YamSql Documentation

Release

Michael Herold

July 14, 2016

CHAPTER 1

YamSql

YamSql is a language to describe SQL schemas (i.e. database structures) based on YAML.

This project contains the joined documentation and defitinition of YamSql available via yamsql.readthedocs.io.

1.1 Implementation

The reference implementation of YamSql is HamSql. It currently supports deployment on PostgreSQL servers and generating documentations of the SQL schemas.

CHAPTER 2

Language Constructs

2.1 Files

2.1.1 Config Load Directory

Only files ending with an alphanumeric character and not beginning with a dot are considered.

2.1.2 Front Matter

YAML front matter is a method to add YAML content to a document originally defined by Jekyll. The YAML part is added to the beginning of the document between triple-dashed lines.

Listing 2.1: General structure of a document with YAML front matter

```
---
<YAML>
---
<CONTENT>
```

2.2 Value Types

2.2.1 List

Lists are just YAML Lists

Listing 2.2: Different ways for providing lists

```
# recommended
list1:
 - this
 - is a
 - list
dlist1:
 -
  a: list with
  b: sub structure
 # this variant saves a line
```

```
 - a: x
   b: y

# mostly discured for YamSql
list2: [this, is a, list]
dlist2: [{a: list with, b: sub structure}, {a: x, b: y}]
```

2.2.2 SQL Identifier

Internally, if no double quote character is present, the parts seperated by periods are escaped or enquoded. If a double quote character is present, it is assumed that the identifier is properly enquoted.

2.2.3 SQL Type

The following characters prevent processing of the string:

- `"` double quotes
- `%` percent sign
- `(  ...  )` pair of parenthesis

as does the occurence of no period (`.`).

Listing 2.3: Examples of the proccessing algorithm

```
varchar      -> varchar
a.b          -> "a"."b"
"a".b        -> "a".b
"a.b"        -> "a.b"
a.b(10)      -> a.b(10)
a.b%ROWTYPE -> a.b%ROWTYPE
```

2.2.4 String

Strings are YAML strings. In most cases they can be given unquoted. However, there are some special cases, where things go wrong.

1. Inputs like `true` or `false` are interpreted as *Bool* and have to be en-quoted.
2. Quotes are used to mark strings. If you need the string `"string"`, you can use `"""string"""`.

```
key1a: this is a string.
# also possible but not required
key1b: "this is a string."
# this one needs quoting
key2: "true"
# this represents the string "string"
key3: """string"""
```

2.2.5 Bool

Bools are Yaml boolean values. Values can be `true` or `false`

CHAPTER 3

Function

The *Config Load Directory* is `functions.d`. For functions two allowed formats for giving the functions body exist. The usual variant is to give the following YAML structure including the `body` value. The second variant is to give the following YAML structure as *Front Matter* (i. e. fenced with `---` lines) providing the body as content following the frontmatter. The *Examples* include both variants.

The recommended practice is to use the *Front Matter* style while using a filename extension that matches the used language (`.pgsql`, `.sql`, `.py`) to enable syntax highlighting in editors.

name ***SQL Identifier*** Function name

description ***String*** Function description

returns ***SQL Type*** Return type of the function, the value *TABLE* is special (see return_table)

parameters ***List*** **[*Variable*]** parameters the function takes

templates ***List*** **[*SQL Identifier*]** list of template names, from which this function derives definitions (see FunctionTpl)

returns_columns ***List*** **[*Parameter*]** If the value of *return* is `TABLE` (case sensitive), this options defines the columns that are returned

priv_execute ***List*** **[*SQL Identifier*]** Role that has the privilege to execute the function

security_definer ***Bool*** If true, the function is executed with the privileges of the owner! Owner has to be given, if this is true

owner ***SQL Identifier*** owner of the function

language ***String*****:** language in which the body is written.

variables ***List*** **[*Variable*]** Variables

body ***String*** The code of the function (body)

3.1 Parameter

name ***SQL Identifier*** Name

type ***SQL Type*** Type

description ***String*** Description

3.2 Variable

name *SQL Identifier* Name

type *SQL Type* Type

description *String* Description

default *String* Default

3.3 Examples

Listing 3.1: Usual definition using plain YAML

```
name: f
description: |
 Always returns ``1``
returns: int
body: |

 RETURN 1;
```

Listing 3.2: Same function with the function body following a *Front Matter*

```
---
name: f
description: |
 Always returns ``1``
returns: int
---

RETURN 1;
```

Listing 3.3: Same function written in Python 3

```
---
name: f
description: |
 Always returns ``1``
returns: int
language: plpython3u
---

return 1
```

3.4 External Resources

- PostgreSQL's CREATE FUNCTION statement

CHAPTER 4

Role

Roles are a unification of the concept of users an groups.

name ***SQL Identifier*** Role name

description ***String*** Description

login ***Bool*** Can role login, non-login roles are groups (default: *false*)

password ***String*** password in plain text

member_in ***List*** **[*SQL Identifier*]** List of roles the role is member of

4.1 External Resources

- PostgreSQL's CREATE ROLE

CHAPTER 5

Schema

A database can contain several *schemas* where each schema can contain objects like tables and functions potentially with identical names without conflicting. Thus, schemas share similarities with the concept of namespaces.

