Indexing Lookup Tables

This feature requires a CommCare Software Plan

This feature (Lookup Tables) will only be available to CommCare users with a Standard Plan or higher. For more details, see the CommCare Software Plan page.

Overview

Indexes are designed to solve a specific problem: references to a large lookup table are slowing down your application. For example, suppose you are using a lookup table to Setup a Question with Filtered Choices. Your lookup table, health_centers_by_region, has the following fields:

- **district_id**
- **health_center_id**
- **mnch_services_available**

and you have a hidden value that contains the calculation:

- count(instance('health_centers_by_region')/health_centers_by_region_list/health_centers_by_region[mnch_services_available="yes"][district_id = /data/district_id])

If this table is large enough that this filter is slowing down performance in your form, you may be able to alleviate this by adding an index to district_id and moving the indexed field to the front of your filter, so that:

- [mnch_services_available="yes"][district_id = /data/district_id]

becomes:

- [district_id = /data/district_id][mnch_services_available="yes"]

Definitions

- A **query** refers to any XPath expression that retrieves data from a lookup table. This includes filters on lookup table questions, as well as any calculations that contain something of the form: instance('table_name')/table_name_list/table_name[*filter_1*][*filter_2*]...[*filter_n*]
- The **cardinality** of a field in a lookup table is how many *unique* values of that property are represented. The more unique values are represented in the table, the higher that field's cardinality. For example, an ID field for which each value only appears once in the lookup table has very high cardinality, while a True/False field has only two possible values over the entire lookup table, so has very low cardinality.

What is an Index?

An index is a data structure that helps make certain XPath expressions that reference a lookup table more efficient. When you index a field in a lookup table, you can improve the speed of expressions on that particular field. The following scenarios highlight the difference between querying a field that is indexed, compared to querying an unindexed field:

- **Example Query:** instance('state')/state_list/state[name="Bihar"]/population
- ***name* is not indexed:** The query evaluator loads a row of the **state** table, checks to see whether its *name* property is equal to "Bihar," returns the *population* field if so. Repeat for *every single row of the table*.
- ***name* is indexed:** The query evaluator checks the index to find the exact location in the table of all elements where *name* is equal to "Bihar," loads all matching elements in one operation, returns their *population* fields.

When Should I Create an Index on my Lookup Table?

Most lookup tables will not need to be indexed. If you think you might need to index one or more fields of a lookup table, ask yourself:

- Is my lookup table very large (many hundreds of records or more)?
- Do I query the same few fields over and over again in my app?
- Have I noticed slow app performance related to these queries?
- Do any of these fields have a high cardinality?

If you answered "yes" to all of the above questions, it might be worth adding an index for one or more fields in your lookup table. If you're not sure whether a particular query is slowing down your app, try the following exercise:

1. Create a form consisting of 3 questions:
    a. label1
    b. lookup_query
    c. label2
2. label1 and label2 are */abel/* questions, and can contain whatever you want
3. lookup_query is a *calculation* question, whose calculate condition is set to the exact query that you believe might be causing the slowdown.
    a. If your query contains references to question_ids in the form whose values are dynamically generated, replace these with a static representative value. For example, in the query from the Overview, we can select an arbitrary district_id (e.g. "123") to use in place of /data/district_id.

b. If you are concerned about a filter on a multiple choice question, your query is going to be: instance('table_name')/table_name_list /table_name[*your filter here*], where table_name is the name of the lookup table. In the example from the overview, our query would be: instance('health_centers_by_region')/health_centers_by_region_list/health_centers_by_region[district_id="123"] [mnch_services_available="yes"]

Indexing Tables Used in Multiple Choice Lookup Table Questions

If you are populating a multiple choice or checkbox question from a lookup table, and are still experiencing performance issues after indexing any appropriate columns from the filter, try adding indexes for the Value Field and the Display Text Field.

How Do I Add an Index?

Suppose you have a lookup table health_centers_by_region with the following fields (this is the same table from the Overview above):

- **district_id**
- **health_center_id**
- **mnch_services_available**

As part of your workflow, a mobile worker often needs to select a health center in a certain region that has MNCH services available. There are thousands of health centers spread over hundreds of districts, and you've noticed that a query that appears throughout the app is taking a long time:

- instance('health_centers_by_region')/health_centers_by_region_list/health_centers_by_region[mnch_services_available = "yes"][region_id=/data /region_id]

Since district_id has hundreds of values represented, it has a high enough cardinality to warrant an index. mnch_services_available will only ever be set to "yes" or "no." Since the cardinality for this field is low, the benefits of indexing the field would be minimal compared to the extra space it would take up on the phone, so it should not be indexed.

Before adding an index to the lookup table, please refamiliarize yourself with the process of Creating and Updating Lookup Tables, specifically Downloading or Uploading Tables for Editing.

When you're ready to add an index, follow these steps:

1. Download the lookup table following the steps in the previously linked documentation.
2. Open the file and navigate to the **types** tab. It should look something like this:

Delete(Y/N)	table_id	is_global?	field 1	field 2	field 3
N	health_centers_by_region	yes	district_id	health_center_id	mnch_services_available

3. Add a column with header "field 1 : is_indexed?" and mark the column as "yes" for the field and table that you want to index:

Delete(Y/N)	table_id	is_global?	field 1	field 1 : is_indexed?	field 2	field 3
N	health_centers_by_region	yes	district_id	yes	health_center_id	mnch_services_available

Once the index has been added, you will need to rewrite your query so that the indexed field or fields appear at the beginning of the string of filters. Since region_id is indexed and mnch_services_available is not, our example becomes:

- instance('health_centers_by_region')/health_centers_by_region_list/health_centers_by_region[region_id=/data/region_id] [mnch_services_available = "yes"]

See the next section for details.

Query Optimization

When writing XPath queries on an indexed lookup table (or on any indexed fixture, such as casedb), it is important to write your query such that all filters on indexed properties are evaluated BEFORE other filters. For example, say you have a lookup table health_centers_by_region with the following fields:

- **region_id** (indexed)
- **health_center_id**
- **mnch_services_available**

If we want a query to return all health centers in region 123 where MNCH services are available, we would write:

- instance('health_centers_by_region')/health_centers_by_region_list/health_centers_by_region[region_id="123"][mnch_services_available = "yes"]

With this query, the XPath query evaluator will first use an efficient key lookup to filter on the indexed field (region_id), before performing single lookups on each of the results to check whether mnch_services_available = "yes" for each one.

The following documents go into much further technical detail on XPath evaluation and optimization:

- Primer: XPath Query Evaluation
- Performance: XPath Query Evaluation

Limitations

Complex Queries

Currently, only the most basic XPath queries will leverage lookup table indexing. For example: *value = 1* will use the indexing, but *value < 2* will not. In addition, composite queries using *and* or *or* will not leverage the indexes.

Lookup Tables with Multiple Languages

When indexing a lookup table with multiple languages, you can only index fields that do **not** have translations. Indexing columns with attributes is unsupported.