Article
Eduard Lebedyuk · Jan 9, 2017 3m read

How to determine row level security at runtime

In addition to its general security, Caché offers SQL security with a granularity of a single row. This is called row-level security. With row-level security, each row holds a list of authorized viewers, which can be either users or roles. By default access is determined at object modification Some time ago I became interested in determining row-level security at runtime. Here's how to implement it.

Some notes on RLS:

- Row-level security is only available for persistent classes.
- Row-level security is only available for tables instantiated on the Caché server. It is not available for link tables (that is, those that are instantiated on foreign servers).
- Row-level security is only enforced when accessing rows from SQL. It is not enforced when directly accessing globals or when accessing globals via the object interface (define %OnOpen callback to add rls for objects).

Here's the simple example of rls enabled class:

```
Class Utils.RLS Extends %Persistent
{
Parameter ROWLEVELSECURITY = 1;

Property %READERLIST As %String;
}
```

Value of %READERLIST property is a comma-delimited string listing users or roles that may view the row (empty string for all users).

So, for example if %READERLIST for one row is "SYSTEM,%Development" then SYSTEM user can access the row and all users who hold %Development role.

%READERLIST value is used in %RLS index which in turn is used by SQL to determine access.

Dynamic row-level security

By default %READERLIST is provided by user, stored, indexed and used to determine access. But I wanted dynamic row-level security, for example during each access attempt call a method which determines - does the user has access or not. To achieve that result two things are required:

- Disable %RLI index (or use %IGNOREINDEX in every query)
- Make %READERLIST property always calculated

Thankfully both of these things are easily achievable, to disable %RLI index for selected class execute once:

```
do $System.SQL.SetMapSelectability(class, "%RLI", $$$NO)
```

You may also need to purge old queries which could use the index:

```
do $System.SQL.PurgeForTable(class)
```

That done, lets make %READERLIST property always calculated:

```
Property %READERLIST As %String [ Calculated, Private, SqlComputeCode = {s {*} = ##cl
ass(Utils.RLS).GetAccess({ID})}, SqlComputed ];

ClassMethod GetAccess(Id) As %String
{
    return:Id>3 "_SYSTEM"
    return "%All"
}
```

Here only _SYSTEM user can access all records with Id>3 and for Id 1 and 2 only users with %All can have access.

And here's the complete example:

```
Class Utils.RLS Extends %Persistent
{

Parameter ROWLEVELSECURITY = 1;

Property %READERLIST As %String [ Calculated, Private, SqlComputeCode = {s {*} = ##cl
ass(Utils.RLS).GetAccess({ID})}, SqlComputed ];

Property data As %String;

ClassMethod GetAccess(Id) As %String
{
    return:Id>3 "_SYSTEM"
    return "%All"
}

/// do ##class(Utils.RLS).Fill()
ClassMethod Fill(N = 5)
{
    do ..%KillExtent()
    for i=1:1:N {
        &sql(insert into Utils.RLS(data) values(:i))
    }
    do $SYSTEM.SQL.SetMapSelectability($classname(),"%RLI",$$$NO)
    do $system.SQL.PurgeForTable($classname())

    do ##class(%SQL.Statement).%ExecDirect(,"select * from "_$classname()).%Display()
}
}
```

If I execute:

```
do ##class(Utils.RLS).Fill()
```

As a user with %All I receive the following output:

```
ID      data
1       1
2       2
3       3

3 Rows(s) Affected
```

And as SYSTEM I receive all 5 rows:

```
ID      data
1       1
2       2
3       3
4       4
5       5

5 Rows(s) Affected
```

Advantages of dynamic row-level security

- You can determine access based on external state: global value, day of week, holidays, %request state, etc.
- You don't need to modify %READERLIST to change access permission, only modification of GetAccess method is required

Disadvantages of dynamic row-level security

- Slow
- Uncompilation enables %RLI index. UnCompilation hook is required to disable index again

Links

- Documentation
- GitHub

Author would like to thank an engineer who provided main implementation ideas for this article at Russian Caché forum.

#Caché #SQL

Source URL:https://community.intersystems.com/post/how-determine-row-level-security-runtime