167 Job.
11	Seminar Price	Decimal		AutoFormat type is 1.
12	Gen. Prod. Posting Group	Code	10	Relation to table 251 Gen. Product Posting Group.
13	VAT Prod. Posting Group	Code	10	Relation to table 324 VAT Product Posting Group.
14	No. Series	Code	10	Must not be editable. Relation to table 308 No. Series.

   - The primary key for this table is the **No.** field with a secondary key of **Search Name**. Set the properties to specify form 123456701 as the lookup and drill-down form for this table.

5. In the Seminar table, enter code to perform the following validation tasks:

   - When the user changes the **No.** value from what it was previously, the program gets the Seminar Setup record and uses the TestManual function of the NoSeriesManagement codeunit to test whether the number series is allowed to be manually changed. The program then sets the **No. Series** field to blank.

HINT: *Use the xRec variable to test whether the* ***No.*** *field has been modified. You need a record variable with a subtype of Seminar Setup and a Codeunit variable of subtype NoSeriesManagement to perform this task. Remember that there is only one record in the Seminar Setup table and the primary key field is left null.*

Solution:

```
IF "No." <> xRec."No." THEN BEGIN
  SeminarSetup.GET;
  NoSeriesMgt.TestManual(SeminarSetup."Seminar Nos.");
  "No. Series" := '';
END;
```

– When the user enters or changes a value in the **Name** field, if the **Search Name** is still equal to the uppercase value of the previous Name or if the Search Name is blank, the program assigns the new Name to Search Name.

HINT: *Use the function UPPERCASE when testing the* ***Search Name*** *field.*

Solution:

```
IF ("Search Name" = UPPERCASE(xRec.Name)) OR
("Search Name" = '') THEN
  "Search Name" := Name;
```

– When the user enters or changes a value in the **Job No.** field, the program retrieves the corresponding **Job** record and checks that the **Blocked** field is set to FALSE.

HINT: *Use the TESTFIELD function to check a field's value.*

Solution:

```
Job.GET("Job No.");
Job.TESTFIELD(Blocked,FALSE);
```

– When the user changes the **Gen. Prod. Posting Group** to a new value from what it was previously, the program checks that the function **ValidateVatProdPostingGroup** for the **Gen. Product Posting Group** table returns true. If it does, the program sets the **VAT Prod. Posting Group** to the **Def. VAT Prod. Posting Group** value from the **Gen. Product Posting Group** table.

Solution:

```
IF xRec."Gen. Prod. Posting Group" <> "Gen. Prod. Posting
Group" THEN
IF
GenProdPostingGrp.ValidateVatProdPostingGroup(GenProdPostin
gGrp,"Gen. Prod. Posting Group") THEN
VALIDATE("VAT Prod. Posting Group",GenProdPostingGrp."Def.
VAT Prod. Posting Group");
```

6. In the **Seminar** table, create a new function named AssistEdit with a return type of Boolean. In this function, enter code that checks for a Seminar Nos. series in the Seminar Setup table. If it finds one, the program uses the SelectSeries function in the NoSeriesManagement codeunit to check the series number. If this function returns true, the program uses the SetSeries function in the NoSeriesManagement codeunit to set the **No.** field, and the program then exits TRUE.

Solution:

```
WITH Seminar DO BEGIN
  Seminar := Rec;
  SeminarSetup.GET;
  SeminarSetup.TESTFIELD("Seminar Nos.");
  IF NoSeriesMgt.SelectSeries(SeminarSetup."Seminar
Nos.",xRec."No. Series","No. Series") THEN BEGIN
    SeminarSetup.GET;
    SeminarSetup.TESTFIELD("Seminar Nos.");
    NoSeriesMgt.SetSeries("No.");
    Rec := Seminar;
    EXIT(TRUE);
  END;
END;
```

7. In the **Seminar** table, enter code in the appropriate table triggers to perform the following tasks:
   - Document the code.
   - When a record is inserted, if the **No.** field is blank, the program gets the **Seminar Setup** record and run the **InitSeries** function of the **NoSeriesManagement** codeunit to initialize the series.

