# JSONSchema2DB

**Feb 07, 2022**

# Contents

A simple utility to convert JSON Schemas into relational tables in Postgres/Redshift.

Also see the Github page for source code and discussion!

CHAPTER 1

# Installation

The easiest way to install is from PyPI:

```
pip install jsonschema2db
```

# CHAPTER 2

## Quick overview

Let's say you have the JSON schema test_schema.json:

```
{
    "$schema": "http://json-schema.org/schema#",
    "title": "Fact schema",
    "type": "object",
    "definitions": {
        "basicAddress": {
            "type": "object",
            "comment": "This is an address",
            "properties": {
                "City": {
                    "type": "string",
                    "comment": "This is a city"
                },
                "State": {
                    "type": "string",
                    "minLength": 2,
                    "maxLength": 2
                },
                "Street": {
                    "type": "string"
                },
                "ZipCode": {
                    "type": "string"
                }
            }
        },
        "address": {
            "allOf": [
                {
                    "$ref": "#/definitions/basicAddress"
                },
                {
```

(continues on next page)

(continued from previous page)

```
                    "type": "object",
                    "properties": {
                        "Latitude": {
                            "type": "number",
                            "minimum": -90,
                            "maximum": 90
                        },
                        "Longitude": {
                            "type": "number",
                            "minimum": -180,
                            "maximum": 180
                        }
                    }
                }
            ]
        },
        "unum": {
            "type": "number",
            "minimum": 0
        }
    },
    "properties": {
        "Loan": {
            "type": "object",
            "description": "Loan information",
            "properties": {
                "Amount": {
                    "type": "number"
                },
                "SomeSuperLongColumnNameThatDoesntFitInPostgresUnfortunately": {
                    "type": "number"
                },
                "AbbreviateThisReallyLongColumn": {
                    "type": "number"
                }
            }
        },
        "SubjectProperty": {
            "type": "object",
            "properties": {
                "Acreage": {
                    "$ref": "#/definitions/unum"
                },
                "Address": {
                    "$ref": "#/definitions/address"
                }
            }
        },
        "RealEstateOwned": {
            "type": "object",
            "patternProperties": {
                "[0-9]+": {
                    "type": "object",
                    "properties": {
                        "Address": {
                            "$ref": "#/definitions/basicAddress"
                        },
```

(continues on next page)

(continued from previous page)

```
                        "RentalIncome": {
                            "type": "number"
                        }
                    }
                }
            }
        }
    }
}
```

This looks a bit complex, but basically:

1. There's a shared definition for an *basicAddress* definition that has the normal address fields: state, zip, etc
2. There's a definition *address* that extends *basicAddress* and adds latitude and longitude
3. Each "loan" (we are in the mortgage industry) tracks a loan amount, a bunch of info (including address) for the subject property (the property that the loan is against), and a list of other properties owned by the borrower (each of which has an address and a rental income)

jsonschema2db creates the following tables automatically:

```
create table "schm"."root" (
   id serial primary key,
   "loan_file_id" int not null,
   "prefix" text not null,
   "loan__amount" float,
   "subject_property__acreage" float,
   "subject_property__address__latitude" float,
   "subject_property__address__longitude" float,
   "subject_property__address_id" integer,
   unique ("loan_file_id", "prefix")
)

create table "schm"."basic_address" (
   id serial primary key,
   "loan_file_id" int not null,
   "prefix" text not null,
   "city" text,
   "root_id" integer,
   "state" text,
   "street" text,
   "zip_code" text,
   unique ("loan_file_id", "prefix")
)

create table "schm"."real_estate_owned" (
   id serial primary key,
   "loan_file_id" int not null,
   "prefix" text not null,
   "address_id" integer,
   "rental_income" float,
   "root_id" integer,
   unique ("loan_file_id", "prefix")
)
```

As you can see, we end up with three tables, each containing a flat structure of scalar values, with correct types. jsonschema2db also converts camel case into snake case since that's the Postgres convention. Unfortunately, Post-

gres limits column names to 63 characters (127 in Redshift). If you have longer column names, provide a list of abbreviations using the *abbreviations* parameter to the constructor.

jsonschema2db also handles inserts into these tables by transforming them into a flattened form. On top of that, a number of foreign keys will be created and links between the tables.

The rule for when to create a separate table is that either:

1. It's a shared definition that is an object (with links from the parent to the child)
2. Any object with *patternProperties* will have its children in a separate table (with links back to the parent, if the link is unique)

## 2.1 Creating tables

The first step is to instantiate a `jsonschema2db.JSONSchemaToPostgres` object (or the corresponding `jsonschema2db.JSONSchemaToRedshift` and create the tables using `jsonschema2db.JSONSchemaToPostgres.create_tables()`:

```
schema = json.load(open('test/test_schema.json'))
translator = JSONSchemaToPostgres(
    schema,
    postgres_schema='schm',
    item_col_name='loan_file_id',
    item_col_type='string',
    abbreviations={
        'AbbreviateThisReallyLongColumn': 'AbbTRLC',
    }
)

con = psycopg2.connect('host=localhost dbname=jsonschema2db-test')
translator.create_tables(con)
```

## 2.2 Inserting data

Now, let's insert some data into the tables:

```
translator.insert_items(con, [
    ('loan_file_abc123', {
        'Loan': {'Amount': 500000},
        'SubjectProperty': {'Address': {'City': 'New York', 'ZipCode': '12345',
↪'Latitude': 43}, 'Acreage': 42},
        'RealEstateOwned': {'1': {'Address': {'City': 'Brooklyn', 'ZipCode': '65432'},
↪ 'RentalIncome': 1000},
                            '2': {'Address': {'City': 'Queens', 'ZipCode': '54321'}}},
    })
])
```

This will create the following rows:

```
jsonschema2db-test=# select * from schm.root;
-[ RECORD 1 ]------------------------+-----------
id                                   | 1
loan_file_id                         | loan_file_abc123
```

(continues on next page)

(continued from previous page)

```
prefix                                |
loan__amount                          | 500000
subject_property__acreage             | 42
subject_property__address__latitude   | 43
subject_property__address__longitude  |
subject_property__address_id          | 1

jsonschema2db-test=# select * from schm.basic_address;
-[ RECORD 1 ]+--------------------------
id           | 2
loan_file_id | 1000000000
prefix       | /RealEstateOwned/1/Address
city         | Brooklyn
root_id      | 1
state        |
street       |
zip_code     | 65432
-[ RECORD 2 ]+--------------------------
id           | 1
loan_file_id | 1000000000
prefix       | /SubjectProperty/Address
city         | New York
root_id      | 1
state        |
street       |
zip_code     | 12345

jsonschema2db-test=# select * from schm.real_estate_owned;
-[ RECORD 1 ]-+------------------
id            | 1
loan_file_id  | 1000000000
prefix        | /RealEstateOwned/1
address_id    | 2
rental_income | 1000
root_id       | 1
```

## 2.3 Post-insertion

After you're done inserting, you generally want to run `jsonschema2db.JSONSchemaToPostgres.create_links()` and `jsonschema2db.JSONSchemaToPostgres.analyze()`. This will add foreign keys and also analyze the table for better performance.

CHAPTER 3

# Full API documentation