CA IdentityMinder External Task Persistence Cleanup & Archival Process

© 2015 CA

<table>
<thead>
<tr>
<th>Document Title</th>
<th>CA IdentityMinder External Task Persistence Cleanup &amp; Archival Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>ALL</td>
</tr>
<tr>
<td>Project</td>
<td></td>
</tr>
<tr>
<td>Last Saved Date</td>
<td>14-Dec-2015</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
</tbody>
</table>
# Document Properties

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Name:</td>
<td>ALL</td>
</tr>
<tr>
<td>Project Name:</td>
<td></td>
</tr>
<tr>
<td>Project ID:</td>
<td></td>
</tr>
<tr>
<td>Document Name:</td>
<td>Solution Design Specification</td>
</tr>
<tr>
<td>Document Number:</td>
<td>0001</td>
</tr>
<tr>
<td>Document Version:</td>
<td>1.0</td>
</tr>
<tr>
<td>Version Date:</td>
<td>14-Dec-2015</td>
</tr>
<tr>
<td>Document Status:</td>
<td>Draft</td>
</tr>
<tr>
<td>Authors</td>
<td>CA Services</td>
</tr>
</tbody>
</table>

Copyright © 2015 CA

All rights reserved. All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

This document is for your informational purposes only. To the extent permitted by applicable law, CA provides this document "As Is" without warranty of any kind, including, without limitation, any implied warranties of merchantability or fitness for a particular purpose, or non-infringement. In no event will CA be liable for any loss or damage, direct or indirect, from the use of this document including, without limitation, lost profits, business interruption, goodwill or lost data, even if CA is expressly advised of such damages.
### Review & Approval

<table>
<thead>
<tr>
<th>Review Date</th>
<th>Action</th>
<th>Name</th>
<th>Company, Organizational Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/01/2015</td>
<td>Author</td>
<td>Amit Sinha</td>
<td>CA, Sr. Architect</td>
</tr>
<tr>
<td></td>
<td>Reviewer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Approver</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Change History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Author</th>
<th>Description of Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/01/2015</td>
<td>1.0</td>
<td>Amit Sinha</td>
<td>Initial</td>
</tr>
<tr>
<td>9/15/2015</td>
<td>1.1</td>
<td>Alan Baugher</td>
<td>Update to allow general use.</td>
</tr>
</tbody>
</table>

### Distribution List

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Name</th>
<th>Company, Organizational Position</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Referenced Documents

**Related Project Documents**

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Name</th>
<th>Company, Organizational Position</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Table of Contents

## TABLE OF FIGURES

- Table of Contents .......................... 5
- Glossary .................................. 6

### 1.1 External Task Persistence Cleanup ......................................................... 6

### 1.2 External Cleanup Results .............................................................................. 7

#### 1.2.1 Example 1 .......................................................................................... 7

#### 1.2.2 Example 2 ........................................................................................ 8

### 1.3 Process ........................................................................................................ 8

### 1.4 New Tables and Stored Procedures ............................................................ 9

#### 1.4.1 Create garbage collection status table 'NEWgcStat' ............................ 10

#### 1.4.2 Create new stored procedure 'spNEWgcStatRunInit' to initialize gsStat ... 11

#### 1.4.3 Create new stored procedure 'spNEWgcStatRunUpdate' to capture deleted and archived record counts .......................................................... 11

#### 1.4.4 Create new stored procedure 'NEWgarbageCollectionTaskPersistence' .......... 13

#### 1.4.5 Create new stored procedure 'NEWgarbageCollectByTaskId' .......... 16

#### 1.4.6 Create new stored procedure 'NEWgarbageCollectById' ..................... 18

#### 1.4.7 Create new stored procedure 'NEWgarbageCollectByEventId' ................. 20

### 1.5 Appendix .................................................................................................... 22
Table of Figures

