

Data Model Editor : Overview

Data Model Editor is intended for creation database description used by Simple Query/Active Query controls. This description allows to hide the details of your database organization from the end-user. It contains information about tables, links between them, some entities of the data domain named "Fields" (not the same as fields in database tables although some entities can be totally equal to fields in database). In addition this description includes list of operators used in conditions (such as "is equal to" or "is more than") and list of custom (manually defined) conditions which can be used in queries. Though being easy to use and self-explanatory, structure editing requires some understanding of the relational databases handling. Users totally unfamiliar with database management should better leave this dialog for an database operator or another experienced user.

This dialog contains such pages:

- Tables - describing tables taking part in the data model;
- Fields - describing the entities which end-user can operate with to build queries;
- Operators - describing operations (like comparisons) upon the data model's fields;
- Conditions - describing custom (user-defined) conditions which then can be used in queries;

Each of the pages is described in detail in appropriate topics of this help.

Use *Load* and *Save* speed buttons to load or save the data model from/to INI-like textual file.

When you have complicated database structure and want to use one table in several different relations then read [How to work with aliases](#)

Data Model Editor: Tables page

Use this page to describe tables taking part in the [data model](#). This page contains two main panels: tables with links between them and properties of the current chosen (active) table.

Tables are represented by movable windows with fields list inside. If Data Model Editor is not currently connected to database then tables contains just the <not connected> caption. Active table is indicated by color and more widely border.

Links between the tables are represented by lines between the tables. Active link is more thick then others.

To select the table or link just click on it.

Properties of the selected table are shown in the right panel. To edit properties of the current link - just double click on it and [Edit Link dialog](#) will appear.

There is "Full Screen Mode" button in the left-top corner of design panel. Using it you can maximize the panel to make it take all place of the screen. It allows to see and operate with more tables and the same time without using scroll bars. Pressing the button once again restore previous size of the design panel.

To add or delete tables and links use the corresponding buttons at the top of the dialog (they should appear when "Tables" page is selected) or right-click pop up menu.

NOTE: any particular database table can be added to structure more than several times with different aliases. It is necessary to eliminate the ambiguities when some two tables have several link paths from one to another. For more information see: [Working with aliases](#)

Table properties:

Table alias is an alias for the table in generated SQL statements (optional but needed if you link the table more than once on different conditions)

Table hint(s) is intended to specify locking method used in MS SQL Server syntax (like NOLOCK, ROWLOCK or READCOMMITTED);

Quote table name check box means if the table name should be put in double quotes in SQL statements; useful for table names including spaces and national characters.

Data Model Editor: Edit link dialog

In this dialog, you are editing a data link between tables.

Table2 combo box chooses a table to be linked to edited table.

Two list boxes below ("Table1 fields" and "Table2 fields") display fields of main table (the edited one) and linked table (the one you selected above). The third list box (between them) allows to select the operator (equal by default) used in join condition.

The user should select the fields and the operator used in join condition and click *Add*.

If you click *Auto*, Structure Editor will try to suggest fields to use for join condition.

Quote field names check box means if the field names should be put in double quotes in SQL JOIN conditions; useful for field names including spaces and national characters.

Joined fields list box displays fields already engaged in join condition. *Delete* button deletes a selected condition; *Clear* button clears all conditions completely.

Join type group of radio buttons allows you to specify the type of join (inner, left outer, right outer or full join).

Operator list allows you to select an operator which will be used in join condition (= by default).

Data Model Editor: Fields page

This page describes fields participating in the data model.

There are two main types of fields:

- Data Field - field which is totally correspond to particular field in some table;
- Virtual Field - calculated field which is defined by some expression containing several fields, operators (+, -, ||, etc.), constants and even function or storage procedure calls;

Fields tree holds list of all defined fields.

To add a data field, right-click the list and choose *Add Field*. Then, in appearing dialog, choose table, its field and click *OK* button.

To add a virtual (i.e., calculated) field, right-click the list and choose *Add Virtual Field*.

To delete a field, choose it from the list, right-click it and choose *Delete Field*.

All operations above can be done also through corresponding speed buttons (at the top of the dialog).

To edit a field, choose it from the list and change its properties in the editor appearing in the dialog's right part.

To move the field from one group to another (or to change its appearance order within the group), drag it to the appropriate place.

The field's property editor has *General*, *Operators* and *Value Editor* pages described below.

