

# The REG107\_PARTIES Table

Welcome to table **REG107\_PARTIES** in PATSTAT Register. This table contain details about applicants, inventors and agents / legal representatives.

Specifically, we can find information about the country, the name, and the address of the parties.

```
In [2]: from epo.tipdata.patstat import PatstatClient
        from epo.tipdata.patstat.database.models import REG107_PARTIES
        from sqlalchemy import func
        import pandas as pd

        # Initialise the PATSTAT client
        patstat = PatstatClient(env='PROD')

        # Access ORM
        db = patstat.orm()
```

## ID (Primary Key)

Technical identifier for an application, without business meaning. Its values will not change from one PATSTAT edition to the next.

```
In [3]: i = db.query(
        REG107_PARTIES.id
        ).limit(1000)

df = patstat.df(i)
df
```

Out [3]:

	id
0	10787181
1	9760944
2	10714106
3	7845261
4	4730083
...	...
995	109692
996	20735171
997	15908473
998	16720997
999	16194544

1000 rows × 1 columns

## IS\_LATEST

Y/N flag. Y indicates the latest, i.e. current or most recent, set of applicants / inventors / representatives / opponents.

```
In [7]: is_latest = db.query(
        REG107_PARTIES.id,
        REG107_PARTIES.is_latest
    ).limit(100)

is_latest_df = patstat.df(is_latest)
is_latest_df
```

Out[7]:

	id	is_latest
0	22924989	Y
1	21954135	Y
2	9156089	Y
3	21857126	Y
4	3700707	Y
...	...	...
95	4776185	N
96	98905870	Y
97	87109929	Y
98	11774292	Y
99	15751677	Y

100 rows × 2 columns

We can count how many applications in the database have the latest set of applicants / inventors / representatives / opponents.

```
In [12]: latest = db.query(
          REG107_PARTIES.is_latest,
          func.count(REG107_PARTIES.id).label('total_applications')
        ).group_by(
          REG107_PARTIES.is_latest
        ).order_by(
          func.count(REG107_PARTIES.id).desc()
        )

latest_df = patstat.df(latest)
latest_df
```

Out [12]:

	is_latest	total_applications
0	Y	30700223
1	N	9935611

## CHANGE\_DATE

It is the date of when the record was saved in the database.

```
In [4]: change_date = db.query(
        REG107_PARTIES.change_date,
        REG107_PARTIES.id
    ).limit(100)

change_date_df = patstat.df(change_date)
change_date_df
```

Out[4]:

	change_date	id
0	2009-03-20	7823702
1	2023-05-19	22892416
2	1985-05-18	85300098
3	2011-09-02	9799085
4	1995-07-07	94114808
...	...	...
95	2011-04-08	10190489
96	2001-05-18	204587
97	2021-06-11	19749918
98	2022-09-23	21931324
99	2003-10-10	96104114

100 rows × 2 columns

## BULLETIN\_YEAR

For actions that have been published in the EPO Bulletin, it is the year of the publication in the bulletin. The default value is 0, used for applications that are not published or for which the year is not known. The format is YYYY otherwise.

```
In [5]: years = db.query(
        REG107_PARTIES.bulletin_year,
        REG107_PARTIES.id
    ).limit(1000)

years_df = patstat.df(years)
years_df
```

Out[5]:

	bulletin_year	id
0	2015	13858370
1	2001	401917
2	2001	99916684
3	1991	90312346
4	2016	10189501
...	...	...
995	2020	17832629
996	0	4001600
997	0	7869625
998	2015	13804990
999	0	13851599

1000 rows × 2 columns

## BULLETIN\_NR

This is the issue number of the EPO Bulletin for actions that have been published in it. The Bulletin number indicates the calendar week the Bulletin has been published. The default value 0 is used when the attribute `bulletin_year` is 0.

```
In [6]: bulletin_nr = db.query(
        REG107_PARTIES.id,
        REG107_PARTIES.bulletin_nr,
        REG107_PARTIES.bulletin_year
    ).limit(100)

bulletin_nr_df = patstat.df(bulletin_nr)
bulletin_nr_df
```