Solution: Enter the following code in the OnInsert trigger:

```
IF "No." = '' THEN BEGIN
  SeminarSetup.GET;
  SeminarSetup.TESTFIELD("Seminar Nos.");
  NoSeriesMgt.InitSeries(SeminarSetup."Seminar
Nos.",xRec."No. Series",0D,"No.","No. Series");
END;
```

– When a record is modified or renamed, the program sets the **Last Date Modified** to the current date.

HINT: *Use the TODAY function to get the current date.*

Solution: Enter the following code in the OnModify and OnRename triggers:

```
"Last Date Modified" := TODAY;
```

– When a record is deleted, the program deletes the corresponding records from the **Comment Line** table and the **Extended Text Header** table.

HINT: *You created similar code for the Seminar Room comment and extended text records.*

8. Create the Seminar Setup form (123456702) as shown in the GUI design.
9. Enter code in the appropriate trigger of the Seminar Setup form so that when the user opens the form, the program resets the record, and if it does not get a record, it inserts a new one.

Solution: Enter the following code in the Form – OnOpenForm trigger:

```
RESET;
IF NOT GET THEN
  INSERT;
```

10. Create form 123456700 Seminar Card as shown in the GUI design.
    – Set the property to update the form when it is activated.
    – Add a menu button and menu items as follows:

Menu Button	Options	Comment
Seminar	List (f5)	Opens the lookup form.
	Comments	Opens the **Comment Sheet** form (124) for the selected entry.
	<Separator>	
	Extended Texts	Opens the form 386 Extended Texts for the selected entry.

– Add a command button next to the **No.** field to provide access to the Comment Sheet for the corresponding record.

- Set the **RunFormLink** property for this button so that only the comments corresponding to the selected Seminar are displayed.
- Enter code in the appropriate trigger so that when the user clicks the AssistEdit for the **No.** field, the program runs the **AssistEdit** function, and if it returns true, updates the current form.

Solution: Enter the following code in the **No. – OnAssistEdit** trigger:

```
IF AssistEdit THEN
  CurrForm.UPDATE;
```

- Enter code in the appropriate trigger so that after the form gets the record, the program removes the table's filter on the **No.** field.

Solution: Enter the following code in the Form - OnAfterGetRecord trigger:

```
SETRANGE("No.");
```

11. Create the Seminar List form (123456701) according to the GUI Design section. Include the fields No., Name, Seminar Duration, Minimum Participants (not visible), Maximum Participants (not visible), Seminar Price, Gen. Prod. Posting Group, VAT Prod. Posting Group and Job No.
    - Make the form not editable.
    - Glue the **Name** field to both sides of the form.
    - Remember to reset the width of fields with data type Code, Decimal, and Integer to 1650.
    - The navigation from this form is be the same as the Seminar Card form, except that instead of a List option, there will be an option called Card (SHIFT + F5) from which the Seminar Card form opens for the selected seminar.

Testing

With the master tables and forms in place, it is critical that the functionality is tested. To this end, a test script can be used. However, since not all solution pieces have been coded, the test script may seem incomplete.

NOTE: *To test the Email functionality, the system you are using needs to have Microsoft® Outlook® running with a profile set up.*