General page

Display name is a string meaning how the field name will be displayed for an end-user

Group name means the group holding the field. This "group" doesn't affect any built SQL text; it's introduced just for user's convenience when it comes to choosing a field in the visual query builder.

Choose a group from the drop-down list or enter a new group name.

Following parameters are different for data and virtual fields.

For **data field**:

- **Quote field name** check box means if the field name should be put in double quotes in SQL statements; useful for field names including spaces and national characters.

For **virtual field**:

- **Field type** is data type for resulting field. See [Field types](#) for details.
- **Expression** is SQL expression for calculating the field's value.

Operators page holds list of all operations applicable to the field. Add or remove operations by corresponding buttons. Use *Clear* button to remove all operators from the list.

See [Operators page](#) for details about operators used in data models.

Value editor page defines how the field value (or, rather the value of a parameter to which the field is compared and having the same type as the field) will be edited in the visual query builder.

Choose way of editing from *Editor type* drop-down list. Editor parameters vary depending on editor type.

- **Auto**: means that the most appropriate value editor will be used depending on field's data type and operator which used in condition. For example date field - DateTime Picker control will be shown, for boolean field - the user will get an ability to select the value from the list of two items: False and True;
- **Edit**: field will edited in plain edit field. *Default value* is value for the field if one isn't specified. *Use mask* check box indicates if an editor mask should be used when editing the field. See [Editor masks](#) for details on mask syntax.
- **List**: user will be prompted for field value by a fixed list. *Items* column stands for values accepted as the choice result (and actually inserted into SQL text); *Values* are captions displayed to user in the pop-up list.
- **Custom**: The programmer should provide the value editor for this field manually using OnCustomValue event. There is an auxiliary text field "Custom Data" where developer can enter any textual information and get access to it at run-time through CustomData property of any condition object;
- **SQL**: Similar to "List" value editor type, but the list of available values is the result of some SQL query to database. First column of the resulting data set is interpreted as actual value which will be used in

result SQL statement and the second one is used as captions (the text shown to user).

Data Model Editor: Operators page

This page defines operators which can be used in conditions (such as 'equal to', 'less than' and others). List in the left part shows defined operations. Add or delete one by right-clicking the list and choosing appropriate topic from the pop up menu or using corresponding speed buttons on the top of the page. To edit an operation choose it from the list and modify its properties in the right part of the page.

Operator name is internal identifier for the operator.

Display name means how the operation will be displayed for a user working with visual query editor.

SQL expression is template for expression in generated SQL query. It may contain any correct SQL expressions (operators such as =, >, <, functions or even names of stored procedures) and the following special variables:

@f - is substituted with the field's name;

@1 and @2 - are substituted with 1st and 2nd constant parameter.

Constant value variables must be enclosed by single quotes in template expression (the quotes will be deleted for expressions where there are not necessary, for example for integer or real data types).

Examples:

For the simple "is equal to" operator the format string is:

```
@f = '@1'
```

The "starts with" operator has the following format:

```
@f LIKE '@1%'
```

If you want to provide case insensitive "starts with" operator use such format string:

```
LOWER(@f) LIKE LOWER('@1%')
```

Values format is a string which will separate parameters (in case there are two of them) in the visual query editor. For example for "between" operator Value Format field contains 'and' word which means that this word will be placed between two constants values in condition:

SomeField is between 1000 and 2000

Value kind defines kind of data which will be used in condition.

"Scalar" means simple single value is needed: one string, one number, etc.

"List" type requires list of scalar values separated by comma. E.g., having this option checked, when the user enters **a, b, c** as parameter value, it's treated as '**a**', '**b**', '**c**' instead of '**a, b, c**' in the generated SQL text.

"Sub Query" type means that operator requires SQL SELECT statement as value in the right part of condition. To build this statement query panel opens separate dialog.

Type of expression means expected type for parameter. It can be the same as field type or of some certain type (see [Field types](#)).

Apply to types is list of check boxes defining field types to which the operation is applicable.

Operators page has also page have two additions buttons: "Add operator to appropriate fields" and "Add/Update default operators".

"Add operator to appropriate fields" button can be used to add new created operator to all fields in the structure which has appropriate type.

"Add/Update default operators" button add the default operators into the list or update the display name for existing default operators.

