EXCEL APPLICATIONS

10 Steps for Database Developers

Sergey Vaselenko

EXCEL APPLICATIONS

10 Steps for Database Developers

Written by Sergey Vaselenko

This e-book shows how to create feature-rich database client applications using Microsoft Excel.
As a database developer, you may create such applications with the SaveToDB add-in using SQL knowledge only.

Copyright © 2017 Sergey Vaselenko

Introduction

The SaveToDB add-in allows creating database client applications using Microsoft Excel.

Moreover, application features are being configured in a database and extended using SQL.

So, using the add-in is the best choice to create client applications for database developers.

As a database developer, you may apply your current skills to create amazing Excel applications.

I believe we help developers realize their potentials, build a successful career and make users happy.

Here are the basic steps to create a complete client application using Microsoft Excel:

1. Connect to tables, views, and stored procedures
2. Configure validation lists
3. Configure ribbon parameters
4. Translate field, parameter, and object names
5. Configure formats, formulas, and table views
6. Configure saving changes
7. Add cursors and form fields
8. Create master-detail forms
9. Configure detail windows and task panes
10. Configure context and action menus

In this book, we start with a new Excel workbook and finish with a ready-to-use application.

You have to download and install the SaveToDB add-in, version 7.2 or higher, at www.savetodb.com.

All features described in this book are available in the free SaveToDB Express edition.

You may download example workbooks and SQL codes at

https://www.savetodb.com/download.php?file=10-steps-for-database-developers.zip

This book contains an example database for Microsoft SQL Server hosted in Microsoft SQL Azure Database.

You may also use Oracle Database, IBM DB2, MySQL, PostgreSQL, and others. The steps remain the same.

Best regards,

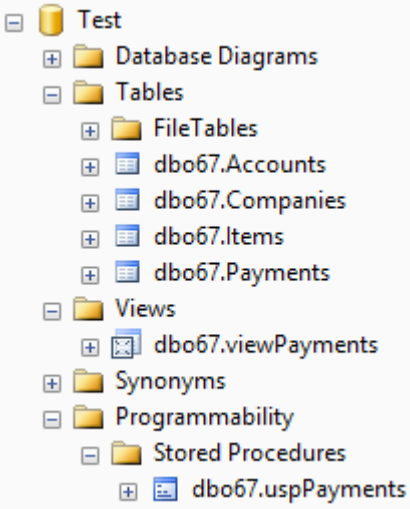
Sergey Vaselenko

March 20, 2017

Table of Contents

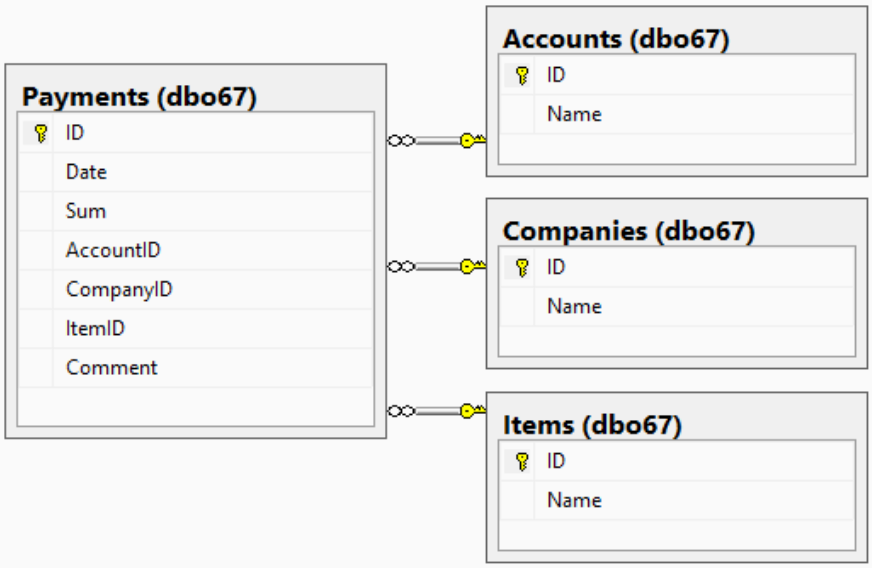
Chapter 1. Example Database

We will create an Excel application for a simple database that contains several tables, a view, and a procedure:



The database objects belong to the **dbo67** schema.

The tables have the following relations:



We have three master tables and the **dbo67.Payments** table with foreign keys.

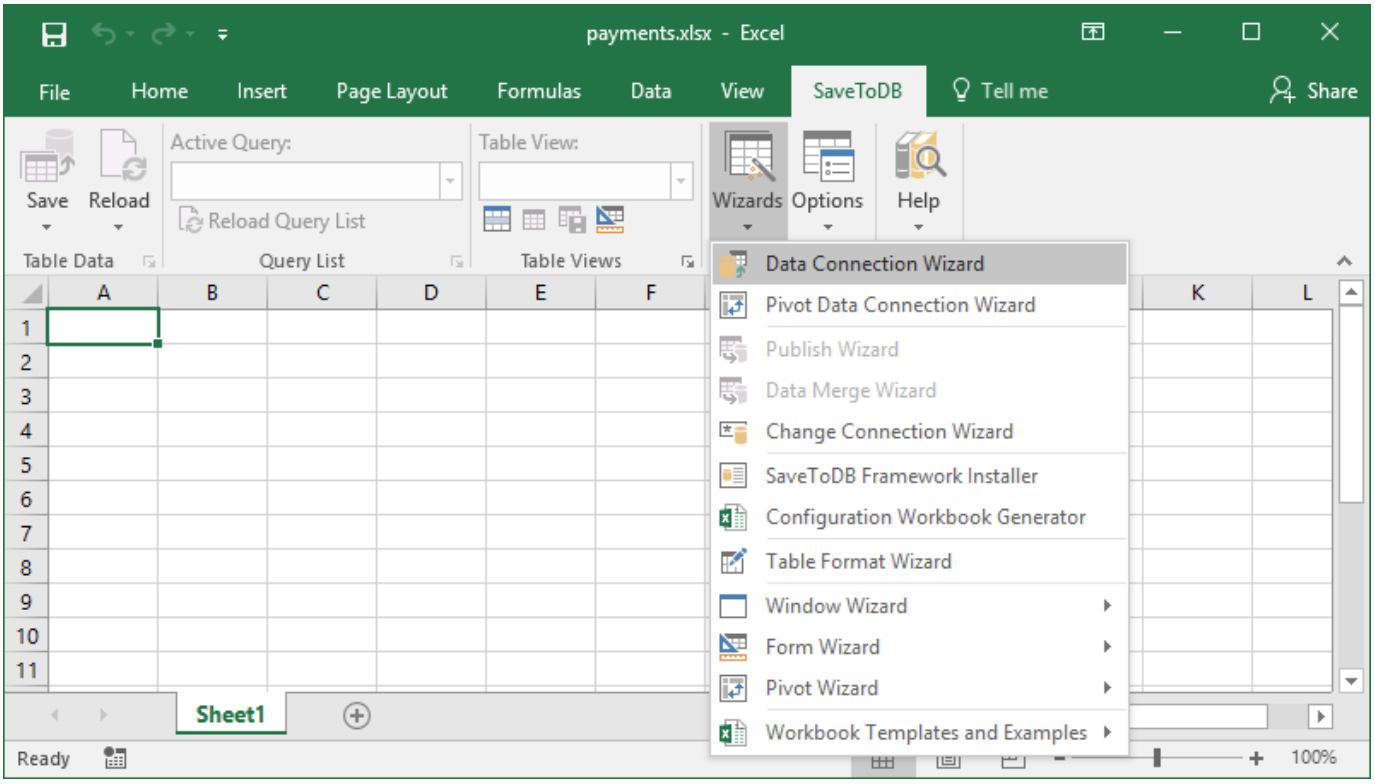
The **dbo67.viewPayments** view and **dbo67.uspPayments** stored procedure select data from the Payments table.

You may find a complete source code with comments and download links in Appendix 1.

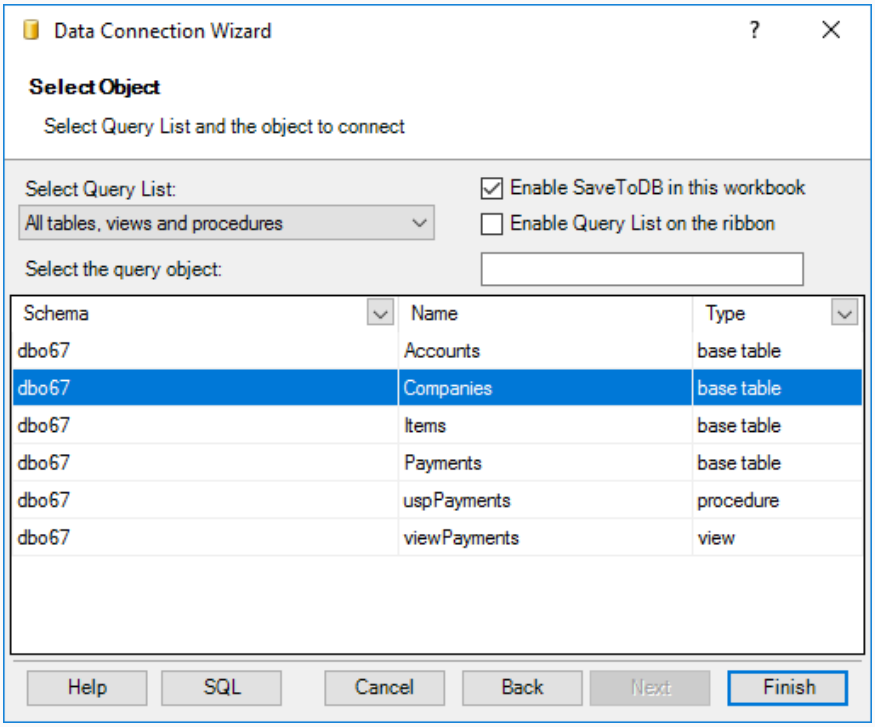
Chapter 2. Excel as Table Editor

Let's create a new workbook and save it as **payments.xlsx**.

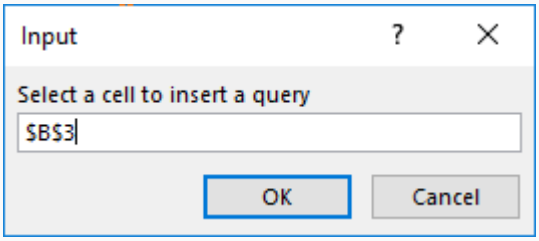
Then let's run the **Data Connect Wizard** and connect to the **Companies** table.



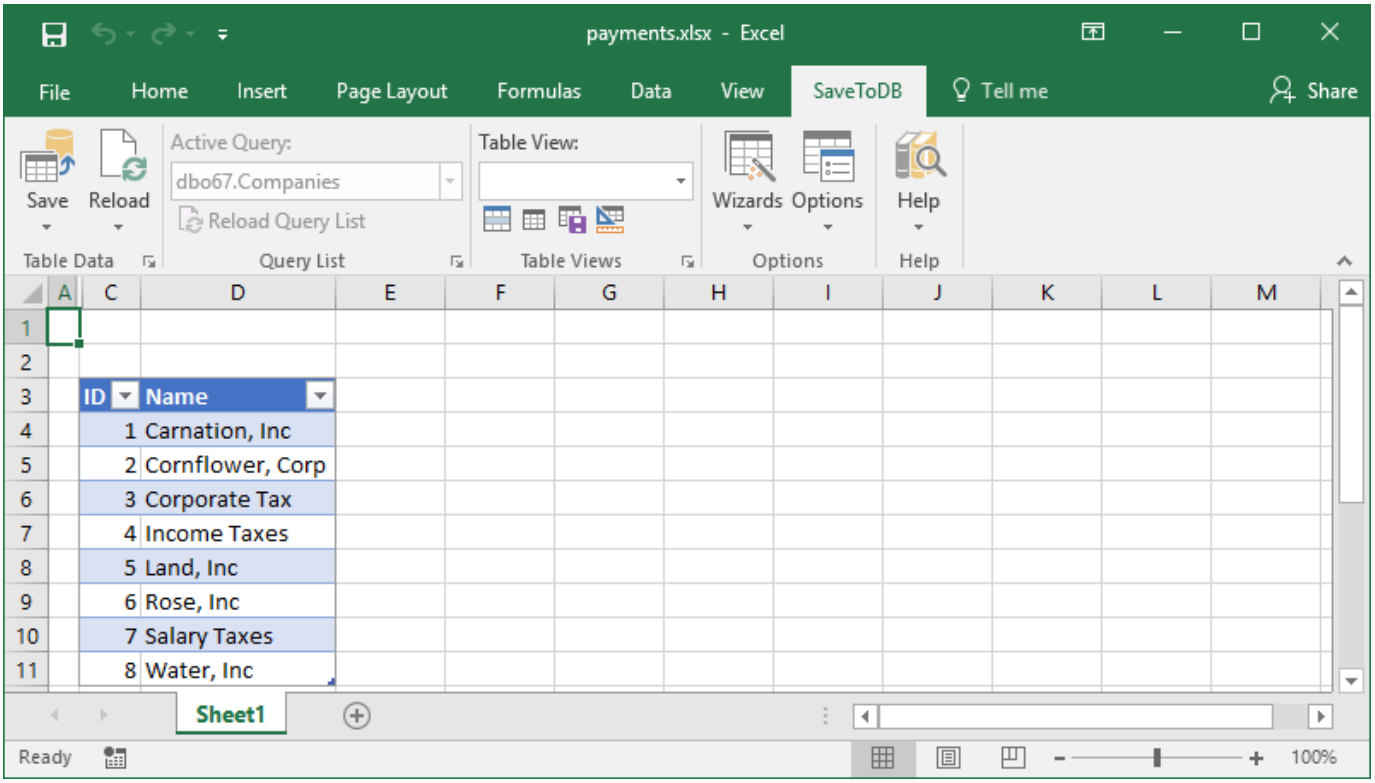
Follow wizard steps. At the following screen, uncheck **Enable Query List on the ribbon**.



Insert a table at cell B3.



We have the following result:



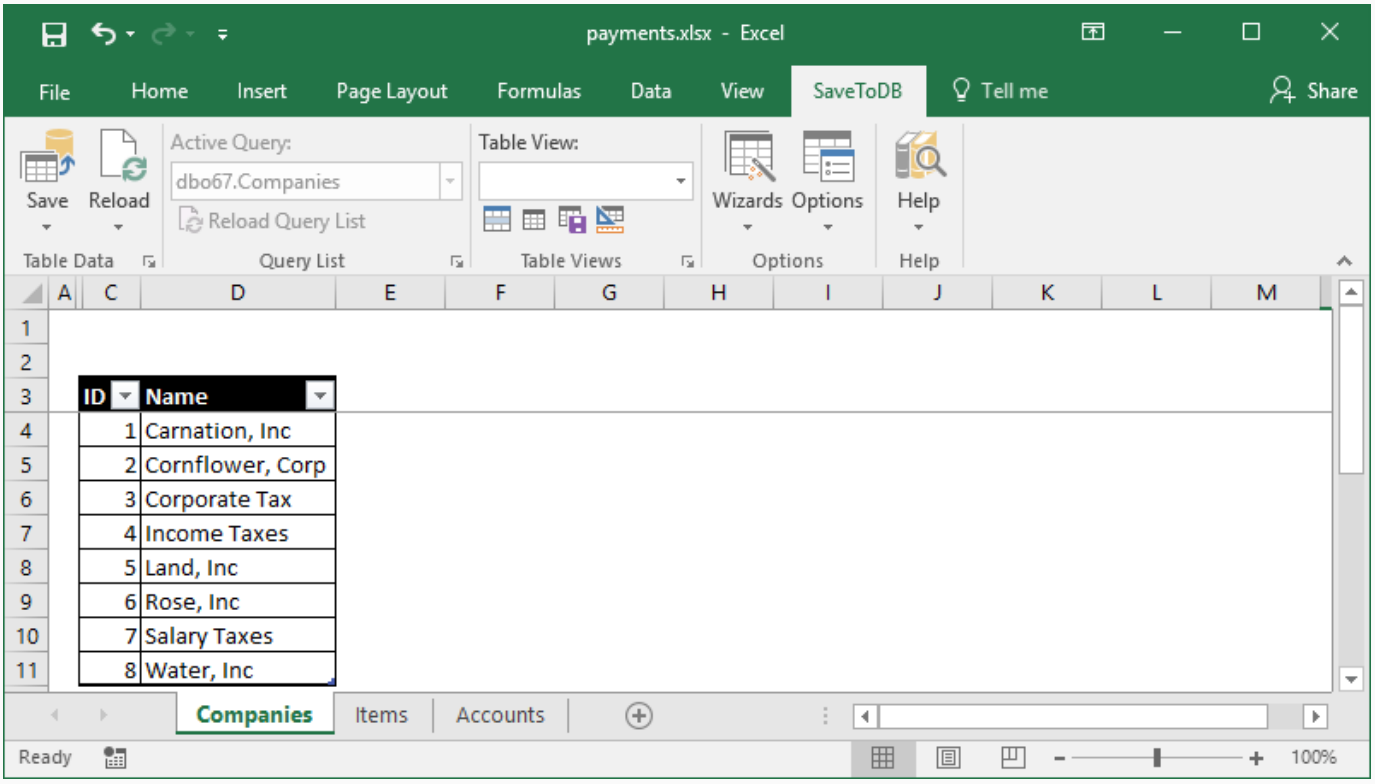
We may edit data, add and delete rows. Then just click the **Save** button to save changes.

Let’s customize the design:

1. Click on a table. Select the **Design** tab. Select the desired design in the **Table Styles** gallery. I prefer White, Table Style Medium 15.
2. Right click on the selected design and click **Set as Default**.
3. Uncheck **Bunded Rows** in the **Table Style Options** group.
4. Select the **View** tab. Uncheck **Gridlines** in the **Show** group.
5. Select cell A4 and click **Freeze Panes** in the **Window** group.
6. Rename the worksheet to Companies.

Repeat the steps for the **Items** and **Accounts** tables.

Now we have the workbook that allows editing master tables.

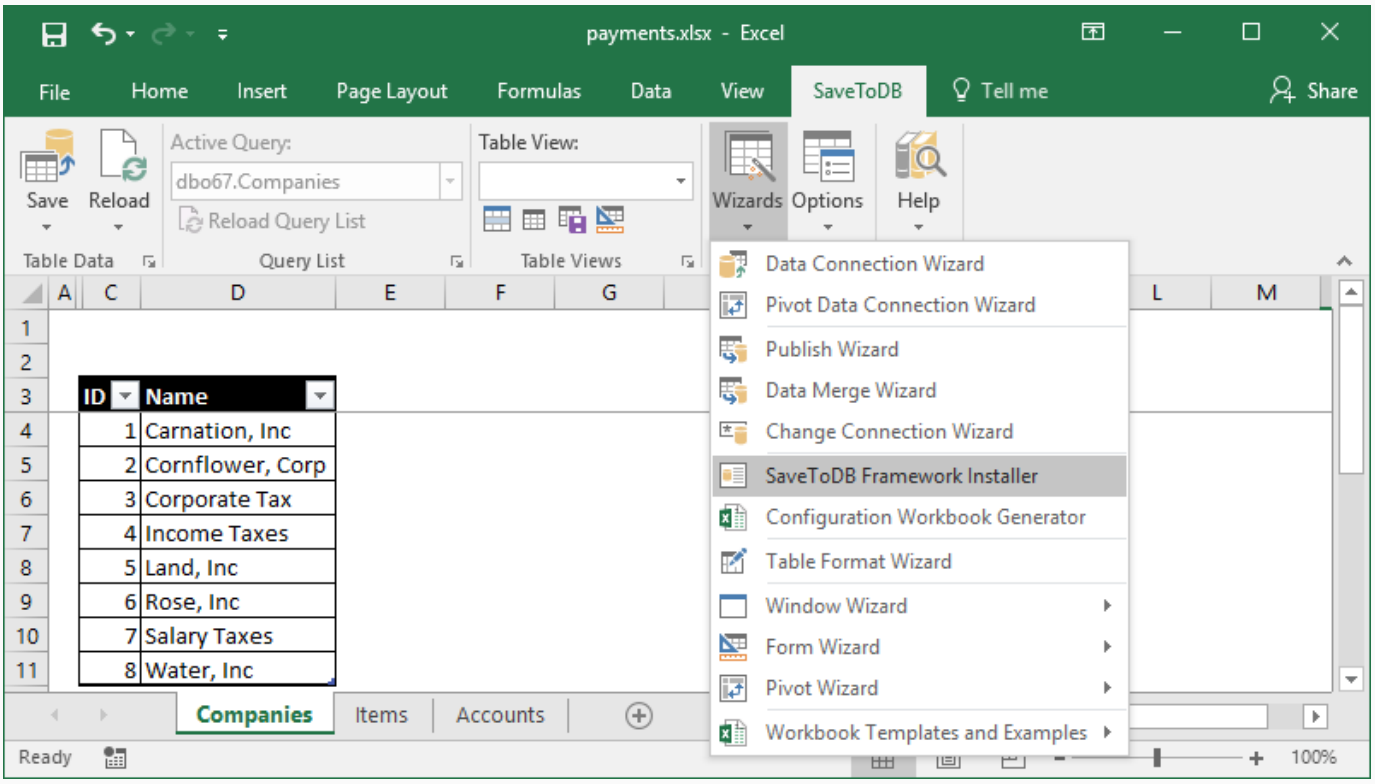


Chapter 3. SaveToDB Framework Installer

You may use Microsoft Excel as a table editor by default.

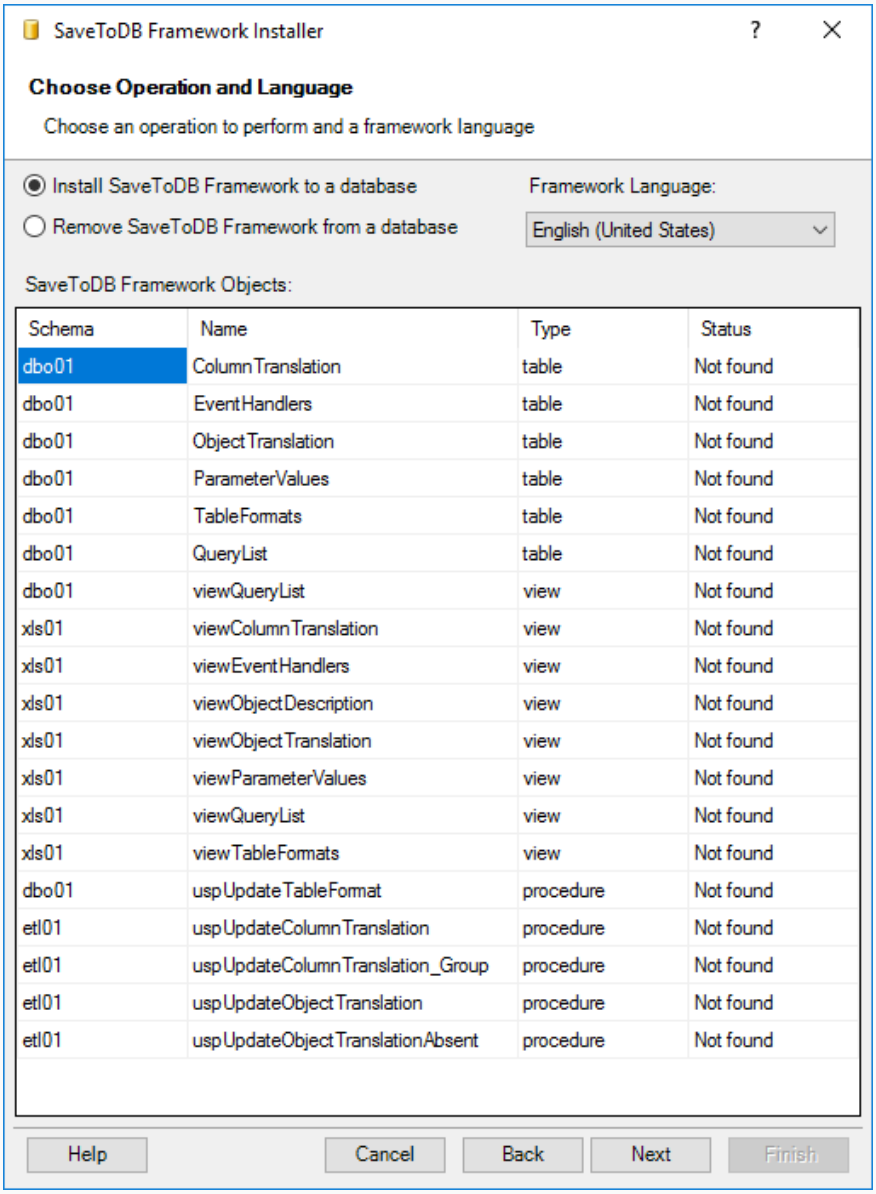
To use advanced features, we have to install the SaveToDB Framework to a database.

Run the **SaveToDB Framework Installer**:



Follow wizard steps.

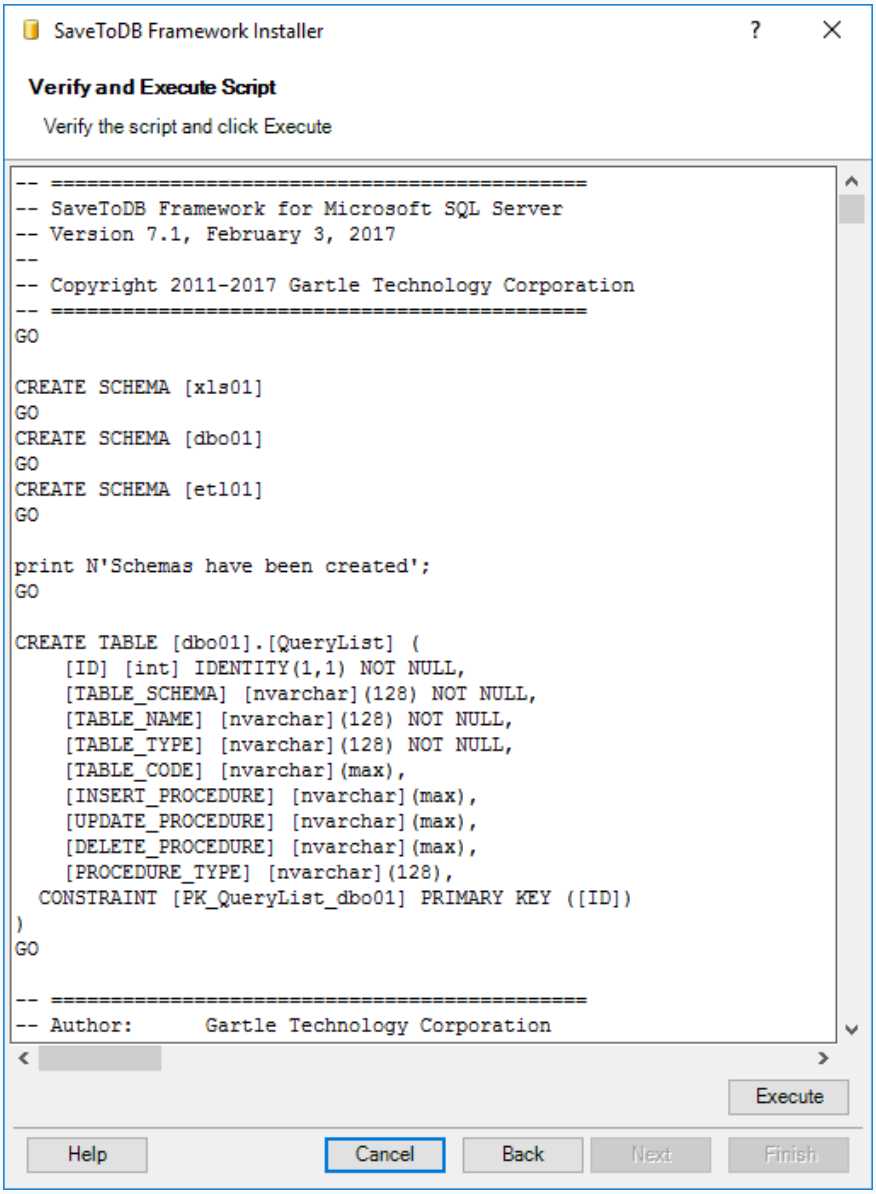
At this screen, you see SaveToDB Framework objects:



These objects allow configuring SaveToDB add-in behavior.

