# Timeline

[※ DIAPO ※] 1 year to set up a reliable validation and HTML conversion process. Now is not the time to change the schema, except for addressing problems, of course. I need time to catch up with the current encoding. An important issue that will need to be adressed is that the current encoding is sometimes too permissive and too ambiguous for a computer to process. Given the complexity of the encoding, addressing this will take a lot of time.

For the database itself, It’ll easily take two years. We’re in a situation where a lot of low-level, non-trivial programming is necessary. It’s not a walk in the park. At this level, doing approximative work, doing something that is kinda okay but not completely, is not an option. You need to do things correctly from the very beginning, because there are no facilities to help you find busgs later on.

I’m telling you this to point out that we’re on a relatively tight schedule and that it’s necessary to set priorities.

# Technological considerations

[※ DIAPO ※] 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: it retains the simplicity of DBM, but adds an SQL layer. In fact, it was initially an extension of DBM.

Later on, at the beginning of this century, XML was all the rage. Mostly due to the Internet and the need for data interchange. XML was used for pretty much everything. There has been research into XML information retrieval: Indri. But no production IR system. This research field is dead today. A few XML-based systems are still developed. One of them is eXistdb. It found a niche in the digital humanities.

It doesn’t particularly impresss me, SQLite is better. The main reason eXistdb is used, I believe, is that XML is pretty much the only thing taught in digital humanities curricula. This is a local phenomenon, however. People with a computer science degree, webmasters, etc., all know SQL. It’s the lingua franca of databases, everybody speaks it. XQuery, the query language used by eXistdb, is less widespread. Its first version has several implementations, but the later ones have a single one.

I think SQLite is a better choice. It’s also much more popular and very well documented.  
[※ DIAPO ※].

[※ DIAPO ※] The two systems don’t have the same purpose. eXistdb has a server, a web interface, an administration interface, a XML editor, a package manager.

# Non-subjects

[※ DIAPO ※] I won’t talk about data storage and stuff related to updating the database and backing it up. I’m not sure yet what we should do. I suggest we keep using git for updating things. People are used to it already, and I see no compelling reason to set up a new system. Well, there’s one: git doesn’t have transaction semantics. It can easily corrupt itself. It’s also hard to backup.

I won’t talk about the search interface, its appearance, how to website will look, because it isn’t yet a priority. But I keep this concern in mind. I’ll write things in such a way that changing the interface is straightforward. Don’t hesitate to tell me if I’m not pragmatic enough.

I also won’t talk about traditional database queries. [※ DIAPO ※] Here are a two examples of what I mean.

There isn’t much to discuss, besides of course which fields should be made searchable. An important aspect of this kind of query is that they involve binary logic. Either a document matches or it doesn’t, there is no in-between. Databases are specifically designed to answer this kind of problem. There is not much to think about, the functionalities are already present in the database engine. The main difficulty is to translate a user query into a database query. Generally speaking, a lot of work is needed, but not much thinking.

[※ DIAPO ※] There are other types of queries that are much more complicated to answer, though. Everything that is related to full-text search, the main functionality we’re interested in. This is a completely different field of research. Example:

Full-text search is tricky to get right. Answers are not right or wrong: they are more or less relevant. There are often many different ways to answer a query, which have different merits. The system can always be improved, it’s a never-ending process.s

# Regular expressions

[※ DIAPO ※] 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, TRE, has an extra feature that’s interesting, namely approximate regex matching. It’s a very rare feature. I think it would be great if we had this. Example :

You also find approximate search features elsewhere. But not for substring matching. And not together with regular expressions. Example: archive.org.

[※ DIAPO ※] There are other ways to look for parallels. See my verses parallels.

# Query expansion

[※ DIAPO ※] Now we can enhance the engine to account for spelling variations, inflection, etc. This 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 search 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. 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

[※ DIAPO ※] 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.

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

# Matching behaviour

[※ DIAPO ※] 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 :

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.