Simple Query has such predefined operators:

Name	Display name	SQL expression
Equal	is equal to	@f = '@1'
NotEqual	is not equal to	@f <> '@1'
LessThan	is less than	@f < '@1'
LessOrEqual	is less than or equal to	@f <= '@1'
GreaterThan	greater than	@f > '@1'
GreaterOrEqual	greater than or equal to	@f >= '@1'
IsNull	is null	@f is null
InList	is in list	@f in (@1)
StartsWith	starts with	@f like '@1%'
NotStartsWith	does not start with	not(@f like '@1%')
Contains	contains	@f like '%@1%'
NotContains	does not contain	not(@f like '%@1%')
Between	is between	@f between '@1' and '@2'
InSubQuery	InSubQuery	@f in (@1)

Data Model Editor : Conditions page

This page allows to specify custom conditions which then can be added into the query. Use this ability to declare the conditions which are used very often or which can not be defined using standard <field> <operator> <value> scheme. For example, the following query:

"select all orders which are made in the same day where new employer was hired" does not match the standard condition scheme used in query panel but can be simply defined in SQL:

Orders.SaleDate = Employee.HireDate

To provide our users with an ability to build such queries we should define new condition with the following properties:

DisplayFormat: "Order made when new employer was hired"

SQL Expression: "Orders.SaleDate = Employee.HireDate"

Here DisplayFormat is text which will be shown for the users and which will be added into query panel as one more condition. DisplayFormat property can contain fields marked by @ symbol. It means that user will need to enter some value in this place. Each @ symbol correspond to variable which should be defined in SQL Expression property. Variables are specified by @ symbol and following number starting from 1. Here @1 variable correspond to first @ symbol in DisplayFormat property, @2 - to second.

For example, the following condition "Address contains <some value>" can be defined by standard methods (using <field> <operator> <value> scheme) but can be also declared as custom condition with the following properties:

DisplayFormat: "Address contains @"

SQL Expression: "Customer.Addr1 LIKE '%@1%'"

Value Data type: String

Value kind : Scalar

The last two properties can be specified only when you define one or more values in DisplayFormat property. In such case new item(s) will appear in "Values" list box (at the bottom of the "Conditions" page). For each value can be defined its data type (String, Integer, Date and so on) and its kind (Scalar, List or Sub-Query). See [Operators page](#) topic for more information about kinds of values.

Data Model Editor: Working with aliases

When do you need aliases? Sometimes you want to use one table for several different links and only your users query can define which link you want to use to build correct query. To build such links you need to use aliases. Let's see the following example.

You want to have links like Table1.FieldA -> Table2.FieldC and Table1.FieldB -> Table2.FieldC. You have a database that contains the information about people migration within the United States. There are tables **People** and **States** in your database. **StateFrom** field contains 2 letter state abbreviation of the state where the person from, and **StateLives** contains the state abbreviation of the state where the person lives now. State table contains **StateID** - this 2 letter key and **StateName** - long state name.

People	States
PersonID	StateId
Name	StateName
StateFrom	
StateLives	
Phone	

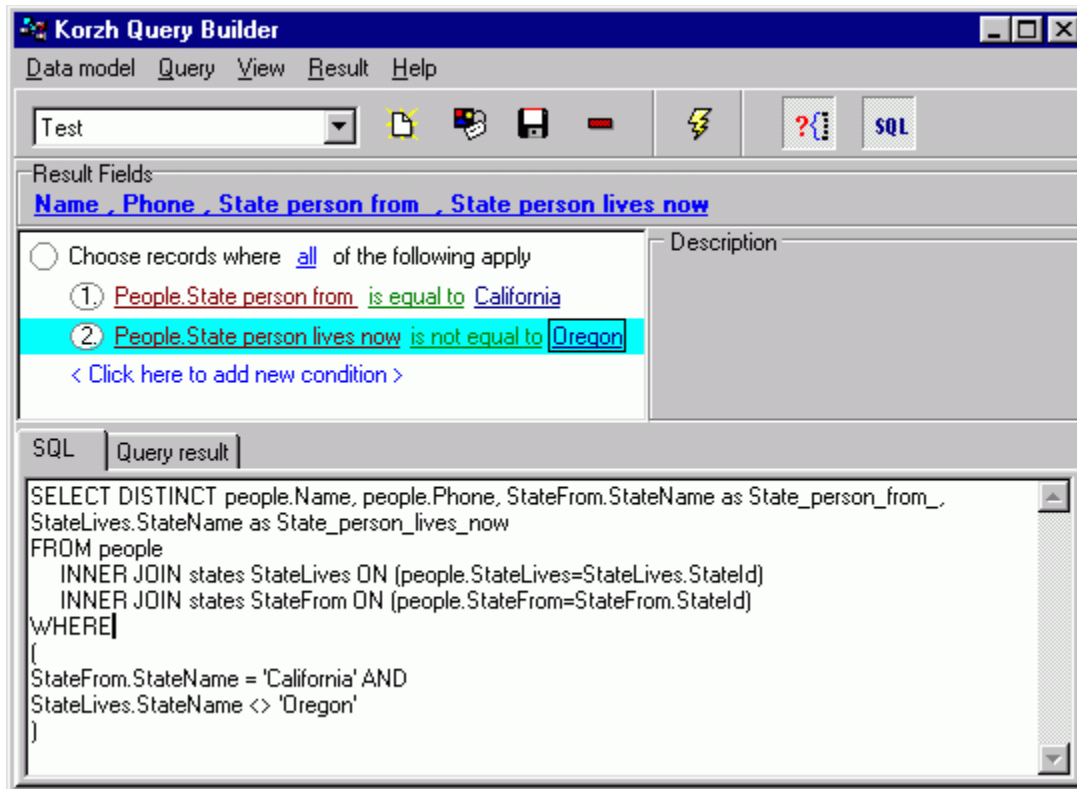
Open Data Model Editor and add People table. After this add States table twice and set Table Aliases to 'StateFrom' for first table and 'StateLives' for second. Add 2 different links to table People:

