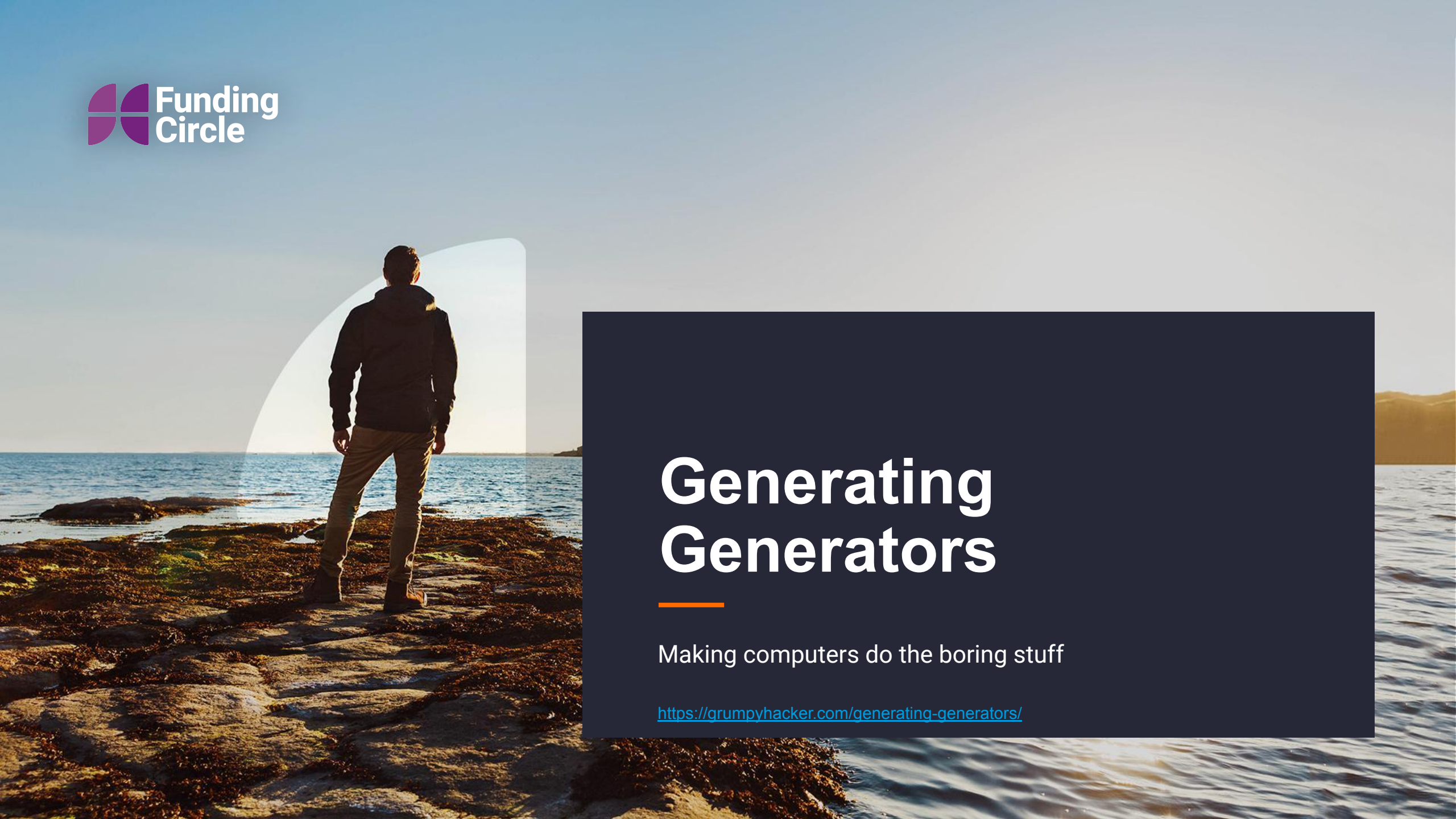


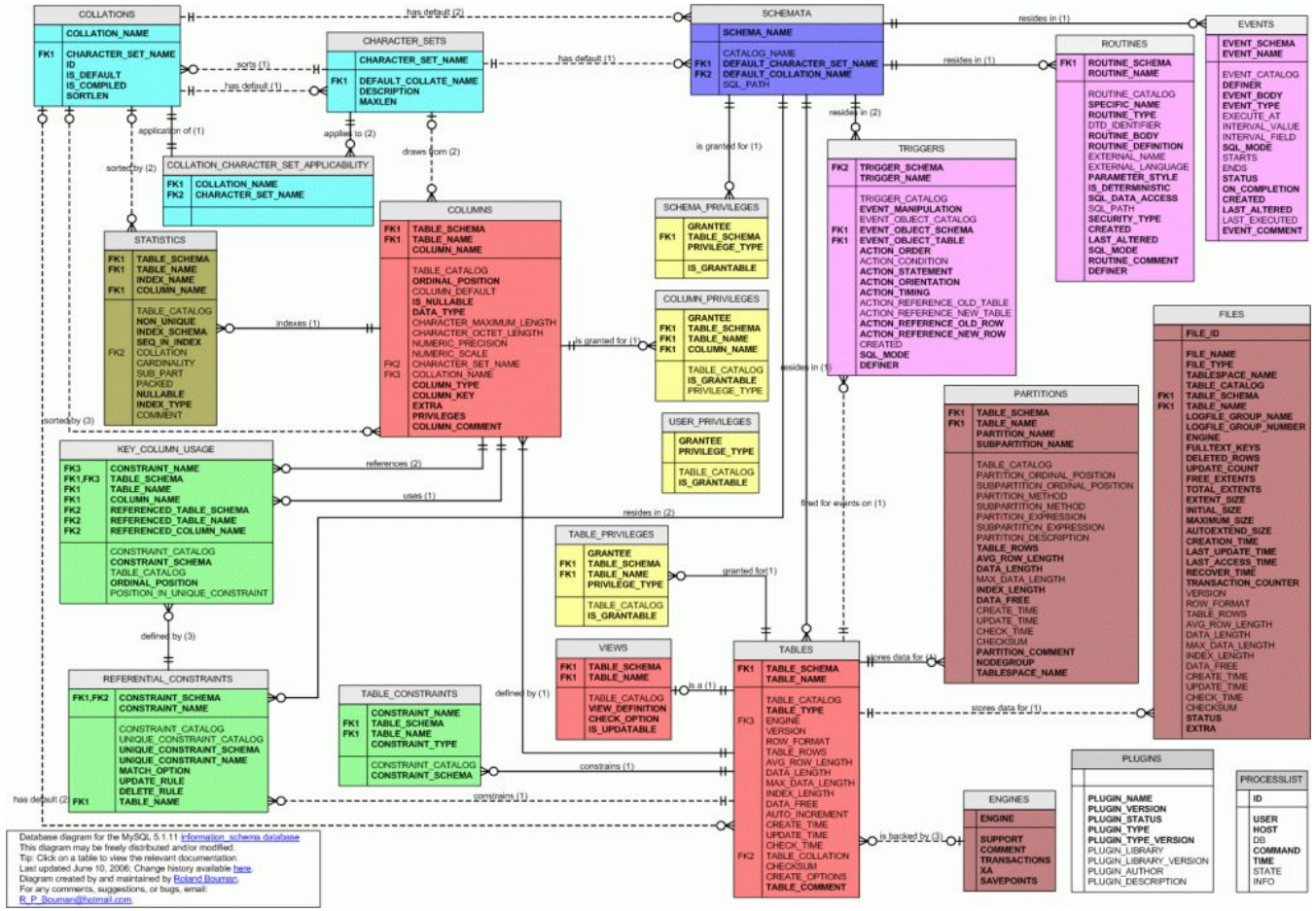
Generating Generators

Making computers do the boring stuff

https://grumpyhacker.com/generating-generators/

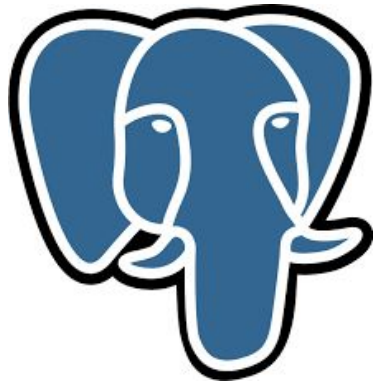
Intro

Information Schema



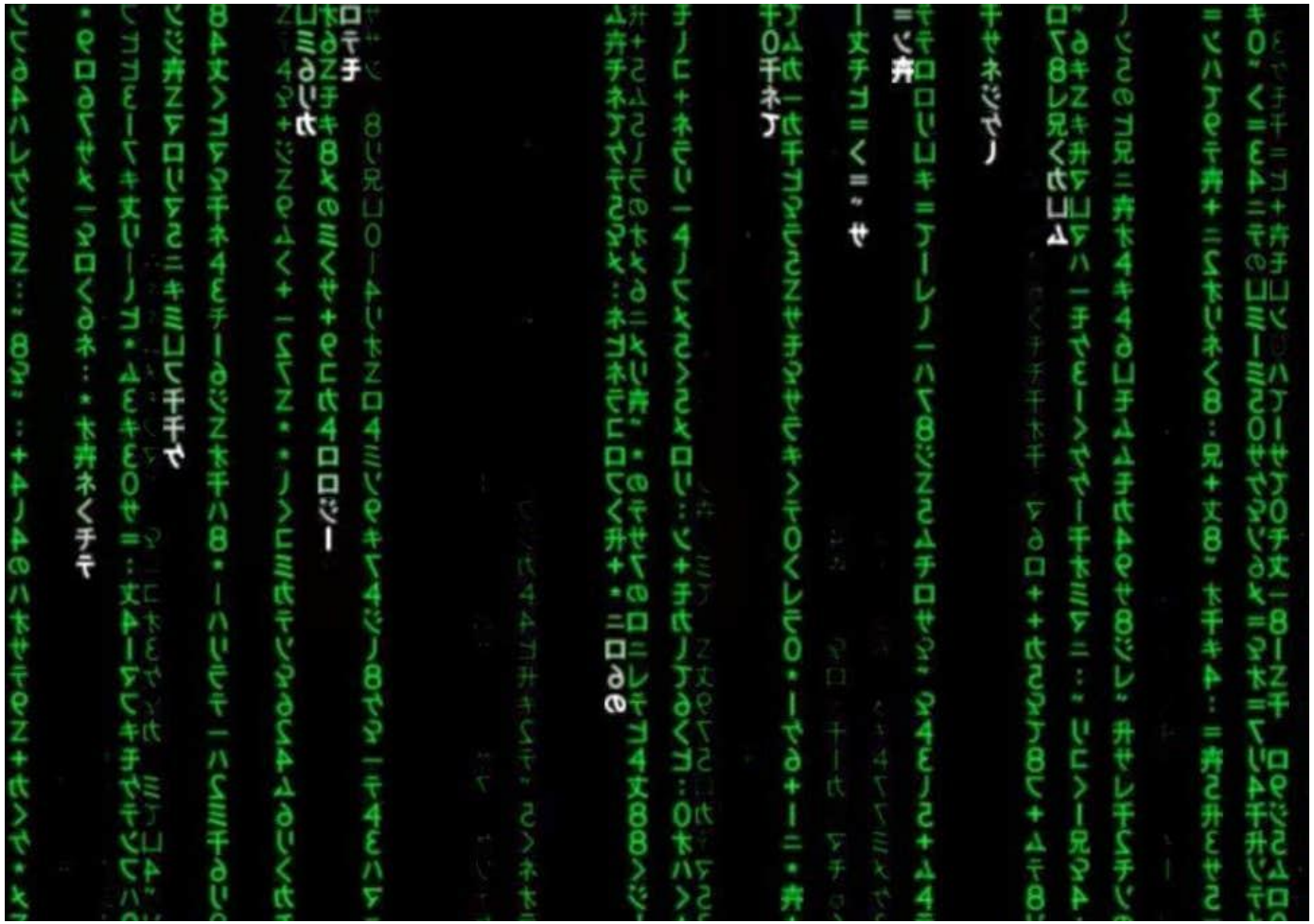
https://www.jorgeoyhenard.com/modelo-er-del-information-schema-de-mysql-51/906/

ANSI Standard



Tabular Data Structures

Machine Readable



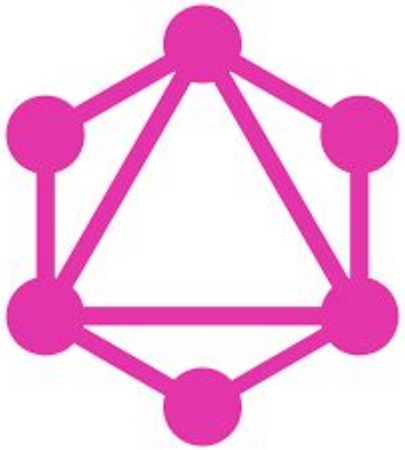
Column/Type Information

```
mysql> describe information_schema.columns;
+--------------------------+---------------------+------+-----+---------+-------+
| Field                    | Type                | Null | Key | Default | Extra |
+--------------------------+---------------------+------+-----+---------+-------+
| TABLE_CATALOG            | varchar(512)        | NO   |     |         |       |
