Everything Search

The goal of this project is to design a prototype of entity-aware web indexing system to search by and for keywords, entities and documents.

System Design

The design of the system is divided into two major components, which are explained below.

Initialization

This component deals with converting the flat file to an optimised binary indexed format file based on ES guidelines. This components has two subparts which are as follows:

Data Input

This is first part of initializing the system where an admin populates the data into the system. The admin, who acts as a system engineer, provides two inputs - Schema, a set of base relations and functional dependencies (FDs), and Base Table Streams, a stream of input from a flat file of user's choice. Thus, the input data stream is stored on disk, which is sorted based on given column order and separated into multiple files for each column store and row store.

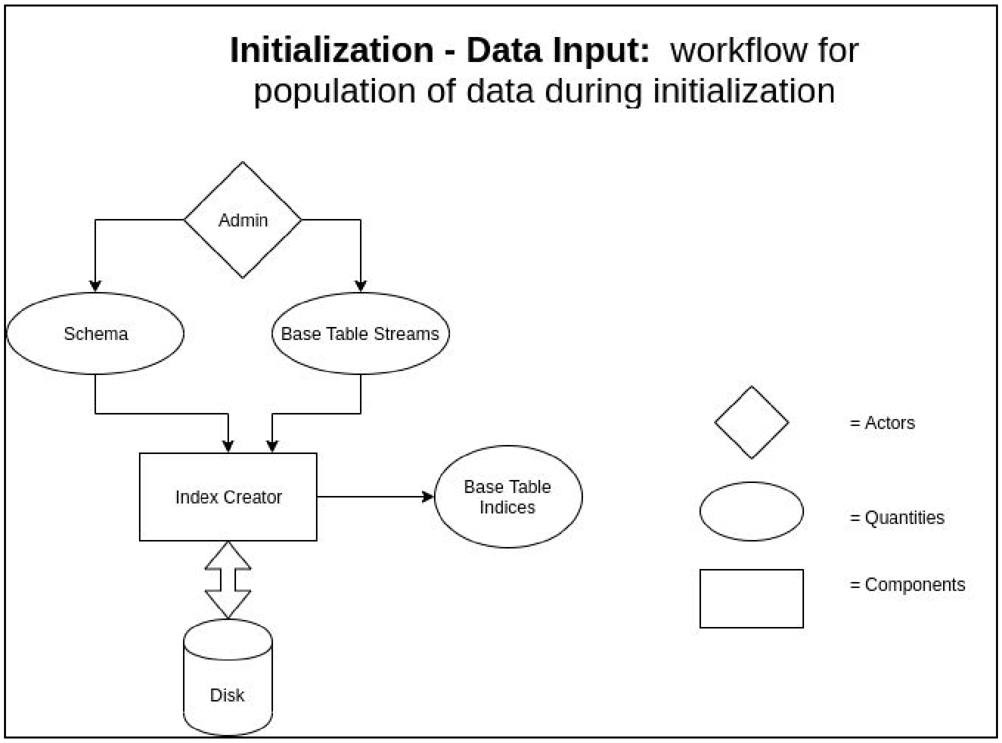

Figure 1: Data Input schema

For instance, if an admin gives the schema, S=({KD}, {KD.text, KD.url -> KD.tf}), then the schema just has a single base relation KD with columns {text, url, tf} and single functional dependency.

Indexing

In this step, indexing is performed in order to efficiently answer the query workload and creating indices. The admin inputs the query and the system determines the query workload, and executes an instance of the query for given parameters and query plan. Then the system parses the metadata from the string representation of query given by admin and adds implied conditions based on FDs and logical implication.