1. People.StateFrom -> StateFrom.StateId
2. People.StateLives -> StateLives.StateId

Then go to the [Fields page](#) and create two fields:

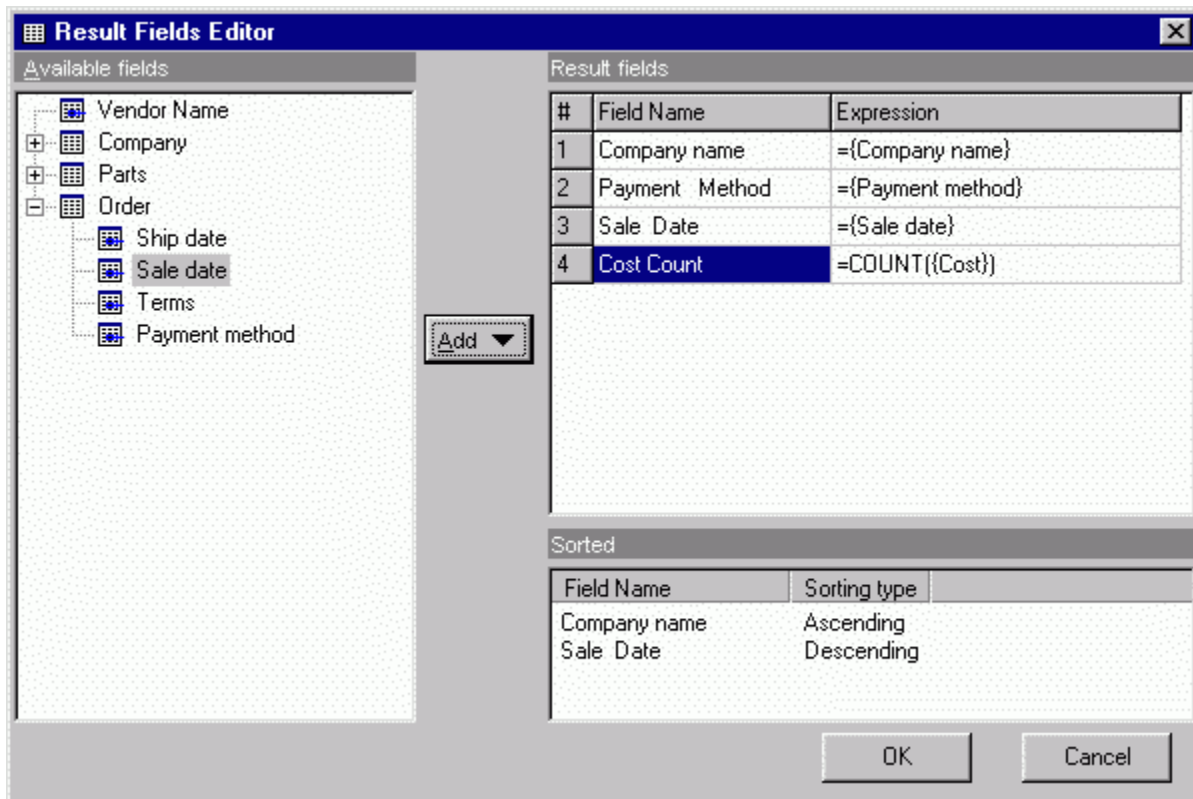
1. "State person from" which corresponds to StateName field from the States table with StateFrom alias
2. "State person lives now" - corresponds to StateName field from States table with StateLives alias.

