Notes on Advanced Datastore Querying

CS 493: Cloud Application Development

Last update: May 17, 2024.

In Module 3, we discussed how we can use GCP's Datastore Python client to perform CRUD operations against Datastore. In this document, we will look at some advanced features for querying Datastore that are needed in Assignment 6.

Contents

Composite Query Filters

In Datastore we can add filters to a query like how we can specify conditions in the WHERE clause of SQL SELECT statements. We saw an example of adding a query filter in Module 3. SQL SELECT statements allow adding multiple conditions in the WHERE clauses combining them using AND or OR. We can do the same thing with Datastore queries.

Combining Conditions Using Logical AND

Adding multiple filters to a Datastore query with the AND semantics and using the = relational operator is very simple – we simply call the `add_filter` method multiple times and all the filters added to the query are AND-ed together

Example: Suppose for our example Lodging API we want to get only those lodgings that had a particular value of price and a specific description. We can meet this requirement by adding 2 filters on a query to fetch lodgings, as show below:

```
    lodging_query = client.query(kind=LODGINGS)
    lodging_query.add_filter(filter=PropertyFilter('price', '=', 200))
    lodging_query.add_filter(filter=PropertyFilter('description', '=', 'Very
colorful!'))
    results = list(lodging_query.fetch())
```

We are using `PropertyFilter` so the code shown above also needs the following import statement

```
from google.cloud.datastore.query import PropertyFilter
```

Optional: Combining Conditions Using Logical OR (not needed for Assignment 6)

If we want to combine filters using OR, then we will need to call the `Or` method on the `query` class.

Example: The following query will match all entities for which either the value of price is 200 or the description is 'Very colorful!' or both.

```
    lodging_query = client.query(kind=LODGINGS)
    lodging_filter = query.Or(
        [
            query.PropertyFilter('price', '=', 200),
            query.PropertyFilter("description", "=", 'Very colorful!'),
        ]
    )
    lodging_query.add_filter(filter=lodging_filter)
    results = list(lodging_query.fetch())
```

For this we need the following import statement

```
from google.cloud.datastore import query
```

Reference

To learn more about this topic, see

https://cloud.google.com/datastore/docs/concepts/queries#composite_filters

Note that there is some additional setup we need to do if we want to combine conditions using inequality relational operators.

Adding a sort order

We can add ORDER BY clause to a SQL statement to sort the results. Datastore also support sorting the results.

Sorting by one property

In Datastore, we can set the order property to sort the results by that property. By default, the results are sorted in ascending order.

Example: In the following example, the results will be ordered by ascending value of price.

```
    lodging_query = client.query(kind=LODGINGS)
    lodging_query.order = ['price']
    results = list(lodging_query.fetch())
```

To sort results by descending order, we can add the minus character before the name of the property.

Example: In the following example, the results will be ordered by descending value of price.

```
    lodging_query = client.query(kind=LODGINGS)
    lodging_query.order = ['-price']
    results = list(lodging_query.fetch())
```

Optional: Sorting by multiple properties (not needed for Assignment 6)

To sort the results of a query by multiple properties, we need to create composite indexes on those properties. For more details see the following reference:

https://cloud.google.com/datastore/docs/concepts/indexes#index_configuration

Reference

https://cloud.google.com/datastore/docs/concepts/queries#sort_orders

Pagination

In Module 5, we studied how to implement pagination when using MySQL database. Datastore also provides support for pagination. In fact, Datastore supports both limit-offset based pagination and cursor-based pagination.

Limit-Offset Based Pagination

The method `fetch` on a Datastore query can be passed the arguments `limit` and `offset` to specify the values of limit and offset for this execution of the query. When called with these arguments, the method `fetch` returns an iterator object. We can get the next page of results from this iterator.

Example:

In this example, we are asking the Datastore client to fetch the first page of results with a page size of 3.

- We call `fetch` with the value of `limit` set to 3 and value of `offset` set to 0.
- The query returns an iterator.
- We call Python's next() function to get a page of data from the iterator.

```
query = client.query(kind=LODGINGS)
lodging_query = client.query(kind=LODGINGS)
l_iterator = lodging_query.fetch(limit=3, offset=0)
pages = l_iterator.pages
results = list(next(pages))
```

Note that the iterator object returned by the `fetch` method has a property `next_page_token`. If the value of this property is `None`, then the query will not return any more pages of results.

Reference

To learn more about this topic, including how to use cursor-based pagination, see the following reference

https://cloud.google.com/datastore/docs/concepts/queries#cursors_limits_and_offsets