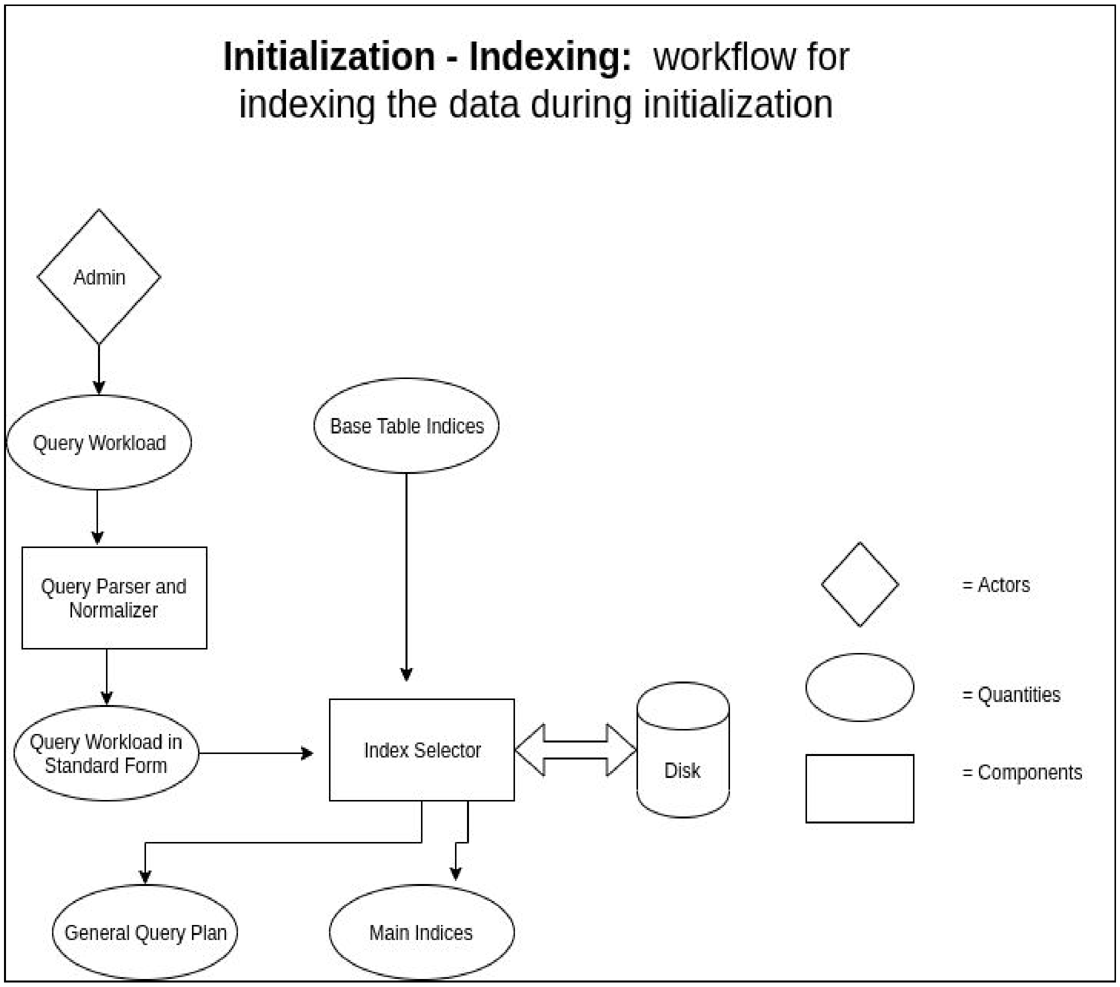
Initialization - Indexing: workflow for indexing the data during initialization
Admin
Query Workload
Base Table Indices
Query Parser and Normalizer
Query Workload in Standard Form
Index Selector
Disk
General Query Plan
Main Indices
= Actors
= Quantities
= Components

Figure 2: Indexing schema

Using the index selector, we map indexes and query to each other and then search space of all solutions is explored using A*-search and Branch-and-bound heuristics. Finally, indexes are created as given by the algorithm and general queral plan cost is estimated along with it. Also, we create mapping between indexes and query instances. The query execution engine uses late materialization and handles cross products in a space efficient manner. The absolute working of Index Selector is still in progress.