Table 1 - Glossary .................................................................................................................................................. 6
Table 2 – Cleanup impacted tables ....................................................................................................................... 7
Table 3 – Archive impacted tables ....................................................................................................................... 7
Table 4 – Results Example 1 .................................................................................................................................. 7
Table 5 – Results Example 2 .................................................................................................................................. 8
Table 6 – Garbage Collection Parameters ......................................................................................................... 9
Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition of Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>The process of verifying the identity of a user logging onto a network. Passwords, digital certificates, smart cards and biometrics can be used to prove the identity of the client to the network.</td>
</tr>
<tr>
<td>Authorization</td>
<td>The process of granting access to a system resource based on a user’s rights or permissions.</td>
</tr>
<tr>
<td>CA IdentityMinder</td>
<td>CA IdentityMinder is an administrative management product that provides automated identity management services for the creation, modification and eventual deletion of accounts and entitlements based on user relationships. It manages access and entitlements on a whole range of enterprise systems—from mainframes to Web applications.</td>
</tr>
<tr>
<td>Task Persistence</td>
<td>Tasks invoked in the Identity Manager system persist in the database to guarantee execution. Task persistence records are stored in the IDM database.</td>
</tr>
</tbody>
</table>

Table 1 - Glossary

1.1 External Task Persistence Cleanup

CA IdentityMinder provides an out-of-the-box process for performing database maintenance/cleanup. This process must be implemented right away after implementation of the solution to frequently clean-up and optionally archive the cleaned up data.

In the absence on this process, the database records quickly grow eventually making the Identity solution run slow.

The process defined in this document is an ‘External’ process that runs directly at the database tier to speed up the cleanup and archival process and has the capability to perform efficient cleanup as well as archival at the same time.

Task persistence cleanup (also known as garbage collection) involves cleaning up of records from the following IdentityMinder tables.

<table>
<thead>
<tr>
<th>Cleanup Impacted DB tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasksession12_5</td>
</tr>
<tr>
<td>Object12_5</td>
</tr>
<tr>
<td>RuntimeStatusDetail12</td>
</tr>
<tr>
<td>RuntimeStatusDetailAttribute12</td>
</tr>
<tr>
<td>Lock12_5</td>
</tr>
</tbody>
</table>
If archival of the records is also desired, the following IdentityMinder tables are impacted.

**Cleanup Impacted DB tables**

<table>
<thead>
<tr>
<th>Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Archive_tasksession12_5</td>
</tr>
<tr>
<td>Archive_object12_5</td>
</tr>
<tr>
<td>Archive_runtimeStatusDetail12</td>
</tr>
<tr>
<td>Archive_event12_5</td>
</tr>
<tr>
<td>Archive_event_object12_5</td>
</tr>
</tbody>
</table>

**Table 3 – Archive impacted tables**

### 1.2 External Cleanup Results

The time taken to complete a cleanup job depends on criteria such as the number of dependent child records as well as the total number of records in the database for the affected tables.

As an example, running the out-of-box process on large dataset (approximately 3 million tasksession12_5 records) cleans up and archives approximately 50k task persistence (tasksession12_5) records in 5 days. In contrast this external process, in the same environment, cleans up 1 million records in 5 days. That is **20 (twenty) times faster!**

#### 1.2.1 Example 1

Small tasksession12_5 count (23k)

Cleanup and archival takes approximately 24 minutes.

**Table 4 – Results Example 1**

<table>
<thead>
<tr>
<th>Table</th>
<th>Record Count Before Cleanup</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_object12_5</td>
<td>23,150</td>
</tr>
<tr>
<td>event12_5</td>
<td>24,309</td>
</tr>
<tr>
<td>lock12_5</td>
<td>58,312</td>
</tr>
<tr>
<td>object12_5</td>
<td>70,139</td>
</tr>
<tr>
<td>runtimeStatusDetail12</td>
<td>133,479</td>
</tr>
<tr>
<td>runtimeStatusDetailAttribute12</td>
<td>0</td>
</tr>
<tr>
<td><strong>tasksession12_5</strong></td>
<td><strong>22,915</strong></td>
</tr>
</tbody>
</table>
The results of each run gets to be stored in the ‘NEWgcStat’ table and a query into the table would show the start and finish times as well as the number of records cleaned up and archived.

1.2.2 Example 2
Large tasksession12_5 count (over 3 million).
This external process running continuously cleans and archives approximately 1 million records every 5 days.

