forceAMP.com

DBAmp

SQL Server Integration with Salesforce.com

Version 3.1.5

Copyright © 2009 by forceAmp.com. All rights reserved.

Table of Contents

Acknowledgments **6**
Chapter 1: Installation/Upgrading **7**
Upgrading an existing installation 7
Prerequistes 7
Restrictions with Professional Edition 8
Running the DBAmp installation file 8
Configure the DBAmp provider options 8
Connecting DBAmp to SQL Server 8
Verifying the linked server 10
Install the DBAmp Stored Procedures 10
Running the DBAmp Configuration Program 10
Pointing DBAmp to your Salesforce Sandbox Instance 11
Chapter 2: Using DBAMP as a Linked Server **12**
Four Part Object Names 12
SQL versus SOQL 12
Using the four part object name and SQL 12
Using OPENQUERY and SOQL 13
Inserting rows using SQL 15
Updating and Deleting rows using SQL 16
Joining Salesforce.com Tables 17
Analyzing Performance when Joining Tables 17
Using BIT datatype with DBAmp 19
Using Dates with DBAmp 19
Using DBAmp System Tables (sys_sf tables) 20
Using Count() with salesforce.com objects 21
Using DBAmp to convert currency amounts to a default currency 21
Using DBAmp to return translated values for picklists 22
Retrieving Archived and Deleted records 22
Using Column Subset views 23
DBAmp and Salesforce API call Counts 24
Chapter 3: Making Local Copies of Salesforce Data **25**
How to run the SF_Replicate proc to make a local copy 25
Viewing the job history 27

Replicating all Salesforce Objects 27
How to run the SF_ReplicateAll proc to replicate all objects 27
Copying only the rows that have changed 28
Replicating Large Tables 28
Including Archived and Deleted rows in the local copy 28
Best Practices for Replicate and Refresh schedules 29
Chapter 4: Bulk Insert, Upsert, Delete and Update into Salesforce30
Checking the Column Names of the Input Table 30
Using External Ids as Foreign Keys (without bulkapi switch) 31
Using External Ids as Foreign Keys (with bulkapi switch) 32
Understanding the Error Column 32
Bulk Inserting rows into Salesforce 32
Bulk Upserting rows into Salesforce 33
Bulk Updating rows into Salesforce 33
Bulk Deleting rows from Salesforce 34
Bulk UnDeleting rows from Salesforce 34
Controlling the batch size 34
How to run the SF_BulkOps proc 34
How to run the SF_BulkOps proc without using xp_cmdshell 36
Understanding SF_Bulkops failures (Web Services API) 37
Using the Bulk API with SF_BulkOps 38
Controlling the batch size with the Bulk API 39
Understanding a Sort Column when using the Bulk API 39
Using the HardDelete operation with the Bulk API 40
Controlling Concurrency Mode with the Bulk API 40
Using Optional SOAP Headers 40
Converting Leads with SF_Bulkops 41
Chapter 5: Using SSIS with DBAmp 44
Create a Connection for DBAmp 44
Using DBAmp as an OLE DB Source 44
Pushing Data to Salesforce.com using SSIS 45
Chapter 6: Uploading files into Content, Documents and Attachments 48
Chapter 7: Creating Database Diagrams and Keys 52
Creating a Primary Key 53
Creating Foreign Keys 53

Creating a Database Diagram 53
Chapter 8: Using Excel with Views to Linked Server Tables 55
Create Views of the SALESFORCE linked server tables 55
Using Excel 56
Chapter 9: DBAmp Stored Procedure Reference 61
SF_BulkOps 62
SF_ColCompare 67
SF_CreateKeys 68
SF_DropKeys 69
SF_Generate 70
SF_Refresh 71
SF_RefreshIAD 73
SF_RefreshAll 74
SF_Replicate 76
SF_ReplicateHistory 78
SF_ReplicateAll 79
SF_ReplicateIAD 81
SF_ReplicateLarge 82
Chapter 10: DBAmp Registry Settings 84
Metadata Override 84
Base64 Maximum Field Size: 84
Receive Timeout 84
BulkAPI Polling Interval 85
BulkAPI Status Timeout 85
Use ConvertCurrency Function 85
Use ToLabel Function 85
TriggerAutoResponseEmail, TriggerOtherEmail, TriggerUserEmail 85
UseDefaultAssignment 86
Chapter 11: Retrieving Salesforce Metadata 87
How to run the SF_Metadata proc 87
Using the LIST and RETRIEVE operations 87
Requirements for the input table 88
Example: Retrieve Dependent Picklist Information 90
Example: Retrieve Field Descriptions 91
Chapter 12: Using DBAmp Performance Package 93

Installing the DBAmp Performance Package.....93
Using the DBAmp_Log Table 95
Using the Performance Views 96
DBAmp_Replicate_Perf view 96
DBAmp_Refresh_Perf view 97
DBAmp_BulkOps_Perf view.....98
Using Excel with the Performance Views.....99

Acknowledgments

Thanks to Sarah Parra of Microsoft. Without her excellent support, DBAmp wouldn't exist.

Also, thanks to Dave Carroll at Salesforce.com for being the "Original" sForce programmer. Dave's sample code always points the way for the rest of us.

And finally, thanks to those customers who have contributed ideas and designs for several important features of DBAmp:

C.J. Land	Local copy replication
Andy Hilliard	Sys_sfPickList
Darrell Grissen	Sys_sfLastId
Tad Tjornhom	Bulk Inserting
Paul Coyne	sf_replicateIAD
John Gee	Metadata support

Chapter 1: Installation/Upgrading

Upgrading an existing installation

If you are upgrading an existing installation, please do the following.

1. Stop SQL Server.
2. Run the DBAmp installation program. You will need your serial number for installation. Please contact support@forceamp.com if you need help with this value.
3. Your previous linked server definition can be use without modification.
4. The DBAmp stored procedures change with every release. You must upgrade every SQL database that currently contains DBAmp stored procs with the new versions. Follow the instructions in the **Install the DBAmp Stored Procedures** section later in this chapter. Failure to do this will result in errors.
5. Because the new version may connect to a newer API endpoint, additional fields and objects may become visible with the upgrade. If you are using sf_refresh for local copies, you must run sf_replicate on that object to pickup these schema changes. Then you can resume your normal sf_refresh schedule.

Note that there are major, breaking changes that have occurred recently with DBAmp.

- **DBAmp only supports SQL 2005 or higher.**

- **DBAmp only supports Windows 2008 R2 or higher.**

- **SQL 2008 or greater and datetime2(7).** On SQL 2008 or greater systems, date and datetime fields of salesforce.com objects are now created as datetime2(7) fields in the local database. To force these fields to be created as datetime fields instead, set the Database Compatibility Level of the Salesforce backup database to 90 prior to replicating the data (step 5 above). This change applies to SQL 2008 and greater only.

Prerequistes

Before installing DBAmp, make sure that an instance of SQL Server 2005 or greater is installed on the machine. If you do not have SQL Server, you may download the SQL Server 2008 Express with Database Tools, which is available for free from Microsoft. In addition, be aware that DBAmp only supports Windows 2008 R2 or higher.

IMPORTANT: If you are using SQL Server Express, make sure you download the package from Microsoft that contains the Database

Tools. You will need the SQL Management Studio tool to complete the DBAmp installation.

There is an outstanding Microsoft issue that affects DBAmp. This issue only occurs when the service account that you specify for SQL Server is the **Network Service** account. Please use a different service account (like a user account) for the SQL Server instance. We recommend that you use the LocalSystem account or an admin domain.

Restrictions with Professional Edition

DBAmp works with the Professional Edition of salesforce but certain functions like the bulkapi option and the metadata support are not available unless the customer purchases the api option from salesforce.com.

Running the DBAmp installation file

To install DBAmp, unzip the DBAmp package to a temporary directory and run the Setup program. Setup will prompt you for the DBAmp program directory and install the software.

To uninstall DBAmp, use the Windows Add/Remove Programs option on the control panel.

Configure the DBAmp provider options

NOTE: DO NOT SKIP THIS STEP. DBAMP WILL NOT FUNCTION PROPERLY.

Expand the Providers tree entry in the Object Explorer (Server Objects/Linked Servers/Providers). Right click the DBAmp.DBAmp provider entry and choose **Properties**.

Check the following (leaving all other options unchecked):

Dynamic Parameters

Allow InProcess

Non transacted Updates

Verify the above options for proper operation of the provider.

The next step is to create the linked server.

Connecting DBAmp to SQL Server

Also, please see the note at the beginning of the chapter concerning the Microsoft issue of using **Network Service** as the SQL Server Service account.

DBAmp is designed to be used as a linked server. To install DBAmp as a linked server, use the SQL Management Studio and perform the following steps:

1. Using the SQL Server Management Studio, use the Object Explorer window and expand the **Server Objects** branch to display **Linked Servers.**

2. Right click on **Linked Servers** and choose **New Linked Server...**

   Enter the following information for the new Linked Server:

   General Page

   Linked Server: Enter **SALESFORCE**

   Provider: Choose **DBAmp OLE DB Provider**

   Product Name: Enter **DBAmp**

   Location: If you are connecting to a sandbox, enter **https://test.salesforce.com.** Otherwise, **leave blank.**

   Security Page

   Click **Be made using this security context:**

   For **Remote Login:,** enter your salesforce.com UserId.

   For **With password:** enter your salesforce password. If needed by your salesforce organization, append the salesforce security token to the end of the password. For more details on salesforce security tokens, see the Security Tokens section in the online salesforce help.

   Server Options

   Check the following are **true** (leaving all other options **false**):

   - **Collation Compatible**
   - **Data Access**
   - **Use Remote Collation**
   - **RPC Out**
   - **Enable Promotion of Distributed Transactions**

3. Press **OK** to create the SALESFORCE linked server.

Verifying the linked server

Use the following procedure to verify that the linked server is set up correctly:

Execute the following query using the SQL Management Studio:

Select * from SALESFORCE...sys_sfobjects

You should see a list of all your salesforce.com objects.

Install the DBAmp Stored Procedures

The next step to install DBAmp is to create a database and create the DBAmp stored procedures. The database you create contains not only the DBAmp stored procedures but also the local replicated tables you make from your live Salesforce.com data.

To install the DBAmp Stored Procedures:

1. 1. Using either the SQL Enterprise Manager or the SQL Management Studio, create a new database named **salesforce backups**. . This database will hold all the local replicated tables as well as the DBAmp stored procedures.

2. Open the file "**Create DBAmp SPROCS.sql**" in Query Analyzer or Management Studio but do not execute it yet. The file is located in the \Program Files\DBAmp\SQL directory.

   The stored procedures assume that you have installed DBAmp in the directory c:\"Program Files"\DBAmp. If you used an alternate drive or directory, you must find all occurrences of C:\"Program Files"\DBAmp\ and replace them with the correct directory.

3. Make sure that default database shown on the toolbar is the **salesforce backups** database (and not the **main** database). Then, execute (F5) to add the stored procedures to the database.

Running the DBAmp Configuration Program

In order for the DBAmp stored procedures to work properly, you must run the DBAmp configuration program and enter your SQL credentials along with any additional proxy information needed by DBAmp.

You must display the Options dialog and press OK for the settings to be saved (press OK even if you do not make changes).

Note: Normally, DBAmp handles the proxy automatically. If you are having trouble connecting or need to setup your proxy information manually, you can use the DBAmp Configuration Program to enter your proxy information.

To run the DBAmp Configuration Program:

1. From the Start menu, run the **DBAmp Configuration** program located under DBAmp. Under the **Configuration** menu, **select Options.**

2. Enter your SQL credentials. If you are using Windows Authentication, use the default value of **Trusted_Connection=Yes**

3. If you need to enter proxy information, check the **Use Proxy for Salesforce connection** checkbox.

4. Enter the appropriate proxy information:

   Proxy Username - Username for the proxy login.
   Proxy Password - Password for the above username.
   Proxy URL - Direct proxy URL.
   Proxy ConfigURL - Proxy script URL.

   When a script URL is set but the proxy address cannot be accessed, for example, the address is only available inside a corporate network but the user is logging in from home, DBAmp will use the direct URL if it has been set, or try a direct connection if the direct URL has not been set.

   If a direct URL is set and it cannot be accessed, DBAmp will not try a direct connection. This is the same behavior as Internet Explorer.

Click OK. The credentials are stored in encrypted form for use by the DBAmp stored procedures.

Pointing DBAmp to your Salesforce Sandbox Instance

By default, DBAmp points to your production Salesforce.com instance. If you need to change DBAmp to point to your Sandbox instance or need to use a different endpoint for DBAmp, alter the Location parameter of your linked server.

The Location parameter is normally blank. If your Sandbox Instance is at https://test.salesforce.com then you would enter https://test.salesforce.com for the Location Parameter on the linked server properties page.

Chapter 2: Using DBAMP as a Linked Server

When using DBAmp as a linked server, you can access salesforce.com tables as if they were SQL server tables.

Four Part Object Names

To refer to a salesforce.com object in a SQL statement, use the four part object name containing the name of the linked server and the object name separated by three periods. For example, to select all rows and columns of the Contact object:

Select * from SALESFORCE…Contact

The linked server name (SALESFORCE) and the table name (Contact) are case sensitive.

SQL versus SOQL

There are 2 ways to query real time data from salesforce: use the four part object name with SQL or use the OpenQuery clause with SOQL.

Using the four part object name and SQL

You may use the full Transact SQL syntax when entering SQL statements. Internally, SQL Server and DBAmp will translate your SQL statement into the appropriate SOQL statements for salesforce.com. Any elements that cannot be done in SOQL (like SQL functions) will be done locally by the SQL Server Distributed Query optimizer after retrieving the result set from salesforce.com.

The SQL Server Distributed Query Optimizer will choose a plan for every SQL statement that executes. Often, the plan chosen will be the most efficient and there will be no need to modify your SQL.

Should you suspect a poorly performing plan, use the Query Analyzer and enter the text of the SQL statement. Remember to use the 4 part naming convention for the Salesforce.com tables, i.e. SALESFORCE…Account.

For maximum performance when joining, consider using the OpenQuery clause with SOQL (described in the next section.

Note the following when using SQL:

- Do not enter unquoted date literals. Instead, use Transact SQL syntax for date literals (i.e. include quotes)
- For SOQL Boolean fields, use quoted literals ('false' instead of false).
- You may use * to indicate all columns.

- Following Transact SQL rules for where clause AND/OR precedence. Parentheses are only needed when explicit grouping is needed and are not required (unlike SOQL).
- User and Case are keywords in Transact SQL and must be quoted when used as a four part name to refer to the salesforce.com object. For example, specify the User Object as SALESFORCE…[User]

Using OPENQUERY and SOQL

When additional join performance is needed, consider using the OPENQUERY clause with DBAmp. Using OPENQUERY allows you to pass salesforce.com SOQL statements (not SQL) directly to DBAmp. A full description of the SOQL language can be found on the salesforce.com website at :

http://www.salesforce.com/us/developer/docs/api/index_Left.htm#StartTopic=Content/sforce_api_calls_soql.htm

Using OPENQUERY with SOQL can make dramatic performance differences on data that is joined. With SOQL, the join is performed back at the salesforce.com server as opposed to locally at the SQL server.

```
select * from openquery(salesforce,
'SELECT Type, BillingCountry,
    GROUPING(Type) grpType, GROUPING(BillingCountry) grpCty,
    COUNT(id) accts
FROM Account
GROUP BY CUBE(Type, BillingCountry)
ORDER BY GROUPING(Type), GROUPING(BillingCountry)')
```

- DBAmp currently supports both child to parent relationship queries and Parent to child queries.

For example,

```
select * from openquery(salesforce,
'SELECT Account.Name, (SELECT OwnerId FROM Account.Notes) FROM
Account')
```

```
select * from openquery(salesforce,
'SELECT Id, Who.FirstName, Who.LastName FROM Task');
```

- The where clause of the SOQL statement must be expressed using SOQL syntax, not SQL syntax.

For example,

```
Select * from OpenQUery(SALESFORCE,

'Select Opportunity.Account.Name,
Opportunity.Account.AnnualRevenue, Opportunity.Name,
Opportunity.CloseDate, Opportunity.StageName, Description,
Quantity

From OpportunityLineItem

Where (Opportunity.Account.AnnualRevenue >200 AND
Opportunity.CloseDate < 2009-08-29)')
```

is a supported SOQL statement because the date value is not in quotes.

```
Select * from OpenQuery(SALESFORCE,'SELECT Id FROM Account

WHERE Owner.CreatedDate = LAST_N_DAYS:200')
```

is also supported because it uses a SOQL date literal.

Note that datetime constants must be entered in ISO8601 format per the SOQL requirements.