You may remove framework objects using the same wizard. So, this is an entirely safe operation.

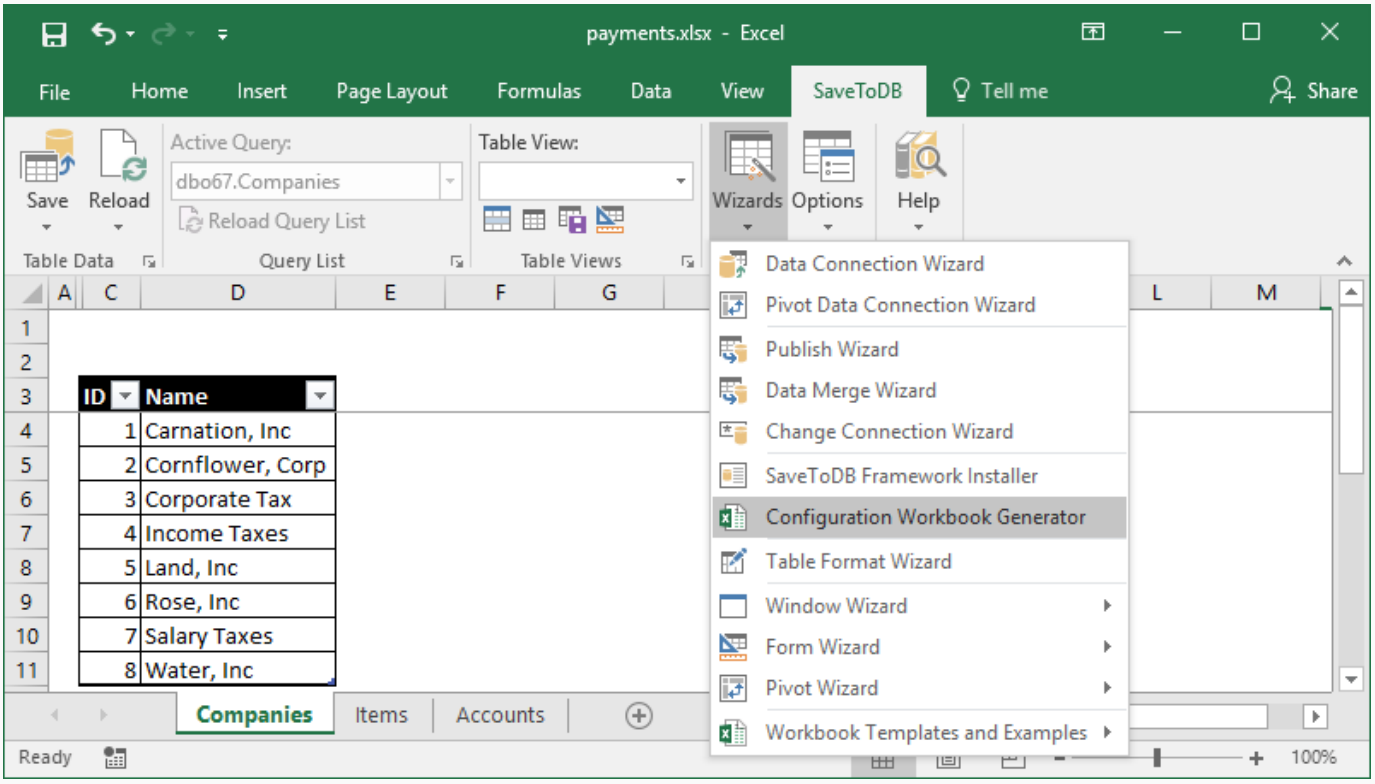
At this step, you see the SaveToDB Framework code. Click **Execute** to install it.



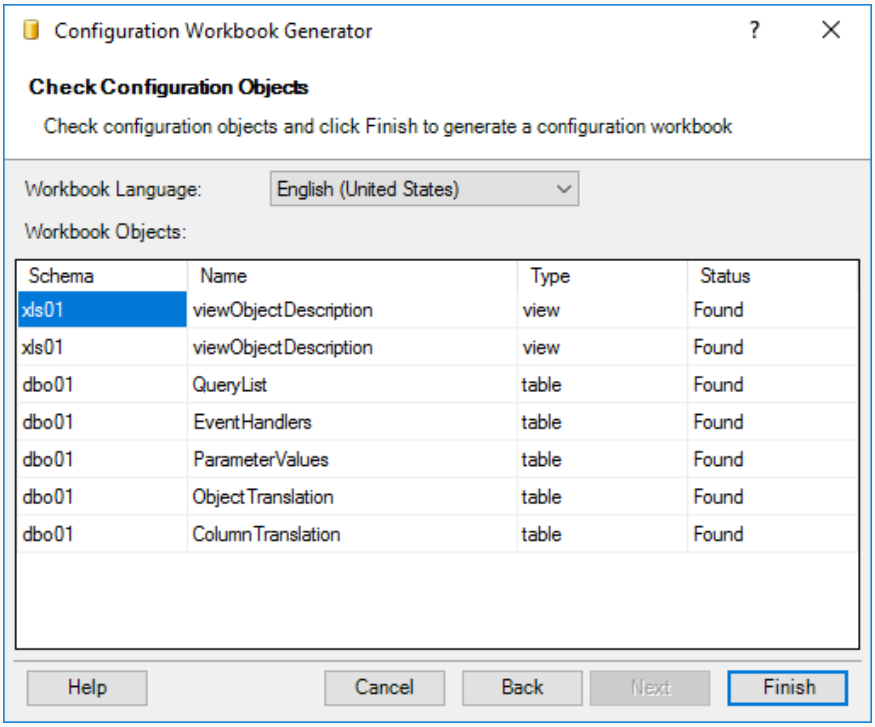
Chapter 4. Configuration Workbook

We have added SaveToDB Framework to a database.

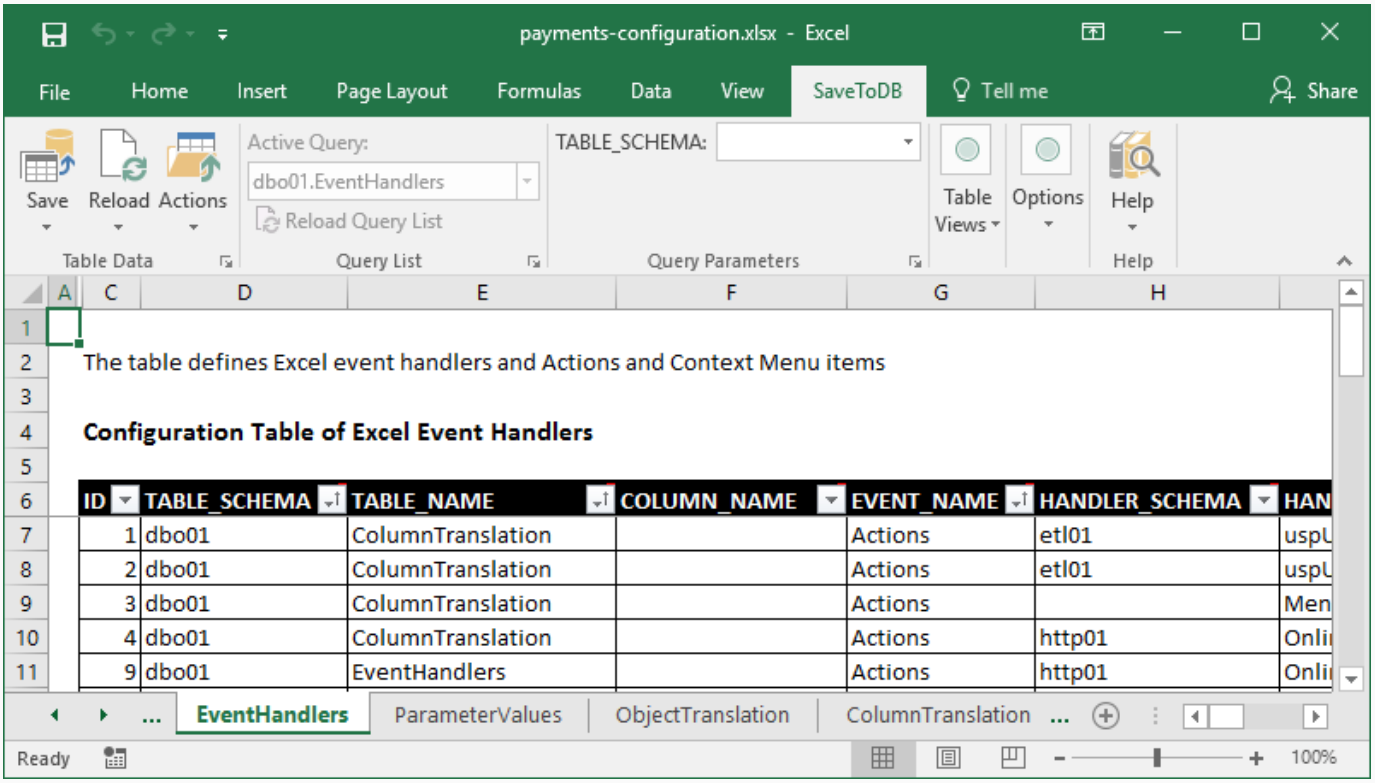
Now we may use Microsoft Excel to edit framework tables. Run **Configuration Workbook Generator**:



At this screen, you see SaveToDB Framework configuration tables:

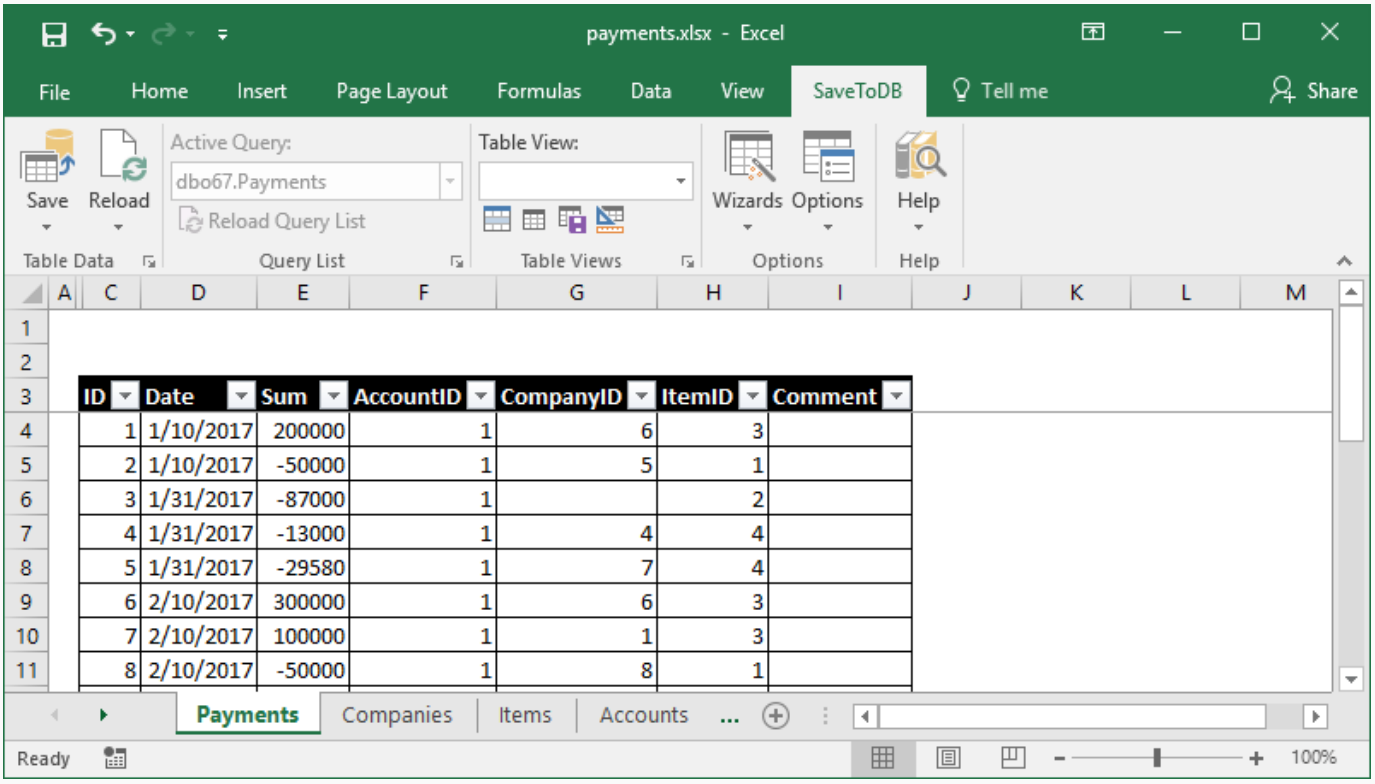


When you click **Finish**, the SaveToDB add-in generates a workbook.



You may generate it again at any time. However, let's save it as **payments-configuration.xlsx** for further use.

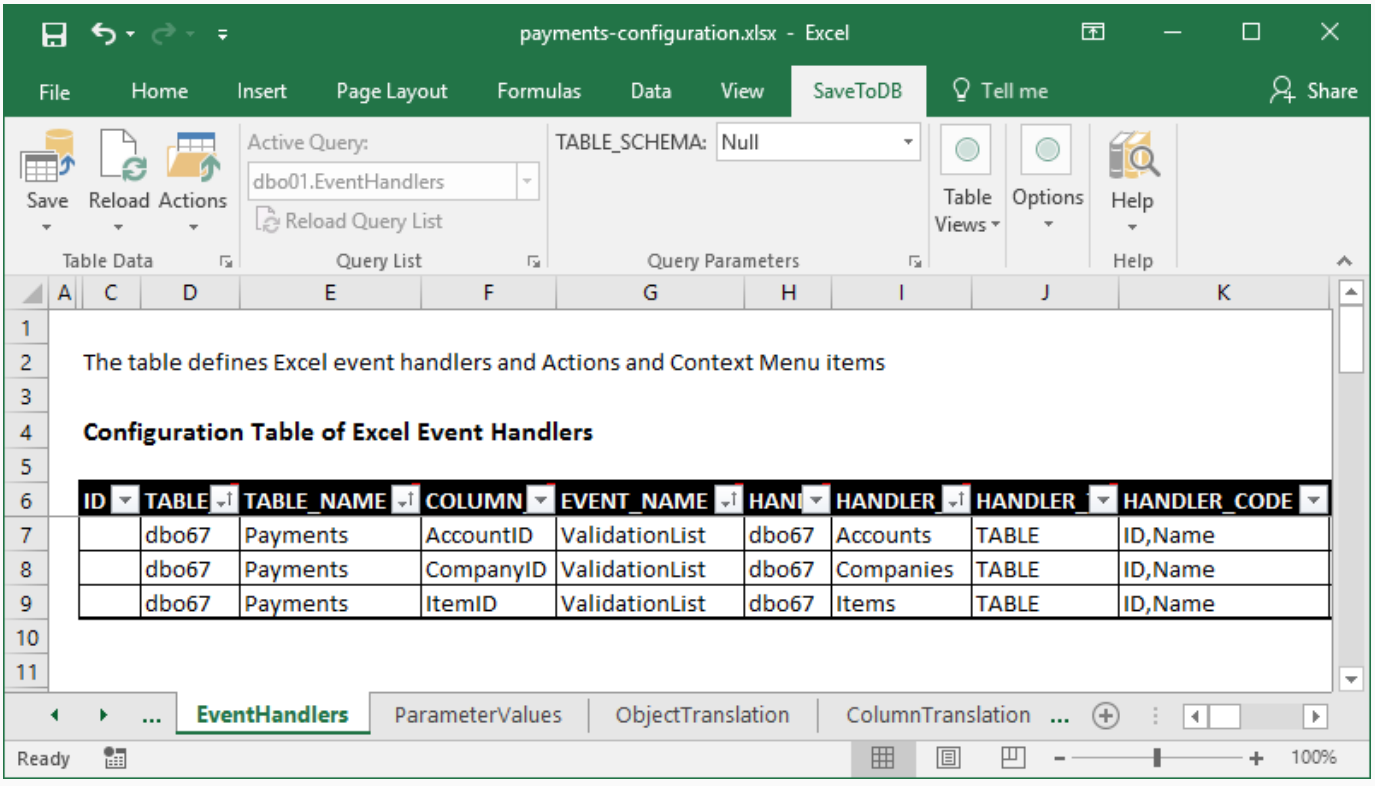
Let's add a worksheet, rename it to **Payments**, and connect to the **Payments** table.



ID	Date	Sum	AccountID	CompanyID	ItemID	Comment
1	1/10/2017	200000	1	6	3	
2	1/10/2017	-50000	1	5	1	
3	1/31/2017	-87000	1		2	
4	1/31/2017	-13000	1	4	4	
5	1/31/2017	-29580	1	7	4	
6	2/10/2017	300000	1	6	3	
7	2/10/2017	100000	1	1	3	
8	2/10/2017	-50000	1	8	1	

As expected, we see foreign key values. Let's change this.

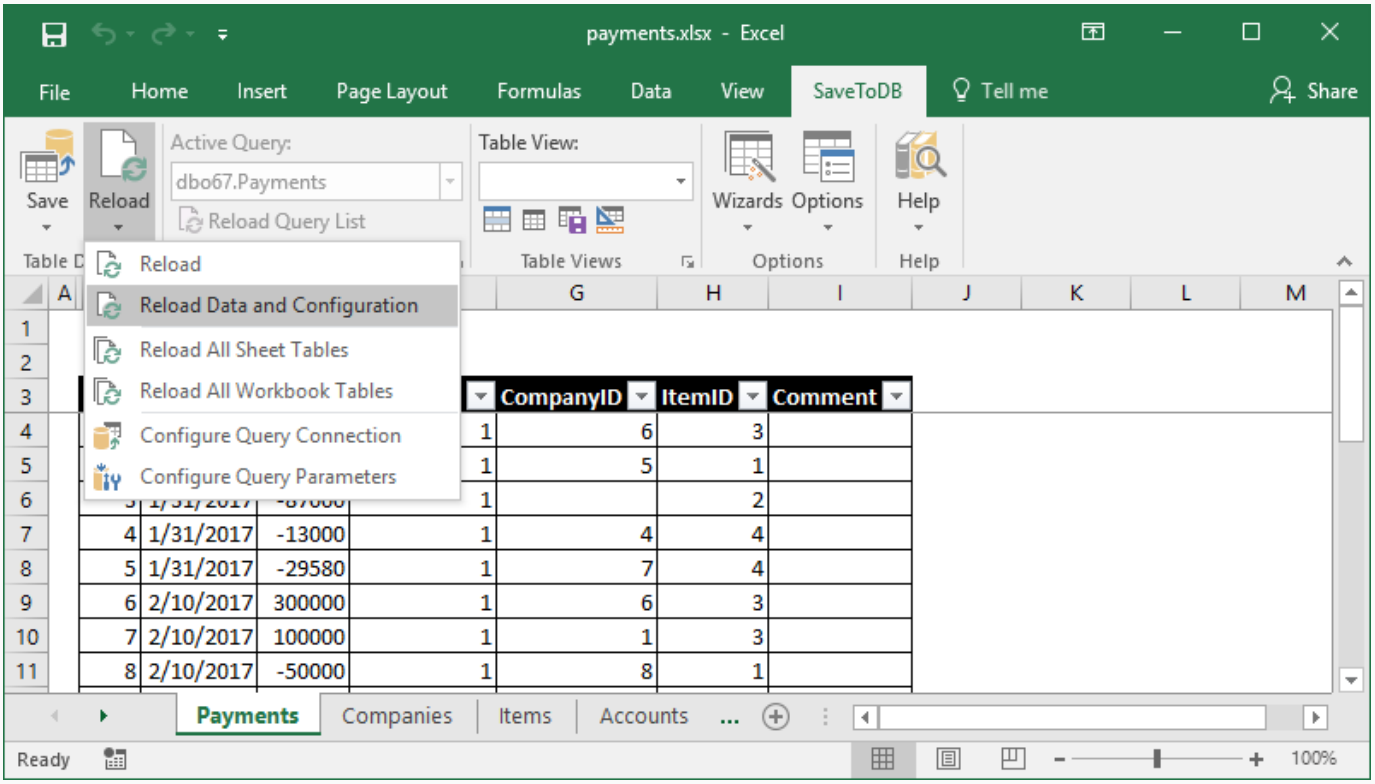
Switch to the **payments-configuration** workbook, select the **EventHandlers** worksheet and add the configuration:



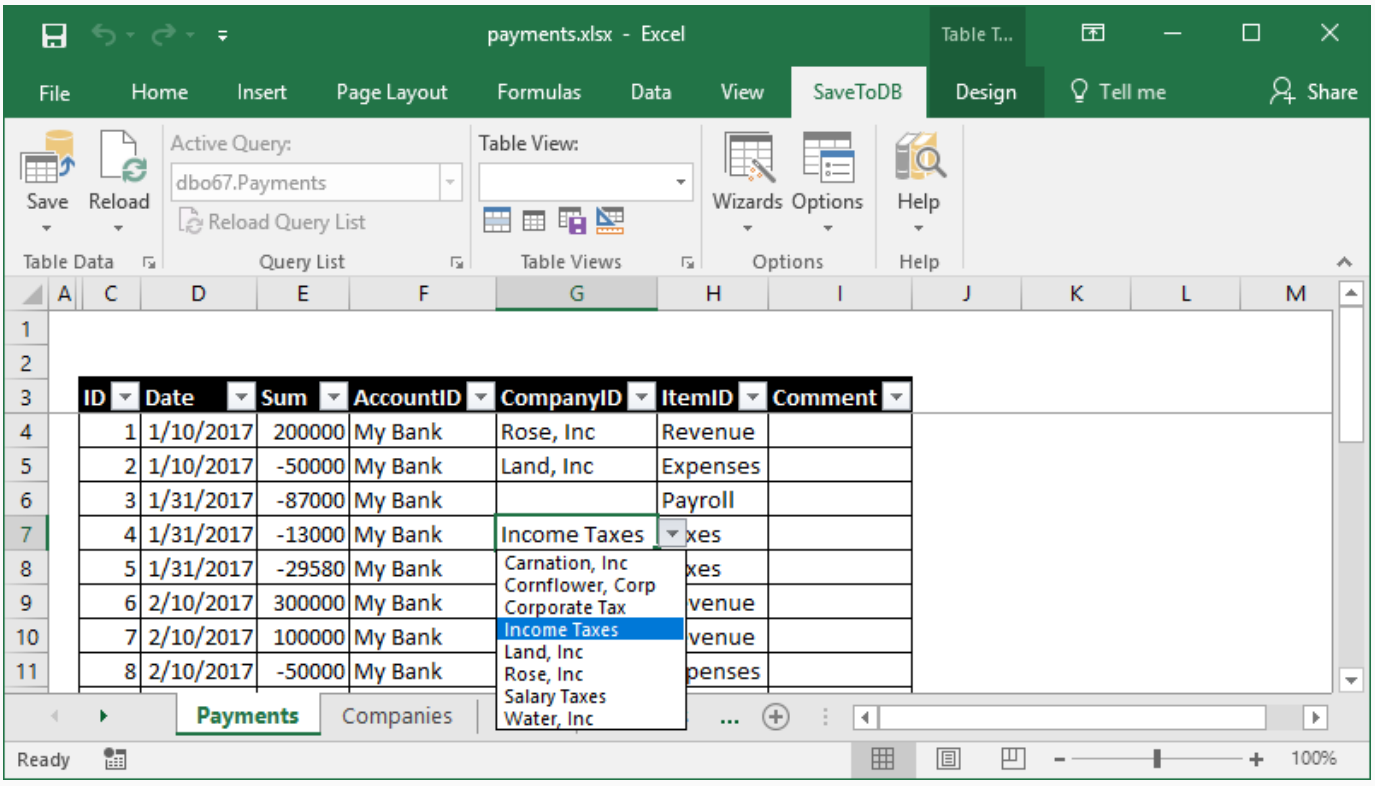
ID	TABLE	TABLE_NAME	COLUMN	EVENT_NAME	HANI	HANDLER	HANDLER_	HANDLER_CODE
	dbo67	Payments	AccountID	ValidationList	dbo67	Accounts	TABLE	ID,Name
	dbo67	Payments	CompanyID	ValidationList	dbo67	Companies	TABLE	ID,Name
	dbo67	Payments	ItemID	ValidationList	dbo67	Items	TABLE	ID,Name

Click the **Save** button to save the configuration.

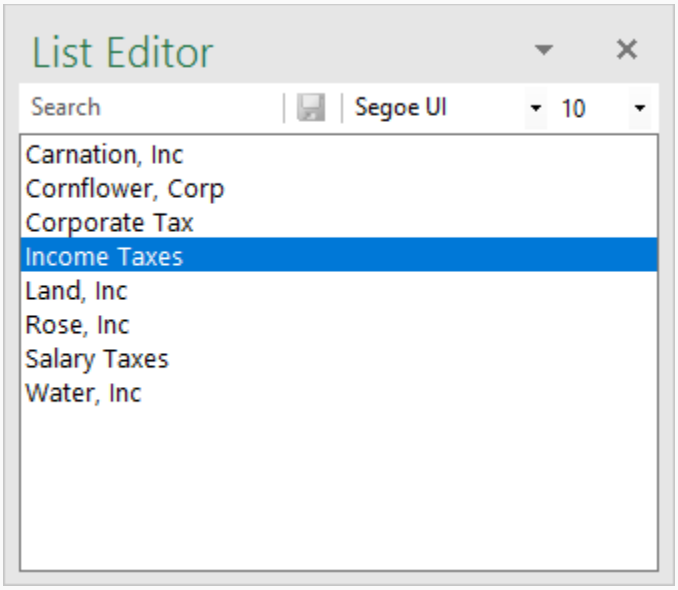
Let's switch to the **Payments** worksheet and click **Reload**, **Reload Data and Configuration**:



The add-in replaces id values with names and adds validation lists:



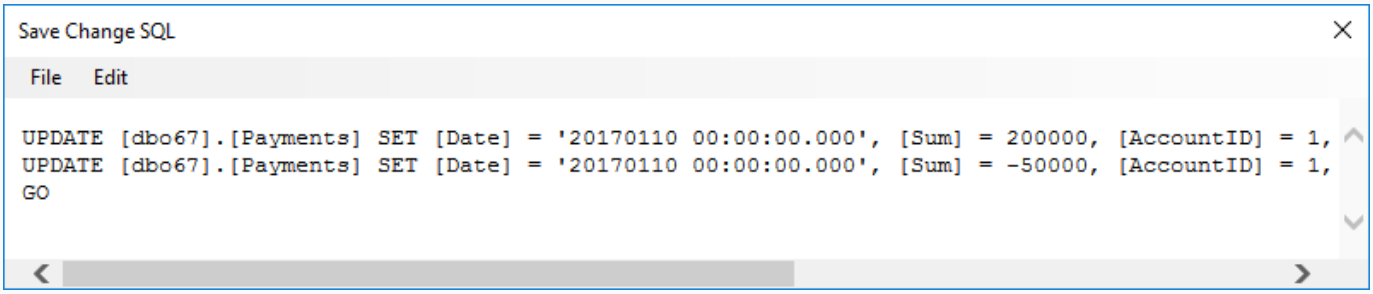
Moreover, the add-in activates the separate **List Editor** that allows users to select values from large lists in a comfortable way using search.



You may turn on/off the **List Editor** using the **Options**, **Show List Editor Task Pane** option.

Let's change a couple of rows in the table and click the **Save**, **View Save Changes SQL**.

