Article
Matthew Giesmann · Nov 8, 2016 4m read

Introduction to Outlier Selectivity

Beginning in Caché 2013.1, InterSystems introduced Outlier Selectivity to improve query plan selection involving fields with one atypical value.

In this article, I hope to use an example 'Projects' table to demonstrate what Outlier Selectivity is, how it helps SQL performance and a few considerations for writing queries.

Selectivity

First, let's take a quick look at Selectivity. Selectivity is meta information about the values in one column in a table. Assuming a typical distribution of data, it answers the question "If I ask for all the rows in this table that have a certain value in this column, what fraction of the table will I typically get back?"

Consider a hypothetical " Pojects " table that has two fields: Owner and Status. Owner is the employee responsible for the project and Status can be one of four options: PREP, OPEN, REVIEW, and COMPLETE. In the class storage, we see the selectivity values after running Tune Table:

```
<Property name="Owner">
    <Selectivity>3.3726%</Selcetivity>
</Property>
<Property name="Status">
    <Selectivity>25.000%</Selcetivity>
</Property>
```

Now consider the following 2 queries:

```
SELECT * FROM Projects WHERE Owner = ?

SELECT * FROM Projects WHERE Status = ?
```

On average, the first query will return a little more than 3% of all the projects in the projects table. On average, the second query will return 25%. If these tables were involved in a query with JOINs or multiple WHERE conditions, 3% vs. 25% could make a big difference in execution time, and could change which query plan Caché chooses to run.

Outlier Selectivity

Selectivity doesn't tell the whole story! Sometimes, the distribution of possible values in a field aren't typical. Outlier Selectivity allows us to be more clever for fields that have one value that is atypical⋯ an outlier.

In our example Projects table, projects can have one of four Statuses, but after a few years there will be many more projects in COMPLETE than any of the other categories.

Again, *on average*, the following query will return 25% of the projects table:

```
SELECT * FROM Projects WHERE Status = ?
```

But, we can make a better estimation than that! If the WHERE clause is "WHERE Status = 'COMPLETE', the we know that we will get most of the table back. "WHERE Status = 'PREP', on the other hand, will only return a small percentage.

Where before we stored:

```
<Property name="Status">
    <Selectivity>0.25</Selectivity>
</Property>
```

With the introduction of Outlier Selectivity, we now store:

```
<Property name="Status">
    <OutlierSelectivity>0.9:"COMPLETE"</OutlierSelectivity>
    <Selectivity>0.03333</Selectivity>
</Property>
```

Now, we can differentiate between the following 2 queries:

```
SELECT * FROM Projects WHERE Status = 'COMPLETE'

SELECT * FROM Projects WHERE Status = 'PREP'
```

The first is expected to return 90% of all projects in the table, but the second is only expected to return 3%.

For queries involving multiple tables or choices between multiple indexes, 90% vs. 3% of a table can make a big performance difference and can again change which query plan the SQL engine chooses.

Queries with Outlier Selectivity

Outlier Selectivity does give some benefit with no changes to your application, but there are some things to consider to take full advantage. By default, Caché generates one query plan for all queries with the same form. (Like above, WHERE Status = 'COMPLETE' and WHERE Status = 'PREP')

By default, Caché assumes a non-outlier value for your query parameters. To force a query to consider the outlier value, use double parens to suppress literal substitution for outlier values:

```
SELECT * FROM Projects WHERE Status = (('COMPLETE'))

SELECT * FROM Projects WHERE Status = 'PREP'
```

The double parens force the SQL engine to generate a plan for the specific value of the parameter in your query. Caché can now use 2 different plans for this query, knowing when it expects to retrieve 90% of projects and when it

expects 3%.

You can also control whether Caché defaults to assuming non-outlier values by setting the BiasQueriesAsOutlier value to 1 or 0. The following will cause Caché to assume that queries that use the outlier value are not rare:

```
<Property name="Status">
    <BiasQueriesAsOutlier>1</BiasQueriesAsOutlier>
    <OutlierSelectivity>0.9:"COMPLETE"</OutlierSelectivity>
    <Selectivity>0.03333</Selectivity>
</Property>
```

Hopefully, these examples shed some light on what Outlier Selectivity is and why it improves query performance. For another presentation of this information and more on other SQL stats, see the docbook for Selectivity and Outlier Selectivity.

#SQL #Caché