# STORED PROCEDURES AND TRIGGERS

**DOUG MORRIS, COMPUTER SYSTEM INNOVATIONS, INC.**

<table>
<thead>
<tr>
<th>AGENDA</th>
<th>STORED PROCEDURES AND TRIGGERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stored Procedures</td>
<td>What</td>
</tr>
<tr>
<td></td>
<td>When</td>
</tr>
<tr>
<td></td>
<td>How</td>
</tr>
<tr>
<td>Triggers</td>
<td>What</td>
</tr>
<tr>
<td></td>
<td>When</td>
</tr>
<tr>
<td></td>
<td>How</td>
</tr>
<tr>
<td>Tips to leave with</td>
<td></td>
</tr>
<tr>
<td>Special Thanks to Bruce Wilson of RSM</td>
<td></td>
</tr>
</tbody>
</table>
STORED PROCEDURES - WHAT

• Saved script of SQL commands, run as a unit
• Similar to “function call” or subroutine in other languages
  – Accepts parameters, returns parameters
  – Can also produce result sets
• Stored procedures are saved inside the database

STORED PROCEDURES - WHEN

DEGREES OF INTRUSIVENESS

1. Assemble data and output as result set (more complex than a view can handle)
2. System administration
3. Insert/update/delete non-iMIS tables
4. Update iMIS tables
5. Insert or delete in iMIS tables
• Behavior is similar to a view
• Output can be sorted, can use "compute"
• May output more than one result set
• May not be joined to other tables
• May use parameters to limit scope of data being handled

create procedure csi_sp_ShowRecentUsers
as
begin
    set nocount on

    select UserId, UserName, LastLogin
    from Users
    order by LastLogin desc
end
go
grant exec on csi_sp_ShowRecentUsers to IMIS
**OUTPUT ONLY PROCEDURE**

```
cREATE PROCEDURE csi_sp_ShowMembersNotRegistered
AS
BEGIN
    DECLARE @tPeople TABLE (ID VARCHAR(10))
    INSERT @tPeople
    SELECT n.ID
    FROM Name n
    WHERE n.MEMBER_RECORD = 1
    DELETE p
    FROM @tPeople p
    JOIN Orders o
        ON o.ST_ID = p.ID
    JOIN Order_Meet om
        ON om.ORDER_NUMBER = o.ORDER_NUMBER
    WHERE om.MEETING = 'CONF08'
    SELECT n.ID, n.FULL_NAME, n.EMAIL
    FROM @tPeople p
    JOIN Name n
        ON n.ID = p.ID
END
---and what is missing here?
```

**NOTES FROM THE FIELD**

**TEMP TABLES**

- #temp tables are created in tempdb, are slower but can be much larger
- @table vars live only in memory, faster but smaller
- Both can be used anywhere a table can
- Cannot contain "text" or "image" types
- Create all as early in the procedure as you can
- ##globaltemp tables are visible to all processes - don't use
NOTES FROM THE FIELD

DOUG LIKES

• #temp tables over @table vars
• Because that’s what he grew up with!

POPULATING A SUMMARY FIELD

• Create a Customizer field for your summary
• Calculate and overwrite the data
• Schedule procedure as needed to keep it fresh
• Example: populate Name_Demo.STAFF_SIZE by counting people linked to this company
create procedure csi_sp_UpdateStaffSize as
begin

-- make a temp table
Create Table #TotalStaff (ID varchar(10), STAFF_SIZE int)

-- total up count of staff per ID, Member's only!
Insert #TotalStaff(ID, STAFF_SIZE)
Select CO_ID, count(*)
From Name
Where MEMBER_RECORD=1 and CO_ID > ' '
group by CO_ID

-- Log your changes!
insert Name_Log (DATE_TIME, LOG_TYPE, SUB_TYPE, USER_ID, ID,
LOG_TEXT)
Select getdate(), 'CHANGE', 'CHANGE', 'csi_sp_StaffSize', nd.ID,
'Name_Demo.STAFF_SIZE: ' + convert(varchar(10),nd.STAFF_SIZE) + ' -> '
+ convert(varchar(10), t.STAFF_SIZE)
From Name_Demo nd
Join #TotalStaff t on nd.ID=t.ID
Where nd.STAFF_SIZE<>t.STAFF_SIZE
CSI_SP_UPDATESTAFFSIZE

-- now update the actual data in iMIS

Update nd
Set nd.STAFF_SIZE = t.STAFF_SIZE
From Name_Demo nd
Join #TotalStaff t on nd.id = t.id
Where nd.STAFF_SIZE <> t.STAFF_SIZE -- this line is important - don't waste updates when not needed

drop table #TotalStaff -- clean up!
end

Grant Exec on csi_sp_UpdateStaffSize to iMIS

NOTES FROM THE FIELD - UPDATING IMIS

• Warning: no lifeguard on duty
  – The reality is many organizations do it
• Pick your battles carefully and stay as close to the farm as you can
• Update in places iMIS isn't very particular
  – Customizer, Activity, descriptive fields
• Mimic exactly unless you have a reason not to
• Log incessantly, think about how to "undo"
• Testing and backups are essential
NOTES FROM THE FIELD - NAMING AND OTHER CONVENTIONS

- Table, view names are nouns, procedure names start with verbs, functions are usually nouns
- Don’t start with sp_, create your own prefix (e.g. csi_ or abc_)
- Every name should be descriptive and unambiguous
- Table nicknames are the capital letters from table name (E.G. Name_Address = na)
- Remember, the person dealing with your stored procedure down the road, may not be you - add comments and think about how it looks to the outside world - sometimes an elegant script is not easy to read!
- Indent, space, keep it clean..