| TABLE_SCHEMA             | varchar(64)         | NO   |     |         |       |
| TABLE_NAME               | varchar(64)         | NO   |     |         |       |
| COLUMN_NAME              | varchar(64)         | NO   |     |         |       |
| ORDINAL_POSITION         | bigint(21) unsigned | NO   |     | 0       |       |
| COLUMN_DEFAULT           | longtext            | YES  |     | NULL    |       |
| IS_NULLABLE              | varchar(3)          | NO   |     |         |       |
| DATA_TYPE                | varchar(64)         | NO   |     |         |       |
| CHARACTER_MAXIMUM_LENGTH | bigint(21) unsigned | YES  |     | NULL    |       |
| CHARACTER_OCTET_LENGTH   | bigint(21) unsigned | YES  |     | NULL    |       |
| NUMERIC_PRECISION        | bigint(21) unsigned | YES  |     | NULL    |       |
| NUMERIC_SCALE            | bigint(21) unsigned | YES  |     | NULL    |       |
| DATETIME_PRECISION       | bigint(21) unsigned | YES  |     | NULL    |       |
| CHARACTER_SET_NAME       | varchar(32)         | YES  |     | NULL    |       |
| COLLATION_NAME           | varchar(32)         | YES  |     | NULL    |       |
| COLUMN_TYPE              | longtext            | NO   |     | NULL    |       |
| COLUMN_KEY               | varchar(3)          | NO   |     |         |       |
| EXTRA                    | varchar(30)         | NO   |     |         |       |
| PRIVILEGES               | varchar(80)         | NO   |     |         |       |
| COLUMN_COMMENT           | varchar(1024)       | NO   |     |         |       |
| GENERATION_EXPRESSION    | longtext            | NO   |     | NULL    |       |
+--------------------------+---------------------+------+-----+---------+-------+
21 rows in set (0.00 sec)
```

