Recordset: Fetching Records in Bulk (ODBC)

Article • 08/02/2021

This topic applies to the MFC ODBC classes.

Class `CRecordset` provides support for bulk row fetching, which means that multiple records can be retrieved at once during a single fetch, rather than retrieving one record at a time from the data source. You can implement bulk row fetching only in a derived `CRecordset` class. The process of transferring data from the data source to the recordset object is called bulk record field exchange (Bulk RFX). Note that if you are not using bulk row fetching in a `CRecordset`-derived class, data is transferred via record field exchange (RFX). For more information, see Record Field Exchange (RFX).

This topic explains:

- How CRecordset supports bulk row fetching.
- Some special considerations when using bulk row fetching.
- How to implement bulk record field exchange.

How CRecordset Supports Bulk Row Fetching

Before opening your recordset object, you can define a rowset size with the `SetRowsetSize` member function. The rowset size specifies how many records should be retrieved during a single fetch. When bulk row fetching is implemented, the default rowset size is 25. If bulk row fetching is not implemented, the rowset size remains fixed at 1.

After you have initialized the rowset size, call the Open member function. Here you must specify the `CRecordset::useMultiRowFetch` option of the *dwOptions* parameter to implement bulk row fetching. You can additionally set the `CRecordset::userAllocMultiRowBuffers` option. The bulk record field exchange mechanism uses arrays to store the multiple rows of data retrieved during a fetch. These storage buffers can be allocated automatically by the framework or you can allocate them manually. Specifying the `CRecordset::userAllocMultiRowBuffers` option means

that you will do the allocation.

The following table lists the member functions provided by `CRecordset` to support bulk row fetching.

Expand table

Member function	Description
CheckRowsetError	Virtual function that handles any errors that occur during fetching.
DoBulkFieldExchange	Implements bulk record field exchange. Called automatically to transfers multiple rows of data from the data source to the recordset object.
GetRowsetSize	Retrieves the current setting for the rowset size.
GetRowsFetched	Tells how many rows were actually retrieved after a given fetch. In most cases, this is the rowset size, unless an incomplete rowset was fetched.
GetRowStatus	Returns a fetch status for a particular row within a rowset.
RefreshRowset	Refreshes the data and status of a particular row within a rowset.
SetRowsetCursorPosition	Moves the cursor to a particular row within a rowset.
SetRowsetSize	Virtual function that changes the setting for the rowset size to the specified value.

Special Considerations

Although bulk row fetching is a performance gain, certain features operate differently. Before you decide to implement bulk row fetching, consider the following:

- The framework automatically calls the `DoBulkFieldExchange` member function to transfer data from the data source to the recordset object. However, data is not transferred from the recordset back to the data source. Calling the `AddNew`, `Edit`, `Delete`, or `Update` member functions results in a failed assertion. Although `CRecordset` currently does not provide a mechanism for updating bulk rows of data, you can write your own functions by using the ODBC API function `SQLSetPos`. For more information about `SQLSetPos`, see the ODBC Programmer's Reference.

- The member functions `IsDeleted`, `IsFieldDirty`, `IsFieldNull`, `IsFieldNullable`, `SetFieldDirty`, and `SetFieldNull` cannot be used on recordsets that implement bulk row fetching. However, you can call `GetRowStatus` in place of `IsDeleted`, and `GetODBCFieldInfo` in place of `IsFieldNullable`.

- The `Move` operations repositions your recordset by rowset. For example, suppose you open a recordset that has 100 records with an initial rowset size of 10. `Open` fetches rows 1 through 10, with the current record positioned on row 1. A call to `MoveNext` fetches the next rowset, not the next row. This rowset consists of rows 11 through 20, with the current record positioned on row 11. Note that `MoveNext` and `Move( 1 )` are not equivalent when bulk row fetching is implemented. `Move( 1 )` fetches the rowset that begins 1 row from the current record. In this example, calling `Move( 1 )` after calling `Open` fetches the rowset consisting of rows 2 through 11, with the current record positioned on row 2. For more information, see the Move member function.

- Unlike record field exchange, the wizards do not support bulk record field exchange. This means that you must manually declare your field data members and manually override `DoBulkFieldExchange` by writing calls to the Bulk RFX functions. For more information, see Record Field Exchange Functions in the *Class Library Reference*.

How to Implement Bulk Record Field Exchange

Bulk record field exchange transfers a rowset of data from the data source to the recordset object. The Bulk RFX functions use arrays to store this data, as well as arrays to store the length of each data item in the rowset. In your class definition, you must define your field data members as pointers to access the arrays of data. In addition, you must define a set of pointers to access the arrays of lengths. Any parameter data members should not be declared as pointers; declaring parameter data members when using bulk record field exchange is the same as declaring them when using record field exchange. The following code shows a simple example:

C++

```
class MultiRowSet : public CRecordset
{
public:
```

```
   // Field/Param Data
      // field data members
      long* m_rgID;
      LPSTR m_rgName;

      // pointers for the lengths
      // of the field data
      long* m_rgIDLengths;
      long* m_rgNameLengths;

      // input parameter data member
      CString m_strNameParam;

   .
   .
   .
}
```

You can either allocate these storage buffers manually or have the framework do the allocation. To allocate the buffers yourself, you must specify the `CRecordset::userAllocMultiRowBuffers` option of the *dwOptions* parameter in the `Open` member function. Be sure to set the sizes of the arrays at least equal to the rowset size. If you want to have the framework do the allocation, you should initialize your pointers to NULL. This is typically done in the recordset object's constructor:

C++

```
MultiRowSet::MultiRowSet( CDatabase* pDB )
   : CRecordset( pDB )
{
   m_rgID = NULL;
   m_rgName = NULL;
   m_rgIDLengths = NULL;
   m_rgNameLengths = NULL;
   m_strNameParam = "";

   m_nFields = 2;
   m_nParams = 1;

   .
   .
   .
}
```

Finally, you must override the `DoBulkFieldExchange` member function. For the field data

members, call the Bulk RFX functions; for any parameter data members, call the RFX functions. If you opened the recordset by passing a SQL statement or stored procedure to `Open`, the order in which you make the Bulk RFX calls must correspond to the order of the columns in the recordset; similarly, the order of the RFX calls for parameters must correspond to the order of parameters in the SQL statement or stored procedure.

C++

```
void MultiRowSet::DoBulkFieldExchange( CFieldExchange* pFX )
{
   // call the Bulk RFX functions
   // for field data members
   pFX->SetFieldType( CFieldExchange::outputColumn );
   RFX_Long_Bulk( pFX, _T( "[colRecID]" ),
                  &m_rgID, &m_rgIDLengths );
   RFX_Text_Bulk( pFX, _T( "[colName]" ),
                  &m_rgName, &m_rgNameLengths, 30 );

   // call the RFX functions for
   // for parameter data members
   pFX->SetFieldType( CFieldExchange::inputParam );
   RFX_Text( pFX, "NameParam", m_strNameParam );
}
```

> **Note**
>
> You must call the `Close` member function before your derived `CRecordset` class goes out of scope. This ensures that any memory allocated by the framework are freed. It is good programming practice to always explicitly call `Close`, regardless of whether you have implemented bulk row fetching.

For more information about record field exchange (RFX), see Record Field Exchange: How RFX Works. For more information about using parameters, see CFieldExchange::SetFieldType and Recordset: Parameterizing a Recordset (ODBC).

See also

Recordset (ODBC)
CRecordset::m_nFields
CRecordset::m_nParams