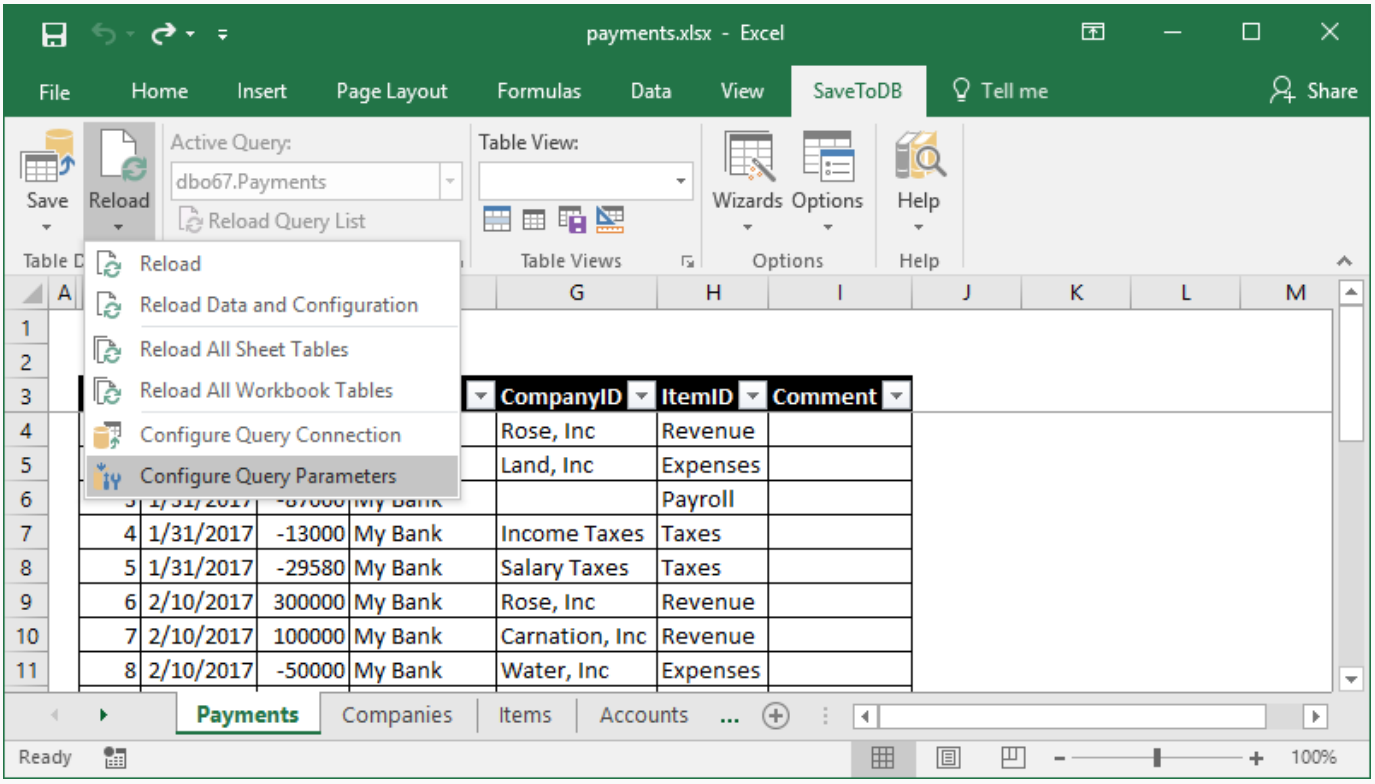
In my case, the add-in generates the following SQL commands:



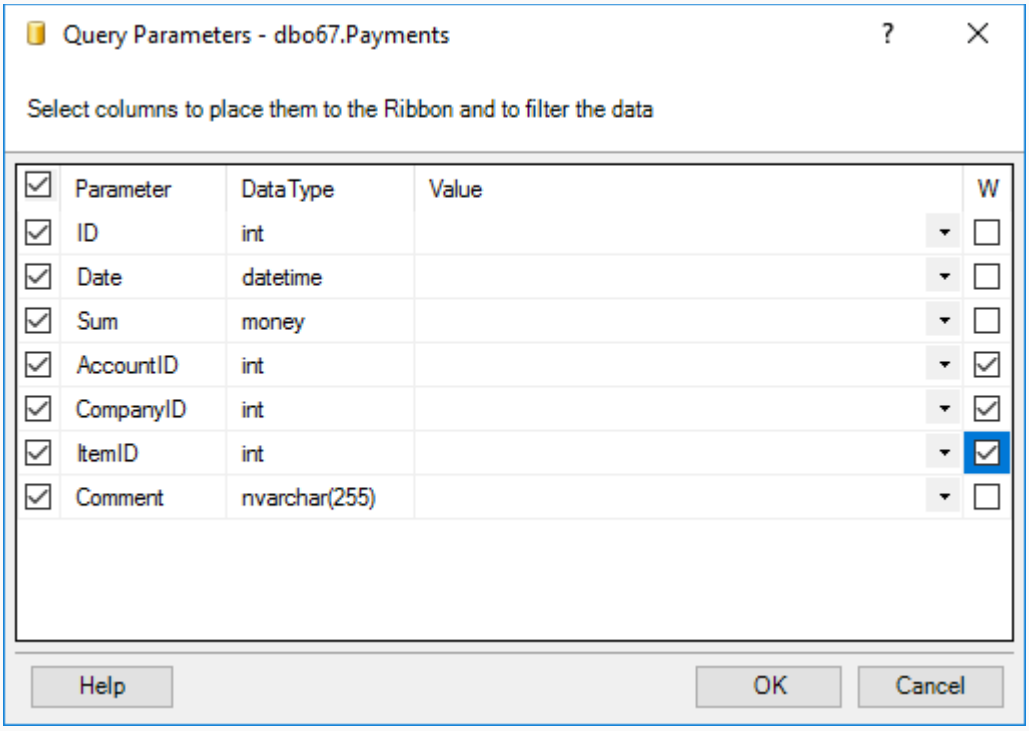
As we may see, the SaveToDB add-in uses id values instead of names as it should be.

Chapter 6. Query Parameters

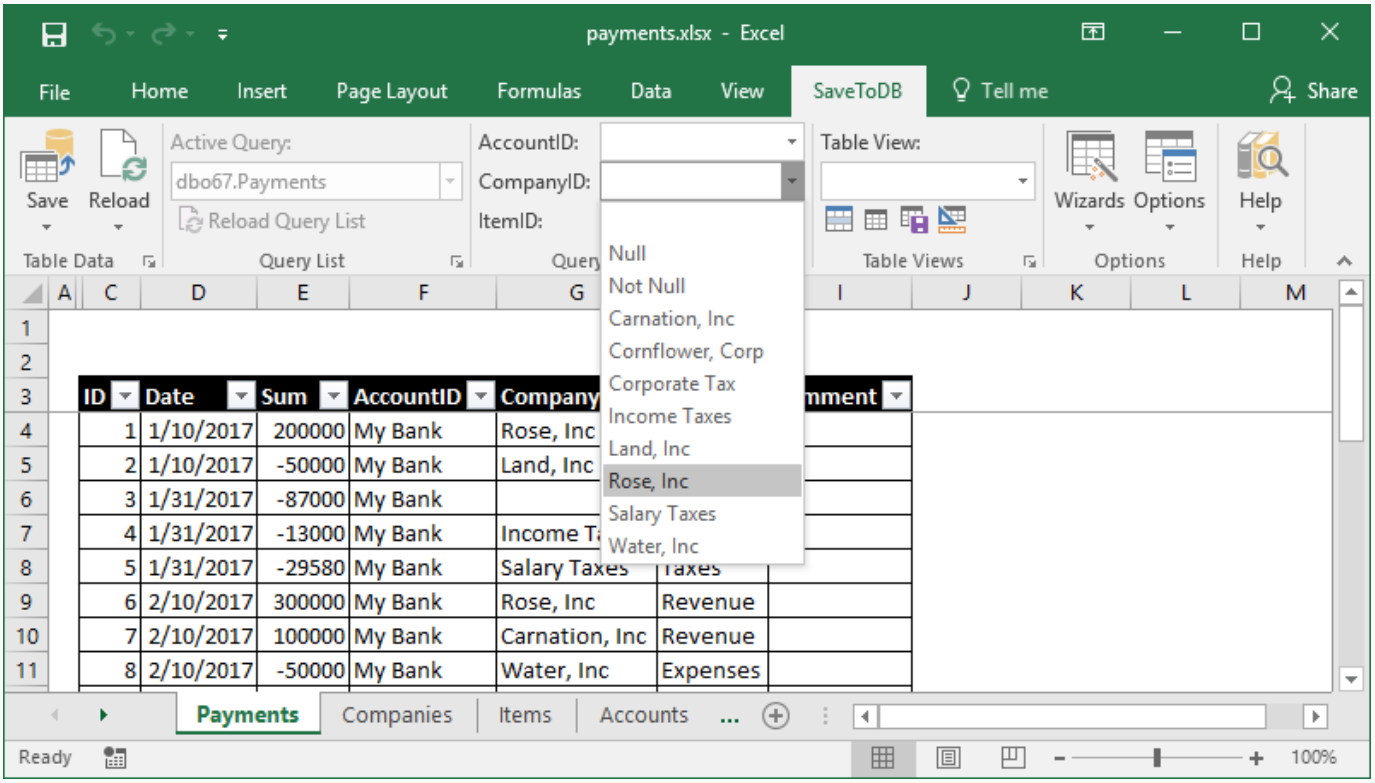
Let's make our table more interactive. Run **Reload**, **Configure Query Parameters**:



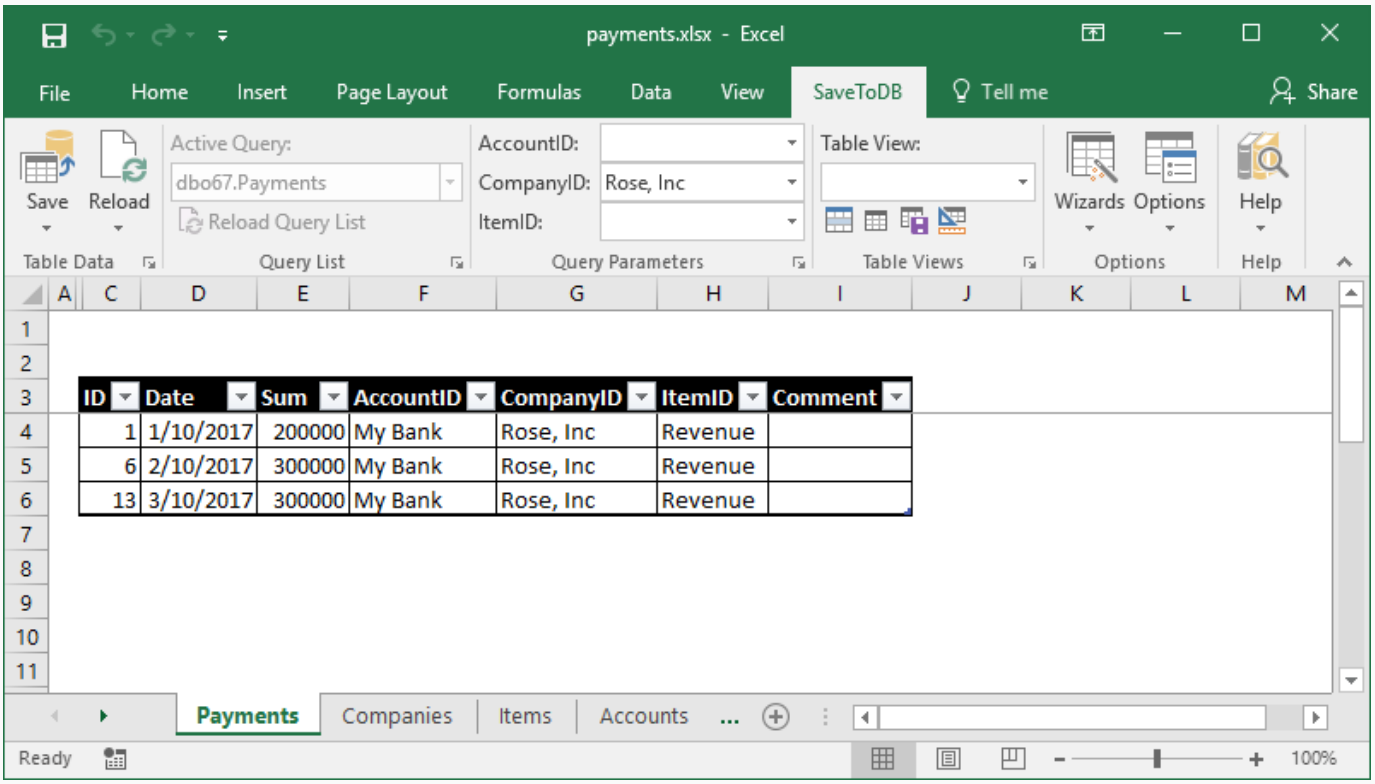
Check the AccountID, CompanyID, and ItemID fields in the **W** (WHERE) column:



The SaveToDB add-in places the selected fields to the ribbon. So, our users may filter data:



Let’s choose Rose, Inc.



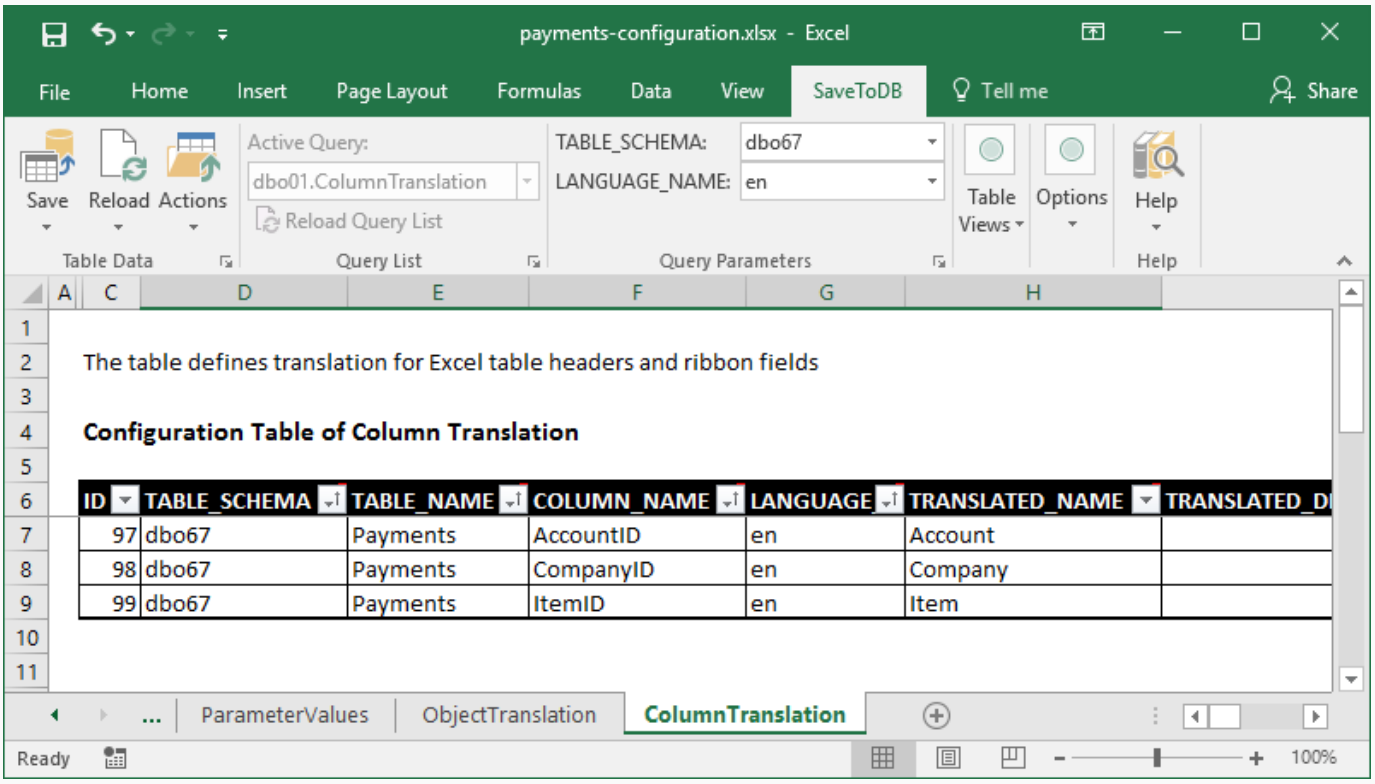
This feature allows working without auto-filters and loading fewer data.

Chapter 7. Column Name Translation

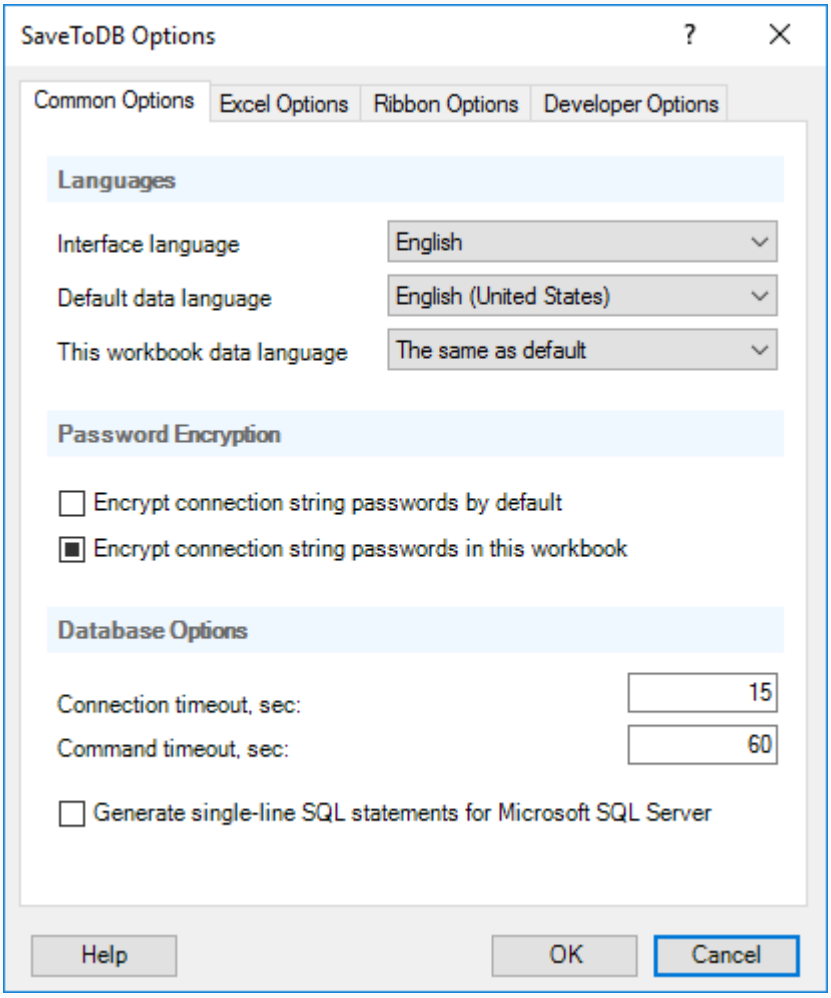
The Payments table shows fields in the table and at the ribbon like AccountID, CompanyID, and ItemID.

Let's change these database field names to business names.

Switch to the **payments-configuration** workbook, select the **ColumnTranslation** worksheet and add the data:



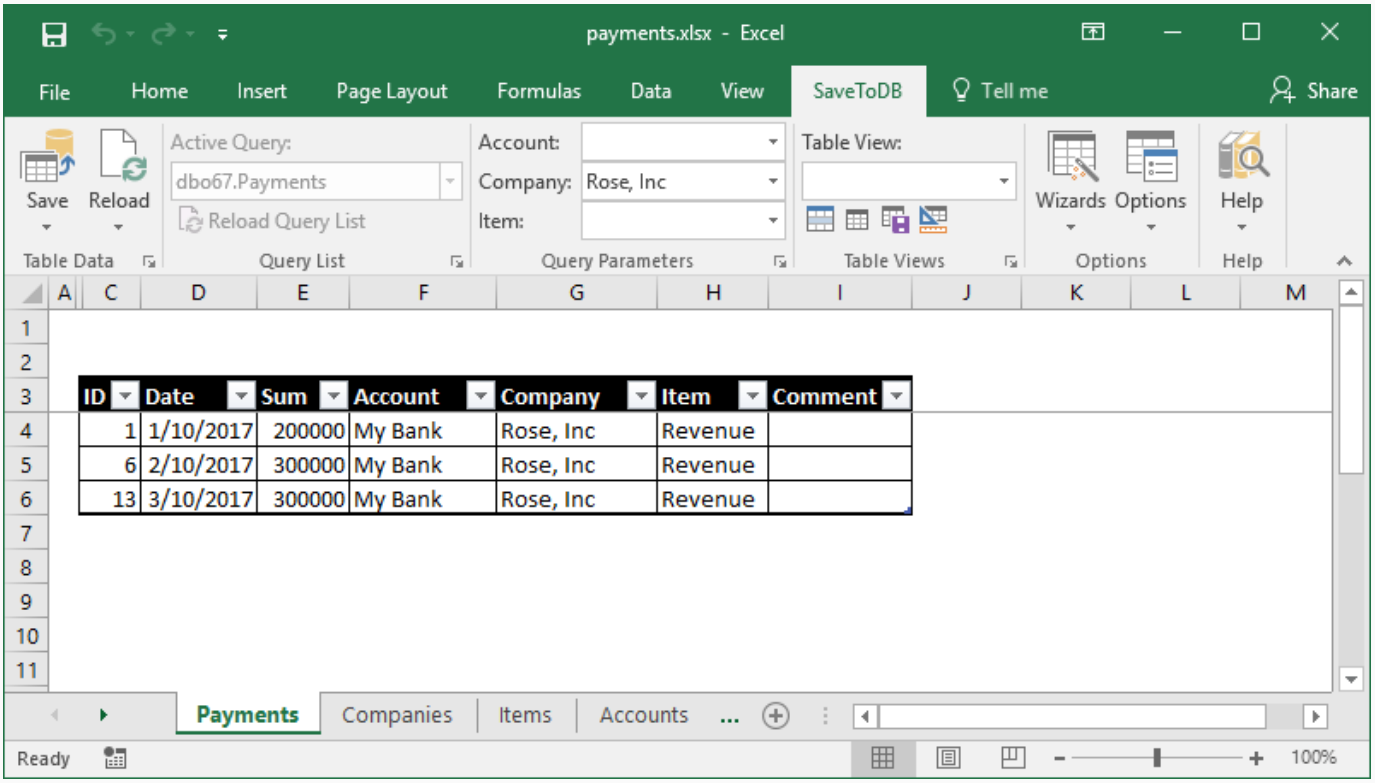
Then switch to the **payments** workbook and click **Options**:



Choose the **Default data language** as English and click **OK**.

Of course, in your applications, you may use any language or even multiple languages.

Click **Reload**, **Reload Data and Configuration**.



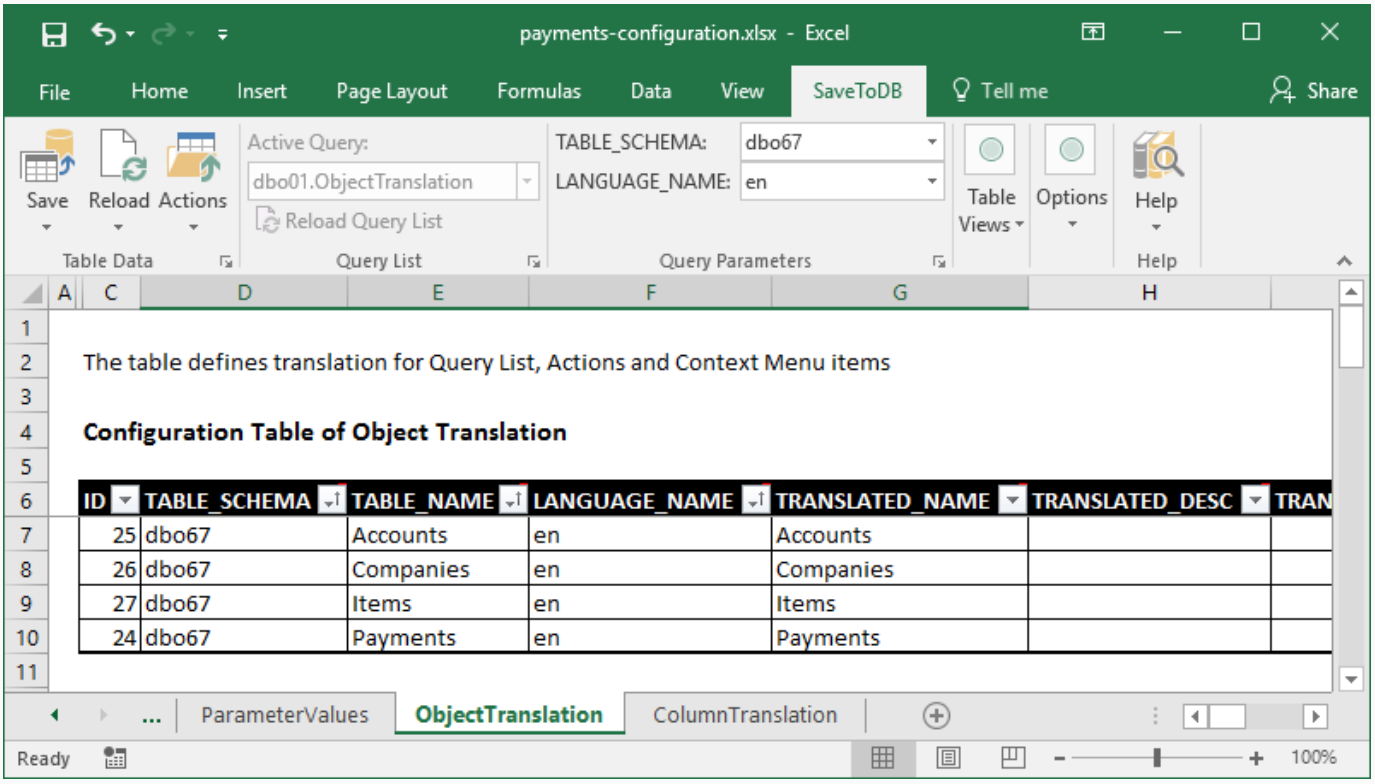
The add-in shows Account, Company, and Item instead of AccountID, CompanyID, and ItemID.

You may turn off this feature in the **Options** dialog box as described above.

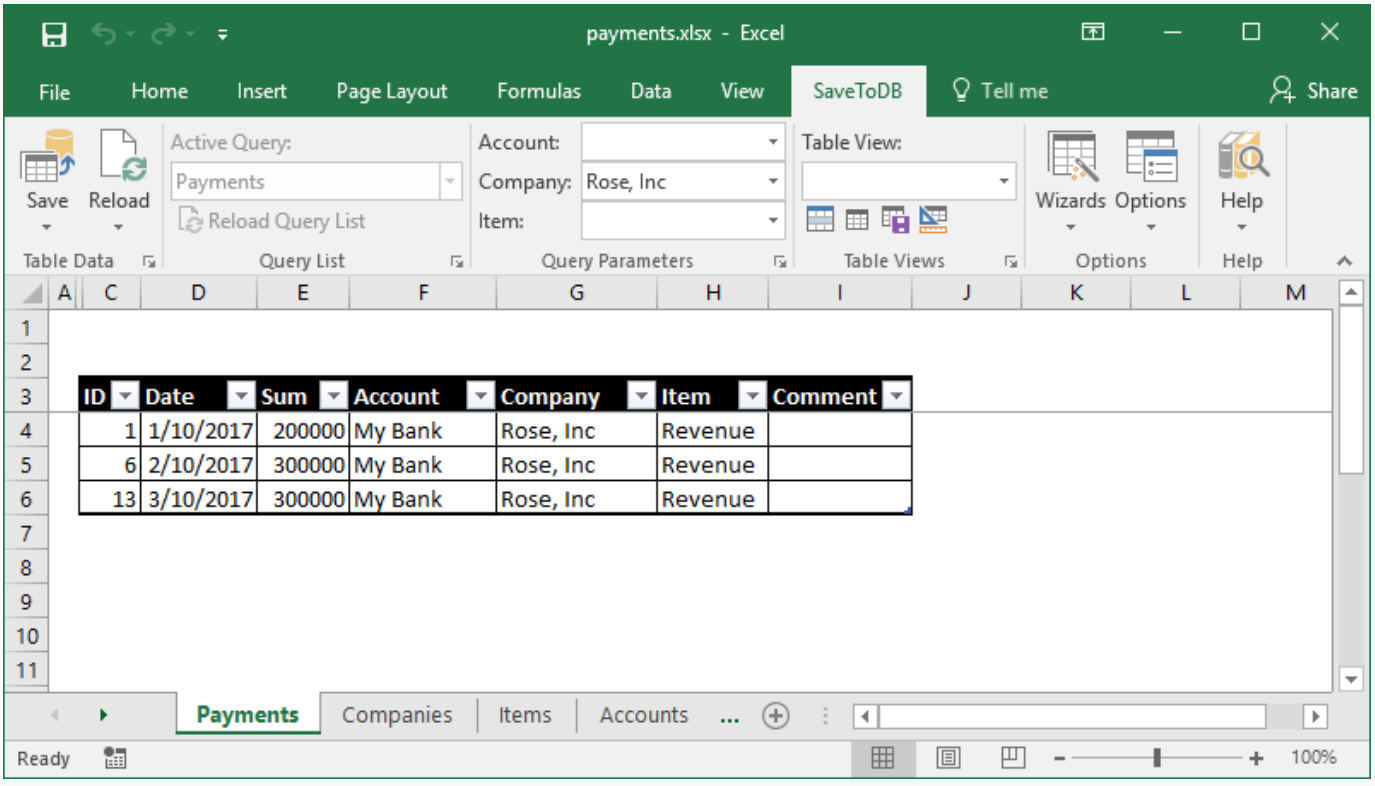
Chapter 8. Object Name Translation

We have translated field names. Also, we may translate database object names shown in Excel.