1. Start by selecting form 123456702 Seminar Setup in the Object Designer and clicking Run. The fields in this form are blank as they have not yet been set up. Click the lookup button for Seminar Nos. to open the No. Series form. You have setup seminars to use the standard Microsoft ® numbering functionality. Set up all three numbers with values of your choosing, set to Default automatically and close and re-open the Seminar Setup form. See that the values were saved.
2. Select form 123456700 Seminar Card in the Object Designer and click Run. Tab off the No. field and see that a number is assigned with the values you set up in step 1. Fill in the fields on the General tab and then move to the Invoicing tab. Use the lookup to select a Gen. Prod. Posting Group value with a Default VAT Prod. Posting Group set and see the VAT Prod. Posting Group fill in automatically. Click F5 to move to the Seminar List form. Close the Seminar List form.
3. On the Seminar Card form, use the menu items on the Seminar Command button to enter comments and extended texts. When you have finished close the Seminar Card form.
4. You have designed the Seminar module so that Instructors are set up as Resources and Contacts in the system. Set up a new Resource (type Person) and Contact to use to test the Instructor functionality. When done, select form 123456705 Instructors in the Object Designer and click **Run**. Enter a new instructor using the **Resource** and **Contact** lookup buttons to select the Resource and Contact you set. Note that when using the **Resource** lookup, all the Resources in the list should be of type Person. Try entering a **Resource No**. or **Contact No**. that does not exist to test your validation. When you have finished close the Instructors form.

5. You have designed the Seminar module so the Rooms are set up as Resources of type Machine and Contacts in your system. Set up a new **Resource** and **Contact** that you use to test the Seminar Room functionality. When finished, select form 123456703 Seminar Room Card in the Object Designer and click **Run**. Create a new Seminar Room by entering a value in the **Code** field. You added code so that the **Name** field would default from the **Resource** or **Contact** if the **Name** field was blank. See if that code is functioning. On the **Communications** tab, try entering an E-Mail address and Home Page and use the command buttons next to them. Check the menu items under the Seminar Room command button.

Conclusion

In this chapter, two of the most fundamental object types, tables and forms, are used to put the foundation for the solution in place. The tables define and store the data for the solution, and the forms provide an intuitive interface with which the user can interact with the table data.

Defining the tables requires analysis of the relationships between the entities represented by each table. Implementation of the tables is easier if the simpler tables – those with fewer dependencies – are defined first.

Test Your Knowledge – Master Tables and Forms

Review Questions

1. Where is internal documentation located for new objects?

2. To ensure that Microsoft Dynamics NAV's multilanguage functionality is properly enabled, which property must be set for controls?

3. Suppose that in the Seminar table, you want to make sure that the value in the **Minimum Participants** field is always less than the value in the **Maximum Participants** field. You want to perform this check whenever a record is inserted, whenever a record is changed and whenever a value is entered in the **Maximum Participants** field (and the **Minimum Participants** field is not empty). Which table and field event triggers would you use for these three checks? What code would you use to perform these checks?

4. What do the **Rec** and **xRec** record variables do?

5. What function would you use to retrieve a specific record using its primary key?

6. Using the table example Employee below, answer the following questions:
   - Assuming you are working with the record variable EmployeeVar, what code would you write to get the record for Employee No. 3475? What code would you write to jump to the fourth next record?
   - What code would you write to jump to the last record in the table?
   - What code would you use to define a record set to include the records between employee number 3475 and 6434?
   - What code would you use to define a record set to include the records greater than 5834?
   - What code would you use to define a record set for employees in the Purchasing department only? What code would you use to remove the filter on department?
   - What code would you use to change Ann Smith's department from Sales to Purchasing?
   - What code would you use to delete all Receivables employees?

Table: Employee (key = Employee No.)

Employee No.	Name	Department
5834	John Doe	Purchasing
3723	Ann Smith	Sales
3475	Bill Skaggs	Receivables
6434	Todd Lawrence	Sales
9482	Janet Davila	Receivables
0980	Susan Morris	Purchasing
9483	Rick Hamilton	Sales

7. What is the standard shortcut to open a Card form from the associated List form? What about to open the List form from the Card form?

8. What two triggers does a Codeunit have by default?

9. Name two ways to set the lookup form for a table field. What happens if both are set?

Quick Interaction: Lessons Learned

Take a moment to note three key points you have learned from this chapter:

1.

2.

3.