Recordset: Adding, Updating, and Deleting Records (ODBC)

Article • 08/02/2021

This topic applies to the MFC ODBC classes.

> **Note**
>
> You can now add records in bulk more efficiently. For more information, see **Recordset: Adding Records in Bulk (ODBC)**.

> **Note**
>
> This topic applies to objects derived from `CRecordset` in which bulk row fetching has not been implemented. If you are using bulk row fetching, see **Recordset: Fetching Records in Bulk (ODBC)**.

Updateable snapshots and dynasets allow you to add, edit (update), and delete records. This topic explains:

- How to determine whether your recordset is updatable.
- How to add a new record.
- How to edit an existing record.
- How to delete a record.

For more information about how updates are carried out and how your updates appear to other users, see Recordset: How Recordsets Update Records (ODBC). Normally, when you add, edit, or delete a record, the recordset changes the data source immediately. You can instead batch groups of related updates into transactions. If a transaction is in progress, the update does not become final until you commit the transaction. This allows you to take back or roll back the changes. For information about transactions, see Transaction (ODBC).

The following table summarizes the options available for recordsets with different update characteristics.

Recordset Read/Update Options

Expand table

Type	Read	Edit record	Delete record	Add new (append)
Read-only	Y	N	N	N
Append-only	Y	N	N	Y
Fully updatable	Y	Y	Y	Y

Determining Whether Your Recordset is Updateable

A recordset object is updateable if the data source is updateable and you opened the recordset as updateable. Its updateability also depends on the SQL statement you use, the capabilities of your ODBC driver, and whether the ODBC Cursor Library is in memory. You cannot update a read-only recordset or data source.

To determine whether your recordset is updatable

1. Call the recordset object's CanUpdate member function.

   `CanUpdate` returns a nonzero value if the recordset is updateable.

By default, recordsets are fully updateable (you can perform `AddNew`, `Edit`, and `Delete` operations). But you can also use the appendOnly option to open updateable recordsets. A recordset opened this way allows only the addition of new records with `AddNew`. You cannot edit or delete existing records. You can test whether a recordset is open only for appending by calling the CanAppend member function. `CanAppend` returns a nonzero value if the recordset is either fully updateable or open only for appending.

The following code shows how you might use `CanUpdate` for a recordset object called `rsStudentSet`:

```C++
if( !rsStudentSet.Open( ) )
    return FALSE;
if( !rsStudentSet.CanUpdate( ) )
{
    AfxMessageBox( "Unable to update the Student recordset." );
    return;
}
```

Caution

When you prepare to update a recordset by calling `Update`, take care that your recordset includes all columns making up the primary key of the table (or all of the columns of any unique index on the table). In some cases, the framework can use only the columns selected in your recordset to identify which record in your table to update. Without all the necessary columns, multiple records might be updated in the table, possibly damaging the referential integrity of the table. In this case, the framework throws exceptions when you call `Update`.

Adding a Record to a Recordset

You can add new records to a recordset if its CanAppend member function returns a nonzero value.

To add a new record to a recordset

1. Make sure the recordset is appendable.

2. Call the recordset object's AddNew member function.

   `AddNew` prepares the recordset to act as an edit buffer. All field data members are set to the special value Null and marked as unchanged so only changed (dirty) values are written to the data source when you call Update.

3. Set the values of the new record's field data members.

   Assign values to the field data members. Those you do not assign are not written to the data source.

4. Call the recordset object's `Update` member function.

   `Update` completes the addition by writing the new record to the data source. For information about happens if you fail to call `Update`, see Recordset: How Recordsets Update Records (ODBC).

For information about how adding records works and about when added records are visible in your recordset, see Recordset: How AddNew, Edit, and Delete Work (ODBC).

The following example shows how to add a new record:

C++

```cpp
if( !rsStudent.Open( ) )
    return FALSE;
if( !rsStudent.CanAppend( ) )
    return FALSE;                      // no field values were set
rsStudent.AddNew( );
rsStudent.m_strName = strName;
rsStudent.m_strCity = strCity;
rsStudent.m_strStreet = strStreet;
if( !rsStudent.Update( ) )
{
    AfxMessageBox( "Record not added; no field values were set." );
    return FALSE;
}
```

Tip

To cancel an `AddNew` or `Edit` call, simply make another call to `AddNew` or `Edit` or call `Move` with the *AFX_MOVE_REFRESH* parameter. Data members are reset to their previous values and you are still in `Edit` or `Add` mode.

Editing a Record in a Recordset

You can edit existing records if your recordset's CanUpdate member function returns a nonzero value.

To edit an existing record in a recordset

1. Make sure the recordset is updateable.

2. Scroll to the record you want to update.

3. Call the recordset object's Edit member function.

   `Edit` prepares the recordset to act as an edit buffer. All field data members are marked so that the recordset can tell later whether they were changed. The new values for changed field data members are written to the data source when you call Update.

4. Set the values of the new record's field data members.

   Assign values to the field data members. Those you do not assign values remain unchanged.

5. Call the recordset object's `Update` member function.

   `Update` completes the edit by writing the changed record to the data source. For information about happens if you fail to call `Update`, see Recordset: How Recordsets Update Records (ODBC).

After you edit a record, the edited record remains the current record.

The following example shows an `Edit` operation. It assumes the user has moved to a record he or she wants to edit.

C++

```cpp
rsStudent.Edit( );
rsStudent.m_strStreet = strNewStreet;
rsStudent.m_strCity = strNewCity;
rsStudent.m_strState = strNewState;
rsStudent.m_strPostalCode = strNewPostalCode;
if( !rsStudent.Update( ) )
{
    AfxMessageBox( "Record not updated; no field values were set." );
    return FALSE;
}
```

Tip

To cancel an `AddNew` or `Edit` call, simply make another call to `AddNew` or `Edit` or call `Move` with the *AFX_MOVE_REFRESH* parameter. Data members are reset to their previous values and you are still in `Edit` or `Add` mode.

Deleting a Record from a Recordset

You can delete records if your recordset's CanUpdate member function returns a nonzero value.

To delete a record

1. Make sure the recordset is updateable.

2. Scroll to the record you want to update.

3. Call the recordset object's Delete member function.

   `Delete` immediately marks the record as deleted, both in the recordset and on the data source.

   Unlike `AddNew` and `Edit`, `Delete` has no corresponding `Update` call.

4. Scroll to another record.

   Note

   When moving through the recordset, deleted records might not be skipped. For more information, see the **IsDeleted** member function.

The following example shows a `Delete` operation. It assumes that the user has moved to a record that the user wants to delete. After `Delete` is called, it is important to move to a new record.

```
rsStudent.Delete( );
rsStudent.MoveNext( );
```

For more information about the effects of the `AddNew`, `Edit`, and `Delete` member functions, see Recordset: How Recordsets Update Records (ODBC).

See also

Recordset (ODBC)
Recordset: Locking Records (ODBC)