Understanding hierarchical salesforce.com data when using OPENQUERY and SOQL

Note: The following is **only** applicable when using SOQL and OPENQUERY. When joining the linked server tables using standard SQL, the result table is constructed using normal relational concepts and not as describe here.

For OPENQUERY SOQL, DBAmp uses a special algorithm to "flatten" parent-child salesforce.com data into a two-dimensional table.

SQL Server results are two-dimensional with rows and columns. Because salesforce.com data can have more than two dimensions, a flattening algorithm is used to force the data into a two-dimensional format.

When flattening salesforce.com data in SQL Server, the column headings are an indication of the source of the column and essentially contain the navigation through the "tree" of returned data to get to that column. You can read the column structure backwards to get to the root object, the lookup objects, and related lists. For example, the column Account_LastModifiedBy_Alias is the Alias field of the LastModifiedBy lookup object for the Account root object.

There is a row of the root object for each object in a related list. When there are two related lists, the root object in the flattened result gets repeated by the sum of the count of all of the rows of the related lists. For example, if an Account root object has five Contacts and eight Cases, the root-object data is repeated in the result table thirteen times.

In the flattened result, fields of the Contact related list are shown with the root object, along with fields of the Cases related list and the root object. For rows where Contact data is returned, the Cases columns are null; for rows where Cases data is returned, the Contact columns are null. The fields are null because there really is no relationship between Contacts and Cases.

When the query contains a root object and multiple related lists, DBAmp repeats the root-object data, the sum of the count of all of the related lists. For example, if five related lists each had five items in them, the root object is repeated 25 times. Rows for related lists are displayed and the values in each row for the other related lists are null because they are not applicable.

Passing Parameters in SOQL queries

To use parameters in a SOQL query, you must use the EXECUTE statement of T-SQL. Here is an example:

```
CREATE TABLE RevByAccount
( Name   nvarchar(255) NULL,
  AnnualRevenue   decimal(18,0) NULL
);

DECLARE @MinRev INT
SET @MinRev = 20

INSERT RevByAccount
EXEC ( 'SELECT Name, AnnualRevenue FROM Account WHERE
AnnualRevenue > ?',
       @MinRev) AT Salesforce

go
```

Inserting rows using SQL

To insert new rows, use the standard SQL Insert statement. Do not include the read-only columns (i.e. Id, LastModifiedId, etc.) in the fields list. For example, to insert a new Note use the following SQL:

INSERT INTO SALESFORCE...Note (Body, IsPrivate, ParentId, Title)

VALUES('Body of Note 2','false', '00130000005ZsG8AAK','ToDelete')

For maximum scalability, please consider using the **sf_bulkops** stored procedure instead of SQL Insert statement. The **sf_bulkops** stored procedure takes advantage of the ability to batch together insert requests to the salesforce.com api.

Updating and Deleting rows using SQL

DBAmp supports updating and deleting Salesforce.com objects with SQL. In order to get the maximum performance with your UPDATE and DELETE statements, you need to understand how SQL Server handles UPDATE/DELETE statements with a linked server (like DBAmp).

For example, take the following SQL UPDATE

```
Update SALESFORCE...Account

    Set AnnualRevenue = 4000

    Where Id='00130000005ZsG8AAK'
```

Using the **Display Estimated Execution Plan** option from the **Query Analyzer,** you can see that SQL Server will retrieve the entire Account table from Salesforce and then search for the one row that has the Id of 00130000005ZsG8AAK. Then, SQL Server will update the AnnualRevenue of that row.

Obviously, this UPDATE statement has poor performance which gets worse as the size of the Account table grows. What we need is a way to retrieve only the row with Id 00130000005ZsG8AAK and then update the AnnualRevenue of that row. To do this, use an OPENQUERY clause as the table name.

```
Update OPENQUERY(SALESFORCE,

    'Select Id, AnnualRevenue from Account

     where Id=''00130000005ZsG8AAK'' ')

    set AnnualRevenue = 4000
```

Using an OPENQUERY clause insures that we retrieve only the row with the proper Id.

You can construct stored procedures that make your code more readable and that use the above technique. See the **Create SF_UpdateAccount.sql** file in the DBAmp program directory as an example. Using this stored procedure, we can do updates to the Account table using the following SQL:

```
exec SF_UpdateAccount '00130000008hz55AAA','BillingCity','''Denver'''
```

or

```
exec SF_UpdateAccount '00130000008hz55AAA','AnnualRevenue','20000'
```

You can use the SF_UpdateAccount stored procedure as a template for building your own specialized stored procedures. See the file **Create SF_UpdateAnnualRevenue.sql** for an example. Then, use the following SQL to update the Annual Revenue of an account.

exec SF_UpdateAnnualRevenue '00130000009DCEcAAO', 30000

Deleting rows with SQL has the same caveats. For best performance with deletion by Id, use an OPENQUERY clause in the SQL statement. An example of a stored procedure that deletes Accounts by Id is in the file **Create SF_DeleteAccount.sql.**

For maximum scalability, please consider using the **sf_bulkops** stored procedure instead of SQL Update or Delete statements. The **sf_bulkops** stored procedure takes advantage of the ability to batch together requests to the salesforce.com api.

Joining Salesforce.com Tables

Using joins, you can retrieve data from two or more tables based on logical relationships between the tables. Joins indicate how SQL Server should use data from one table to select the rows in another table.

Joins can be specified in either the FROM or WHERE clauses. The join conditions combine with the WHERE and HAVING search conditions to control the rows that are selected from the base tables referenced in the FROM clause.

Specifying the join conditions in the FROM clause helps separate them from any other search conditions that may be specified in a WHERE clause.

In addition, consider using the OPENQUERY and SOQL feature (see above) for maximum performance when joining to salesforce.com tables.

Analyzing Performance when Joining Tables

The SQL Server Distributed Query Optimizer will choose a plan for every SQL statement that executes. Often, the plan chosen will be the most efficient and there will be no need to modify your SQL.

Should you suspect a poorly performing plan, use the Query Analyzer and enter the text of the SQL statement. Remember to use the 4 part naming convention for the Salesforce.com tables, i.e. SALESFORCE…Account.

Choose the **Display Estimated Execution Plan** option from the **Query** menu to view the execution plan.

While a full discussion of execution plans is beyond this document, most SQL Select with join statements involving Salesforce.com data will choose to either return the entire result set of a table or read the needed rows with a parameterized query.

For example, consider the following SQL Select:

```
Select T1.Name, T2.Salutation, T2.FirstName, T2.LastName
from SALESFORCE...Account as T1, SALESFORCE...Contact as T2
where T1.Id = T2.AccountId and T1.AnnualRevenue > 20000
```

Here is the initial execution plan:

```
|--Hash Match(Inner Join, HASH:
 ([SALESFORCE]...[Account].[Id])=([SALESFORCE]...[Contact].[AccountId]),
RESIDUAL:([SALESFORCE]...[Contact].[AccountId]=[SALESFORCE]...[Accou
nt].[Id]))

     |--Remote Query(SOURCE:(SALESFORCE), QUERY:(SELECT T1."Id"
     Col1004,T1."Name" Col1005 FROM "Account" T1 WHERE
     T1."AnnualRevenue">(20000.0000)))

     |--Remote Query(SOURCE:(SALESFORCE), QUERY:(SELECT
     T2."AccountId" Col1007,T2."LastName" Col1010,T2."FirstName"
     Col1008,T2."Salutation" Col1011 FROM "Contact" T2))
```

This plan will bring down from the Salesforce.com server all of the Contact records. If most of our Accounts have Annual Revenue of > 20000, then the plan is efficient because most of the Contact records will be needed.

If, however, only 3 Accounts have AnnualRevenue > 20000 and the other 1000 Accounts do not, then the plan is inefficient. The Contact query will be retrieving more Contact records than we actually need to build the result set.

Let's change the SQL Select to use an inner remote join:

```
Select T1.Name, T2.Salutation, T2.FirstName, T2.LastName
from SALESFORCE...Account as T1
inner remote join SALESFORCE...Contact as T2 on T1.Id = T2.AccountId
where T1.AnnualRevenue > 20000
```

Now the execution plan shows a different choice.

```
  |--Nested Loops(Inner Join, OUTER
REFERENCES:([SALESFORCE]...[Account].[Id]))

     |--Remote Query(SOURCE:(SALESFORCE), QUERY:(SELECT T1."Id"
     Col1010,T1."Name" Col1011 FROM "Account" T1 WHERE
     T1."AnnualRevenue">(20000.0000)))

     |--Remote Query(SOURCE:(SALESFORCE), QUERY:(SELECT
     T2."Salutation" Col1007,T2."FirstName" Col1004,T2."LastName"
     Col1006 FROM "Contact" T2 WHERE T2."AccountId"=?))
```

In the Contact Query, we will now use a parameter in the query ("AccountID"=?) to read only the contact records we need. This is a much more efficient way to get the same result.

Using BIT datatype with DBAmp

When returning results to SQL Server, DBAmp must choose a datatype to use for salesforce.com Checkbox fields. By default, DBAmp uses VARCHAR(5) and populates the column with either the values of FALSE or TRUE.

If you are using SQL Server 2005 or later, you may wish to use the BIT datatype instead for salesfore.com Checkbox fields. Use RegEdit and alter the value of LOCAL_MACHINE/SOFTWARE/DBAmp/BitBoolean to a value of 1. Then restart SQL Server for the new value to take effect.

If you are replicating tables locally, you must run a replicate of those tables after changing this setting. This will recreate the tables using the BIT datatype.

Using Dates with DBAmp

When returning results to SQL Server, DBAmp converts Datetime values from UTC into the local timezone.

In addition, any datetime values used in a WHERE clause are assumed to be local times and not UTC times.

If you would prefer to have DBAmp always use UTC for all datetime values, you can modify the DBAmp registry settings with the following procedure. Note: this is not recommended but possible. Please contact forceAmp.com support to understand the ramifications of UTC and DBAmp.

1. Using the Start/Run option, run the **regedit** program.

2. Navigate to the following key: HKEY_LOCAL_MACHINE / Software / DBAmp .

3. Right click **DBAmp** and choose **New DWORD Value**. Name the key **NoTimeZoneConversion** (watch case and spelling).

4. Right click the newly created **NoTimeZoneConversion** and choose **Modify.** Then assign a value of 1.

Using DBAmp System Tables (sys_sf tables)

In addition to the Salesforce.com tables, DBAmp also provides various system tables that you can access with SQL SELECT statements. These tables are read-only; they cannot be updated or deleted.

Also, **Select** statements for these tables cannot contain a WHERE clause. If you need to use a WHERE clause, define a user-defined-function that encapsulates the table. See **Create DBAMP UDFS.sql** for an example.

Table Name	Contents
sys_sfsession Select * from SALESFORCE…sys_sfsession	The sys_sfsession table contains information about the current Salesforce.com session. Some of the columns in this table are: SessionId – Current Session Id OrganizationId – 18 char OrgId ServerURL – URL of SForce Server
sys_sfpicklists Select * from SALESFORCE…sys_sfpicklists	The sys_sfpicklists table contains information about the picklist values for each picklist field There is one row for each per picklist value. Some of the columns in this table are: ObjectName – Name of object FieldName – Field of the above object PickListValue – A single picklist value PickListLabel – Label for the above value
sys_sfobjects Select * from SALESFORCE…sys_sfobjects	The sys_sfobjects table contains information about the Salesforce.com objects. There is one row for each object in your organization. Some of the columns in this table are: Name – Name of object Createable – Is object createable ? Deletable – Is object deletable ? URLDetail – URL Detail for this object URLNew – URL New for this object

sys_sffields Select * from SALESFORCE…sys_sffields	The sys_sffields table contains information about the Salesforce.com object fields. There is one row for each object field in your organization. Some of the columns in this table are: ObjectName – Name of object Name – Name of the field Createable – Is the field insertable ? Type – Field Type using sf terminology SQLDefinition – SQL Column definition

Using Count() with salesforce.com objects

There are two methods of obtaining a row count of salesforce.com objects.

The first method uses the following SQL:

Select Count() from SALESFORCE…Account

This SQL statement executes by retrieving all the Id values of the object and counting the total number of Id values fetched. While this method performs quickly for small tables, large tables perform badly because all the Id's are fetched to the local SQL Server to be counted.

The second method performs much better because it takes advantage to the salesforce api SOQL Count function:

Select * from

OPENQUERY(SALESFORCE,'Select Count() from Account')

In the OPENQUERY clause, replace **SALESFORCE** with the name of your link server. Also, notice that the table name **Account** is NOT prefixed with "**SALESFORCE…**" .

Using DBAmp to convert currency amounts to a default currency

International organizations can use multiple currencies in opportunities, forecasts, reports, and other currency fields. The administrator sets the "corporate currency," which reflects the currency of the corporate headquarters.

If an organization is multicurrency enabled, you can configure DBAmp to convert currency fields to a single currency. DBAmp uses the default currency of the salesforce.com user id configured in the link server. DBAmp

converts currencies using the ConverCurrency() function of the salesforce.com API.

Note that **the default is NOT to convert currencies**. You must set the registry entry ConvertCurrency in the DBAmp hive for currency conversions to occur. The ConvertCurrency registry setting is found under the following registry key:

LOCAL_MACHINE\SOFTWARE\DBAmp\ConvertCurrency

A value of 1 causes the conversion to occur. A SQL restart is required after modifying this value.

SOQL statements entered via an OPENQUERY phrase do not honor this setting. If you need to convert currency inside an OPENQUERY, then use the CONVERTCURRENCY function:

```
select * from openquery(salesforce,
'Select Id, convertcurrency(annualrevenue), ToLabel(type)
from Account')
```

Using DBAmp to return translated values for picklists

If an organization uses multiple languages, you can configure DBAmp to return translated values for picklist fields by using the ToLabel function.

Note that **the default is NOT return translated values**. You must set the registry entry ToLabel in the DBAmp hive to use translated values. The ToLabel registry setting is found under the following registry key:

LOCAL_MACHINE\SOFTWARE\DBAmp\ToLabel

A value of 1 causes the ToLabel function to be used. A SQL restart is required after modifying this value.

SOQL statements entered via an OPENQUERY phrase do not honor this setting. If you need to return translated values inside an OPENQUERY, then use the ToLabel function:

```
select * from openquery(salesforce,
'Select Id, convertcurrency(annualrevenue), ToLabel(type)
from Account')
```

Retrieving Archived and Deleted records

Normally, the salesforce api does not return archived and deleted records as part of the result of a query. Therefore, the query result from DBAmp does not contain these records.

If you would like to include the archived and deleted records, add the _QueryAll prefix to the table name. For example, the following query retrieves only the task records that have been archived:

Select * from SALESFORCE…Task_QueryAll
where IsArchived = 'true'

You may also replicate all records including archived and task records to a local table by using the sf_replicateIAD stored procedure. See the SF_ReplicateIAD section in chapter DBAmp Stored Procedure Reference.

Using Column Subset views

Objects in salesforce that contain over 325 columns may produce an error when either replicated or refreshed. The error occurs because the maximum limit of the Select query statement in the salesforce api is 10,000 characters. A large number of columns in an object will produce a Select query larger than 10,000 characters.

The solution is to take advantage of Column Subset views. These views represent a user specified subset of the columns designed to 'fit' within the 10,000 character limit.

By attaching a specific suffix to the table name, DBAmp will include only those columns with names that fall within the alphabetic range. For example, the following SQL statement will return all columns with names beginning with any letter between A and M inclusive:

Select * from SALESFORCE…Account_ColumnSubsetAM

Some system columns are returned unconditionally for every subset view. The Id, SystemModstamp, LastModifiedDate, and CreatedDate columns are always returned.

The suffix must have the following format: a single underscore, the word ColumnSubset and two single letters indicating the alphabetic range.

In order to retrieve a full copy of the object data, use two or more column subset views. For example, to replicate a large Account using column subset views use the following command:

Exec sf_replicate 'SALESFORCE','Account_ColumnSubsetAM'

Exec sf_replicate 'SALESFORCE','Account_ColumnSubsetNZ'

Note that there is nothing special about the column partition used. Account_ColumnSubsetAK and Account_ColumnSubsetLZ would work equally as well.

Column Subset Views can be used in Select statements (but not OPENQUERY) as well as the sf_replicate and sf_refresh stored procedures.

DBAmp and Salesforce API call Counts

Like all third party salesforce.com tools, DBAmp uses the salesforce.com api to send and receive data from salesforce.com. Every salesforce.com customer has a limit on the total number of API calls they can execute, org wide, from all tools they are using. This limit is found on the Company Information screen in the salesforce.com application.

Here are some rough guidelines for api call counts for various operations in DBAmp:

SELECT against link server tables, SF_Replicate and SF_Refresh – DBAmp requests data in batches of 2000 records. The salesforce server may reduce that amount based on the width of the row. Our experience has been that the average batch size is 1000. So for every 1000 rows of data retrieved = 1 API call

UPDATE and INSERT statements – 1 api call for each record updated or inserted.