<table>
<thead>
<tr>
<th>Table</th>
<th>Record Count Before Cleanup</th>
</tr>
</thead>
<tbody>
<tr>
<td>tasksession12_5</td>
<td>3,384,408</td>
</tr>
<tr>
<td>object12_5</td>
<td>13,235,841</td>
</tr>
<tr>
<td>event12_5</td>
<td>6,467,033</td>
</tr>
<tr>
<td>lock12_5</td>
<td>13,180,567</td>
</tr>
<tr>
<td>event_object12_5</td>
<td>4,048,505</td>
</tr>
<tr>
<td>runtimeStatusDetail12</td>
<td>13,013,286</td>
</tr>
<tr>
<td>runtimeStatusDetailAttribute12</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5 – Results Example 2

1.3 Process

Invoke the stored procedure ‘NEWgarbageCollectTaskPersistence’ to start the ‘Cleanup & Archival’ process.

Query ‘NEWgcStat’ table for results.

The newly installed stored procedure has the following parameters which may be tweaked for runs.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition of Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>ime_id</td>
<td>Environment ID for which the cleanup is to be performed. Ex: 1, 2 (Hint: Login to the management console and click the environment. Look at the ‘oid’ query string parameter to get the environment id for your environment.)</td>
</tr>
<tr>
<td>cutoff_time</td>
<td>All records before the cutoff time would be considered to be in-scope for cleanup.</td>
</tr>
<tr>
<td>Audit_timeout_days</td>
<td>Selection criteria for ‘In-Scope’ records that are in an ‘audit’ state for over this specified number of days. Defaults to 1.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition of Term</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Chunk_record_count</td>
<td>The number of records that should be committed during each transaction during the cleanup process run. Defaults to 500.</td>
</tr>
<tr>
<td>Archive_records</td>
<td>A value of ‘1’ indicates archival of records is desired. Set it to ‘0’ if archival is not desired.</td>
</tr>
<tr>
<td>Max_gc_runtime_hours</td>
<td>The maximum time in hours for which this cleanup process should run. Defaults to 12 hours. If this process is executed at 5PM local time the cleanup and archival process will stop by 5AM.</td>
</tr>
</tbody>
</table>

**Table 6 – Garbage Collection Parameters**

1.4 **New Tables and Stored Procedures**

The below process flow diagram depicts the details of the direct cleanup. The dotted lines show Primary-Foreign key constraints implemented for the solution.
In order to accomplish the external cleanup, the following database tasks must be implemented.

1.4.1 Create garbage collection status table ‘NEWgcStat’

This table will collect information about each time the garbage collection process is run and will have counts of the number of records cleaned up as well as archived as a result of the run.

```sql
/*** Create table to track all delete and archive counts for gc runs ***/
DROP table NEW_gcStat
CREATE table NEW_gcStat (run_id datetime, finish datetime,
tasksession12_5 int, object12_5 int, runtimeStatusDetail12 int, event12_5 int,
```
1.4.2 Create new stored procedure ‘spNEWgcStatRunInit’ to initialize gsStat

```sql
/* Create procedure to initialize gcStat table for a run */
DROP PROCEDURE [dbo].[spNEWgcStatRunInit]
CREATE PROCEDURE [dbo].[spNEWgcStatRunInit]
    @run_id datetime
AS
BEGIN
    INSERT into NEW_gcStat (run_id, tasksession12_5, object12_5, runtimeStatusDetail12, event12_5, event_object12_5, lock12_5, archive_tasksession12_5, archive_object12_5, archive_runtimeStatusDetail12, archive_event12_5, archive_event_object12_5) values (@run_id,0,0,0,0,0,0,0,0,0,0,0);
END
```

1.4.3 Create new stored procedure ‘spNEWgcStatRunUpdate’ to capture deleted and archived record counts

```sql
/* Create procedure to increment counters for a run */
DROP PROCEDURE [dbo].[spNEWgcStatRunUpdate]
CREATE PROCEDURE [dbo].[spNEWgcStatRunUpdate]
    @run_id datetime,
    @update_type nvarchar(32),
    @count int
AS
BEGIN
    if(@update_type = 'tasksession12_5')
    BEGIN
        UPDATE NEW_gcStat SET tasksession12_5 = tasksession12_5 + @count
    END
```

