



January's Tip - "Optimistic" Concurrency

This month's tip was submitted by Club Tech iSeries courtesy of iSeriesnetwork.com

There are two predominant approaches to preventing conflicting updates when multiple jobs access the same data.

1. **"Pessimistic"** - an application locks records as read and releases the locks only after the user completes or aborts the transaction. For interactive or Web applications, records may remain locked during user "think" time, potentially preventing access by others.
2. **"Optimistic"** - records aren't locked when they're first read. Instead, the application checks that records aren't changed between the time they're first retrieved and the point at which the transaction is to be completed (i.e. updated). Because optimistic concurrency techniques minimize lock duration, they're the better approach for many situations, especially for Web applications. And, coding optimistic concurrency is fairly easy in SQL.

Here are the basic steps:

- a. Read data without a lock.
- b. Make a copy of the data.
- c. Present data to the user for possible changes.
- d. When the user is ready to complete the transaction, compare the copy of the original data to the current database contents:
 - If these are equal, update the database with the user's changes
 - If the database has changed, restart the transaction

Here is a very simple example. Assume the application lets the user view and modify a customer's name and address. To retrieve data from a single row without a lock, use a Select Into statement, such as:

```
Select CustId, Name, Address, City, State, PostalCode
Into :OldCustId, :OldName, :OldAddress, :OldCity, :OldState, :OldPostalCode
From Customer
Where CustId = :RqsCustId
```

This assumes the :RqsCustId host variable has been set with the requested customer ID. The retrieved values are put in host variables whose names begin with "Old" to indicate these are the old, or original, values.

Next, use high-level language statements to assign all the OldXxx variables to NewXxx variables and display these to the user. When the user changes one or more of these values and indicates that they want to complete the update, execute the following Update statement:

```
Update Customer
Set Name=:NewName, Address = :NewAddress, City = :NewCity, State = :NewState, PostalCode = :NewPostalCode
Where CustId = :RqsCustId
And Name = :OldName
And Address = :OldAddress
And City = :OldCity
And State = :OldState
And PostalCode = :OldPostalCode
```

If this statement completes successfully (SqlState is '00000'), then execute a Commit statement (if using commitment control) to commit the transaction and release the row lock (placed by the Update statement).

Otherwise, an SqlState of '02000' indicates no row was found that satisfied the search condition. This means either the row for that customer was deleted or one of the name or address fields was changed. Other SqlState values indicate a warning or error and should be handled appropriately. If the Update statement isn't successful, you should execute a rollback and restart the transaction by attempting to re-retrieve the changed data and present it to the user with an explanatory message.

Note: check out this link for a nice explanation of commitment control (Transaction Isolation Levels):

<http://www.iseriesnetwork.com/resources/clubtech/index.cfm?fuseaction=ShowNewsletterIssue&ID=10913>

If you submit a tip to tips@drewcorp.com, and it is selected for a TIP of the month, I may buy you a cup of coffee!!!! Please send comments of the types of tips you would like to see to Andy_Johnson@drewcorp.com.