Data Entry and Maintenance with SAS/IntrNet Software:

The Locking Techniques

Wilfried Schollenberger
WS Unternehmensberatung und Controlling-Systeme GmbH, Heidelberg

Abstract

This paper is based on the requirements and experiences at EURESCOM. The SAS/IntrNet based Project Management System requires capturing and maintenance of information via the Internet and the local Intranet. In order to provide secure updates, it is necessary to lock areas of the database, while somebody is working on them. The presentation introduces various requirements and the appropriate solutions.

Introduction

Every application that includes data entry or maintenance must provide some means to manage the different transactions, so that the different attempts to update data do not disturb each other. In batch processing, it is often very simple, because whole tables (or SAS-Libraries in MVS) are allocated by a batch job. On the other side, online-applications require a transaction processing which locks only the area of the tables, which the user is going to update. Basically it is a three step approach:

1. lock the area for the user
2. perform the work, human and technical
3. release the lock

Additionally we can distinguish between technical processing to ensure that the update process can proceed without problems, and logical locks to reserve an area in the database for one user, and to ensure that users do not innocently overwrite their inputs. This paper deals with logical locks.

In many cases, simple record locking is not sufficient, because the application requires an update of a number of tables and records in one big step. An easier way is, to lock a logical area via a locking table. This approach requires, that you can determine the different areas, and that these areas do not overlap. Then you can create a locking table where the application enters and releases the lock for the appropriate area. In principle the abstract code may look like this:

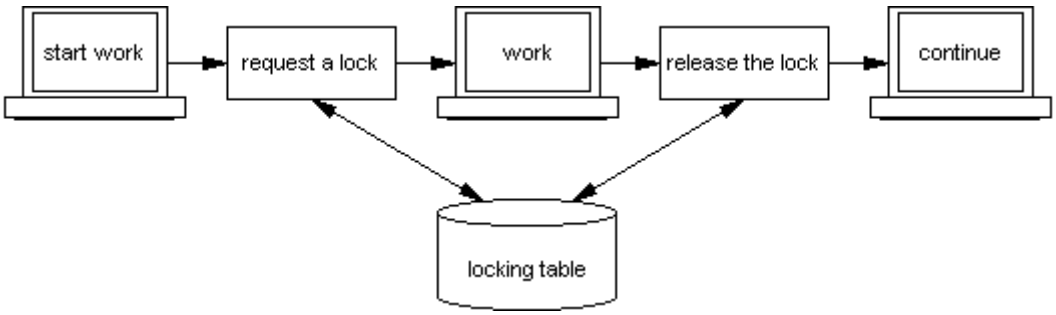
```
* Request a lock for Area "xyz";
* If not successful return with error;
* --- work and update the data ---;
* Release the lock;
```

If you use a locking table, the abstract code to request and release the lock may look like this:

```
request_a_lock: char: rUserId, char: rArea
* open the locking table for update, retry if not successful;
* search for record with area = rArea;
* if record found and userId NE rUserId, return with error "area is locked by user";
* insert record with rUserId and rArea;
* close the locking table and return "successful";
```

release_a_lock: char: rUserId, char: rArea
* open the locking table for update, retry if not successful;
* search for record with area = rArea and userId = rUserId;
* if record found, delete the record;
* close the locking table and return "successful";

Graphically, the procedure may look like this:



With SAS/IntrNet applications logical locks become more complicated, because we cannot force the user to complete the procedure and release the lock. If he closes the browser window, or simply branches to another URL after he received a lock, the server will not get any notification. The solution to this problem depends on the type of application. At EURESCOM we currently have three applications with different solutions for this locking problem:
1. maintenance of budget figures with a Java applet
2. reporting of Work Summaries via the Internet
3. maintenance of keywords for the keyword search application

Basics

In principle, each application uses the same mechanism to get a lock and to perform the update:

