Use the following best practices.

* Have a delete action on every relation between two tables.
* Use table delete actions instead of writing code to specify whether deletes are restricted or cascaded.

**Add a Delete Action**

1. In the Application Object Tree (AOT), expand the **Data Dictionary**.
2. Expand **Tables**, and then locate the table that you want to add a delete action to.
3. Click the table, right-click **DeleteActions**, and then click **New DeleteAction**.
4. Right-click the new delete action, and then click **Properties**.
5. Select a related table from the **Table** property list.
6. Set the **DeleteAction** property. The following table describes the available values.

| **Delete Action** | **Description** | **Comments** |
| --- | --- | --- |
| None | Delete action disabled |  |
| Cascade | Deletes related records. | Setting the DeleteAction property to Cascade extends the functionality of the table's delete method. As a result, super(), in delete, initiates a cascaded deletion, propagating the delete from table to table.  A cascaded delete is implicitly protected by tts. Database changes aren't committed until the entire transaction is complete.  Example  On the CustTable table, a cascading delete action has been defined for the CustBankAccount table. When a customer is deleted from the CustTable table, the delete method also ensures that the corresponding bank account information is automatically deleted. |
| Restricted | Restricts deletion in the current table if data is present in related tables. | Setting the DeleteAction property to Restricted extends the functionality of the table's validateDelete method.  As a result, super(), in validateDelete, checks whether records exist on related tables. If records do exist, validateDelete returns false. The forms system ensures that the deletion is not performed. In your own X++ code, check the return value of validateDelete. Don't delete the primary or related records if the method returns false.  Example  On the CustTable table, a restricted delete action has been defined for the CustTrans table. When a customer is deleted in the CustTable table, the validateDelete method ascertains whether transactions exist for the customer in the CustTrans table. If so, validateDelete returns false. |
| Cascade+Restricted | Cascade the delete, even though records exist on related tables. | Setting the DeleteAction property to Cascade+Restricted extends the functionality of the table's validateDelete and delete methods.  As a result, super(), in validateDelete, ascertains whether records exist on related tables. Whether deleting records from forms or X++, if validateDelete returns false, the primary record isn't deleted and the cascading delete isn't performed. You should first delete the records in the related table before deleting the primary record.  If the primary record is being deleted as part of a cascading delete, the primary record and the records in the related table will be deleted.  Example  The Cascade+Restricted delete action is used in the standard application for LedgerJournalTrans on LedgerJournalTable.  This type of delete action is useful when you prefer a total clean-up—when you delete a customer, you also delete all the transactions associated with that customer. |