WHERE run_id = @run_id;
END

if(@update_type = 'archive_tasksession12_5')
BEGIN
    UPDATE NEW_gcStat SET archive_tasksession12_5 = archive_tasksession12_5 + @count WHERE run_id = @run_id;
END

if(@update_type = 'object12_5')
BEGIN
    UPDATE NEW_gcStat SET object12_5 = object12_5 + @count WHERE run_id = @run_id;
END

if(@update_type = 'archive_object12_5')
BEGIN
    UPDATE NEW_gcStat SET archive_object12_5 = archive_object12_5 + @count WHERE run_id = @run_id;
END

if(@update_type = 'runtimeStatusDetail12')
BEGIN
    UPDATE NEW_gcStat SET runtimeStatusDetail12 = runtimeStatusDetail12 + @count WHERE run_id = @run_id;
END

if(@update_type = 'archive_runtimeStatusDetail12')
BEGIN
    UPDATE NEW_gcStat SET archive_runtimeStatusDetail12 = archive_runtimeStatusDetail12 + @count WHERE run_id = @run_id;
END

if(@update_type = 'event12_5')
BEGIN
    UPDATE NEW_gcStat SET event12_5 = event12_5 + @count WHERE run_id = @run_id;
END

if(@update_type = 'archive_event12_5')
BEGIN
    UPDATE NEW_gcStat SET archive_event12_5 = archive_event12_5 + @count WHERE run_id = @run_id;
END
if(@update_type = 'event_object12_5')
BEGIN
    UPDATE NEW_gCStat SET event_object12_5 = event_object12_5 + @count
    WHERE run_id = @run_id;
END
if(@update_type = 'archive_event_object12_5')
BEGIN
    UPDATE NEW_gCStat SET archive_event_object12_5 =
    archive_event_object12_5 + @count WHERE run_id = @run_id;
END
if(@update_type = 'lock12_5')
BEGIN
    UPDATE NEW_gCStat SET lock12_5 = lock12_5 + @count WHERE run_id =
    @run_id;
END
if(@update_type = 'finish')
BEGIN
    UPDATE NEW_gCStat SET finish = getdate() WHERE run_id = @run_id;
END

1.4.4 Create new stored procedure ‘NEWgarbageCollectTaskPersistence’

CREATE PROCEDURE [dbo].[NEWgarbageCollectTaskPersistence]
@ime_id VARCHAR(50),
@cutoff_time DATETIME,
@audit_timeout_days FLOAT = 1,
@chunk_rec_count int =500,
@archive_records int = 1,
@max_gc_runtime_hours int = 12
AS
    DECLARE @gcTaskId VARCHAR(255);
    DECLARE @counter int;
    DECLARE @transactionsCount int;
    DECLARE @Error int;
DECLARE @run_id datetime = getdate();

/* Initialize gcStat */
EXEC spNEWgcStatRunInit @run_id;

DECLARE gcTasksCursor CURSOR LOCAL FOR
    SELECT tasksessionid
    FROM tasksession12_5
    WHERE environmentid = @ime_id
    AND ( (tasksession12_5.state = 512) -- mark for deletion
        OR (tasksession12_5.state = 128) -- completed
        OR (tasksession12_5.state = 256) -- cancelled
        OR (tasksession12_5.state = 32) -- rejected
        OR (tasksession12_5.state = 2) -- invalid
        OR (tasksession12_5.state = 1048576) -- task aborted
        )
        AND last_access_time < @cutoff_time
    )
    OR
    ( tasksession12_5.state = 16384 -- audit state
    AND last_access_time <
    CASE WHEN @cutoff_time < (GETDATE() - @audit_timeout_days) THEN @cutoff_time ELSE GETDATE() - @audit_timeout_days END
    )
    )
    AND (initiatorid IS NULL OR initiatorid = tasksessionid);
BEGIN
    set @counter = 0;
    OPEN gcTasksCursor;
    begin transaction;
    WHILE 1 = 1
    BEGIN
        IF select DATEDIFF(hour, @run_id, getdate()) > @max_gc_runtime_hours BREAK;
        FETCH NEXT FROM gcTasksCursor INTO @gcTaskId;
        IF @@FETCH_STATUS <> 0 BREAK;
        -- exit loop if condition is true
        set @counter = @counter + 1;
        /* Insert here in Archive */
        if (@archive_records <> 0)
        BEGIN
            /* Archive this record for PK_FK constraint */
            INSERT into [imdb_scheduled].dbo.archive_tasksession12_5 (tasksessionid, state, created_time, last_access_time, user_dn, org_dn, environmentid, name, description, acknowledged, parent_of_wf_id, action_type, priority, initiatorid, nesting_type, next_state) SELECT tasksessionid, state, created_time, last_access_time, user_dn, org_dn, environmentid, name, description, acknowledged, parent_of_wf_id, action_type, priority, initiatorid, nesting_type, next_state FROM [imdb_active].dbo.tasksession12_5 WHERE tasksessionid = @gcTaskId;
            /* Update NEWgcStat */
            EXEC spNEWgcStatRunUpdate @run_id, 'archive_tasksession12_5', @@ROWCOUNT
            END
        EXEC NEWgarbageCollectByTaskId @gcTaskId, @archive_records, @run_id;
        set @Error = @@ERROR
        if @Error <> 0 --if error is raised
            goto LogError
        IF (@counter = @chunk_rec_count)
BEGIN
  commit;
  set @counter = 0;
  begin transaction;
END
END