SF_Bulkops without the bulkapi switch – 1 api call for each batch of 200 records.

SF_Bulkops with the bulkapi switch – 1 api call for each batch of 10,000 records. If you use the batchsize option, then 1 api call per batchsize

There are other miscellaneous calls DBAmp makes to fetch schema data. These api calls are in addition to the above guidelines.

Chapter 3: Making Local Copies of Salesforce Data

One common usage of DBAmp is to make periodic copies of Salesforce.com data into a local SQL Server database. Using a combination of Microsoft SQL Server jobs scheduled by the SQL Server Agent and DBAmp, you can import data from Salesforce.com and make local replicated table copies.

Conceptually, the local replicated tables are all located in a single database that you create. On a schedule you setup, a job runs that backups the current local table into a table name ending with _Previous. The job then drops the previous replicated table, creates a new replicated table of the same name, and inserts all the rows from the corresponding table of the linked server.

You can setup retry options if the job is unable to run, perhaps delaying an hour and retrying again.

By default, DBAmp does not download the values of Base64 fields but instead sets the value to NULL. This is done for performance reasons. If you require the actual values, modify the Base64 Fields Maximum Size using the DBAmp Configuration Program to some value other than 0.

How to run the SF_Replicate proc to make a local copy

Now you are ready to run the stored procedure.

Note: The SF_Replicate stored procedure uses the xp_cmdshell command. If you are not an SQL Server administrator, you must have the proper permission to use this command. See the SQL Server documentation under the topic xp_cmdshell for more information. To quickly test, run the following sql in Query Analyzer:

Exec master..xp_cmdshell "dir"

To run the SF_Replicate stored procedure and make a local copy, use the following commands in Query Analyzer:

Use "salesforce backups"

Exec SF_Replicate 'SALESFORCE', 'Account'

where 'SALESFORCE' is the name you gave your linked server in at installation and Account is the Salesforce.com object to copy.

You can also setup a SQL Server job to run SF_Replicate on the schedule needed.

1. Go to the jobs subtree in Enterprise Manager and right click to create a new job.

2. Create a job with one job step with the following:

```
EXEC SF_Replicate  'SALESFORCE' , 'Account'
```

where **SALESFORCE** is the name of your linked server and **Account** is the name of the object. Be sure to set the database to the database you created earlier. Under the Advanced tab, setup the retry options. Also check **the Append output to job history** option.

3. Modify the job schedule for your execution schedule. You can also execute the job now by right-clicking the newly created job and choosing **Start Job**.

Viewing the job history

The output from the DBAmp stored procedures can be long and is often truncated in the normal job history. For this reason, you should modify the job step to retain the job output in a table or file.

To retain the entire step output, edit the job step and navigate to the Advanced tab. Check "Route to table" to have SQL Server retain the entire message output in a table.

To view the output, return to the Advanced tab and click View.

Replicating all Salesforce Objects

You can use the SF_ReplicateAll stored procedure to replicate all of your Salesforce objects (including custom objects). When run, the SF_ReplicateAll proc compiles a list of all existing salesforce objects and calls the SF_Replicate stored procedure for each object.

Salesforce objects that cannot be queried via the salesforce api with no where clause (like ActivityHistory) will NOT be included. In addition, Chatter Feed objects are also skipped by the sf_replicateall/sf_refreshall stored procedures because of the excessive api calls required to download those objects. You can modify the stored procedures to include the Feed objects if needed.

Note: SF_Replicate assumes that there are no foreign keys defined on the current set of local tables. If you have used the SF_CreateKeys stored procedure to define keys, you must drop those keys with the SF_DropKeys stored procedure prior to running SF_Replicate or SF_ReplicateAll. Later, you can recreate the keys using SF_CreateKeys. See the chapter entitled *Creating Database Diagrams and Keys* for more information.

How to run the SF_ReplicateAll proc to replicate all objects

Now you are ready to run the stored procedure.

To run the SF_ReplicateAll stored procedure and make a local copy, use the following commands in Query Analyzer:

Use "salesforce backups"

Exec SF_ReplicateAll 'SALESFORCE'

where 'SALESFORCE' is the name you gave your linked server in at installation.

You can also create a job to run the SF_ReplicateAll procedure on a periodic basis.

Copying only the rows that have changed

Once you have created an initial set of local, replicated tables, you can keep those tables up-to-date by using the SF_Refresh and SF_RefreshAll stored procedures. The SF_Refresh stored procedure attempts to 'sync' the local table with the Salesforce.com object without having to download the entire data for the object.

For more information, see the SF_Refresh and SF_RefreshAll stored procedure reference in the chapter entitled *DBAmp Stored Procedure Reference.*

Replicating Large Tables

Tables with large row counts (> 1 million) may require special handling with DBAmp. DBAmp has several methods to download the rows using different salesforce api's.

For most large tables using the salesforce bulkapi along with the pkchunking option is the most successful method. To use the pkchunk option, use the following syntax:

Exec SF_Replicate 'SALESFORCE', 'Account','pkchunk'

The normal batch size for pkchunk is 100,000 rows per batch. You can alter this with the following syntax:

Exec SF_Replicate 'SALESFORCE', 'Account','pkchunk,batchsize(50000)'

Here is a recommended order of options to try when replicating large tables:

1. **SF_Replicate** with no options. This command will use the salesforce web services api.

2. **SF_Replicate** with the pkchunk option. This command will use the salesforce bulkapi and the PKChunking header. The initial batch size will be 100K but you may need to reduce that to as low as 25,000 to get a successful result and avoid timeouts.

3. **SF_ReplicateLarge**. If both #1 and #2 fail, then try the SF_ReplicateLarge stored proc. The SF_ReplicateLarge proc uses the salesforce bulkapi and a local batching technique to bring down the rows. Again, try reducing the batch size if timeout errors occur.

Including Archived and Deleted rows in the local copy

To include archived and deleted rows, use **sf_replicateIAD** and **sf_refreshIAD.** Note that these stored procedures can only retrieve

deleted rows that are in the recycle bin. Rows that have been permanently deleted are not available with the salesforce.com api.

SF_ReplicateIAD will retain the permanently deleted rows from run to run. Once you begin to use SF_ReplicateIAD for a table, DO NOT USE sf_replicate on that table. If you run sf_replicate instead of sf_replicateIAD, you will lose all the permanently deleted rows in the local table.

Best Practices for Replicate and Refresh schedules

Most customers will run sf_replicate at night and use sf_refresh during the day.

If the schema of an object on salesforce is changing daily and the table is under 25,000 records, then use the 'Yes' option of sf_refresh on runs made during the day to force DBAmp to replicate the table and pick up the schema changes.

If the schema of an object on salesforce is changing daily and the table is greater than 25,000 records, then use the 'Subset' option of sf_refresh on runs made during the day. With this option, you can avoid time consuming replicates of large tables during the day while still keeping a subset of the columns up-to-date. A sf_replicate run that night will pick up the schema changes and the new data.

Our recommendation is to run sf_replicate either nightly or weekly. In the salesforce api, changes in formula fields will NOT be flagged as changed records. Therefore if you have formula fields on objects and only their value changes, the record will not be picked up by sf_refresh. This is because the salesforce api does not update the last modified date of that record for a formula field change. We therefore recommend that you run a sf_replicate on a nightly or weekly basis for your tables in order to pickup these modifications.

Large binary blobs may not be downloaded if their size is greater than MaxBase64Size in the DBAmp registry. See MaxBase64Size in the DBAmp Registry Settings chapter.

Chapter 4: Bulk Insert, Upsert, Delete and Update into Salesforce

Normal SQL Insert, Delete and Update statements are processed one at a time and are not sent in batches to Salesforce.com. To perform bulk operations use the **SF_BulkOps** stored procedure.

Conceptually, **the SF_BulkOps** proc takes as input a local SQL Server table you create that is designated as the "input" table. The input table name must begin with a valid Salesforce object name followed by an underscore and suffix. For example, **Account_Load** and **Account_FromWeb** are valid input table names. **XXX_Load** is not a valid input table name (XXX is not a valid Salesforce.com object).

Do not allow other applications to write to the input table while sf_bulkops is running.

Checking the Column Names of the Input Table

The input table must contain a column named **Id** defined as nchar(18) and a column named **Error** defined as nvarchar(255). In addition, the input table can contain other columns that match the fields of the Salesforce object.

For example, below is a valid definition of an Account_Load table:

Id	nchar(18)
Name	nvarchar(80)
Error	nvarchar(255)

Note that in this example, the Account_Load table does not contain most of the fields of the Account object.

How the input table is used depends on the operation requested. When using the above example table with an **Insert** operation, the missing fields are loaded as null values. When using the above example table with an **Update** operation, the Name field becomes the only field updated on the Salesforce side. When using the above example table with a **Delete** operation, the Name field is ignored and the objects with the Id value are deleted.

The **SF_BulkOps** proc looks at each field of the Salesforce object and tries to match it to a column name in the input table. One easy way to create a input table is to copy the definition of a table replicated by the **SF_Replicate** proc and add an Error column. Note that columns of the input table that do not match a field name are ignored. In addition, columns that match a computed fields (like SystemModstamp) are ignored if they exist in the input table.

The **SF_BulkOps** proc will identify column names of the input table that do not match with valid Salesforce.com column names and produce a warning message in the output. Note that in a properly constructed input table you may also have other columns in the input table that are for your own use and that should be ignored as input to **SF_BulkOps**. The **SF_ColCompare** stored procedure will also compare column names and identify errors without having to run **SF_BulkOps**.

You can easily have DBAmp generate a valid local table for any salesforce.com object by using the **SF_Generate** stored procedure. **SF_Generate** will automatically create an empty local table with all the proper columns of the salesforce.com object needed for that operation. See the chapter DBAmp Stored Procedure Reference for more information on **SF_Generate** and **SF_ColCompare**.

Using External Ids as Foreign Keys (without bulkapi switch)

You can use external ID fields as a foreign key, allowing you to bulk create, update, or upsert records in a single step instead of querying a record to get the ID first.

Note: This feature is not currently available when using the BulkAPI switch.

To do this, specify the external ID field name along with a colon and the external ID value. For example, let's look at bulk insert of contact records with the following table:

ID	LastName	AccountId	Error
	Emerson	0016000000G8lSsAAJ	
	Harrison	SAPXID__c:C01202	

In the first contact to be created ('Emerson'), the relationship to the Account is specified using a traditional 18 char id of the actual account.

The second contact to be created uses an external id field on the Account object (SAPXID__c) and tells DBAmp/Salesforce to lookup the needed salesforce.com AccountId by searching for an account where SAPXID__c is equal to C01202.

Note that the column name ('AccountId') does not change; we simply prefix the value with the external id field name and a colon.

You can use external ids as foreign keys when bulk inserting, updating, or upserting.

Using External Ids as Foreign Keys (with bulkapi switch)

You can use external ID fields as a foreign key, allowing you to bulk create, update, or upsert records in a single step instead of querying a record to get the ID first.

To do this, modify the column name of the input table and add a period followed by the external ID field name. For example, let's look at bulk insert of contact records with the following table:

ID	LastName	AccountId.SAPXID__c	Error
	Emerson	C01203	
	Harrison	C01202	

In the first contact to be created ('Emerson'), the relationship to the Account is specified using the SAP Id of C01203.

Note that when using the bulkapi switch, you do not prefix the value with a field name. Instead, you add the external id name to the column name of the table.

Also, you must use the external id value for all rows of the input table.

You can use external ids as foreign keys when bulk inserting, updating, or upserting.

Understanding the Error Column

For all rows that were successfully processed, sf_bulkops writes the phrase 'Operation Successful" to the Error column. Successfully processed rows can therefore be selected using the following SQL Select:

Select * from Account_Load where Error like '%Operation Successful%'

Rows that were not successfully processed will contain either a row specific error or nothing if there was a global failure.

Additional values appear in the Error column when using the BulkAPI switch. See **Error Handling when using the Bulk API** later in this chapter for details.

Bulk Inserting rows into Salesforce

When the operation requested is **Insert**, the **SF_BulkOps** reads each row of the input table, matches the columns to the fields of the Salesforce object, and attempts to insert the new object into Salesforce. **Important**: **SF_BulkOps** attempts to insert all rows of the load table regardless of any existing values in the Id and Error columns. In other words, the Id and Error columns are ignored on input when doing an **Insert** operation.

After execution of the **SF_BulkOps** proc, the Id column of the input table is overwritten with the Id assigned by Salesforce for each successfully inserted row. If the row could not be inserted, the Error column contains the error message for the failure.

Note: See the section Using the Bulk API with SF_BulkOps for important differences in Error column handling when using the BulkAPI switch.

Bulk Upserting rows into Salesforce

When the operation requested is **Upsert**, the **SF_BulkOps** reads each row of the input table, matches the columns to the fields of the Salesforce object, and attempts to upsert the new object into Salesforce. You must specify which field to use as the External Id field in the SF_BulkOps call. **Important**: **SF_BulkOps** attempts to upsert all rows of the load table regardless of any existing values in the Id and Error columns. In other words, the Id and Error columns are ignored on input when doing an **Upsert** operation.

After execution of the **SF_BulkOps** proc, the Id column of the input table is overwritten with the Id assigned by Salesforce for each successfully upserted row. If the row could not be upserted, the Error column contains the error message for the failure.

Note: See the section Using the Bulk API with SF_BulkOps for important differences in Error column handling when using the BulkAPI switch.

Bulk Updating rows into Salesforce

When the operation requested is **Update**, the **SF_BulkOps** reads each row of the input table, maps the columns to the fields of the Salesforce object, and attempts to update an object in Salesforce using the Id column of the input table.

Important: the input table should only contain columns for those fields that you want to update. . If the data in a column is an empty string or NULL, sf_bulkops will update that field on salesforce.com to be NULL. You may modify this behavior by using the following value for the operation: **Update:IgnoreNulls** . The **IgnoreNulls** option tells sf_bulkops to ignore null values in columns. However, empty string values will still set the field on salesforce.com to NULL.

For each row in the input table that failed to update, the Error column will contain the error message for the failure.

Note: See the section Using the Bulk API with SF_BulkOps for important differences in Error column handling when using the BulkAPI switch.

Bulk Deleting rows from Salesforce

When the operation requested is **Delete**, the **SF_BulkOps** reads each row of the input table and uses the Id field to delete an object in Salesforce.

For each row in the input table that failed to delete, the Error column will contain the error message for the failure.

Note: See the section Using the Bulk API with SF_BulkOps for important differences in Error column handling when using the BulkAPI switch.

Bulk UnDeleting rows from Salesforce

When the operation requested is **UnDelete**, the **SF_BulkOps** reads each row of the input table and uses the Id field to undelete an object in Salesforce.

You can identify deleted rows in a table with the following query:

Select Id from SALESFORCE…Account_QueryAll where IsDeleted='true'

Controlling the batch size

SF_BulkOps uses the maximum allowed batch size of 200 rows. You may need to reduce the batch size to accommodate APEX code on the salesforce.com server. To specify a different batch size, use the batchsize(xx) option after the operation.

For example, to set the batch size to 50:

Exec SF_Bulkops 'Update:batchsize(50)','Salesforce','User_Upd'

If you are also using the IgnoreNulls option, then separate the options with a comma:

Exec sf_bulkops 'Update:IgnoreNulls,batchsize(50)','Salesforce','User_Upd'

How to run the SF_BulkOps proc

Now you are ready to run the stored procedure.

Note: The SF_BulkOps stored procedure uses the xp_cmdshell command. If you are not an SQL Server administrator, you must have the proper permission to use this command. See the SQL Server documentation under the topic xp_cmdshell for more information. To quickly test, run the following sql in Query Analyzer:

Exec master..xp_cmdshell "dir"

To run the **SF_BulkOps** stored procedure, use the following commands in Query Analyzer. Be sure your default database is **salesforce backups**.

Exec SF_BulkOps 'Insert', 'SALESFORCE', 'Account_Load'

Or

Exec SF_BulkOps 'Upsert','SALESFORCE','Account_Load', 'ED__c'
(where ED__c is the name of the external id field)

Exec SF_BulkOps 'Delete', 'SALESFORCE', 'Account_Load'

Or

Exec SF_BulkOps 'Update', 'SALESFORCE', 'Account_Load'

Exec SF_BulkOps 'UnDelete', 'SALESFORCE', 'Account_Load'

where 'SALESFORCE' is the name you gave your linked server in at installation and Account_Load is the name of the input table to use.

Similar to the **SF_Replicate** proc, you can schedule the **SF_BulkOps** proc using the SQL Server job agent.

