Contents

SQL-Programmer Batch Objects

Batch Object Concepts

The concept of a *batch* has been around since the days of the first mainframe computers. Batch files and batch jobs have made their way through all the innovations and improvements and much like a keyboard have survived with the same basic principles.

A batch is generally a collection of commands which an operating system or environment will play or execute for you. The commands within a batch job are usually required to be executed more than once; and therefore they are saved for repeated use. This is a real convenience for the operator, or user who does not have to re-key in the commands, wait for the first task to end to key in another command and so on.

Imagine we had a computer, and every time we started the computer up, we wanted a set of commands to run automatically. To accomplish this we could create a file, and in this file we would enter each command which would run in the desired sequence. Assuming the operating system or environment had a method of indicating to it the name of the file to execute on start up, we could customize the environment to the way we wanted it to work for us. This "*Batch Object*" would save us time and ensure consistency every time we needed to use our computer.

In the world of SQL Servers, "**Batch Objects**" are extremely useful. They could be used to backup our system every day. They could be used to update information in our database coming in from external sources. They could even be used to reconstruct an entire database. Further more in development, we often need to reset our database back to a virgin state. This may mean truncating tables; re-initializing data; re-indexing; and dropping temporary work tables each time. Smart programmers develop "**Batch Objects**" to accomplish this.

SQL Server Batch Transactions

Microsoft SQL Server and Sybase SQL Server both support Batch Concepts. Most people familiar with these products are also familiar with a utility called ISQL. This utility permits SQL Server statements to be grouped together in batches. The server would process the batch of commands and return results. A batch of SQL Server statements is terminated with a "GO" command which instructs the server to go ahead and execute the preceding commands. Multiple batches are often kept in operating system files on the developer's workstation and on the file server. The statements before each "GO" are considered to be within the same transaction.

The Batch Language of Microsoft SQL Server and Sybase SQL Server supports many SQL extensions; including (IF ELSE, BEGIN, END, WHILE, BREAK, CONTINUE, PRINT, GOTO, RETURN and DECLARE). This is indicative of the recognition of batch processing as an important element of SQL Server management.

Typically SQL Server developers and DBAs have several of these script files which they've developed to perform a variety of tasks. Some trivial and others mission critical to the entire organization they represent. In an attempt to appeal to one's sense of responsibility, we ponder, Where are these Files? On your hard drive?, Are they safe?, Are they documented? Will I be able to come back to them in six months and understand their inherent organization?

Even though most people in these positions of responsibility, truly do care, and although they truly don't like having to manage these crucial objects in such a rudimentary fashion ... What's the alternative?

SQL-Programmer Batch Objects

SQL-Programmer has always upheld "Batch Object" concepts. Users enter SQL Server Batch Transactions directly into a FreeSQL! window, execute them and view their results. SQL-Programmer supports importing ISQL compatible scripts into a FreeSQL! window, as well as execution directly from an operating system file.

SQL-Programmer also supports the generation of scripts in the form of SQL Server Batch Transactions, providing the ability to regenerate all or portions of all the objects within a single or multi-server environment.

Although these features are extremely useful, and augment the productivity of any SQL Server developer, or DBA, they still rely on operating system files for the management and organization of the often mission critical batch files.

In response to this significant problem SQL-Programmer has introduced a new entity to the Microsoft SQL Server, and Sybase SQL Server database environments which allows these transaction files to be stored as programmable objects within the database. **This new entity is appropriately named "Batch Object".**

This invention provides developers and database administrators with an integrated method of safely housing SQL Server batch transaction scripts within the protected server environment. Furthermore the organization of the scripts can take on the inherent organization of the databases on the server.

Now SQL Server developers and DBAs can create, edit, execute, drop, backup and restore their SQL Server Batch Transactions in precisely the same way as they manage all their other programmable objects, which in many cases are of no more importance.

Why use Batch Objects?

To deal with the question **"Why use Batch Objects?"** we should first examine some possible uses for SQL Server Batch Transaction scripts.

⇒ A script that truncates some tables.

⇒ A script that re-indexes tables (Drops and Creates indexes).

⇒ A script that performs complex data manipulation to roll up tables from detailed transactions.