if (@counter > 0)
  commit;
set @transactionsCount = @@trancount;
if (@transactionsCount > 0)
  commit;
CLOSE gcTasksCoursor;
DEALLOCATE gcTasksCoursor;
goto ProcEnd
LogError:
  set @transactionsCount = @@trancount;
  if (@transactionsCount > 0)
    rollback transaction
ProcEnd:
  /* Update NEWgcStat */
  EXEC spNEWgcStatRunUpdate @run_id, 'finish', '0'
END --NEWgarbageCollectTask Persistence

1.4.5  Create new stored procedure 'NEWgarbageCollectByTaskId'

CREATE PROCEDURE [dbo].[NEWgarbageCollectByTaskId]
@task_id VARCHAR(255),
@archive_records int,
@run_id datetime
AS
DECLARE @workingId VARCHAR(255);

DECLARE tasksCoursor CURSOR LOCAL FOR
    SELECT tasksessionid
    FROM tasksession12_5
    WHERE initiatorid = @task_id AND tasksessionid != initiatorid;

DECLARE eventCoursor CURSOR LOCAL FOR
    SELECT eventid
    FROM event12_5
    WHERE tasksessionid = @task_id;

BEGIN

    -- PRINT 'GC task ID ' + @task_id;

    OPEN tasksCoursor;

    WHILE 1 = 1
    BEGIN
        FETCH NEXT FROM tasksCoursor INTO @workingId;
        IF @@FETCH_STATUS <> 0 BREAK; -- exit loop if condition is true

        /* Insert here in Archive */
        if (@archive_records <> 0)
        BEGIN
            /* Archive this record for PK_FK constraint */
            INSERT into [imdb_scheduled].dbo.archive_tasksession12_5 (tasksessionid, state, created_time, last_access_time, user_dn, org_dn, environmentid, name, description, acknowledged, parent_of_wf_id, action_type, priority, initiatorid, nesting_type, next_state)
            SELECT tasksessionid, state, created_time, last_access_time, user_dn, org_dn, environmentid, name, description, acknowledged, parent_of_wf_id, action_type, priority, initiatorid, nesting_type, next_state FROM [imdb_active].dbo.tasksession12_5 WHERE tasksessionid = @workingId;

            /* Update NEWGcStat */
            EXEC spNEWGcStatRunUpdate @run_id, 'archive_tasksession12_5',
            @@ROWCOUNT
        END
    END

END
EXEC NEWgarbageCollectByTaskId @workingId, @archive_records, @run_id;
END
CLOSE tasksCoursor;
DEALLOCATE tasksCoursor;

OPEN eventCoursor;
WHILE 1 = 1
BEGIN
    FETCH NEXT FROM eventCoursor INTO @workingId;
    IF @@FETCH_STATUS <> 0 BREAK; -- exit loop if condition is true
    EXEC NEWgarbageCollectByEventId @workingId, @archive_records, @run_id;
END
CLOSE eventCoursor;
DEALLOCATE eventCoursor;

EXEC NEWgarbageCollectById @task_id, @archive_records, @run_id;

DELETE FROM tasksession12_5
WHERE tasksessionid = @task_id;

/* Update NEWgcStat */
EXEC spNEWgcStatRunUpdate @run_id, 'tasksession12_5', @@ROWCOUNT

END --NEWgarbageCollectByTaskId

1.4.6 Create new stored procedure ‘NEWgarbaseCollectById’

CREATE PROCEDURE
[dbo].[NEWgarbageCollectById]
@workingId VARCHAR(255),
@archive_records int,
@run_id datetime