Switch to the **payments-configuration** workbook, select the **ObjectTranslation** worksheet and add the data:



Then return and click **Reload**, **Reload Data and Configuration**. Now we see business names everywhere.

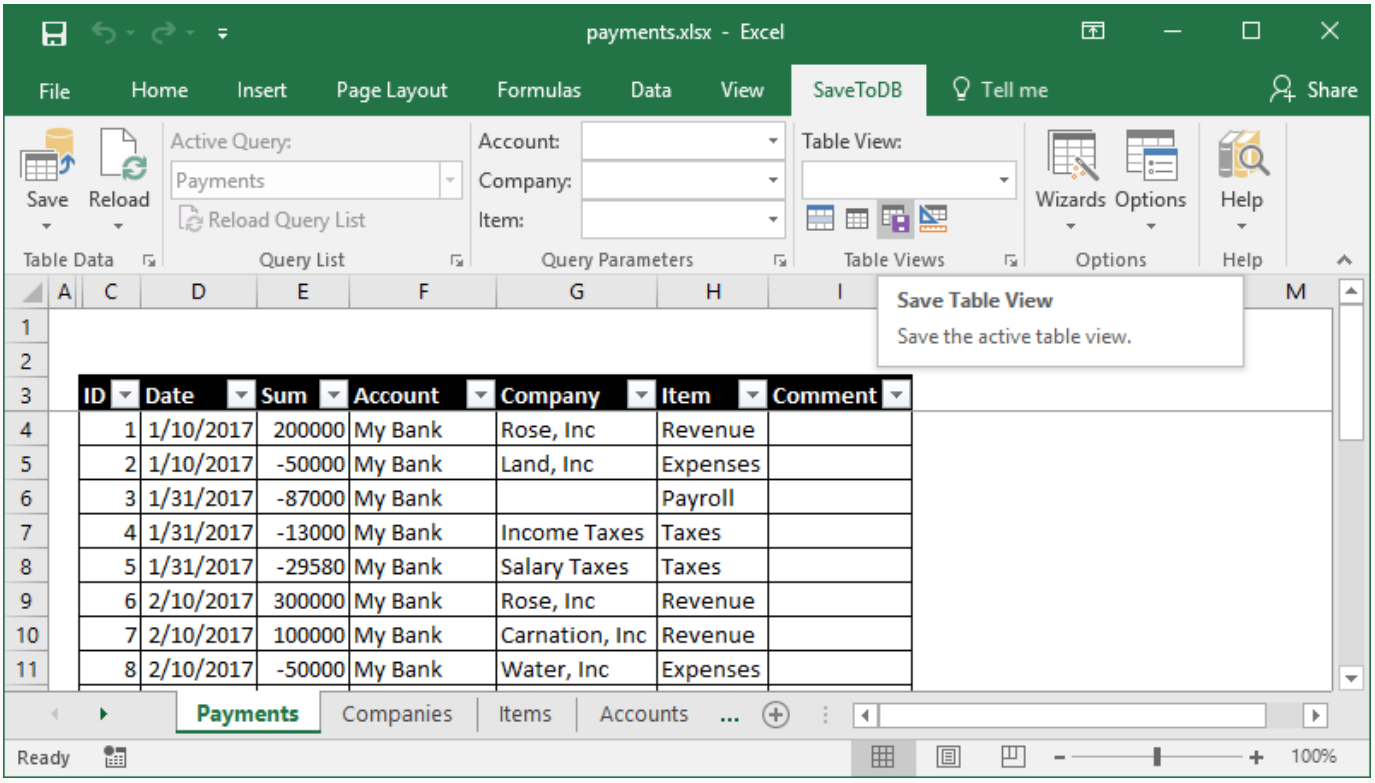


Chapter 9. Table Views

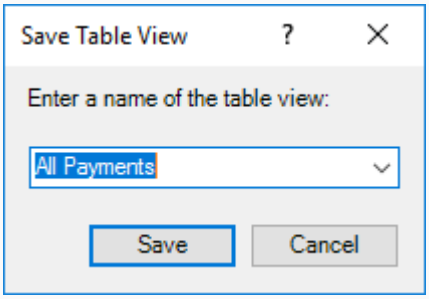
Users often apply different filters to loaded data, hide and unhide columns, sort in various ways, etc.

The SaveToDB add-in may help to save such user views and even share them with colleagues.

Let's remove all WHERE filters and click the **Save Table View** button in the **Table Views** group:

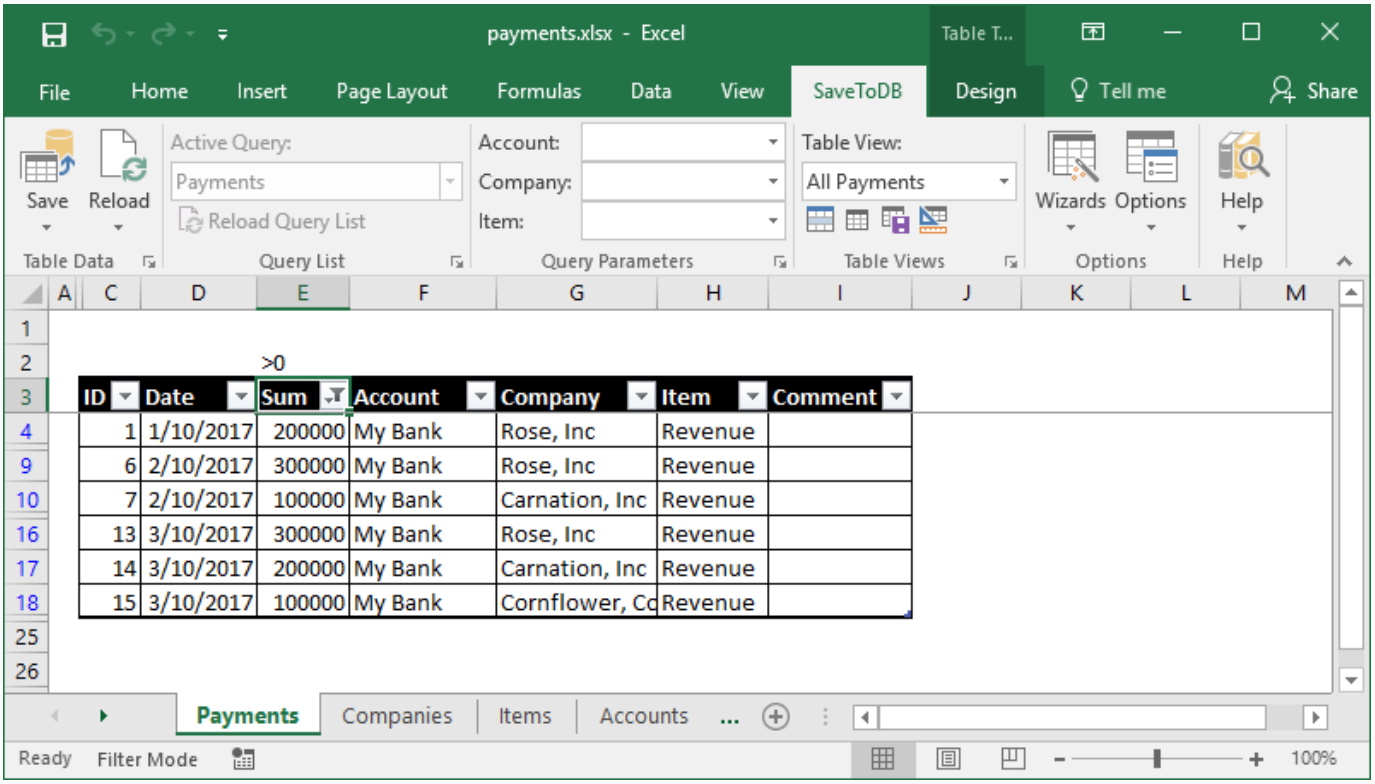


Type **All Payments** and click **Save**.



We see the name of the current view, All Payments, in the **Table View** field.

Type **>0** in cell E2.

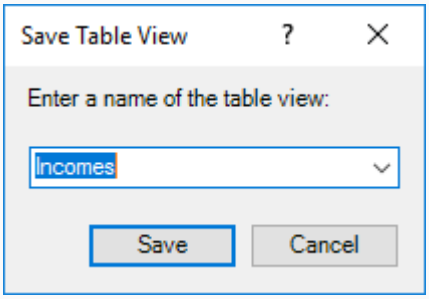


The add-in applies the filter to the Sum column.

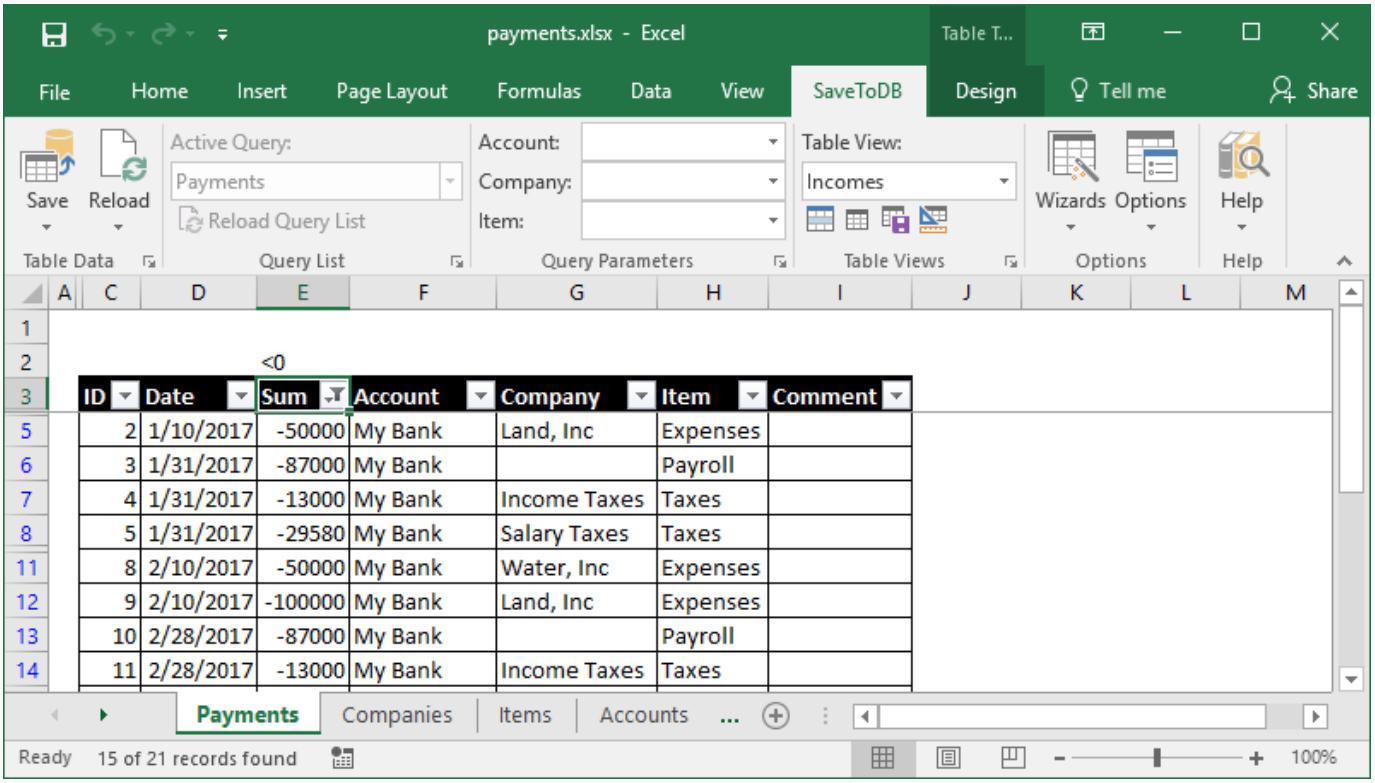
This is a reason why it is better to insert tables at cell B3.

Users may use row 2 (as a row over the table) as auto-filters. Also, they may place formulas in row 1.

Let's continue and save the view as **Incomes** (click the **Save Table View** button again):

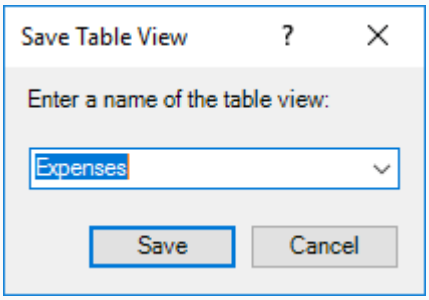


Type **<0** in cell E2.

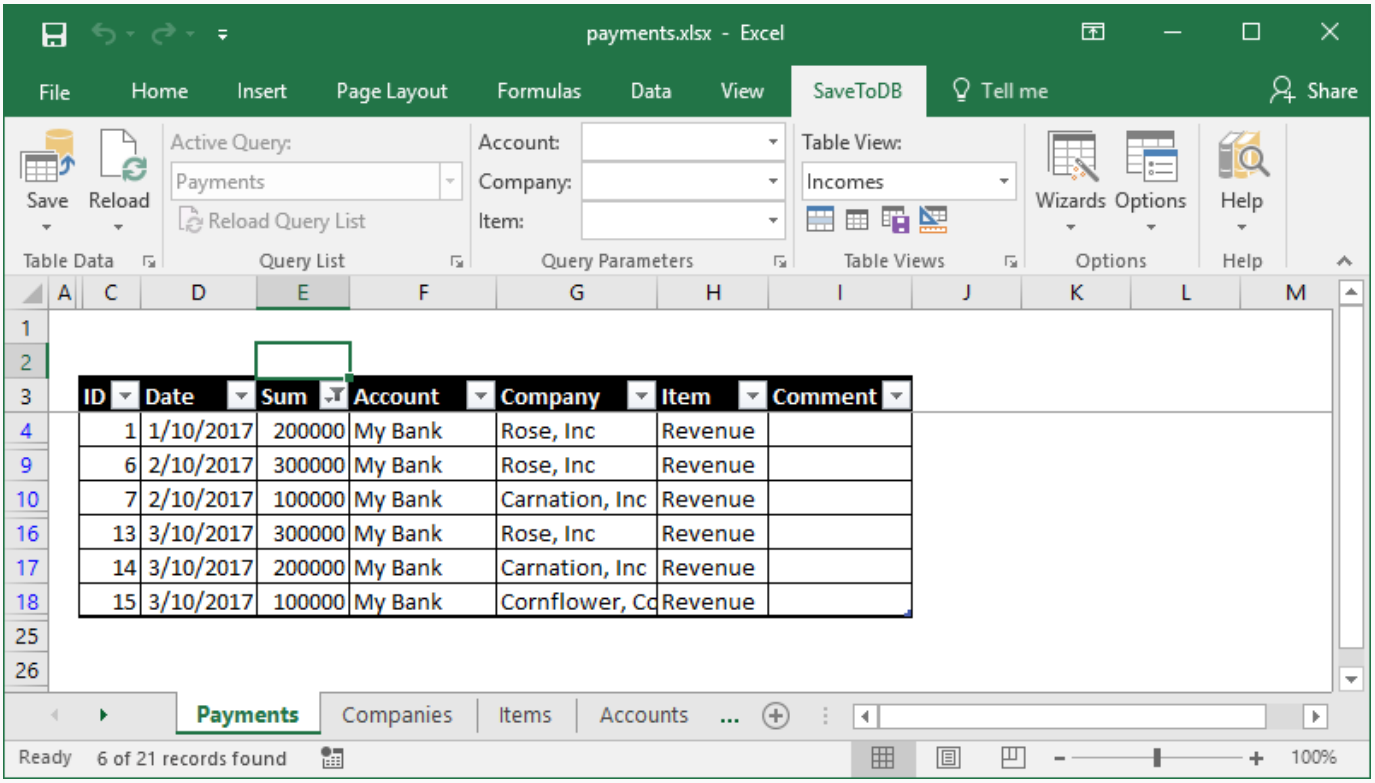


The add-in applies the new filter to the Sum column.

Save the view as **Expenses**.



Remove the filter in cell E2 and apply the **Incomes** view:



ID	Date	Sum	Account	Company	Item	Comment
1	1/10/2017	200000	My Bank	Rose, Inc	Revenue	
6	2/10/2017	300000	My Bank	Rose, Inc	Revenue	
7	2/10/2017	100000	My Bank	Carnation, Inc	Revenue	
13	3/10/2017	300000	My Bank	Rose, Inc	Revenue	
14	3/10/2017	200000	My Bank	Carnation, Inc	Revenue	
15	3/10/2017	100000	My Bank	Cornflower, Cc	Revenue	

As we may expect, the add-in applies the saved filter to the Sum column.

I am sure, your users will be happy, and you will have fewer requests for new small database views.

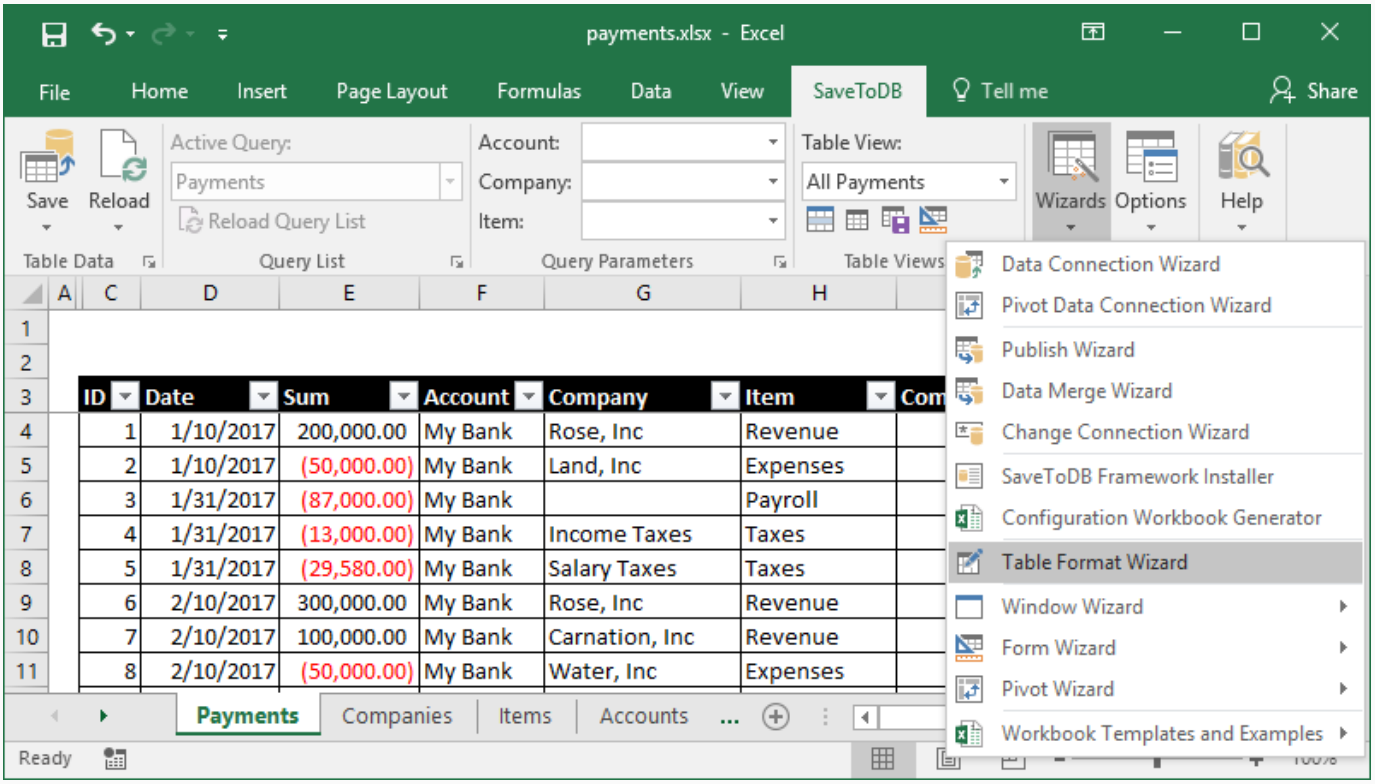
Chapter 10. Table Format Wizard

We have formatted tables and added views in the previous steps in the payments.xlsx workbook.

If a user connects to a database from a new workbook, he will have Excel defaults.

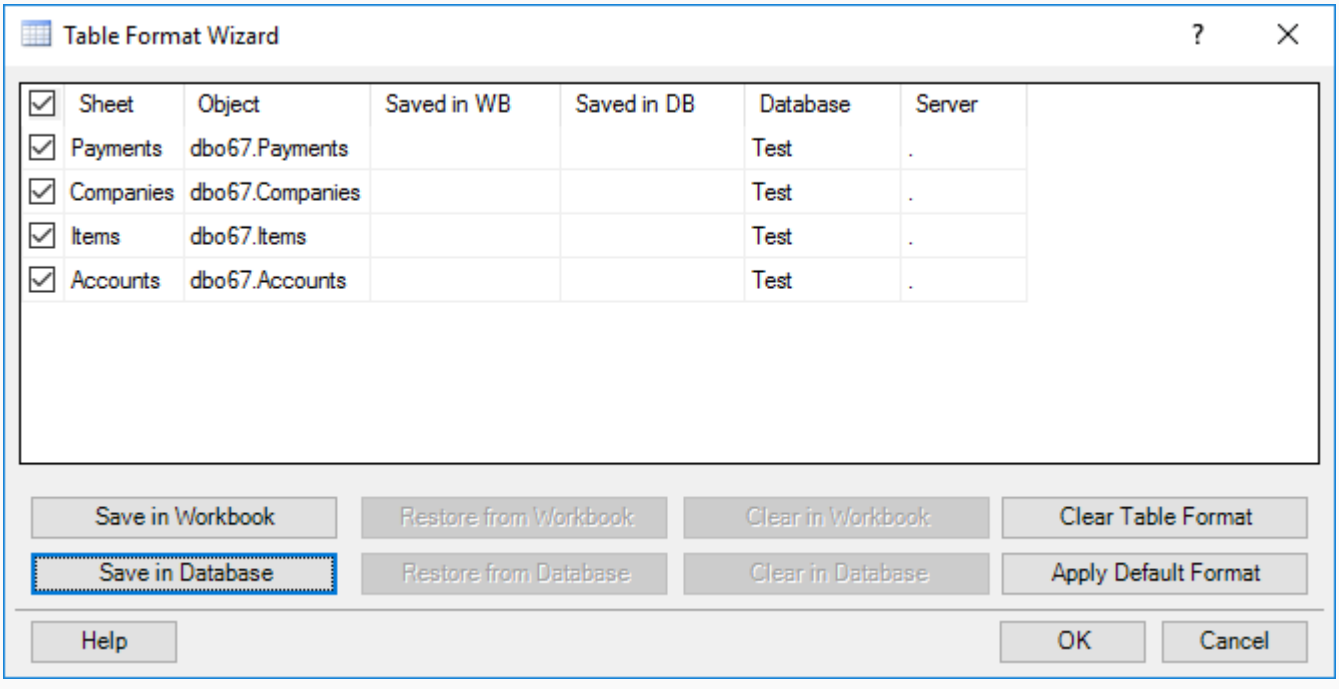
We can fix this publishing table formats and views to a database using **Table Format Wizard**.

Let's format the Sum column, set default column widths, apply the default table view and run the wizard:

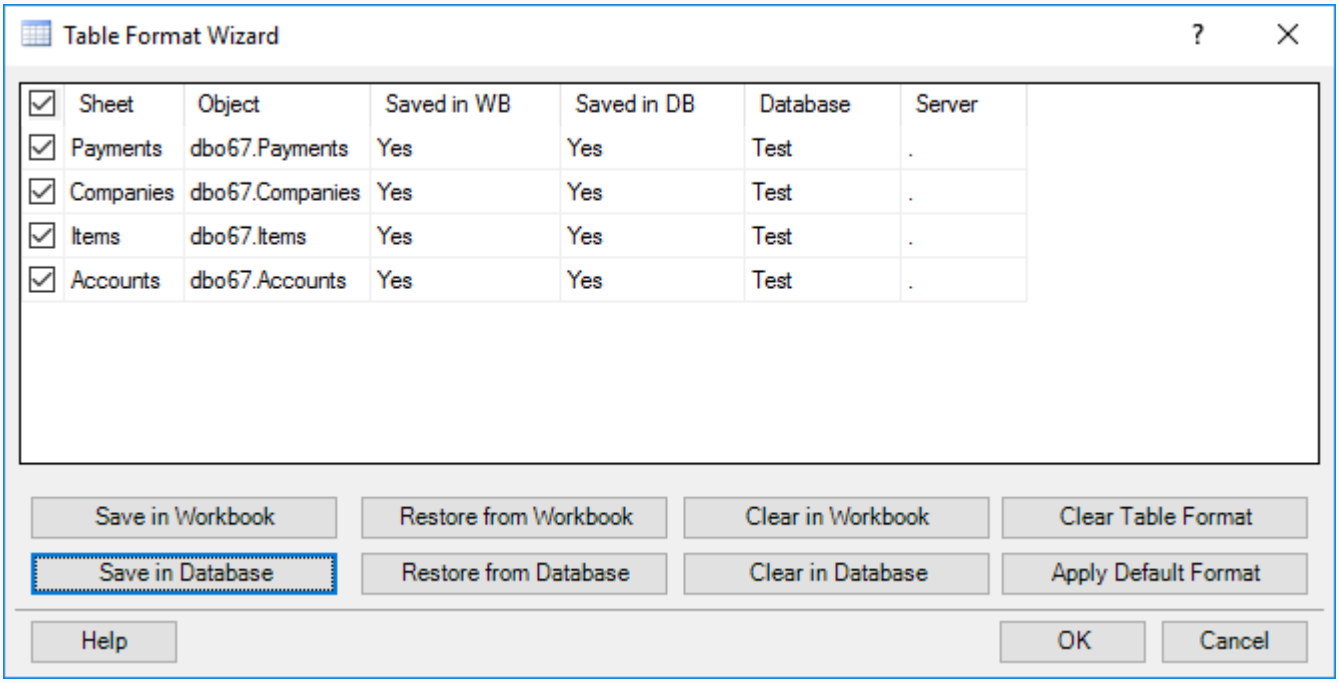


ID	Date	Sum	Account	Company	Item
1	1/10/2017	200,000.00	My Bank	Rose, Inc	Revenue
2	1/10/2017	(50,000.00)	My Bank	Land, Inc	Expenses
3	1/31/2017	(87,000.00)	My Bank		Payroll
4	1/31/2017	(13,000.00)	My Bank	Income Taxes	Taxes
5	1/31/2017	(29,580.00)	My Bank	Salary Taxes	Taxes
6	2/10/2017	300,000.00	My Bank	Rose, Inc	Revenue
7	2/10/2017	100,000.00	My Bank	Carnation, Inc	Revenue
8	2/10/2017	(50,000.00)	My Bank	Water, Inc	Expenses

In the wizard, select all tables and click the **Save in Database** button:



The wizard saves formats and changes its state:



Now, users will get the same formats, views, and formulas of the tables when they connect to a database.

