SeaJUG Clojure talk: Adding Units/Day column to meter readings

stand **17d**

Here is a summary of my clojure talk at SeaJUG on April 19th.

The demo was to add a `units-per-day` field to a list of utility meter readings. A reading is a clojure hashmap that looks like this:

```
{:idreadings 893,
  :rtype "gas",
  :rdttm #inst "2011-01-26T19:45:00.000000000-00:00",
  :rvalue 1095,
  :rcomment ""}
```

I am showing the model namespace which loads some functions from an external date/time library and my database functions.

```
(ns com.standyck.meetrz.model
  (:require [com.standyck.meetrz.model.db :as db]
            [clj-time.core :as dt]
            [clj-time.coerce :as coerce]))
```

The `get-all-readings` function returns a sequence of these reading hashmaps for a given `rtype` value of `"gas"`, `"electric"` or `"water"`.

```
(defn get-all-readings [rtype]
  (db/select-readings-of-type (name rtype)))
```

I next create a hashmap that maps a field from a reading to a function that takes a pair of values and calculates the difference between them. I only care about the `:rvalue` and `:rdttm` fields. The `:rvalue` field calculates a simple arithmetic difference. The `:dttm` field requires using the date/time library (Joda underneath a clojure coating).

```
(def diff-fn
  {:rvalue (fn [pair] (apply - pair))
   :rdttm (fn [pair] (dt/in-minutes (apply dt/interval
                                           (map #(coerce/to-date-time %)
                                                (reverse pair))))) })
```

The `sequence-of-deltas` function takes a `reading` and a `field` and returns a sequence of

differences between each reading value or date/time and it's prior reading field value. The `partition` function does the heavy lifting here. It takes a sequence and returns a sequence of value pairs with an offset of 1. So for example:

```
user> (partition 2 1 [0 1 2 3 4 5])
((0 1) (1 2) (2 3) (3 4) (4 5))
```

```
(defn sequence-of-deltas [reading field]
  (map (field diff-fn)
       (partition 2 1 (map field reading))))
```

After creating a sequence of such pairs, I map the parameterized difference function for the input `field` over each pair to produce a sequence of deltas.

The `units-per-day-sequence` function takes a sequence of `readings` and calculates the delta value/delta time in units/day. Here we take advantage of the fact that `map` takes multiple sequences. Like this:

```
user> (map + [1 2 3] [10 20 30])
(11 22 33)
```

The function in the map divides the `value` by the number of minutes and converts it from minutes to days rounding to two decimal places.

```
(defn units-per-day-sequence [readings]
  (map (fn [value minutes]
         (Float/parseFloat (format "%.2f" (* 60.0 24 (/ value minutes)))))
       (sequence-of-deltas readings :rvalue)
       (sequence-of-deltas readings :rdttm)))
```

Finally, the `introduce-units-per-day` function takes a sequence of readings and returns a sequence of readings with a `:units-per-day` field added to it.

```
(defn introduce-units-per-day [readings]
  (map
   (fn [reading units-per-day]
     (assoc reading :units-per-day units-per-day))
   readings
   (units-per-day-sequence readings)))
```

Again, we map across two sequences. The first are the readings and the second is the sequence of units/day returned by the `units-per-day` function. We use the `assoc` function to add the `:units-per-day` field and value to the reading. After calling this function with a sequence of readings a typical returned reading will look like this:

```
{:idreadings 893,
  :rtype "gas",
  :rdttm #inst "2011-01-26T19:45:00.000000000-00:00",
  :rvalue 1095,
  :rcomment ""
  :units-per-day 2.06}
```

Note the new field in the final pair.

If you have any questions, please leave a reply below.