How to run the SF_BulkOps proc without using xp_cmdshell

In some SQL Server environments, the use of xp_cmdshell may be restricted. In this case you can use a CmdExec feature of the SQL job step to run the underlying bulkops program directly (i.e. instead of using the sf_bulkops stored procedure). The name of the exe is DBAmp.exe and it is located in the DBAmp Program Files directory. Normally the directory is c:\Program Files\DBAmp but DBAmp may installed in a different location.

The DBAmp.exe program takes the following 7 parameters:

1. **Operation**: Must be either **Insert**, **Delete**, **Update** or **Upsert**. This is similar to the first parameter of sf_bulksops. Batchsize and other options are handled the same way as the sf_bulkops proc.

2. **Input Table**: The name of the local SQL table containing the data.

3. **SQL Server Name**: The name of the SQL instance to connect to.

4. **SQL Database Name**: The name of the database to connect to. Enclose in double quotes if the name contains a blank.

5. **Link Server Name**: The name of the DBAmp link server.

6. **External Id Colum (Optional):** The name of the external Id column to use when the operation is Upsert. Do not include this parameter for other operations.

Here is an example of a complete command:

```
"C:\Program Files\DBAmp\DBAmp.exe"  Update Account_Load BUDDY
"salesforce backup" SALESFORCE
```

Note that even though the command appears on multiple lines in this document, the command must be entered as a single line in the job step. Also notice the use of double quotes around both the program and the database. This is required because those values contain blanks.

When setting up a job step to call the program directly, you must change the **Type** of the job step to: **Operating System (CmdExec).** Then enter your complete command in the Command text box. Again, the command must be on a single line.

The DBAmp.exe program returns 0 for a successful completion and -1 if any rows failed. Ensure that the **Process exit code of a successful command** is 0 (zero). A -1 will be returned for situations where some of the rows succeeded and some failed. Use the error column of the table to determine the failed rows. Rows that succeeded do not need to be resubmitted.

Below is a screen shot of a sample job step calling the DBAmp.exe.

Your command may be different depending on the install directory.

Understanding SF_Bulkops failures (Web Services API)

Note: See the section Using the Bulk API with SF_BulkOps for important differences in failure handling when using the BulkAPI switch.

When individual rows of the input table fail to complete the operation, sf_bulkops writes the error message back to the Error column of that row and continues processing the next row. Thus, in a batch of 200 rows it is possible that 175 rows were successful and 25 rows failed.

The sf_bulkops stored procedure outputs an error message in the log indicating the sf_bulkops failed when 1 or more rows failed. The correct interpretation of this error message is that at least 1 row of the input table contained an error. Rows that have a blank error message were still successful. In addition, sf_bulkops outputs messages indicating the total number of rows processed the number of rows that failed and the number of rows that succeeded.

If sf_bulkops is run in a job step, then the job step will fail if one or more rows contain an error. Again, the rows that contain a blank error message were still successful; the failure is thrown to indicate to the operator that at least one row failed.

Using the Bulk API with SF_BulkOps

There are two different API's available from salesforce.com that applications can use to push data : the Web Services API or the Bulk API. You can use either API with SF_BulkOps with the Web Services API being the default.

The Web Services API is synchronous, meaning that for every 200 rows that are sent to salesforce, an immediate response is sent indicating the success or failure of those 200 rows. SF_BulkOps has traditionally used the Web Services API. The disadvantage of this API is that the maximum number of rows that can be sent to salesforce at a time is 200. So if the input table to SF_BulkOps contains 1000 rows, there will be at least 5 API calls to send the data to the salesforce.com server.

The Bulk API is asynchronous, meaning that rows sent to salesforce.com are queued as a job. The job is executed at some time in the future. The application must enquire about the status of the job at a later time to retrieve the success or failure of the rows sent. The advantage of the Bulk API is that up to 10,000 rows can be sent in a single request or API call. An input table of 5000 rows would require a single API call to send the data, along with API calls to retrieve the status at some point in the future.

By default, SF_BulkOps uses the Web Services API. To use the Bulk API, add the BulkAPI switch to the operation parameter of the SF_BulkOps call:

Exec SF_BulkOps 'Insert:bulkapi', 'SALESFORCE', 'Account_Load'

Because the Bulk API is asynchronous, the error column is populated with a tracking token indicating the job and batch id for that row along with the current status. For example, the following Error value indicates that this row has been submitted to salesforce.com but the result is currently unknown:

BulkAPI:Insert:750600000004DbhAAE:751600000004FJaAAM:1:Submitted

After the job completes on salesforce, SF_BulkOps populates the Error column with the success or failure of the operation on that row. For successfully processed rows, then the Error is changed to indicate a successful operation:

BulkAPI:Insert:750600000004DbhAAE:751600000004FJaAAM:2:Operation Successful.

You can use this to remove successfully processed rows from the table:

```
Delete Account_Load where Error like '%Operation Successful%'
```

Any rows remaining either have not been processed yet or have failed to process. The error message associated with the failure is written to the Error value:

BulkAPI:Insert:750600000004DbhAAE:751600000004FJaAAM:2:Error:
INVALID CROSS REFERENCE ID

DBAmp will poll salesforce every 1 minute until the job completes on salesforce. Then the final success or error message will be written to the Error column.

If you prefer not to have DBAmp poll for the status (i.e "fire and forget") then add the phrase (ns) after the bulkapi option: **Insert:bulkapi(ns)**

Putting this all together, the workflow for using SF_BulkOps with the Bulk API is:

1. Call SF_Bulkops to submit a job to salesforce to process the data and wait for the job to complete. This command will not return until the job completes on salesforce.com. It will poll salesforce every 1 minute to determine if the job has completed.

Exec SF_BulkOps 'Insert:bulkapi', 'SALESFORCE','Account_Load'

2. Remove the successful records from the table with the following command:

```
Delete Account_Load

where Error like '%Operation Successful%'
```

Examine the remaining rows in the table and determine the failure using

Controlling the batch size with the Bulk API

The maximum allowed batch size when using the Bulk API is 10,000 rows. By default, the Bulk API uses a batch size of 5000 rows. You may need to reduce the batch size to accommodate APEX code on the salesforce.com server. To specify a different batch size, use the batchsize(xx) option after the operation.

For example, to set the batch size to 2500:

Exec SF_Bulkops 'Update:bulkapi,batchsize(2500)','Salesforce','User_Upd'

Understanding a Sort Column when using the Bulk API

For maximum performance when using the bulkapi option, the load table should have also have a Sort column. Here is a quick way you can add a Sort column to your load table. Assume that the load table is named Account_upd1:

```
Alter table Account_upd1
Add [Sort] int identity (1,1)
```

This adds a Sort column to the table that is a consecutive integer number. Having this column will dramatically improve the run time for large (>50,000 rows) operations.

In addition, the sort column can be used to reduce locking issues on salesforce. Salesforce recommends ordering a detail load table by the master record id to improve locking (See https://developer.salesforce.com/page/Loading_Large_Data_Sets_with_the_Force.com_Bulk_API).

Suppose you are uploading Contact records using a load table named Contact_upd1. In this case, you could create a Sort column as an nchar(18) field:

```
Alter table Contact_upd1
Add [Sort] nchar(18) null
```

Then populate the Sort column with the value of the AccountId field:

```
Update Contact_upd1 Set Sort = AccountId
```

SF_Bulkops will send the records to salesforce in AccountId order to reduce locking when inserting the contacts.

Using the HardDelete operation with the Bulk API

When using the Bulk API, there is an additional operation available called HardDelete. With the HardDelete operation, the deleted records are not stored in the Recycle Bin. Instead, they become immediately available for deletion. The administrative permission for this operation, Bulk API Hard Delete, is disabled by default and must be enabled by an administrator. A Salesforce user license is required for hard delete.

Exec SF_Bulkops 'HardDelete:bulkapi','Salesforce','Account_Delete'

Controlling Concurrency Mode with the Bulk API

By default, the Bulk API uses a concurrency mode of Serial. This guarantees that batches are processed one at a time.

You can request processing in parallel using the parallel option but this option may cause locking issues on the salesforce.com server. When this is severe, the job may fail. If you're experiencing this issue, submit the job with serial concurrency mode.

To use parallel concurrency mode (instead of the default serial mode):

Exec SF_Bulkops 'Update:bulkapi,parallel','Salesforce','User_Upd'

Using Optional SOAP Headers

The salesforce api allow you to pass additional SOAP Headers that alter the behavior of the sf_bulkops operation. The SOAP Headers are described in

detail in the salesforce.com api documentation:
http://www.salesforce.com/us/developer/docs/api/Content/soap_headers.htm

The headers are specified in the form of 3 values separated by commas. The first value is the header name, the next value is the section name and the last value is the value for the section. The entire parameter is enclosed in quotes. The salesforce.com api is case sensitive with respect to these values; use the exact token given in the salesforce.com documentation.

For example, to use the default assignment rule for these inserted Leads you would add the following SOAP Header parameter:

```
exec sf_bulkops 'Insert','SALESFORCE','Lead_Test','AssignmentRuleHeader,useDefaultRule,true'
```

The DBAmp Registry settings can also be used to add SOAP headers. The difference is the SOAP header parameter on the sf_bulkops call is a "one-time" use. The DBAmp Registry settings apply the SOAP header to all operations of DBAmp. Therefore, using the SOAP header parameter allows a finer control over the header usage.

Here are some other examples of SOAP headers:
Trigger auto-response rules for leads and cases: 'EmailHeader,triggerAutoResponseEmail,true'
Changes made are not tracked in feeds: 'DisableFeedTrackingHeader,disableFeedTracking,true'

Note: SOAP Headers cannot be used with the bulkapi switch of sf_bulkops.

Converting Leads with SF_Bulkops

SF_BulkOps can be used to convert lead records to accounts/contacts/opportunities.

The first step is to create a table to hold the information needed for the conversion. At minimum the table needs to have the following columns:

```
CREATE TABLE [dbo].[LeadConvert](
    [LeadId] [nchar](18) NULL,
    [convertedStatus] [nvarchar](255) NULL,
    [Error] [nvarchar](512) NULL,
    [AccountId] [nchar](18) NULL,
    [OpportunityId] [nchar](18) NULL,
    [ContactId] [nchar](18) NULL
) ON [PRIMARY]
```

Additional columns listed below may be added to the table if the functionality of the column is needed.

Name	Type	Description
accountId	nchar(18) NULL	ID of the Account into which the lead will be merged. Required only when updating an existing

Name	Type	Description
		account, including person accounts. If no `accountID` column is specified, then the API creates a new account. DBAmp will populate this column with the ID of the newly created Account.
contactId	nchar(18) NULL	ID of the Contact into which the lead will be merged (this contact must be associated with the specified accountId, and an accountId must be specified). Required only when updating an existing contact. Important If you are converting a lead into a person account, do not specify the contactId or an error will result. Specify only the accountId of the person account. If no `contactID` is specified, then the API creates a new contact that is implicitly associated with the Account. DBAmp will populate this column with the ID of the newly created Contact.

convertedStatus	nvarchar(255)
	NULL
		Valid LeadStatus value for a converted lead. Required. To obtain the list of possible values, you must query the LeadStatus object. For example:
		
		`Select Id, MasterLabel`
		
		`from SALESFORCE...LeadStatus where IsConverted=true`

doNotCreateOpportunity	varchar(5) NULL	Specifies whether to create an Opportunity during lead conversion (`false`, the default) or not (`true`). Set this flag to `true` only if you do not want to create an opportunity from the lead. An opportunity is created by default.
leadId	nchar(18) NULL	ID of the Lead to convert. Required.
opportunityId	nchar(18) NULL	DBAmp populates the field with the Id of the newly created Opportunity
opportunityName	nvarchar(80) NULL	Name of the opportunity to create. If this column is not included, then this value defaults to the company name of the lead.
overwriteLeadSource	varchar(5) NULL	Specifies whether to overwrite the LeadSource field on the target Contact object with the contents of the LeadSource field in the source Lead object (`true`), or not (`false`, the default). To set this field to `true`, you must specify a contactId for the target

Name	Type	Description
		contact.
ownerId	nchar(18) NULL	Specifies the ID of the person to own any newly created account, contact, and opportunity. If the client application does not specify this value, then the owner of the new object will be the owner of the lead.
sendNotificationEmail	varchar(5) NULL	Specifies whether to send a notification email to the owner specified in the ownerId (`true`) or not (`false`, the default).

Use the following command to convert leads:

Exec SF_BulkOps 'ConvertLead', 'SALESFORCE', 'LeadConvert'

Be sure to examine the error column after running the command to look for possible errors that may have occurred.

Chapter 5: Using SSIS with DBAmp

DBAmp can be used with SSIS to build complex integrations. Within SSIS, you can use DBAmp in two ways:

- Directly connecting to DBAmp **to pull data from salesforce.com**

- Connecting to SQL Server and using the link server **to push data to salesforce.com.**

Create a Connection for DBAmp

In order to use DBAmp in any integration project, you must first create a new OLE DB Connection that uses the DBAmp provider .

1. Right click in the **Connection Managers** panel and choose **New OLE DB Connection.** When the **Configure OLE DB Connection Manager** dialog, click the **New** button.

2. The **Connection Manager** dialog is displayed. Enter the following information:

   Provider: DBAmp OLE DB Provider
   Location: Leave blank to connect to production org. For sandbox orgs use https://test.salesforce.com
   User name: Your salesforce.com user id
   Password: Your salesforce.com password. Include the security token if needed.
   Allow saving password: Check this box.

   Click the Test Connection button and correct any errors as needed.

3. Click OK to save the new connection. The new connection should now appear in the **Connection Managers** panel.

4. Optionally, right click on the newly created connection and rename to a friendlier name.

Using DBAmp as an OLE DB Source

SSIS can connect directly to DBAmp to pull data from salesforce.com. Use the following steps to create a Data Flow task in SSIS that reads data from salesforce.com using DBAmp:

1. While in the Control Flow panel, drag and drop a **Data Flow Task** from the Toolbox. Right click on the new Data Flow Task and choose **Edit**. The Data Flow panel should now be displayed

2. From the Toolbox, drag and drop the **OLE DB Source** item onto the edit panel. Right click the new **OLE DB Source** item and choose **Properties**.

3. Set the **AlwaysUseDefaultCodePage** property to TRUE. This must be done for the DBAmp OLE DB Source to work correctly.

4. Now, right click on the **OLE DB Source** item and choose **Edit**. Set the **OLE DB Connection Manager** to the DBAmp connection created above.

5. **Data Access Mode** can be either a Table or View or a SQL command.

   When using a SQL command, remember **that DBAmp is expecting SOQL (not SQL). Do not use the Build Query button.**

   Instead, type your **SOQL** statement directly into the **SQL Command Text** field.

   A full description of the SOQL language can be found on the salesforce.com website at :

http://www.salesforce.com/us/developer/docs/api/index_Left.htm#StartTopic=Content/sforce_api_calls_soql.htm

This OLE DB Source can now be used as the source of the data flow.

Pushing Data to Salesforce.com using SSIS

The most scalable way to push data to salesforce.com is the sf_bulkops stored procedure. The SF_Bulkops stored procedure is described in detail in the chapter titled **Bulk Insert, Upsert, Delete and Update into Salesforce.**

In SSIS, you can use the **Execute SQL Task** to call the SF_Bulkops stored procedure. The connection manager for the task should be a connection to the SQL Server (NOT the DBAmp OLE DB provider). The SQL Source Type should be Direct Input and the SQL Statement should be the call to the SF_BulkOps stored procedure.

The **Execute SQL Task** that contains the SF_BulkOps call normally has 2 precedence constraints: 1 for SUCCESS and 1 for FAIL.

You can use the Precedence Constraints to direct flow based on the SF_BulkOps outcome. SF_Bulkops (and therefore the **Execute SQL Task**) fails if any row of the table cannot be processed successfully. If only a partial number of rows succeed, the FAIL precedence constraint fires. When this occurs, you can identify the successful rows by using the following SQL:

Select * from Account_SSISUpdate
where Error like '%Operation Successful%'

Chapter 6: Uploading files into Content, Documents and Attachments

You can use DBAmp to upload files into salesforce.com as Content, Documents or Attachments with the SF_Bulkops stored procedure. When you place a file path in the VersionData or Body column, SF_Bulkops will use the path to obtain the data needed.

SALESFORCE guidelines for uploading documents in ContentVersion object:

"To create a document, create a new version via the ContentVersion object without setting the ContentDocumentId. This automatically creates a parent document record. When adding a new version of the document, you must specify an existing ContentDocumentId which initiates the revision process for the document. When the latest version is published, the title, owner, and publish status fields are updated in the document."

To upload Content, use the following steps:

1. Use the SF_Generate stored procedure to generate a table to be used for the upload. See SF_Generate in the Stored Procedure reference for more details on SF_Generate.