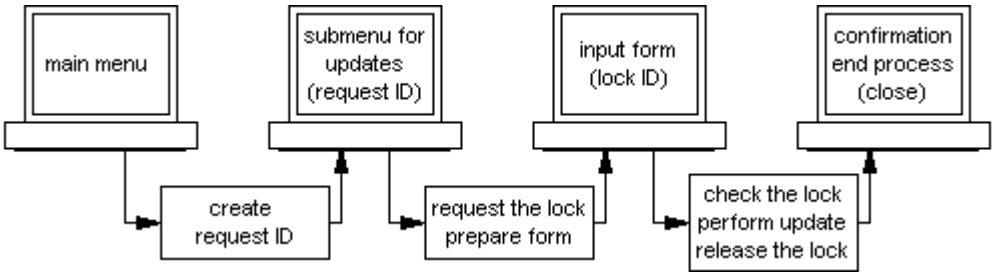
- When the user attempts to enter data, the application requests a lock for a named area, e.g. "Budget Figures for Project 908". If this area is not already locked, a lock ID is created and the lock is entered in a locking table. So all SAS/IntrNet servers access the same locking table and share the locking information. The lock ID is passed to the application and becomes a parameter of the next HTTP request.
- When the user has entered his information and submits the update request to the server, the server checks whether the lock ID is still valid. Then, it performs the update and deletes the entry in the locking table.

Every lock has a datetime value for timeout. If the locking method finds a lock record with a timeout value which is less than the current date and time, that record is deleted. In this way, a lock expires only when another user attempts to update the same area. Depending on the time, the user needs for his work, and the value of his input, the lifetime of a lock varies from 5 minutes to 24 hours.

Sometimes, the user may request an HTML page which allows some update, leave it, change his mind, and recall it again. In this case, we have to keep track, by whom and from where the HTML page is requested. In many cases, we could use the user ID from the HTTP server authentication (_RMTUSER). But in order to recognise reloads and similar situations, we create a request ID for the page from which the whole process is initiated. So the request for a lock may overwrite a previous lock with the same request ID. But only one lock is valid at a time.

The request ID can be composed from the user ID and the datetime()-value or from the server name and the datetime()-value when we do not have a user ID to compose a request ID.

Altogether an update procedure may have three HTTP-Transactions:



The abstract code to request, check, and release a lock may look like this:

request_a_lock: char: rRequestId, char: rArea, num: timeout
* open the locking table for update, retry if not successful;
* compose lockId
* compose datetime of expiration (e.g. 1 hour from now);
* search for record with area = rArea;
* if record NOT found, insert record, close the locking table, and return lockId;
* if requestId NE rRequestID AND expiraton GT now(), return with error "area is locked by user";
* update record with rRequestId and new expiration value;
* close the locking table and return lockId;

check_a_lock: char: cLockId
* open the locking table for update, retry if not successful;
* search for record with lockId = cLockId;
* if record NOT found, close the locking table and return "ERROR";
* compose the new datetime of expiration and update the record;
* close the locking table and return "OK";

release_a_lock: char: rLockId
* open the locking table for update, retry if not successful;
* search for record with lockId = rLockId;
* if record found, delete the record;
* close the locking table and return "successful";

Now, let's have a lock at the different applications.

Maintenance of Budget Figures with a Java Applet

This application is currently under development, but it is the easiest case, and therefore I want to start with this one. The Project Supervisor maintains planned figures for a project, and when a revised contract is prepared, this data may be copied into the tables for "contracted figures". Every project is allocated to one Supervisor, but another person is responsible for the preparation of contracts and the copy management to "contracted figures". So the main task is, to avoid that somebody works on the data, while it should be copied to "contracted". There also may be rare situation where another person than the supervisor has to modify some data, e.g. allocate some budget to a task for a project participant, so that the participant can report on this task.

The input is handled by a Java applet because there is a lot of processing necessary on the client side. The biggest advantage of the applet is, that it opens and closes windows (Frames), and that it can request and release locks when a window is opened or closed.

In this case, we use the user ID as request ID. The lock is requested together with the data, when a window is opened. Updates do not release the lock, in order to allow intermediate saves. It is released when the window is closed by a special unlock request.

Since we do not expect attempts to update the date by another person, the timeout value is not of much importance. It may be anywhere between 3 and 12 hours. We are still evaluating the best value. The biggest advantage of a Java applet is, beside the processing capabilities, that it can refresh the look by itself. E.g. if the timeout value would be one hour, it could send automatically every 45 minutes a little HTTP-request to refresh the lock.

Reporting of Work Summaries via the Internet

A Quarterly Work Summary Report is a comprehensive report about the work of the last quarter. Each participating shareholder in a project has to prepare it. So it might happen, that two participants try to enter the data at almost the same time. This has to be handled by the application.

Later the report has to be accepted by the project supervisor. The supervisor must be sure that he sees the very last version, and that nobody submits changes while he is going to approve the report.

