# Technological considerations

Database systems are not necessarily complicated: key-value stores; DBM (1979) For commodity, query languages were introduced. SQL was introduced in the 80’s, remains the most important one; all major databases use it. An interesting one is SQLite: retains the simplicity of DBM but has an SQL layer.

At the beginning of this century, XML was all the rage. There has been research into XML information retrieval: Indri. This a dead field today. Some XML-based systems are still developed. One of them is eXistdb, found a niche in the digital humanities. Also MarkLogic.

For instance, it has an R\*Tree, a data structure for geospatial search. SQLite was designed to be extended, which makes it really convenient to work with. For serious work, however, it’s necessary to program in a low-level language, which is difficult.

# Functionalities

There are two kinds of queries : traditional database queries that use basic operators, and full-text search queries.

\* Traditional database query : find all articles published before 1967

\* Traditional database query : find all articles that bear the tag « cooking »

\* Full-text search query : find all Tolstoy’s books

I won’t talk about the first type of query, because there isn’t much to discuss. This is what databases are designed to do.

Full-text search is much more tricky to get right. There are many different possibilities, which have different merits, thus there are many decisions to take. The system can always be improved. It’s also much more complicated in terms of implementation.

## Regular expressions

Functionality-wise, I’ve understood that you want grep-like functionalities, regular expression, or, less specifically, substring search. Many software libraries provide this kind of functionality, but one of them has an extra feature that’s interesting, namely approximate regex matching. It’s a very rare feature, because it’s really tricky to implement. I think it would be great if we had this. Example :

cd ~/programs/dharma/repos/electronic-texts.hid/muktabodha  
agrep -1 "(mahā)?mantreśvara" \*.txt

This is particularly useful for corrupt texts.

cd ~/programs/dharma  
sqlite3 -column ngram.sqlite "select file, verse, normalized, coeff from padas join jaccard on id = id2 where id1 = 54 order by coeff desc"

## Query expansion

Now we can enhance the engine to account for known variations. Arlo talked about it. For instance, when looking for dharma, we also want to find dharmma, thus when the user enters « dharma », we look instead for the regex dharmm?a. Doing this kind of thing is the bread and butter of full-text search : we increase recall by expanding a query to a more general one.

See <https://michaelmeyer.fr/sanskrit/meta/terms/gandharva>

I did stuff like that for external sandhi in Sanskrit in a tool I wrote a few years ago. The idea is to expand the query string to all the forms it can take in various euphonic contexts. Thus :

mantras → mantras, mantraḥ, mantro, mantraś

Of course, this generates false positives, but this isn’t much of an annoyance in practice. The alternative would be to have a real parser. But parsers have their problems too.

To do this kind of thing for the various langages of the project, I will need assistance from people who know the target language. It might be possible to have a generic system that works reasonably well across langages. I can’t tell for now, I need more information.

## Inflected forms

A particularly useful application of the query expansion mechanism is searching for inflected forms. For instance, the user supplies a stem, the system retrieves all the inflected forms by looking at a lexical database (or with an algorithm), and finally it searches them all at once. For instance:

faire → (faire|fais|fait|faisons...)

There are lexical databases for modern occidental languages and for Sanskrit, I don’t know about the other languages.

## Matching behaviour

Besides the things related to query expansion, at some point we’ll also have to decide how we want regex matching to work. Firstly, we must decide how to interpret the text at the character-level. It’s necessary to determine phoneme boundaries and to use an internal encoding for the text, like :

dhr̥tarāṣṭra → [35, 7, 32, 1, 43, 2, 47, 27, 43, 1]

Of course, this changes between languages, so I will need linguistic assistance. In the same vein, it’s necessary to determine which characters are not important for matching and should be ignored. This concerns hyphens, for instance : the query « mantraśabda » should match both « mantraśabda » and « mantra-śabda ».

Furthermore, it’s necessary to decide what text unit a regex should match. Traditionally, grep is line-based : it matches lines separately, it doesn’t try to find matches that cross line boundaries. In our case, it’s not useful to do line-based matching, but I’m not sure yet which boundaries we should take into account (verse ? paragraph?). What annoys me is that if we want to be able to support <choice>, it’s necessary, for practical reasons, to do string matching at the level of phoneme clusters. I don’t really like this solution.