5 Strategies for ArcObjects Developers

By Robert Stauder, ESRI Applications Developer

Achieving better enterprise database performance

What can ArcObjects developers do to increase performance of an enterprise application that uses data from ArcSDE? While many application developers and database administrators (DBAs) approach database tuning as an exercise in creating missing indexes, updating statistics, adding disks or RAM, and analyzing hardware performance, there are other techniques available. Here are five suggestions, presented in order of increasing difficulty, that will help developers squeeze more performance from their applications.

Using the Code Samples and Data

Each performance-enhancing suggestion is illustrated with a code listing that uses the Parcels features class from the Building a Geodatabase data that comes with the ArcGIS sample dataset. The Parcels feature class resides in the LandBase feature dataset of the Montgomery File Geodatabase. Load the data into an ArcSDE database using ArcCatalog. Register the Parcels feature class as versioned.

Download the code samples from *ArcUser Online* (**www.esri.com/arcuser**). These samples use ArcGIS 9.3.1 and Microsoft Visual Studio .NET 2008. Sample programs are VB.NET and C# console applications that use ArcInfo licenses. The code accesses licenses using the LicenseInitializer classes generated by Visual Studio .NET. Error, pathing, and null reference checking were omitted to make samples as lean as possible and improve readability.

Finally, these console applications require a path to an ArcSDE connection file. Use ArcCatalog to copy an ArcSDE connection file to a directory of your choice and reference this file when running the programs. Because two of the samples will change data, create a new version in ArcSDE and save your connection file referencing that version.

ArcGIS supports combined spatial and attribute queries against enterprise geodatabase data. The query in Code Listing 2 restricts the number of rows traversed because features must be contained in the red box and have an area greater than 121,000. Those features are highlighted in yellow.

Code Listing 1: Use SubFields

What It Does:

The first performance improvement suggestion involves limiting how much data an application fetches. The SubFields property of IQueryFilter, IQueryDef, and ISpatialFilter lists the fields to include in a query. Set this property to just the fields values you need, rather than all the data for each row, and you will fetch less data. The database will search less data and send fewer packets across the network.

How to Use:

Populate the SubFields property of IQueryFilter, ISpatialFilter, or IQueryDef with a comma-delimited string of column names.

When to Use:

Use to optimize read-only queries.

Code Listing 2: Combine Spatial and Attribute Queries

What It Does:

One of the most powerful aspects of ArcGIS is the ability to combine the spatial and attribute components of a query so you can issue fewer queries and make those queries more selective. A single query with two filters—spatial and attribute—limits the number of features searched.

How to Use:

Set the Geometry and WhereClause properties of ISpatialFilter. Set the SearchOrder property to control which query component is applied first. SearchOrder has two choices:
esriSearchOrderAttribute sets the search order to attribute first.
esriSearchOrderSpatial sets the search order to spatial first.
Set SearchOrder to the most restrictive option. If your query geometry is small, choose esriSearchOrderSpatial. If your query's where clause is very selective and your geometry larger, choose esriSearchOrderAttribute. Create a test case and try both options.

When to Use:

Use for any query having both spatial and attribute qualities. Optimize spatial queries by applying an attribute constraint to them.

Code Listing 3: Faster Deletes

What It Does:

Most ArcObjects developers employ either the ITable DeleteSearchedRows() or IRow Delete methods to remove features in bulk from a geodatabase table or feature class. Switch to a low-level interface, ITableWrite, to improve the speed of bulk deletes. During a feature delete, this interface sends fewer (or no) messages and bypasses geodatabase behaviors. This can translate into much faster bulk operations on simple features. Do not use this method with complex features like geometric networks.

How to Use:

Use a query filter and fetch rows into an ISet.
Cast the table or feature class queried to ITableWrite.
Pass the ISet to the DeleteRows method of ITableWrite.

When to Use:

Use to improve the speed of bulk deletes on simple data.

Code Listing 4: Don't Be Too Selective (or Chatty)

What It Does:

Two previous sections discussed how to make queries more selective. However, if queries are too selective, you may need to issue too many of them. Query-intensive applications are called "chatty." Chatty applications do not perform well.

For example, you query a collection of hundreds of Parcel IDs. For each ID, you query the database for other features and process them in some fashion. The excessive round-trips to the server to fetch more data slows your application. To improve performance, issue one query with a where clause containing all Parcel IDs, then store the result set in client-side memory as a Geodatabase RecordSet inside an in-memory table. Using this approach will allow you to requery your data without returning to the server. You'll also have access to geodatabase table functionality and geometries within spatial fields.

How to Use:

Build a where clause containing all items you want to fetch.

- Create a query filter and set the where clause property to the where clause you just created.
- Create a Geodatabase RecordSet and set the source table property.
- Create a new InMemoryWorkspace and save the RecordSet as a table within it.

When to Use:

Use in any situation where you will need to repeatedly query a feature class or table on the same attribute. Instead of using a join, query two different feature classes or tables, fetch the data into client-side RecordSets, then process the data from the feature classes or tables together.

Code Listing 5: Faster Updates

What It Does:

This example combines the previous two to perform faster bulk updates. When you invoke the IRow Update or ITable UpdateSearchRows methods against multiversioned data, ArcSDE creates rows in both A and D tables. You can mimic this by deleting the rows using ITableWrite's DeleteRows, then reinserting them using an InsertCursor. Before deleting the rows, store a copy of them in a client-side RecordSet. This method is faster because ITableWrite will bypass geodatabase behaviors. Use this technique with simple features only.

Employing a delete-insert will impact the versioning reconcile process. Reconcile checks for conflicts between two versions by querying for change types within those versions. One of those change types is update-update, where a row is updated in one version, and that same row is updated in another. If you update your data using a delete-insert, the update-update filter will not find conflicts because your code only removes and reinserts rows. Bulk update workflows usually do not care about conflicts. If your bulk update process must consider conflicts, exercise caution when using this technique.

How to Use:

Query and fetch features into an ISet and a RecordSet.

- Pass the ISet to the DeleteRows method of ITableWrite to remove the features from the database.
- Loop through the features in the RecordSet, inserting features and modifying them as necessary.

This works against both spatial and tabular data.

When to Use:

Improve speed of bulk updates on simple features.

Some Final Words

Improving the performance of an enterprise application means examining all parts of it. Applications using relational data require not only good indexes and up-to-date statistics but also good data access practices. Making queries more selective, caching data on the client to reduce network trips, and changing batch delete and update operations are valid performance tuning tricks. By making the client more efficient, all parts of the enterprise application benefit.

About the Author

Robert Stauder is an applications developer with ESRI Professional Services in Redlands, California. After seeing an ArcView 1.0 demo in 1994, he decided to change his career focus to GIS and joined ESRI in 1996.

Visit *ArcUser Online* (**www.esri.com/arcuser**) to download these listings.

For more information, take the ESRI instructor-led course *Building Geodatabases.*