PARAMETERS

- Use the procedure for more than one thing
- Pass in information to control what the procedure does
  - Which ID or Meeting to use
  - What mode to work in
  - Whether to update or just display
  - Limit access to one row/group at a time for performance
- Use Crystal to prompt for the parameter
create procedure csi_sp_ShowMembersNotRegistered
    @psMeeting varchar(10) not NULL
as
begin
...
    join Order_Meet om
    on om.ORDRhower NUMBER = o.ORDRhower NUMBER
    where om.MEETING = @psMeeting

PARAMETERS

- Create the table outside of the procedure
- Don't insert using "select *" or without specifying target fields
- Don't use "select into"
- Use temp tables (@temp or #temp) for interim work tables
CREATING NON-I MIS TABLES

- Use a script, not the GUI
- Good basic fields:
  - `SEQN int identity` (if you don't have a natural key)
  - `ADDED_DATE datetime not NULL default getdate()`
  - `ADDED_BY nvarchar(128) not NULL default USER_NAME()`
  - `STATUS int not NULL default 0` (can be useful to track records for other processing)
- Grant permissions

NOTES FROM THE FIELD - CREATING I MIS COMPATIBLE TABLES

- TableName in mixed case
- `FIELD_NAME` in upper with underscores
- not NULL except datetime and text
- Match iMIS lengths
- Define a primary key with clustered index
- Include relevant keys for appropriate linking
- Don't duplicate non-key data unless you want a history
MEMBERSHIP COUNTS SNAPSHOT

• Custom table with snapshot date, member type, category, chapter and count
• Procedure run monthly to capture numbers

create table csi_MembershipCounts
(
    SEQN int identity,
    SNAPSHOT_DATE datetime not NULL,  -- required
    MEMBER_TYPE varchar(5) not NULL default '',
    CATEGORY varchar(5) not NULL default '',
    CHAPTER varchar(15) not NULL default '',
    QUANTITY int
)
go
alter table csi_MembershipCounts
    add PK_MembershipCounts primary key clustered (SEQN)
go
create index iMembershipCountsSNAPSHOT_DATE on
  csi_MembershipCounts(SNAPSHOT_DATE)
go
MEMBER COUNTS PROCEDURE

create procedure csi_sp_CaptureMembershipCounts as
begin
    declare @SnapshotDate datetime

    set @SnapshotDate = convert(varchar(8), getdate(), 112) -- date with no time

    insert csi_MembershipCounts (SNAPSHOT_DATE, MEMBER_TYPE, CATEGORY, CHAPTER, QUANTITY)
    select @SnapshotDate, n.MEMBER_TYPE, n.CATEGORY, n.CHAPTER, count(*)
    from Name n
    where @SnapshotDate not in (select distinct SNAPSHOT_DATE from csi_MembershipCounts)
    group by n.MEMBER_TYPE, n.CATEGORY, n.CHAPTER
end

Activity is an area ripe for customization

iMIS has low requirements for this table:
- SEQN from Counter table
- Valid ID
- Activity type
- Transaction Date
- MEMBER_TYPE and CO_ID preferred
- If fields have lookups, they must match
NOTES FROM THE FIELD - COUNTER TABLE

- Must update and extract value in a way that prevents overlap
- sp_asi_GetCounter is **unsafe** without a transaction
- For counters without checksums, there is an easier way
  ```
  update Counter
  set LAST_VALUE = LAST_VALUE + N,
  @myVar = LAST_VALUE + 1
  where COUNTER_NAME = 'CounterYouNeed'
  ```
- It's OK to add new counters to the table