Now your users can create queries and use 2 States tables which are really connected to one table in the database. SQL standard allows you to create such queries when you use aliases and Simple Query does too. See the example on the picture:



Here we ask for the people who have arrived from California and do not live in Oregon now.

Result Fields Editor : Overview




Available fields tree shows fields available in data model. Fields are grouped as defined in Structure Editor; unclassified fields (i.e., not belonging to any group) are represented as the tree's root level.

To add a field select it in the list and press  button. From appearing menu select how would you like to add the field. Variants are:

- **as is**: the field is added as is; no formulas are applied
- **MIN, MAX, AVERAGE, COUNT, SUM**: the field is included as argument of the corresponding SQL aggregate function.

To create a calculated field, follow next steps:

1. Choose first field from the tree and add it to the list
2. Choose second field from the tree; select the field you just added in the *Result fields* list

3. Press  button – and chose the operation from drop down menu depending on what you need to do (Add to result field, Subtract from result field, Multiply by result field, Divide by result field).
4. Using steps 2-3, you can add another fields to the expression, like: field1+field2*field3-field4.

5. To insert brackets, select the result field and click *Expression* cell. In the edited expression, fields are shown as @1, @2,...,@n.

To remove a field select it in the *Result field* list, press right mouse button and choose "Delete result field" menu item.

To add the field to sorting order, select it in the *Result field* list press right mouse button and choose "Sort result field" menu item. Then in the sorted fields list below use can change type of sorting

You can change name of the result field: just select it and rename the way you want to see it.

Simple Query/Active Query

Simple Query/Active Query is a technology providing user-friendly interface for building queries. It is irreplaceable for developing end-user applications intended for users who are totally unfamiliar with SQL or QBE yet needing to form sophisticated requests to databases.

Simple Query is a Delphi/C++Builder/VCL component set.

Active Query is an ActiveX wrapper under Simple Query components which provide the same functionality for VB developers (as well as VC++, Access, PowerBuilder or any other development environment which supports ActiveX) .

Detailed information about latest changes in **Simple Query** and **Active Query** could be found at:

<http://www.korzh.com/delphi/> for Delphi **Simple Query** library

<http://www.korzh.com/activex/aq/> for ActiveX **Active Query** control

Data model

Description of a database structure including:

- list of tables
- list of links between tables;
- list of entities of the current data domain named "Fields"
- list of conditional operators used in query conditions;
- list of custom conditions;

Structure file (.dbs) format

Structure file has INI-like format. All information is divided on several blocks. Each block represents one type of the objects or common information about the data model (like [Main] section) and can be represent by one or more sections. Section names are always enclosed in brackets "[]".

General structure of the .dbs file:

<Main section>
<Operators block>
<Tables and Links block>
<Fields block>
<Field groups block>
<Custom conditions block>
<Special DB information section>

Main section

Main section represent the common information about the data model. Here is its syntax:

[Main]
Version=<number>
OperatCount=<number>
TableCount=<number>
FieldCount=<numbe>
CondCount=<number>
FieldGroups=<number>
DefaultMainTable=<table name>
RootFieldsFirst=<boolean>

Version parameter represents the format version used in the current .dbs file. It helps to read old files when we change the format of the file by some reason. Current format is 3.5

OperatCount represents the number of operators defined in current data model;

TableCount represents the number of tables defined in current data model;

FieldCount represents the number of fields defined in current data model;

CondCount represents the number of custom conditions defined in current data model;

FieldGroups represents the number of field groups defined in current data model;

DefaultMainTable represents the name of the default main table. Can get empty value.

RootFieldsFirst indicates if Data Model Editor and other dialogs should show root fields before or after all other field groups. Can be 1(means root fields will be shown first) or 0 (root fields are shown after all field groups).