Relationship Information

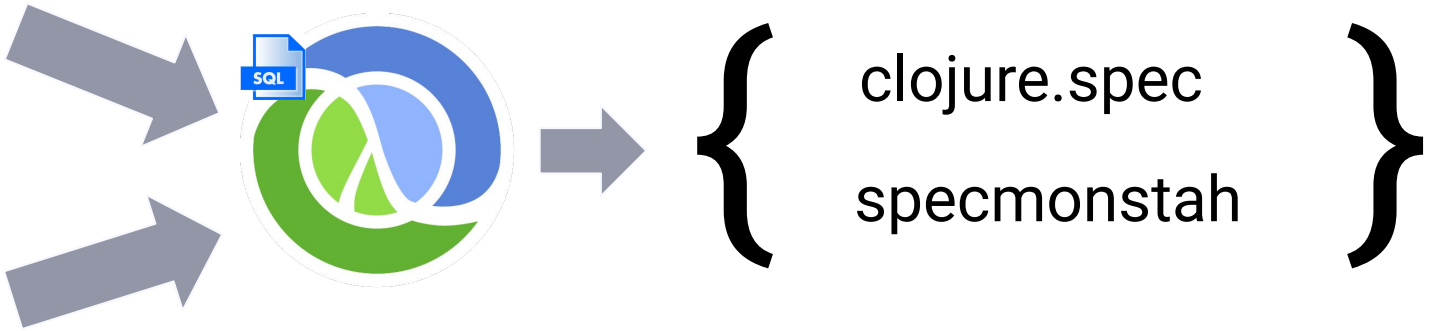
```
mysql> describe information_schema.key_column_usage;
+------------------------------+-------------+------+-----+---------+-------+
| Field                        | Type        | Null | Key | Default | Extra |
+------------------------------+-------------+------+-----+---------+-------+
| CONSTRAINT_CATALOG           | varchar(512) | NO   |     |         |       |
| CONSTRAINT_SCHEMA            | varchar(64)  | NO   |     |         |       |
| CONSTRAINT_NAME              | varchar(64)  | NO   |     |         |       |
| TABLE_CATALOG                | varchar(512) | NO   |     |         |       |
| TABLE_SCHEMA                 | varchar(64)  | NO   |     |         |       |
| TABLE_NAME                   | varchar(64)  | NO   |     |         |       |
| COLUMN_NAME                  | varchar(64)  | NO   |     |         |       |
| ORDINAL_POSITION             | bigint(10)   | NO   |     | 0       |       |
| POSITION_IN_UNIQUE_CONSTRAINT | bigint(10)   | YES  |     | NULL    |       |
| REFERENCED_TABLE_SCHEMA      | varchar(64)  | YES  |     | NULL    |       |
| REFERENCED_TABLE_NAME        | varchar(64)  | YES  |     | NULL    |       |
| REFERENCED_COLUMN_NAME       | varchar(64)  | YES  |     | NULL    |       |
+------------------------------+-------------+------+-----+---------+-------+
12 rows in set (0.01 sec)
```

