Manual by practice

Practical Course

translated by Yann Yvinct

How to create Windows® application using databases

With simple words
I – INTRODUCTION

1.1 Introduction

Suppose you keep saving something in a classic Excel spreadsheet, as shown in Fig 1.

<table>
<thead>
<tr>
<th>Client name</th>
<th>Phone</th>
<th>Equipment name</th>
<th>Cost per day</th>
<th>Start date</th>
<th>End date</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>KH Services</td>
<td>555-55-55</td>
<td>Bulldozer TM-10</td>
<td>800</td>
<td>5/1/2016</td>
<td>5/3/2016</td>
<td>1600</td>
</tr>
<tr>
<td>euroasfalt</td>
<td>455-66-77</td>
<td>Paver</td>
<td>1200</td>
<td>5/2/2016</td>
<td>5/5/2016</td>
<td>3600</td>
</tr>
<tr>
<td>KH Services</td>
<td>555-55-55</td>
<td>Truck crane K-4561</td>
<td>950</td>
<td>5/2/2016</td>
<td>5/3/2016</td>
<td>950</td>
</tr>
<tr>
<td>Master builder</td>
<td>777-77-77</td>
<td>9.5-ton, Truck, Man</td>
<td>1450</td>
<td>5/6/2016</td>
<td>5/7/2016</td>
<td>1450</td>
</tr>
<tr>
<td>Concrete master</td>
<td>844-44-44</td>
<td>Kenworth W900</td>
<td>1600</td>
<td>5/7/2016</td>
<td>5/9/2016</td>
<td>3200</td>
</tr>
<tr>
<td>euroasfalt</td>
<td>455-66-77</td>
<td>Paver</td>
<td>1200</td>
<td>5/9/2016</td>
<td>5/10/2016</td>
<td>1200</td>
</tr>
<tr>
<td>KH Services</td>
<td>555-55-55</td>
<td>Kenworth W900</td>
<td>1600</td>
<td>5/7/2016</td>
<td>5/13/2016</td>
<td>9600</td>
</tr>
</tbody>
</table>

This would seem sufficient for complete accounting. Unfortunately, this approach has a number of problems that we will now analyze.

Suppose you are an entrepreneur who offers the rental of construction equipment to other companies and keeps the records on a simple board, as shown in Fig. 1.

**Problem 1**

First, what will happen if a company changes its phone number? Let's say that the company "KH Services" has as new telephone number 565-55-56. To keep the table up to date, you need to make changes on three lines (and if there are already 500?).

If we do not, our data will be inconsistent, after which we will not be able to determine which of the phone numbers is up to date.

**Problem 2**

We can make a typo in the company name and accidentally write KJ Service instead of KH Service. Afterwards, it will be difficult to know if it is a new client or if a typo was made. Such an error could not have happened if we had chosen a company from a list.

**Problem 3**

You can not manage to keep records with several people.
For example, your accountant has to manage invoices and payments, and your company's employees want to keep records of orders and meetings with customers, but for example, accounting would not want rental employees to have access to payment information. More serious problems will arise when it comes to analyzing the accumulated data, for example:

- Calculate the income for a given location period.
- Calculate the number of equipment rented by a given company.
- Know what kind of equipment is currently available, etc.

When a lot of information accumulates in your table, such problems will necessarily arise, as well as others, depending on the specificities of your accounting activity.

1.2 Database

Now that we have obvious problems keeping the records in a table, we will understand how to organize everything properly.

To manage a complete accounting, it is necessary to use what is called a "database".

What is a database? You can imagine it yourself in the form of several Excel spreadsheets linked together in a particular way. In addition, each table is created for a specific entity, but you will learn more about it in the next chapter.
1.3 Identifying the entities

As you can see in the table in Figure 1, all the elements are stacked, these are:

- Client  
  (column: "Client Name", "Phone")
- Equipment we own  
  (column: "Equipment name", "Cost per day")
- Rental time  
  (column: "Start Date", "End date", "Cost")

**IMPORTANT!**

The above entities are independent and each entity must have its own table. This process is also called normalization.

As a result, instead of a table, we will have 3, as shown in Figure 2.

![Figure 2](image)

- **Client** table - contains all our customers and their phone numbers.
- **Equipment** Table - contains the names of the equipment we own, as well as their cost for 1 day rental.
- **Rent** Table - contains information about the terms of the rented equipment.

Now, the problems described in chapter 1.1 are solved:

1. When a company changes phone number, simply change it to a single line in the table named Customer
2. It is not necessary to manually write the name of the company each time, thus avoiding typing errors. We simply choose the necessary company from a list.
3. Multiple database users can access only certain tables, for example, only a manager can add a new company as a customer in the Customer table.
1.4 Links between tables and foreign keys

As you can see, we now have three separate tables that are not connected to each other and looking at them, it is unclear what equipment such a company has rented and for how long. To make this obvious, you must add links to the table.

After we add the necessary links, our tables will look like the one shown in Fig. 3.

Looking at Fig. 3, let’s take a closer look at the connections:

Note that now, each table has a column named id. Because of this column, each record of the table has a unique identifier. Let’s call it correctly "Primary Key". When adding a new record to the table, it will automatically be assigned to it.

In the Rent table, you can also notice the appearance of columns id_Client and id_Equipment. These are the "foreign keys".

It is through the primary and foreign keys that the relationships between the tables are formed. Moreover everything is simple, the foreign key id_Client of Rent table points to the primary key of the table Client.
Thus, we know which client to process for each entry in the **Rent** table. To be clearer, take a look at **Fig. 4**.

<table>
<thead>
<tr>
<th>id</th>
<th>Client name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>KH Services</td>
<td>555-55-55</td>
</tr>
<tr>
<td>2</td>
<td>euroasphalt</td>
<td>455-66-77</td>
</tr>
<tr>
<td>3</td>
<td>Master builder</td>
<td>777-77-77</td>
</tr>
<tr>
<td>4</td>
<td>Concrete master</td>
<td>844-44-44</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>Equipment name</th>
<th>Cost per day</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bulldozer TM-10</td>
<td>12000</td>
</tr>
<tr>
<td>2</td>
<td>Paver</td>
<td>8500</td>
</tr>
<tr>
<td>3</td>
<td>Truck crane K-4561</td>
<td>6500</td>
</tr>
<tr>
<td>4</td>
<td>9.5-ton, Truck, Man</td>
<td>14000</td>
</tr>
<tr>
<td>5</td>
<td>Kenworth W900</td>
<td>15500</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>id_Client</th>
<th>id_Equipment</th>
<th>Start date</th>
<th>End date</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5/1/2016</td>
<td>5/3/2016</td>
<td>1600</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>5/2/2016</td>
<td>5/5/2016</td>
<td>3600</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>5/2/2016</td>
<td>5/3/2016</td>
<td>950</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td>5/6/2016</td>
<td>5/7/2016</td>
<td>1450</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>5</td>
<td>5/7/2016</td>
<td>5/9/2016</td>
<td>3200</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1</td>
<td>5/9/2016</td>
<td>5/10/2016</td>
<td>1200</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>5</td>
<td>5/7/2016</td>
<td>5/13/2016</td>
<td>9600</td>
</tr>
</tbody>
</table>

**Figure 4.**

For example, in the **Rent** table, for the record with **ID 6**, the "euroasphalt" client has rented the "Bulldozer TM-10" equipment.
1.5 Types of data

Before we start creating our first database application, we have to consider another topic. These are the data types used in the columns.

