Recordset: Declaring a Class for a Predefined Query (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 how to create a recordset class for a predefined query (sometimes called a stored procedure, as in Microsoft SQL Server).

> **Note**
>
> This topic applies to objects derived from `CRecordset` in which bulk row fetching has not been implemented. If bulk row fetching is implemented, the process is very similar. To understand the differences between recordsets that implement bulk row fetching and those that do not, see **Recordset: Fetching Records in Bulk (ODBC)**.

Some database management systems (DBMSs) allow you to create a predefined query and call it from your programs like a function. The query has a name, might take parameters, and might return records. The procedure in this topic describes how to call a predefined query that returns records (and perhaps takes parameters).

The database classes do not support updating predefined queries. The difference between a snapshot predefined query and a dynaset predefined query is not updateability but whether changes made by other users (or other recordsets in your program) are visible in your recordset.

> **Tip**
>
> You do not need a recordset to call a predefined query that does not return records. Prepare the SQL statement as described below, but execute it by calling

the `CDatabase` member function **ExecuteSQL**.

You can create a single recordset class to manage calling a predefined query, but you must do some of the work yourself. The wizards do not support creating a class specifically for this purpose.

To create a class for calling a predefined query (stored procedure)

1. Use the MFC ODBC Consumer Wizard from **Add Class** to create a recordset class for the table that contributes the most columns returned by the query. This gives you a head start.

2. Manually add field data members for any columns of any tables that the query returns but that the wizard did not create for you.

   For example, if the query returns three columns each from two additional tables, add six field data members (of the appropriate data types) to the class.

3. Manually add RFX function calls in the DoFieldExchange member function of the class, one corresponding to the data type of each added field data member.

   C++

   ```
   Immediately before these RFX calls, call <MSHelp:link key-
   words="_mfc_CFieldExchange.3a3a.SetFieldType"
   TABINDEX="0">SetFieldType</MSHelp:link>, as shown here:
   pFX->SetFieldType( CFieldExchange::outputColumn );
   ```

   > **Note**
   >
   > You must know the data types and the order of columns returned in the result set. The order of RFX function calls in `DoFieldExchange` must match the order of result set columns.

4. Manually add initializations for the new field data members in the recordset class constructor.

   You must also increment the initialization value for the m_nFields data member. The wizard writes the initialization, but it only covers the field data members it

adds for you. For example:

```
C++

m_nFields += 6;
```

Some data types should not be initialized here, for example, `CLongBinary` or byte arrays.

5. If the query takes parameters, add a parameter data member for each parameter, an RFX function call for each, and an initialization for each.

6. You must increment `m_nParams` for each added parameter, as you did `m_nFields` for added fields in step 4 of this procedure. For more information, see Recordset: Parameterizing a Recordset (ODBC).

7. Manually write a SQL statement string with the following form:

```

{CALL proc-name [(? [, ?]...)]}
```

where **CALL** is an ODBC keyword, **proc-name** is the name of the query as it is known on the data source, and the "?" items are placeholders for the parameter values you supply to the recordset at run time (if any). The following example prepares a placeholder for one parameter:

```

CString mySQL = "{CALL Delinquent_Accts (?)}";
```

8. In the code that opens the recordset, set the values of the recordset's parameter data members and then call the `Open` member function, passing your SQL string for the *lpszSQL* parameter. Or instead, replace the string returned by the `GetDefaultSQL` member function in your class.

The following examples show the procedure for calling a predefined query, named `Delinquent_Accts`, which takes one parameter for a sales district number. This query returns three columns: `Acct_No`, `L_Name`, `Phone`. All columns are from the Customers

table.

The following recordset specifies field data members for the columns the query returns and a parameter for the sales district number requested at run time.

C++

```
class CDelinquents : public CRecordset
{
// Field/Param Data
    LONG m_lAcct_No;
    CString m_strL_Name;
    CString m_strPhone;
    LONG m_lDistParam;
    // ...
};
```

This class declaration is as the wizard writes it, except for the `m_lDistParam` member added manually. Other members are not shown here.

The next example shows the initializations for the data members in the `CDelinquents` constructor.

C++

```
CDelinquents::CDelinquents(CDatabase* pdb)
   : CRecordset(pdb)
{
    // Wizard-generated params:
    m_lAcct_No = 0;
    m_strL_Name = "";
    m_strPhone = "";
    m_nFields = 3;
    // User-defined params:
    m_nParams = 1;
    m_lDistParam = 0;
}
```

Note the initializations for m_nFields and m_nParams. The wizard initializes `m_nFields`; you initialize `m_nParams`.

The next example shows the RFX functions in `CDelinquents::DoFieldExchange`:

C++

```
void CDelinquents::DoFieldExchange(CFieldExchange* pFX)
{
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Long(pFX, "Acct_No", m_lAcct_No);
    RFX_Text(pFX, "L_Name", m_strL_Name);
    RFX_Text(pFX, "Phone", m_strPhone);
    pFX->SetFieldType(CFieldExchange::param);
    RFX_Long(pFX, "Dist_No", m_lDistParam);
}
```

Besides making the RFX calls for the three returned columns, this code manages binding the parameter you pass at run time. The parameter is keyed to the `Dist_No` (district number) column.

The next example shows how to set up the SQL string and how to use it to open the recordset.

C++

```
// Construct a CDelinquents recordset object
CDelinquents rsDel( NULL );
CString strSQL = "{CALL Delinquent_Accts (?)}"
// Specify a parameter value (obtained earlier from the user)
rsDel.m_lDistParam = lDistrict;
// Open the recordset and run the query
if( rsDel.Open( CRecordset::snapshot, strSQL ) )
    // Use the recordset ...
```

This code constructs a snapshot, passes it a parameter obtained earlier from the user, and calls the predefined query. When the query runs, it returns records for the specified sales district. Each record contains columns for the account number, customer's last name, and customer's phone number.

Tip

You might want to handle a return value (output parameter) from a stored procedure. For more information and an example, see **CFieldExchange::SetFieldType**.

See also

Recordset (ODBC)
Recordset: Requerying a Recordset (ODBC)
Recordset: Declaring a Class for a Table (ODBC)
Recordset: Performing a Join (ODBC)