SqlMake

Release 0.3.0

AmvTek developers

Dec 06, 2021

CONTENTS

Contents:

CHAPTER

ONE

OVERVIEW

1.1 Who needs SqlMake ?

You will benefit from SqlMake if you are confortable with SQL and see the value of defining your sql schemas (tables, stored functions, indexes, roles...) directly in sql and not through an ORM system like the Django ORM, SqlAlchemy or Hibernate to name a few.

What SqlMake allows you to do is to split your SQL schema accross multiples sql files accurately defining **dependencies** which may exist in between such files by mean of special SQL comments. When in need to recreate your database schema, the **sqlmake** tool will collect all files resources that composes the schema, parse them and emit the SQL commands they contain in optimal order so as to respect the dependencies that have been defined.

Once your schema has been split in between several files, it will be very easy to read and maintain. If you archive it in a version control system like git, subversion or mercurial it will also be very easy to prepare **migration** scripts.

1.2 Installing SqlMake

Installing the **sqlmake** CLI tool currently requires you have some familiarities with the way python packages are distributed. For now **sqlmake** has been tested only with python **2.7** interpreter.

To install SqlMake and its dependencies using pip, run

```
pip install SqlMake
```

1.3 Running the sqlmake CLI

Getting help

```
sqlmake -h
```

Compiling a schema from a set of resources

```
sqlmake --out=myschema.sql path/to/project/folder
```

CHAPTER

TWO

WORKING WITH SQLMAKE

A SqlMake project consists of files called **resources** stored in a folder. Every file with *.sql* extension, in project folder or subfolders are project resources.

SqlMake allows to add special instructions to a resource file, in a non obtrusive way :

SQL comment line starts with

```
--
```

SqlMake instructions starts with

```
--#
```

2.1 Defining dependencies (DEPS)

To add dependencies to a resource file you add **DEPS** instructions at the top of the file. Each **DEPS** instruction provides a comma separated list of relative paths to resource files or folder in your project. If you are using folder dependency, SqlMake will automatically assumes that all the resources it contains are dependencies of the file that defines it.

2.1.1 Dependencies example

Assume the following project structure:

```
project/
├── appschema
│   ├── init.sql
│   └── mytable.sql
├── public
│   ├── add_extensions.sql
│   └── functions.sql
├── README.txt
└── roles.sql
```

So as get the *appschema/mytable.sql* resource to depends of the *appschema/init.sql* resource and of all the resources in the public folder just add the followings DEPS instruction at the top of the *mytable.sql* file.

```
--# DEPS: init, ../public

CREATE TABLE t_mytable(
    ...
```

2.2 Renaming schema elements (VARS)

SqlMake resources maybe used during development as *normal* SQL file without the help of the **sqlmake** command. The **VARS** instruction allows to define which *name* maybe redefined when compiling the schema. The **sqlmake** command allows to redefine some of the schema name by means of the **–def option**.

2.2.1 Renaming example

Let's assume that in file mytable.sql, we want to allows renaming at *compilation time* the table t_mytable into something else and also to change table owner amvtek into another role defined by variable schema_owner. A **VARS** instruction will be added at the top of the file to make this possible.

```
--# DEPS: init, ../public
--# VARS: t_mytable, amvtek=owner_role

create table t_mytable(
    id integer primary key,

    name varchar(80) not null,
    ...
);

-- set table owner to role amvtek
alter table t_mytable owner to amvtek;
```

To rename t_mytable into t_othertable and amvtek role into titus, one may use the sqlmake command like so

```
sqlmake --def t_mytable=t_othertable --def owner_role=titus path/to/mytable.sql
```

2.3 Unleashing the power of Jinja templates

SqlMake is built on top of the well known Jinja template engine . You may use any of the statements exported by Jinja such as if/endif, for/endfor embedding those in SQL comment line that starts with

```
--#
```

2.3.1 Jinja instruction example

Assumes that when in development we want our example table to be created in schema tests, and that tests shall be recreated each time we are loading the mytable.sql file in the development database. When compiling the full schema using **sqlmake** the commands necessary for this to happen shall not be executed.

A simple Jinja conditional block, will make this a snapp :

```
--# DEPS: init, ../public
--# VARS: t_mytable, amvtek=owner_role

--# if __development__ :
```

(continues on next page)

(continued from previous page)

```
-- sqlmake will not render this block
-- as long as __development__ stays undefined...

drop schema if exists tests;
create schema tests;
set search_path to tests, public;

--# endif

create table t_mytable(
    id integer primary key,

    name varchar(80) not null,
    ...
);

-- set table owner to role amvtek
alter table t_mytable owner to amvtek;
```

CHAPTER

THREE

INVOCATION OF SQLMAKE

```
usage: sqlmake [-h] [-d name=value] [--out OUTFILE] [--ext EXT] IPATH

build a SQL schema from a set of files

positional arguments:
  IPATH                 path to folder or file that contains schema
                        definitions

optional arguments:
  -h, --help            show this help message and exit
  -d name=value, --def name=value
                        list variable definition as name=value
  --out OUTFILE         file in which SQL will be saved (default -)
  --ext EXT             file extension for schema resources (default sql)
```

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search