At first glance, all the data in the tables seems to be just text, but that is not the case. Each column of the table corresponds to a certain type of data, which we will list:

- **text** (just any text, for example the name or the name of the company)
- **integer** (without fractional part, for example, the quantity of something)
- **floating point number** (for example, the number 3.14)
- **currency** (for example $25.00)
- **boolean** (takes the value of Yes or No)
- **date** (for example 02/06/2016)
- **time** (for example, 19:57:00)
- **date and time** (for example 02/06/2016 19:57:00)
- **file** (allows you to save the file directly into the database)
- **image** (allows you to save a photo in the database)
- **a foreign key** (we talked about it in section 1.3)

Therefore, when you create columns in a table, you must select its type. Why is it necessary? This makes it possible to store data more rationally and provides the database with higher performance. It also helps to avoid mistakes. For example, if a column is of integer type, the user will not be able to save text.
2. Put into practice.

2.1 Let's create our first application working with a database.

Now we know enough database theory to create your own application working with it. A more experienced reader could talk about the SQL quest language that has not been addressed yet, but fortunately modern development tools allow us to do it for the first time.

But if you decide to do more ambitious work on databases, you will probably have to familiarize yourself with SQL.

For our practical work, I chose to use a fairly simple database development environment, in which everything is brought together to immediately start creating a database structure and the application that will work with it.

Visit [http://myvisualdatabase.com/](http://myvisualdatabase.com/) to get the latest version of My Visual Database. The program, unfortunately, is not free, but it has a trial period, which will suffice for educational purposes. But you can find a previous free version via this link: [http://myvisualdatabase.com/download/myvisualdb1.44.exe](http://myvisualdatabase.com/download/myvisualdb1.44.exe)

Let’s look briefly at the possibilities of the program:

- Creating the database structure
- Creating forms to work with the database
- Reporting system for printing and analysis
- A scripting language that allows to implement any non-standard functionality
- Supports SQLite and MySQL DBMS
- Ability to create simple Web access into the database
- Create a stand-alone Windows application

Once installed, My Visual Database is immediately ready for use, no additional configuration is required (Fig.5)
Now we can start creating our first application. As an example, we will create the building equipment accounts considered above.
Before you start creating an application, save the project in a separate folder (menu: File> Save as ...), call our project e.g. Rent (Fig.6)

![Save As dialog box](image)

Figure 6. - Save the project

As we saw in the first chapter of the book, the database is made up of tables and columns, which we now have to create.

Let's start by creating tables:
- Client
- Equipment
- Rent

On the Database Tables tab, click the [New Table] button to create our first Client table. In the same way, we will create the Equipment and Rent tables
After creating the tables, we should get this, as shown in Fig. 7.

![Figure 7](image)

Again, pay attention to Fig. 7: for each table created above there is a [New field] button. This button is used to create the fields of the tables. Let's create the first field. Click the [New field] button in the Client table, then select the TEXT type from the menu that appears and enter the name of the Name field. In the same way, create all the following fields for the remaining tables. Do not forget to select the right type of field.

**Client table:**
- Name (client name, type TEXT)
- Phone (client's phone, type TEXT)

**Equipment Table**
- Tech (name techniques, type TEXT)
- CostPerDay (price for 1 day, type CURRENCY)

**Rent Table**
- DateStart (lease start date, type DATE)
- DateEnd (end date of lease, type DATE)
- TotalCost (total cost, type of CURRENCY)
The result of your work should be as shown in Fig. 8.

You may remember that each table must contain a field named id, which stores a unique identifier for each record. You do not need to worry about it, the program creates it automatically, but it does not display this field in the tables.

The structure of the database is almost ready, it only remains to create two external keys in the Rent table that will be associated with the Client and Equipment tables.

This is necessary so that when we create a new rental record, we can select a client and equipment from the list and this option will be saved in foreign keys.

Creating foreign keys is almost similar to creating fields. In the Rent table, click [New field], and from the menu that appears, select [Relationship], then select Client from the list and click OK, as shown in Fig. 9.
Do the same to create a foreign key in the **Equipment** table.

Again, in the **Rent** table, click the [New field] button and select the "**Relationship**" item from the menu that appears, and then select the **Equipment** table from the list and click "**OK**".
The final form of the tables and fields is shown in Fig. 10.

![Figure 10]

At this point, the structures of our database are fully ready.

We then need to create a user interface to interact with the database so that we can create, edit, and delete records. It is also necessary for the application to easily search for records according to different criteria.

From now on, you can launch your application by clicking the button on the toolbar. You will see the Form1 window, on which there is nothing.

After the first launch of the application, in the project folder, you will find an exe file, which becomes an independent application, ready to run on any computer, without installing additional components.
Let's define what forms we need:

1. **Form1**
   Form to search. The same form will be the main one; it will appear exactly when you launch your application.

2. **frmRent**
   Form for creating and editing a rental record. This form will write data to the DB Rent.

3. **frmClientList**
   Form with a list of customers.

4. **frmClient**
   Form for creating and editing a client. This form will write data to the Client DB.

5. **frmTechList**
   Form with a list of equipment.

6. **frmTech**
   Form for creating and editing information of equipment. This form will write data to the Equipment DB.

Typically, for each database table, you need two forms, one to view all the records, and one to create or edit the records in that table.

To create an interface, you have the components you see on the toolbar (**Fig. 11**).

![Components](image)

**Figure 11.**

Almost all components are standard and are typically used in most Windows applications. But we will review them by giving a brief description:

<table>
<thead>
<tr>
<th><strong>Component</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Arrow</strong></td>
<td>Cancels the selection of any selected component.</td>
</tr>
<tr>
<td><strong>Label</strong></td>
<td>Allows you to place a label on the form. In general, the component has no functionality.</td>
</tr>
<tr>
<td><strong>Button</strong></td>
<td>An important and frequently used component. For example, it can be used to save a record in a database.</td>
</tr>
<tr>
<td><strong>TextBox</strong></td>
<td>Serves for entering numeric and text information.</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Multi-line text field (Memo)</td>
<td>Also serves for entering text information.</td>
</tr>
<tr>
<td>Drop-down list (ComboBox)</td>
<td>Allows you to select a value from the list.</td>
</tr>
<tr>
<td>CheckBox</td>
<td>Has Two or Three statuses, is Checked, Unchecked, or Grayed.</td>
</tr>
<tr>
<td>DateTimePicker</td>
<td>To enter the date or time.</td>
</tr>
<tr>
<td>DBImage</td>
<td>Allows you to save an image in a database.</td>
</tr>
<tr>
<td>DBFile</td>
<td>Allows you to save a file to the database.</td>
</tr>
<tr>
<td>Calendar</td>
<td>Serves exclusively for selecting the date.</td>
</tr>
<tr>
<td>TableGrid</td>
<td>View database records as a tabular view.</td>
</tr>
<tr>
<td>Counter</td>
<td>Assigns a unique number to the recording.</td>
</tr>
<tr>
<td>Panel</td>
<td>Decorative interface element</td>
</tr>
<tr>
<td>GroupBox</td>
<td>Decorative interface element</td>
</tr>
<tr>
<td>PageControl</td>
<td>Allows you to create switchable tabs with the possibility of placing other components on each tab</td>
</tr>
<tr>
<td>Picture</td>
<td>Allows placing images on the form.</td>
</tr>
<tr>
<td>Google Maps</td>
<td>Displays a Google Map in your app and markers.</td>
</tr>
</tbody>
</table>