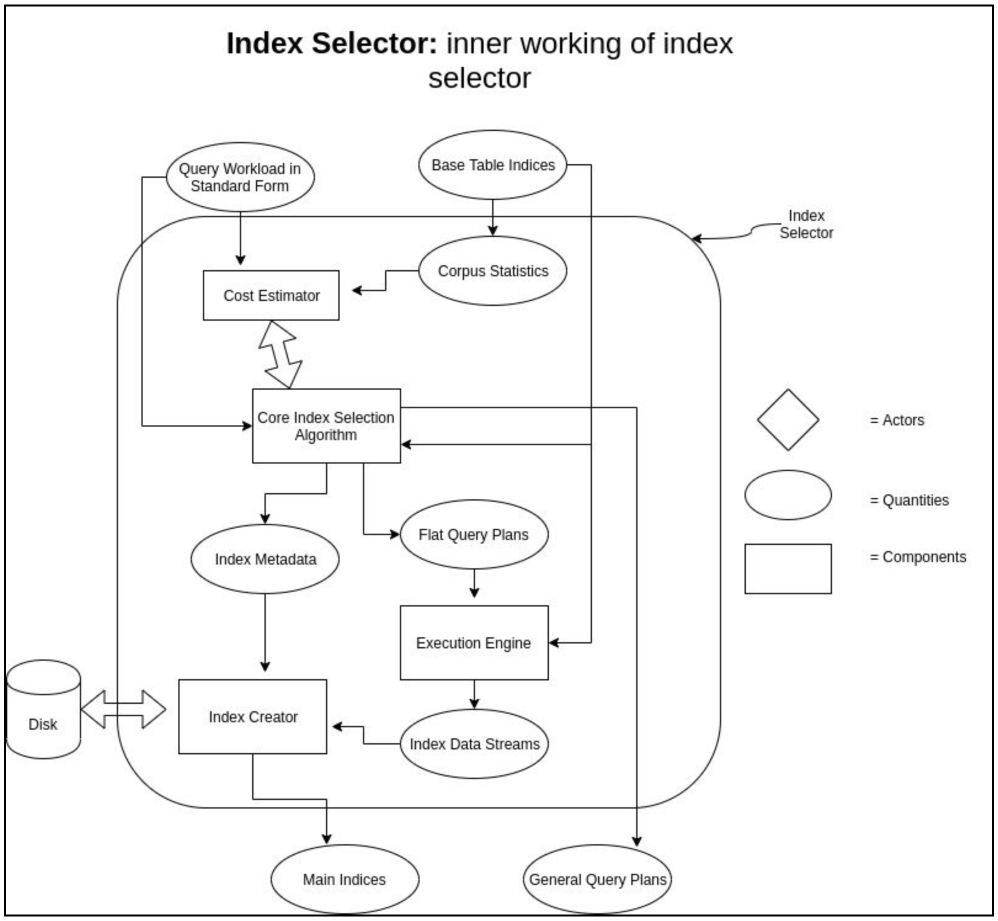

Figure 3: Index Selector schema

An illustrative example, continuing from the previous component, for Indexing is as follows:

- Admin gives the query workload to the system. For this example, there is just one query in the workload given as follows:

```
SELECT 1.url, 1.tf, 2.tf, 3.tf
FROM KD as 1, KD as 2, KD as 3
WHERE (1.text=x) AND (2.text=x)
                 AND (3.text=x)
                 AND (1.url=2.url)
                 AND (2.url=3.url)
```

  This query template has three parameters corresponding to three keywords.

- Query parser and normalizer now produce following query template in standard form (Labels:L, BaseRelationMapping:B, SelectionProjectionConditions:F, ProjectedColumns:X; To know more, please read Everything Search paper):

```
(L={1,2,3}, B={1:KD, 2:KD, 3:KD}, F={(1.text=x), (2.text=x), (3.text=x), (1.url=2.url),
(2.url=3.url), (1.url=3.url)}, X={1.url, 1.tf, 2.tf, 3.tf})
```

  Note that the normalized query has join condition (1.url=3.url) which was absent from the original query.