   ```
   exec sf_generate 'Insert','SALESFORCE','ContentVersion_Load'
   ```

2. Using SQL, modify the VersionData column type to be a nvarchar(500) instead of an image type.

   ```
   Alter table ContentVersion_Load Drop Column VersionData
   ```

   ```
   Alter table ContentVersion_Load Add VersionData nvarchar(500) null
   ```

3. Insert rows into ContentVersion_Load with the following values:

   - Title - file name.
   - ContentDocumentId – ID of the document.
   - Origin -The source of the content version. Valid values are:
     - C—This is a Content document from the user's personal library. Label is Content. The FirstPublishLocationId must be the user's ID. If FirstPublishLocationId is left blank, it defaults to the user's ID.
     - H—This is a Chatter file from the user's My Files. Label is Chatter. The FirstPublishLocationId must be the user's ID. If FirstPublishLocationId is left blank, it defaults to the user's ID. Origin can only be set to H if Chatter is enabled for your organization.

This field defaults to C. Label is Content Origin.

- OwnerId - ID of the owner of this document.
- Description - (optional) file or link description.
- VersionData - complete file path on the local drive of the computer where DBAmp is installed. For example: c:\serialnumber.txt
- PathOnClient - complete file path on the local drive of the computer where DBAmp is installed.
- ContentUrl - URL (for uploading links only, leave blank for files).
- FirstPublishLocationId - workspace ID.
- RecordTypeId - content type ID. If you publish to a workspace that has restricted content types, you must specify RecordTypeId.

4. Upload the table to salesforce.com with SF_Bulkops. SF_Bulkops will automatically read the file using the location found in the VersionData column and pass the contents to salesforce as the file.

   Note: You cannot use the bulkapi switch when uploading content with sf_bulkops.

   ```
   exec sf_bulkops 'Insert','SALESFORCE','ContentVersion_Load'
   ```

5. Check the Error column of ContentVersion_Load table for any error messages that may have occurred during the upload.

To upload Attachments, use the following steps:

1. Use the SF_Generate stored procedure to generate a table to be used for the upload. See SF_Generate in the Stored Procedure reference for more details on SF_Generate.

   ```
   exec sf_generate 'Insert','SALESFORCE','Attachment_Load'
   ```

2. Using SQL, modify the Body column type to be a nvarchar(500) instead of an image type.

   ```
   Alter table Attachment_Load Drop Column Body

   Alter table Attachment_Load Add Body nvarchar(500) null
   ```

3. Insert rows into Attachment_Load with the following values:

   - Name - file name.
   - Description - (optional) file description.

- Body- complete file path on the local drive of the computer where DBAmp is installed. For example: c:\serialnumber.txt
- IsPrivate - false/true
- OwnerId - (optional) file owner, defaults to the user uploading the file.
- ParentId – ID of the parent object of the attachment. The following objects are supported as parents of attachments:

  Account, Asset, Campaign, Case, Contact, Contract, Custom objects, EmailMessage, EmailTemplate, Event, Lead, Opportunity, Product2, Solution, Task.

4. Upload the table to salesforce.com with SF_Bulkops. SF_Bulkops will automatically read the file using the location found in the Body column and pass the contents to salesforce as the file.

   Note: You cannot use the bulkapi switch when uploading attachments with sf_bulkops.

   ```
   exec sf_bulkops 'Insert','SALESFORCE','Attachment_Load'
   ```

5. Check the Error column of Attachment_Load table for any error messages that may have occurred during the upload.

Chapter 7: Creating Database Diagrams and Keys

Using DBAmp and a database diagramming tool, you can construct Database Diagrams of Salesforce.com tables like the example below. DBAmp works with all major ERD and database diagramming tools.

Creating a Primary Key

Do not use **SF_CreateKeys** if all you want is a permanent Primary Key on the ID field of the tables. Instead, the **SF_Replicate** stored procedure will automatically create the primary key on the ID field of every table it replicates.

Creating Foreign Keys

The table can have many foreign keys. The foreign keys created by **SF_CreateKeys** are disabled and will not be enforced by SQL Server. This is because salesforce.com allows a field of a table to reference multiple other tables. For example, the field ParentId on the Attachment table can refer to Id field of six or more other tables. It would not be possible for SQL Server to enable this as a foreign key.

Creating a Database Diagram

Creating database diagrams is a 4 step process:

1. Replicate the needed tables using **SF_Replicate**. This creates a local table with a primary key.
2. Use the DBAmp stored procedure **SF_CreateKeys** to add the foreign keys to the local replicated tables.
3. Use the ERD tool of choice (like SQL Enterprise Manager's Data Diagrams) to build a diagram from the tables and keys.
4. Drop the foreign keys using the stored procedure **SF_DropKeys**. Failure to remove the foreign keys from the table causes problems with the later replication of the table.

There are two DBAmp stored procedures for key creation and deletion. They are **SF_DropKeys**, which drops all foreign keys on the local tables in the database and **SF_CreateKeys**, which creates the foreign key constraints on the same tables. These procedures work only on the local tables that appear to be replicated copies of Salesforce.com tables

Note that **SF_CreateKeys** will only create foreign keys for **existing** local tables; the procedure does not create the local table itself. Therefore, you must replicate down either all the salesforce.com tables (using **SF_ReplicateAll**) or a subset of salesforce.com tables (using **SF_Replicate**) prior to running **SF_CreateKeys.**

In addition, the foreign keys should not exist when running SF_Replicate or SF_ReplicateAll. **Therefore, we recommend that you only use SF_CreateKeys and SF_DropKeys when you need to build a database diagram.** The procedure to build the diagram is:

1. Create the database to hold the local replicated tables.
2. Run **SF_ReplicateAll** to make a complete local set of replicated tables.

3. Run **SF_CreateKeys** to add the foreign keys to the local tables.
4. Create the database diagrams as needed using the ERD tool of your choice or SQL Management Studio.
5. Run **SF_DropKeys** to drop the foreign keys.

Chapter 8: Using Excel with Views to Linked Server Tables

When accessing the linked server from Excel or other programs, you are really accessing SQL server and then using SQL Server to access the linked tables. To avoid four part object names in this scenario, use the following scripts to create views of the linked server tables.

Create Views of the SALESFORCE linked server tables

The SF_CreateViews procedure is a stored proc that can be run every night and it will automatically create views for those that don't exist and drop/recreate the views that do exist.

To use this stored procedure in Query Analyzer:

1. Create a new SQL Server database to contain the view definitions. Name this new database **SFViews** . Navigate to or create a database that will contain the views.

Open the 'Create Views.sql' file located in the DBAmp installation directory. Ensure that you are using the proper database (check the QA Toolbar), then press F5 to add the SF_CreateViews stored procedure to the database.

2. As often as needed, run the following to create the views:

   exec SF_CreateViews 'SALESFORCE'

   where 'SALESFORCE' is the name of your linked server. The stored procedure will create view definitions in the new database for each of the salesforce.com objects. The view name will be the object name with _View appended (Account_View).

Using Excel

With the views created, you can now easily import data into Excel spreadsheets and pivot tables. Here's how to do it:

1. To import data to a spreadsheet, choose **New Database Query…** from the **Data – Import External Data** menu.

To import data to a pivot table, choose **Pivot Table and Pivot Chart Report** from the menu and click **External Data** on the dialog. Then click the **Get Data** button.

2. If you have already created a data source for Salesforce.com, skip to step 6. If not, check **Use the Query Wizard**, choose **<New Data Source>** and click **OK**.

3. Name the new data source, select the **SQL Server** driver and click **Connect**.

4. Enter the required connection information for your SQL Server. Click the Options button and select the SFViews database (the database created earlier in the chapter).

5. Do **not** select a default table. Click **OK**.

6. Select the data source you created in the previous steps and click OK.

7. When the **Query Wizard – Choose Columns** dialog appears, click **Cancel**. Click Yes on the next dialog to continue editing the query in Microsoft Query.

8. Finally, use Microsoft Query to build a query from the Salesforce.com views by dragging and dropping columns from the views. Consult the Microsoft Query help for information on how to join tables. Also, review the information on joining Salesforce.com tables in Chapter 2.

Chapter 9: DBAmp Stored Procedure Reference

SF_BulkOps

Usage

SF_BulkOps takes as input a local SQL Server table you create that is designated as the "input" table. The input table name must begin with a valid Salesforce object name followed by an underscore and suffix. For example, **Account_Load** and **Account_FromWeb** are valid input table names. **XXX_Load** is not a valid input table name (XXX is not a valid Salesforce.com object).

The input table must contain a column named **Id** defined as nchar(18) and a column named **Error** defined as nvarchar(255). In addition, the input table can contain other columns that match the fields of the Salesforce object. SF_BulkOps produces warning messages for all columns that do not match a field in the salesforce.com object. Non-matching columns are not considered an error because you may want to have column data in the table for reference but that should be intentionally ignored.

Do not allow other applications to write to the input table while sf_bulkops is running.

NOTE: There are two different API's available from salesforce.com that applications can use to push data : the Web Services API or the Bulk API. You can use either API with SF_BulkOps with the Web Services API being the default.

The Web Services API is synchronous, meaning that for every 200 rows that are sent to salesforce, an immediate response is sent indicating the success or failure of those 200 rows. SF_BulkOps has traditionally used the Web Services API. The disadvantage of this API is that the maximum number of rows that can be sent to salesforce at a time is 200. So if the input table to SF_BulkOps contains 1000 rows, there will be at least 5 API calls to send the data to the salesforce.com server.

The Bulk API is asynchronous, meaning that rows sent to salesforce.com are queued as a job. The job is executed at some time in the future. The advantage of the Bulk API is that up to 10,000 rows can be sent in a single request or API call. An input table of 5000 rows would require a single API call to send the data, along with API calls to retrieve the status at some point in the future.

By default, SF_BulkOps uses the Web Services API.

The **SF_Generate** stored procedure can be used to quickly build input tables for **SF_BulkOps.**

The **SF_ColCompare** stored procedure can be used to compare 'hand built' tables against the salesforce.com object to ensure correct column names.

SF_BulkOps can perform one of twelve operations:

1. **Insert** – When the operation requested is **Insert**, the **SF_BulkOps** reads each row of the input table, matches the columns to the fields of the Salesforce object, and attempts to insert the new object into Salesforce. **Important**: **SF_BulkOps** attempts to insert all rows of the load table regardless of any existing values in the Id and Error columns.

2. **Insert:BulkAPI** – Insert rows from the table using the Bulk API instead of the Web Services API.

3. **Upsert** - When the operation requested is **Upsert**, the **SF_BulkOps** reads each row of the input table, matches the columns to the fields of the Salesforce object, and attempts to upsert the new object into Salesforce using the specified external id field. **Important**: **SF_BulkOps** attempts to upsert all rows of the load table regardless of any existing values in the Id and Error columns.

4. **Upsert:BulkAPI –** Upsert row using the Bulk API instead of the Web Services API.

5. **Update** – When the operation requested is **Update**, the **SF_BulkOps** reads each row of the input table, maps the columns to the fields of the Salesforce object, and attempts to update an object in Salesforce using the Id column of the input table.

   Important: the input table should only contain columns for those fields that you want to update. If the data in a column is an empty string or NULL, sf_bulkops will update that field on salesforce.com to be NULL. You may modify this behavior by using the following value for the operation: Update:IgnoreNulls . The IgnoreNulls option tells sf_bulkops to ignore null values in columns. However, empty string values will still set the field on salesforce.com to NULL.

6. **Update:BulkAPI** – Update salesforce objects using the Bulk API instead of the Web Services API.

7. **Delete** - When the operation requested is **Delete**, the **SF_BulkOps** reads each row of the input table and uses the Id field to delete an object in Salesforce.

8. **Delete**:**BulkAPI** – Delete objects in salesforce using the Bulk API instead of the Web Services API.

9. **HardDelete:BulkAPI** – Delete objects in salesforce using the Bulk API. In addition, the deleted records are not stored in the Recycle Bin.

10. **Status** – Populate the Error column with the current job/batch status. This is used when using BulkAPI operations to determine the result of the operation.

11. **ConvertLead** – Converts Lead records. See **Converting Leads with SF_BulkOps** in Chapter 4 for more details.

12. **UnDelete** – Use this option to undelete rows from the Recycle bin. You can identify deleted rows using a query against the _QueryAll table:

    Select Id from SALESFORCE…Account_QueryAll
    where IsDeleted='True'

For each row in the input table that the operation fails, the Error column will contain the error message for the failure.

Syntax

```
exec SF_BulkOps 'Insert','linked_server','object','OptionalSoapHdr'
```

Or

```
exec SF_BulkOps 'Delete','linked_server','object','OptionalSoapHdr'
```

Or

```
exec SF_BulkOps 'Update:BulkAPI','linked_server','object','OptionalSoapHdr'
```

or

```
exec SF_BulkOps 'Upsert','linked_server','object','eid', ,'OptionalSoapHdr'
```

where *linked_server* is the name of your linked server , *object* is the name of the object, and *eid* is the name of the external id field.

The *OptionalSoapHdr* parameter is optional and may be used to pass salesforce.com SOAP headers for this execution only. See *Using Optional SOAP Headers* later in this section.

Example

The following example bulk inserts rows from the local table named Account_Load into the Account object at Salesforce.com using the SALESFORCE linked server.

```
exec sf_bulkops 'Insert','SALESFORCE','Account_Load'
```

Controlling the batch size

SF_BulkOps uses a batch size of 200 rows (Web Services API) or 5,000 (Bulk API). You may need to reduce the batch size to accommodate APEX code on the salesforce.com server. To specify a different batch size, use the batchsize(xx) option after the operation.

For example, to set the batch size to 50:

Exec SF_Bulkops 'Update:batchsize(50)','Salesforce','User_Upd'

If you are also using the IgnoreNulls option, then separate the options with a comma:

Exec sf_bulkops 'Update:IgnoreNulls,batchsize(50)','Salesforce','User_Upd'

Controlling the Concurrency Mode

If you are using the bulkapi switch, the default concurrency mode is Serial. To specify parallel concurrency mode instead, use the parallel option:

Exec SF_Bulkops 'Update:bulkapi,parallel,'Salesforce','User_Upd'

Skipping the Status check

If you prefer not to have DBAmp poll for the status (i.e "fire and forget") then add the phrase (ns) after the bulkapi option: **'Insert:bulkapi(ns)_'**

Using Optional SOAP Headers

The salesforce api allow you to pass additional SOAP Headers that alter the behavior of the sf_bulkops operation. The SOAP Headers are described in detail in the salesforce.com api documentation:
http://www.salesforce.com/us/developer/docs/api/Content/soap_headers.htm

The headers are specified in the form of 3 values separated by commas. The first value is the header name, the next value is the section name and the last value is the value for the section. The entire parameter is enclosed in quotes. The salesforce.com api is case sensitive with respect to these values; use the exact token given in the salesforce.com documentation.

For example, to use the default assignment rule for these inserted Leads you would add the following SOAP Header parameter:

```
exec sf_bulkops 'Insert','SALESFORCE','Lead_Test','AssignmentRuleHeader,useDefaultRule,true'
```

The DBAmp Registry settings can also be used to add SOAP headers. The difference is the SOAP header parameter on the sf_bulkops call is a "one-time" use. The DBAmp Registry settings apply the SOAP header to all operations of DBAmp. Therefore, using the SOAP header parameter allows a finer control over the header usage.

Here are some other examples of SOAP headers:

Trigger auto-response rules for leads and cases: 'EmailHeader,triggerAutoResponseEmail,true'

Changes made are not tracked in feeds: 'DisableFeedTrackingHeader,disableFeedTracking,true'

SOAP Headers cannot be used along with the bulkapi switch.

Notes

A full explanation of the SF_BulkOps stored procedure can be found in *Chapter 4: Bulk Insert, Upsert, Delete and Update into Salesforce*.

When individual rows of the input table fail to complete the operation, sf_bulkops writes the error message back to the Error column of that row

and continues processing the next row. Thus, in a batch of 200 rows it is possible that 175 rows were successful and 25 rows failed.

The sf_bulkops stored procedure outputs an error message in the log indicating the sf_bulkops failed when 1 or more rows failed. The correct interpretation of this error message is that at least 1 row of the input table contained an error. In addition, sf_bulkops outputs messages indicating the total number of rows processed the number of rows that failed and the number of rows that succeeded.

For all rows that were successfully processed, sf_bulkops writes the phrase 'Operation Successful" to the Error column. Successfully processed rows can therefore be selected using the following SQL Select:

Select * from Account_Load where Error like '%Operation Successful%'

This technique works for the bulkapi switch as well.

If sf_bulkops is run in a job step, then the job step will fail if one or more rows contain an error. Again, the rows that contain a blank error message were still successful; the failure is thrown to indicate to the operator that at least one row failed.

SF_ColCompare

Usage

SF_ColCompare compares the column structure of a local input table you create to the column structure of a Salesforce.com object. The input table name must begin with a valid Salesforce object name followed by an underscore and suffix. For example, **Account_Load** and **Account_FromWeb** are valid input table names. **XXX_Load** is not a valid input table name (XXX is not a valid Salesforce.com object).