Let's start by creating the first form that will be used to search for records. On this same form, we will place buttons that will call another form to create or modify a record.

To create this form, the following components will suffice:

Click the Form1 tab as shown in Fig. 12.

![Form1 tab](image)
After selecting the desired component from the toolbar, left-click on the Form1 form where you want to place this component. Once the component is placed on the form, if you need to edit the text, you can do this by using the Caption property in the "Component Properties" panel (Fig. 13).
Place all the components on the Form1 form as shown in Fig. 14.

For your convenience, the colored lines indicate where and what components should be placed on the form.

Figure 14.

The search form is ready. On this form, you can find all the records relating to the rental of a specific customer, or specific equipment and also find records according to the start or end date of the rent.

Call the form to create a new rent record, modify an existing rent, or delete a rent.

On this same form, we find the buttons [Clients] and [Equipment]. These buttons are required to call the form displaying respectively a list of all the clients and all the equipment.

Out of curiosity, you can click on the button to launch your project and see what it will look like. Because the form components are not yet configured, nothing will happen when you click on it.

Let’s start by creating the second form, necessary to create or modify the registration of a rental.

I remind you, as a general rule, the same form is used, both for the creation and for the modification of the record.
Create a second form by pressing the button \[\text{Add}\]. Give the form as name: \text{frmRent}.

Place the components on the \text{frmRent} form, as shown in Fig. 15.
Create a third form named **frmClientList** by clicking the button.

On the created form, we will be able to see all our customers, we will also have the possibility to call a form to create a new customer or modify an existing customer.

Place the components on this form as shown in Fig. 16.
Create a fourth form named **frmClient** by clicking the button ![button](image).

This form is designed to create a new customer record or edit an existing one.

Place the components on this form as shown in **Fig. 17**.
Create a fifth form named **frmTechList** by clicking the button.

On the created form, you will be able to see all the materials you own. You will also be able to call a form to add new material, or edit information on existing material.

Place the components on this form as shown in Fig. 18.
Create the sixth and last form of our project, named **frmTech**, by clicking on the button \[button\].

This form is designed to enter new equipment or edit an existing one.

Place the components on this form as shown in **Fig. 19**.

![Picture of frmTech form](image)

*Figure 19.*

If you launch the project by clicking the button \[play\] and click on the buttons on the form, you will notice that nothing is happening.

This is because the components on our created forms are not yet configured and no action is assigned to the buttons. Therefore, we will proceed to the final step of creating an application, by **configuring the components on the form**.

Typically, component configuration is reduced to assigning actions to buttons \[button\] and components for entering information (such as \[caption\], \[edit\], etc.) indicate which **database table and which field they belong to**.
Let’s start with the buttons. You must define the button action via the "Object inspector" panel, using the **Action** property (Fig. 20).

Let’s list the available actions for the buttons we will use to customize our project:

- **Search** - searches for records in the database.
- **New record** - calls a form to create a new record.
- **Save record** - saves the record in the database and closes the form (if this option is checked).
- **Show record** - calls the form and fills it with the record data from the database.
- **Delete record** - deletes the record selected in the table component.
- **Show form** - displays a form without further action.
- **Close form** - close the form.
Customizing the form "Form1":

The button “New rent” (Form1)

This button is intended to create a new record, so we select the New Record action for this button. This action will call the form from which the new record will be created. For this we have a form named frmRent. Set this button as shown in Fig. 21.

Sometimes, instead of choosing the "New record" action, users choose the "Show form" action. This is a mistake, not to be confused!
The button “Edit” (Form1)

It is intended to modify the record that the user will have selected from the TableGrid component. Choose the action "Show record". To configure this action, you must specify the form that will be used to modify the record and the TableGrid component, in which the user chooses the record required for editing. Set this button as shown in Fig. 22.
The button “Delete Record” (Form1)

It is designed to remove the user-selected record in the TableGrid component. Configure this button as shown in Fig. 23.

![Action for the button]

*Figure 23.*
The button “Search” (Form1)

It is designed to search for records according to arbitrary criteria. The search criteria are entered into components such as . Each component can serve as a criterion for searching a specific field in the database, for example, search only by company name or phone number.

Depending on the type of field in the database, you must use the appropriate component. You can see below which types of database fields can be used by the search component.

- **Text field (TextBox).** DB field type: “TEXT” “INTEGER” “CURRENCY”
- **ComboBox.** Search only foreign keys
- **Checkbox.** Search only for field with type "YES / NO"
- **DateTimePicker.** Search by fields with types "DATE / TIME" "DATE" "TIME"
- **TableGrid.** Searches only for foreign keys. Although the component is usually used to display the search results, it can be used with foreign key lookup criteria because in fact it is similar to component ComboBox.
- **Counter.** Search only for the field with the type "Counter"

Now configure this button. To do this, select the "Search" action. The button configuration consists of four steps:

1. **Select the components that will be used as search criteria for our records.** In our case, choose the following components: ComboBox1 (Client), ComboBox2 (Equipment), DateTimePicker1 (DateStart), DateTimePicker2 (DateEnd).
2. **Select the table in which we are going to search.** Since we are going to look for rental data, we select the DB "Rent" table.
3. **Select the fields from the table that you want to display as a result of the search in the TableGrid component.** In our case, choose the following fields: Rent.DateStart, Rent.DateEnd, Rent.TotalCost, Client.Name, Equipment.Tech.
4. **Select the TableGrid component that will be displayed the result of the research.**

With a button launching the action "Search", if you want to display all the records, just click on it with the empty search criterion.
Fig. 24 shows what the [Search] button setting should look like.
The button “Clients” (Form1)

It is designed to display the form on which we will be able to see all our customers.

We select the action "Show form" and also select the form “frmClientList”, which will be displayed by clicking on this button (Fig. 25).

![Action for the button](image)

Figure 25.

The button “Equipment” (Form1)

It is designed to show the shape on which we will be able to see all the materials that we possess. We select the action "Show form" and also select the form "frmTechList", which will be displayed by clicking on this button (Fig. 26).

