Synchronization with remote databases

Context and Problem Statement

Synchronize the data in a library to a remote database, while handling conflicts and supporting offline-first paradigm.

Decision Drivers

- Updates from the remote should be pulled in
- No updates should get lost
- Easy to implement
- Easy to maintain

Considered Options

- “Optimistic offline lock” with hashes for local file support
- Algorithm based on “optimistic offline lock”
- Use CRDTs

Decision Outcome

Chosen option: “‘Optimistic offline lock’ with hashes for local file support”, because simplest option to resolves all forces.

Pros and Cons of the Options

“Optimistic offline lock” with hashes for local file support

The Optimistic Offline Lock is good for synchronizing clients with a server when there is no other modification of data on client side. However, users might modify the `.bib` file external of JabRef. They might also open an existing `.bib` file and synchronize that. Thus, there are additions needed to handle the local synchronization.

Moreover, the optimistic offline lock does not say how a set of data is synchronized.

Both shortcomings are resolved by our algotihm. This algorithm is described at Remote JabDrive storage.

Algorithm based on “optimistic offline lock”

Optimistic Offline Lock is a well-established technique to prevent conflicts in concurrent business transactions. It assumes that the chance of conflict is low. Implementation details are found at https://www.baeldung.com/cs/offline-concurrency-control.

This is implemented for the SQL database synchronization, which is described at Remote SQL Storage.

- Good, because this algorithm is already in place since 2016 for JabRef synchronizing with a PostgreSQL backend and a MySQL backend.
- Bad, because it assumes the client to be online 100% and does not have handlings of cases where the client disconnects and alters data in other ways.

Use CRDTs

See https://automerge.org/blog/automerge-2/ for details.

- Bad, because one needs to locally store a lot more metadata (e.g. for operational CRDTs you essentially need to have the full history of all edits). So you would need another file next to the bib file to store these.
- Bad, because CRDTs are mainly used when you need low latency and high frequency of edits (e.g. multi-user chat or text editing). Not really something we care about.
