Performance Best Practices

Last Modified on 01/16/2018 7:07 am EST

You can improve performance for several dynamic workflow components. There are an unlimited number or ways to develop a solution. When building a solution, balance your requirements with your resources.

Database Performance

You can improve database performance:

- During development
- During production
- With ongoing maintenance

During development

Avoid creating tables a high number of columns.

During production

Performance issues might occur any time a select is made, such as during form loading, reports, and so on. If you encounter these issue, you should:

1. Determine where DB queries occur in your solution.
2. Run SQL Profiler to identify which queries take the longest time to run.
3. Determine why these queries take a long time to run.
4. Run each of these queries with an execution plan on it. Check which *Joins* take the most time.
5. Add indexes to the appropriate tables, or write a more effective query.

Be aware that if you add an index, retrieval time is shortened, but data input takes longer. For more information about enhancing DB performance, read this Red Gate E-Book, specifically the sections on T-SQL Tips and Index Tips.

Ongoing maintenance

- Periodically run the SQL Profiler to identify slow procedures. Review the execution plan for these slow procedures to determine where you can place an index on UACTS, UWF, or UTCMB.
- Back up your database on a daily basis.
- Regularly check the error log file to determine if the log size is normal. If no, investigate the errors.

Forms

- Keep it simple. Only add necessary functionality to forms. Unnecessary functionality extends creation time and does not always result in better user experience.
- Use wizards, for example Next -> Next -> Next, rather than including all fields on a single screen.
- Try using server code.
- If your form loads information from a service, make sure you optimize the message size. To track the message size, enable the integration diagnostic in the `web.config` file.
- Consolidate SQL calls. If you have many `{sql:...}` expressions in your form markup, check whether you can use a single call and split the data afterwards.
- Try to define a select key in each table data query. If you do not define a select key, we recommend

that you apply a filter on each execution.

- If your forms use a *Stored Procedure* or a *Web Service* that returns a large amount of data, use paging, such as for grids or combos. We recommend that you implement pagin on the side of the *Stored Procedure* or a *Web Service.*
- After you develop a form, run the form when the SQL Profiler is open to verify that your DB calls are optimized.
- If you are mapping more than one field (column) from the same record, use data source.

```
ds={Form1}.Query("vltemslnStock")
ds.Name
ds.Price
...
```

- When you use a combo with many entries, use the load-on demand option. If the message is too large, ask the party that is exposing the service for optimization. If your service returns many records, ask the party if it supports paging.
- Avoid using extensive logic with third-party integration before the first form. Instead, look for an offline solution.
- For very slow system, consider loading the grids from the service side, or on demand. We discourage using code behind forms, use this as an option of last resort.
- Cora SeQuence forms are standard .NET forms, so the standard best practices also apply. For more information see the .NET Forms Best Practices.

Expressions

When you use queries in expressions, be aware that Cora SeQuence retrieves the entire table, which it filters by the select key defined in the table query. The retrieved data is cached for each workflow execution until the activity is executed again, or explicitly modified by the code.

- When you have a table without a key definition, use *AsQueryable()* to filter the data in the DB, not at the server level.

```
{Form1}.Query("vItemsInStock").AsQueryable().Where(...
```

- If you need user details, use the User function to get the user by *user id*, or *rt.currentuser* if it is the current user. Cora SeQuence then uses the cached object and does not bring the value from the DB.
- In the expression, use *AsQueryable()* and then *Where(...)*. This applies the Where filter on the DB side. Consider also using *Select(...)* to retrieve specific columns. For example:

```
{StartTest}.Query("UACT1").AsQueryable().Where(Field("fldld")>0).Select(Field("fldld"))
```

Other Performance Tips

- Do not use activities for decoration. On high-volume systems, activities can have negative effects on the DB, and eventually on performance.
- Set Link mode to automatic wherever possible, especially after a human activity. This way, the user receives an automatic response that data was saved and the BRS will perform the remaining activity execution.
- If you use Stored Procedures, Views, or Keys, add indexes on the relevant fields in User Tables. Do not add indexes to System Tables.

- If you use a Web Service, check how often it is called. If it is called frequently, consider a different implementation.
- Each query execution can degrade performance. For example, if your form includes a query that is executed from three different places, the query is executed three times.
- Try to calculate and render on the server side, not the client side.
- Try to minimize post-backs and calls to the server.