Other info schemas



Programming with Metadata

Column Metadata



Relationship Metadata

Spec Generator

```
(clojure.spec.alpha/def :celm.columns.addresses/addressable-id :ce-data-aggregator-tool.streams.info-schema/banded-id)
(clojure.spec.alpha/def :celm.columns.addresses/addressable-type #{"person" "company_loan_data"})
(clojure.spec.alpha/def :celm.columns.addresses/city (clojure.spec.alpha/nilable (info-specs/string-up-to 255)))
(clojure.spec.alpha/def :celm.columns.addresses/company (clojure.spec.alpha/nilable (info-
(clojure.spec.alpha/def :celm.columns.addresses/country-id :ce-data-aggregator-tool.stream
(clojure.spec.alpha/def :celm.columns.addresses/created-at :ce-data-aggregator-tool.stream
(clojure.spec.alpha/def :celm.columns.addresses/debezium-manual-update
  (clojure.spec.alpha/nilable :ce-data-aggregator-tool.streams.info-schema/datetime))
```

```
(clojure.spec.alpha/def :celm.tables/addresses
 (clojure.spec.alpha/keys
   :req-un
   [:celm.columns.addresses/addressable-id
    :celm.columns.addresses/addressable-type
    :celm.columns.addresses/city
    :celm.columns.addresses/company
    :celm.columns.addresses/country-id
    :celm.columns.addresses/created-at
    :celm.columns.addresses/debezium-manual-update
    :celm.columns.addresses/id
    :celm.columns.addresses/name
    :celm.columns.addresses/phone-number
    :celm.columns.addresses/postal-code
    :celm.columns.addresses/province
    :celm.columns.addresses/resident-since
    :celm.columns.addresses/street1
    :celm.columns.addresses/street2
    :celm.columns.addresses/street3
    :celm.columns.addresses/street-number
    :celm.columns.addresses/updated-at]))
```

Generate the specs

```
lein from-info-schema gen-specs > src/ce_data_aggregator
```

Column Query

```
(def +column-query+
  "Query to extract column meta-data from the mysql info schema"
  "select c.table_name
        , c.column_name
        , case when c.is_nullable = 'YES' then true else false end as is_nullable
        , c.data_type
        , c.character_maximum_length
        , c.numeric_precision
        , c.numeric_scale
        , c.column_key
     from information_schema.columns c
    where c.table_schema = ? and c.table_name in (<table-list>)
 order by 1, 2")
```

Integer Types -> Specs

```
(s/def ::tinyint (s/int-in -128 127))
(s/def ::smallint (s/int-in -32768 32767))
(s/def ::mediumint (s/int-in -8388608 8388607))
(s/def ::int (s/int-in 1 2147483647))
```

https://dev.mysql.com/doc/refman/8.0/en/integer-types.html

Date Types -> Specs

```
(s/def ::date (s/with-gen #(instance? java.sql.Date %)
                #(gen/fmap (fn [x]
                             (Date/valueOf (time/minus (time/local-date) (time/days x))))
                           (s/gen (s/int-in 0 30)))))
(s/def ::datetime (s/with-gen #(instance? java.sql.Timestamp %)
                    #(gen/fmap (fn [x]
                                 (Timestamp. (-> (time/minus (time/instant) (time/seconds x))
                                                 .toEpochMilli)))
                               (s/gen (s/int-in 0 10000)))))
```

Decimal Types -> Specs

```
(let [int-part (- numeric_precision numeric_scale)
      fraction-part numeric_scale
      max (read-string (format "%s.%s"
                               (string/join "" (repeat int-part "9"))
                               (string/join "" (repeat fraction-part "9"))))
      min (read-string (format "-%s.%s"
                               (string/join "" (repeat int-part "9"))
                               (string/join "" (repeat fraction-part "9"))))]
  `(precision-numeric ~max ~min))
```

```
(defn precision-numeric [max min]
  (s/with-gen number?
    #(s/gen (s/double-in :max max :min min))))
```

String Types -> Specs

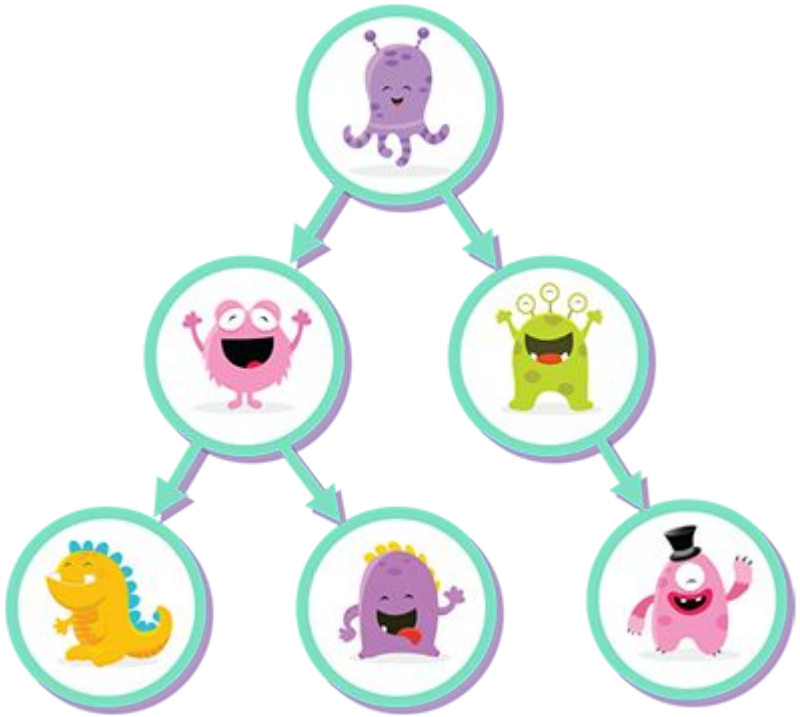
```
(contains? #{"char" "varchar"} data_type)
`(info-specs/string-up-to ~character_maximum_length)

(contains? #{"longtext"} data_type)
`(info-specs/string-up-to 500)
```

```
(defn string-up-to [max-len]
  (s/with-gen string?
    #(gen/fmap (fn [x] (apply str x))
               (gen/bind (s/gen (s/int-in 0 max-len))
                         (fn [size]
                           (gen/vector (gen/char-alpha) size))))))
```

Custom Types -> Specs

```
(def custom-specs
  "A map of database [table field] pairs to a custom type so that we can generate
   better specs for things like email-addresses, phone-numbers, enumerations that
   aren't explicitly enumerated in the DB"
  {["addresses" "addressable_id"] ::banded-id
   ["addresses" "addressable_type"] #{"company_loan_data" "person"}
   ["banks" "intermediary_bank_id"] ::self-ref
   ["loans_persons_roles" "signatory_id"] ::self-ref
   ["loans_persons_roles" "warrantor_id"] ::self-ref
   ["loan_parts" "parent_id"] ::self-ref})
```

Specmonstah



Foreign-Key Query

```
(def +foreign-key-query+
  "Query to extract foreign key meta-data from the mysql info schema"
  "select kcu.table_name
        , kcu.column_name
        , kcu.referenced_table_name
        , referenced_column_name
    from information_schema.key_column_usage kcu
   where kcu.referenced_table_name is not null
     and kcu.table_schema = ? and kcu.table_name in (<table-list>)
 order by 1, 2")
```

Specmonstah Example

```
:addresses
{:prefix :addresses,
 :spec :celm.tables/addresses,
 :relations {:country-id [:countries :id]},
 :constraints {:country-id #{:uniq}}},
```

TDD your CDC

Andy Chambers
@cddr

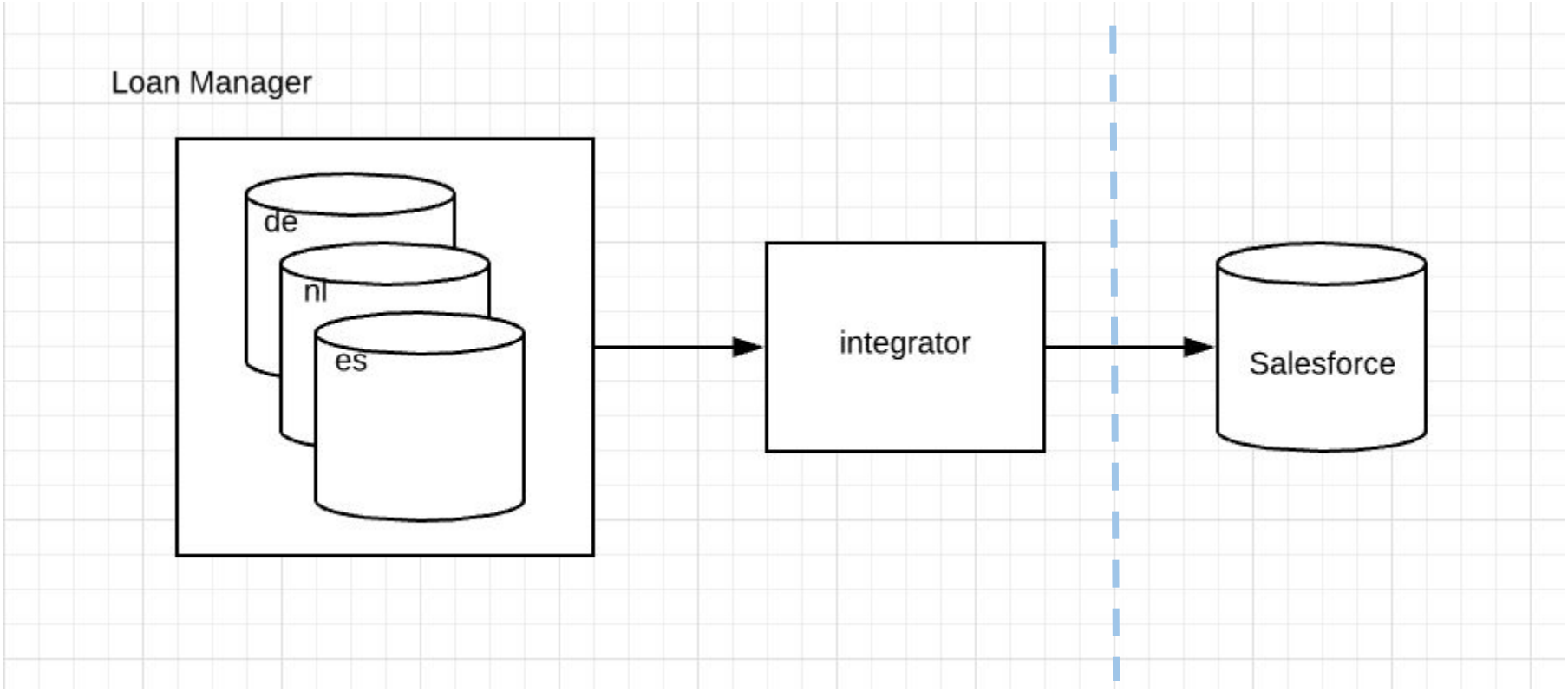
Replying to @gunnarmorling

TDD your CDC pipeline

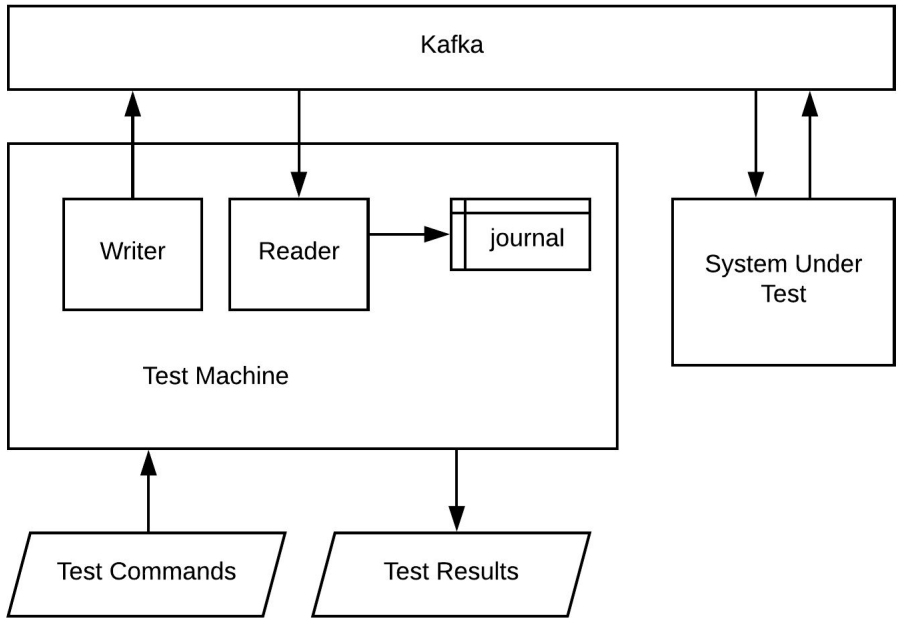
```
(deftest ^:dbz test-company-loan-data-wrangling
  (test-wrangler
    (integration-test-config :company-loan-data)
    (do-assertions
     (wrangled-ok? default-checks)
     (uuids? [:company-uuid])
     (banded-dwh-keys? [:company-id :ranking-id :legal-form-id])
     (includes? (whitelisted-fields wrangler-config :company-loan-data))
     (excludes? [:last-year-revenue :biggest-client-revenue-percentage])
     (namespaced-logs? :company-loan-data))))
```