⇒ A script that updates the statistics on all your tables pertinent indexes.

⇒ A script that contains all the appropriate grant and revoke commands for all the objects in your database.

⇒ A script that dumps all your databases to your dump device.

⇒ A script you use as a template to create data and log devices, databases, and sets database options.

⇒ A script which runs Database Consistency Checker (DBCC) against a set of specific tables to monitor them closely.

⇒ A script that initializes table values.

⇒ A script that recreates stored procedures in their proper order of reference.

⇒ A script that recreates a set of triggers.

Most Microsoft SQL Server and Sybase SQL Server developers have scripts like these related to several databases, often spread between their local workstation, and some tucked away area on the file server. Up until the emergence of SQL-Programmer Batch Objects this inferior means has been the only alternative.

Rather than storing these scripts outside the server environment, it makes much more sense to store them within the database they are operating on. Storing them on the database provides inherent organization, accessibility and security. Furthermore since SQL-Programmer provides a state of the art environment for developing, maintaining and executing these crucial SQL Server programs, there will be a time and cost savings to the organization due to the intrinsic productivity gains acquired by using SQL-Programmer.

SQL-Programmer expands the capability of the traditional SQL Server transaction batch, by allowing Batch Objects to be executed from within other Batch Objects. This allows developers to modularize their scripting. I.E. a script to drop and create a trigger, which is executed from a script to drop and create a table which is executed from a script which contains an execute batch object statement for each table.

Utilizing SQL-Programmer Batch Objects not only enhances the functionality, it provides you with an integrated means of documenting all your database programmable objects through the extensive SQL-Programmer reporting facility.

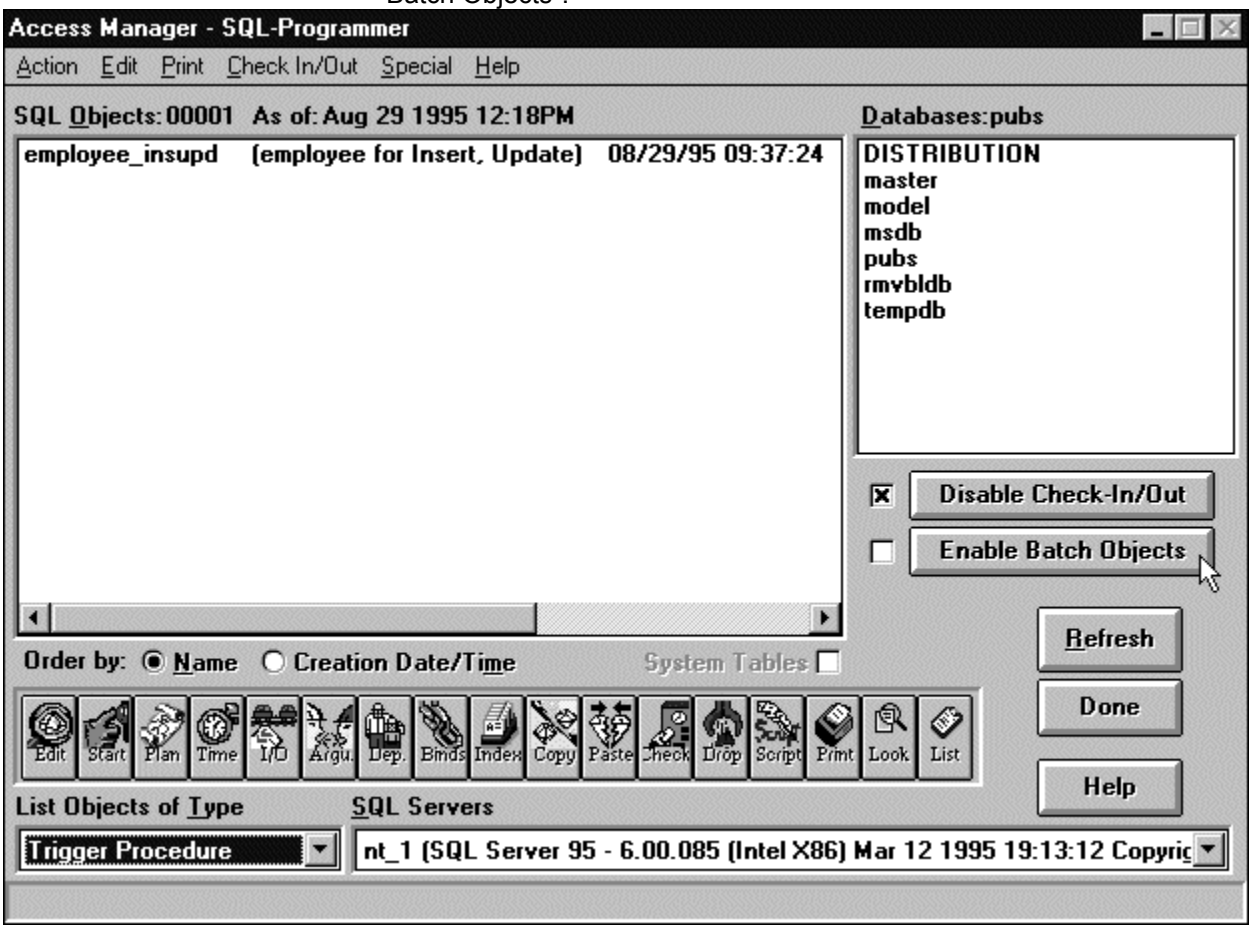
SQL-Programmer's Check-In / Check-out facility also applies itself to Batch Objects. This means in multi-developer environments your scripts are protected from inadvertent loss, and have a means of recording a change history.