CREATE COMMITTEE ACTIVITY

```sql
create p_CreateCommitteeActivity
  (@psID varchar(10), @psCommittee varchar(40), @pxEffDate datetime, @pxThruDate datetime, @psPosition varchar(40))
as
begin
  declare @lSeqn int
  update c
  set LAST_VALUE = c.LAST_VALUE + 1,
  @lSeqn = c.LAST_VALUE + 1,
  LAST_UPDATED = getdate(),
  UPDATED_BY = 'CreateCommittee'
  from Counter c
  where c.COUNTER_NAME = 'Activity'
insert Activity (SEQN, ID, ACTIVITY_TYPE, PRODUCT_CODE, OTHER_CODE, DESCRIPTION, EFFECTIVE_DATE, THRU_DATE, ACTION_CODES, MEMBER_TYPE, CO_ID)
  select @lSeqn, @psID, p.PRODUCT_CODE, p.PRODUCT_MINOR, p.TITLE, @pxEffDate, @pxThruDate, @psPosition, n.MEMBER_TYPE, n.CO_ID)
  from Name n
cross join Product p
  where n.ID = @psID
  and p.PROD_TYPE = 'COMMITTEE'
  and p.PRODUCT_MINOR = @psCommittee
end
```
NOTES FROM THE FIELD – WATCH OUT FOR

- Adding Name records to iMIS
  - There are a lot of tables and fields that are touched
- Updating Member_Type
  - Don’t forget to set MEMBER_RECORD, COMPANY_RECORD, PREVIOUS_MT, MT_CHANGE_DATE and log your change in Name_Log!
- Using Begin Tran and End Tran
  - These can lock your db if your script fails!
  - There are better ways to do this with Try and Catch

TRIGGERS

- Cue scary music....
**TRIGGERS - WHAT**

- Stored procedures in response to an insert, update or delete on a table
- Can examine before/after versions of the table
- A trigger will *always* run, so it had better do the right thing
- The trigger must deal correctly with multi-row updates
- Greater than normal chance of causing iMIS to misbehave
- Important not to allow rowcounts or other output
  - Because a response will crash iMIS

**TRIGGERS - WHEN**

- Witness important changes to certain tables, insert ID in a "queue" table for later review and processing (find who and when that data is being deleted by)
- Additional change logging
- "Suppress" an illegal change instead of rolling it back. (Update table again with info from deleted)
  - Risky because remember, you may not be around when something goes wrong!
TRIGGERS

• Triggers can be for Inserts, Deletes and Updates (see final slide for why updates are not reliable)
• Inserted is the data that was inserted – treat it like a copy of your table
• Deleted is the data that was deleted – treat it the same as Inserted in terms of structure.
• Here is sample code to track mysterious deletions on certain tables – we use this at CSI

TRIGGER - SAMPLE CODE

1. Create the backup table

```sql
-- Create log table
CREATE TABLE [dbo].[csi_Deckup_DeleteLog](
    TABLE_NAME sysname,
    SEQN int,
    ID varchar(10),
    date_time datetime,
    username sysname,
    system_username sysname,
    current_username sysname,
    session_username sysname,
    hostname nvarchar(128),
    application_name nvarchar(128),
    inserted xml,
    deleted xml,
    time_stamp timestamp
)
GO
```
TRIGGER - SAMPLE CODE

2. Create a DELETE trigger on the Activity table

```sql
CREATE TRIGGER [dbo].[esi_TR_Activity_Delete] ON [dbo].[Activity] FOR DELETE AS
BEGIN NOCOUNT ON DELETE
INSERT [dbo].[esi_Backup_DeleteLog] (TABLE_NAME, SEQN, ID, date_time, username, system_username, current_username, hostname, application_name, inserted, deleted)
SELECT TABLE_NAME = 'Activity'
, SEQN
, ID
, date_time = GETDATE()
, username = USER
, system_username = SYSPRO_USER
, current_username = CURRENT_USER
, session_username = SESSION_USER
, hostname = HOST_NAME()
, application_name = APP_NAME()
, inserted =
    SELECT SEQN, ID, ACTIVITY_TYPE, TRANSACTION_DATE, PRODUCT_CODE, OTHER_CODE, DESCRIPTION
    FROM inserted [Activity]
    WHERE [Activity] SEQN =d SEQN
    FOR XML AUTO
) , deleted =
    SELECT SEQN, ID, ACTIVITY_TYPE, TRANSACTION_DATE, PRODUCT_CODE, OTHER_CODE, DESCRIPTION
    FROM deleted [Activity]
    WHERE [Activity] SEQN =d SEQN
    FOR XML AUTO
) FROM deleted d
```

TRIGGER - COMMENTS

- ASI is using Triggers now for their new Panel Designer tables
- Please try to avoid using them unless writing to non-iMIS tables
- Even then - make sure you do not return any data! (Set NOCOUNT On
TRIGGERS - FROM THE FIELD

- A script calling a stored procedure, running every 5 minutes is almost always better than a trigger
- iMIS writes data in an order that you are not aware of and WHEN you are not aware of
  - A trigger on a demographic table that updates the name table might have its data overwritten if iMIS writes to the demographic table first and then the Name table
  - iMIS re-writes data when you don’t expect it to - e.g. all Activities are re-written when you post a batch!
- Update Triggers are not reliable to detect changing of data

THANK YOU

Doug Morris
Computer System Innovations, Inc.
855 W Prairie Ave
Wheaton, IL 60187
USA
+1 630 681-1100

dmorriss@csiinc.com

@dougcsi (don’t tweet 😊)