2:53 PM · Oct 4, 2019 · Twitter Web App

Mergers & Acquisition

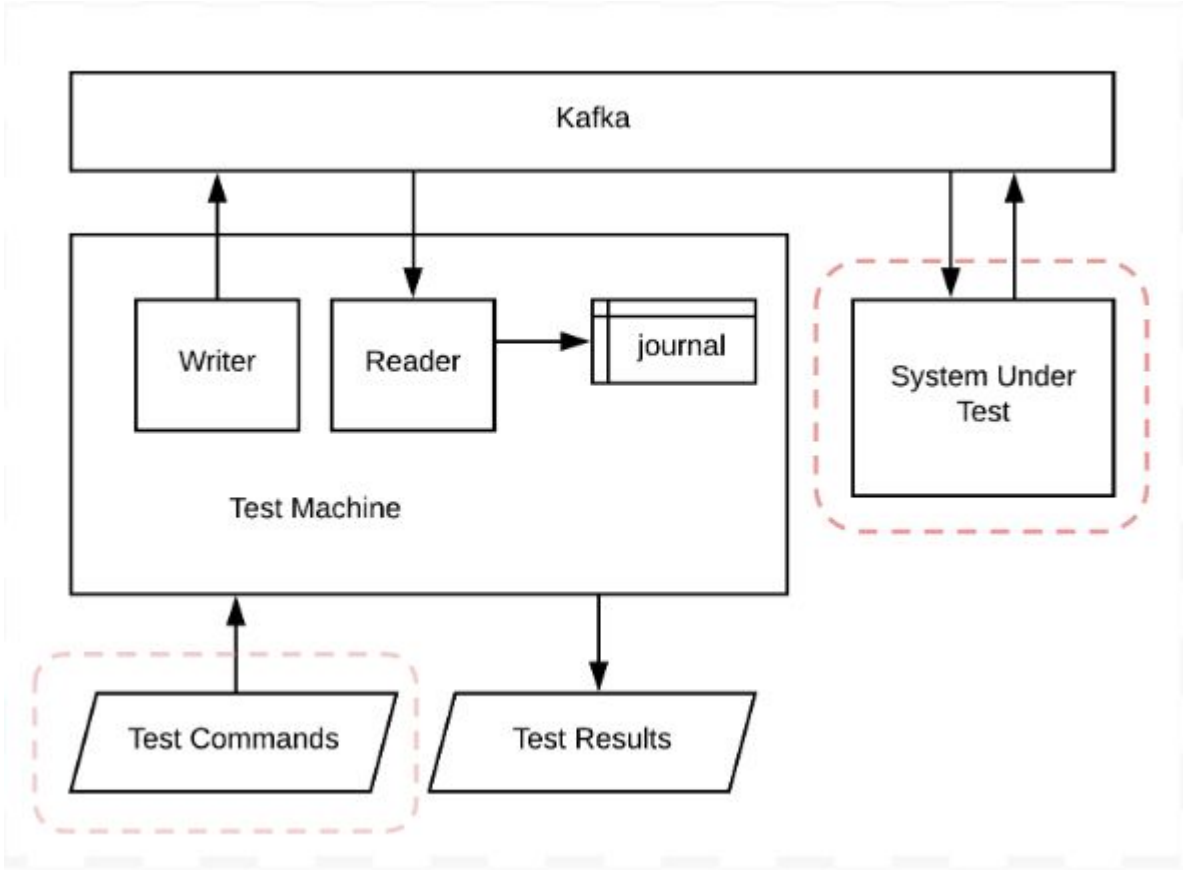


The Test Machine



https://cljdoc.org/d/fundingcircle/jackdaw/0.6.9/doc/the-test-machine
https://github.com/FundingCircle/jackdaw

Building the Test Helper



Test Commands

```
(jd.test/with-test-machine (jd.test/kafka-transport +kafka-config+ topic-metadata)
  (fn [machine]
    (jd.test/run-test machine
                      [[:println "> Starting test ..."]
                       [:do! (fn [_]
                               (jdbc/with-db-connection [db +mysql-spec+]
                                 (jdbc/with-db-transaction [tx db]
                                   (process-mysql-commands tx inputs))))]
                       [:println "> Watching for results ..."]
                       [:watch (every-pred
                                 (partial watch-fn inputs "fc_es_prod")
                                 (partial watch-fn inputs "fc_de_prod")
                                 (partial watch-fn inputs "fc_nl_prod"))
                        {:timeout 45000}]
                       [:println "> Got results, checking ..."]])))]
```

System Under Test

```
(fix/with-fixtures [(fix/topic-fixture +kafka-config+ topic-metadata)
                    (fn [t]
                      (jdbc/with-db-connection [db +mysql-spec+]
                        (jdbc/with-db-transaction [tx db]
                          (without-constraints tx
                                               (fn []
                                                 (truncate-tables inputs)
                                                 (t))))))
                    (connector-fixture {:base-url +dbz-base-url+
                                        :connector (dbz-connector "fc_de_prod" inputs)})
                    (fix/kstream-fixture {:topology (partial build-fn logger)
                                          :config (sut/config)})]
```

Assertion Helpers

```
(defn includes?
  [included-keys]
  (fn [{:keys [after]}]
    (println "  - checking includes?" included-keys)
    (is (every? #(clojure.set/superset? (set (keys %)) included-keys) after))))

(defn excludes?
  [excluded-keys]
  (fn [{:keys [before after]}]
    (println "  - checking excludes?" excluded-keys)
    (doseq [k excluded-keys]
      (testing (format "checking %s is excluded" k)
        (is (every? #(not (contains? % k)) after))))))

(defn uuids?
  [uuid-keys]
  (fn [{:keys [before after]}]
    (println "  - checking uuids?" uuid-keys)
    (doseq [k uuid-keys]
      (testing (format "checking %s is a uuid" k)
        (is (every? #(uuid? (java.util.UUID/fromString (get % k))) after))))))
```

Et Voilà

```
(deftest ^:dbz test-company-loan-data-wrangling
  (test-wrangler
    (integration-test-config :company-loan-data)
    (do-assertions
     (wrangled-ok? default-checks)
     (uuids? [:company-uuid])
     (banded-dwh-keys? [:company-id :ranking-id :legal-form-id])
     (includes? (whitelisted-fields wrangler-config :company-loan-data))
     (excludes? [:last-year-revenue :biggest-client-revenue-percentage])
     (namespaced-logs? :company-loan-data))))
```

Twitter/Slack: @cddr
Email: achambers.home@gmail.com