In summary you gain organization, safety, time, and considerable functionality by using SQL-Programmer based Batch Objects.

How to use Batch Objects

Enabling the Use of Batch Objects

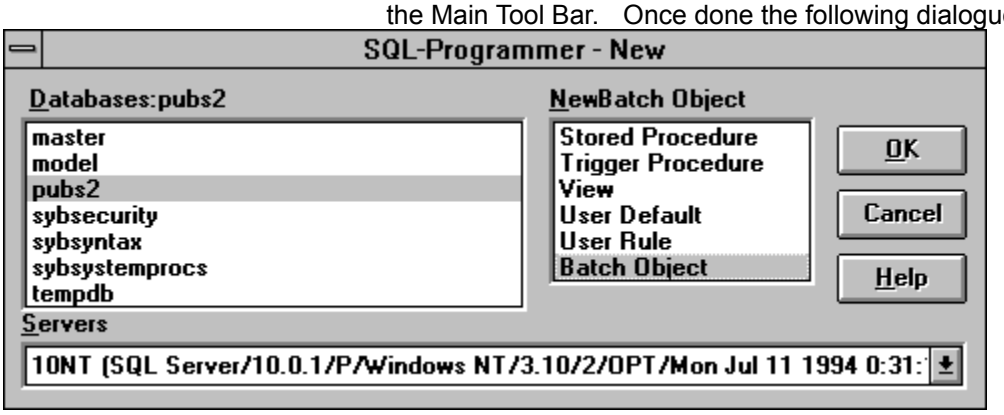
Before any SQL-Programmer user can take advantage of "Batch Objects", the SA must enable the option. This is achieved via the SQL-Programmer Access Manager. The screen shot below illustrates a command button on the Access Manager dialogue which is used to both enable and disable Batch Objects. When disabled the text on the button reads "Enable Batch Objects" and when enabled the text reads "Disable Batch Objects".



Enabling batch objects causes SQL-Programmer to create a table called SQL_Programmer_Objects and a trigger called SFI_objects_triggger_iud to be created.

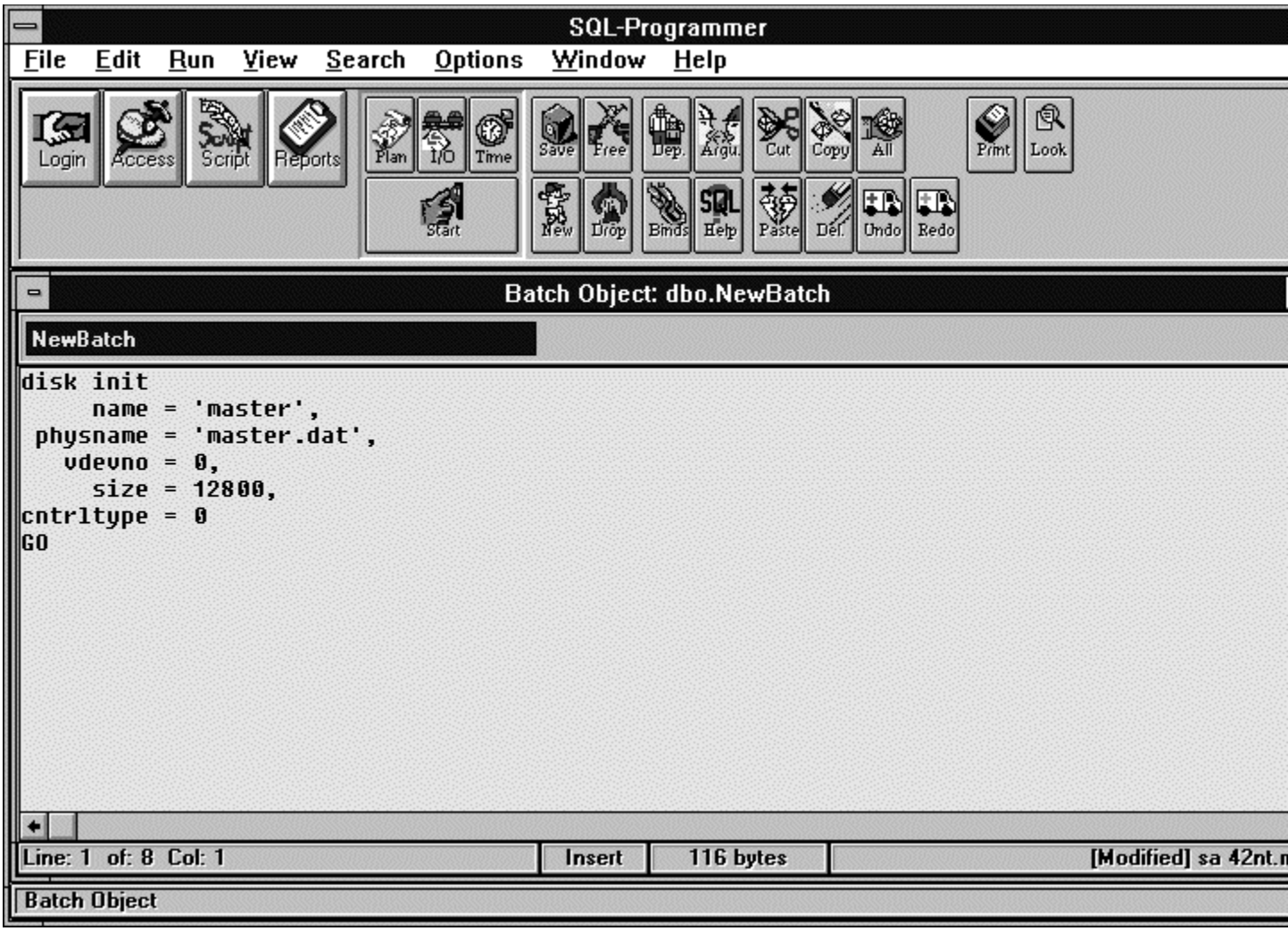
Creating New Batch Objects

Creating a new Batch Object is performed from the SQL-Programmer Workplace. To create a new Batch Object click on the New button on the Main Tool Bar. Once done the following dialogue is presented.



In this dialogue select the SQL Server from the drop down list box at the bottom of the dialogue, the database you wish to create the Batch Object in, and Batch Object from the list of Objects. You will then be presented with a SQL-Programmer Edit window which is virtually the same as a FreeSQL! Window.

Please notice the single line edit box containing the default name "NewBatch". The intention here is that this name be changed to one of appropriate meaning. All that's left to do is to code or import and save the Batch Object. The example below shows a Batch Object used to create a new device. The Batch Object has been named *InitProd1* by changing the name in the single line edit box.



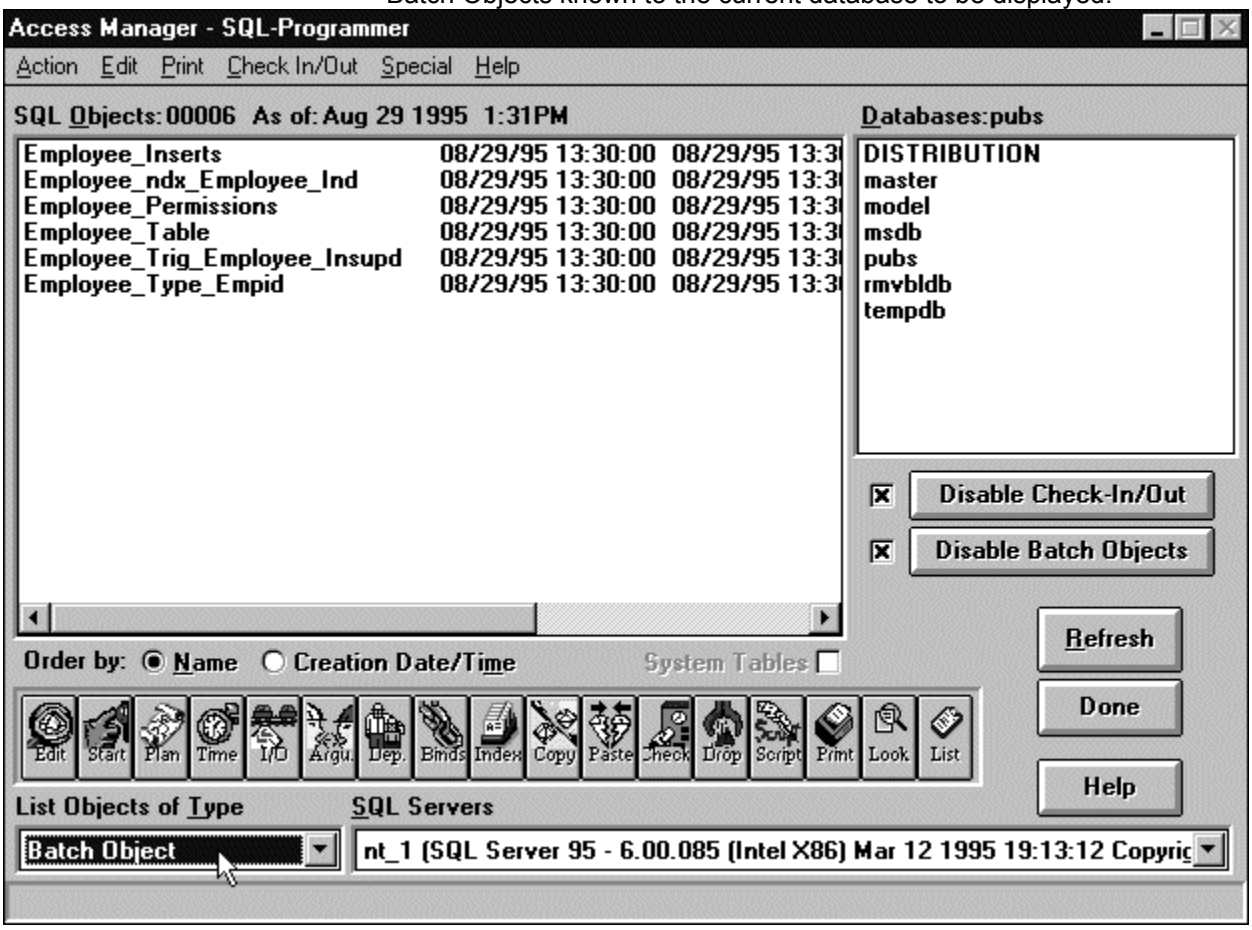
To save your new Batch Object click the Save Button on the Main tool bar. You will then be informed that the Batch Object has been saved as shown below. Note that the Batch Object created will be created with the user's name as the owner. This could impact the execution of the Batch Object.

If a Batch Object already exists in the selected database you will be informed and given the choice to replace the existing object. If you choose to replace the existing object you will be informed that the object has been replaced. Otherwise you will be returned to the Edit window where may change the name of the Batch Object and click the Save button to save the Batch Object under the new name.

You may also create Batch Objects by opening a FreeSQL! window and entering the Batch Object creation syntax **or by simply recording your SQL transaction statements and clicking on the save button**.

Using Batch Objects via SQL-Programmer Access Manager

Once Batch Objects have been enabled, they are accessible from the SQL-Programmer Access Manager dialogue. The "List Objects of Type" drop down list box at the lower left hand corner of the dialogue contains the "Batch Object" option. Selecting this will result in a list of all the Batch Objects known to the current database to be displayed.



To perform any operation on an existing batch object, simply click the batch object(s) of choice in the Access Manager SQL Objects list and use the Access Manager tool bar to invoke the desired operation.

To edit existing Batch Objects either double click on the desired batch object or select it by single clicking it and click on edit in the Access Manager Tool Bar. The selected Batch Object contents will be returned to a SQL-Programmer Batch Objects edit window as described for creating Batch Objects above.

Batch Object Syntax

A special syntax has been added to accommodate Batch Objects which are created in ISQL fashion. The following outlines the Batch Object syntax which should be followed for creating, dropping and executing Batch Objects in batches. The [...] designate optional entries.

Pay particular attention to the "*endbatch*" keyword which should accompany each create command. This is used to enable SQL-Programmer to distinguish between the traditional "GO" command which will often be found within a batch and the end of the *Create Batch* transaction. The SQL-Programmer Script Generator will make use of this "*endbatch*" keyword when scripting Batch Objects.

Related Topics:

Create Batch

Execute Batch

Drop Batch

Create Batch

Syntax create batch [*owner*.]*batch_name*

sql_statements

endbatch

owner

Is the *owner* of the created batch. If the owner is not specified, the user name of the creator will be used as the default. If one is provided it must be a valid user in the database where the batch is created.

If the creator of the batch is the 'dbo' of the database there is no restriction as far as the specified owner. If the 'dbo' wants to create a batch that can be executed by all the users in the database, the owner name must be "publicbatch".

If the creator of the batch is not 'dbo' of the database, only his own user name may be specified, any other user name specified will generate an error.

batch_name

This is the name of the object. The *batch_name* is case sensitive if the SQL Server is case sensitive. The *batch_name* follows the standard convention for identifiers in SQL Server, that is the first character must be an alphabetic character, the symbol _(underscore) may be used, and the length of the batch_name must not exceed 30 characters.

sql_statements

These are TRANSACT SQL statements or other SQL-Programmer Batch Object statements.

endbatch

This is used to signal the end of the Batch Object.

Execute Batch

Syntax execute batch [*owner*.]*batch_name*

owner

"owner" is the owner of the batch. If the owner is not specified, the user name of the user requesting a Batch Object execution will be used as the default during the execution of the statement. If owner is provided it must be a valid user in the database where the statement is executed.

If the user requesting the execution is the 'dbo' of the database there is no restriction as to which Batch Objects may be executed.

If the user is not 'dbo' of the database, only Batch Objects created with the user's name or those Batch Objects created as 'publicbatch' may be executed.

batch_name

This is the name of the object. The *batch_name* is case sensitive if the SQL Server is case sensitive. The *batch_name* follows the standard convention for identifiers in SQL Server, that is the first character must be an alphabetic character, the symbol _(underscore) may be used, and the length of the *batch_name* must not exceed 30 characters.

There are no limits to the number of nested levels of Batch Objects. You can nest CREATE statements such that a Batch Object could create, execute or drop other Batch Objects.

There must be exactly one space between the CREATE and BATCH, the BATCH and *batch_name*.

Drop Batch

Syntax

drop batch [*owner*.]*batch_name*

owner

"owner" is the owner of the batch. If the owner is not specified, the user name of the user requesting a drop will be used as the default during the execution of the statement. If owner is provided it must be a valid user in the database where the statement is executed.

If the user requesting the drop is the 'dbo' of the database there is no restriction as to which Batch Objects may be dropped. Only the 'dbo' can drop 'publicbatch' Batch Objects.

If the user is not 'dbo' of the database, only Batch Objects created using the user's name may be specified, any other user name specified will generate an error.

batch_name

This is the name of the object. The *batch_name* is case sensitive if the SQL Server is case sensitive. The *batch_name* follows the standard convention for identifiers in SQL Server, that is the first character must be an alphabetic character, the symbol _(underscore) may be used, and the length of the *batch_name* must not exceed 30 characters.

There must be exactly one space between the DROP and BATCH, the BATCH and *batch_name*

Examples of Using Batch Objects

Executing Batch Objects Within Batch Objects

The following Batch Object Code is used to Drop and Create a Table. Having a script to drop and create a table is nothing new to Microsoft SQL Server and Sybase SQL Server developers. These people are also aware, and they never forget, that dropping a table also automatically, without warning drops the indexes and the triggers associated with that table. Many of these people will therefore also have scripts to recreate the indexes and triggers. In some cases scripts to recreate the table with the triggers and indexes will exist bundled together and separate scripts will exist to recreate the indexes and triggers respectively. This causes maintenance nightmares if for example the trigger code changes.

Related Topics:

Batch Object: employee_table

Batch Object: employee_type_empid

Batch Object: employee_ndx_employee_ind

Batch Object: employee_trig_employee_insupd

Batch Object : employee_permissions

Batch Object: employee_table

/* Script to drop and create the employee table with all it's associated objects */

execute batch dbo.employee_type_empid /* to create the table we need the type empid */

GO

PRINT 'CREATING TABLE : dbo.employee'

GO

if exists (select * from dbo.sysobjects where id = Object_id('dbo.employee') and type in ('U','S'))

begin

drop table dbo.employee

end

GO

create table dbo.employee (

emp_id empid not null

constraint PK_emp_id

primary key nonclustered constraint CK_emp_id

check ((emp_id like '[A-Z][A-Z][A-Z][1-9][0-9][0-9][0-9][0-9][FM]') or

(emp_id like '[A-Z]-[A-Z][1-9][0-9][0-9][0-9][0-9][FM]')),

fname varchar(20) not null,

minit char(1) null,

lname varchar(30) not null,

job_id smallint default (1) not null

references pubs.dbo.jobs(job_id),

job_lvl tinyint default (10) not null,

pub_id char(4) default ('9952') not null

references pubs.dbo.publishers(pub_id),

hire_date datetime default (getdate()) not null)

GO

execute batch dbo.employee_ndx_employee_ind

GO

execute batch dbo.emp_trig_employee_insupd

GO

execute batch dbo.employee_permissions

GO

Batch Object: employee_type_empid

/* Script to create the empid datatype */

```
PRINT 'CREATING DATA TYPE : empid'
GO
if not exists (select * from dbo.systypes where name = 'empid')
begin
execute sp_addtype empid, 'char(9)', nonull
end
GO
```

Batch Object: employee_ndx_employee_ind

/* Script to drop and create the employee_ind index */

```
PRINT 'CREATING INDEX ON TABLE : dbo.employee'
GO
if exists (select * from sysindexes where id = object_id('dbo.employee') and
keys1 != null and name ='employee_ind')
begin
drop index employee.employee_ind
end
GO
create clustered index employee_ind
on dbo.employee ( lname, fname, minit )
GO
```

