The Postal Service

The Postal Service is a system to verify address uniqueness. It must store a database of known addresses, worldwide. Users will be able to enter new addresses, as well as search for addresses using underspecified inputs. Address formats vary by country and region.
The system must understand these varying formats, dynamically updating both form entry as well as search fields. Personal information is neither stored with, nor searched for within, system addresses.

As an example, Hans is specifying an address in Belgium. He enters `Rue du Cornet 6 B-4800 VERVIERS BELGIUM` into the system. Judith, from England, is searching to verify the address she wrote after a phone call. She enters the address in the UK format, `6 Rue du Cornet Belgium`. The system will find `Rue du Cornet 6 B-4800 VERVIERS BELGIUM`, as well as `Rue du Cornet 6 B-1040 ETTERBEEK BELGIUM`.

The Postal Service System requires four distinct components. Metadata must be globally available describing the format of countries' postal systems. Once defined, this metadata can be distributed globally through a CDN, embedded directly in binaries, or through other appropriate means. A dynamic form frontend will use the metadata to reactively create form elements, apply validation, and format requests to the API. This dynamic form can be used both for data entry as well as partial data search. The API server will accept create and search requests. It will use the metadata to normalize these requests to a canonical internal format, and then dispatch them appropriately to a database. A database system will store address fields in a sparse, normalized manner. The database will have indices on each column.

- Alternative: Shard database on country (192 databases?), and only have the normalized columns per database.
- Alternative: Store denormalized addresses, and maintain a custom set of indices over fields indicated by the metadata.

This system can grow in several directions, with each part dependent primarily on the metadata format. The primary performance constraint will be in accessing the database indices. Choosing a normalized format for searching these indices will move much of that potential time to a smaller, fixed-cost reshuffling of data in the API. As the system grows, we assume that reads will outweigh writes one hundred to one (100:1). This nicely allows a secondary sharding scheme of read-only copies of the database.