YamSql schema definitions consist of a folder which shares it's name with the schema. The folder must contain a `schema.yml` file.

5.1 Syntax of schema.yml

name ***SQL Identifier*** Schema name.

description ***String*** Schema description. It is recommended to use *reStructuredText* for adding markup to this fields content.

dependencies ***List*** **[*SQL Identifier*]** Other schemas required for this schema to work.

5.2 Examples

```
name: my_app
description: |
 Main strucutre for MyApp

dependencies:
 - other_app
```

5.3 External Resources

- PostgreSQL Schemas
- PostgreSQL's CREATE SCHEMA statement

CHAPTER 6

Table

The *Config Load Directory* is `tables.d` and must contain files with the following strucutre.

name ***SQL Identifier*** table name

description ***String*** what this table is good for

columns ***List*** **[*Column*]** columns contained in this table

primary_key ***List*** **[*SQL Identifier*]** list of column names that define the primary key

foreign_keys ***List*** **[*Foreign Key*]** constains values via foreign keys

checks ***List*** **[*Check*]** validity checks applied to the table

inherits **(*List*** **[*SQL Identifier*]** Inherits

priv_select ***List*** **[*SQL Identifier*]** grant SELECT to given roles for this table

priv_insert ***List*** **[*SQL Identifier*]** grant INSERT

priv_update ***List*** **[*SQL Identifier*]** grant UPDATE

priv_delete ***List*** **[*SQL Identifier*]** grant DELETE

templates ***List*** **[*SQL Identifier*]** (see TableTpl)

6.1 Column

name ***SQL Identifier*** column name

type ***SQL Type*** column type (see also *Type*, *Domain*)

description ***String*** description

template ***SQL Identifier*** if a ColumnTpl is used, _type_ and _description_ can be omitted

default ***String*** default value (sql code)

null ***Bool*** Sql _NULL_ is allowed as value (default _false_)

references ***SQL Identifier*** References

on_ref_delete ***String*** On Ref Delete

on_ref_update ***String*** On Ref Update

unique ***Bool*** Unique

checks ***List*** **[*Check*]** Checks

6.2 Foreign Key

name ***SQL Identifier*** Just a name

columns ***List*** **[*SQL Identifier*]** Columns in this table

ref_table ***SQL Identifier*** Table to reference

ref_columns ***List*** **[*SQL Identifier*]** Columns in referenced table (order must match the one in *columns*)

on_delete ***String*** Action when entry in foreign table is deleted

on_update ***String*** Action when entry in foreign table is update

6.3 External Resources

- PostgreSQL's CREATE TABLE statement

CHAPTER 7

Check

name *SQL Identifier* Name

description *String* Description

check *String* SQL code

CHAPTER 8

Domain

Domains are basically semantic aliases for build in types with some extra constraints called *Checks*.

The *Config Load Directory* is `domains.d` and must contain files with the following strucutre.

name ***SQL Identifier*** Name

description ***String*** Description

type ***SQL Type*** Type

default ***String*** Default

checks ***List*** **[*Check*]** Checks

8.1 Examples

```
name: email_address
description: Valid email address
type: varying(254)

checks:
 -
  name: email_regex
  description: |
   Ensures that the address contains an ``@`` and something before the ``@``
  check: |
   POSITION('@' IN VALUE) > 1
```

8.2 External Resources

- PostgreSQL's CREATE DOMAIN statement

CHAPTER 9

Type

Composite Type

name *SQL Identifier* Type Name

description *String* Description

elements *List* **[***Type Element***]** TypeElements

9.1 Type Element

name *SQL Identifier* Name

type *SQL Type* Type

9.2 Examples

```
name: plant
description: |
 Stores numbers of ``flowers`` and ``leaves`` of a plant.
elements:
 -
  name: flowers
  type: integer
 -
  name: leaves
  type: integer
```

9.3 External Resources

- PostgreSQL's CREATE TYPE statement