![Action for the button](image)

Figure 26.
This completes the configuration of the buttons on the form. It remains to configure the ComboBox \(\text{ComboBox}\) and DateTimePicker \(\text{DateTimePicker}\) components, which are also present on Form1.

The configuration of such components consists, as a rule, in specifying to which specific field of the database they belong, for that the present components, they are:

**TableName** and **FieldName** properties, the exception is only a ComboBox \(\text{ComboBox}\) component, instead of the **TableName** property, the **ForeignKey** property will be used, which results in a foreign key. Thus, the "Search" button will know which field of the database to search for if the user enters the search criteria in the component. We proceed to the configuration of the components.

We start with the DateTimePicker1 component (“Start rent”), the location is contained in the Rent table in the DateStart field, so in the **TableName** property, we select the Rent DB table, and in the **FieldName** property the field in this DateStart table, as shown in Fig 27.

![Figure 27](image)

For the DateTimePicker2 (End rent) component, do the same, but in the **FieldName** property select the DateEnd field.
The configuration of the ComboBox components, as mentioned above, differs in that the **ForeignKey** property is used instead of the **TableName** property. This component is linked to the foreign key of the table, so that the component displays the entries of the table referenced by the foreign key, allowing the user to select an entry from this table.

Maybe that sounds confusing, so let's look at an example: let's set up the **ComboBox1** (Client).

The search button "**Search**" is configured so that it looks in the **Rent** table. In this table, there is a **Rent.id_Client** foreign key, which we must select in the **ForeignKey** property of the **ComboBox1** (Client) component. The foreign key **id_client**, as its name indicates, refers to the **Client** table. From the **ComboBox1** component, the user can select from the list of clients.

It will remain to select the field of the database in the property **FieldName**, in our case it will be the field **Name**, therefore the list of the names of customers will be visible in the component (see **Fig. 28**).
In the same way, we configure the ComboBox2 (Equipment) component, in the **ForeignKey** property we select the *Rent.id_Equipment* foreign key and in the **FieldName** property we select the *Tech* DB field.

While we have configured a single form, but without taking into account the form and its purpose, the button and component settings are rather monotonous. For the buttons, you must select the correct action and, for the components, select the field of the database to which they belong using the properties **TableName** and **FieldName**, in the case of the **ComboBox** component, instead of the **TableName** property, there will be a **ForeignKey** property.

Therefore, later, when configuring the other forms, I will only list the actions of the buttons you need to assign and which database fields to assign to the components of the form. I will only dwell on setting up some buttons.

**Settings form “frmRent”:**

**ComboBox1** (Client)
- **ForeignKey** = Rent.id_Client
- **FieldName** = Name

**ComboBox2** (Equipment)
- **ForeignKey** = Rent.id_Equipment
- **FieldName** = {Tech} \*${CostPerDay} \* more about this entry below

**DateTimePicker1** (Start rent)
- **TableName** = Rent
- **FieldName** = DateStart

**DateTimePicker2** (End rent)
- **TableName** = Rent
- **FieldName** = DateEnd

**Edit1** (Cost)
- **TableName** = Rent
- **FieldName** = TotalCost
A little more detail on the button [Button1] (Save). This button is used to save data to the database. For this button, select the action "Save record".

In the settings of this button, you only need to choose which components on the form will be useful for storing the information in the database. In our case, you must select all the components. It is also necessary to select the table in which the information will be saved, in our case it is the Rent table (Fig. 29).

In the FieldName property of the ComboBox, you can specify multiple fields. To do this, they must be enclosed in braces.

For example, you want to display the name of the equipment and the cost per day:

`{Tech} ${CostPerDay}`

where “CostPerDay” and “Tech” based on field names will braces and the rest of the text without the braces will POSTER unchanged. Thus, in the combo box, you'll see. "Bulldozer TM-10 $800".

![Figure 29.](image-url)
Also, let's take a closer look at setting up Button2 (Close).

The button closes the form without saving the information in the database. So just choose the action "Close form". This action requires no parameters (Fig. 30).

![Action for the button](image)
Configuring the frmClientList form:

This form is intended to visualize all our customers present in the database. There are also buttons on the form to create, edit or delete a client from the database.

For all clients in the database to appear in the TableGrid component, you must configure it. To access the parameters dialog of this component, click on the "Properties" panel of the component then on the item "Settings" (Fig. 31).

In the settings of this component, you must select the table from which you want to view and select the necessary fields that we would like to see in the component. Configure this component as shown in Fig 32.
Pay attention to the check mark with the title "Show all records from the table", it must be selected.

Then, just list the parameters for the buttons for creating, modifying and deleting the records, (we have already passed these buttons in detail when configuring Form1).

**New client**

Action = New record  
Form = frmClient

**Edit**

Action = Show record  
Table component = TableGrid1  
Form = frmClient

**Delete**

Action = Delete record  
Table component = TableGrid1
Then, I suggest you customize the rest of the forms, you just need to assign the right actions to the buttons and the rest of the components by correctly specifying the **TableName** and **FieldName** properties.

The [frmClient](#) form is intended to create or modify customer data; its configuration is similar to the [frmRent](#) form configuration.

The configuration of the [frmTechList](#) form is similar to the configuration of the [frmClientList](#) form.

The configuration of the [frmTech](#) form is similar to the configuration of the [frmRent](#) and [frmClient](#) forms.

Once you are done with the settings of the remaining forms, it's time to start our project by clicking the button ![Image](#) to test your work.

Before entering data on equipment rental, you must enter all your customers. To do this, click on the [Clients] button located on the main form. The [frmClientList](#) form appears from which you can enter customers by pressing the [New client] button (Fig. 33).
Also, you must enter in the database the equipment you own. For this, on the main form, there is a button [Equipment], which will call the form frmTechList. Press the [New equipment] button to enter the required data (Fig. 34).
Let’s start entering information about renting our equipment. To do this, on the main form, click on [New rent]. As you can see now we have the client and the equipment rented by him, (we have chosen them in the list), it remains to enter manually the dates of the beginning and the end of the hiring as well as the final cost of it (Fig. 35).

![Figure 35.](image)

Perhaps you expected the “costs” to be calculated automatically after you specified the equipment and the rental dates, but to do this, you have to do a job we’ll talk about a little later.

We must therefore independently calculate the "cost".
In Fig. 36, you can see the main window of the program with rental information.

Now, from the main form, you can use the search, for example, by selecting a client from the "Client" drop-down list, then clicking on the [Search] button, you will have all the data about that client.

Congratulations! Your first application using a database is ready!

If you open the project folder, you will find an executable (.exe) file, so that your project can run on any computer without having to install the My Visual Database program. Just copy the folder of your project with all its files.
Honestly, the interface we created has a drawback.

For example, when a new client (whose information is not yet in the database) wants to rent, we must first click on the [Clients] button to enter their data. Then, on the main form, click on the [New rent] button. That is, on the "frmRent" form, select the client from the list of clients and only then enter data on the equipment rented by him. And now, imagine that the customer wants to rent 10 different equipment? We will have to select the same customer 10 times from the list on the "frmRent" form. Okay, that's not practical!

