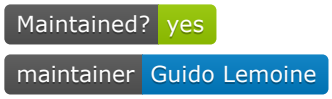
Onboarding a Member State on CDSE



Version 1.1, 15 February 2022

Version 1.2, 25 February 2022 - Updated extraction section for status

Version 1.3, 26 April 2022 - Minor updates

Version 1.4, 25 May 2023 - Updates to reflect migration to CDSE (ongoing)

Account set up

An Onboarding Member State needs a CDSE account. For the account, a contact person (name, email, phone) has to be identified.

ESA is asked to forward these contact details to the CDSE provider with the explicit authorization to activate the account.

The CDSE provider confirms, sets up the account and provides the MS contact with the relevant details. The CDSE provider has to allocate credits to the account, so that resources can be used (needs to be confirmed!)

VM set up

A tenant VM is not set up with account creation. This needs to be done via the Horizon/Openstack GUI after login)

If done by MS (see Essential steps below):

OPTIONAL - For hands-on support to the Member State from JRC:

- JRC needs to have ssh access to the account. JRC provides the public key that is used on it's own CDSE resources.
- The public key needs to be appended to the ~/.ssh/authorized_keys file on the MS VM.
- JRC to confirm ssh access.

Essential steps

Create SSH key pair during VM creation. Copy the **PRIVATE** key to a local `~/.ssh/keys/{name}.key` file (`chmod 0600 {name}.key`). The PUBLIC key will be copied to the new VM instance.

Select a machine type that can handle extraction on a single box (KISS). An 8 vCPU with 16 GB RAM is OK to run parallel extraction for cases with up to 1 million features, parallel to the database server.

Add a volume (for db). See detailed instructions on the CREODIAS FAQ page.

The VM needs to be created with sufficient disk space to run the database, you can also create a separate volume and attach it to the VM. Rule of thumb: 200 GB disk space for 1 Million parcels for one year (all S1 and S2).

We assume mount volume is at /data

Create a new Security group that allows Ingress on a non-standard port (e.g. 11039) and add to instance. This is a simple security provision to complicate port specific hacks.

Configure VM

On any new VM start with:

```
sudo apt update && sudo apt upgrade -y
sudo apt install vim
```

(cannot be run unattended if kernel update!!).

vi is needed for minimal text file editing.

Docker install (also on any VM if parallel work in expected):

```
sudo apt install docker.io
sudo usermod -aG docker $USER
```

Logout and login to take effect.

Installing the database container

PostgreSQL/postgis will run in a docker container.

Since the database needs access to a large volume, this requires a re-direction of the docker repository (normally on `/var/lib/docker`) to `/data/docker`

Create docker volume on mounted disk partition. See instructions

```
sudo service docker stop
sudo vi /etc/docker/daemon.json
```

Add the following in daemon.json

```
{
  "data-root": "/data/docker"
}
```

and

```
sudo rsync -aP /var/lib/docker/ /data/docker
sudo mv /var/lib/docker /var/lib/docker.old
sudo service docker start
```

Check whether docker runs OK. If so:

```
sudo rm -Rf /var/lib/docker.old
```

First create a volume for the database and pull a postgis image

```
docker volume create database
docker pull mdillon/postgis:latest
```

Do not use `kartoza/postgis`, because it leads to a convoluted installation, with 200 MB of useless locales. Start a container using the `mdillon/postgis image`:

```
docker run --name ams_db -d --restart always -v database:/var/lib/postgresql --shm-size=2gb -p 11039:5432 mdi
```

The port redirect (-p 11039:5432) uses the port that was opened at VM installation.

You need to login into the new container to set the `pg_hba.conf` to allow connections via the network. You are sudo inside the container.

```
docker exec -it ams_db /bin/bash

apt update
apt upgrade -y
apt install vim
vi /var/lib/postgresql/data/pg_hba.conf
```

Change all permissions from `trust` to `md5`, except the local ones.
Network connection already set to '*' in `postgresql.conf`.

The changes to the (running) container have to be committed to an updated image (so as to not loose the updates).

On the host VM, the following sequence will:

- list the running containers (`mdillon/postgis` is in daemon mode)
- commits the updated container to a new image
- list all images (which should show the new one next to the `mdillon/postgis:latest`)
- tags the committed image with the name `mdillon/postgis:updated`
- removes the running `mdillon/postgis:latest` daemon container, making sure the mounted database volume is safely unmounted
- starts the updated `mdillon/postgis:updated` as a new daemon container, using the database volume and the same db name

```
docker ps -a
docker commit <container-id>
docker images
docker tag <image-id> mdillon/postgis:updated
docker rm -vf ams_db
docker run --name ams_db -d --restart always -v database:/var/lib/postgresql --shm-size=2gb -p 11039:5432 mdi
```

Configure the database

Since the VM itself has no postgresql clients installed (yet), access the database from within the new container. Inside the container, you are a local user, so do not need a password. Remove the pre-installed schema that come with `mdillon/postgis` and set a password for the postgres user.

```
docker exec -it ams_db /bin/bash
psql -U postgres
```

```
DROP extension tiger;
DROP SCHEMA tiger;
ALTER ROLE postgres with password 'YOURPASSWORD';
```

One can now communicate with the postgresql database from external machines, but only with the password.

Stuff the GSAA shape file

Upload the parcels polygons file to the database. This can be done from a remote machine. Make sure to pass in the spatial reference system code and default MULTIPOLYGON type.

Test with QGIS connection to database.

```
ogr2ogr -f "PostgreSQL" PG:"host=YOURIP port=YOURPORT dbname=postgres user=postgres password=YOURPASSWORD" -n
```

Set up required tables

The aois table holds the definition of the Area Of Interest, which is the outline of the area for which CARD data needs to be processed. This can be generated from the extent of the parcel set.

```
CREATE TABLE aois (name text);
SELECT addgeometrycolumn('aois', 'wkb_geometry', 4326, 'POLYGON', 2);
INSERT into aois values ('parcels_2021', (SELECT st_transform(st_setsrid(st_extent(wkb_geometry), (SELECT Fin
```

Metadata for CARD data needs to be transferred from the CDSE catalog to the database. The cross-section of parcels and CARD data sets is stored in the hists table (cloud-occurence statistics for Sentinel-2 L2A) and sigs table (all bands extracts for Sentinel-2 L2A, Sentinel-1 CARD-BS and Sentinel-1 CARD-COH6).

Create the tables:

```
CREATE TABLE public.dias_catalogue (
    id serial NOT NULL,
    obstime timestamp without time zone NOT NULL,
    reference character varying(120) NOT NULL,
    sensor character(2) NOT NULL,
    card character(2) NOT NULL,
    status character varying(24) DEFAULT 'ingested'::character varying NOT NULL,
    footprint public.geometry(Polygon,4326)
);

ALTER TABLE ONLY public.dias_catalogue
    ADD CONSTRAINT dias_catalogue_pkey PRIMARY KEY (id);

CREATE INDEX dias_catalogue_footprint_idx ON public.dias_catalogue USING gist (footprint);

CREATE UNIQUE INDEX dias_catalogue_reference_idx ON public.dias_catalogue USING btree (reference);

CREATE TABLE public.sigs (
    pid integer,
    obsid integer,
    band character(3),
    count real, mean real, std real,
    min real, max real,
    p25 real, p50 real, p75 real
);

CREATE INDEX sigs_bidx ON public.sigs USING btree (band);
CREATE INDEX sigs_obsidx ON public.sigs USING btree (obsid);
CREATE INDEX sigs_pidx ON public.sigs USING btree (pid);
```

```
CREATE TABLE public.hists (
    pid integer,
    obsid integer,
    hist json
);

CREATE INDEX hists_obsidx ON public.hists USING btree (obsid);
CREATE INDEX hists_pidx ON public.hists USING btree (pid);
```

Setting up extraction

Extraction logic is coded in ***python 3*** compatible scripts.

Build dias_numba_py image

All python dependencies for fast extraction are packaged in the `dias_numba_py` docker image. Thus, all extraction routine will be run from within a derived container. The docker build command builds Docker images from a Dockerfile, the Dockerfile is available in the docker/dias_numba_py folder.

To build the image from the Dockerfile run:

```
cd docker/dias_numba_py
docker build -t dias_numba_py .
```

as an alternative, you can get the configured docker directly:

```
docker pull glemoine62/dias_numba_py
```

git install catalog and extraction code

The latest extraction code is currently on the **PUBLIC** gt4cap extraction repository. It will migrate to the public ec-jrc/cbm repository at some point.

Install the code on the new VM

```
mkdir cbm
cd cbm
git init
git remote add origin https://github.com/gt4cap/extraction.git
git pull origin main
```

Transfer records from datahub.creodias.eu

NB. THIS IS A MAJOR UPDATE, REFLECTING THE MIGRATION TO CDSE IN SPRING 2023

The metadata of Sentinel CARD needs to be transcribed into the dias_catalogue table. This is done via scripts that parse the JSON output of the OpenSearch requests to the CDSE catalog for the respective S-2 and S-1 data sets.

The key differences with previous the CREODIAS catalog are related to the change in URL of the catalog server and the query response parsing as json (was xml). No that the CREODIAS catalog is no longer up to date and will likely be discontinued in the near future.

Preparation

The script assumes that the geometry of the area of interest (aoi) is defined in the project database table `aois`. This table uses EPSG:4326 as the default projection.

aoi definition can be imported via standard procedures (e.g. `ogr2ogr`) or directly inserted via the psql client. An example for Denmark, using a GeoJSON formatted bounding polygon is below:

```
insert into aois values ('dk', st_setsrid(st_geomfromgeojson(
  $${"type":"Polygon","coordinates":
  [[[7.994683223865975,54.79423424555831],
  [12.543023067615975,54.540088425896286],
  [15.377495723865975,54.996407489112244],
  [15.025933223865975,55.38523613348353],
  [13.191216426990975,55.235171965073455],
  [11.158745723865975,58.08938516937467],
  [7.434380489490975,56.84285231095577],
  [7.994683223865975,54.79423424555831]]]}$$), 4326));
```

(NB. we use st_setsrid explicitly so that it does not throw errors for postgis version before 3.0)

Script parameters:

`cdse.py` reads the database connection details from `db_config.json`. Edit the parameters to reflect your database set up.

```
{
    "database": {
        "connection": {
            "host": "YOURIP",
            "dbname": "postgres",
            "dbuser": "postgres",
            "dbpasswd": "YOURPASSWORD",
            "port": YOURPORT
        },
        "tables": {
            "aoi_table": "aois",
            "catalog_table": "dias_catalogue"
        },
        "args": {
            "aoi_field": "name"
        }
    }
}
```

Calling syntax:

```
python cdse.py aoi start end card ptype|plevel
```

all parameters are **required**:

- *aoi*: the name of the aoi in database table `aois`
- *start*, *end*: YYYY-MM-dd formatted start and end date of period
- *card*: CARD type: one of {bs | c6 | c12 | s2}
- *ptype*|*plevel*: product type (s1) or processing level (s2) and one of {CARD-BS | CARD-COH6 | CARD-COH12} for s1 and {S2MSI1C | S2MSI2A} for s2

Example:

```
python cdse.py dk 2023-03-01 2023-04-01 s2 S2MSI2A
```

will get the S-2 Level 2A metadata. Note that the datahub.creodias.eu response can have a **maximum of 2000 records** per request. This means that the selection parameters need to be tuned to return less than 2000 records. Since the aoi is fixed, this can only be done by limiting the start and end date parameters.

For DK, run the Sentinel-2 selection over 3 months periods (for the entire date range). Check whether less than 2000 records are found (if 2000, adapt the range to a shorter period).

For Sentinel-1 the total number of scenes is typically lower (much larger footprints), so longer periods can be used.

Output

Will list insert statements for catalog entries found that are executed on the database. Duplicate entries (with same `reference`) will be dropped.

Will list summary of all metadata entries for aoi.

```
['card', 'sensor', 'count', 'min', 'max']
('bs', '1A', 320) 2020-10-02 05:32:37 2021-02-19 17:17:31
('bs', '1B', 346) 2020-10-01 05:40:09 2021-12-16 16:28:13
('c6', '1A', 181) 2020-10-02 05:32:35 2021-12-28 17:17:38
('c6', '1B', 156) 2020-10-01 05:40:08 2020-12-31 16:44:48
('s2', '2A', 3341) 2020-10-01 10:30:31 2021-12-31 10:54:41
('s2', '2B', 3392) 2020-10-02 10:47:59 2021-12-30 10:33:39
```

Currently, all Sentinel-2 L2A is directly available in the CDSE catalog, but only a subset of Sentinel-1 CARD-BS and CARD-COH6, since no specific order have been made (the available scenes are "spill over" for other actions run on CDSE).

Check which UTM projections are in the CARD data sets

Parcel extraction uses rasterized versions of the parcel features. These need to be generated (once) for the UTM projections and resolutions (10, 20 m) of the CARD data. The UTM projections over the AOI can be retrieved from the Sentinel-2 L2A image names, which are stored as reference in the dias_catalogue

```
SELECT distinct substr(split_part(reference, '_', 6),2,2)::int FROM dias_catalogue, aois WHERE footprint && w
```

(returns 32 and 33)

Create raster versions of the parcels in required projection and resolution

```
CREATE TABLE parcels_2021_32632_10_rast as (SELECT ogc_fid pid, st_asraster(st_transform(wkb_geometry, 32632)
alter table parcels_2021_32632_10_rast add primary key(pid);
```

(repeat for every other combination of UTM projection and resolution)

There is no need for a spatial index on the rast field, because all spatial selection are done on the original parcel feature table.

Make sure that the table naming convention {aoi}*{UTM}*{res}_rast is strictly adhered to (will be used in extraction code).

Houston, everything ready for extraction!

Running extraction

Extraction code is on cbm/extraction. The directory data is needed for temporary storage of VRT files:

```
cd cbm/extraction
mkdir data
```

Single runs

Executables take their parameters from a configuration file (db_config.json). Note that database tables need to specify the schema. The docker section defines the address of the swarm master.

```
{
    "database": {
        "connection": {
            "host": "YOURIP",
            "dbname": "postgres",
            "dbuser": "postgres",
            "dbpasswd": "YOURPASSWORD",
            "port": YOURPORT
        },
        "tables": {
            "aoi_table": "public.aois",
            "parcel_table": "public.parcels_2021",
            "catalog_table": "public.dias_catalogue",
            "sigs_table": "public.sigs",
            "hists_table": "public.hists"
        },
        "args": {
            "aoi_field": "name",
            "name": "parcels_2021",
            "startdate": "2020-10-01",
            "enddate": "2022-01-01"
        }
    },
    "docker": {
        "masterip": "192.168.0.8"
    }
}
```

For Sentinel-2 L2A, the order of execution is to first extract the SCL histograms and then the 10 m and 20 m band extracts. The SCL histogram run identifies S2 granules that have no parcels. These are then excluded in successive runs.

The order of the Sentinel-1 CARD-BS and CARD-COH6 runs is not important.

A single histogram extraction run is executed as follows:

```
docker run -it --rm -v`pwd`:/usr/src/app -v/eodata:/eodata -v/1/DIAS:/1/DIAS dias_numba_py python factoredWin
```

For the 10 m band run, change the argument -1 to 10, for the 20 m band run, change to 20.

For S1 CARD-BS ans CARD-COH6 extraction use the arguments `bs 10` and `c6 20` respectively.

Note that **BOTH** the /eodata and /1/DIAS volumes need to be mounted by the container. The latter is a local cache where all data read from S3 is stored. This needs to be cleared after each run (done inside the code).

Extraction manipulates the status field of the dias_catalogue, as follows:

- After catalogue transfer all new images are marked with status 'ingested'.
- At the start of extraction an image candidate will be marked as 'inprogress'.
- If no parcels are found in the image footprint, status is changed to 'No parcels'.
- If extraction fails (for a variety of reasons) status is changed to a meaningful error status (e.g. 'No in db', 'Rio error', 'Parcel SQL') which can be traced to the relevant code fragment.
- If extraction completes without error for a set of parcels, status is changed to 'extracted'.
- Extraction exits after a particular status (other than 'inprogress') is reached.

Check status statistics in the database:

```
SELECT card, status, count(*) from dias_catalogue group by card, status;
```

docker stack runs

In order to benefit from parallel processing, the easiest way is to use docker stack, even on a single machine (multiple VM swarms are for later).

The stack requires a docker compose configuration as follows:

```
version: '3.5'
services:
  vector_extractor:
    image: dias_numba_py:latest
    volumes:
            - /home/eouser/cbm/extraction:/usr/src/app
            - /eodata:/eodata
            - /1/DIAS:/1/DIAS
    deploy:
      replicas: 4
    command: python factoredWindowedExtraction.py s2 -1
```

and can be deployed as follows:

first initialize swarm and set current node as manager

```
docker swarm init
```

Then run

```
docker stack deploy -c docker-compose_scl.yml scl
```

This will start 4 parallel processes that run the histogram extraction.

You can check the output of the processes with:

```
docker service ls
docker service logs -f scl_vector_extractor
```

The first command list the running services, the second log details of the extraction processes.

A drawback of running a stack (in this context) is that, after all ingested images are processed, the stack needs to be terminated manually. It will simply continue to launch processes that will not find 'ingested' candidates and exit.

The stack can be removed as follows:

```
docker stack rm scl
```

python_on_whales runs

A solution to the stack termination issue is to integrate stack deployment with logic that can tear down the stack after checking 'ingested' status in the database. This can, in principle, be done in the bash shell (e.g. running a background process). A slightly more elegant solution is with python_on_whales, which controls docker processes from within python.

A drawback is the need to install some python modules on the VM:

```
sudo apt install python3-pip
pip3 install psycopg2-binary
pip3 install python_on_whales
```

The script `pow_extract_s2.py` runs the Sentinel-2 L2A extraction in a series of 3 stack deployments. The script checks the database for 'ingested' candidates at regular intervals and removes the relevant stack when exhausted. For successive runs, it will set the 'extracted' candidates to 'ingested' and starts the next stack.

Run as a background process (so you can log out of the VM):

```
nohup python pow_extract_s2.py &
```

The `pow_extract_s1.py` is the equivalent for Sentinel-1 CARD extraction.

Post extraction checks

Extraction may run for several days, depending on archive size and number of parcel features. A full list of status flags and their meaning and possible remedy is as follows:

Status	Meaning	Remedy
ingested	the image has not been processed	run extraction
extracted	the image has been successfully processed	
inprogress	the image is currently being processed or failed	see below
Rio error	an error occurred when reading the image	most likely cause is that the image does not cover the parcel selection (this is actually a bug)
!S3 VV.img	the VV image is not found on /eodata	check /eodata

Status	Meaning	Remedy
No in db	the connection to the database for read could not be made	check database run configuration
No out db	the connection to the database for write could not be made	check database run configuration
No extent	the extent of the parcel selection could not be generated	check parcel table in database
Parcel SQL	a query was made on a rasterized parcel table that does not exist	check dias_catalogue for images which have a projection that was not prepared during rasterization. Check if these are relevant for extraction and, if so, rasterize for the projection, reset status to 'inprogess' and rerun extraction
No parcels	the image has been processed, but no parcels are found inside its footprint	Do NOT delete these records because they will otherwise be re-ingested in a subsequent catalog search

At the end of the extraction run, some images may have been left in 'inprogress' status. First verify that the extraction process has completely finished (the 'inprogress' status DURING extraction is perfectly normal). Images are left in 'inprogress' status AFTER extraction if their processing suffered from memory overflow or a dropped database connection, though usually not a script error.

The 'inprogress' status may have been reached after a subset of parcels were already extracted. Thus, it is best to clean out the hists and sigs tables and redo the extraction. Save the 'inprogress' records to a separate table and use it to clean up.

```
CREATE TABLE faulties as (SELECT id FROM dias_catalogue WHERE status = 'inprogress' and card ='s2');

DELETE FROM hists WHERE obsid in (SELECT id FROM faulties);
DELETE FROM sigs WHERE obsid in (SELECT id FROM faulties);
UPDATE dias_catalogue set status = 'ingested' WHERE status = 'inprogress';
```

DO NOT USE the `pow_extract_s2.py` script, because it will reset ALL extracted status to ingested for the second stack run!!!

Instead, run as many individual runs as the number of 'inprogress' records. Use a bash script, if needed. Start with the SCL extraction to histograms.

After it has finished, reset:

```
UPDATE dias_catalogue set status = 'ingested' WHERE id in (SELECT id FROM faulties)
```

Run the extraction for 10 m bands, reset again, run the 20 m bands and wrap up:

```
drop table faulties;
```

Ordering S1 CARD data

Contrary to Sentinel-2, CDSE does not store any (Copernicus) Application Ready Data for Sentinel-1. S1-CARD needs to be processed on demand. A CDSE user has two options: (1) using the open source SNAP s1tbx with the specific recipes to generate CARD-BS and CARD-COH6 from S1-GRD and S1-SLC, respectively or (2) order CARD-BS and CARD-COH6 directly from the Processing-as-a-Service (PaaS) offered by the CDSE.

Although the first option provides more control, e.g. in the fine-tuning of s1tbx processing graphs, it requires significant compute resources and storage space. Option (2) is offered at a per-unit price, which includes the storage of the CARD output (for the duration of the project). The PaaS does the same as in option (1) but with flexible scaling of the compute resources.

Installing ordering scripts

The ordering procedure and relevant script are described for the CREODIAS PaaS.

Note that orders have to be created on the VM that is linked to the correct account, for billing.

```
mkdir creodias
cd creodias
git clone https://gitlab.cloudferro.com/bjaroszkowski/ordering_script.git
cd ordering_script/
```

This script installs a crontab task for (future) systematic daily orders. For now, it is better to switch this task off:

```
crontab -e
```

prepend a # to the line, so that it looks like this:

```
#0 0 * * * . /home/eouser/creodias/ordering_script/my_env.sh; /home/eouser/creodias/ordering_script/venv/bin/
```

Create a catalog request on finder.creodias.eu

Start with defining the query that lists all **Level 1 source data** for the CARD products. For CARD-BS this is GRD, for CARD-COH6 this is SLC.

Since March 2021, all Level 1 GRD data is produced with 'NRT-3h' timeliness. Before that date, 'Fast-24h' should be the default timeliness parameter. Make sure to select the IW sensor mode.

```
https://finder.creodias.eu/resto/api/collections/Sentinel1/search.json?maxRecords=10&startDate=2020-10-01T00%
```

For SLC selection, set productType=SLC and remove the timeliness=NRT-3h parameter.

Note that this query returns 807 records for GRD, and 737 for SLC.

Check for existing CARD-BS and CARD-COH6 by setting processingLevel=LEVEL2 and productType=CARD-BS and productType=CARD-COH6 (remove the timeliness parameter).

This will return 284 and 159, from CARD-BS and CARD-COH6 respectively. The order will only process the complement to the existing products.

Configure the ordering script input

The query has to be parsed into the `settings_historical.json` as follows:

For GRD:

```
{
  "platform": "creodias",
  "start_date": "2020-10-01",
  "final_date": "2022-01-01",
  "processor_names": ["card_bs"],
  "resto_queries": ["https://finder.creodias.eu/resto/api/collections/Sentinel1/search.json?maxRecords=200&pr
  "order_names": ["dk_2021_bs"]
}
```

For SLC

```
{
  "platform": "creodias",
  "start_date": "2020-10-01",
  "final_date": "2022-01-01",
  "processor_names": ["coh"],
  "resto_queries": ["https://finder.creodias.eu/resto/api/collections/Sentinel1/search.json?maxRecords=200&pr
  "order_names": ["dk_2021_c6"]
}
```

Launch the order, follow progress

```
export OS_USERNAME=YOURUSERNAME
export OS_PASSWORD=YOURPASSWORD
python historical_order.py
```

The credentials are needed to be enable the use of the ordering service. The script lists how many products results from the order (not yet corrected for existing products), and prompts the user to proceed with ordering.

The script ends with providing the link at which ordering progress can be reviewed.

```
python get_order_summary.py 1331871
Validation successful, getting order summary
{'status': 'processing', 'processing_order_items_count': 40, 'downloading_order_items_count': 0, 'done_order_
```

Revise settings_historical.json to run for the SLC card-coh6 selection:

```
python historical_order.py
Validation successful, proceeding with the order
Preparing order dk_2021_c6
Getting next result page...
Getting next result page...
Getting next result page...
Getting next result page...
Getting next result page...
You are about to make an order containing 831 products.
Do you want to review, proceed or abort? (r/p/a)p
Order dk_2021_c6 with ID 1331979 made successfully.
You can monitor progress of you order by calling https://finder.creodias.eu/api/order/1331979/summary
```

Maintenance

Deleting fully cloud covered parcel observations

If database size is an issue, one easy reduction step is to delete all sigs records for which parcels are fully cloud covered (SCL value 9):

```
WITH cnt_keys as (SELECT pid, obsid, (SELECT count(*) FROM jsonb_object_keys(hist::jsonb)) FROM hists WHERE h
SELECT 49879887
postgres=# SELECT count(*) FROM hists;
   count
-----------
 131393590
```

i.e. almost 38% of all sigs records can be removed (~ 21 GB) without significant loss of information (the cloud cover information is not removed).

Vacuum and cluster

The hists and sigs hold many millions of records after extraction. Both are indexed on the parcel id (pid) and observation id (obsid). The sigs table is also indexed on band. Since the typical access pattern is expected to be based on the parcel id, the tables are clustered on the pid index.

```
vacuum analyze hists;
cluster hists using (hists_pidx);
vacuum analyze sigs;
cluster sigs using (sigs_pidx)
```

This will take considerable time and should be done at the end of an extended extraction run. Clustering can only be performed if sufficient disk space is available as each table is rewritten in its entirety.