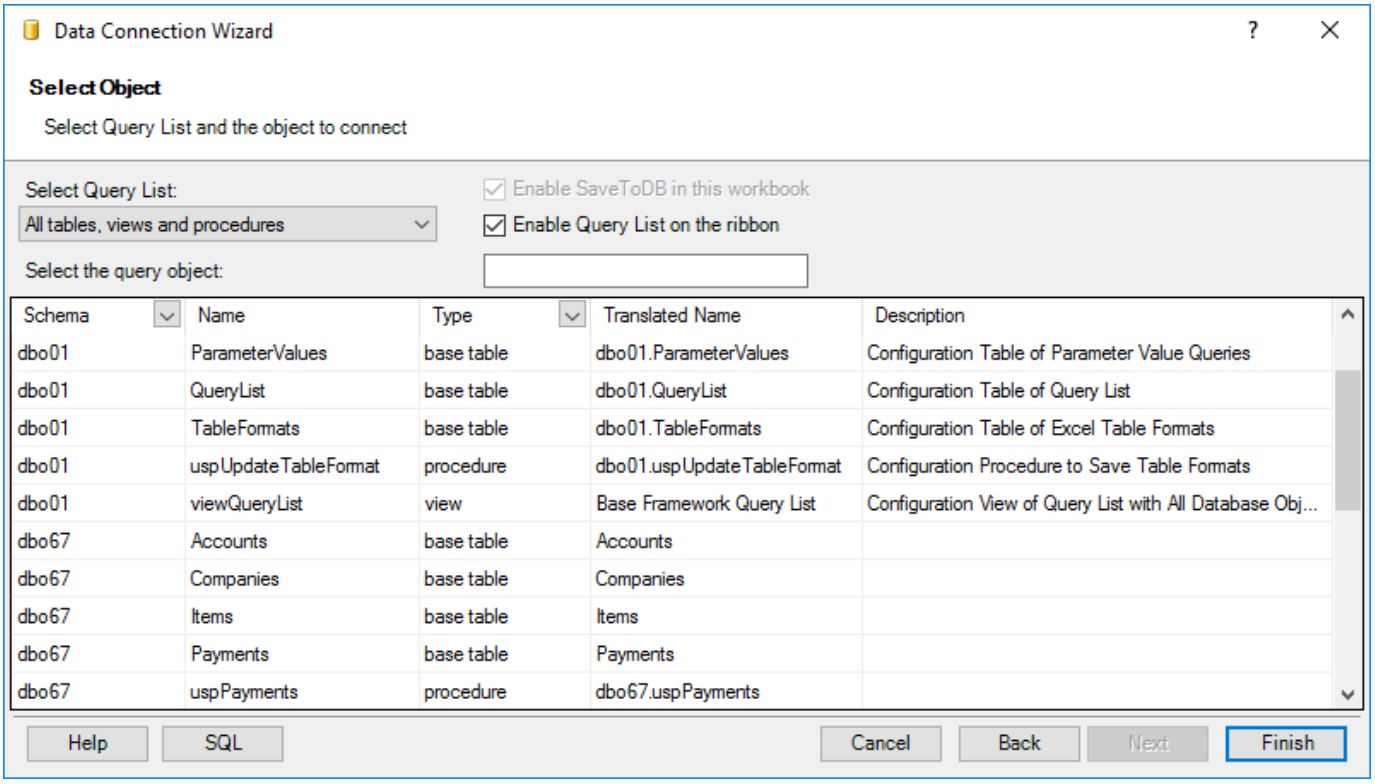
Use the wizard to republish new views later.

Users may use the **Restore from Database** button to reload the updated views.

Chapter 11. Framework Query List

Let's create a worksheet, name it as **Reports** and connect to the **viewPayments** view.

The **Select Object** screen has significantly changed since the last connection:

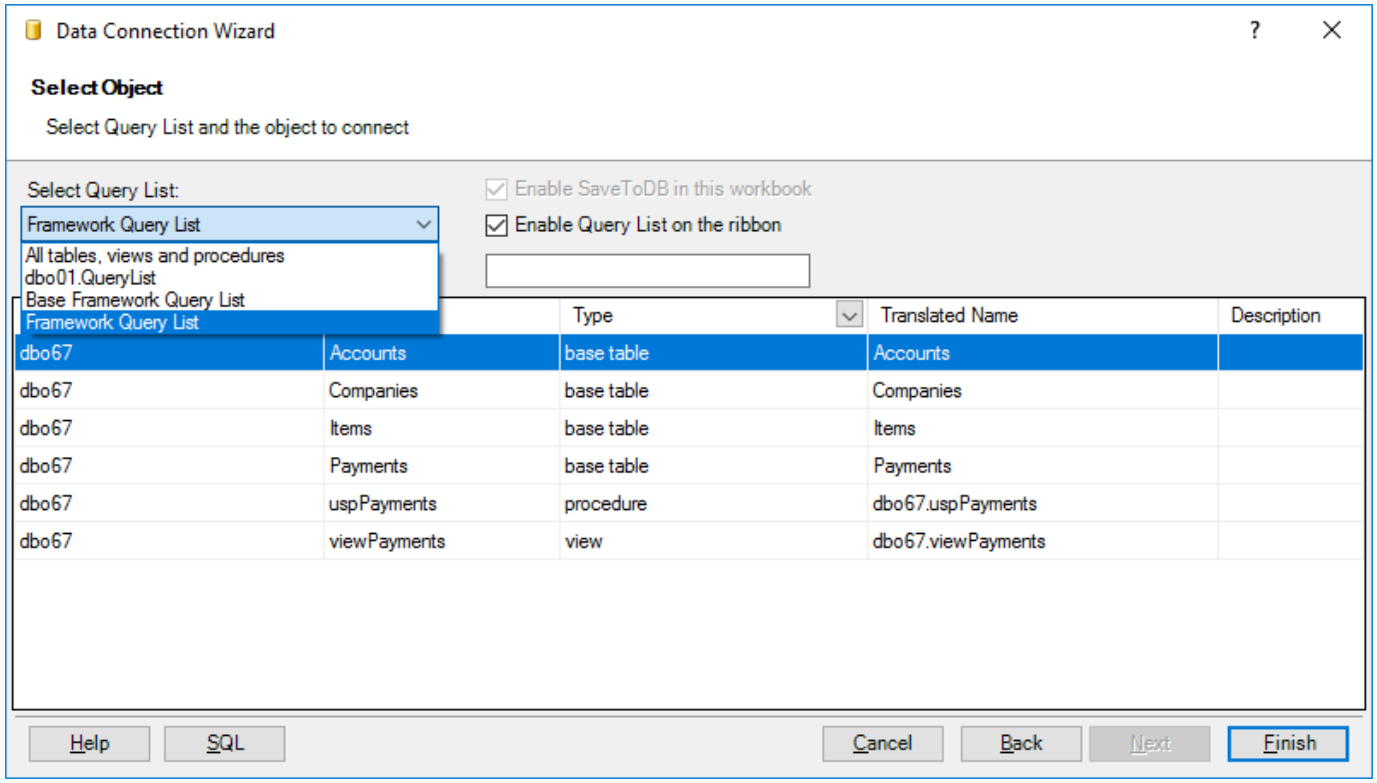


It includes much more objects and additional columns like **Translated Name** and **Description**.

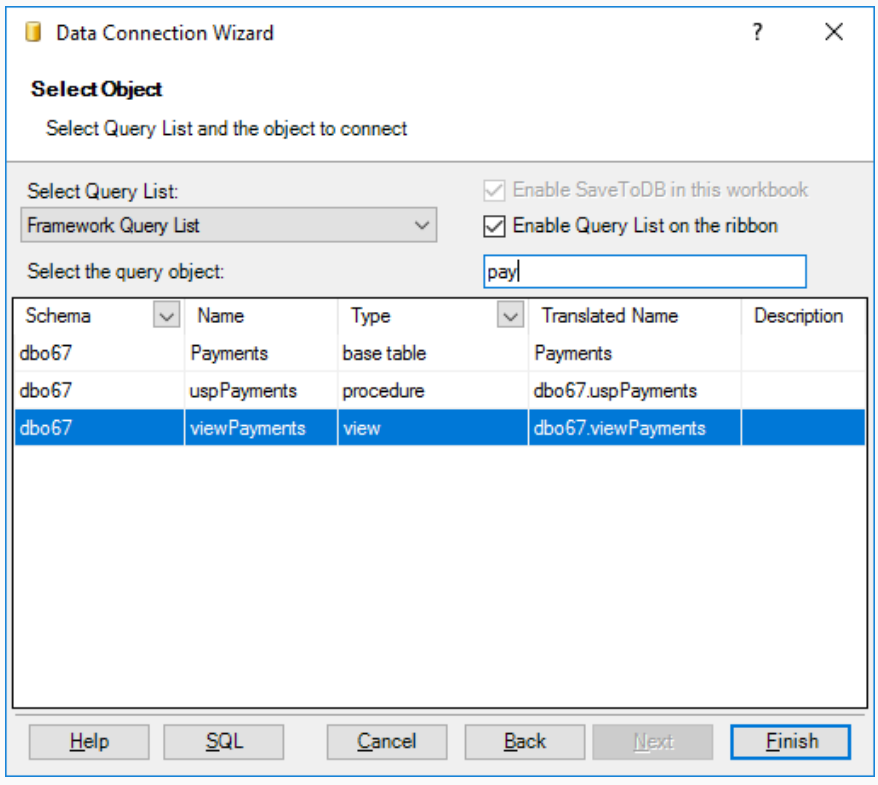
First of all, **Translated Name** and **Description** are shown as we added the SaveToDB Framework, added the object translation, and selected English in the **Options**. This feature helps users to understand database objects better.

The second, the wizard shows installed SaveToDB Framework objects.

Select the **Framework Query List** item in the **Select Query List** combobox, and the wizard displays source objects:

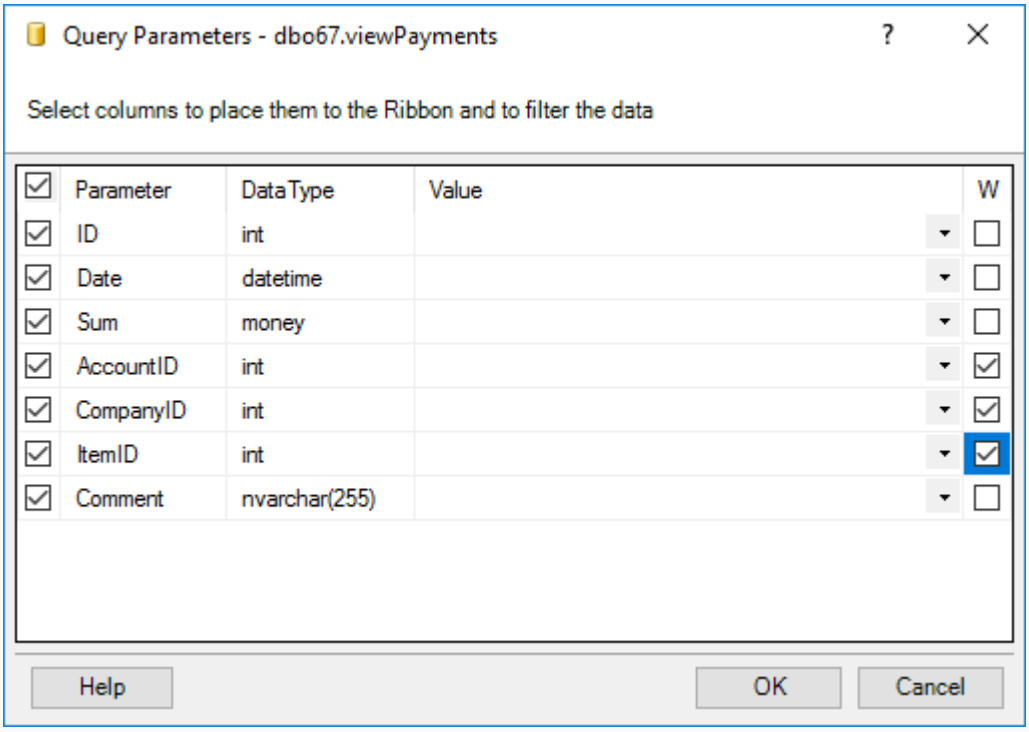


You may filter objects, also. Type **pay**, for example.

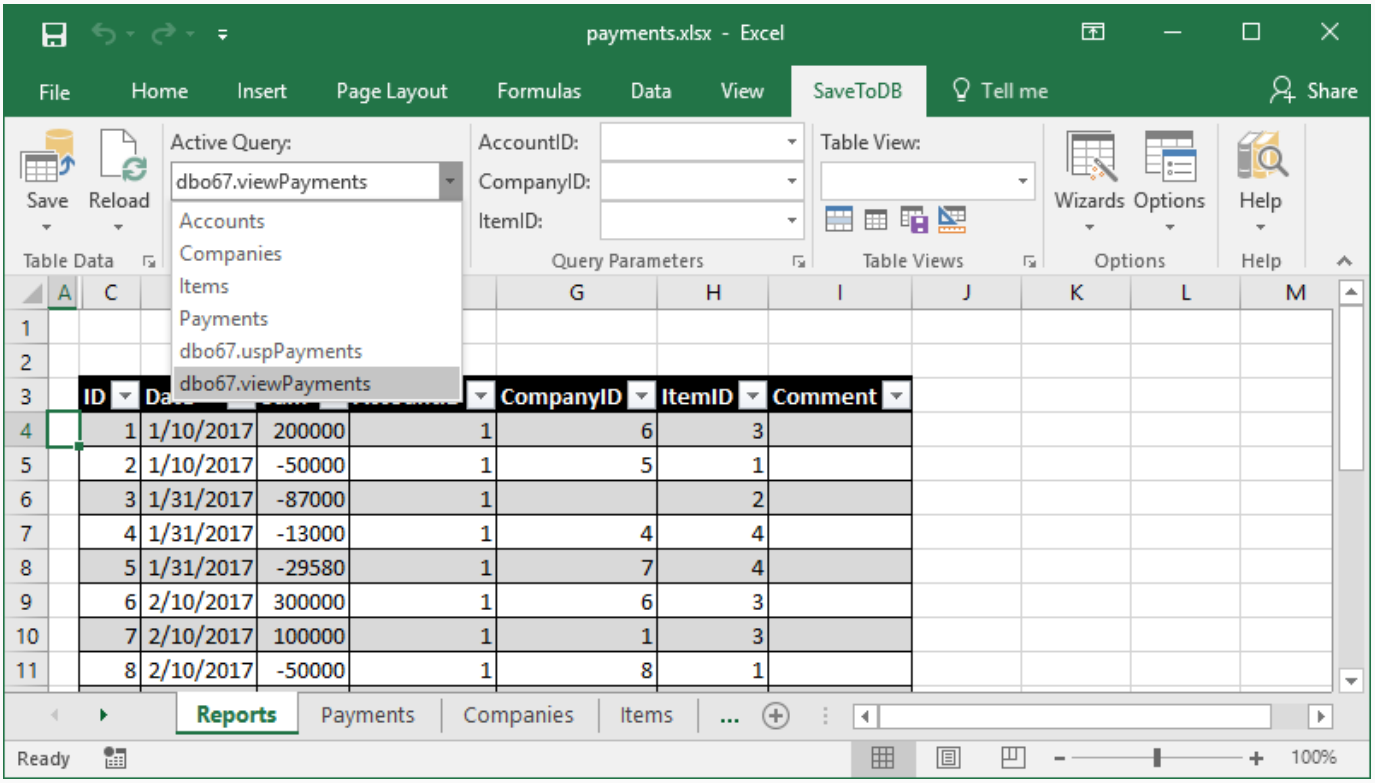


Let's select the **dbo67.viewPayments** view and leave **Enable Query List on the ribbon** checked.

Let's check the AccountID, CompanyID, and ItemID in the **W** column, and insert the view at cell B3.



We see a new feature. The ribbon **Query List** allows us to change the query.

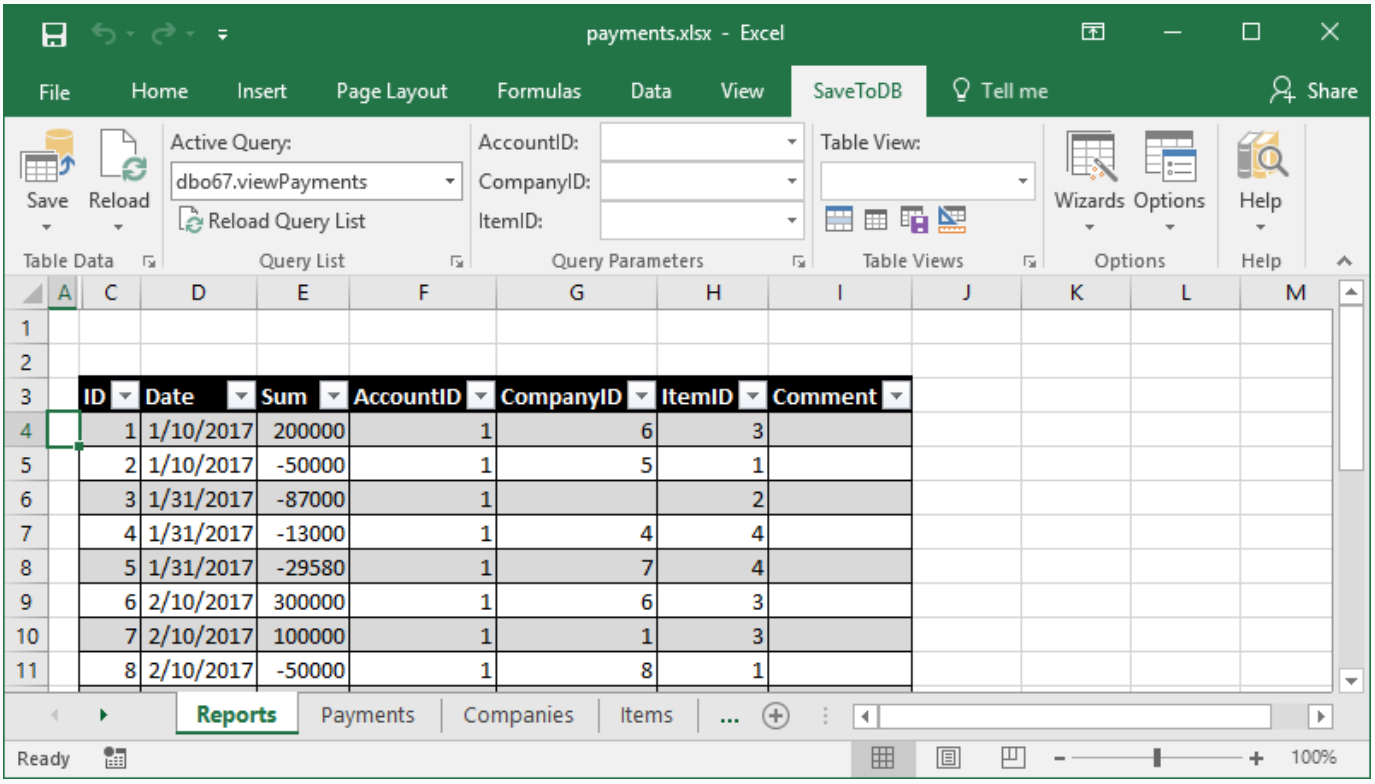


ID	Date	Sum	AccountID	CompanyID	ItemID	Comment
1	1/10/2017	200000	1	6	3	
2	1/10/2017	-50000	1	5	1	
3	1/31/2017	-87000	1		2	
4	1/31/2017	-13000	1	4	4	
5	1/31/2017	-29580	1	7	4	
6	2/10/2017	300000	1	6	3	
7	2/10/2017	100000	1	1	3	
8	2/10/2017	-50000	1	8	1	

This is a very useful feature. Users may use a few worksheets to work with multiple database objects.

Moreover, when developers add new objects to a database, users just reload the query list and can connect.

We have connected to a view:



ID	Date	Sum	AccountID	CompanyID	ItemID	Comment
1	1/10/2017	200000	1	6	3	
2	1/10/2017	-50000	1	5	1	
3	1/31/2017	-87000	1		2	
4	1/31/2017	-13000	1	4	4	
5	1/31/2017	-29580	1	7	4	
6	2/10/2017	300000	1	6	3	
7	2/10/2017	100000	1	1	3	
8	2/10/2017	-50000	1	8	1	

Good news:

1. We can save changes (as the view is updateable).
2. We can use ribbon parameters to filter data.

Bad news:

1. We see foreign key values instead of names in columns and ribbon parameters.
2. We see AccountID, CompanyID, and ItemID in columns and ribbon parameters.
3. We see the dbo67.viewPayments database name in the Query List.
4. The table is unformatted.
5. There are no predefined table views.

Let's fix the "bad news."

Replacing foreign keys values with names

Switch to the **payments-configuration** workbook, select the **EventHandlers** worksheet, copy and paste three rows of the **Payment** table, and change Payments to viewPayments in the **TABLE_NAME** field. Click **Save**.

ID	TABLE	TABLE_NAME	COLUMN	EVENT_NAME	HAN	HANDLER	HANDLER_	HANDLER_CODE
26	dbo67	Payments	AccountID	ValidationList	dbo67	Accounts	TABLE	ID,Name
27	dbo67	Payments	CompanyID	ValidationList	dbo67	Companies	TABLE	ID,Name
28	dbo67	Payments	ItemID	ValidationList	dbo67	Items	TABLE	ID,Name
29	dbo67	viewPayments	AccountID	ValidationList	dbo67	Accounts	TABLE	ID,Name
30	dbo67	viewPayments	CompanyID	ValidationList	dbo67	Companies	TABLE	ID,Name
31	dbo67	viewPayments	ItemID	ValidationList	dbo67	Items	TABLE	ID,Name

It is easy. We just copied the existing Payments table configuration.

Translating column names

Select the **ColumnTranslation** worksheet, copy and paste three rows of the **Payment** table, and change Payments to viewPayments in the **TABLE_NAME** field. Click **Save**.

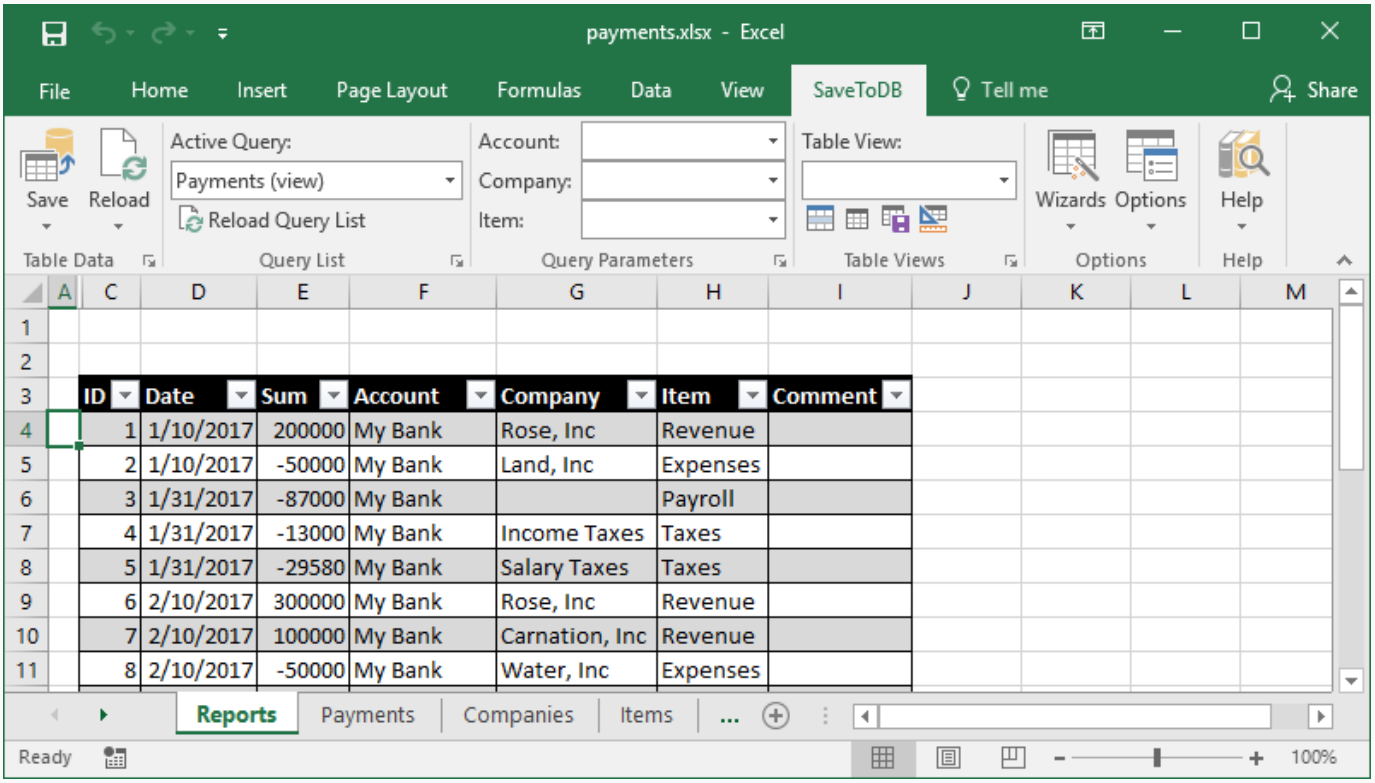
ID	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	LANGUAGE	TRANSLATED_NAME	TRANSLATED_DESC
97	dbo67	Payments	AccountID	en	Account	
98	dbo67	Payments	CompanyID	en	Company	
99	dbo67	Payments	ItemID	en	Item	
100	dbo67	viewPayments	AccountID	en	Account	
101	dbo67	viewPayments	CompanyID	en	Company	
102	dbo67	viewPayments	ItemID	en	Item	

Translating object names

Select the **ObjectTranslation** worksheet and add the translation for **viewPayments** and **uspPayments**:

ID	TABLE_SCHEMA	TABLE_NAME	L	TRANSLATED_NAME	TRANSLATED_DESC	TRANSLATED_COMMENT
25	dbo67	Accounts	en	Accounts		
26	dbo67	Companies	en	Companies		
27	dbo67	Items	en	Items		
24	dbo67	Payments	en	Payments		
29	dbo67	uspPayments	en	Payments (sp)		
28	dbo67	viewPayments	en	Payments (view)		

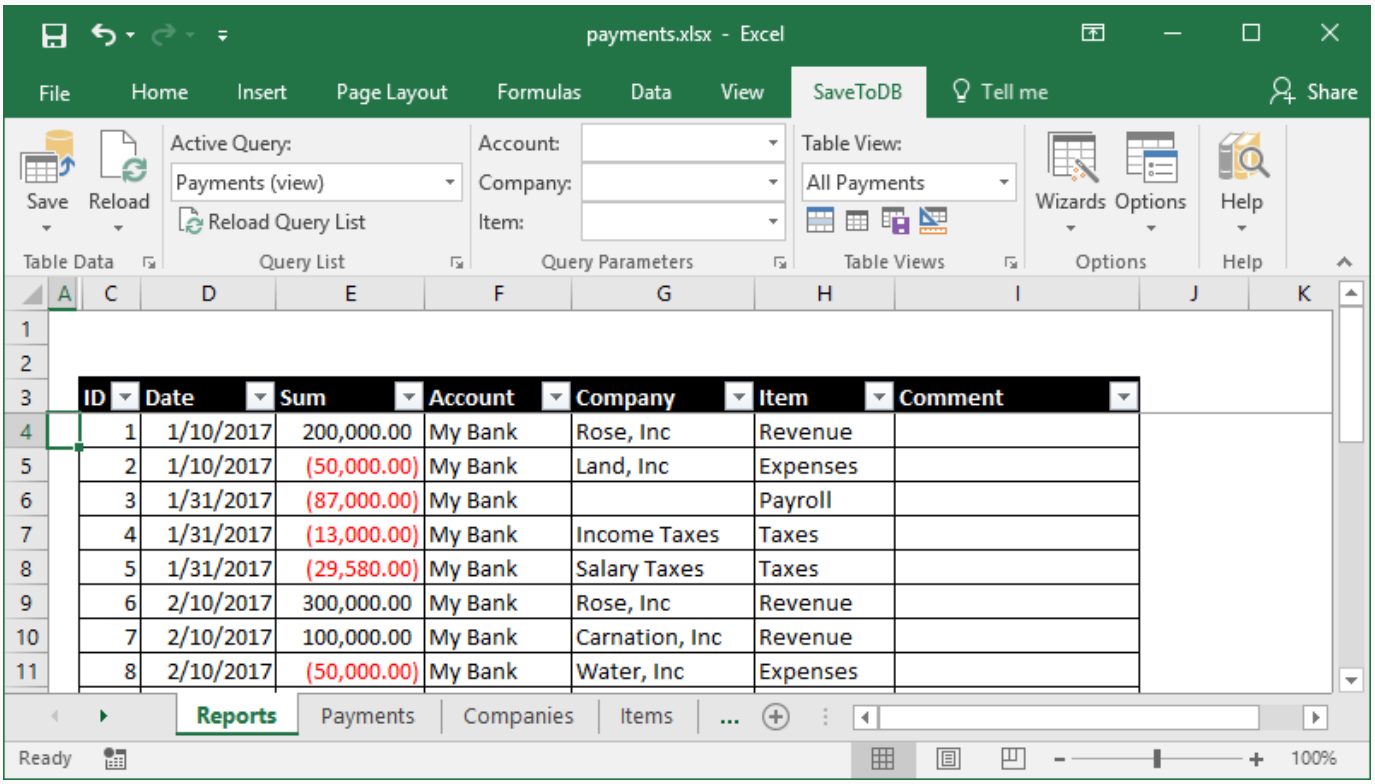
Now we can switch to the **payments** workbook, click the **Reload**, **Reload Data and Configuration** button, and the **Reload Query List** button in the **Query List** group. We see that we have fixed the first three points.