SF_ColCompare requires you to specify an operation of either 'Insert','Update','Upsert', or 'Delete'. The local table is checked to make sure that all columns are valid for that operation.

SF_ColCompare is used to verify that the column names of your input table match the column names of the Salesforce.com object. That confirms that the input table will be successfully used by a later **SF_BulkOps** job.

The output of **SF_ColCompare** is a single result table containing any errors.

One error that **SF_ColCompare** detects is column names in the local table that do not exist in the Salesforce.com object. Column names that appear should be checked for misspellings or other errors. Note: it is possible to have columns in the input table that are intended to be ignored by the **SF_BulkOps** job (for reference or other purposes). These column names will appear as errors even though they are ignored when used with **SF_BulkOps**.

Another error that is detected by **SF_ColCompare** is column names that exist in salesforce.com object but are not applicable to the operation. For example, CreatedDate is a valid column but cannot be inserted or updated and will be flagged by **SF_ColCompare** as an error. Note: if these columns remain in the local table, SF_BulkOps will simply ignore them.

Syntax

```
exec SF_ColCompare 'op','linked_server', 'local_table'
```

where *op* is either 'Insert','Update','Upsert' or 'Delete', *linked_server* is the name of your linked server and *local_table* is the name of the local input table.

Example

The following example compares the local table named Account_Load to the Account object at Salesforce.com using the SALESFORCE linked server for inserting:

```
exec sf_colcompare 'Insert','SALESFORCE','Account_Load'
```

SF_CreateKeys

Usage

SF_CreateKeys creates foreign keys for all local replicated tables of a database. This is useful for creating database diagrams and proving ad-hoc query tools with join hints.

You should run **SF_DropKeys** to ensure that all previous foreign keys are removed before recreating them with **SF_CreateKeys.**

For more information on **SF_CreateKeys**, see the chapter entitled *Creating Database Diagrams and Keys* .

Syntax

```
exec SF_CreateKeys 'linked_server'
```

where *linked_server* is the name of your linked server.

Example

The following example creates foreign keys for all local, replicated tables in the database using the SALESFORCE linked server.

```
exec sf_createkeys 'SALESFORCE'
```

Notes

SF_CreateKeys will only create foreign keys for **existing** local tables; the procedure does not create the local table itself. Therefore, you must replicate down either all the salesforce.com tables (using **SF_ReplicateAll**) or a subset of salesforce.com tables (using **SF_Replicate**) prior to running **SF_CreateKeys.**

SF_DropKeys

Usage

SF_DropKeys drops all foreign keys for all local replicated tables of a database. You should run **SF_DropKeys** to ensure that all previous foreign keys are removed before recreating them with **SF_CreateKeys.**

For more information on **SF_DropKeys**, see the chapter entitled *Creating Database Diagrams and Keys* .

Syntax

```
exec SF_DropKeys 'linked_server'
```

where *linked_server* is the name of your linked server.

Example

The following example drops all foreign keys for all local, replicated tables in the database using the SALESFORCE linked server.

```
exec sf_dropkeys 'SALESFORCE'
```

Notes

- ✓ **SF_DropKeys** should be run before SF_Replicate or SF_Replicate since these procedures assume that no foreign keys exist on the current local tables. We recommend that you only use SF_CreateKeys and SF_DropKeys when you need to database diagram.
- ✓ To create a permanent primary key on the ID field, do not use SF_CreateKeys. Instead, SF_Replicate will automatically create the primary key on the Id field.
- ✓ **SF_DropKeys** will drop the keys on all tables in the salesforce backups database. Do not use SF_DropKeys if you have created your own, non-salesforce tables with keys in the database.

SF_Generate

Usage

SF_Generate generates a empty local table that can be used as input of **SF_BulkOps** for the operation specified. All columns of the salesforce.com obect that are valid for the operation are included in the table. The input table name must begin with a valid Salesforce object name followed by an underscore and suffix. For example, **Account_Load** and **Account_FromWeb** are valid input table names. **XXX_Load** is not a valid input table name (XXX is not a valid Salesforce.com object).

SF_Generate requires you to specify an operation of either 'Insert','Update','Upsert', or 'Delete'. The local table generate will have all columns that are valid for that operation.

The output of **SF_ColCompare** is a single empty table and the Create Table SQL used to create it.

Syntax

```
exec SF_Generate 'op','linked_server', 'local_table'
```

where *op* is either 'Insert','Update','Upsert' or 'Delete', *linked_server* is the name of your linked server and *local_table* is the name of the local input table.

Example

The following example creates the local table named Account_Load for the Account object at Salesforce.com using the SALESFORCE linked server.

```
exec sf_generate 'Insert','SALESFORCE','Account_Load'
```

SF_Refresh

Usage

SF_Refresh compares the current, local replicated table with the contents of the same object at Salesforce.com. Any changes (insert, deletes or updates) are detected and the local table is updated. Use the **SF_Refresh** stored procedure when you need to 'synch' your local copy with Salesforce.com.

SF_Refresh can only be used on objects in salesforce that contain the necessary timestamp columns for tracking changes.

Syntax

exec sf_refresh '*LS*,'*object*','*SchemaError*','*verify*','*bulkapi*'

where *LS* is the name of your linked server and *object* is the name of the object.

The **optional parameter** *SchemaError* should be set to **'Yes'** if you want sf_refresh to automatically call sf_replicate if there is a schema change to the salesforce object.

The **optional parameter** *SchemaError* can also be set to **'Subset'**. If there is a schema change to the salesforce object, sf_refresh will try to determine a valid subset of columns that exist in both the local table and the table on salesforce.com and will refresh the local table based on that column subset. **'Subset'** implies that new fields added to the salesforce object will not be captured by the sf_refresh. In addition, deleted fields will still remain in the local table. To alter the local table and immediately delete columns no longer in the salesforce object, set *SchemaError* to **'SubsetDelete'**. To match the schemas back up, either run sf_replicate or sf_refresh with SchemaError of **'Yes'**.

SchemaError can also be set to `**Repair'**. With the `**Repair'** option, sf_refresh alters the method used for incrementally updating the local table. Specifically, the Max(SystemModstamp) of the local table is used to set the start time of the interval (as opposed to the last time sf_refresh ran). In addition, deleted records are determined by comparing a list of the Id's locally with a list of Id's from the salesforce.com table (as opposed to using the GetDeleted function).

Note: the 'Subset' and 'SubsetDelete' options are not available for SQL 2000.

If *SchemaError* is not provided than sf_refresh prints an error message and throw an error if the two schemas do not match.

The **optional parameter** *verify* can be set to **'no' , 'warn' or 'fail'.** The default value is 'no'. If the *verify* parameter is set to warn or fail, the sf_refresh proc compares the row count of the local table with the row count of the table on salesforce and reports any difference. If the parameter is set to **'fail'** the sf_refresh proc will fail.

The **optional parameter** bulkAPI allows sf_refresh to use the bulkAPI instead of the salesforce web services API. This option should only be used if you are having problems with the sf_refresh. Using the bulk API will always be slower but may be the only way to get the rows down from salesforce.com. **Normally, this option should not be specified.** To use the bulkAPI, set this option to **'bulkapi'** :

```
exec sf_refresh 'SALESFORCE','Account','Yes','no','bulkapi'
```

Example

The following example refreshes the local Account table with the current data on Salesforce.com using the SALESFORCE linked server.

```
exec sf_refresh 'SALESFORCE' , 'Account'
```

Notes

The table must contain a **SystemModstamp** column in order to be refreshed. An initial local copy of the table must exist and be less than 30 days old. If the table does not exist, use the **sf_replicate** procedure to make a local copy before refreshing the table.

SF_RefreshIAD

Usage

SF_RefreshIAD compares the current, local replicated table with the contents of the same object at Salesforce.com. Any inserted or updated rows are detected and the local table is updated. Use the **SF_RefreshIAD** stored procedure when you need to 'synch' your local copy (created with **SF_ReplicateIAD**) with Salesforce.com.

SF_RefreshIAD adds to the local table all deleted rows that are currently in the recycle bin. This is an important difference between **SF_RefreshIAD** and **SF_Refresh**. **SF_RefreshIAD** uses the QueryAll api call.

SF_RefreshIAD can only be used on objects in salesforce that contain the necessary timestamp columns for tracking changes.

Syntax

```
exec SF_RefreshIAD 'linked_server','object_name','SchemaError'
```

where *linked_server* is the name of your linked server and *object_name* is the name of the object.

The optional parameter *SchemaError* should be set to 'Yes' if you want SF_RefreshIAD to automatically call sf_replicateIAD if there is a schema change to the salesforce object.

If *SchemaError* is not provided than SF_RefreshIAD prints an error message and throw an error if the two schemas do not match.

Example

The following example refreshes the local Account table with the current data on Salesforce.com using the SALESFORCE linked server.

```
exec SF_RefreshIAD 'SALESFORCE' , 'Account'
```

Notes

The table must contain a **SystemModstamp** column in order to be refreshed. An initial local copy of the table must exist and be less than 30 days old. If the table does not exist, use the **sf_replicateIAD** procedure to make a local copy before refreshing the table.

SF_RefreshAll

Usage

SF_RefreshAll retrieves a list of the current objects from salesforce and compares the current, local replicated table with the contents of the same object at Salesforce.com. Any changes (insert, deletes or updates) are detected and the local table is updated. Use the **SF_RefreshAll** stored procedure when you need to 'synch' all your local tables with Salesforce.com.

SF_RefreshAll does not refresh all the tables created by **SF_Replicateall** because some of the objects in salesforce cannot be refreshed. These objects do not contain a timestamp field that tracks the datetime of the last modification. In addition, Chatter Feed objects are also skipped by the sf_replicateall/sf_refreshall stored procedures because of the excessive api calls required to download those objects. You can modify the stored procedures to include the Feed objects if needed.

Syntax

```
exec sf_refreshall 'linked_server','SchemaError','verify'
```

where *linked_server* is the name of your linked server.

The optional parameter *SchemaError* should be set to 'Yes' if you want sf_refreshall to automatically call sf_replicate if there is a schema change to the salesforce object. *SchemaError* of 'Yes' will also cause DBAmp to replicate those tables that are not refreshable.

If *SchemaError* is not provided than sf_refreshall prints an error message and throw an error if the two schemas do not match.

The optional parameter *verify* can be set to `'no' , 'warn' or 'fail'.` The default value is 'no'. If the *verify* parameter is set to warn or fail, the sf_refresh proc compares the row count of the local table with the row count of the table on salesforce and reports any difference. If the parameter is set to `'fail'` the sf_refresh proc will fail.

Example

The following example refreshes all the local tables with the current data on Salesforce.com using the SALESFORCE linked server.

```
exec sf_refreshall 'SALESFORCE'
```

Notes

The tables must contain a **SystemModstamp** column in order to be refreshed. An initial local copy of the table must exist and be less than 30 days old. If the tables do not exist, use the **sf_replicateall** procedure to make a local set of tables before refreshing the tables.

Tables that do not contain a **SystemModstamp** column are ignored unless the SchemaError parameter is 'Yes'. These are typically the Salesforce.com tables that end with Status (like CaseStatus) .

The **SF_RefreshAll** stored procedure calls the **SF_Refresh** procedure for each valid local table.

There are some tables, like Vote and UserProfileFeed, in Salesforce that are not included in sf_refreshall. The salesforce.com API does not allow selecting all rows from these tables. In addition, Chatter Feed objects are also skipped by the sf_replicateall/sf_refreshall stored procedures because of the excessive api calls required to download those objects. You can modify the stored procedures to include the Feed objects if needed.

SF_Replicate

Usage

SF_Replicate creates a local replicated table with the contents of the same object at Salesforce.com. The name of the local table is the same name as the Salesforce.com object (i.e. Account). Any schema changes in the object at Salesforce.com are reflected in the new table.

In addition, SF_Replicate creates a primary key on the Id field of the table.

Syntax

```
exec sf_replicate 'linked_server','object_name','options'
```

where *linked_server* is the name of your linked server and *object_name* is the name of the object. There are several optional options you may include as well.

Example

The following example replicates the local Account table with the current data on Salesforce.com using the SALESFORCE linked server.

```
exec sf_replicate 'SALESFORCE' , 'Account'
```

Options

Batchsize: SF_Replicate uses the maximum allowed batch size of 2000 rows. You may need to reduce the batch size to accommodate APEX code on the salesforce.com server. To specify a different batch size, use the batchsize(xx) option after the operation.

For example, to set the batch size to 50:

Exec SF_Replicate 'Salesforce','Account','batchsize(50)'

pkchunk: SF_Replicate uses the salesforce.com web services api by default. If you would like to use the salesforce.com bulkapi with the pkchunking header instead, add the optional pkchunk switch. SF_Replicate will submit a bulkapi job using the pkchunking header and poll every minute for completion. This option should only be used for large tables.

For example, to use the pkchunk and poll every 1 minutes for completion:

Exec SF_Replicate 'Salesforce','Account','pkchunk'

The default batch size will be 100,000. You can alter this using the batchsize parameter:

Exec SF_Replicate 'Salesforce','Account','pkchunk,batchsize(50000)'

Bulkapi: SF_Replicate uses the salesforce.com web services api by default. If you would like to use the salesforce.com bulkapi instead, add the optional bulkapi switch. SF_Replicate will submit a bulkapi job and poll every minute for completion. The bulkapi should only be used for large tables.

For example, to use the bulkapi and poll every 1 minutes for completion:

Exec SF_Replicate 'Salesforce','Account','bulkapi'

Notes

The **SF_Replicate** stored procedure creates a full copy and downloads all the data for that object from Salesforce. If you only want to download the any changes made since you created the local copy, use the **SF_Refresh** stored procedure instead.

A primary index on the Id column will be automatically created when the table itself is replicated.

By default, DBAmp does not download the values of Base64 fields but instead sets the value to NULL. This is done for performance reasons. If you require the actual values, modify the Base64 Fields Maximum Size using the DBAmp Configuration Program to some value other than 0.

SF_ReplicateHistory

Usage

SF_ReplicateHistory is an alternate to **SF_Replicate** that can only be used for History tables. Occasionally, the salesforce server is unable to process a Select * from History table without timing out. SF_ReplicateHistory can sometimes be used to replicate the history table by structuring the SOQL query differently.

Syntax

```
exec sf_replicatehistory 'linked_server','object_name'
```

where *linked_server* is the name of your linked server and *object_name* is the name of a history object.

Example

The following example replicates the local Account table with the current data on Salesforce.com using the SALESFORCE linked server.

```
exec sf_replicatehistory 'SALESFORCE' , 'AccountHistory'
```

SF_ReplicateAll

Usage

SF_ReplicateAll creates a full backup of your Salesforce.com data as local replicated tables with the contents of the same object at Salesforce.com. Any schema changes in the object at Salesforce.com are reflected in the new table.

Salesforce objects that cannot be queried via the salesforce api with no where clause (like ActivityHistory) will NOT be included. In addition, Chatter Feed objects are also skipped by the sf_replicateall/sf_refreshall stored procedures because of the excessive api calls required to download those objects. You can modify the stored procedures to include the Feed objects if needed.

Syntax

```
exec sf_replicateall 'linked_server'
```

where *linked_server* is the name of your linked server.

Example

The following example replicates all the current data on Salesforce.com using the SALESFORCE linked server.

```
exec sf_replicateall 'SALESFORCE'
```

Notes

The **SF_ReplicateAll** stored procedure calls the **SF_Replicate** procedure for each Salesforce.com object.

There are some tables, like Vote and UserProfileFeed, in Salesforce that are not included in **sf_ReplicateAll**. The salesforce.com API does not allow selecting all rows from these tables. In addition, Chatter Feed objects are also skipped by the sf_replicateall/sf_refreshall stored procedures because of the excessive api calls required to download those objects. You can modify the stored procedures to include the Feed objects if needed.

By default, DBAmp does not download the values of Base64 fields but instead sets the value to NULL. This is done for performance reasons. If you require the actual values, modify the Base64 Fields Maximum Size using the DBAmp Configuration Program to some value other than 0.

SF_ReplicateIAD

Usage

SF_ReplicateIAD creates a local replicated table with the contents of the same object at Salesforce.com, including any archived or deleted records from the recycle bin. The name of the local table is the same name as the Salesforce.com object (i.e. Account). Any schema changes in the object at Salesforce.com are reflected in the new table.

Syntax

```
exec sf_replicateIAD 'linked_server','object_name'
```

where *linked_server* is the name of your linked server and *Account* is the name of the object.