Therefore, I suggest that we remodel this project, in which we will create another more practical interface. The structure of the database will remain the same.

Do not think it's a whim. On the example of the new interface, we will consider the possibility of outputting child records in the TableGrid component.

So, let's get started.

Create a new project (Menu: File> New Project) and save it immediately in a separate folder, for example, "Rent project 2". Name the project "Rent2", as shown in Fig. 37.

![Image of creating a new project](image)
Then, in the **Database table** tab, we should create exactly the same database structure as the previous project. You can see the structure of the database in **Fig. 38**.

![Database structure](image)

**Figure 38.**

As we already know the process of creating tables and fields, we will not dwell on this in detail.

Click the "**Form1**" tab and we will talk about a concept of child records.

The main difference between our current project and its previous version will be that on the "**frmClient**" form, which is intended to create/modify the client, we will also place the **TableGrid** component, which will display the records of the equipment that this client has rented. In other words, child records belonging to the client will be displayed in this component.
If that sounds a bit confusing, then take a look at Fig 39.

Let's take the example of a customer named "KH Services", who rented equipment from the "Rent" table.

As you can see, these are marked in red and these records are children, in relation to the customer "KH Services" of the table "Rent".

Also note that for the child file, there is always a foreign key that refers to the parent record.

![Figure 39.](image)
And for that to become clear, I'll go a little forward and show what the "frmClient" form will look like in the already completed project (Fig. 40).

![Figure 40.](image)

To make the use of the program more convenient, I propose to show the list of clients on the main form ("Form1"), instead of the rental records. Also on the form to create/modify the client ("frmClient"), place the TableGrid component that will display the equipment rented by the client all the time. If we speak in a formal language, we will show the client's child records. Also on this form, there are buttons to add/edit/delete equipment rented by this client.

Thus, when a new client comes to us, we just have to click on the button [New client] on the form Form1, after which the form "frmClient" will appear in which we will enter his data and on the same form we will press on the button [New rent], which will cause the appearance of the form "frmRent", in which we select the rented equipment, the rental dates and its cost.

In addition, if the customer needs to rent the same equipment 10 times, we will not have to select this customer 10 times to create a rental file.
Let's start creating the forms, let's list them.

**Form1**
Form to search for customers. The same form will be the main one, it will appear as soon as you launch your application.

**frmClient**
Form of creation and edition of a client. This form will write data to the customer's DB table. Also on the form, there will be a drop down list of materials rented by this client.

**frmRent**
Form to create and edit a rental file. This form will write data to the DB Rent.

**frmTechList**
Form with a list of equipment.

**frmTech**
Form for creating and publishing information about equipment. This form will write the data into the equipment database.

As you can see, we now need one less shape, which is pretty good. Let's start.
Form1:

Arrange the components on Form1 as shown in Fig. 41.

![Figure 41](image)

On this form, you can search for the client by name or phone number. There are also buttons to call the form to create or edit the client, or also delete the client from the database. And a button with the name [Equipment], which will call another form, with a list of the equipment you own.
**frmClient**:

Create a new form by clicking the button ![button](image). Enter the name of the **frmClient** form. Arrange the components on this form as shown in **Fig. 42**.

![Figure 42](image)

This form is the main difference from our previous project. Now, the equipment rental is not on the main form, but on the form **frmClient**, and on this form, you see not only the customer data, such as his name and phone, but also all the records associated with the rentals from this client.
frmRent:

Create a new form by clicking the button. Enter the form name **frmRent**. Arrange the components on this form as shown in **Fig. 43**.

![Image of frmRent form](image)

**Figure 43.**

This form is designed to create a rental record. Note, unlike our previous project, there is no drop-down list on this form from which we would choose the customer who rents the specified equipment.

The fact is that the **frmRent** form will be called from the **frmClient** form, so the program will automatically associate the customer with this form, and the registration of the rented equipment will automatically belong to the customer whose data will be on the **frmClient** form. In other words, you do not need to worry, the program will automatically connect the customer and the equipment he has rented.
**frmTechList:**

Create a new form by clicking the button ![button](image). Enter the form name `frmTechList`. Arrange the components on this form as shown in **Fig. 44**.

![Figure 44](image)

This form is exactly the same as in the previous project.

On the form created, we will be able to see all the equipment we have, as well as call a form for the addition of new equipment, or modify the information on existing equipment.
**frmTech:**

Create a new form by clicking the button ![button icon].
Enter the name of the **frmTech** form.
Arrange the components on this form as shown in Fig. 45.

![Image](image_url)

**Figure 45.**

This form is exactly the same as in the previous project.

Designed for creating/editing equipment that you own, indicating its cost per day.

Go to the last stage of creating the application, to configure the components on the form.

As you'll remember, component configuration is reduced to assigning actions to buttons ![button icon], and components for entering information (such as ![text icon] ![input icon] etc.) indicate which database table and which field they belong.

We proceed to the configuration of the forms; I will enumerate only the actions to be assigned to the buttons and the fields of the database to be assigned to the components of the form. I am going to dwell on the components whose setting is different from the previous project.
Configuring Form1:

**Client** (EditBox)
   - **TableName** = Client
   - **FieldName** = Name

**Phone** (EditBox)
   - **TableName** = Client
   - **FieldName** = Phone

**New client** (Button)
   - **Action** = New record
   - **Form** = frmClient

**Edit** (Button)
   - **Action** = Show record
   - **Table component** = TableGrid1
   - **Form** = frmClient

**Delete** (Button)
   - **Action** = Delete record
   - **Table component** = TableGrid1
A little more detail on the definition of the **Search** button.

As I mentioned above, on the main form, we will now see the customer records, rather than the rental records, as in the previous project: so this button will look for the data in the **Client** table, instead of the table **Rent**.

In **Fig. 46**, you can see the settings of the [Search] button.
Configuring frmClient:

**Client** (EditBox)
- **TableName** = Client
- **FieldName** = Name

**Phone** (EditBox)
- **TableName** = Client
- **FieldName** = Phone

**New rent** (Button)
- **Action** = New record
- **Form** = frmRent

**Edit rent** (Button)
- **Action** = Show record
- **Table component** = TableGrid1
- **Form** = frmRent

**Delete rent** (Button)
- **Action** = Delete record
- **Table component** = TableGrid1
A little more detail on setting the [Save] button.

You can see the setting of this button in Fig. 47.

![Action for the button](image)

**Figure 47.**

Note that the **TableGrid1** component does not participate in saving the record; it simply displays the rent records belonging to the client on the form. So, you leave it in the list on the left.
Let's take a closer look at the setting (Fig. 48) of the **TableGrid1** component (**Rented equipment by customer**). In this component, we will see the rental records of the equipment belonging to the client, whose data we see on the current form (**frmClient**).

![Set up auto query](image)

**Figure 48.**

Note the selected option "**Show child records (if present)**". Thus, in this component, child records will automatically be displayed, in our case, relative to the client.

