Recordset: How Recordsets Select Records (ODBC)

Article • 08/02/2021

Note

The MFC ODBC Consumer wizard is not available in Visual Studio 2019 and later. You can still create a consumer manually.

This topic applies to the MFC ODBC classes.

This topic explains:

- Your role and your options in selecting records.
- How a recordset constructs its SQL statement and selects records.
- What you can do to customize the selection.

Recordsets select records from a data source through an ODBC driver by sending SQL statements to the driver. The SQL sent depends on how you design and open your recordset class.

Your Options in Selecting Records

The following table shows your options in selecting records.

How and When You Can Affect a Recordset

Expand table

When you	You can

When you	You can
Declare your recordset class with the **Add Class** wizard	Specify which table to select from. Specify which columns to include. See Adding an MFC ODBC Consumer.
Complete your recordset class implementation	Override member functions such as `OnSetOptions` (advanced) to set application-specific options or to change defaults. Specify parameter data members if you want a parameterized recordset.
Construct a recordset object (before you call `Open`)	Specify a search condition (possibly compound) for use in a **WHERE** clause that filters the records. See Recordset: Filtering Records (ODBC). Specify a sort order for use in an **ORDER BY** clause that sorts the records. See Recordset: Sorting Records (ODBC). Specify parameter values for any parameters you added to the class. See Recordset: Parameterizing a Recordset (ODBC).

|Run the recordset's query by calling `Open`|Specify a custom SQL string to replace the default SQL string set up by the wizard. See CRecordset::Open in the *Class Library Reference* and SQL: Customizing Your Recordset's SQL Statement (ODBC).|

|Call `Requery` to requery the recordset with the latest values on the data source|Specify new parameters, filter, or sort. See Recordset: Requerying a Recordset (ODBC).|

How a Recordset Constructs Its SQL Statement

When you call a recordset object's Open member function, `Open` constructs a SQL statement using some or all of the following ingredients:

- The *lpszSQL* parameter passed to `Open`. If not NULL, this parameter specifies a custom SQL string or part of one. The framework parses the string. If the string is a SQL **SELECT** statement or an ODBC **CALL** statement, the framework uses the string as the recordset's SQL statement. If the string does not begin with "SELECT" or "{CALL", the framework uses what is supplied to construct a SQL **FROM** clause.
- The string returned by GetDefaultSQL. By default, this is the name of the table you

specified for the recordset in the wizard, but you can change what the function returns. The framework calls `GetDefaultSQL` — if the string does not begin with "SELECT" or "{CALL", it is assumed to be a table name, which is used to construct a SQL string.

- The field data members of the recordset, which are to be bound to specific columns of the table. The framework binds record columns to the addresses of these members, using them as buffers. The framework determines the correlation of field data members to table columns from the RFX or Bulk RFX function calls in the recordset's DoFieldExchange or DoBulkFieldExchange member function.
- The filter for the recordset, if any, contained in the m_strFilter data member. The framework uses this string to construct a SQL **WHERE** clause.
- The sort order for the recordset, if any, contained in the m_strSort data member. The framework uses this string to construct a SQL **ORDER BY** clause.

  Tip

  To use the SQL **GROUP BY** clause (and possibly the **HAVING** clause), append the clauses to the end of your filter string.

- The values of any parameter data members you specify for the class. You set parameter values just before you call `Open` or `Requery`. The framework binds the parameter values to "?" placeholders in the SQL string. At compile time, you specify the string with placeholders. At run time, the framework fills in the details based on the parameter values you pass.

`Open` constructs a SQL **SELECT** statement from these ingredients. See Customizing the Selection for details about how the framework uses the ingredients.

After constructing the statement, `Open` sends the SQL to the ODBC Driver Manager (and the ODBC Cursor Library if it is in memory), which sends it on to the ODBC driver for the specific DBMS. The driver communicates with the DBMS to carry out the selection on the data source and fetches the first record. The framework loads the record into the field data members of the recordset.

You can use a combination of these techniques to open tables and to construct a query based on a join of multiple tables. With additional customization, you can call predefined queries (stored procedures), select table columns not known at design time and bind them to recordset fields or you can perform most other data-access tasks. Tasks you cannot accomplish by customizing recordsets can still be accomplished by calling ODBC API functions or directly executing SQL statements with CDatabase::ExecuteSQL.

Customizing the Selection

Besides supplying a filter, a sort order, or parameters, you can take the following actions to customize your recordset's selection:

- Pass a custom SQL string in *lpszSQL* when you call Open for the recordset. Anything you pass in *lpsqSQL* takes precedence over what the GetDefaultSQL member function returns.

  For more information, see SQL: Customizing Your Recordset's SQL Statement (ODBC), which describes the types of SQL statements (or partial statements) you can pass to `Open` and what the framework does with them.

  Note

  If the custom string you pass does not begin with "SELECT" or "{CALL", MFC assumes it contains a table name. This also applies to the next bulleted item.

- Alter the string that the wizard writes in your recordset's `GetDefaultSQL` member function. Edit the function's code to change what it returns. By default, the wizard writes a `GetDefaultSQL` function that returns a single table name.

  You can have `GetDefaultSQL` return any of the items that you can pass in the *lpszSQL* parameter to `Open`. If you do not pass a custom SQL string in *lpszSQL*, the framework uses the string that `GetDefaultSQL` returns. At a minimum, `GetDefaultSQL` must return a single table name. But you can have it return multiple table names, a full **SELECT** statement, an ODBC **CALL** statement, and so on. For a list of what you can pass to *lpszSQL* — or have `GetDefaultSQL` return — see SQL: Customizing Your Recordset's SQL Statement (ODBC).

  If you are performing a join of two or more tables, rewrite `GetDefaultSQL` to customize the table list used in the SQL **FROM** clause. For more information, see Recordset: Performing a Join (ODBC).

- Manually bind additional field data members, perhaps based on information you obtain about the schema of your data source at run time. You add field data members to the recordset class, RFX or Bulk RFX function calls for them to the DoFieldExchange or DoBulkFieldExchange member function, and initializations of the data members in the class constructor. For more information, see Recordset: Dynamically Binding Data Columns (ODBC).

- Override recordset member functions, such as `OnSetOptions`, to set application-specific options or to override defaults.

If you want to base the recordset on a complex SQL statement, you need to use some combination of these customization techniques. For example, perhaps you want to use SQL clauses and keywords not directly supported by recordsets or perhaps you are joining multiple tables.

See also

Recordset (ODBC)
Recordset: How Recordsets Update Records (ODBC)
ODBC Basics
SQL
Recordset: Locking Records (ODBC)