Example

The following example replicates the local Account table with the current data on Salesforce.com using the SALESFORCE linked server. Any archived or deleted records will be included in the local table

```
exec sf_replicateIAD 'SALESFORCE' , 'Account'
```

Notes

The **SF_ReplicateIAD** stored procedure creates a full copy and downloads all the data for that object from Salesforce.

Do not try **to SF_Refresh** tables create with **SF_ReplicateIAD**. Instead you can use **SF_RefreshIAD**.

SF_ReplicateIAD only retrieves the deleted records that are currently in the salesforce recycle bin.

SF_ReplicateIAD will retain the permanently deleted rows from run to run. Once you begin to use SF_ReplicateIAD for a table, DO NOT USE sf_replicate on that table. If you run sf_replicate instead of sf_replicateIAD, you will lose all the permanently deleted rows in the local table.

SF_ReplicateLarge

Usage

SF_ReplicateLarge should only be used as a last resort when SF_Replicate fails due to the number of rows in the table.

This stored procedure cannot be used by Professional Edition customers due to the lack of access to the salesforce Bulk API.

SF_ReplicateLarge creates a local replicated table with the contents of the same object at Salesforce.com using the salesforce bulkapi. The name of the local table is the same name as the Salesforce.com object (i.e. Account). Any schema changes in the object at Salesforce.com are reflected in the new table. In addition, **SF_ReplicateLarge** creates a primary key on the Id field of the table.

SF_ReplicateLarge always uses the bulkapi. In addition, it uses a partitioning strategy based on the Id column to pull rows in multiple chunks. Salesforce does not allow binary fields to be queried using the bulkapi. Therefore, **SF_ReplicateLarge cannot be used on tables with binary columns.**

SF_ReplicateLarge requires that the sf user used by the linked server be an salesforce administrator or have the "View All Data" privledge.

SF_ReplicateLarge will almost also take hours to run for tables with many million rows.

Syntax

```
exec sf_replicatelarge 'linked_server','object_name'
```

where *linked_server* is the name of your linked server and *object_name* is the name of the object.

Controlling the batch size

SF_ReplicateLarge uses a batch size of 250,000 .. You may need to reduce the batch size to accommodate Query Timeouts on the salesforce.com server. To specify a different batch size, specify a third parameter with the desired batch size.

For example, to set the batch size to 100000:

```
exec sf_replicatelarge 'SALESFORCE' , 'Account', 100000
```

Example

The following example replicates the local Account table with the current data on Salesforce.com using the SALESFORCE linked server.

```
exec sf_replicatelarge 'SALESFORCE' , 'Account'
```

Chapter 10: DBAmp Registry Settings

DBAmp registry settings are found under Registry Settings menu choice of the DBAmp Configuration Program.

Metadata Override

This entry allows you to modify the Scale of a decimal field or the length of a string field. In some cases, salesforce.com returns data with a greater scale then the reported metadata allows.

For example, in the RevenueForecast table, the scale of the COMMIT column is 0. But salesforce returns data for this column using a scale of 2. To alter DBAmp to use 2 as the scale, set the MetadataOverride field to the following value:

Revenueforecast:Commit(2)

Another example is the Field column in the FieldPermissions table. Use this to make the column larger so that the field names are not truncated:

FieldPermissions:Field(100)

If you need to alter multiple fields, separate the entries with a semicolon.

Base64 Maximum Field Size:

This entry modifies how DBAmp handles large binary fields when downloading from Salesforce (like the Body field of Attachments). If the field has a value greater in length then MaxBase64Size, DBAmp will not attempt to download the binary contents and instead set the value to NULL.

A value of 0 causes DBAmp to set all Base64 fields to NULL. This is the initial setting for performance reasons.

Be sure to restart SQL Server after changing this setting.

Receive Timeout

This entry is the number of seconds DBAmp waits for a response from the salesforce server.

If you are receiving "Operation timed out" error messages, increase this value. For some organizations you may to set this as high as 3000 (i.e. 50 minutes).

BulkAPI Polling Interval

This entry is the number of seconds DBAmp waits between querying for bulkapi job completion.

BulkAPI Status Timeout

This entry is the number of seconds DBAmp waits for a bulkapi job to complete. SF_Replicate with the bulkapi option ignores this value and always uses a tmeout of 12 hours.

Use ConvertCurrency Function

This entry controls whether DBAmp uses the ConvertCurrency function when retrieving currency amounts from salesforce. A checked value forces DBAmp to use the ConvertCurrency function. See chapter 2 of this manual for more details. This setting does not apply to OpenQuery selects.

Be sure to restart SQL Server after changing this setting.

Use ToLabel Function

This entry controls whether DBAmp uses the ToLabel function when retrieving picklists from salesforce. A checked value forces DBAmp to use the ToLabel function. See chapter 2 of this manual for more details. This setting does not apply to OpenQuery selects.

Be sure to restart SQL Server after changing this setting.

TriggerAutoResponseEmail, TriggerOtherEmail, TriggerUserEmail

These entries control whether DBAmp adds an EmailHeader to all requests made to salesforce.com. A checked value forces DBAmp to include the header.

Note: Setting this registry switch forces DBAmp to add the header to all DBAmp operations. If you need finer control then use the optional SOAP header of the SF_BulkOps stored procedure.

These EmailHeaders control the following:

`triggerAutoResponseEmail`	Indicates whether to trigger auto-response rules (`checked`) or not (`not checked`), for leads and cases. In the Salesforce user interface, this email can be automatically triggered by a number of events, for example resetting a user password.

`triggerOtherEmail`	Indicates whether to trigger email outside the organization (`checked`) or not (`not checked`). In the Salesforce user interface, this email can be automatically triggered by creating, editing, or deleting a contact for a case.
`triggerUserEmail`	Indicates whether to trigger email that is sent to users in the organization (`checked`) or not (`not checked`). In the Salesforce user interface, this email can be automatically triggered by a number of events; resetting a password, creating a new user, adding comments to a case, or creating or modifying a task.

UseDefaultAssignment

This entry controls whether DBAmp adds an AssignmentHeader to all requests made to salesforce.com. A checked value forces DBAmp to include the header.

Note: Setting this registry switch forces DBAmp to add the header to all DBAmp operations. If you need finer control then use the optional SOAP header of the SF_BulkOps stored procedure.

Chapter 11: Retrieving Salesforce Metadata

DBAmp can retrieve Salesforce metadata information using the Salesforce metadata api. The **SF_Metadata** stored procedure implements this functionality.

A couple of items to note when using this functionality:

1. Due to the nature of Salesforce metadata information, the metadata is returned to an XML type column in Salesforce. Knowledge of the XML column type and the use of XQuery expressions in SQL Select statements is required to produce results.

2. The **SF_Metadata** stored procedure implements the List and Retrieve functions of the Salesforce Metadata API. These functions require specific type and member inputs as defined in the Salesforce Metadata API Developer's Guide found at http://www.salesforce.com/us/developer/docs/api_meta/index.htm .

   Successfully using **SF_Metadata** is not possible without a review of the Metadata API Guide and an understanding of metadata types.

How to run the SF_Metadata proc

The SF_Metadata stored proc can be executed in a query window or job step.

Note: The SF_Metadata stored procedure uses the xp_cmdshell command. If you are not an SQL Server administrator, you must have the proper permission to use this command. See the SQL Server documentation under the topic xp_cmdshell for more information. To quickly test, run the following sql in Query Analyzer:

Exec master..xp_cmdshell "dir"

To run the **SF_Metadata** stored procedure, use the following command:

Exec SF_Metadata 'List', 'SALESFORCE', 'MD_Input'

Or

Exec SF_Metadata 'Retrieve', 'SALESFORCE', 'MD_Input'

where 'SALESFORCE' is the name you gave your linked server in at installation and MD_Input is the name of the input table to use.

Using the LIST and RETRIEVE operations

The SF_Metadata stored procedure takes as input an operation of either **List** or **Retrieve**.

The **Retrive** operation is used to retrieve xml representations of components in an organization. The input table contains rows that your provide which indicate the components you want to retrieve.

The **List** operation is used when you want a high-level view of particular metadata types in your organization. For example, you could use this operation to return a list of names of all the CustomObject or Layout components in your organization, and use this information to make a subsequent SF_Metadata call with the **Retrieve** operation to return a subset of these components.

Requirements for the input table

Conceptually, **the SF_Metadata** proc takes as input a local SQL Server table you create that is designated as the "input" table. The input table name must not contain embedded blanks. Though not enforced, a naming standard for the input table to **SF_Metadata** should be used. For example, an input table used to retrieve Settings information could be called MD_Settings.

The input table must have the following structure:

```
CREATE TABLE MD_Settings (

    [Name] [nvarchar](255) NULL,

    [Member] [nvarchar](255) NULL,

    [MetadataXML] [xml] NULL,

    [CreatedByWildcard] [bit] NULL,

    [CreatedByList] [bit] NULL,

    [Error] [nvarchar](255) NULL,

    [Id] [nchar](18) NULL

)
```

The purpose of each column is described below:

Name	Type	Description
Name	Nvarchar(255) NULL	The type of metadata component to be retrieved. For example, a value of CustomObject will retrieve one or more custom objects as specified in the member column
Member	Nvarchar(255) NULL	One or more named components, or the wildcard character (*) to retrieve all custom metadata components of the type specified in the <name> element. To retrieve a standard object, specify it by name. For example a value of Account will retrieve the standard Account object.
MetadataXML	xml NULL	The xml describing the component is output to this colum as a result of the Retrieve operation. The xml contents are described by compenent in the salesforce Metadata API documentation.
CreatedByWildcard	bit NULL	Upon input this column should be NULL. If an asterisk was used for the Member column and the operation is Retrieve, then new rows will be created with a value of TRUE for this column. When the SF_Metadata procedure is executed again with operation Retrieve, the rows containing TRUE will be deleted and repopulated again.
CreatedByList	bit NULL	Upon input this column should be NULL. If an asterisk was used for the Member column and the operation is List, then new rows will be created with a value of TRUE for this column. When the SF_Metadata procedure is executed again with operation List, the rows containing TRUE will be deleted and repopulated again.
Error	Nvarchar(255) NULL	Upon input this column should be NULL. The Error column is an output column and is populated with any error messages that are returned from the salesforce server.
Id	nchar(18) NULL	Upon input this column should be NULL. Specifies the ID of the component as returned by the salesforce server.

Example: Retrieve Dependent Picklist Information

This example shows the steps needed to retrieve all dependent Picklist information for the Lead Object.

1. Create an empty input table:

```
CREATE TABLE MD_LeadPicklists (
    [Name] [nvarchar](255) NULL,
    [Member] [nvarchar](255) NULL,
    [MetadataXML] [xml] NULL,
    [CreatedByWildcard] [bit] NULL,
    [CreatedByList] [bit] NULL,
    [Error] [nvarchar](255) NULL,
    [Id] [nchar](18) NULL
)
```

2. Populate the input table. Insert a single row into the table with the Name column of CustomObject and the Member column of Lead

```
INSERT INTO  MD_LeadPicklists  (Name,Member)
Values ( 'CustomObject', 'Lead')
```

3. Run the SF_Metadata proc to retrieve the information.

```
Exec SF_Metadata  'Retrieve', 'Salesforce',  'MD_LeadPicklists'
```

4. Run the following query against the table to generate the results:

```
-- Query to select dependent picklists
;WITH XMLNAMESPACES(DEFAULT 'http://soap.sforce.com/2006/04/metadata')
SELECT Member
    ,fn.c.value('(../../../fullName)[1]','nvarchar(50)') as FieldName
    ,fn.c.value('(../../controllingField)[1]','nvarchar(50)') as
ControllingFieldName
    ,fn.c.value('(../fullName)[1]','nvarchar(50)') as PicklistValue
    ,fn.c.value('(.)[1]','nvarchar(50)') as ControllingPicklistValue
FROM MD_LeadPicklists
cross apply metadataxml.nodes
('/CustomObject/fields/picklist/picklistValues/controllingFieldValues') as fn(c)
```

5. Result:

Member	FieldName	ControllingFieldNa	PicklistValue	ControllingPicklistValue
Lead	ProductInterest__c	Industry	GC1000 series	Agriculture
Lead	ProductInterest__c	Industry	GC1000 series	Apparel
Lead	ProductInterest__c	Industry	GC1000 series	Banking
Lead	ProductInterest__c	Industry	GC1000 series	Biotechnology
Lead	ProductInterest__c	Industry	GC1000 series	Construction
Lead	ProductInterest__c	Industry	GC1000 series	Education
Lead	ProductInterest__c	Industry	GC5000 series	Biotechnology
Lead	ProductInterest__c	Industry	GC5000 series	Chemicals
Lead	ProductInterest__c	Industry	GC5000 series	Construction
Lead	ProductInterest__c	Industry	GC5000 series	Electronics

Example: Retrieve Field Descriptions

This example shows how to retrieve field description information using the salesforce metadata api.

```
Drop Table MD_FieldDesc
go

CREATE TABLE MD_FieldDesc (
[Name] [nvarchar](255) NULL,
[Member] [nvarchar](255) NULL,
[MetadataXML] [xml] NULL,
[CreatedByWildcard] [bit] NULL,
[CreatedByList] [bit] NULL,
[Error] [nvarchar](255) NULL,
[Id] [nchar](18) NULL
)

INSERT INTO MD_FieldDesc (Name,Member) Values ( 'CustomObject', '*')
-- Get a list of objects with customer fields
Exec SF_Metadata 'List', 'Salesforce', 'MD_FieldDesc'

-- Cleanup wildcard and objects that will error
Delete MD_FieldDesc where Member = '*'
Delete MD_FieldDesc where Member = 'SiteChangeList'

-- Retrieve the field metadata
Exec SF_Metadata 'Retrieve', 'Salesforce', 'MD_FieldDesc'

-- Query to select descriptions
;WITH XMLNAMESPACES(DEFAULT
'http://soap.sforce.com/2006/04/metadata') SELECT Member
,fn.c.value('(fullName)[1]','nvarchar(50)') as FieldName
,fn.c.value('(description)[1]','nvarchar(50)') as Description
--,fn.c.value('(../fullName)[1]','nvarchar(50)') as PicklistValue
,fn.c.value('(.)[1]','nvarchar(50)') as ControllingPicklistValue
FROM MD_FieldDesc
```

```
cross apply metadataxml.nodes ('/CustomObject/fields') as fn(c)
```

Chapter 12: Using DBAmp Performance Package

The DBAmp Performance package allows you to capture the message output from the DBAmp stored procedures and summarize the information into performance metrics.

There are many reasons to use the DBAmp Performance Package (DPP):

- DPP creates a DBAmp_Log table to log all message output from stored procedure execution, this allows you to locate message output errors
- DPP creates views to summarize SF_Replicate, SF_Refresh, and SF_Bulkops
- DPP allows you to view run times, number of rows copied, deleted, updated, inserted, etc.
- DPP allows you to easily view which tables failed
- DPP allows you to connect to an outside analytics tool to visualize performance (ex: Excel)

Using the DBAmp Performance package you can answer questions like:

- How long does on average does it take to replicate or refresh a table ?
- What is the average throughput of sf_ bulkops ?
- What is the failure rate of the DBAmp stored procedures?

The DBAmp Performance Package contains two components:

1. The DBAmp_Log table that contains the message output from all stored procedure execution
2. Performance Views that summarize the DBAmp_Log table into a set of usable performance metrics.

Installing the DBAmp Performance Package

The first step to install the DBAmp Performance Package is to run a script to create the needed objects.

If you are currently using DBAmp_Log table, installing the DBAmp Performance package will delete all data in your current DBAmp_Log.

DBAmp_Log can hold up to 250,000 rows, which is approximately 50 MB of data storage. Once DBAmp_Log reaches 250,000 rows, it deletes ¼ of itself.

To install the DBAmp Performance Package:

1. Open the file "Create DBAmp Perf.sql" in Query Analyzer or Management Studio but do not execute it yet. The file is located in the \Program Files\DBAmp\SQL directory.

2. Make sure the default database shown on the toolbar is the salesforce backups database (and not the main database). Then, execute (F5) to add the script to the database.

3. In order to make sure that the Create DBAmp Performance script worked properly, perform two actions:

Verifying

Run the statement below to verify that the **DBAmp_Log table** was created:

Select * from DBAmp_Log

You should see a table similar to the screenshot below:

- Under **Views**, in the **salesforce backups** database under **Object Explorer**, check to see that the **three performance views** were created. It should look similar to the screenshot provided below:

If these are **working properly**, you are ready to begin using the DBAmp Performance Package.