ID	Date	Sum	Account	Company	Item	Comment
1	1/10/2017	200000	My Bank	Rose, Inc	Revenue	
2	1/10/2017	-50000	My Bank	Land, Inc	Expenses	
3	1/31/2017	-87000	My Bank		Payroll	
4	1/31/2017	-13000	My Bank	Income Taxes	Taxes	
5	1/31/2017	-29580	My Bank	Salary Taxes	Taxes	
6	2/10/2017	300000	My Bank	Rose, Inc	Revenue	
7	2/10/2017	100000	My Bank	Carnation, Inc	Revenue	
8	2/10/2017	-50000	My Bank	Water, Inc	Expenses	

Now we can format the table, add the required table views, and save them to a database using the **Table Format Wizard**, as described in the previous topics.

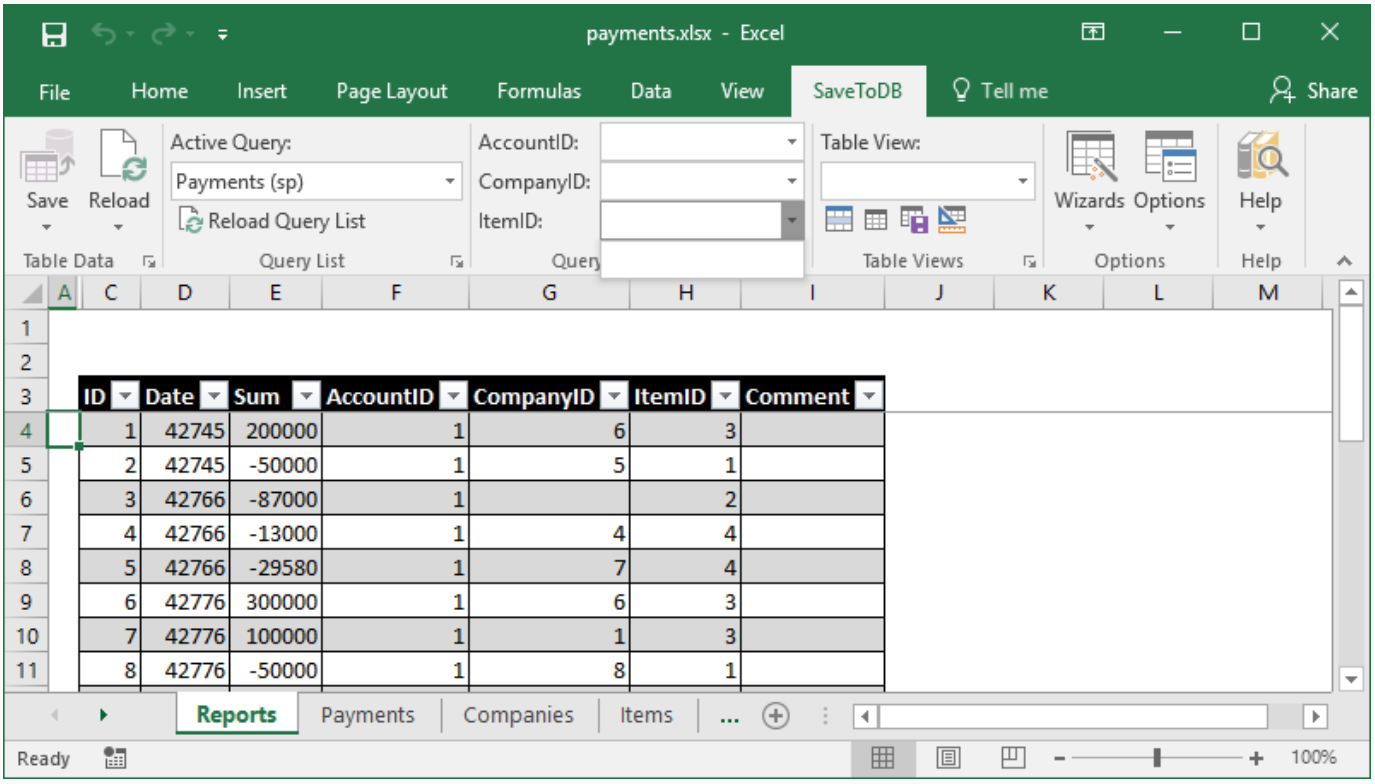
After these steps, users will get the same preconfigured and formatted view from a database.



ID	Date	Sum	Account	Company	Item	Comment
1	1/10/2017	200,000.00	My Bank	Rose, Inc	Revenue	
2	1/10/2017	(50,000.00)	My Bank	Land, Inc	Expenses	
3	1/31/2017	(87,000.00)	My Bank		Payroll	
4	1/31/2017	(13,000.00)	My Bank	Income Taxes	Taxes	
5	1/31/2017	(29,580.00)	My Bank	Salary Taxes	Taxes	
6	2/10/2017	300,000.00	My Bank	Rose, Inc	Revenue	
7	2/10/2017	100,000.00	My Bank	Carnation, Inc	Revenue	
8	2/10/2017	(50,000.00)	My Bank	Water, Inc	Expenses	

Chapter 13. Configuring Stored Procedures

Let's select the **Payments (sp)** object in the **Query List**, a shorter way of **Data Connection Wizard**.



ID	Date	Sum	AccountID	CompanyID	ItemID	Comment
1	42745	200000	1	6	3	
2	42745	-50000	1	5	1	
3	42766	-87000	1		2	
4	42766	-13000	1	4	4	
5	42766	-29580	1	7	4	
6	42776	300000	1	6	3	
7	42776	100000	1	1	3	
8	42776	-50000	1	8	1	

We see the same points as for the view in the previous chapter:

- Foreign key values instead of names in columns and ribbon parameters;
- AccountID, CompanyID, and ItemID in columns and ribbon parameters;
- Unformatted table;
- No predefined table views.

Plus:

1. The ribbon parameters have no value lists.
2. We can't save the changes.

We already know the way to solve the first group issues.

Here is a configuration for the validation lists to replace id to names:

ID	TABLE	TABLE_NAME	COLUMN	EVENT_NAME	HANI	HANDLER	HANDLER_	HANDLER_CODE
26	dbo67	Payments	AccountID	ValidationList	dbo67	Accounts	TABLE	ID,Name
27	dbo67	Payments	CompanyID	ValidationList	dbo67	Companies	TABLE	ID,Name
28	dbo67	Payments	ItemID	ValidationList	dbo67	Items	TABLE	ID,Name
32	dbo67	uspPayments	AccountID	ValidationList	dbo67	Accounts	TABLE	ID,Name
33	dbo67	uspPayments	CompanyID	ValidationList	dbo67	Companies	TABLE	ID,Name
34	dbo67	uspPayments	ItemID	ValidationList	dbo67	Items	TABLE	ID,Name
29	dbo67	viewPayments	AccountID	ValidationList	dbo67	Accounts	TABLE	ID,Name
30	dbo67	viewPayments	CompanyID	ValidationList	dbo67	Companies	TABLE	ID,Name
31	dbo67	viewPayments	ItemID	ValidationList	dbo67	Items	TABLE	ID,Name

Here is a configuration to change database column and parameter names to business ones:

ID	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	LANGUAGE	TRANSLATED_NAME	TRANSLATED_DESC
97	dbo67	Payments	AccountID	en	Account	
98	dbo67	Payments	CompanyID	en	Company	
99	dbo67	Payments	ItemID	en	Item	
103	dbo67	uspPayments	AccountID	en	Account	
104	dbo67	uspPayments	CompanyID	en	Company	
105	dbo67	uspPayments	ItemID	en	Item	
100	dbo67	viewPayments	AccountID	en	Account	
101	dbo67	viewPayments	CompanyID	en	Company	
102	dbo67	viewPayments	ItemID	en	Item	

Configuring ribbon parameters

To configure ribbon parameters, use the **ParameterValues** table of the **payments-configuration** workbook:

ID	TABLE	TABLE_NAME	PARAMETER_NAME	SELECT_SCHEMA	SELECT_NAME	SELECT_TYPE	SELECT_CODE
1	dbo67	uspPayments	AccountID	dbo67	Accounts	TABLE	ID,Name
2	dbo67	uspPayments	CompanyID	dbo67	Companies	TABLE	ID,Name
3	dbo67	uspPayments	ItemID	dbo67	Items	TABLE	ID,Name

The configuration is like the known **EventHandlers** table.

Thus, we use master tables to select ID and Name pairs for the ribbon parameters of the stored procedure.

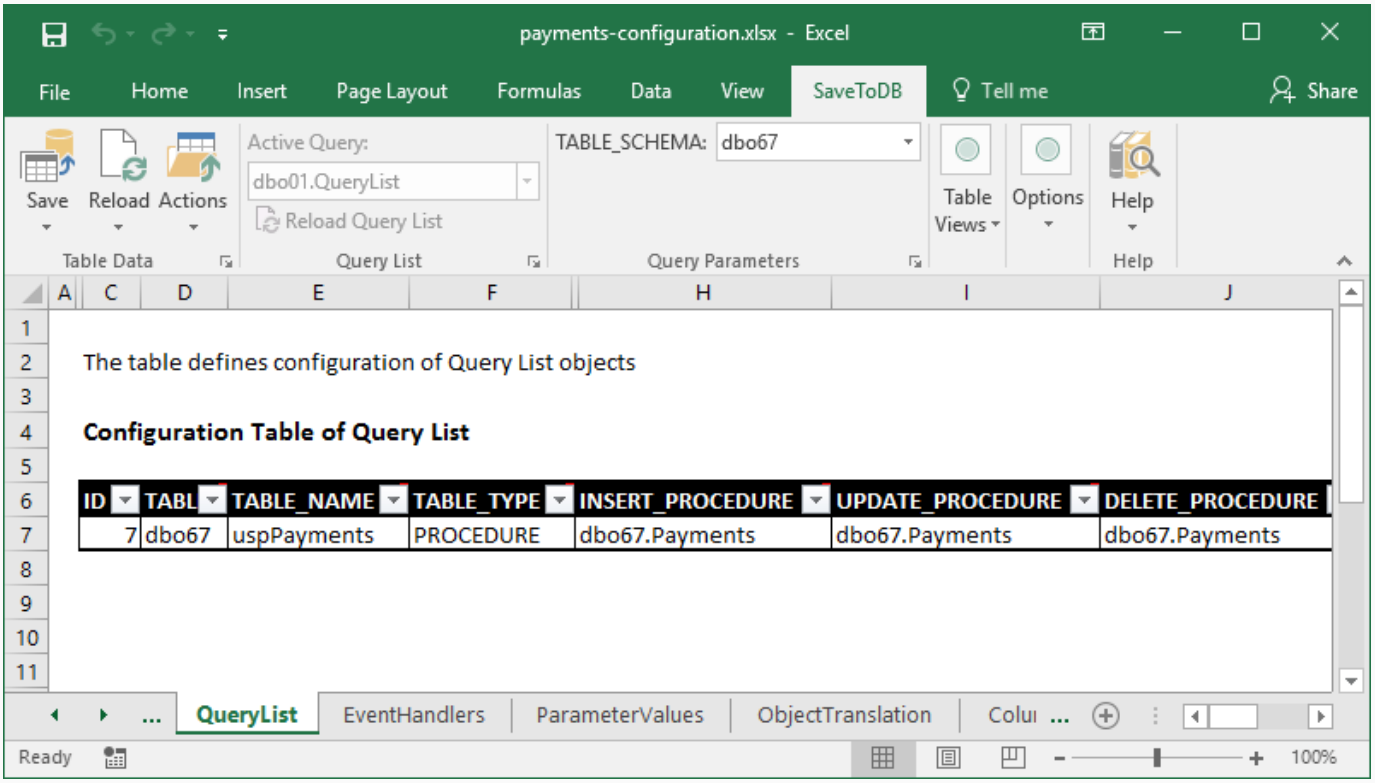
Configuring saving changes

Let’s discuss this in the next important chapter.

Chapter 14. Configuring Saving Changes

Saving changes to a single table

If we need to save changes to a single underlying table, we may specify the target table in the **INSERT_PROCEDURE**, **UPDATE_PROCEDURE**, and **DELETE_PROCEDURE** fields of the **QueryList** table:



In this example, this is a right case as the procedure selects data from a single table.

Saving changes to multiple tables

You may save changes from an Excel table to multiple database tables.

However, this requires using stored procedures or SQL codes.

Let's create such procedures for our example.

Here is the procedure used to insert new rows:

```
CREATE PROCEDURE [dbo67].[uspPayments_insert]
    @Date datetime = NULL
    , @Sum money = NULL
    , @AccountID int = NULL
    , @CompanyID int = NULL
    , @ItemID int = NULL
    , @Comment nvarchar(255) = NULL
AS
BEGIN

SET NOCOUNT ON

INSERT INTO dbo67.Payments
    ( [Date]
    , [Sum]
    , AccountID
    , CompanyID
    , ItemID
    , Comment
    )
VALUES
    ( @Date
    , @Sum
    , @AccountID
    , @CompanyID
    , @ItemID
    , @Comment
    )

END
GO
```

You may see that the procedure has the parameters named as the selected column names.

So, the add-in just calls the procedure passing values from a new row.

You may implement any logic in stored procedures. This procedure just inserts a row into the **Payments** table.

Here is the code of the update procedure:

```
CREATE PROCEDURE [dbo67].[uspPayments_update]
      @ID int
    , @Date datetime = NULL
    , @Sum money = NULL
    , @AccountID int = NULL
    , @CompanyID int = NULL
    , @ItemID int = NULL
    , @Comment nvarchar(255) = NULL
AS
BEGIN

SET NOCOUNT ON

UPDATE dbo67.Payments
SET
    [Date] = @Date
    , [Sum] = @Sum
    , AccountID = @AccountID
    , CompanyID = @CompanyID
    , ItemID = @ItemID
    , Comment = @Comment
WHERE
    ID = @ID

END
GO
```

You may see that the logic is the same. The procedure declares and uses parameters with the column names.

The update procedure has the @ID parameter as the updated row exists. The insert procedure has not.

Here is the code of the delete procedure:

```
CREATE PROCEDURE [dbo67].[uspPayments_delete]
    @ID int
AS
BEGIN

SET NOCOUNT ON

DELETE dbo67.Payments
WHERE
    ID = @ID

END
GO
```

In most cases, delete procedures use primary key column values only.

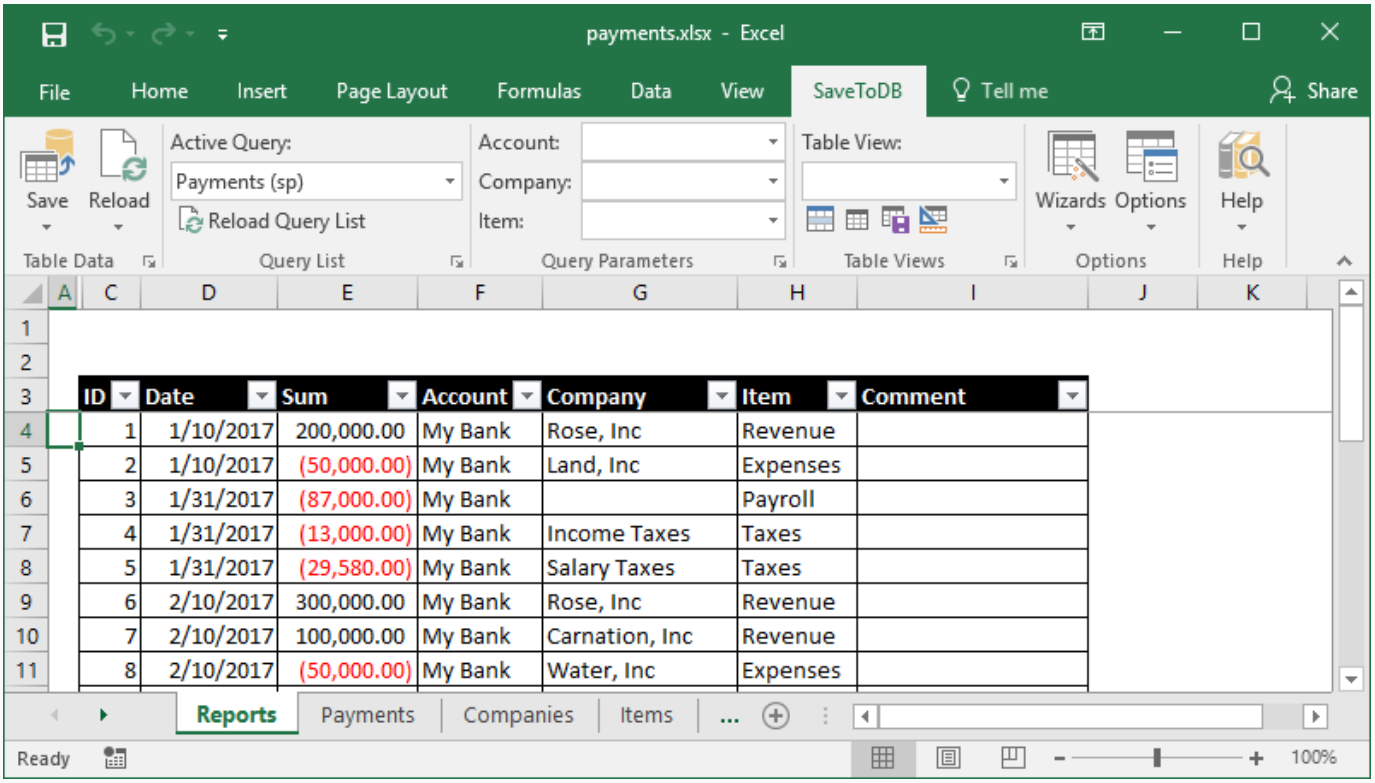
Now, we may replace the configuration created in the previous step

ID	TABL	TABLE_NAME	TABLE_TYPE	INSERT_PROCEDURE	UPDATE_PROCEDURE	DELETE_PROCEDURE	PROCEDURE_TYPE
7	dbo67	uspPayments	PROCEDURE	dbo67.Payments	dbo67.Payments	dbo67.Payments	TABLE

to a new configuration with stored procedures:

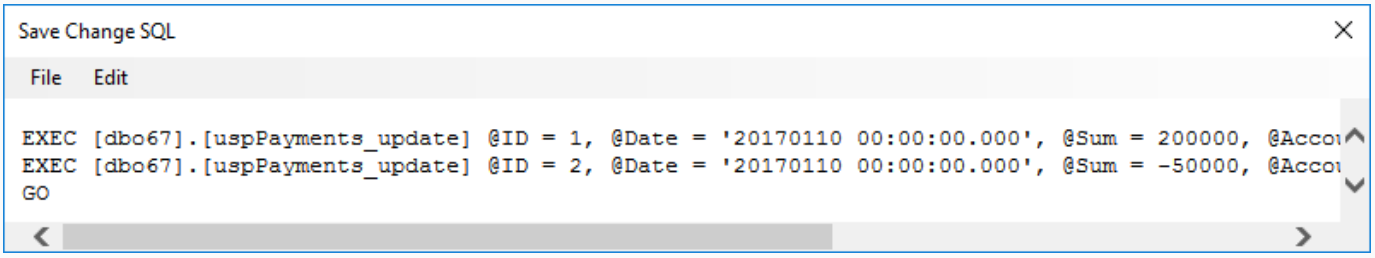
ID	TABL	TABLE_NAME	TABLE_TYPE	INSERT_PROCEDURE	UPDATE_PROCEDURE	DELETE_PROCEDURE	PROCEDURE_TYPE
7	dbo67	uspPayments	PROCEDURE	dbo67.uspPayments_insert	dbo67.uspPayments_update	dbo67.uspPayments_delete	PROCEDURE

After these steps, we have a completely configured stored procedure.



ID	Date	Sum	Account	Company	Item	Comment
1	1/10/2017	200,000.00	My Bank	Rose, Inc	Revenue	
2	1/10/2017	(50,000.00)	My Bank	Land, Inc	Expenses	
3	1/31/2017	(87,000.00)	My Bank		Payroll	
4	1/31/2017	(13,000.00)	My Bank	Income Taxes	Taxes	
5	1/31/2017	(29,580.00)	My Bank	Salary Taxes	Taxes	
6	2/10/2017	300,000.00	My Bank	Rose, Inc	Revenue	
7	2/10/2017	100,000.00	My Bank	Carnation, Inc	Revenue	
8	2/10/2017	(50,000.00)	My Bank	Water, Inc	Expenses	

Let's change a couple of rows and check the generated SQL code using the **Save**, **View Save Change SQL** button.



As we see, the add-in generated EXEC commands.

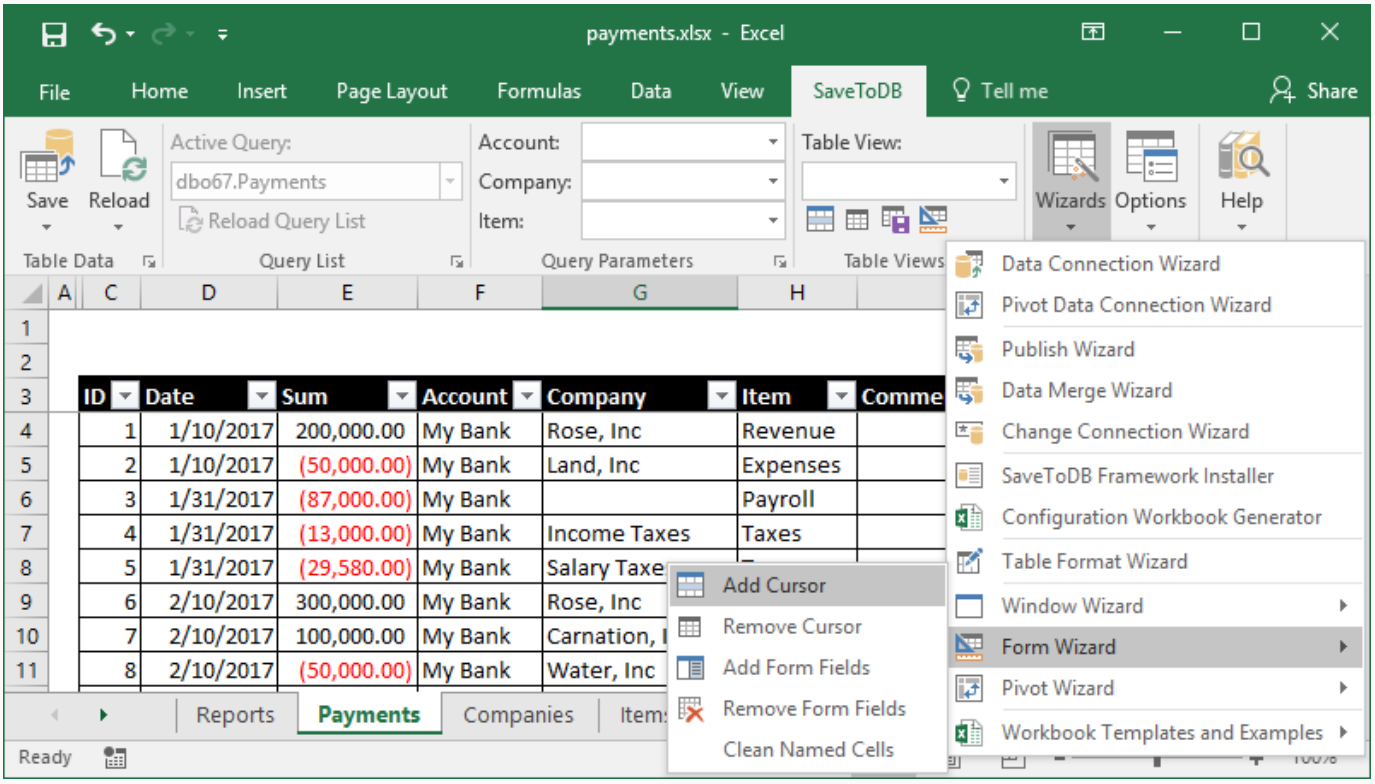
Using stored procedures to save changes is a common way. In this case, we have four procedures like these:

1. dbo67.uspPayments
2. dbo67.uspPayments_insert
3. dbo67.uspPayments_update
4. dbo67.uspPayments_delete

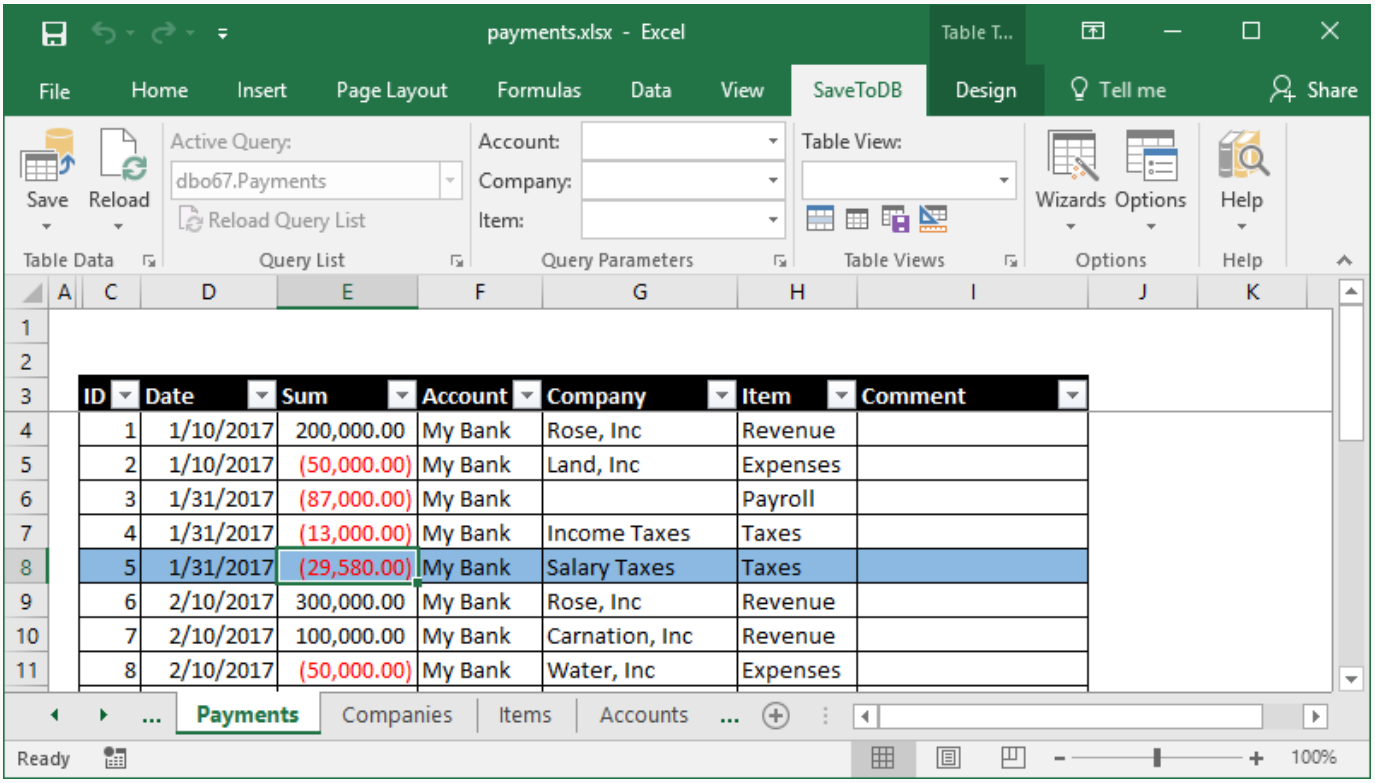
If you use the **_insert**, **_update**, and **_delete** name convention, the add-in links edit procedures automatically, and you may skip adding the configuration to the **QueryList** table.