Out[6]:

	id	bulletin_nr	bulletin_year
0	10787181	0	0
1	9760944	0	0
2	10714106	0	0
3	7845261	0	0
4	4730083	4	2006
...	...	...	...
95	15866299	0	0
96	96202826	0	0
97	16913416	8	2024
98	1939197	0	0
99	22916817	0	0

100 rows × 3 columns

## TYPE

Type of party ("A" for applicant / "I" for inventor / "R" for legal representative).

Let's see what is the most common type of party.

```
In [11]: type_party = db.query(
    REG107_PARTIES.type,
    func.count(REG107_PARTIES.id).label('number_of_applications')
).group_by(
    REG107_PARTIES.type
).order_by(
    func.count(REG107_PARTIES.id).desc()
)

type_party_df = patstat.df(type_party)
type_party_df
```

Out [11]:

	type	number_of_applications
0	I	21280446
1	A	10809571
2	R	8545817

## WISHES\_TO\_BE\_PUBLISHED

An inventor can agree to waive his entitlement for designation. The EP register then does not publish name or address of those inventors. Nevertheless, it is explicit that there is an inventor, even if name and address are not given. Therefore, this attribute indicates that the inventor has agreed to waive his entitlement for designation.

The corresponding field is either empty or 'N'.

```
In [13]: wish = db.query(
    REG107_PARTIES.wishes_to_be_published,
    func.count(REG107_PARTIES.id).label('number_of_applications')
).group_by(
    REG107_PARTIES.wishes_to_be_published
).order_by(
    func.count(REG107_PARTIES.id).desc()
)

wish_df = patstat.df(wish)
wish_df
```

Out [13]:

	wishes_to_be_published	number_of_applications
0		40621886
1	N	13948

## SEQ\_NR

Sequence number of party (concerns applicant, inventor and legal representative). All applicants (one or more) of an application at a certain point of time are regarded as a “set” of applicants. The `seq_nr` defines the order of applicants within this set. The same applies to inventors and legal representatives (see attribute `type` ).

```
In [2]: seq = db.query(
        REG107_PARTIES.id,
        REG107_PARTIES.seq_nr
      ).limit(1000)

seq_df = patstat.df(seq)
seq_df
```

Out [2]:

	id	seq_nr
0	7301295	1
1	6812357	1
2	2720260	1
3	5738609	6
4	19915557	1
...	...	...
995	6705209	1
996	21819148	1
997	9002520	1
998	89104183	1
999	9738238	1

1000 rows × 2 columns

## DESIGNATION

Type of designation. Default value apart, i.e. empty string, this attribute can assume two values: 'all' and 'as-indicated'.

```
In [3]: design = db.query(
        REG107_PARTIES.designation,
        func.count(REG107_PARTIES.id).label('number_of_applications')
    ).group_by(
        REG107_PARTIES.designation
    ).order_by(
        func.count(REG107_PARTIES.id).desc()
    )

design_df = patstat.df(design)
design_df
```

Out [3]:

	designation	number_of_applications
0		29826263
1	all	9699014
2	as-indicated	1110557

## CUSTOMER\_ID

The identifier of a customer, which can be a legal or natural person. If a customer has several roles (applicant, inventor, representative, licensee, opponent) he usually has several `customer_id` s.

Not every person has a `customer_id` assigned. Persons which are only inventors (and not e. g. also applicants) do not have a `customer_id` .

```
In [9]: customer = db.query(
        REG107_PARTIES.customer_id
        ).limit(1000)

customer_df = patstat.df(customer)
customer_df
```

Out [9]:

	customer_id
0	0100060488
1	0101257739
2	
3	17105408348728179
4	0101359935
...	...
995	1 00151060
996	
997	
998	
999	

1000 rows × 1 columns

## NAME

Name of a person (natural person or legal person).

```
In [14]: name = db.query(
          REG107_PARTIES.name
        ).limit(100)

name_df = patstat.df(name)
name_df
```

Out [14]:

	name
0	Nexans
1	CHEONG, Cheol-Ho
2	Nokia Corporation
3	RINCON, Sergio
4	LI, Yong
...	...
95	CHIDDICK, Kelvin, Spencer
96	SONY CORPORATION
97	Asahi Glass Company, Limited
98	Rolls-Royce plc
99	MAEDA, Yasuhiro

100 rows × 1 columns

## ADDRESS\_1

First address line of a person. Default value: empty. In PATSTAT Online these attributes are not populated.

```
In [4]: addr1 = db.query(
        REG107_PARTIES.id,
        REG107_PARTIES.name,
        REG107_PARTIES.address_1
    ).limit(100)

addr1_df = patstat.df(addr1)
addr1_df
```

Out [4]:

	id	name	address_1
0	13858370	Hiebl, Inge Elisabeth	Kraus & Weisert
1	401917	Girin, Lionel, Thomson-CSF P.I. Dépt. Brevets	13, av. du Président Salvador Allende
2	99916684	MARIANO, Richard	140 Codfish Hill Road
3	90312346	Pettit, George R.	6232 Bret Hills Drive
4	10189501	Ulupinar, Faith	5775 Morehouse Drive
...	...	...	...
95	6714808	AKUTA, Teruo	1-1, Honjo 1-chome, Kumamoto-shi
96	12825370	OrthoGrid Systems S.a.r.l.	40 Avenue Monterey
97	15723560	Schwan Schorer & Partner mbB	Patentanwälte
98	22876744	MURGAI, Vishal	Bangalore 560037
99	23832093	OKAMURA, Susumu	San Jose, California 95119

100 rows × 3 columns

## ADDRESS\_2

Second address line of a person. Default value: empty. In PATSTAT Online these attributes are not populated.

```
In [5]: addr2 = db.query(
        REG107_PARTIES.id,
        REG107_PARTIES.name,
        REG107_PARTIES.address_2
    ).limit(100)

addr2_df = patstat.df(addr2)
addr2_df
```

Out [5]:

	id	name	address_2
0	7301295	Nexans	75008 Paris
1	6812357	CHEONG, Cheol-Ho	Seoul 120-113
2	2720260	Nokia Corporation	02150 Espoo
3	5738609	RINCON, Sergio	Manizales, Caldas
4	19915557	LI, Yong	Qingdao Shandong 266000
...	...	...	...
95	97939912	CHIDDICK, Kelvin, Spencer	N. Vancouver, British Columbia V7N 2S6
96	4736466	SONY CORPORATION	Tokyo 141-0001
97	10761728	Asahi Glass Company, Limited	1-5-1 Marunouchi
98	17742395	Rolls-Royce plc	90 York Way
99	17773855	MAEDA, Yasuhiro	

100 rows × 3 columns

## ADDRESS\_3

Third address line of a person. Default value: empty. In PATSTAT Online these attributes are not populated.

From `address_3` onwards, there are not many applicants with an  $n$ -th address, with  $n > 3$ . Therefore, we filter the addresses that do not consist of an empty string.

```
In [3]: addr3 = db.query(
        REG107_PARTIES.id,
        REG107_PARTIES.name,
        REG107_PARTIES.address_3
    ).filter(
        REG107_PARTIES.address_3 != ''
    )

addr3_df = patstat.df(addr3)
addr3_df
```

Out [3]:

	id	name	address_3
0	1203283	Donné, Eddy	2000 Antwerpen
1	14869526	ORIBE, Akinobu	Mizuho-ku
2	18178502	Iglesias, Eva Rajo	28911 Leganés (Madrid)
3	13717201	GARRIC, Xavier	Faculté de Pharmacie
4	11190680	Spina, Alessandro, et al	20016 Pero (MI)
...	...	...	...
9406340	17747187	KATAGIRI, Takashi	Minato-ku
9406341	18775318	YOSHIDA Kenichi	Minato-ku
9406342	10713407	HAYASHI, Yoshiyuki	Minato-ku
9406343	14800667	MAENO, Yoshiharu	Minato-ku
9406344	12782430	YOSHIDA, Tsuyoshi	Minato-ku

9406345 rows × 3 columns

## ADDRESS\_4

Forth address line of a person. Default value: empty. In PATSTAT Online these attributes are not populated.

```
In [4]: addr4 = db.query(
        REG107_PARTIES.id,
        REG107_PARTIES.name,
        REG107_PARTIES.address_4
    ).filter(
        REG107_PARTIES.address_4 != ''
    )

addr4_df = patstat.df(addr4)
addr4_df
```

Out [4]:

	id	name	address_4
0	20800322	Schröer, Gernot H.	90402 Nürnberg
1	8835625	Jané Bonet, Montserrat	08008 Barcelona
2	18155830	Schlögl, Markus	90402 Nürnberg
3	16807567	KOLAR Johann W.	Zurich 8092
4	14847091	ZHANG, Guicai	Minhang District
...	...	...	...
4329097	18835973	AKITA, Masayoshi	Tokyo 108-0075
4329098	15815368	OOTSUKA, Wataru	Tokyo 108-0075
4329099	17724946	TAKAHASHI, Hiroo	Tokyo 108-0075
4329100	11828911	KITAZATO, Naohisa	Tokyo 108-0075
4329101	10000991	Sato, Noritaka	Tokyo 108-0075

4329102 rows × 3 columns

## ADDRESS\_5

Fifth address line of a person. Default value: empty. In PATSTAT Online these attributes are not populated.

```
In [5]: addr5 = db.query(
        REG107_PARTIES.id,
        REG107_PARTIES.name,
        REG107_PARTIES.address_5
    ).filter(
        REG107_PARTIES.address_5 != ''
    )

addr5_df = patstat.df(addr5)
addr5_df
```

Out [5]:

	id	name	address_5
0	16884905	FUTATSUYAMA, Takuya	Tokyo 105-8001
1	13790703	SHICHIJO, Shunichi	Yokohama-shi Kanagawa 221-0022
2	14847091	ZHANG, Guicai	Shanghai 200240
3	1916499	Makovski, Priscilla Mary, et al	B16 8QQ
4	14184692	Rao, Rachana Gurunandan	Morristown, NJ 07962-2245
...	...	...	...
1149550	12847531	MORITA, Takashi	Tokyo 100-6150
1149551	16861961	KIYOSHIMA, Kohei	Tokyo 100-6150
1149552	13740578	SHIMIZU, Ryouichi	Tokyo 100-6150
1149553	17782471	MOROGA, Hideyuki	Tokyo 100-6150
1149554	13867295	HANAKI, Akihito	Tokyo 100-6150

1149555 rows × 3 columns

## COUNTRY

Two-letter country/territory code for patent parties (applicant/inventor/agent), designated states of applicant, or country of licensees. Default value: empty. The domain consists of up to 2 alphabetic characters, according to WIPO ST.3, plus minor additions.

Let's check which are the most frequent countries/territories for patent parties.

```
In [15]: country = db.query(
    REG107_PARTIES.country,
    func.count(REG107_PARTIES.id).label('number_of_applications')
).group_by(
    REG107_PARTIES.country
).order_by(
    func.count(REG107_PARTIES.id).desc()
)

country_df = patstat.df(country)
country_df
```

Out [15]:

	country	number_of_applications
0	US	8912078
1	DE	7724223
2	JP	5688552
3	GB	3680422
4	CN	2765510
...	...	...
289	ja	1
290	Ja	1
291	Be	1
292	Dk	1
293	GS	1

294 rows × 2 columns

In [ ]: