OCDS Kingfisher Process

Release 0.0.1

Jan 15, 2020

Contents

Kingfisher Process stores and pre-processes OCDS data. An accompanying tool, Kingfisher Scrape, downloads OCDS data from sources, and optionally sends it to Kingfisher Process.

If you are viewing this on GitHub or PyPi, open the full documentation for additional details.

CHAPTER 1

Typical usage

Kingfisher Process receives OCDS data either via the *web API* (used by Kingfisher Scrape) or the *local-load* command.

OCDS data are stored in a PostgreSQL database and organized into "collections", as described in the *Data Model* and specified in the *Database structure*. Once incoming data are stored:

- If the *Default pre-processing pipeline* is enabled, the data are automatically pre-processed into new collections.
- If the *schema checks* are enabled, the data are automatically checked for schema errors.

Once the data are stored, you can query the PostgreSQL database; refer to the *Data Model* and *Database structure* for an orientation to the database tables.

A *Command-line tool* allows you to list collections, add notes to collections, run schema checks, and *transform collections*.

You can run the *web app* to view metadata about collections and files.

And that's it! In short, Kingfisher Process accepts "raw" OCDS data, and then checks and pre-processes it, so that your data analysis can be more predictable and repeatable.

1.1 Installation

1.1.1 Requirements

- Python 3.5 or higher
- PostgresSQL 10 or higher
- Redis

1.1.2 Installation

Create a virtual environment:

```
virtualenv -p python3 .ve
```

Activate the virtual environment:

```
source .ve/bin/activate
```

Install the requirements:

```
pip install -r requirements.txt
```

Install Redis with your package manager on Linux, for example:

```
sudo apt-get install redis-server
```

or with Homebrew on macOS:

```
brew install redis
```

1.1.3 Database

Create a user, for example:

```
sudo -u postgres createuser ocdskingfisher --pwprompt
```

Create a UTF8-encoded PostgreSQL database and give the user write access, for example:

```
sudo -u postgres createdb ocdskingfisher -O ocdskingfisher --template template0 --
↪encoding UTF8 --lc-collate en_US.UTF-8 --lc-ctype en_US.UTF-8
```

Set the tool's database connection setting, replacing at least `PASSWORD` in this example:

```
export KINGFISHER_PROCESS_DB_URI='postgres://ocdskingfisher:PASSWORD@localhost/
↪ocdskingfisher'
```

Note: This configures the tool within your current command-line session only. For longer-term options, see *Configuration*.

Create the tables in the database (more information on the *upgrade-database* command):

```
python ocdskingfisher-process-cli upgrade-database
```

Next: *Configuration*

1.2 Configuration

1.2.1 Setup

Create the tool's configuration directory:

```
mkdir ~/.config/ocdskingfisher-process
```

Download the sample main configuration file:

```
curl https://raw.githubusercontent.com/open-contracting/kingfisher-process/master/
↪samples/config.ini -o ~/.config/ocdskingfisher-process/config.ini
```

Open the main configuration file at `~/.config/ocdskingfisher-process/config.ini`, and follow the instructions below to update it.

1.2.2 PostgreSQL

Note: This step is required. All other steps are optional.

Configure the database connection settings:

```
[DBHOST]
HOSTNAME = localhost
PORT = 5432
USERNAME = ocdskingfisher
PASSWORD =
DBNAME = ocdskingfisher
```

If you prefer not to store the password in `config.ini`, you can use the PostgreSQL Password File, `~/.pgpass`, which overrides any password in `config.ini`. Otherwise, if you used the same settings as in the examples during *Installation*, you only need to set `PASSWORD` above.

To override `config.ini` and/or `.pgpass`, set the `KINGFISHER_PROCESS_DB_URI` environment variable. This is useful to temporarily use a different database than your default database. For example, in a bash-like shell:

```
export KINGFISHER_PROCESS_DB_URI='postgresql://user:password@localhost:5432/dbname'
```

1.2.3 Logging

This tool uses the Python logging module. Loggers are in the `ocdskingfisher` namespace.

Logging from the *Command-line tool* can be configured with a `~/.config/ocdskingfisher-process/logging.json` file. To download the default configuration:

```
curl https://raw.githubusercontent.com/open-contracting/kingfisher-process/master/
↪samples/logging.json -o ~/.config/ocdskingfisher-process/logging.json
```

To download a different configuration that includes debug messages:

```
curl https://raw.githubusercontent.com/open-contracting/kingfisher-process/master/
↪samples/logging-debug.json -o ~/.config/ocdskingfisher-process/logging.json
```

1.2.4 Web API

To allow access to the *web API*, set API keys, separated by commas. For example, to set `1234` and `5678` as keys (in practice, you should use long, random keys):

```
[WEB]
API_KEYS = 1234,5678
```

To override `config.ini`, set the `KINGFISHER_PROCESS_WEB_API_KEYS` environment variable.

1.2.5 Collection flags

When a *new collection* is created, *flags* are set to indicate what operations to perform on them.

All flags are off by default. To turn any on:

```
[COLLECTION_DEFAULT]
CHECK_DATA = true
CHECK_OLDER_DATA_WITH_SCHEMA_1_1 = false
```

1.2.6 Default pre-processing pipeline

To enable the *Default pre-processing pipeline*:

```
[STANDARD_PIPELINE]
RUN = true
```

1.2.7 Redis

Configure the Redis connection settings:

```
[REDIS]
HOST = localhost
PORT = 6379
DATABASE = 0
```

1.2.8 Sentry

To track crashes, sign up for Sentry, and set the DSN:

```
[SENTRY]
DSN = https://<key>@sentry.io/<project>
```

Note: Sentry has its own environment variables. Further reading: Sentry for Python.

1.3 Data Model

1.3.1 Collections

Collections are distinct sets of OCDS data. They are the largest unit on which this tool operates.

A collection is uniquely identified by the combination of:

- Name (`source_id`): A string. If the collection was created by Kingfisher Scrape, this is the `name` attribute of the spider.

- Date (`data_version`): The date and time at which the collection was created. If the collection was created by Kingfisher Scrape, this is the `start_time` statistic of the crawl.
- Sample (`sample`): A boolean. Whether the collection is only a sample of the data from the source.
- Base collection (`transform_from_collection_id`): An integer. The ID of the collection that was transformed into this collection.
- Transform type (`transform_type`): A string. The identifier of the transformer that was used to produce this collection.

Each collection is given an integer ID; this is used to refer to the collection in the *Command-line tool* and the database.

Collections are created by Kingfisher Scrape, the *web API*, or the *new-collection* command.

Schema check flags

Collections have flags that indicate what operations to perform on them. These are:

check_data Run CoVE schema checks on the data in this collection

check_older_data_with_schema_version_1_1 Force OCDS 1.1 checks to be run on OCDS 1.0 data (instead of OCDS 1.0 checks)

To configure the default values for these flags, see *Configuration*.

Transformed collections

Presently, the tool offers two transformers:

upgrade-1-0-to-1-1 upgrade a collection's data from OCDS 1.0 to OCDS 1.1

compile-releases merge a collection's releases into compiled releases

To transform a collection, create a new collection that refers to the base collection, with either the *new-transform-compile-releases* or *new-transform-upgrade-1-0-to-1-1* command, then run the *transform-collection* command.

1.3.2 Files

A collection contains one or more files. A file is uniquely identified by its collection and filename. Files can have:

errors The file could not be retrieved. Presently, errors are either reported by Kingfisher Scrape or caught by the *local-load* command.

warnings The file contents had to be modified in order to be stored. Presently, the only warning is about the removal of control characters.

File types

The *local-load* command must be given the type of the file to load:

record A single record

release A single release

record_list A JSON array of records, like `[ { record-1 }, { record-2 } ]`

release_list A JSON array of releases

record_package A single record package

release_package A single release package

record_package_list A JSON array of record packages, like `[ { record-package-1 }, { record-package-2 } ]`

release_package_list A JSON array of release packages

record_package_json_lines Line-delimited JSON, in which each line is a record package

release_package_json_lines As above, but release packages

record_package_list_in_results A JSON object with a `results` key whose value is a JSON array of record packages, like `{ "results": [ { record-package-1 }, { record-package-2 } ] }`

release_package_list_in_results As above, but release packages

release_package_in_ocdsReleasePackage_in_list_in_results A JSON object has a `results` key whose value is a list. Every item in that list is a JSON object. The object has a `ocdsReleasePackage` key who's value is a release package

1.3.3 Items

A file contains one or more items. An item is an OCDS resource: a release, record, release package or record package. An item is uniquely identified by its index within the file. Indices are `0`-based.

Files of the type `record`, `release`, `record_package`, or `release_package` have one item only. Files of other types have one or more items.

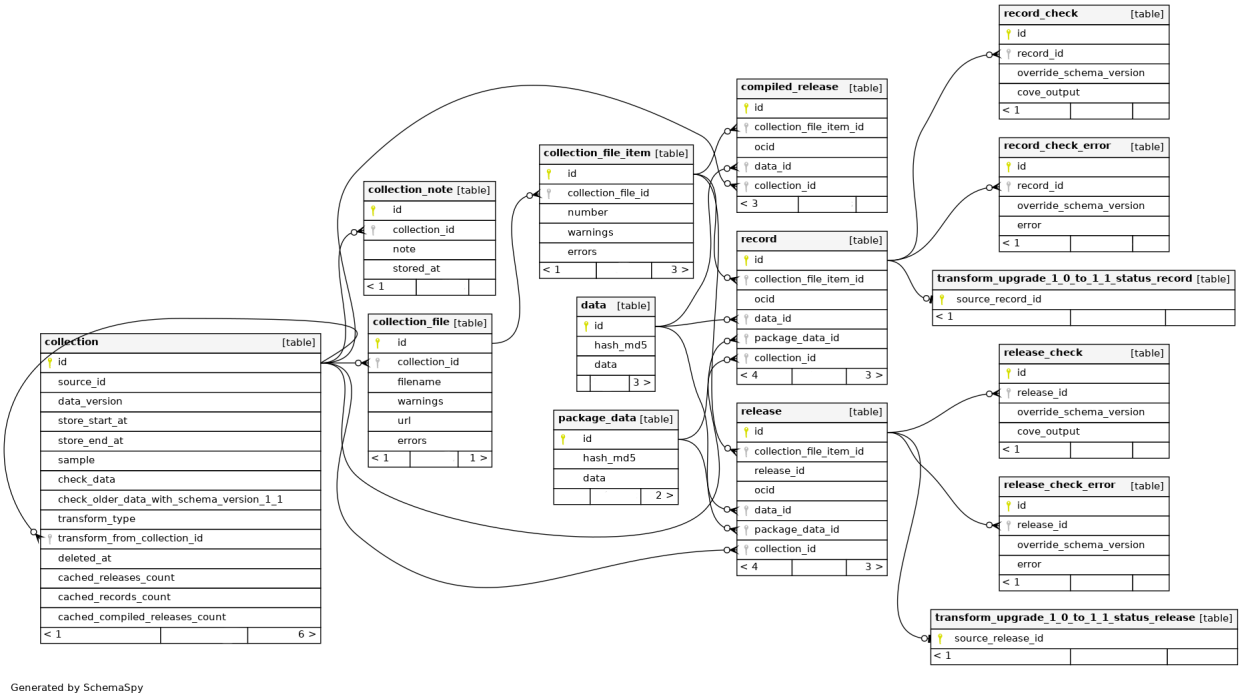
1.4 Database structure

Because Analysts query the data directly in the database, we document the structure here for them.

This is not a full description as you are expected to use normal tools to explore the database structure, but it hopefully contains the main points to get you started.

Reading *Data Model* before reading this will probably help.

The diagram below excludes the *release, record and compiled_release views with added collection information*.



1.4.1 collection table

This has columns to store the variables that make up the unique definition of a collection:

- *source_id*: the name of the collection that was run, for example 'canada_buyandsell'
- *data_version*: the date and time when the run command was executed
- *sample*: a mark that indicates if (false) the collection has all the available data or just a sample of it (true)
- *transform_from_collection_id*
- *transform_type*

This has columns to store what operations should be done to it:

- *check_data*
- *check_older_data_with_schema_version_1_1*

This has columns to track it's current state:

- *store_start_at*: the date and time when the store stage begun
- *store_end_at*: the data and time when the store stage end. For transformed collections, this means the time the transformer finished.
- *deleted_at*: the date and time when the collection was deleted. If this is set, this row will remain in the database but data about this collection may be cleared out of all other tables at any point and should NOT be relied on.

It also has some columns with cached values, for ease of use:

- *cached_releases_count*: Number of releases. Currently only calculated when store has finished.
- *cached_records_count*: Number of records. Currently only calculated when store has finished.
- *cached_compiled_releases_count*: Number of compiled releases. Currently only calculated when store has finished.

1.4.2 collection_note table

This table stores each note in a collection.

1.4.3 collection_file table

This table stores each file in a collection.

1.4.4 collection_file_item table

This table stores each item in a file.

1.4.5 data and package_data tables

These tables contain the actual data that is downloaded and stored.

The *hash_md5* column is used so that if duplicate data is noticed, we only store one copy of it.

This situation might arise when:

- storing one source - the package data is often the same
- storing one source at two different points in time - a lot of the data will be unchanged and the same

1.4.6 release, record and compiled_release tables

Each row is linked to *collection_file_item* and thus to collections. However, we also include a *collection_id* column so it's easy to select all data in one collection.

Each row is also linked to the *data* and *package_data* tables that actually hold the data.

Note that the `compiled_release` table is only populated by the compile-releases transform, and not by loading records from a data source.

1.4.7 release, record and compiled_release views with added collection information [deprecated]

In the past, three views were provided to make something easier:

- *release_with_collection*
- *record_with_collection*
- *compiled_release_with_collection*

The data in these views is now exactly the same as the normal tables:

- *release*
- *record*
- *compiled_release*

If you see any use of these views, please change to using the tables directly - you will get better performance.

1.4.8 release_check, record_check, release_check_error and record_check_error tables

These tables store the results of running the Data Review Tool (also called CoVE) on each piece of data. See http://standard.open-contracting.org/review/

1.4.9 transform_upgrade_1_0_to_1_1_status_release and transform_upgrade_1_0_to_1_1_status_record

These tables are simply used to store the progress of a Transform.

1.5 Default pre-processing pipeline

The default pre-processing pipeline operates on any new collection that is not a *transformed collection*; that is, any collection loaded via the *web API* or the *local-load* command.

The pipeline uses transforms to:

- upgrade the collection's incoming data from OCDS 1.0 to OCDS 1.1
- merge the collection's upgraded releases into compiled releases

The pipeline is off by default. To turn it on, see *Configuration*.

1.6 Command-line tool

You can use the tool with the provided CLI script. There are various sub-commands.

You can pass the *quiet* flag to all sub commands, to get less output printed to the terminal.

```
python ocdskingfisher-process-cli --quiet <command> ...
```

Installing and upgrading:

1.6.1 upgrade-database

This creates or upgrades the tables in the PostgreSQL database:

```
python ocdskingfisher-process-cli upgrade-database
```

To drop tables (and clear the Redis queue, if used) before upgrading, use the `--deletefirst` flag:

```
python ocdskingfisher-process-cli upgrade-database --deletefirst
```

OCDS Helpdesk deployment

Don't use this. It is run by the SaltStack scripts that upgrade the tool.

Working with the data and marking that you want actions to happen:

1.6.2 list-collections

This command lists all the collections this install of the app knows about.

```
python ocdskingfisher-process-cli list-collections
```

1.6.3 new-collection

This command creates a new collection in the system.

```
python ocdskingfisher-process-cli new-collection my-own-source-id  "2019-01-20␣
↪10:00:12"
python ocdskingfisher-process-cli new-collection my-own-source-id  "2019-01-20␣
↪10:00:12" --sample
```

You may not need to run this; collections will be created at certain points automatically for you. For instance, when data is pushed to the Web API.

But you may need to create a collection specially. For instance, you may want to create a new collection to load some local files into. See *local-load*.

1.6.4 local-load

This command loads files from disk into an existing collection in the system.

You need to create a collection to load the data into - see *new-collection*.

```
python ocdskingfisher-process-cli local-load 1 /data/moldova release_package
```

- Pass the ID of the collection you want to load the data into. Use *list-collections* to look up the ID you want.
- Pass the directory you want to load files from.
- Pass the type of the files. For possible options, see data types for files in *Data Model*

It will load files with a default encoding of *utf-8*. You can change it with the option *–encoding*.

```
python ocdskingfisher-process-cli local-load 2 /data/uk_contracts_finder release_
↪package --encoding ISO-8859-1
```

By default, afterwards the collection store will be marked as ended. If you want to leave it open (eg. so you can load more files) use the optional flag *–keep-collection-store-open*:

```
python ocdskingfisher-process-cli local-load --keep-collection-store-open 1 /data/
↪moldova release_package
```

If you want to manually end the store see *end-collection-store*.

1.6.5 end-collection-store

This command marks the storage of a collection as ended.

Pass the ID of the collection you want to end storage on. Use *list-collections* to look up the ID you want.

```
python ocdskingfisher-process-cli end-collection-store 17
```

1.6.6 new-collection-note

This command adds a note to a collection in the system.

```
python ocdskingfisher-process-cli new-collection-note 1 "The note you want added.␣
↪YOUR-NAME."
```

1.6.7 new-transform-compile-releases

This command takes an existing source collection that you give it, and creates a new destination collection with a transformer that creates compiled releases.

Note this does not actually do the transforming - it simply marks that you want the work to be done, and creates the destination collection ready for the finished work to be put into.

The compile releases transformer can only work when it has all the data! You can create the transformed collection at any time, but the source collection must be completely stored before any work will be done by the transformer.

Pass the ID of the source collection. Use *list-collections* to look up the ID you want.

It will create a new destination collection to hold the compiled releases and return the ID of this to you.

```
python ocdskingfisher-process-cli new-transform-compile-releases 17
```

After creating it, you should run transform-collection to actually do the work. See *transform-collection*

1.6.8 new-transform-upgrade-1-0-to-1-1

This command takes an existing source collection that you give it, and creates a new destination collection with a transformer that upgrades 1.0 data to 1.1.

Note this does not actually do the transforming - it simply marks that you want the work to be done, and creates the destination collection ready for the finished work to be put into.

Pass the ID of the source collection. Use *list-collections* to look up the ID you want.

It will create a new destination collection to hold the upgraded data and return the ID of this to you.

```
python ocdskingfisher-process-cli new-transform-upgrade-1-0-to-1-1 17
```

After creating it, you should run transform-collection to actually do the work. See *transform-collection*

1.6.9 delete-collection

This command deletes a collection in the system.

A collection can only be deleted if it doesn't have any transformed collections that refer to it.

Note this does not actually do the work of deleting - it simply marks that you want the deletion to be done.

Pass the ID of the collection you want the work done in. Use *list-collections* to look up the ID you want.

```
python ocdskingfisher-process-cli delete-collection 17
```

After marking it as deleted, you should run delete-collections to actually do the work. See *delete-collections*

Processing actions that have been requested:

1.6.10 check-collection

This command checks all data so far in a collection.

It can be run multiple times on a collection, and data already checked will not be rechecked.

Pass the ID of the collection you want checked. Use *list-collections* to look up the ID you want.

```
python ocdskingfisher-process-cli check-collection 17
```

OCDS Helpdesk deployment

Don't use this. A cron job runs *check-collections* once per hour.

1.6.11 check-collections

This command checks all data so far in all collections.

It can be run multiple times, and data already checked will not be rechecked.

You should only run one of these at once, as if two are run at once they may try and do the same work.

```
python ocdskingfisher-process-cli check-collections
```

Running from cron

You can also pass a maximum number of seconds that the process should run for.

```
python ocdskingfisher-process-cli check-collections --runforseconds 60
```

Soon after that number of seconds has passed, the command will exit. (The command will finish the check it's currently doing before stopping, so it may run slightly longer than specified. Allow a minute extra to be safe.)

You can use this option with a cron entry; set a cron entry for this command to run every hour and pass runforseconds as 3540 (60 seconds/minute * 59 minutes).

Then when new data appears in the system, there is no need for someone to run *check-collection* by hand - the process run by cron will pick up the new data itself eventually.

The runforseconds option will make sure that only one of these cron processes runs at once.

OCDS Helpdesk deployment

Don't use this. A cron job runs this once per hour.

1.6.12 transform-collection

This command runs the configured transformer for the collection.

It can be run multiple times on a collection, and data already transformed will not be retransformed.

Pass the ID of the collection you want the work done in. Use *list-collections* to look up the ID you want.

```
python ocdskingfisher-process-cli transform-collection 17
```

OCDS Helpdesk deployment

Don't use this. A cron job runs *transform-collections* once per hour.

1.6.13 transform-collections

This command runs the configured transformers for all collections.

It can be run multiple times on a collection, and data already transformed will not be retransformed.

You should only run one of these at once, as if two are run at once they may try and do the same work.

```
python ocdskingfisher-process-cli transform-collections
```

Running from cron

You can also pass a maximum number of seconds that the process should run for.

```
python ocdskingfisher-process-cli transform-collections --runforseconds 60
```

Soon after that number of seconds has passed, the command will exit. (The command will finish the transforming it's currently doing before stopping, so it may run slightly longer than specified. Allow a minute extra to be safe.)

You can use this option with a cron entry; set a cron entry for this command to run every hour and pass runforseconds as 3540 (60 seconds/minute * 59 minutes).