Using the DBAmp_Log Table

All DBAmp stored procedures write their output message to the DBAmp_Log table created by the DBAmp Performance Package. By querying the **DBAmp_Log table**, you can view the message output from recently executed DBAmp stored procedures. This allows you to view information and find any errors related to each DBAmp stored procedure execution. The columns in the table are:

Results | Messages

	SPName	Status	Message	LogTime
1	SF_Replicate:42842B81-A6EA-4DBE-A1EA-6C9782A0DC9B	Starting	Parameters: SALESFORCE Account	2015-05-26 17:00:00.373
2	SF_Replicate:42842B81-A6EA-4DBE-A1EA-6C9782A0DC9B	Message	Drop Account_Previous if it exists.	2015-05-26 17:00:00.383
3	SF_Replicate:42842B81-A6EA-4DBE-A1EA-6C9782A0DC9B	Message	Create Account_Previous with new structure.	2015-05-26 17:00:00.383
4	SF_Replicate:42842B81-A6EA-4DBE-A1EA-6C9782A0DC9B	Message	Run the DBAmp.exe program.	2015-05-26 17:00:00.730
5	SF_Replicate:42842B81-A6EA-4DBE-A1EA-6C9782A0DC9B	Message	17:00:00: DBAmp Bulk Operations. V2.20.4 (c) Copy...	2015-05-26 17:00:02.727
6	SF_Replicate:42842B81-A6EA-4DBE-A1EA-6C9782A0DC9B	Message	17:00:00: Populating local table Account_Previous , ...	2015-05-26 17:00:02.730
7	SF_Replicate:42842B81-A6EA-4DBE-A1EA-6C9782A0DC9B	Message	17:00:00: DBAmp is using the SQL Native Client.	2015-05-26 17:00:02.730
8	SF_Replicate:42842B81-A6EA-4DBE-A1EA-6C9782A0DC9B	Message	17:00:00: Opening SQL Server rowset	2015-05-26 17:00:02.730
9	SF_Replicate:42842B81-A6EA-4DBE-A1EA-6C9782A0DC9B	Message	17:00:02: 3361 rows copied.	2015-05-26 17:00:02.730
10	SF_Replicate:42842B81-A6EA-4DBE-A1EA-6C9782A0DC9B	Message	Drop Account if it exists.	2015-05-26 17:00:02.733
11	SF_Replicate:42842B81-A6EA-4DBE-A1EA-6C9782A0DC9B	Message	Rename previous table from Account_Previous to A...	2015-05-26 17:00:02.737
12	SF_Replicate:42842B81-A6EA-4DBE-A1EA-6C9782A0DC9B	Message	Create primary key on Account	2015-05-26 17:00:02.737
13	SF_Replicate:42842B81-A6EA-4DBE-A1EA-6C9782A0DC9B	Successful	Ending - Operation Successful.	2015-05-26 17:00:02.753
14	SF_ReplicateAll:9FD0C1E0-6DA4-45E2-B60E-FD7E1DA705	Starting	Parameters: SALESFORCE	2015-05-26 17:46:11.760
15	SF_Replicate:39212D54-0F1B-4A19-846C-4DB2F8E48378	Starting	Parameters: SALESFORCE AcceptedEventRelation	2015-05-26 17:46:18.773
16	SF_Replicate:39212D54-0F1B-4A19-846C-4DB2F8E48378	Message	Drop AcceptedEventRelation_Previous if it exists.	2015-05-26 17:46:18.777
17	SF_Replicate:39212D54-0F1B-4A19-846C-4DB2F8E48378	Message	Create AcceptedEventRelation_Previous with new s...	2015-05-26 17:46:18.777
18	SF_Replicate:39212D54-0F1B-4A19-846C-4DB2F8E48378	Message	Run the DBAmp.exe program.	2015-05-26 17:46:18.853
19	SF_Replicate:39212D54-0F1B-4A19-846C-4DB2F8E48378	Message	17:46:18: DBAmp Bulk Operations. V2.20.4 (c) Copy...	2015-05-26 17:46:19.310

Column Name	Documentation
SPName	Unique ID of each execution
Status	Status of the execution
Message	All messages related to each execution
LogTime	The date and time the execution started (status = starting) and ended (status = successful/failed)

Run the statement below to select all rows and columns of the DBAmp_Log Table:

Select * from DBAmp_Log

Using the Performance Views

Why Views?

- The views summarize the raw message output in the DBAmp_Log table into views that can be analyzed for performance.

- The views can be used to import performance data into Excel or other analytical tools

There are three performance views included in the DBAmp Performance Package. The three views and their documentation are listed below:

DBAmp_Replicate_Perf view

The DBAmp_Replicate_Perf view contains the data and metrics of all SF_Replicate, SF_ReplicateIAD, and SF_Replicate3 executed. The columns in the view are:

Results | Messages

	SPName	LogTime	LinkedServer	Object	RowsCopied	RunTimeSeconds	Failed
1	SF_Replicate:028D7A6A-B270-4A3B-BAEA-BF025BA8F9BC	2015-05-26 17:48:13.667	SALESFORCE	OpportunityStage	10	1	False
2	SF_Replicate:03F8407F-6B4D-4244-8539-185B51A4A1D2	2015-05-26 17:48:37.193	SALESFORCE	ThirdPartyAccountLink	0	1	False
3	SF_Replicate:04F533C1-0180-4260-AC84-EE8C920201D3	2015-05-26 17:46:47.533	SALESFORCE	Campaign	4	1	False
4	SF_Replicate:055438FC-1273-49B6-AFE9-7DD830259A72	2015-05-26 17:47:28.043	SALESFORCE	CustomPermissionDependency	0	1	False
5	SF_Replicate:061D20F0-EA1F-4C20-91B5-B51F75D3B4C0	2015-05-26 17:48:05.733	SALESFORCE	MacroHistory	0	1	False
6	SF_Replicate:07A70340-5D41-47F8-A5B7-E4D30588B4AC	2015-05-26 17:47:39.797	SALESFORCE	ExternalDataSource	0	1	False
7	SF_Replicate:08096C9C-2A73-43D3-BFAE-7406248FB00B	2015-05-26 17:46:31.607	SALESFORCE	AccountShare	0	1	False
8	SF_Replicate:08375692-0A7E-4436-BB48-E7B0ED201FB7	2015-05-26 17:47:08.970	SALESFORCE	CollaborationGroup	1	1	False
9	SF_Replicate:0A8A79E5-B5A3-4B29-9A79-22947A846A76	2015-05-26 17:47:11.377	SALESFORCE	Contact	28	1	False
10	SF_Replicate:0AF41E86-B8E7-420E-A4B7-BBF85BFBEC7F	2015-05-26 17:47:12.453	SALESFORCE	ContactHistory	0	1	False
11	SF_Replicate:0C5081A8-D6F4-44BC-AFF9-3F364FD8F7D1	2015-05-26 17:47:49.190	SALESFORCE	GrantedByLicense	0	1	False
12	SF_Replicate:0E24E691-91BD-4F25-9771-21BE822EDC0E	2015-05-26 17:48:14.180	SALESFORCE	Order	0	1	False
13	SF_Replicate:0E680709-429D-4E64-A384-137F5D4B9278	2015-05-26 17:46:41.640	SALESFORCE	Attachment	1	1	False
14	SF_Replicate:0E81AF80-8F49-4489-BC0F-E2E599AEF961	2015-05-26 17:47:32.173	SALESFORCE	DcSocialProfileHandle	0	1	True
15	SF_Replicate:0F2C25D7-40B8-405C-8EC1-07FF1E0C7AF6	2015-05-26 17:47:59.837	SALESFORCE	LoginHistory	740	5	False
16	SF_Replicate:0FF96702-F1A8-4860-9751-864A6FE14022	2015-05-26 17:46:30.370	SALESFORCE	AccountContactRole	0	1	False
17	SF_Replicate:14B7FDE3-03F9-4419-A1DC-B362FBAF6720	2015-05-26 17:47:57.417	SALESFORCE	LeadHistory	0	1	False
18	SF_Replicate:150FFCCF-84F5-4D18-83A2-B3837A16084C	2015-05-26 17:47:37.650	SALESFORCE	EntityDefinition	386	1	False
19	SF_Replicate:15B2DB45-4B70-4C3B-9217-745D6FB0C280	2015-05-26 17:48:28.097	SALESFORCE	RecentlyViewed	50	1	False

Column Name	Documentation
SPName	Unique ID of each execution
LogTime	The date and time the execution started (status = starting) and ended (status = successful/failed)
LinkedServer	Name of the DBAmp linked server used

Object	Name of object
RowsCopied	Number of rows copied during each execution
RunTimeSeconds	Number of seconds the execution took to run
Failed	If the execution failed or not (True = failed)

Run the statement below to select all rows and columns of the DBAmp_Replicate_Perf:

Select * from DBAmp_Replicate_Perf

DBAmp_Refresh_Perf view

The DBAmp_Refresh_Perf view contains the data and metrics of all SF_Refresh and SF_RefreshIAD executed. The columns in the view are:

Results | Messages

	SPName	LogTime	LinkedServer	Object	RowsUpdatedOrInserted	RowsDeleted	RunTimeSeconds	Failed
1	SF_Refresh:93E061B9-84EC-4D03-B39C-8DD255B39067	2015-05-26 19:24:10.680	SALESFORCE	AcceptedEventRelation	0	0	1	False
2	SF_Refresh:895FB393-6DE5-45E3-9DA9-0A1E4280D561	2015-05-26 19:24:10.990	SALESFORCE	Account	0	0	1	False
3	SF_Refresh:E6509F15-08D1-4E5C-B7DA-9AD15ABAFEF2	2015-05-26 19:24:11.297	SALESFORCE	AccountCleanInfo	0	0	1	False
4	SF_Refresh:2B61BA16-4BC3-497B-BDA6-53CEFF59FB2D	2015-05-26 19:24:11.643	SALESFORCE	AccountContactRole	0	0	1	False
5	SF_Refresh:D02CA223-65CA-4454-8778-8C334162E21F	2015-05-26 19:24:11.817	SALESFORCE	AccountHistory	0	0	1	False
6	SF_Refresh:37B5D25B-E005-4837-B531-6AF9D5B77E6B	2015-05-26 19:24:12.080	SALESFORCE	AccountPartner	0	0	1	False
7	SF_Refresh:AB2CB37F-5EB5-4F2B-8DAF-959799D08BC4	2015-05-26 19:24:12.250	SALESFORCE	ActionLinkGroupTemplate	0	0	1	False
8	SF_Refresh:2DF72790-8FE3-45FC-AD8E-DBB543CDDFBE	2015-05-26 19:24:12.427	SALESFORCE	ActionLinkTemplate	0	0	1	False
9	SF_Refresh:FDA21ADE-1BA8-4403-99BD-52C7EEC6012C	2015-05-26 19:24:12.643	SALESFORCE	AdditionalNumber	0	0	1	False
10	SF_Refresh:A9C9D5C8-A32D-4F73-A18A-37958D5FE451	2015-05-26 19:24:12.850	SALESFORCE	Announcement	0	0	1	False
11	SF_Refresh:B252C601-0F83-4B6F-BBDB-FBF7C116365E	2015-05-26 19:24:13.083	SALESFORCE	ApexClass	0	0	1	False
12	SF_Refresh:65E6E9C9-CD59-469A-9FEA-BEA9DB577D79	2015-05-26 19:24:13.447	SALESFORCE	ApexComponent	0	0	1	False
13	SF_Refresh:B77E84DD-FDA5-4A60-B0DE-204884480190	2015-05-26 19:24:13.677	SALESFORCE	ApexLog	0	0	1	False
14	SF_Refresh:FCB7BCB9-52FE-41AC-B937-732C6A3FD44C	2015-05-26 19:24:13.840	SALESFORCE	ApexPage	0	0	1	False
15	SF_Refresh:FB6B266E-4C84-4407-BE0F-843F03CCD335	2015-05-26 19:24:14.037	SALESFORCE	ApexTestQueueItem	0	0	2	False
16	SF_Refresh:ED2C8EA8-5B9D-4503-96E4-A739852437A7	2015-05-26 19:24:16.210	SALESFORCE	ApexTestResult	0	0	1	False
17	SF_Refresh:970AB25B-385F-49D0-8EAF-157042698C12	2015-05-26 19:24:17.207	SALESFORCE	ApexTrigger	0	0	1	False
18	SF_Refresh:CE6A5FA6-95E4-44E3-BC4E-ECC9E92F905C	2015-05-26 19:24:17.520	SALESFORCE	AppMenuItem	0	0	1	False
19	SF_Refresh:46A25A3E-F057-47ED-892C-CC529472D04A	2015-05-26 19:24:18.233	SALESFORCE	Asset	0	0	1	False

Column Name	Documentation
SPName	Unique ID of each execution
LogTime	The date and time the execution started (status = starting) and ended (status = successful/failed)
LinkedServer	Name of the DBAmp linked server used
Object	Name of object
RowsUpdatedOrInserted	Number of rows updated/inserted

RowsDeleted	Number of rows deleted
RunTimeSeconds	Number of seconds the execution took to run
Failed	If the execution failed or not (True = failed)

Run the statement below to select all rows and columns of the DBAmp_Refresh_Perf:

Select * from DBAmp_Refresh_Perf

DBAmp_BulkOps_Perf view

The DBAmp_BulkOps_Perf view contains the data and metrics of all SF_BulkOps executed. The columns in the view are:

Results | Messages

	SPName	LogTime	BulkOpsAction	LinkedServer	LoadTable	RowsRead	RowsSuccessfull	RowsFailed	RunTimeSeconds	Failed
1	SF_BulkOps:0542246E-1693-48D9-A474-8D7627A7D327	2015-05-26 19:33:43.400	Update	SALESFORCE	Account_Update	40	20	20	1	True
2	SF_BulkOps:CD6CAB8D-45A9-4B02-9E72-41A8E7E02856	2015-05-26 19:34:42.903	Update	SALESFORCE	Account_Insert	0	0	0	1	False
3	SF_BulkOps:4E6CF239-C951-4894-BD39-CB761FD268C0	2015-05-26 19:34:51.813	Insert	SALESFORCE	Account_Insert	0	0	0	1	False
4	SF_BulkOps:08727AB7-94D7-460A-80AD-17FA7BC34E4F	2015-05-26 19:35:46.340	Update	SALESFORCE	Account_Update2	2	2	0	1	False
5	SF_BulkOps:264D2096-B28C-491B-9F97-0DE85FF59141	2015-05-26 19:35:53.600	Update	SALESFORCE	Account_Update3	2	2	0	1	False

Column Name	Documentation
SPName	Unique ID of each execution
LogTime	The date and time the execution started (status = starting) and ended (status = successful/failed)
BulkOpsAction	The execution action (update, insert, upsert, delete, etc.)
LinkedServer	Name of the DBAmp linked server used
LoadTable	Name of the local SQL input table used containing the data
RowsRead	Total number of rows read during each execution
RowsSuccessful	Number of rows successfully read
RowsFailed	Number of rows that failed

RunTimeSeconds	Number of seconds the execution took to run
Failed	If the execution failed or not (True = failed)

Run the statement below to select all rows and columns of the DBAmp_BulkOps_Perf:

Select * from DBAmp_BulkOps_Perf

Using Excel with the Performance Views

To import data from the SQL Performance views into Excel spreadsheets and pivot tables, visit the **Using Excel** section in **chapter 8: Using Excel with Views to Linked Server Tables**. Below is an example of how you could use the performance view data for analysis:

The screenshot below is of performance data from the DBAmp_Replicate_Perf view fed into Excel for analysis purposes:

1	LogTime	LinkedServer	Object	RowsCopied	RunTimeSeconds	Failed
2	4/23/2015 10:02	SALESFORCE	Account	3341	17	False
3	4/30/2015 14:06	SALESFORCE	Account	3341	16	False
4	4/23/2015 9:55	SALESFORCE	FieldPermissions	5263	11	False
5	4/23/2015 9:55	SALESFORCE	LoginHistory	545	7	False
6	4/30/2015 13:59	SALESFORCE	LoginHistory	514	6	False
7	4/30/2015 13:58	SALESFORCE	Account	3341	4	False
8	4/23/2015 9:54	SALESFORCE	Account	3341	4	False
9	4/23/2015 9:54	SALESFORCE	ApexClass	20	3	False
10	4/23/2015 9:53	SALESFORCE	AT_Workspace__History	0	3	False
11	4/30/2015 13:59	SALESFORCE	FieldPermissions	5263	2	False
12	4/30/2015 14:00	SALESFORCE	FieldPermissions	5263	2	False
13	4/30/2015 13:59	SALESFORCE	Account	3341	2	False
14	4/23/2015 9:54	SALESFORCE	Contact	2305	2	False
15	4/30/2015 13:59	SALESFORCE	Contact	2305	2	False
16	4/30/2015 13:59	SALESFORCE	Contact	2305	2	False
17	4/30/2015 14:00	SALESFORCE	RecentlyViewed	219	2	False

The graph below is of the performance data shown above. In this example, it is showing you the time it took for the stored procedures to execute vs. the number of rows that were copied for each execution. Below is an example of a performance measure that you can see by using the DBAmp Performance Package: