

Using Database Views and Stored Procedures

Offloading some of the processing from ColdFusion to the database server



By Jeff Houser

Most of us are never going to work on a Web site that gets the amount of traffic that Google or Yahoo (or MySpace)

gets. Unfortunately, you can get away with

bad coding practices on a small site that

only gets 100 unique visitors a month and no one will ever know.

With such a low visitor count, you'll never have to deal with the

"why isn't my site loading quick enough" problem.

Unfortunately, a successful Web site will most likely grow. With growth come more users and more load. When it comes time to examine the application you'll find that more often than not the ColdFusion server is the cause of the bottleneck, and that the database server is barely being hit. Maybe there's a way to offload some of the work that the ColdFusion server is doing to the database server? Yes, there is and this article is going show you how you can use SQL Server views and Stored Procedures to offload some of the processing from ColdFusion to the database server. (Note that if you're looking for tips on designing your database structure they can be found in the April '05 issue of this column).

Using a View to Pre-Join Tables

Suppose you had an application that you designed to keep track of your record collection. You might design a database to look like what you see in Figure 1. The bands table, albums table, and songs table contain the content. An intersection table, bands_albums, connects the bands and albums table. This is a many-to-many relationship: A band can have many albums, and some albums, such as compilations or soundtracks often have multiple bands on them. The albums_songs table is another intersection table connecting the songs with the album

it appears on. This is another many-to-many relationship, since an album can (and better) have more than one song. And in many cases, an album song will end up on a compilation or movie soundtrack.

Now suppose you want to write a query that retrieves all the songs by a particular band. In ColdFusion it might look something like the JH Code Segment 1 at the end of this article. You're joining five tables to get the information you need, and this can be considered a fairly complicated join. If we were to move this code into an SQL Server view, you'd achieve efficiency. A view is a virtual table that lets you represent data in alternate ways. The most common way to create them is using Enterprise Manager, but you can also use T-SQL. In Enterprise Manager, you can follow these steps:

1. Launch Enterprise Manager, open your database server, and select your database.
2. Click the views selection in your database.
3. Right-click in the content window and select "New View."
4. You can create your code either using the graphic interface or just writing the SQL in the SQL Window. I copied and pasted the SQL code from the ColdFusion code.

Since a view is like a table, each column must have a unique name. When doing joins with tables that share common field names, you'll have to make sure that you select one of the fields, or give the duplicate an alias. In this case, the BandID and SongID are both in two tables. Instead of selecting both in our view, I specified one table over the other.

When writing ColdFusion code you can access the view just like a table, as shown in JH Code Segment 2.

Notice that the view doesn't contain the query qualifier 'band = #variables.bandID#'. The view returns all the bands and their respective songs. SQL Server will optimize views using built-in optimization techniques. These techniques aren't applied to single queries. You can also index views the same way you index tables.

Advanced Logic in a Stored Procedure

Suppose that our record database is implemented using CFCs and a hint of object orientation. There's a band object that

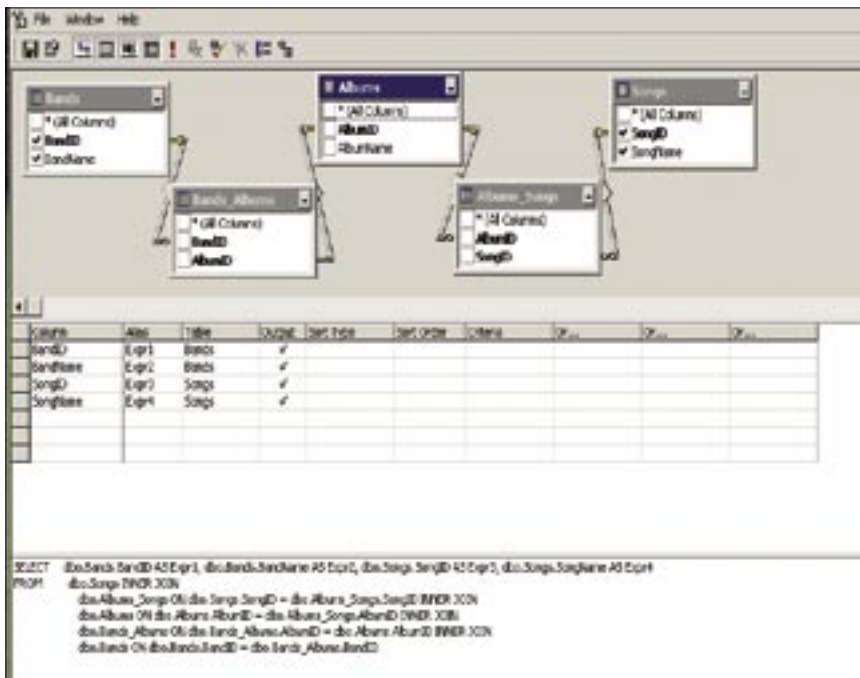


Figure 1: Record collection database diagram

contains an array of album objects that contains an array of song objects. When you init the band object, it runs a query to find out about the band's albums. In a loop it initializes the album objects, and each album initialization results in another query call to get the album information. In each album init function, a query is executed to get the song information. The song information is looped over to initialize the song objects. Each song makes yet-another query call. For a band with two 12-song albums, you'll end up with at least 27 different calls to the database. No wonder your application server is starting to smoke.

In many of my objects, I'll often implement an "initByData" method. Instead of a method that makes a database call and sets up the component, all data is manually passed into this method to set up the component. Using this method, I won't have to daisy chain calls to the database. So instead of 27 different database calls, I can use one query to get the band information, one query to get the band's album information, and one query to get the song information about each album. Our database calls went from 27 queries to three. Not bad.

Using a stored procedure, we can put

all three queries in a single stored procedure call, resulting in only one trip to the database. This will put more pressure on the database server, but will most likely give your ColdFusion application server a break. (It's been working hard, so it's earned it.) You can follow these steps to create a stored procedure:

1. Launch enterprise manager, open your database server, and select your database.
2. Click the Stored Procedure selection in your database.
3. Right-click in the content window and select "New Stored Procedure." Does this sound familiar yet?
4. Write your stored procedure. The finished code is in JH Code Segment 3.

The stored procedure code starts out with the "Create Procedure" command. Then comes the name of the stored procedure (GetBandInfo). Then a comma delimited list of all arguments to the stored procedure. In this case, we just have one argument, the bandID. Arguments are distinguished by using the @ symbol before the argument name. The keyword AS comes next and then the code that you want to run. This stored

procedure is a relatively simple one containing only selects. Stored procedures can also contain more advanced logic, including conditionals using if else, looping over recordsets through the use of cursors.

Stored procedures are precompiled objects. Unlike traditional queries (that you execute using the cfquery tag), the code doesn't have to be compiled first, which improves performance at runtime. You can execute a stored procedure using the cfquery tag using the 'exec' command, like JH Code Segment 4.

This method works, but only returns a single recordset to ColdFusion. It's undesirable in situations where you want to return multiple recordsets. The alternative is to use the cfstoredproc tag. Full documentation on the tag can be found at <http://livedocs.macromedia.com/coldfusion/7/html/docs/00000338.htm#2607555>. Some attributes are similar to what you'd find in the cfquery tag, such as a datasource, a username, password, blockfactor, and result. Other attributes are specific to cfstoredproc, such as procedure for the procedure name.

Along with the cfstoredproc, there are two subtags that can be used. Cfprocparam is used to send information to the stored procedure, and full documentation can be found at <http://livedocs.macromedia.com/coldfusion/7/html/>

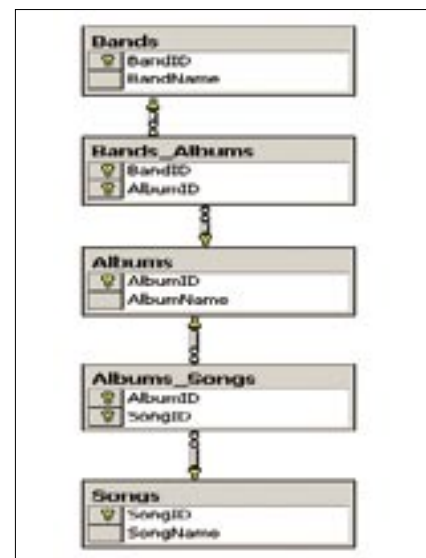


Figure 2: Creating a view

docs/00000313.htm#1102102. Cfprocresult is used to return information from the procedure. Full documentation for cfprocresult can be found at <http://livedocs.macromedia.com/coldfusion/7/html-docs/00000314.htm#1102246>. The code is in JH Code Segment 5.

For cfpropparam, the two parameters needed for this example are the value and the cfsqltype. The cfprocresult tag uses two parameters: name and resultset. The name is the variable that will contain the resultset after the stored procedure has executed. This is similar to the name attribute of the cfquery tag. The resultset should contain a numerical value. If the value is one, then the value will be given the first recordset returned from the stored procedure. If the value is two, then the variable is given the second record set.


Where To Go from Here

This article is intended to give you a peek into the power of using the database to do your dirty work instead



of ColdFusion. There's plenty more to learn, so what should you look into next? You can start by investigating aggregate functions, which can be used to do calculations inside of a query. I talked a little about aggregate functions in the January '05 issue of this column. More info can be found at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tsqlref/ts_fa-fz_9yuk.asp. You might also look up user-defined functions in SQL Server. They're similar

in concept to UDF in ColdFusion just implemented a bit differently. More info can be found at <http://msdn.microsoft.com/msdnmag/issues/03/11/Data-Points/>.

You might also look up how to create views and stored procedures using Transact SQL. With one of my current projects, we decided to create SQL files that contain the T-SQL to create the views and stored procedures. By storing these files in the repository, we're able to track database changes and even add some version control to the code. 

About the Author

Jeff Houser has been working with computers for over 20 years and in Web development for over 8 years. He owns a consulting company and has authored three separate books on ColdFusion, most recently ColdFusion MX: The Complete Reference (McGraw-Hill Osborne Media).

jeff@instantcoldfusion.com

JH Code Segment 1

```
<cfquery name="getSongs" datasource="MyDSN">
select Bands.BandID, Bands.BandName, Songs.SongID, Songs.SongName
from songs join albums_songs on (songs.songID = albums_songs.songID)
join albums on (albums.albumID = albums_songs.AlbumID )
join bands_albums on (bands_albums.albumID = Albums.albumID)
join bands on (bands.bandID = bands_albums.bandID)
where bandID = #variables.bandID#
</cfquery>
```

JH Code Segment 2

```
<cfquery name="getSongs" datasource="MyDSN">
select BandID, BandName, SongID, SongName
from SongsView
where SongsView.bandID = #variables.bandID#
</cfquery>
```

JH Code Segment 3

```
CREATE PROCEDURE GetBandInfo
```

```
@BandID int
```

```
AS
```

```
select * from bands
where bands.bandID = @BandID
```

```
select albums.*
from albums, bands_albums
where bands_albums.albumID = albums.albumID and bands_albums.bandID =
@BandID
```

```
select albums.albumID, songs.*
from albums, bands_albums, albums_songs, songs
where bands_albums.albumID = albums.albumID and
bands_albums.bandID = @BandID and
albums_songs.albumID = albums.albumID and
songs.songID = albums_songs.songID
GO
```

?? JH Code Segment 4

```
<<cfquery name="getSongs" datasource="MyDSN">
exec GetBandInfo #variables.bandID#
</cfquery>
```

?? JH Code Segment 5

```
<cfstoredproc procedure="GetBandInfo" datasource="MyDSN">
<cfpropparam value="#variables.bandID#" cfsqltype="cf_sql_integer">
<cfprocresult name="getBand" resultSet="1">
<cfprocresult name="getAlbum" resultSet="2">
<cfprocresult name="getSong" resultSet="3">
</cfstoredproc>
```

Download the Code...
Go to www.coldfusionjournal.com