Chapter 15. Cursors

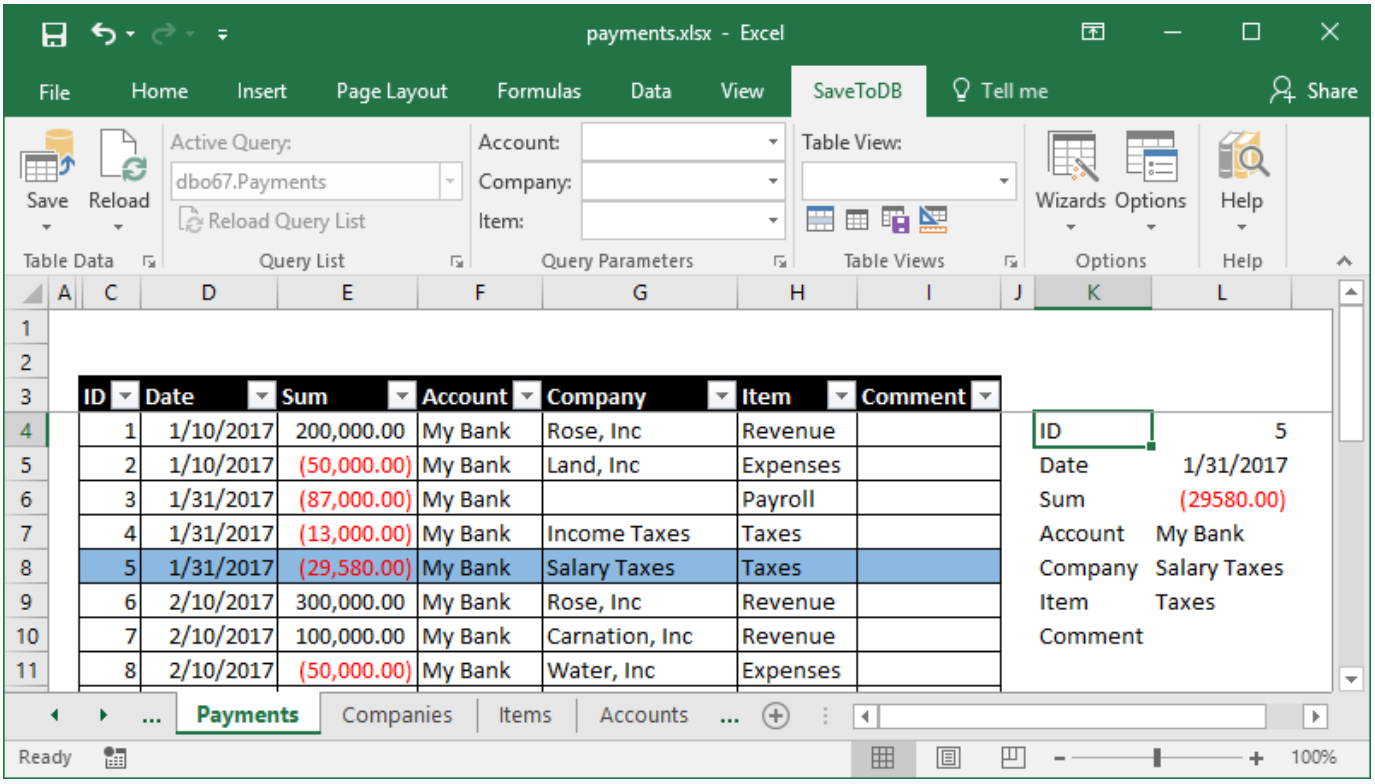
Let's select the **Payments** worksheet and click **Wizards**, **Form Wizard**, **Add Cursor**:



The add-in highlights the active table row:



Let's run the same wizard, click **Add Form Fields** and select cell K4:



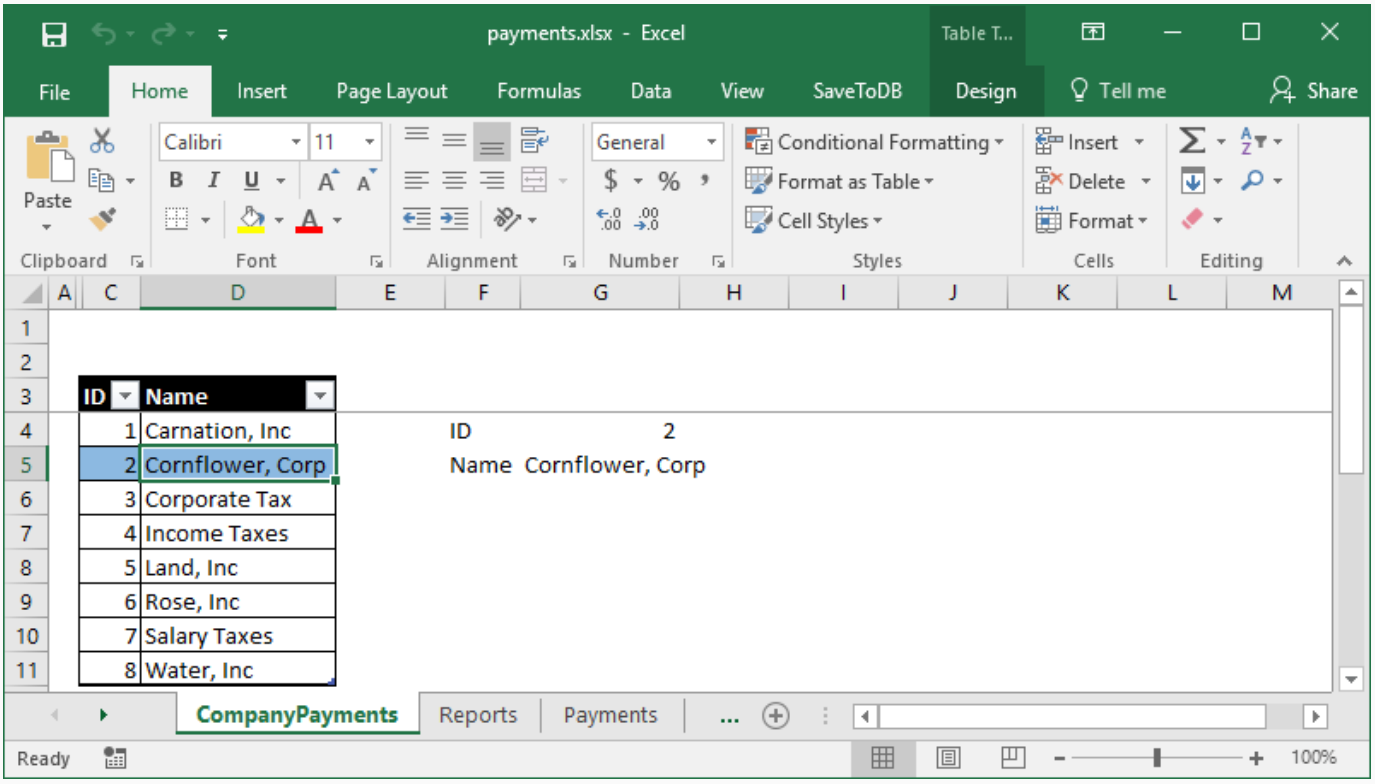
The add-in inserts form fields and updates them when a user changes the active row.

Moreover, you may use such fields to edit table row values.

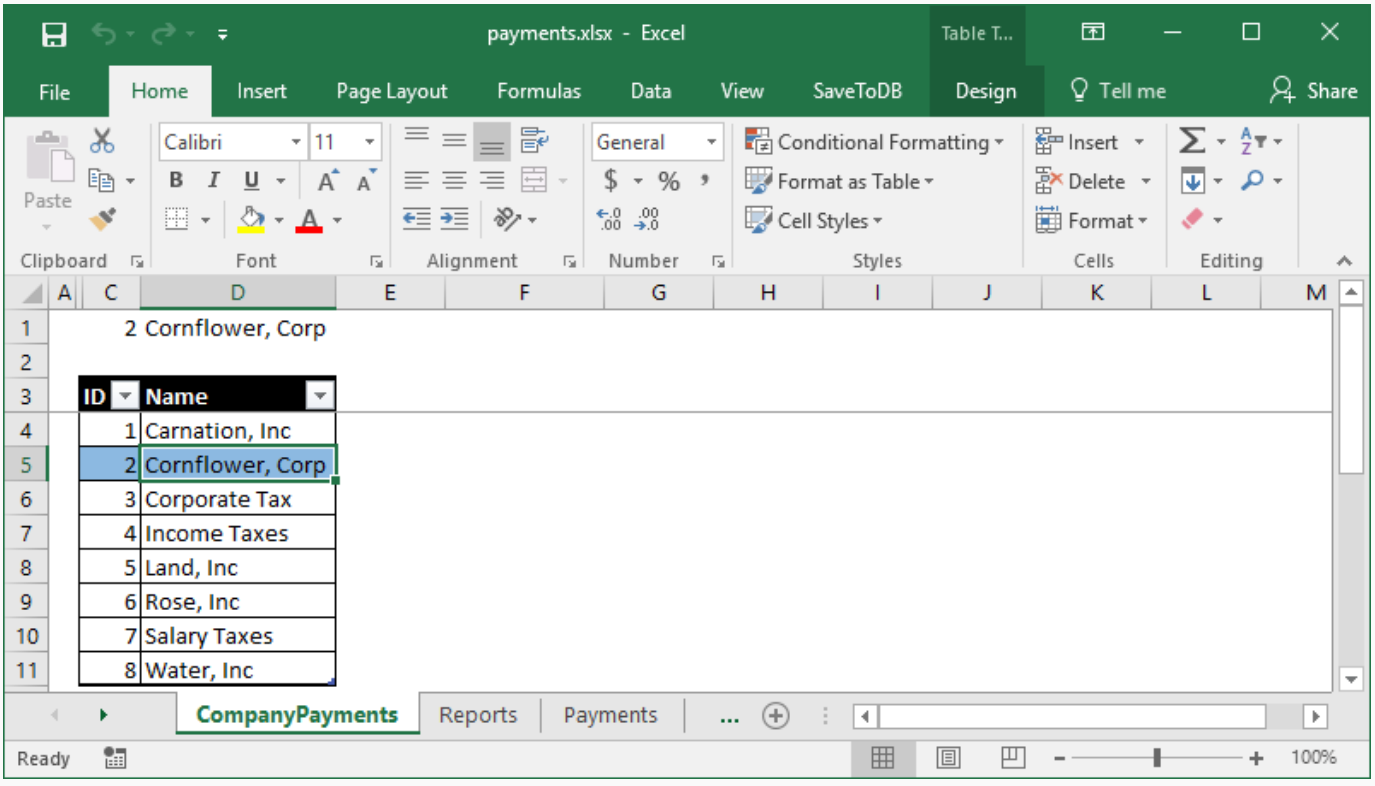
Chapter 17. Master-Details

Let's create a new worksheet, rename it to **CompanyPayments**, and connect to the **Company** table at cell B3.

Then let's add a cursor and add form fields at cell F4.

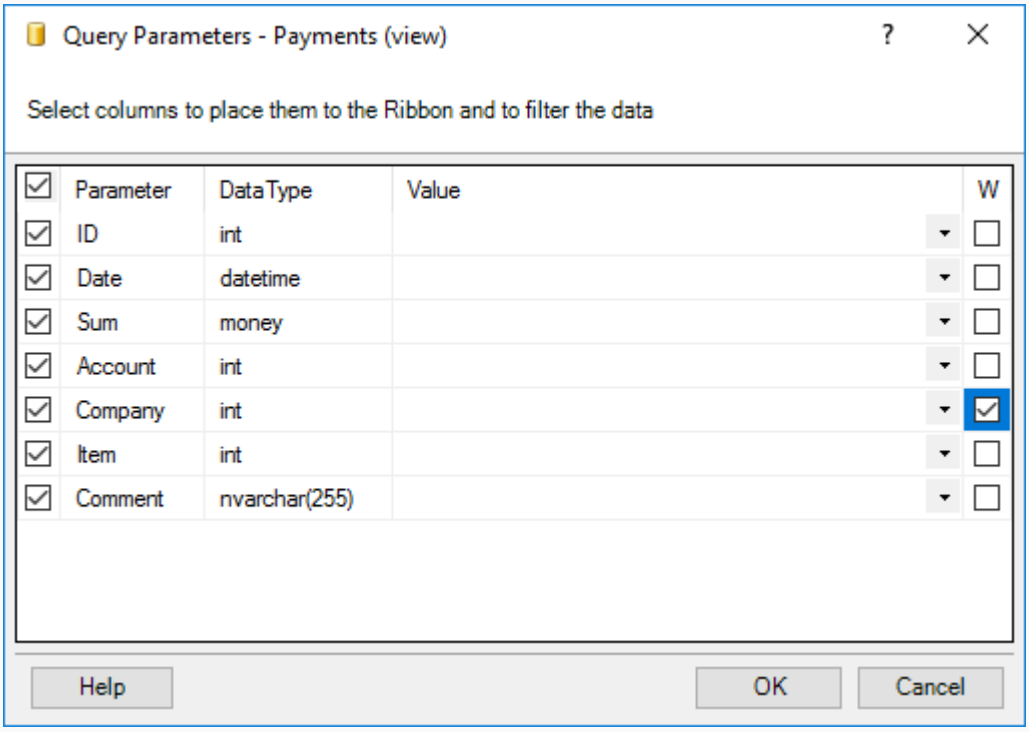


Then let's move the ID and Name fields to cells C1 and D1, and remove labels in column F:

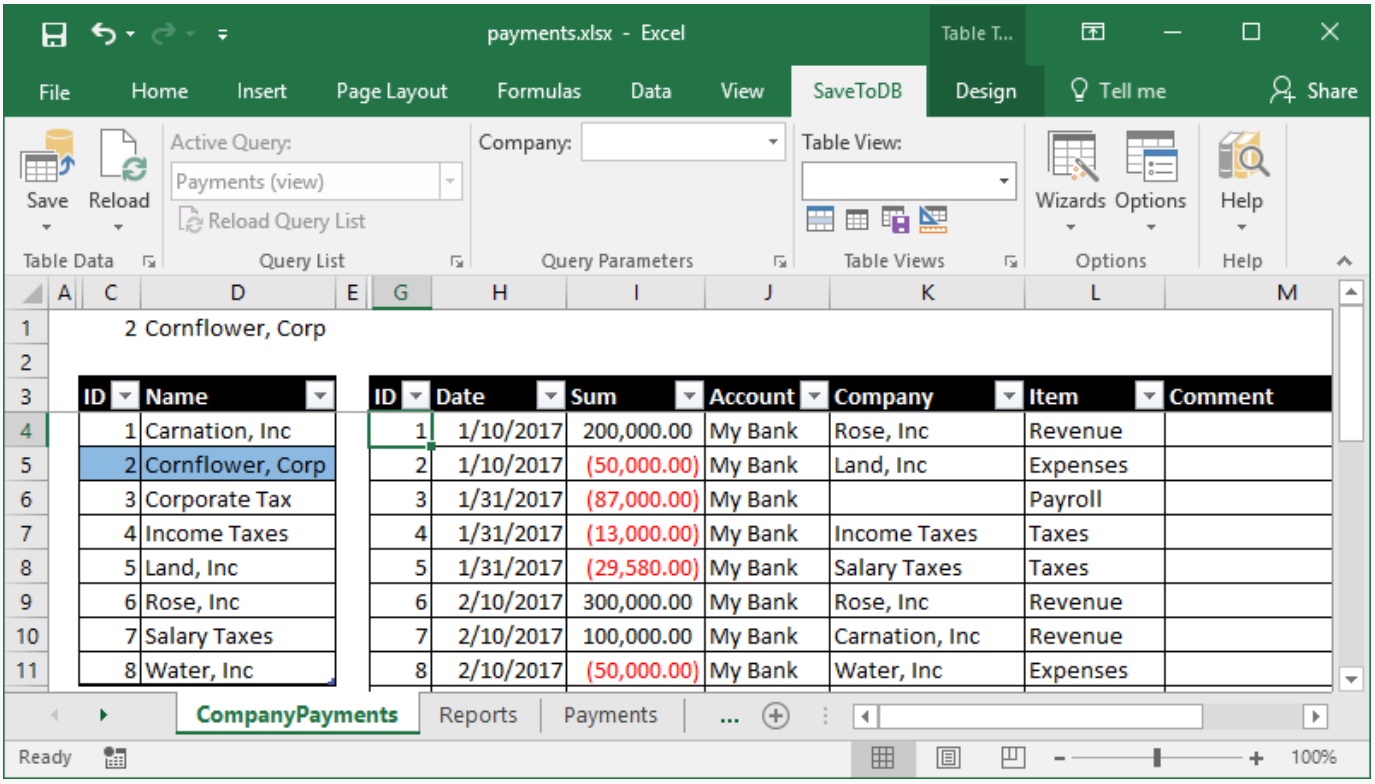


Now, select cell F3 (outside of the active table) and connect to the **viewPayments** view.

In the connection wizard, select only the one **CompanyID** field in the **WHERE** column:



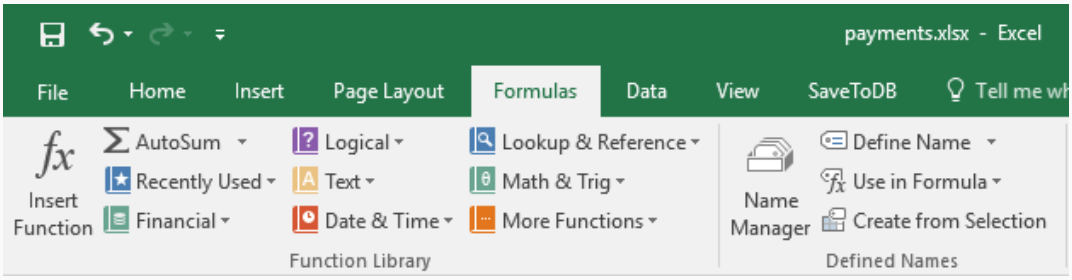
and insert the table at cell F3:



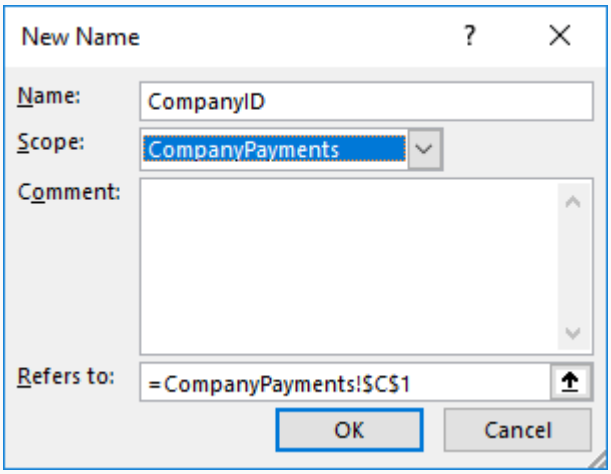
We see that the **viewPayment** view has only one parameter at the ribbon, **Company**.

Also, we see the selected company ID in cell C1 that may be used as a parameter. Let's link them.

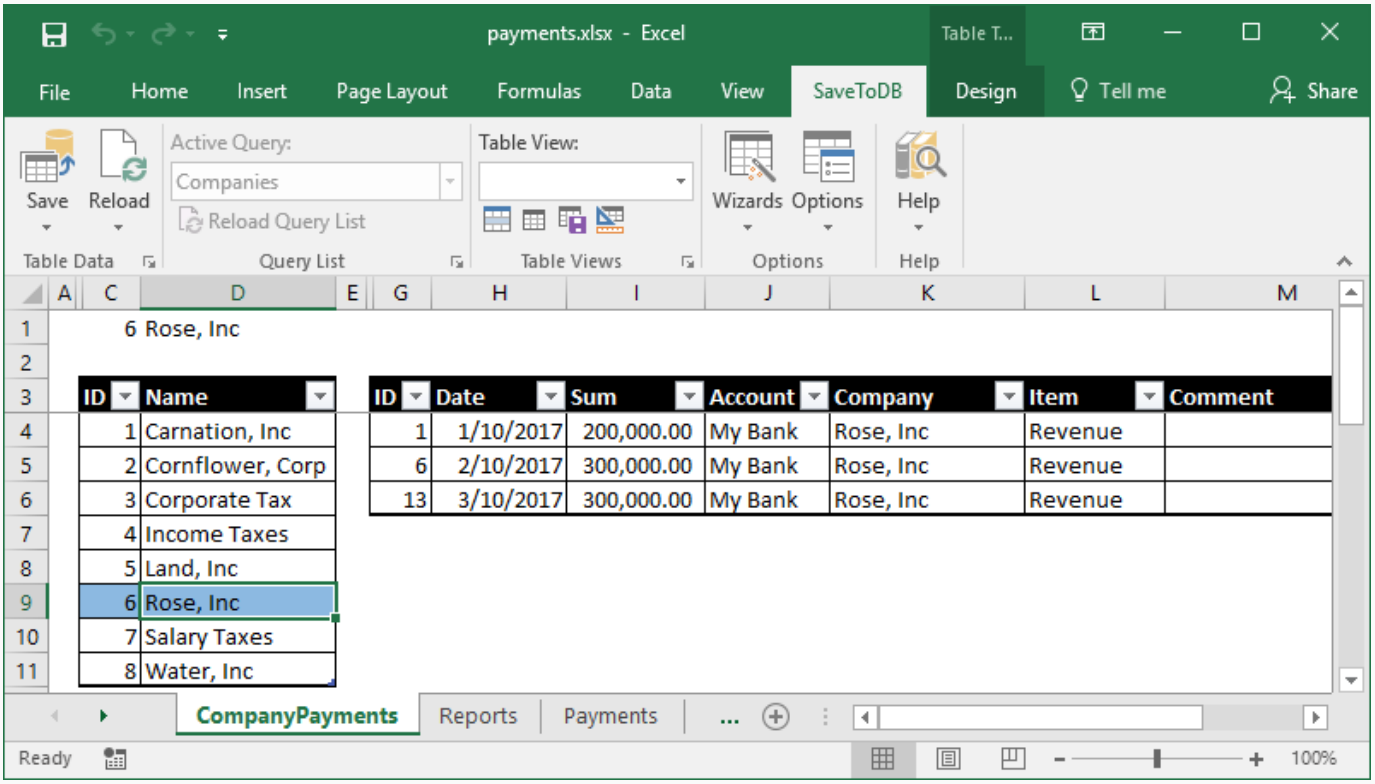
Select cell C1 and click the **Define Name** button in the **Defined Names** group on the **Formulas** tab:



Specify the cell name as the parameter name, **CompanyID**, and select the worksheet name, **CompanyPayments**, in the **Scope** field:



Click **OK**. Then select a row in the Company table, and voilà!



Here is a list of events that implement this behavior:

1. A user selects another row in a master table.
2. The add-in updates form fields (cell C1 with ID).
3. The add-in changes parameters of the details view using named cell values (cell C1 as CompanyID).
4. The add-in reloads the details with new parameter values.

You may build more complex forms using the same technique.

For example, the Northwind example contains the following tables with master-detail relations:

1. Customer first char index (A-Z)
2. Customers
3. Customer orders
4. Orders

Chapter 18. Detail Windows and Task Panes

We may use another technique to show details in windows and task panes using the **SelectionChange** handlers.

Detail Windows

Let's switch to the **payments-configuration** workbook, select the **EventHandlers** worksheet and add the data:

ID	TABLE	TABLE_NAME	COLUMN	EVENT_NAME	HAN	HANDLER_NAME	HANDLER	HANDLER_CODE	TARGET_WORKSHEET
35	dbo67	viewPayments		SelectionChange	dbo67	Company Payments	CODE	SELECT p.[Date]	
36	dbo67	viewPayments		SelectionChange	dbo67	Item Payments	CODE	SELECT p.[Date]	

The **HANDLER_CODE** of the **Company Payments** handler contains the code:

```
SELECT
    p.[Date]
    , p.[Sum]
    , c.Name AS Company
    , i.Name AS Item
    , p.Comment
FROM
    dbo67.Payments p
    LEFT OUTER JOIN dbo67.Items i ON i.ID = p.ItemID
    INNER JOIN dbo67.Companies c ON c.ID = p.CompanyID
WHERE
    p.CompanyID = @CompanyID
```

The **HANDLER_CODE** of the **Item Payments** handler contains the code:

```
SELECT
    p.[Date]
    , p.[Sum]
    , c.Name AS Company
    , i.Name AS Item
    , p.Comment
FROM
    dbo67.Payments p
    INNER JOIN dbo67.Items i ON i.ID = p.ItemID
    LEFT OUTER JOIN dbo67.Companies c ON c.ID = p.CompanyID
WHERE
    p.ItemID = @ItemID
```

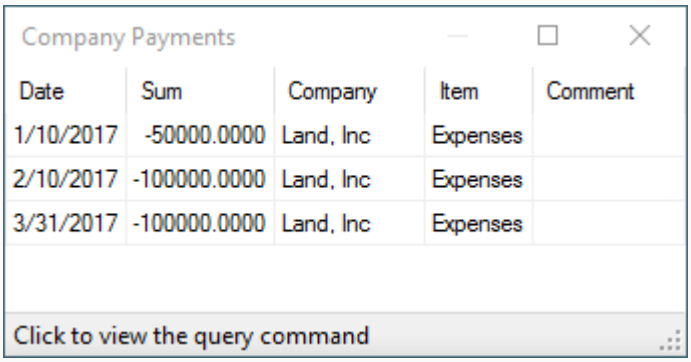
As we may suppose, the add-in executes the specified code on the **SelectionChange** event.

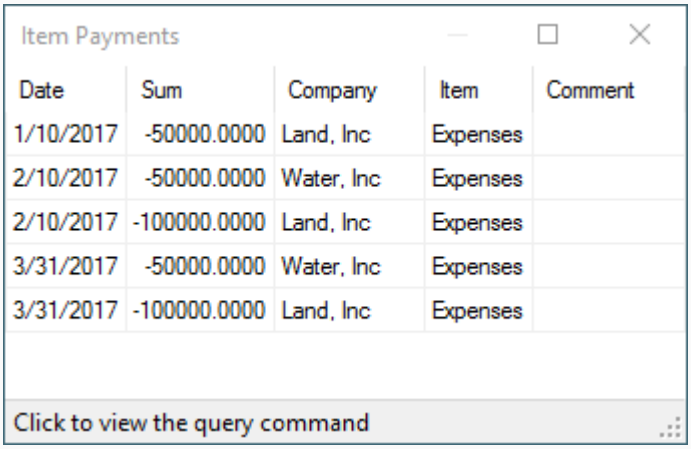
The handlers use @CompanyID and @ItemID parameters accordingly to filter output data.

Let's switch to the **payments** workbook, select the **Reports** worksheet, and select the **Payments (view)** view.

If the view is already activated, click **Reload**, **Reload Data and Configuration**.

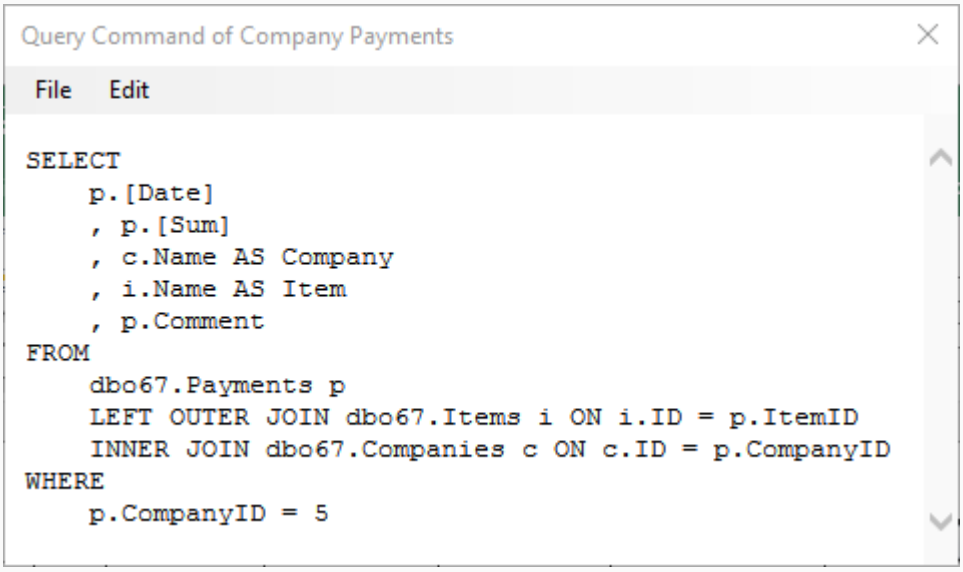
Now, when we change a row, the add-in launches windows like this:





We may select another row. Windows will stay on top and show related information.

You may click on the window status line to see the executed SQL command like this:



In this example, we have used the direct SQL codes stored in the **EventHandlers** table.