Batch Object: employee_trig_employee_insupd

/* Script to drop and create the employee_insupd trigger */

```
PRINT 'CREATING TRIGGER : dbo.employee_insupd'
GO
if exists (select * from dbo.sysobjects where id =
Object_id('dbo.employee_insupd') and type = 'TR')
begin
drop trigger dbo.employee_insupd
end
GO
create trigger dbo.employee_insupd
/* ********************** employee trigger ********************** */
/*          Because CHECK constraints can only reference the column(s)
on which the column- or table-level constraint has
been defined, any cross-table constraints (in this case,
business rules) need to be defined as triggers.
Employee job_lvls (on which salaries are based) should be within
the range defined for their job. To get the appropriate range,
the jobs table needs to be referenced. This trigger will be
invoked for INSERT and UPDATES only. For information about
triggers, see the Microsoft SQL Server Transact-SQL Reference. */
ON employee
FOR INSERT, UPDATE
AS
/* Get the range of level for this job type from the jobs table. */
DECLARE @min_lvl tinyint,
@max_lvl tinyint,
@emp_lvl tinyint,
@job_id smallint
SELECT @min_lvl = min_lvl,
@max_lvl = max_lvl,
@emp_lvl = i.job_lvl,
@job_id = i.job_id
FROM employee e, jobs j, inserted i
```

```
WHERE e.emp_id = i.emp_id AND i.job_id = j.job_id
IF (@job_id = 1) and (@emp_lvl <> 10)
BEGIN
RAISERROR ('Job id 1 expects the default level of 10.',16,-1)
ROLLBACK TRANSACTION
END
ELSE
IF NOT (@emp_lvl BETWEEN @min_lvl AND @max_lvl)
BEGIN
RAISERROR ('The level for job_id:%d should be between %d and %d.',
        16, -1, @job_id, @min_lvl, @max_lvl)
ROLLBACK TRANSACTION
END
GO
```

Batch Object : employee_permissions

/* Script to reestablish permissions on the employee table */

```
PRINT 'CREATING PERMISSIONS ON : dbo.employee'
GO
Revoke all on dbo.employee from public
GO
Grant Delete, Insert, References, Select, Update
on dbo.employee
to guest
GO
```

Batch Objects and Security

- SQL-Programmer Batch Objects are protected globally by the access security built into both Microsoft SQL Server and Sybase SQL Server.
- The SQL-Programmer user must have login rights to the server and access rights to the database housing the Batch Objects.
- The dbo can create objects specifying the user name "***publicbatch***", thereby implicitly granting the general SQL-Programmer population execute privileges to the batch.
- Users other than the dbo can create Batch Objects with their own user name only.
- Users other than the dbo can only execute Batch Objects created by them, or created by the dbo with the "***publicbatch***" user name.
- Users other than the dbo can only drop Batch Objects they have created.
- The dbo can drop or execute any Batch Object created in the database regardless of the user name the Batch Object was created under.

NB *When the SA enables batch objects, the SQL_Programmer_Objects table used to manage the Batch Objects is created. The default rights to this table are as follows;*

Revoke all on SQL_Programmer_Objects to Public

Grant Select on SQL_Programmer_Objects to Public

Grant Insert on SQL_Programmer_Objects to Public

Grant Delete on SQL_Programmer_Objects to Public

Grant Update on SQL_Programmer_Objects to Public

If the SA does not want to allow certain users the rights to create Batch Objects, the privilege to insert into the SQL_Programmer_Objects table must be revoked to the users not requiring this ability.

Although these rights are granted to these tables. No direct Insert, Update, Delete interface is permitted to the SQL_Programmer_Objects table.

The Future of Batch Objects

The tremendous advantage in organization, functionality, productivity, and security offered by using SQL-Programmer Batch Objects will indubitably drive the need to further the capabilities surrounding their use.

SFI, the creator of SQL-Programmer has responded to the needs of Microsoft SQL Server and Sybase SQL Server developers by providing state of the art products aimed at the sole purpose of increasing developer productivity. The invention of Batch Objects is consistent with this purpose. SFI will continue to improve the Batch Object environment.

Currently available from SFI is "BSQL", a command line interface mechanism which allows Batch Objects to be executed from outside the SQL-Programmer environment. This facility permits Batch Object execution to be scheduled using scheduling tools such as AT (NT's built in scheduler). **"BSQL"** is an NT and Windows 95 ready, 32 bit EXE which is executed from a command line with input parameters similar to ISQL and BCP. **"BSQL"** will also output results to parameter specified output and log files.

Coming soon from SFI is "SQL-Agenda" a graphical Remote Batch Object Scheduler (RBOS). This SQL-Programmer sister product will offer Microsoft SQL Server and Sybase SQL Server DBAs and developers a sophisticated means to schedule the execution of Batch Objects managed within their SQL Server environments.

SQL-Programmer will continue to offer the functionality to justify the claim of being "**The Only Complete Development Environment for SYBASE and Microsoft SQL Server Programmers**". Look for the next major release of SQL-Programmer to have even more Batch Object management functionality.