In other words, we will see the rental records belonging to the client.
If someone forgot what the child records are, you can look at Fig. 49.

For example, if we see data on the KH Services client on the frmClient form, then in the TableGrid1 component, we will see child records from the Rent table, which are marked in red in the Fig. 49.

The configuration of other forms, such as frmRent, frmTechList, frmTech, is not different from that of our previous project, configure them yourself.

Once you're done with the settings of the remaining forms, it's time to start our project by clicking the button and testing its work.

As in the previous project, you must first add to the database the equipment you own and its cost for 1 day's rent.
For this, on the main form, click on the [Equipment] button (Fig.50).

![Figure 50.](image)

After adding all the equipment you have in the database, you can go directly to work.

Let's say KH Services asked us to rent a TM-10 bulldozer, a K-4561 crane truck and a Kenworth W900 tractor.
If this company applies to us for the first time, then on the main form, we click on the button [New client].

The form **frmClient** appears in which we enter the data on this customer and on the same form, we click on the button [New rent] in order to attribute to this customer the equipment he rented us (Fig. 51).

![Image of frmClient and frmRent forms]

*Figure 51.*

Note that if the client already exists in the database, you do not need to create it again.

Instead, you should find it on the main form and click on the [Edit] button, which will call the **frmClient** form, where you will also assign the next rented equipment using the [New rent] button.

On the main form, using the "**Client**" field, you can search through the database to find the customer.

By default, the search is performed on an occurrence of the full name, i.e. find customer with the name KH Services.

You must also enter its name in the **EditBox**, which is not always practical.
Therefore, in the settings of this text box, you can change the partial search mode, so you will find the KH Services client even if you simply enter service as a search.

To do this, find the Filter property and select as %% as shown in Fig. 52.
2.2 Normalization of the structure of the database

Let's take a break from the practice and do some theory that will help us avoid the errors associated with designing a database structure.

Let's talk about the normalization of the database. If anyone prefers an explanation in an academic style, they can get information about this Wikipedia link:


I will try to explain the principles of standardization in simpler language using examples. First, we need to understand why we need to standardize the structure of the database.

Normalization is a set of rules that we must adhere to when creating a database structure. This will help us avoid errors in the database structure, which can lead to redundant and even contradictory data.

Without knowing it, at the beginning of the book, we were already doing normalization by considering the problems of storing data in a table and dividing them into several smaller tables.

Imagine that we do not know about the design rules of the database and that we have to create a table with a similar structure (Fig. 53):

<table>
<thead>
<tr>
<th>Staff list</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Figure 53.

Let's proceed to normalization.
The first normal form

The definition of the first normal form according to the academic language is:

*A relation is in first normal form if and only if the domain of each attribute contains only atomic (indivisible) values, and the value of each attribute contains only a single value from that domain.*

In simple terms, there should not be multiple values in the field, a field = a value. In our example, pay attention to the column "Hobby", in which the hobby of the person is indicated by a comma. This is not allowed, remember this.

To ensure that the table does not violate this rule, we will have to create additional records for each hobby of the person, i.e. duplicate the data, thereby intentionally creating redundancy in the table (Fig. 54).

![Staff list table]

The rule of the second normal form will help us to get rid of the redundancy of the data.
The second normal form

The definition of the second normal form in academic language is:

A relation that is in first normal form (1NF) must meet additional criteria if it is to qualify for second normal form. Specifically: a relation is in 2NF if it is in 1NF and no non-prime attribute is dependent on any proper subset of any candidate key of the relation. A non-prime attribute of a relation is an attribute that is not a part of any candidate key of the relation.

In plain language, each table must have a primary key that uniquely identifies the entry in the table. In our case, this is the column with the name `id`, in which the ID of the record is stored; it is unique for each entry of the table.

If you use the My Visual Database program, when creating a table, this key is always created automatically. You just do not need to worry about it.

This rule also states that there should not be duplicate data in the table (redundancy). If you look at the table (Fig. 54), which is in the first normal form, some records contain the same data, that is, the data is duplicated. The reason is the column "Hobby".

To solve the problem of data redundancy, you need to create another table, in which we will store the hobby of the staff and there will also be a foreign key in the table that will determine which person has such hobby specified.

Now that we have two tables, we can create any hobby that can belong to a person, thus avoiding data redundancy. To understand it, look at Fig. 55.
The third normal form

The definition of the third normal form in academic language is:

*Third normal form (3NF) is a normal form that is used in normalizing a database design to reduce the duplication of data and ensure referential integrity by ensuring that (1) the entity is in second normal form, and (2) all the attributes in a table are determined only by the candidate keys of that relation and not by any non-prime attributes. 3NF was designed to improve database processing while minimizing storage costs.*

The third normal form is necessary to fight transitive dependence.

First, let's see what it is. Note that in the staffing table (Fig. 56), there are City and Index columns. Unfortunately, people tend to go wrong. Suppose a person filling a database, made a mistake and introduced an employee living in Paris Lyon index, what kind of information after that to believe? As a result, we obtain conflicting data.

Thus, the City and Index columns depend on each other. But if we know the Index, we can find the city by this index. Therefore, why not store only the index in the staffing table?

Now, to get rid of this transitive dependency, we need to create two more tables. A table with the names of cities and a table with indexes, in which there will also be a foreign key, which determines which city the index belongs to (Fig. 56).

![Figure 56.](image)

In practice, we do not always use the third normal form, leaving fields with a transitive dependency.

If you are designing a database that will be used, for example, in a Manuel, you can override and leave as it is, sacrificing, for the sake of simplicity, the possible emergence of conflicting data. But if you design a database to use, for example in the banking sector, for you, I would not ignore the third normal form.
Let's go back to Fig. 56, because in the **Persons** table there are two other fields with a transitive dependency. These fields are **Salary** and **Tax**. Suppose we have a fixed tax of 30%, knowing the salary, we can always calculate the tax. Normally, you do not have to store data in a table that can be obtained from other fields in the table, so you can safely delete the Tax field.

> Once you understand these rules, they will seem completely natural to you. When designing the databases, you will not even notice how instinctively you go adopt them completely.

### 2.3. Cascading delete and data integrity.

I hope you have not forgotten our common project of managing a rental of construction equipment. We will continue to work with him, adding features and learning new features in databases.

Launch **My Visual Database** and open our project again, which is located in the "Rent project2" folder. Click on the [Database tables] tab to see the tables created in our database (see Fig. 57).

![Figure 57.](image-url)
And let’s talk about cascading deletion.

Sometimes we have to delete a client from the database and all the data associated with it. In our case, if you delete the client (Client table), you must also delete all data on its equipment rentals (Rent table), otherwise we will have rental records referring to a non-existent customer, which is unacceptable.

Note the foreign key id_client in the Rent table, which references the client from the Client table. For this foreign key, it is necessary to operate what is called cascade deletion. As a result, when the client is removed from the Client table, the Rent table records will be automatically deleted. Anything that refers to the client is removed because of the external key id_Client.

