

On this page



Getting Started

This documentation uses [The Princess Bride example application in the HugSQL repository](#). Feel free to view or clone the repo and run the application with `lein as-you-wish`.



Start with SQL

HugSQL provides a clean separation of SQL and Clojure code. You can start developing with HugSQL by deciding where to put your SQL files in your application.

HugSQL can find any SQL file in your classpath. You can place your SQL files under `resources`, `src`, or elsewhere in your classpath.

Our example application puts SQL files under src:

```
.../examples/princess-bride $ tree
.
├── LICENSE
├── project.clj
├── README.md
├── resources
├── src
│   └── princess_bride
│       ├── core.clj
│       ├── db
│       │   ├── characters.clj
│       │   ├── quotes.clj
│       │   └── sql
│       │       ├── characters.sql
│       │       └── quotes.sql
│       └── db.clj
```

Our SQL for The Princess Bride characters is as follows:

SQL

```
-- src/princess_bride/db/sql/characters.sql
-- The Princess Bride Characters

-- :name create-characters-table
-- :command :execute
-- :result :raw
-- :doc Create characters table
-- auto_increment and current_timestamp are
-- H2 Database specific (adjust to your DB)
create table characters (
  id          integer auto_increment primary key,
  name        varchar(40),
```

```
specialty varchar(40),
created_at timestamp not null default current_timestamp
)

/* ...snip... */

-- A :result value of :n below will return affected rows:
-- :name insert-character :! :n
-- :doc Insert a single character returning affected row count
insert into characters (name, specialty)
values (:name, :specialty)

-- :name insert-characters :! :n
-- :doc Insert multiple characters with :tuple* parameter type
insert into characters (name, specialty)
values :tuple*:characters

/* ...snip... */

-- A ":result" value of ":1" specifies a single record
-- (as a hashmap) will be returned
-- :name character-by-id :? :1
-- :doc Get character by id
select * from characters
where id = :id

-- Let's specify some columns with the
-- identifier list parameter type :i* and
-- use a value list parameter type :v* for IN()
-- :name characters-by-ids-specify-cols :? :*
-- :doc Characters with returned columns specified
select :i*:cols from characters
where id in (:v*:ids)
```

HugSQL uses special SQL comments to accomplish its work. These conventions are explained later in this document. Keep reading!

Now Some Clojure

Now we write a bit of Clojure to define our database functions.

Clojure

```
(ns princess-bride.db.characters
  (:require [hugsql.core :as hugsql]))

;; The path is relative to the classpath (not proj dir!),
;; so "src" is not included in the path.
;; The same would apply if the sql was under "resources/..."
;; Also, notice the under_scored path compliant with
;; Clojure file paths for hyphenated namespaces
(hugsql/def-db-fns "princess_bride/db/sql/characters.sql")

;; For most HugSQL usage, you will not need the sqlvec functions.
;; However, sqlvec versions are useful during development and
;; for advanced usage with database functions.
(hugsql/def-sqlvec-fns "princess_bride/db/sql/characters.sql")
```


The `princess-bride.db.characters` namespace now has several functions defined based on the SQL statements in the SQL file. Here's an example of the `sqlvec` output and a sample run of the `characters-by-ids-specify-cols` function:

Clojure

```
(characters/characters-by-ids-specify-cols-sqlvec
  {:ids [1 2], :cols ["name" "specialty"]}) ;=>
["select name, specialty from characters
  where id in (?,?)"
 ,1
 ,2]
```

```
(characters/characters-by-ids-specify-cols db
  {:ids [1 2], :cols ["name" "specialty"]}) ;;=>
({:name "Westley", :specialty "love"}
 {:name "Buttercup", :specialty "beauty"})
```

You've only scratched the surface of HugSQL's functionality. Keep reading for full usage examples.

 [Edit this page](#)