Operators block

Operators block consists of OperatCount sections (where OperatCount - the number defined in OperatCount parameter in Main section). Each section has the following syntax:

[Operator:<Index>]
Name=<string>
DisplayName=<string>
SQLName=<string>
ValFormat=<string>
ExprType=<string>
AppliedTypes=<list of field types>
ValueKind=<value kind constant>

Here <Index> is an ordinal number of described operator started by 1.

Name, *DisplayName*, *SQLName* and *ValFormat* represents corresponding properties of operator object.

For example:

Name=equal

DisplayName=is equal to

SQLName=@f=@1
ValFormat=

ExprType can be one of the type constants or has special value "SameAsFied" (without quotes) which indicates that the type of operator is the same as the type of field which it is used with in condition.

AppliedTypes represents the list of types which this operator can be applied for. List of types contains data type constants delimited by comma.

ValueKind represents the kind of the value required by operator: scalar, list or sub-query. Can be one of the following: vkScalar, vkList, vkSubQuery.

Tables and Links block

Tables block consists of TableCount [Table:X] sections plus some number of [Link X-Y] section depending on link count defined in table sections. Each table section has the following syntax:

[Table:<Index>]
Name=<string>
DisplayName=<string>
Alias=<string>
Quoted=<boolean>
Hints=<string>
DsgnPos=<left>,<top>,<width>,<height>
Links=<list of table aliases which the current table is linked to>

Each link listed in *Links* parameter must be described in separate Link section :

[Link:<table1>-<table2>]
FldName1=<list of field names from table1>
FldName2=<list of field names from table2>
Operators=<operators list>
QuoteFields=<boolean>
LinkType=<type of the link>

Here <table1> and <table2> are aliases of the tables which take part in the link. *FldName1* and *FldName2* represent list of fields from first and second table which take part in link condition. *Operators* parameter represents the list of operators used in link conditions. For example, the following link description:

FldName1=CustNo, OrderNo
FldName2=CustNo, SaleOrderNo
Operators==,<>

represents such link condition: table1.CustNo=table2.CustNo and table1.OrderNo <> table2.SaleOrderNo

LinkType parameter represents the type of the join which will be used in queries: INNER, LEFT OUTER, RIGHT OUTER or FULL.

Can be one of the following values: inner, left, right, full.

Fields block

[Field:<index>]
ID=<number>
Kind=<field kind, kfkData or kfkVirtual>
FieldExpr=<string>
DisplayName=<string>
TableAlias=<string>
EditType=<number>
FieldType=<number>
FieldSize=<number>
Quoted=<boolean>
UIC=<boolean>

UIR=<boolean>
Operators=<list of operator names>

[Field:<index>.SQL]
<SQL SELECT statement>

Field groups block

[FieldGroup<index>]
GroupName=<string>
Fields=<list of fields>

[RootGroup]
Fields=<list of fields>

Here <list of field> - list of fields' IDs delimited by comma. Example: Fields=1,4,10,21

Custom conditions block

[Cond:<Index>]
ID=<number>
DF=<string>
SQLExpr=<string>
Values=<list of value parameters>

<list of value parameters> - list of <type>:<kind> pairs delimited by comma. Example:
Values=ftString:vkScalar, ftInteger:vkScalar

Special DB information section

[DBSpecInfo]
<any database related information>

For example for ADO connections this section can contain ConnectionString parameter:

[DBSpecInfo]
ConnectionString=Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\data\orders.mdb;Persist Security
Info=False

Database

An organized collection of information; a collection of related tables and queries in a given directory or on an SQL server.

Data types
Value**Description**

ftString	Character or string field
ftSmallint	16-bit integer field
ftInteger	32-bit integer field
ftWord	16-bit unsigned integer field
ftBoolean	Boolean field
ftFloat	Floating-point numeric field
ftCurrency	Money field
ftBCD	Binary-coded Decimal field
ftDate	Date field
ftTime	Time field
ftDateTime	Date and time field
ftAutoInc	Auto-incrementing 32-bit integer counter field
ftMemo	Text memo field
ftFmtMemo	Formatted text memo field

Editor masks

A mask consists of three fields with semicolons separating the fields. The first part of the mask is the mask itself. The second part is the character that determines whether the literal characters of a mask are saved as part of the data. The third part of the mask is the character used to represent unentered characters in the mask.