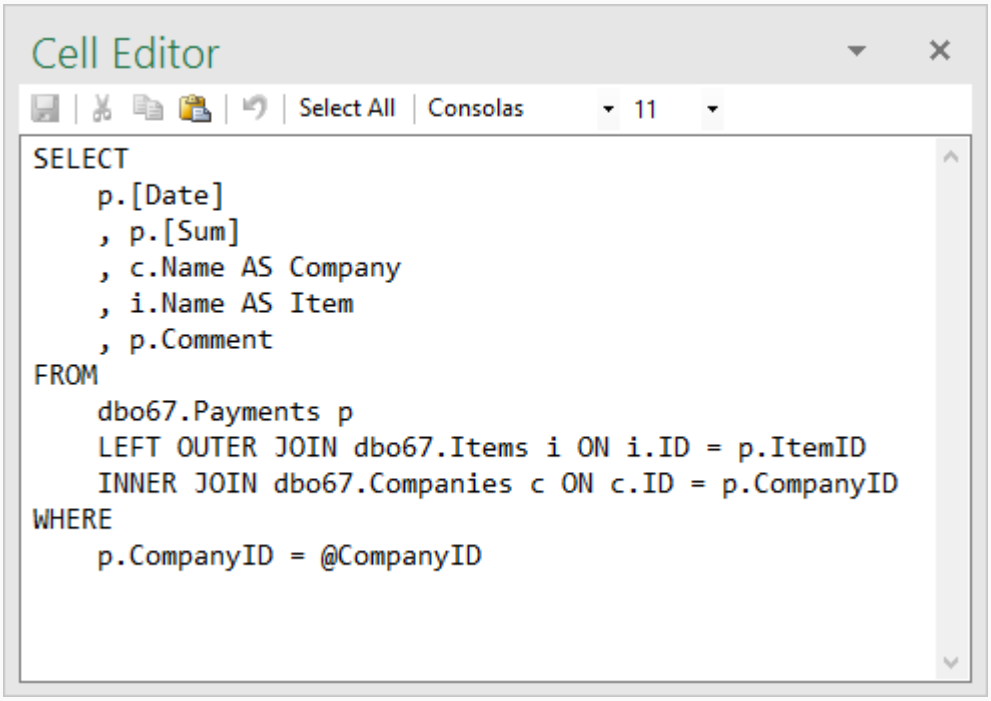
This is useful when you do not want to modify your database.

Otherwise, you may use stored procedures with parameters.

Cell Editor

The SaveToDB add-in adds great support for editing codes in cells.

When the cell contains a multiline value, the add-in launches the **Cell Editor** like this:



You may edit the text in the editor and click the **Save** button to update the underlying cell.

You may turn on/off the editor using the **Options**, **Show Cell Editor Task Pane** option.

Task Panes

Use of detail windows is a good solution when you need windows that are always visible.

If you need context windows, you may use task panes.

Let's add the following handlers to the **EventHandlers** table for the **uspPayments** procedure:

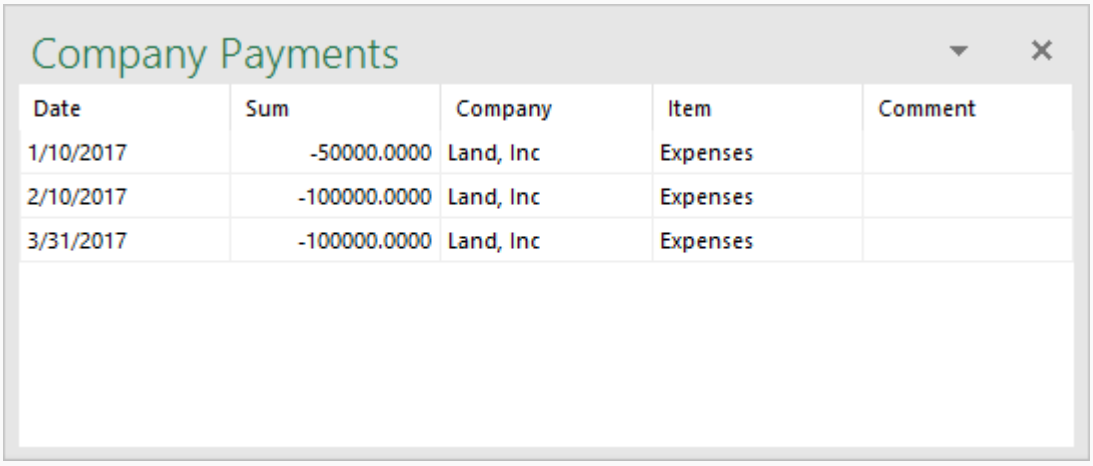
ID	TABLE	TABLE_NAME	COLUMN	EVENT_NAME	HAN	HANDLER_NAME	HANDLER	HANDLER_CODE	TARGET_WORKSHEET
35	dbo67	viewPayments		SelectionChange	dbo67	Company Payments	CODE	SELECT p.[Date]	
36	dbo67	viewPayments		SelectionChange	dbo67	Item Payments	CODE	SELECT p.[Date]	
37	dbo67	uspPayments		SelectionChange	dbo67	Company Payments	CODE	SELECT p.[Date]	_TaskPane
38	dbo67	uspPayments		SelectionChange	dbo67	Item Payments	CODE	SELECT p.[Date]	_TaskPane

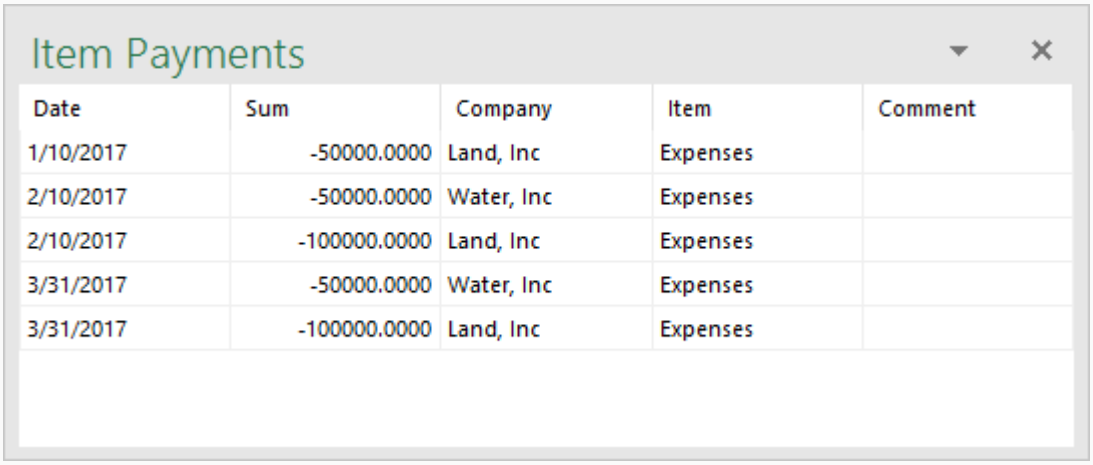
I have copied the view rows, changed the view name, and added **_TaskPane** to the **TARGET_WORKSHEET** field.

The **HANDLER_CODE** codes remain the same.

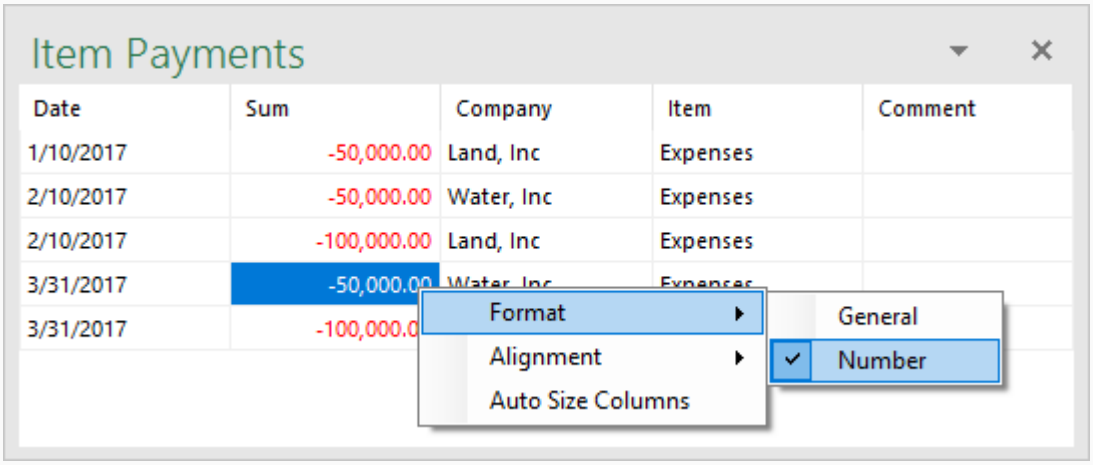
Let's switch to the **payments** workbook, and change the query to the **Payments (sp)** procedure.

When we change a row, the add-in shows task panes like this:





Users may customize column formats:



Also, users may dock task panes and turn them on/off using the **Options**, **Show Task Panes** option.

Chapter 19. Context Menus

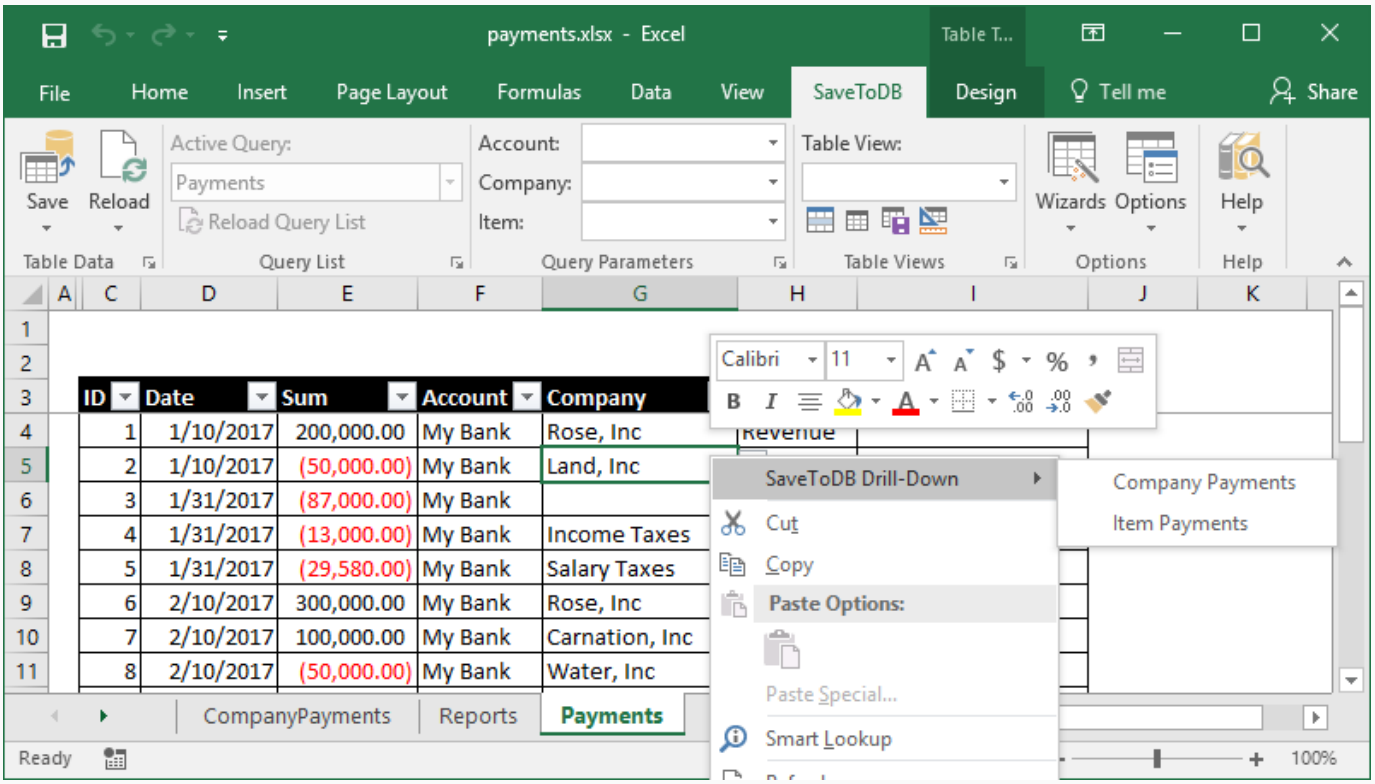
Detail windows and task panes discussed above are shown in the **SelectionChange** event.

We may add such queries and much more to the context menu.

Let's add the following configuration to the **EventHandlers** table (two lines at the bottom):

ID	TABLE	TABLE_NAME	COLUMN	EVENT_NAME	HANI	HANDLER_NAME	HANDLER	HANDLER_CODE	TARGET_WORKSHEET
35	dbo67	viewPayments		SelectionChange	dbo67	Company Payments	CODE	SELECT p.[Date]	
36	dbo67	viewPayments		SelectionChange	dbo67	Item Payments	CODE	SELECT p.[Date]	
37	dbo67	uspPayments		SelectionChange	dbo67	Company Payments	CODE	SELECT p.[Date]	_TaskPane
38	dbo67	uspPayments		SelectionChange	dbo67	Item Payments	CODE	SELECT p.[Date]	_TaskPane
	dbo67	Payments		ContextMenu	dbo67	Company Payments	CODE	SELECT p.[Date]	_TaskPane
	dbo67	Payments		ContextMenu	dbo67	Item Payments	CODE	SELECT p.[Date]	_TaskPane

Then let's switch to the **payments** workbook, select the **Payments** worksheet, click **Reload**, **Reload Data and Configuration**, and right click on a row:



When we click on the **Company Payments** item, the add-in shows the task pane like this:

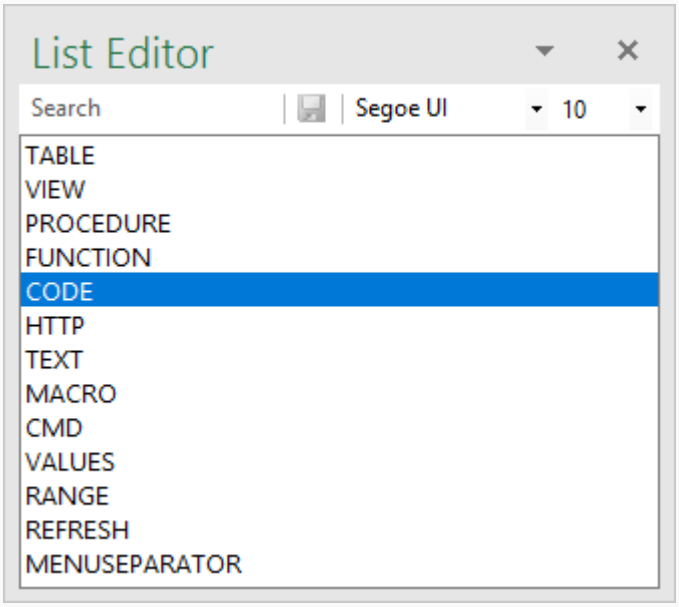
As we learned above, the **SelectionChange** and **ContextMenu** handlers are nearly the same.

They get parameters from the columns of the active row. They differ by the **EVENT_NAME** configuration.

However, the **ContextMenu** handlers support additional configuration parameters like **MENU_ORDER** and **EDIT_PARAMETERS**:

EVENT_NAME	HANI	HANDLER_NAME	HANDLER	HANDLER_CODE	TARGET_WORKSHEET	MENU_ORDER	EDIT_PARAMETERS
ContextMenu	dbo67	Company Payments	CODE	SELECT p.[Date]	_TaskPane		
ContextMenu	dbo67	Item Payments	CODE	SELECT p.[Date]	_TaskPane		

and allow using much more handler types:



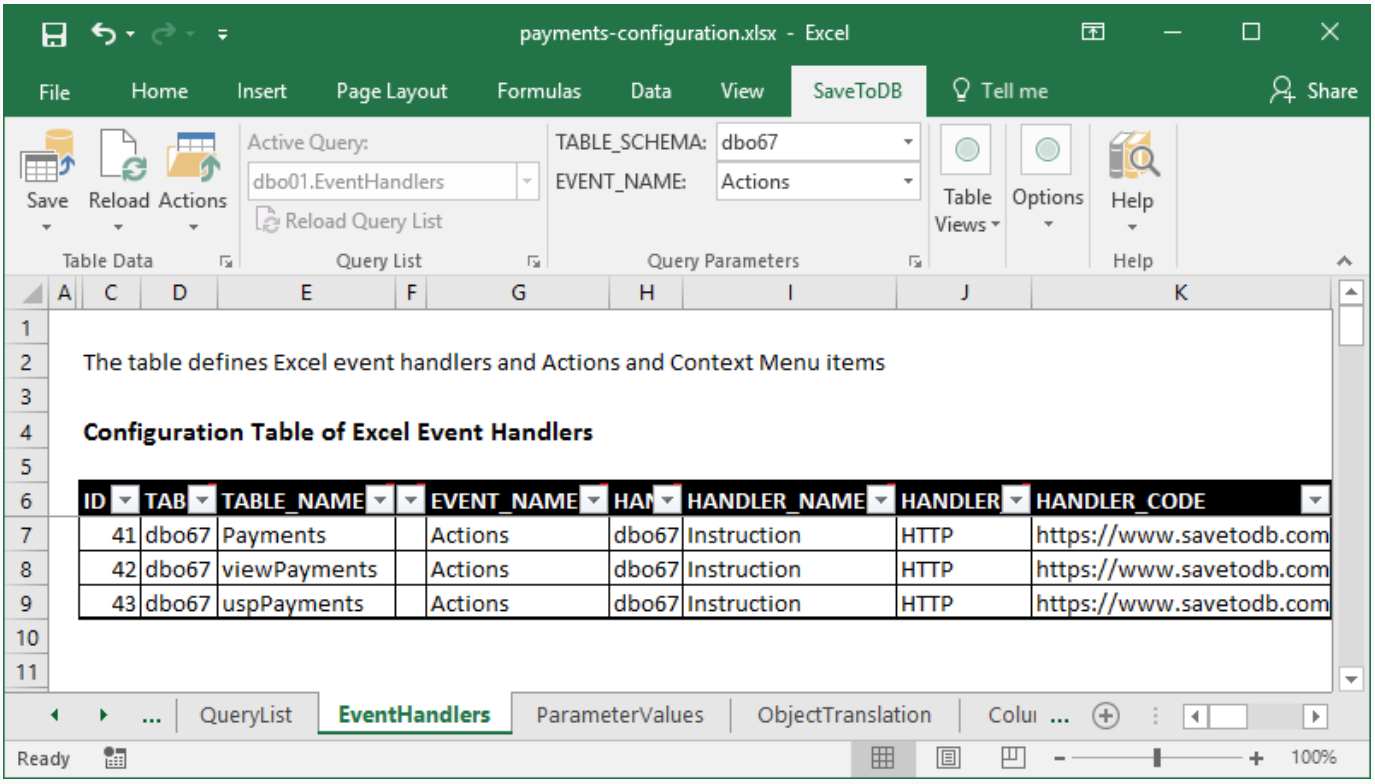
You may configure executing stored procedures, SQL codes, macros, CMD commands, opening URLs.

Use the **MenuSeparator** type to separate menu items.

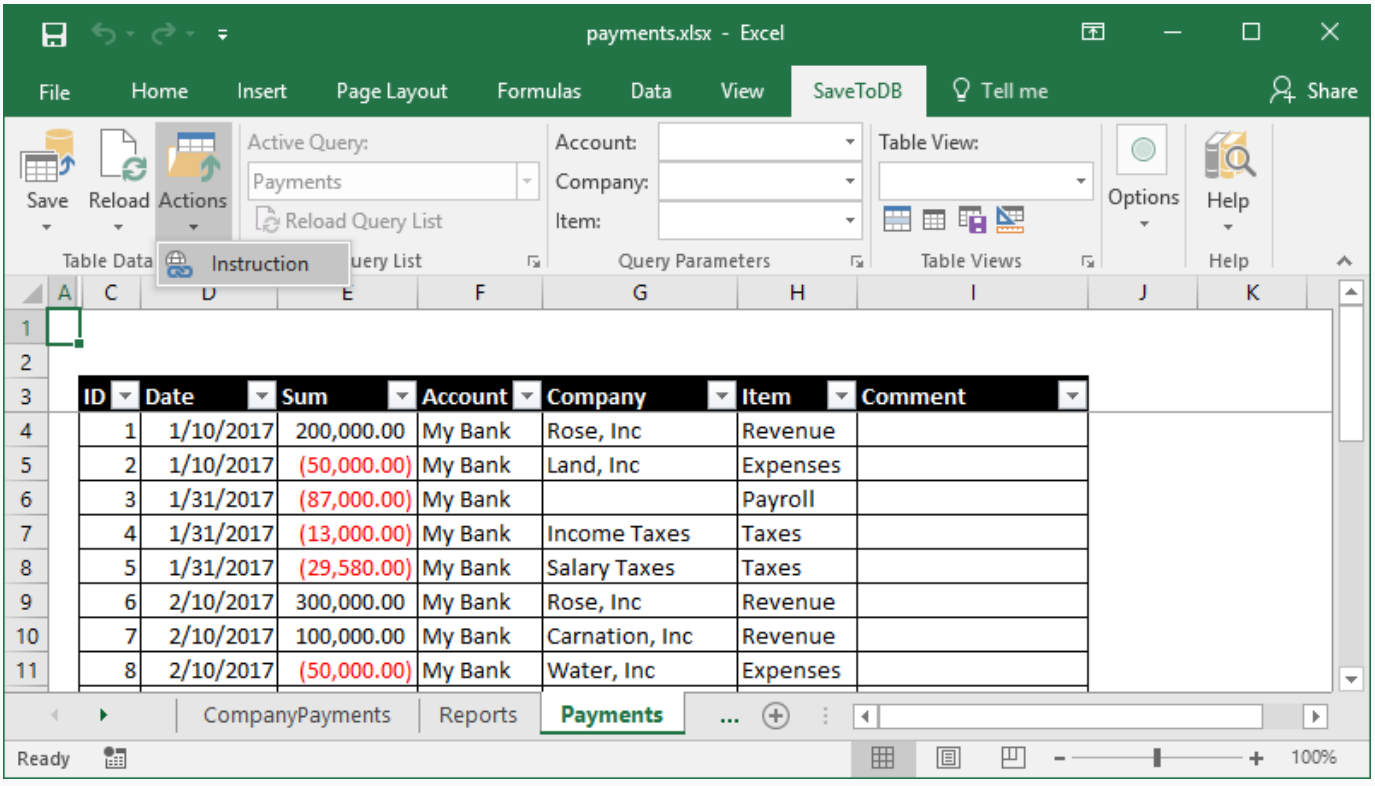
Chapter 20. Actions Menus

The action menus are located at the ribbon and have nearly the same features as context menus.

However, users may execute actions when the active cell is outside of the table. So, they may have no context.



Let's add the **Actions** configuration and reload data and configuration at the **Payments** worksheet:



Conclusion

We have created a complete client application using Microsoft Excel.

Here are ten steps that we made:

1. Connect to tables, views, and stored procedures
2. Configure validation lists
3. Configure ribbon parameters
4. Translate field, parameter, and object names
5. Configure formats, formulas, and table views
6. Configure saving changes
7. Add cursors and form fields
8. Create master-detail forms
9. Configure detail windows and task panes
10. Configure context and action menus

Now you may create such applications yourself.

Creating applications this way has great benefits for you, for users, and for your company:

1. You may create databases and client applications using your database developer skills.
2. You create client applications that have all the features of Microsoft Excel.
3. You may add features step-by-step.
4. You have no headache with deployment and updates.
5. Users are happy as they use Microsoft Excel, the app with the best features and user experience.
6. Users are happy as they use personal, not shared, workbooks and do not depend on others.
7. Your company may solve tasks creating applications with lower risks and lower costs in a shorter time.

This book contains an example database for Microsoft SQL Server.

You may also use Oracle Database, IBM DB2, MySQL, MariaDB, PostgreSQL, NuoDB, or SQLite.

You may download the SaveToDB add-in at www.savetodb.com.

All features described in this book are available in the free SaveToDB Express edition.

I would appreciate your feedback. Feel free to contact me at 10-steps@savetodb.com.

Happy coding!

Sergey Vaselenko

About the Author

My name is Sergey Vaselenko.

I am from Russia, Moscow.

My passion is creating software.

I am a founder and CEO of Gartle Technology Corporation and a leading developer of the SaveToDB add-in.

You are welcome to contact me at

www.facebook.com/sergey.vaselenko

www.linkedin.com/in/vaselenko/

Appendix 1. Database Source Code

You may download the source code at

https://www.savetodb.com/download.php?file=10-steps-for-database-developers.zip

You may download the SaveToDB add-in at www.savetodb.com.

To use all described features, install version SaveToDB 7.2 or higher.

Master Tables

```
CREATE TABLE [dbo67].[Accounts] (
      [ID] int IDENTITY(1,1) NOT NULL
    , [Name] nvarchar(50) NOT NULL
    , CONSTRAINT [PK_Accounts_dbo67] PRIMARY KEY ([ID])
    , CONSTRAINT [IX_Accounts_dbo67] UNIQUE ([Name])
)
GO

CREATE TABLE [dbo67].[Companies] (
      [ID] int IDENTITY(1,1) NOT NULL
    , [Name] nvarchar(50) NOT NULL
    , CONSTRAINT [PK_Companies_dbo67] PRIMARY KEY ([ID])
    , CONSTRAINT [IX_Companies_dbo67] UNIQUE ([Name])
)
GO

CREATE TABLE [dbo67].[Items] (
      [ID] int IDENTITY(1,1) NOT NULL
    , [Name] nvarchar(50) NOT NULL
    , CONSTRAINT [PK_Items_dbo67] PRIMARY KEY ([ID])
    , CONSTRAINT [IX_Items_dbo67] UNIQUE ([Name])
)
GO
```

Add UNIQUE constraints to your tables to avoid name doubles.

Table dbo67.Payments

```
CREATE TABLE [dbo67].[Payments] (
      [ID] int IDENTITY(1,1) NOT NULL
    , [Date] datetime NULL
    , [Sum] money NULL
    , [AccountID] int NULL
    , [CompanyID] int NULL
    , [ItemID] int NULL
    , [Comment] nvarchar(255) NULL
    , CONSTRAINT [PK_Payments_dbo67] PRIMARY KEY ([ID])
)
GO

ALTER TABLE [dbo67].[Payments] ADD CONSTRAINT [FK_Payments_Accounts_dbo67]
FOREIGN KEY ([AccountID]) REFERENCES [dbo67].[Accounts] ([ID]) ON UPDATE CASCADE
GO

ALTER TABLE [dbo67].[Payments] ADD CONSTRAINT [FK_Payments_Companies_dbo67]
FOREIGN KEY ([CompanyID]) REFERENCES [dbo67].[Companies] ([ID]) ON UPDATE
CASCADE
GO

ALTER TABLE [dbo67].[Payments] ADD CONSTRAINT [FK_Payments_Items_dbo67]
FOREIGN KEY ([ItemID]) REFERENCES [dbo67].[Items] ([ID]) ON UPDATE CASCADE
GO
```

View dbo67.viewPayments

```
CREATE VIEW [dbo67].[viewPayments]
AS

SELECT
    p.ID
    , p.[Date]
    , p.[Sum]
    , p.AccountID
    , p.CompanyID
    , p.ItemID
    , p.Comment

FROM
    dbo67.Payments p

GO
```

The view selects data from a single table that makes it updateable.

You may use INSERT, UPDATE, and DELETE statements for such views directly. The add-in uses this feature.

Stored Procedure dbo67.uspPayments

```
CREATE PROCEDURE [dbo67].[uspPayments]
    @AccountID int = NULL
    , @CompanyID int = NULL
    , @ItemID int = NULL
AS
BEGIN

SET NOCOUNT ON

SELECT
    p.ID
    , p.[Date]
    , p.[Sum]
    , p.AccountID
    , p.CompanyID
    , p.ItemID
    , p.Comment

FROM
    dbo67.Payments p
WHERE
    COALESCE(@AccountID, p.AccountID, 0) = COALESCE(p.AccountID, 0)
    AND COALESCE(@CompanyID, p.CompanyID, 0) = COALESCE(p.CompanyID, 0)
    AND COALESCE(@ItemID, p.ItemID, 0) = COALESCE(p.ItemID, 0)

END
GO
```

The stored procedure has parameters used to filter source table data. The NULL value means “all values.”

Use **SET NOCOUNT ON** as the first command. Otherwise, Excel cannot load data from the procedure.