Then when new data appears in the system, there is no need for someone to run *transform-collection* by hand - the process run by cron will pick up the new data itself eventually.

The runforseconds option will make sure that only one of these cron processes runs at once.

OCDS Helpdesk deployment

Don't use this. A cron job runs this once per hour.

1.6.14 delete-collections

This command deletes the collections in the system where *deleted_at* column is not null.

```
python ocdskingfisher-process-cli delete-collections
```

OCDS Helpdesk deployment

Don't use this. A cron job runs this once a month.

1.6.15 update-collection-caches

This command updates cached values for all collections.

It can be run multiple times on a collection, as updating a cache will not cause any problems.

You should only run one of these at once, as if two are run at once they may try and do the same work.

```
python ocdskingfisher-process-cli update-collection-caches
```

Processing work from the Redis queues - there are several workers to process different types of work:

1.6.16 process-redis-queue

This commands processes any data in the Redis queue.

It will keep running until you stop it manually.

It is safe to run more than one of these commands at once.

```
python ocdskingfisher-process-cli process-redis-queue
```

Running from cron

You can also pass a maximum number of seconds that the process should run for.

```
python ocdskingfisher-process-cli process-redis-queue --runforseconds 60
```

Soon after that number of seconds has passed, the command will exit. (The command will finish the work it's currently doing before stopping, so it may run slightly longer than specified. Allow a minute extra to be safe.)

OCDS Helpdesk deployment

Don't use this. A cron job runs this once per hour.

1.6.17 process-redis-queue-collection-store-finished

This commands processes any data in the Redis queue for a collection that has just been marked as finishing it's store stage.

It will keep running until you stop it manually.

It is safe to run more than one of these commands at once.

```
python ocdskingfisher-process-cli process-redis-queue-collection-store-finished
```

Running from cron

You can also pass a maximum number of seconds that the process should run for.

```
python ocdskingfisher-process-cli process-redis-queue-collection-store-finished--
↪runforseconds 60
```

Soon after that number of seconds has passed, the command will exit. (The command will finish the work it's currently doing before stopping, so it may run slightly longer than specified. Allow a minute extra to be safe.)

OCDS Helpdesk deployment

Don't use this. A cron job runs this once per hour.

1.7 Web interface

1.7.1 Web API

The web API allows other applications (notably Kingfisher Scrape) to submit data to this tool to store.

All requests must set an HTTP Authorization request header with an authentication type of `ApiKey`. For example:

```
Authorization: ApiKey <key>
```

To configure the API keys, see *Configuration*.

API endpoints are documented on SwaggerHub.

1.7.2 Web app

The web app allows you to view metadata about collections and files.

To run the app locally in development mode:

```
FLASK_APP=ocdskingfisherprocess.web.app FLASK_ENV=development flask run
```

Then, open <http://127.0.0.1:5000/>

1.8 Development

1.8.1 Run tests

The tests drop and re-create the database; you should specify a testing database with an environment variable. See *Configuration*.

Run the tests with, for example:

```
KINGFISHER_PROCESS_DB_URI='postgres:///ocdskingfisher-test' pytest
```

1.8.2 Create migrations

1. Create a database migration with Alembic, for example:

```
alembic --config=mainalembic.ini revision -m "A short description of what the␣
↪migration does"
```

2. Fill in the migration

3. Add/update tables, indexes and/or constraints in `database.py` to match the migration
4. If a new table is created, update the `delete_tables` method

Note: Do not create simultaneous branches, each with its own migration(s). Instead, merge one branch, then create the next migration, to avoid multiple heads.

1.8.3 Updating Database tables graphic

In https://kingfisher-process.readthedocs.io/en/latest/database-structure.html you will find a graphic of the schema. If you change the database structure you will need to change this.

Download the database driver from https://jdbc.postgresql.org/ and SchemaSpy from https://github.com/schemaspy/schemaspy/releases .

First make sure your local database is up to date with all schema changes.

Then run SchemaSpy, something like:

```
java -jar /bin/schemaspy.jar -t pgsql -dp /bin/postgresql.jar -s public -db ocdskingfisher -u ocdskingfisher
-p ocdskingfisher -host localhost -o /vagrant/schemaspy
```

In the folder of data that results, take the schemaspy/diagrams/summary/relationships.real.large.png file. Copy it over docs/_static/database-tables.png.

Finally, use a standard image editing programme like https://www.gimp.org/ to edit out the row counts.