In other words, deleting a parent record will also delete child records.

Enable cascading deletion for a foreign key, as shown in Fig. 58.

![Figure 58](image)

In fact, without cascading deletion, you would not be able to remove the client from the Client table because this client is registered for its rentals. In other words, because in the Rent table there are entries in which the foreign key client_id refers to the client. In addition, a message will warn you.

But if this client has not yet registered for a rental, you can delete it from the database because while data integrity is not compromised.

The same can be said about the id_Equipment foreign key in the Rent table, you cannot remove the equipment from the Equipment table if at least one entry refers to this equipment in other tables.
Thus, the database automatically retains its integrity to prevent the situation where a foreign key refers to a non-existent record.

Therefore, by using cascade deletion, you delete the record and all data associated with that record in the database, or you do not delete anything at all.

In general, it is not allowed to delete anything from the database, but it is recommended to mark the entry as an archive using a field of type "Yes / No".

2.4. Add a client type

Let's continue our practice. Let's make sure that you can assign a type to the customer, for example, if it's "Individual" or "Company".

And then an error may occur. You may be strongly tempted to create a new text field in the "Client" table, wherever you simply enter the type of client. As a result, your data in the "Client" table looks like, for example, Fig. 59.

<table>
<thead>
<tr>
<th>Client name</th>
<th>Client type</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>KH Services</td>
<td>Company</td>
<td>555-55-55</td>
</tr>
<tr>
<td>euroasfalt</td>
<td>Company</td>
<td>455-66-77</td>
</tr>
<tr>
<td>Master builder</td>
<td>Company</td>
<td>777-77-77</td>
</tr>
<tr>
<td>Concrete master</td>
<td>Company</td>
<td>844-44-44</td>
</tr>
<tr>
<td>John Smith</td>
<td>Individual</td>
<td>222-22-22</td>
</tr>
<tr>
<td>Ivan Ivanov</td>
<td>Individual</td>
<td>333-33-33</td>
</tr>
</tbody>
</table>

*Figure 59.*

Pay attention to the "Customer type" column. According to the purpose of this column, we can only have two values, "Company" or "Individual", where we see the apparent redundancy of the data, which contradicts the second normal form (Chapter 2.2).

There is no point for each customer, to enter their type manually. This is unacceptable when designing databases; later, this will affect its performance, as well as the greater probability of errors in the database, for example, someone decides that just write just "Comp.", or "Indv.", thus losing the integrity of the data and greatly complicate the work with them.

So, how?

The correct solution is to create another table in which to enter the types of authorized clients, let's call it "ClientType". And in the table "Client", create a foreign key that will refer to the table "ClientType", in the
foreign key so there will be the identifier of the selected customer type is saved. To make this clearer, take a look at Fig. 60.

![Table](image)

**Figure 60.**

With this approach, you do not have to manually enter the type of client each time; just select it from the list. In addition, you can add new customer types to the "ClientType" table, for example "Individual Entrepreneur", "Corporate State", etc.

In addition, nothing prevents you from renaming client types and this does not affect the integrity of the data. In the foreign key "id_ClientType" of the "Client" table, only the numeric identifier of the client type is stored, and not its textual representation.

Once the theoretical part is understood, put it now into practice:

1. As mentioned above, we need to create another table in the database, call it "ClientType".
2. In this table you need to create a column with type **TEXT**, call it "TypeName".
3. In the "Client" table, create a foreign key to the "ClientType" table.
As a result, you should get it as shown in Fig. 61.
Look at the **frmTechList** and **frmTech** forms. Exactly the same needs must be created to work with the types of clients. The first is to display the list of added records and the second to create / edit records.

Create a new form named **frmClientTypeList** and place a TableGrid component and three buttons on it, as shown in **Fig. 62**.
Also create a form named `frmClientTypeAdd`, as shown in Fig. 63.

![Figure 63.](image)

Now it remains to configure the components on the created forms. Setting up as usual is pretty simple.

Configuring the `frmClientTypeList` form:

**TableGrid1**

1. Access the settings of the TableGrid1 component to configure it to display data from the "ClientType" table
2. In the settings for this component, add the field "ClientType.TypeName" to the list
3. Select the option "Show all records from the table"

Next, configure the buttons for creating, editing and deleting entries:

**New type of client**

- **Action** = New record
- **Form** = `frmClientTypeAdd`

**Edit**

- **Action** = Show record
- **Table component** = TableGrid1
- **Form** = `frmClientTypeAdd`
Delete
   Action = Delete record
   Table component = TableGrid1

Configuring the **frmCustTypeAdd** form:

**Edit1** (Client Type)
   TableName = ClientType
   FieldName = TypeName

**Button1** (Save)
   Action = Save record
   Component to save = Edit1
   Table = ClientType
Customizing Form1:

The form with the new components will look like (Fig. 64)

![Image of Form1](image)

*Figure 64.*

On the main form, we add the **ComboBox** component that will allow us to filter the records by type of client.

**ComboBox1** (Type client)

*ForeignKey* = Client.id_ClientType  
*FieldName* = TypeName

Then go to the settings of the "Search" button to add this component (**ComboBox1**) to the list "1. Select the components involved in the search", so this component will also participate in the search.

In the "ClientType" table (3. The result), select the "TypeName" field, so that after the search, we will know what type the customer belongs to.

Place a new button on the form that will call the form "frmClientTypeList" so that we can add the types of clients.

**Types of client** (button)

*Action* = Show form  
*Form* = frmClientTypeList
Configuring the **frmClient** form:

Just like on the previous form, we need to add a **ComboBox** component so that we can choose which type the client belongs to

**ComboBox1** (Customer Type)
- **ForeignKey** = Client.id_CustomerType
- **FieldName** = TypeName

Finally, in the settings of the **[Save]** button, add the **ComboBox1** component to the "1. Select the components involved in saving the record" list so that the data from this component is saved.

As a result, the form will look like the one shown in **Fig 65**.

---

![Figure 65](image-url)

**Figure 65.**
You can now start the project and add client types by clicking the "**Type of client**" button on the main form, as shown in **Fig. 66**.
By adding client types, you can start using them for existing and new customers, indicating the type they belong to (Fig. 67).

Figure 67.
And finally, to give life to the interface, for each button you can select your icon using the Icon property (see Fig. 6.8).
3. Printing

3.1 Ways to print

Sooner or later you will want to print the information contained in the database. For that, we have three ways to proceed.

1. The easiest way to print is to send the data contained in the TableGrid component to Microsoft Excel, and then use the built-in functions of this program to print them. The button with the action "Open in Excel" will help you with this (Fig. 69).

2. Use the Microsoft Word document as a print template. Having issued the document in a certain way, you can insert data in pre-prepared places. This method involves the use of a simple script, which will be discussed in the chapter on programming scripts.

3. Use the built-in report designer. This method is the most advanced and allows you to implement any report with other prints and export to other popular formats (doc, xls, pdf and many others). We will see this in detail in the next chapter.
3.2 Report Designer

As mentioned above, the built-in report designer allows you to implement a report of almost any complexity. But for now, we will only talk about its main features, which, as a rule, are suitable for most tasks.

As an example, we print a simple contract, which deals with the customer's responsibility for the security of the equipment rented by him. This will look like Fig. 70.

![Figure 70.](image)
To create a report and then print it, you must use a button with the "Report" action. Place the button on the main form of our project and call it "Responsibility Contract", as shown in Fig. 71.

Using the Icon property, you can select the "Printer" icon

![Image](image.png)

*Figure 71.*

The setting of the button with the "Report" action is almost the same as that of the button with the "Search" action, because in fact both actions search the database with the next output of the result of the search, but in our case, the result of the search will be seen when printing.
The "Report" button is shown in Fig. 72.

Let's take a closer look at these step-by-step parameters:

1. Select the components involved in the search
   Here the **TableGrid1** component is selected, in which we see all our clients, but client has been selected in the **TableGrid1** component.

2. Select the database table to search for
   We will print client information, select the **Client** table.

3. Select the fields from the tables, required in the report result.
In the table, select the fields we need for the report, in our case it’s the client’s name and type (respectively the `Client.Name` and `ClientType.TypeName` fields).

4. Select a report template
Since we still need to create a report template, this parameter will be "Open report designer ..."
Last step. Close the running project and go back to the settings of the button [Print responsibility].

In the settings "4. Select report file", instead of "Open report designer…", select from the list the previously saved report template "Responsability.fr3", as shown in Fig. 75.

Now you can start the project again. After selecting the required client in the TableGrid1 component, click the [Print responsibility] button, to view a ready-to-print report with the selected customer data.

Sometimes it is convenient, put the print button directly on the form of adding/editing. Therefore, let's copy the [Print responsibility] button on the frmClient form.
In the settings for this button, remove the TableGrid1 component from the "1. Select the components involved in the search" list. In fact, if the button with the "Report" action is placed on the form to be added/modified, it will automatically detect which record to send to the printer.

Let's create another report, but a little more complex. Now, we have to print data about the customer, with all his equipment rentals. Therefore, we will publish a report as shown in Fig 76.

![Figure 76](image)

Place on the main form a new button called [Rented equipment], as shown in Fig. 77.

![Figure 77](image)
Select the "Report" action for this button, the setting of this button is shown in Fig. 78.

The setting of this button is practically the same as that of the button [Print responsibility], only the new fields that we must see in the report will be added to the item "3. The result".

Here, it is worth mentioning a nuance. As you can see in the settings, to select the result, we select the fields of 4 different tables (Client, Equipment, Rent, ClientType). In other programs, it is necessary to compile an SQL query that manually specifies how a table is associated with another table and in what order.
The My Visual Database program tries to understand for itself how to link tables to each other to get exactly the data you expect, eliminating the need to study the SQL query language.

Unfortunately, the program can not always predict the need to link tables to get the data you want to appear. This situation can occur when you need to associate 3 or more tables without obvious links between them. Thus, for example, there is no link between the table "Client" and "Equipment", the relationship between these tables is achieved by means of another table "Rent".

What should I do? In the tab [Database tables], under each table, there is a check mark "The table is a dictionary".

On this tab, you should note which tables in your project are dictionaries. But how do you know which tables are dictionaries?

Examples of dictionary tables can contain country names, statuses (Open, Closed), types (Company, Individual), name of goods with prices, etc.

That is, such tables, which, as a rule, are filled first when you start working with them and are not changed later.

In our project, dictionary tables can include "ClientType" and "Equipment", check the "The table is a dictionary" checkbox below these tables.

If checked, you will help the program link the tables properly.

And so, we can continue to create a report on leased equipment by the client.

As you remember, to create a report template, you must run your project.

In the current project, click the [Rented equipment] button, created earlier, to open the Report Designer.

This report will be slightly more complicated than the previous one, since what are called blocks (or bands) will be used.

Blocks allow you to create almost any structure of the report; we will now make arrangements with some of them.

To see all the available blocks, click on the icon on the left, then you will see the menu, as shown in Fig.79.
We will proceed to the creation of a report using blocks, passing we will learn what they serve.

In the menu shown in Fig. 79, select the **Report Title** block, after which the block appears in the report. In this block, you can put the title of the report, for example its name. The information in this block will only be printed on the first page of your report.

Put the name of the report in this block.
As a result, you should get, as shown in Fig. 80.

![Figure 80.](image)

Then place the PageHeader block. The information placed in this block will be displayed on every printed page (in case your report does not fit on one page).

In this block, we will put information on the client. In addition, we will put the headers for the table, which will be located in the next block.

Place text and data fields in this block, as shown in Fig. 81.

![Figure 81.](image)
The next block to be placed in the report is **Master Data**.

This block is designed to display information in the form of a table.

With this block, we will get a table in which all the equipment rented by the customer will be transferred.

Place this block in the report, selecting it from the menu.

Before it appears in the report, you will see a window with the **Select DataSet** header, where you must select the data source.

Select the data source named **Report** from the list and click OK.

In this block, you must put the database fields from which the table will be generated.

As a result, you should get what Fig. 82 shows.

---

**Figure 82.**

![Table](image-url)
Place the next **Footer** block. This unit will print only once, i.e. you will not see it on every page of your report. In this block you can calculate, for example, the total amount spent by the customer for equipment rental.

To calculate the total amount, place the System Text component in this block. Once it is placed, a dialog box appears, which you will need to configure, as shown in **Fig. 83**.

![System Memo dialog box](image-url)

**Figure 83.**
That done, your report template should look like Fig. 84.

![Figure 84.](image_url)

And the last block we put in the report is PageFooter, this block will be printed on every page of the report, we just use it to number the pages.

In this block, place the Page # system variable on the right side of the report designer (Variables tab), drag it, as shown in Fig. 85.

![Figure 85.](image_url)
Your report template is now ready, as shown in Fig. 86.

Go to the File > Preview menu or just press Ctrl + P to see what your report will look like.

Save the report template in the Report folder of your project, via the File> Save As ... menu.

Give a name for the file, for example "RentedEquipment", as shown in Fig. 87.
It remains to return to the settings of the [Equipment Rental] button and select this report template to be used for printing, Fig. 88.
To come up ...

The practical manual of My Visual Database is still in progress.

The latest version of the book you can find here:


In addition, from this link, you can write a review on the book, or ask your question.
# TABLE OF CONTENTS

1 – **Introduction**  
1.1 – Introduction  
1.2 – Database  
1.3 – Identifying entities  
1.4 – Links between tables and foreign keys  
1.5 – Types of data  

2 – **Put into practice**  
2.1 – Let’s create our first application working with a database  
2.2 – Normalization of the structure of the database  
2.3 – Cascading delete and maintain integrity  
2.4 – Add a client type  

3 – **Printing**  
3.1 – Ways to print  
3.2 – Report designer  

Table of contents