Title

SRFI XYZ: Ordered Key Value Store (wip)

Author

Amirouche Boubekki

Status

???

Abstract

This library describe an interface for ordered key-value store that is suitable to implement a storage engine for the generic tuple store SRFI. It maps cleanly with existing ordered key-value databases that may or may not provide transactions.

Rationale

Ordered key-value stores offers a powerful set of primitives to build databases abstractions (also sometimes called layers). The generic tuple store SRFI is an example of such abstraction. A standard interface for such databases will allow more people to experiment with other abstractions.

This SRFI does not overlap with existing SRFIs and complement the generic tuple store SRFI.

Specification

Database and Transaction

Database is a mapping that may or may not support transactions where keys are lexicographically ordered bytevectors and values are bytevectors.

The following specification defines two disjoint types:

- `database` is a handle over the database
- `transaction` is a handle over a currently running transaction

`(make . args)`

Returns a database object. `ARGS` is implementation dependant and allows to configure the underlying database connection. It might be the path to the directory or file where the database is stored or the location of a configuration file or the host and port in case the database is accessed over the network.

`(close database . args)`

Close `DATABASE`. `ARGS` are implementation dependant.

`(begin! database . args)`

Start a transaction and returns a transaction object. `ARGS` allow to configure the transaction and are implementation dependant.

`(commit! transaction . args)`

Commit the transaction. `ARGS` allow to configure the transaction and are implementation dependant.

`(rollback! transaction . args)`

Rollback the transaction. `ARGS` allow to configure the transaction and are implementation dependant.

`(ref transaction key)`

Returns the bytevector associated with `KEY` bytevector using `TRANSACTION`. If there is no such key returns `#f`.

`(set! transaction key value)`

Associates `KEY` bytevector to `VALUE` bytevector using `TRANSACTION`.

`(rm! transaction key)`

Delete the pair associated with `KEY` bytevector using `TRANSACTION`.

`(range transaction prefix)`

Returns a srfi 158 generator of key-value pairs where keys starts with `PREFIX` bytevector. The stream must lexicographically ordered. `PREFIX` can be the empty bytevector, in that case the all the pairs are returned in a generator.

Lexicographic Packing

This section defines two procedures `(pack . items)` and `(unpack bytevector)` which allows to translate back-and-forth scheme objects to bytevectors in a way that preserves lexicographic ordering. The ordering between types is defined as follow:

1. `*null*`
2. bytevector
3. string
4. exact number
5. float
6. double
7. boolean

The implementation might support symbols, lists and vectors at the risk of being incompatible with existing databases.

`*null*` is a singleton that must be provided by the implementation.

Note: This is different from srfi 128 because a) it is not possible to pack inexact and exact numbers using the same algorithm while preserving a total order. b) it allows to be compatible with existing FoundationDB packing function.

`(pack . items)`

Returns a bytevector that preserve lexicographic ordering as described above. The accepted object types is implementation dependant, see the above note.

`(unpack bytevector)`

Returns values packed in `BYTEVECTOR`. It is an error, if `BYTEVECTOR` encode object not supported by the implementation.

Implementation

The sample implementation rely on scheme mapping (srfi 146) and srfi 158.

Acknowledgements

Credits goes to WiredTiger and FoundationDB authors for their work on their respective database engines. The author would like to thank Arthur A. Gleckler, Marc Nieper-Wißkirchen for getting together SRFI 146 and Shiro Kawai, John Cowan, Thomas Gilray for their work on SRFI 158.

Copyright