- Index Selector then figures out that the inverted index should be stored and how it should be used to answer the query. Note that we won't go into the details on index selector in this example.

  Inverted Index = (L={1}, B={1:KD}, F={(1.text=x)}, O=[1.text, 1.url, t.tf])

  Note that instead of unordered set of columns on which output is projected, index also specifies the column order in which the tuples are stored.

  General Query Plan = [{1->1}, {1->2}, {1->3}]

  General Query plan has three index mentions of inverted index in the given query. Query plan is executed in 3 iterations where in first iteration the first index mention is retrieved. In second execution the second index mention is retrieved and merged with the old result to obtain the new results. Finally the third index mention is also retrieved to obtain the final result.

Operation

This component deals with answering the query instances at runtime after the indexing is done.

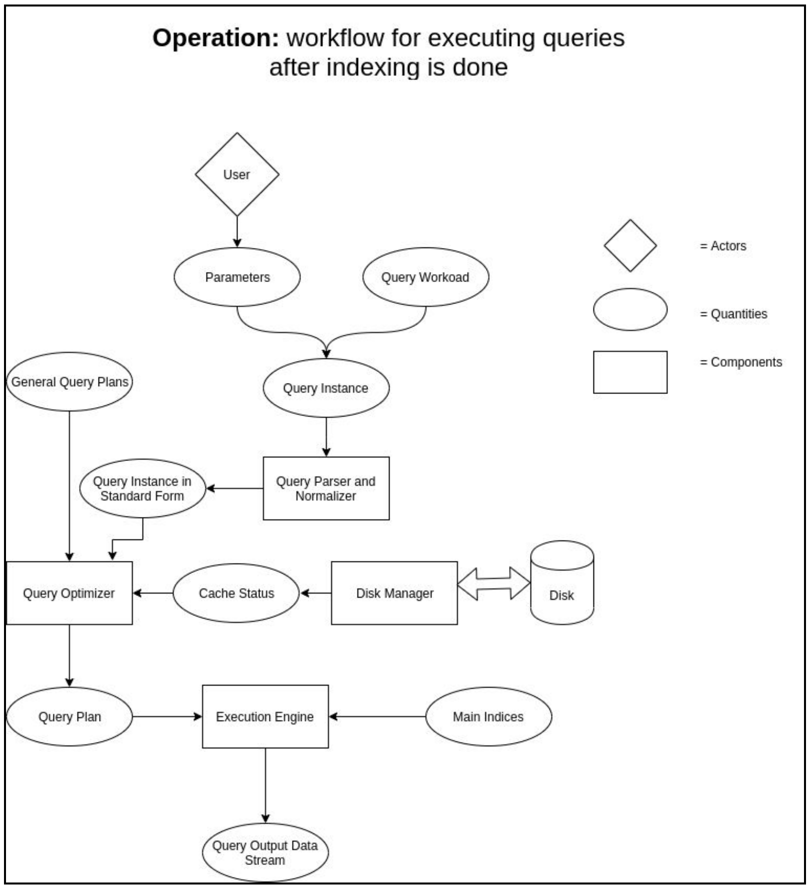

Figure 4: Operation schema

- User gives parameters that defines a query instance. For example:- parameters "I think therefore" specifies the query instance:

  ```
  SELECT 1.url, 1.tf, 2.tf, 3.tf
  FROM KD as 1, KD as 2, KD as 3
  WHERE (1.text="I") AND (2.text="think") AND (3.text="Therefore") AND (1.url=2.url)
  AND (2.url=3.url)
  ```

  Query parser gives a standardized representation of this query instance.

- Query optimizer identifies the appropriate general query plan (built in Initialisation step) for this query instance too.
- Finally, execution engine executes the query with the help of main indices to obtain the final result DataStream.