Since entering all the information requires a lot of time, the reliability of the application is very important. There are a number of ways, how a user could loose his input:

- The lock can expire before the user has finished entering the data. Therefore we use a high timeout value of 24 hours.
- When the user has submitted his input, he might go back via the browser navigation keys to the input form and continue to enter or modify data. In this case, the lock would be invalid.
- Avoiding that type of error by preventing the browser to cache the input form causes the Netscape browsers to reload the page from the server, when the user resizes the window. Then the user would loose his input as well.

With the first pilot we also saw, that users looked at the form and closed the window. In this way, they disabled the data entry for everybody else until the timeout expired.

To circumvent all these problems, we took a lot of care to a proper handling:

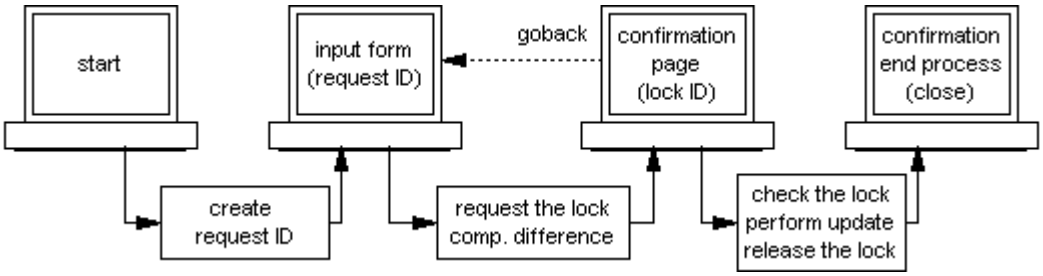
- Like in the Budget Figures application, we take the user ID as request ID, so that the user cannot lockout himself.
- We open a new browser window for the input form, so that the user does not accidentally navigate back.
- The input form contains a "submit" and a "close and unlock" button. The "close and unlock" button issues an unlock request which sends a HTML page to close the browser window via JavaScript.
- The "submit" button sends the input to the server which redisplays the information in a confirmation page. From there the user can go back via the navigation of the browser in order to correct his input. When he confirms his input, the window is closed by an HTML page and JavaScript.
- When the area is locked by another user, we display the full name of that user, so that he can be called by phone and release the lock.

A Supervisor who attempts to accept a report gets a 5 minutes lock only. If really somebody tries to change the report at the same time, he was to early and the approval would fail. Since he does not make any additional input, this failure does not matter. He can easily reload the page, check the modified figures, and approve the new report.

Maintenance of Keywords for the Keyword Search Application

In the keyword search application, the Project Supervisor adds or removes predefined keywords to/from a project or deliverable. In this case, the input is a set of mouse clicks, and the whole procedure is quite different.

- In the first step, the user gets the input form with the current settings and a unique request ID, composed of the server ID and the datetime of the request.
- When he submits the form, he gets the lock, and the difference between his input and the current settings is computed and displayed.
- Now the user can revoke each change and submit the form in order to perform the changes in the database.



In this way, the timeout for the lock can be very short (5 minutes). Nobody will lockout anybody else for a long time. If the lock has timed out or was not available, the user can go back to his original input form and resubmit his input. He always gets the difference to the current state and can revoke unwanted changes.

Conclusions

Data capture and maintenance via an Intranet or the Internet require some additional efforts to provide secure procedures. It is quite similar to client server processing with SAS/Connect without using Remote Library Services. With SAS/Connect, we had to take into account, that the connection could break or the server session could be closed by MVS due to timeout settings.

Java applets provide the easiest way to avoid unwanted locks when the client application terminated improperly. An applet can send HTTP-requests by itself in order to refresh a lock.

If the application is based on HTML, the appropriate compromise depends on the application. If a lot of information has to be entered, we need long timeout values and probably some means to clear a lock which was left by an improperly terminated client. If the amount of input is limited, it depends on the probability, that anybody else tries to maintain the same data at the same time. In some cases, the real update can be shifted to a confirmation page which shows the difference between the database and the input of the user.

The handling can be improved and some errors can be avoided by opening a new browser window.

Contact

Wilfried Schollenberger
WS Unternehmensberatung und Controlling-Systeme GmbH
Bergstraße 7
69120 Heidelberg
Tel: 06221 / 401 409
EMail: wisch@attglobal.net