These are the special characters used in the first field of the mask:

Character	Meaning in mask
!	If a ! character appears in the mask, optional characters are represented in the text as leading blanks. If a ! character is not present, optional characters are represented in the text as trailing blanks.
>	If a > character appears in the mask, all characters that follow are in uppercase until the end of the mask or until a < character is encountered.
<	If a < character appears in the mask, all characters that follow are in lowercase until the end of the mask or until a > character is encountered.
<>	If these two characters appear together in a mask, no case checking is done and the data is formatted with the case the user uses to enter the data.
\	The character that follows a \ character is a literal character. Use this character to use any of the mask special characters as a literal in the data.
L	The L character requires an alphabetic character only in this position. For the US, this is A-Z, a-z.
I	The I character permits only an alphabetic character in this position, but doesn't require it.
A	The A character requires an alphanumeric character only in this position. For the US, this is A-Z, a-z, 0-9.
a	The a character permits an alphanumeric character in this position, but doesn't require it.
C	The C character requires an arbitrary character in this position.
c	The c character permits an arbitrary character in this position, but doesn't require it.
0	The 0 character requires a numeric character only in this position.
9	The 9 character permits a numeric character in this position, but doesn't require it.
#	The # character permits a numeric character or a plus or minus sign in this position, but doesn't require it.
:	The : character is used to separate hours, minutes, and seconds in times. If the character that separates hours, minutes, and seconds is different in the regional settings of the Control Panel utility on your computer system, that character is used instead.
/	The / character is used to separate months, days, and years in dates. If the character that separates months, days, and years is different in the regional settings of the Control Panel utility on your computer system, that character is used instead.
;	The ; character is used to separate the three fields of the mask.
_	The _ character automatically inserts spaces into the text. When the user enters characters in the field, the cursor skips the _ character.

Any character that does not appear in the preceding table can appear in the first part of the mask as a literal character. Literal characters must be matched exactly in the edit control. They are inserted automatically, and the cursor skips over them during editing. The special mask characters can also appear as literal characters if preceded by a backslash character (\).

The second field of the mask is a single character that indicates whether literal characters from the mask should be included as part of the text for the edit control. For example, the mask for a telephone number with area code could be the following string:

```
(000)_000-0000;0;*
```

The **0** in the second field indicates that the text for the edit control would consist of the 10 digits that were entered, rather than the 14 characters that make up the telephone number as it appears in the edit control.

A **0** in the second field indicates that literals should not be included, any other character indicates that they should be included.

The third field of the mask is the character that appears in the edit control for blanks (characters that have not been entered). By default, this is the same as the character that stands for literal spaces. The two characters appear the same in an edit window. However, when a user edits the text in a masked edit control, the cursor selects each blank character in turn, and skips over the space character.

Note

When working with multi byte character sets, such as Japanese shift-JIS, each special mask character represents a single byte. To specify double-byte characters using the **L**, **I**, **A**, **a**, **C**, or **c** specifiers, the mask characters must be doubled as well. For example, **LL** would represent two single-byte alphabetic characters or a one double-byte character. Only single-byte literal characters are supported.

What is Query Builder?

Query Builder is a tool providing an easy-to-use interface for building queries to databases.

It features:

- Storing the database structure and queries for representation in convenient way
- Visually building queries to database with Simple Query editor
- Instant viewing of resulting SQL queries and query results
- User friendly interface allowing non-experienced users to issue fairly sophisticated queries

SQLName format for operators

The SimpleQuery supports only binary conditional operations where the first parameter is the field and the second parameter is the constant value.

The format string of SQLName parameter for conditional operators can include the following:

@f - specify the place for field name;

@1 - specify the place for constant value.

The constant value must be enclosed in quotes for all types of values. For any type of values except the strings, dates or booleans the quotes will be ignored during the building of SQL statement.

Examples:

For the simple "is equal to" operator the format string is:

```
@f = '@1'
```

The "starting with" operator has the following format:

```
@f LIKE '@1%'
```

If you want to provide case insensitive "starting with" operator use such format string:

```
LOWER(@f) LIKE LOWER('@1%')
```

Value editor types

Editor type defines what will happen when user will try to edit value. There are the following types of editors:

- **Auto** the most appropriate value editor will be used depending on field's data type and operator which used in condition.
- **Edit** user will see the edit field for value editing
- **List** user will see the pop up menu with available values
- **Custom OnCustomValue** event will be generated when user selected the field with such value editor. You can use this type to show your own value edit;
- **SQL** similar to List editor type. Available values are returned by some SQL query. (First column of result set - values; second column - captions).