AS
BEGIN

-- PRINT 'GC ID ' + @workingId;

DELETE FROM lock12_5
WHERE lockid = @workingId;

/* Update NEWgcStat */
EXEC spNEWgcStatRunUpdate @run_id, 'lock12_5', @@ROWCOUNT

DELETE FROM runtimeStatusDetailAttribute12
WHERE rsdid IN (SELECT runtimeStatusDetail12.id FROM runtimeStatusDetail12
WHERE eventID = @workingId);

/* Archive record before deletion */
if(@archive_records <> 0)
BEGIN

INSERT into [imdb_scheduled].dbo.archive_runtimeStatusDetail12 (id,
originationTime, eventID, parentID, sourceIdentifier, serverName, provDomainName,
displayName, description, descriptionType, descriptionParam, admindn, adminName,
importance) SELECT id, originationTime, eventID, parentID, sourceIdentifier, serverName,
provDomainName, displayName, description, descriptionType, descriptionParam, admindn,
adminName, importance FROM [imdb_active].dbo.runtimeStatusDetail12 WHERE eventid =
@workingId

/* Update NEWgcStat */
EXEC spNEWgcStatRunUpdate @run_id, 'archive_runtimeStatusDetail12',
@@ROWCOUNT
END

DELETE FROM runtimeStatusDetail12
WHERE eventID = @workingId;

/* Update NEWgcStat */
EXEC spNEWgcStatRunUpdate @run_id, 'runtimeStatusDetail12', @@ROWCOUNT

/* Archive record before deletion */
if(@archive_records <> 0)
BEGIN

    INSERT into [imdb_scheduled].dbo.archive_object12_5 (objectid, objecttype, object) SELECT objectid, objecttype, object FROM [imdb_active].dbo.object12_5 WHERE objectid = @workingId;

/* Update NEWgcStat */
EXEC spNEWgcStatRunUpdate @run_id, 'archive_object12_5', @@ROWCOUNT
END

DELETE FROM object12_5
WHERE objectid = @workingId;

/* Update NEWgcStat */
EXEC spNEWgcStatRunUpdate @run_id, 'object12_5', @@ROWCOUNT

END --NEWgarbageCollectById

1.4.7  Create new stored procedure ‘NEWgarbageCollectByEventId’

CREATE PROCEDURE [dbo].[NEWgarbageCollectByEventId]
@event_id VARCHAR(255),
@archive_records int,
@run_id datetime
AS
BEGIN

    -- PRINT 'GC event ID ' + @event_id;
    EXEC NEWgarbageCollectById @event_id, @archive_records, @run_id;

    /* Archive record for PK_FK into event12_5 first */
if(@archive_records <> 0)
BEGIN

INSERT into [imdb_scheduled].dbo.archive_event12_5 (eventid, state,
tasksessionid, created_on, last_access_time, name, description, type, next_state) SELECT

 eventid, state, tasksessionid, created_on, last_access_time, name, description, type,
next_state FROM [imdb_active].dbo.event12_5 WHERE eventid = @event_id;


/* Update NEWgcStat */
EXEC spNEWgcStatRunUpdate @run_id, 'archive_event12_5', @@ROWCOUNT

END


/* Archive record before deletion */
if(@archive_records <> 0)
BEGIN

INSERT into [imdb_scheduled].dbo.
archive_event_object12_5 (tasksessionid,
eventid, object_type, isprimary, object_uniquename) SELECT tasksessionid, eventid,
object_type, isprimary, object_uniquename FROM [imdb_active].dbo.event_object12_5
WHERE eventid = @event_id;


/* Update NEWgcStat */
EXEC spNEWgcStatRunUpdate @run_id, 'archive_event_object12_5', @@ROWCOUNT

END


DELETE FROM event_object12_5
WHERE eventid = @event_id;


/* Update NEWgcStat */
EXEC spNEWgcStatRunUpdate @run_id, 'event_object12_5', @@ROWCOUNT

DELETE FROM event12_5
WHERE eventid = @event_id;


/* Update NEWgcStat */
EXEC spNEWgcStatRunUpdate @run_id, 'event12_5', @@ROWCOUNT
1.5 Appendix

Database scripts for enabling the above process.

IDM External Garbage Collection-examples.zip