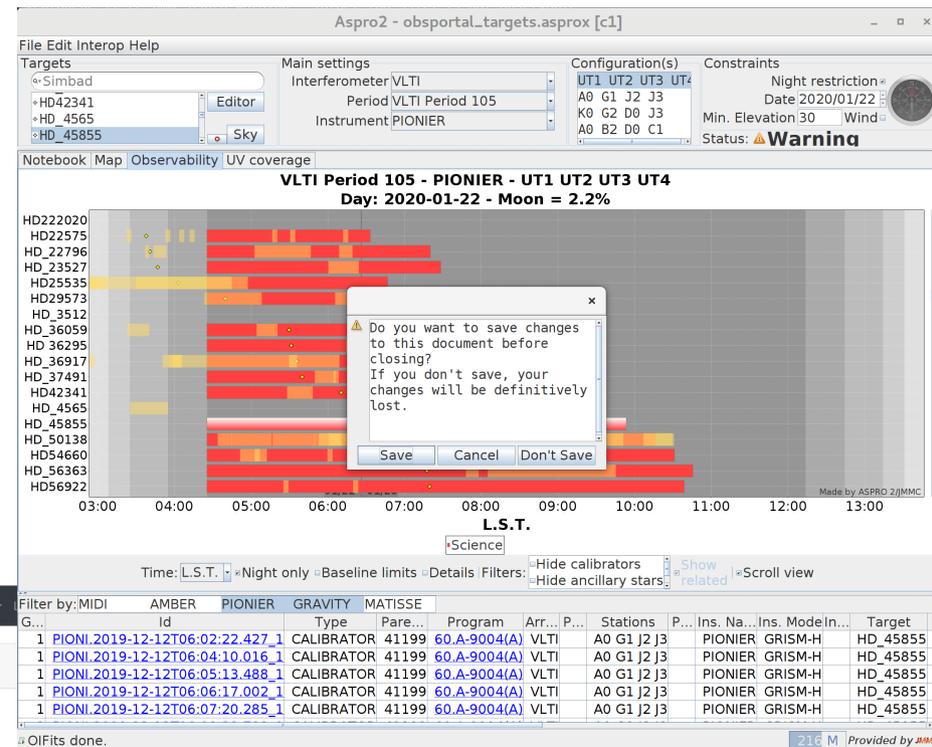


- Base de données des observations déjà réalisées en interférométrie (L0)
 - Préparation des demandes
 - Suivi des programmes
- Observation != acquisition (filtrage)
- Technique :
 - Dev Python + sqlalchemy + postgres
 - Déploiement : k8s (1 pod) + docker (service) : 

Utilisation

- Via ASPRO2
 - Point d'entrée principal
 - Demo
- Via l'interface Web
 - <http://obs.jmmc.fr>
- Interopérabilité :
cone search + VOTable
TAP (futur)



The screenshot shows the JMMC ObsPortal web interface. At the top, there are navigation links: Home, Search, Lists. Below that, there is a 'Search exposure' section with a 'Configuration' dropdown set to 'GRAVITY' and 'MEDIUM-SPLIT'. There are input fields for 'MJD From' and 'MJD To', and a 'Program ID' field. The 'Target' section has a 'Name' field and 'RA', 'DEC', and 'Radius' fields. A 'Search' button is at the bottom right. Below the search section, there is a table of search results. The table has columns for 'ID', 'Header', 'Instrument', 'Ins. Mode', 'Sub mode', and 'Target'. The table shows 1000 exposure(s) found. The first few rows are:

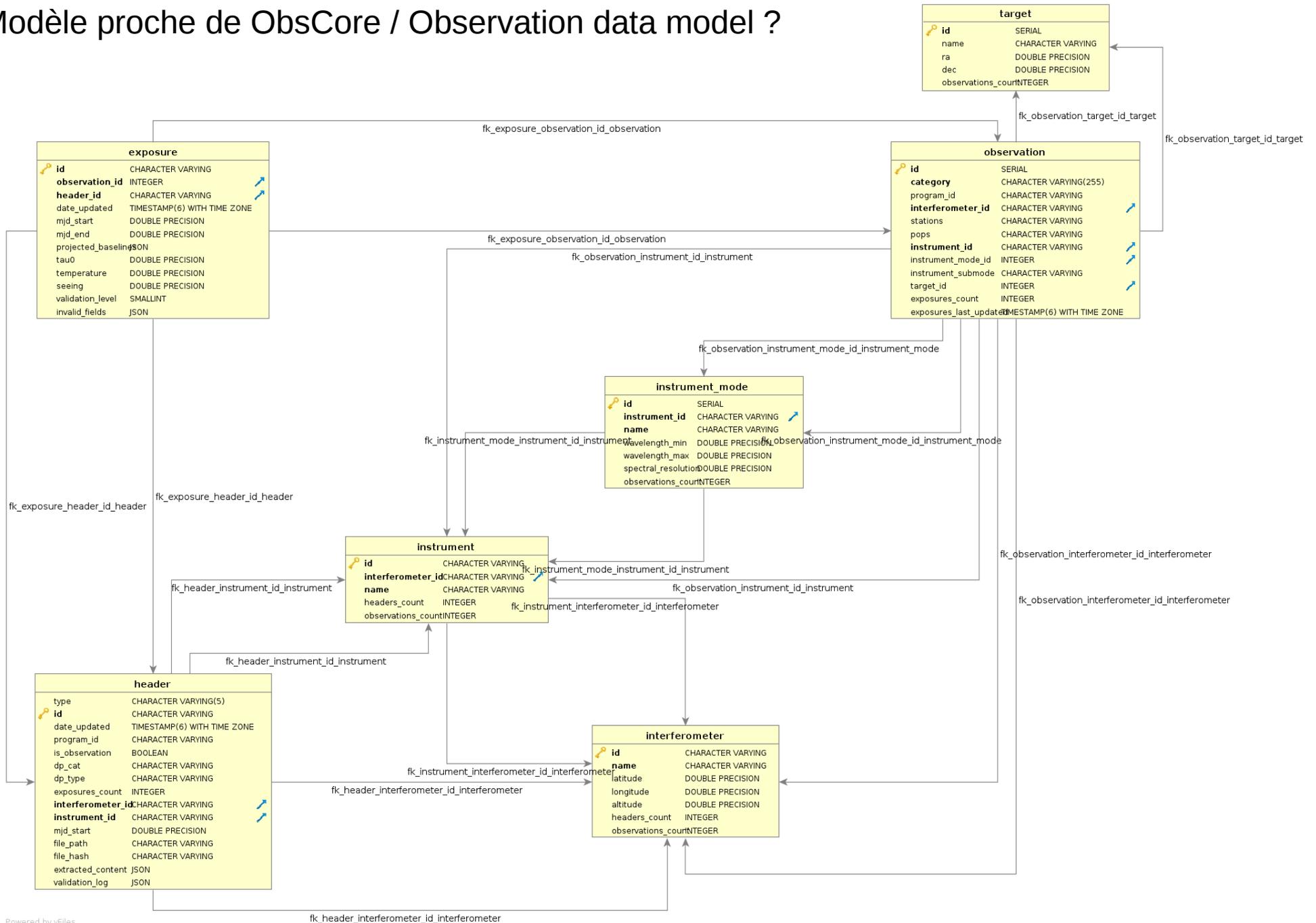
ID	Header	Instrument	Ins. Mode	Sub mode	Target
GRAVI.2020-01-06T02:34:38.064_1	GRAVI.2020-01-06T02:34:38.064	GRAVITY	MEDIUM-SPLIT	SINGLE	Name
GRAVI.2020-01-05T02:45:11.714_1	GRAVI.2020-01-05T02:45:11.714	GRAVITY	MEDIUM-SPLIT	DUAL	GJ65B
GRAVI.2020-01-05T02:38:35.697_1	GRAVI.2020-01-05T02:38:35.697	GRAVITY	MEDIUM-SPLIT	DUAL	GJ65B
GRAVI.2020-01-05T02:31:32.680_1	GRAVI.2020-01-05T02:31:32.680	GRAVITY	MEDIUM-SPLIT	DUAL	GJ65A
GRAVI.2020-01-05T02:28:17.672_1	GRAVI.2020-01-05T02:28:17.672	GRAVITY	MEDIUM-SPLIT	DUAL	GJ65A

Contenu

- Observations ESO qualifiées (300 000 exp)
 - Filtrage des headers (1M)
 - Science/Calibrator (Interferometry)
 - Propriétés extraites
 - Header ID, Program ID
 - RA/DEC + Target name
 - Instrument mode + submode
 - MDJ Start/End
 - Stations, Projected baselines
 - Seeing, tau0, temperature
- Observations CHARA (futur)
- Problème d'identification Targets RA/DEC
 - Topcat inner joins très lents avec votable gzip ?? 1h !
 - Essai de crossmatchs via postgres + pg sphere : 5s

DB Schema

Modèle proche de ObsCore / Observation data model ?



REX Astropy

- Pas d'API REST pour chercher dans l'archive ESO
 - Astroquery parse les pages HTML via BeautifulSoup
 - Quelle perennité ?
 - Inconsistance de l'API Astropy car trop proche des formulaires...
 - **TODO** : utiliser le service TAP fourni par l'ESO
- Backend VOTable reposant sur Numpy
 - Pas du tout efficace pour assembler la structure XML (file)
 - Approche « Quick and Dirty » semble plus rapide : print(f'') 5x plus rapide !
 - Pas de streaming possible !
 - Des idées ?

Index spatial (pg sphere)

- Create index (pg sphere)

```
create index oidb_spatial on oidb using  
GIST(spoint(radians(s_ra),radians(s_dec)));
```

- Utilisé pour toute requête de type cone search (par ex taplib / vollt)
- Gain :
 - Table size: 260 000 rows
 - Query Gain: $13056.12 / 929.76 = 14$

Index spatial (pg sphere)

```
explain select count(*) from oidb where spoint(radians(s_ra),radians(s_dec))  
@ scircle(spoint(radians(0),radians(0)),5e-7) ;
```

QUERY PLAN

Aggregate (cost=13056.11..13056.12 rows=1 width=8)

-> **Seq Scan on oidb** (cost=0.00..13055.44 rows=268 width=0)

Filter: (spoint(radians(s_ra), radians(s_dec)) @ '<(0 , 0) , 5e-07> '::scircle)

(3 rows)

Index spatial (pg sphere)

```
explain select count(*) from oidb where spoint(radians(s_ra),radians(s_dec))  
@ scircle(spoint(radians(0),radians(0)),5e-7) ;
```

QUERY PLAN

Aggregate (cost=929.75..929.76 rows=1 width=8)

-> Bitmap Heap Scan on oidb (cost=14.36..929.08 rows=268 width=0)

Recheck Cond: (spoint(radians(s_ra), radians(s_dec)) @ '<(0 , 0) , 5e-07> '::scircle)

-> **Bitmap Index Scan on oidb_spatial** (cost=0.00..14.29 rows=268 width=0)

Index Cond: (spoint(radians(s_ra), radians(s_dec)) @ '<(0 , 0) , 5e-07> '::scircle)

(5 rows)

Postgres : count(*) slow

- Select count(*) from table = slow
~ 100ms / Million rows

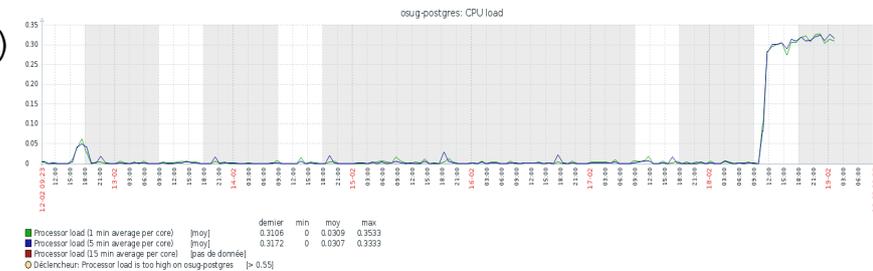
Voir <https://www.citusdata.com/blog/2016/10/12/count-performance/>

- Attention @aggregated() property :

```
@aggregated('observations', Column(Integer)
```

```
def modes_count(self):
```

```
    return func.count('1')
```



<https://sqlalchemy-utils.readthedocs.io/en/latest/aggregates.html>

- Solution : calculer les statistiques à la fin du processus d'import (Once)