TLS207_PERS_APPLN - The Link between Person and Application Table

Welcome to this detailed look at one of the most fundamental tables in the PATSTAT database: the **Link between Person and Application Table**, designated as `TLS207_PERS_APPLN` . This table serves as a crucial connection between the individuals (persons) involved in the patenting process, either as applicants or inventors, and their respective patent applications.

The `TLS207_PERS_APPLN` table is fundamental to understanding the relationships between people and patent applications, acting as a bridge to identify the role of each individual in the application process. This table provides a comprehensive view of the contributions of individuals (or entities) to patent applications, capturing the intricate relationships between inventors and applicants across multiple applications. It allows analysts to track how often individuals or companies are involved in patent filings, and to see the overlap between inventors and applicants.

The `TLS207_PERS_APPLN` table captures the **many-to-many relationship** between persons (applicants and inventors) and patent applications. It serves as an intermediary between Table `TLS201_APPLN` (Applications) and Table `TLS206_PERSON` (Persons), linking the details of individuals or legal entities involved in a patent application.

There are some key aspects regarding the table, one of them is the fact that a single patent application can list multiple inventors and multiple applicants. Applicants can be either physical persons (individuals) or legal entities (such as companies or organisations). Moreover, applicants and inventors can be the same person. A person can be associated with multiple applications, and a single application can involve multiple persons (both as applicants and inventors). It is possible for an individual to be both an applicant and an inventor on the same patent application, and this table captures that relationship.

In [1]:

```
from epo.tipdata.patstat import PatstatClient
from epo.tipdata.patstat.database.models import TLS207_PERS_APPLN, TLS201_APPLN
from sqlalchemy import func, case

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

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

Primary Key

The combination of `PERSON_ID`, `APPLN_ID`, `APPLT_SEQ_NR`, and `INVT_SEQ_NR` constitutes the **primary key** for this table, ensuring each link between a person and an application is unique.

Each part of the key plays an important role:

- `PERSON_ID` by itself is not sufficient because the same individual may be involved in multiple applications, either as an inventor, an applicant, or both.
- `APPLN_ID` alone does not provide enough distinction since a single application can be associated with multiple individuals, either as applicants or inventors.
- `APPLT_SEQ_NR` and `INVT_SEQ_NR` are essential because an individual may fulfill different roles (applicant or inventor) within the same application. These sequence numbers are necessary to differentiate between the two roles.

In [2]:

```
application_query = db.query(
    TLS207_PERS_APPLN.appln_id,          # Application ID
    TLS207_PERS_APPLN.person_id,         # Person ID
    TLS207_PERS_APPLN.applt_seq_nr,      # Applicant sequence number
    TLS207_PERS_APPLN.invt_seq_nr        # Inventor sequence number
).order_by(
    TLS207_PERS_APPLN.appln_id,          # Order by Application ID
    TLS207_PERS_APPLN.applt_seq_nr,      # Then by Applicant sequence number
    TLS207_PERS_APPLN.invt_seq_nr        # Then by Inventor sequence number
)

application_res = patstat.df(application_query)

application_res.head(20)
```

Out[2]:

	appln_id	person_id	applt_seq_nr	invt_seq_nr
0	145	538	1	1
1	146	540	0	1
2	146	541	0	2
3	146	542	0	3
4	146	539	1	0
5	186	637	0	1
6	186	638	0	2
7	186	636	1	0
8	287	837	0	1
9	287	4423581	1	0
10	620	1425	0	1
11	620	1426	0	2
12	620	1427	0	3
13	620	1424	1	0
14	1040	2326	0	1
15	1040	539	1	0
16	1042	2327	1	1
17	2701	44730722	0	1
18	2701	44730723	0	2
19	2701	44730724	0	3

This query allows us to examine the applications and the individuals associated with them. The `APPL_ID` can appear multiple times, as multiple `PERSON_IDs` may be linked to a single application, reflecting the involvement of several individuals. Each person is assigned both an `APPLT_SEQ_NR` and an `INVT_SEQ_NR` .

If the `APPLT_SEQ_NR` is greater than zero, it indicates that the individual is an applicant, and the sequence number reflects the order in which applicants are listed. For example, if an application has three applicants, the `APPLT_SEQ_NR` values will be 1, 2, and 3, respectively. Similarly, the `INVT_SEQ_NR` functions in the same way for inventors. If the `INVT_SEQ_NR` is greater than zero, the individual is listed as an inventor, with the sequence number reflecting the order. For instance, if there are five inventors, the sequence numbers will range from 1 to 5.

In some cases, a person may serve as both an applicant and an inventor. This dual role is identifiable when the individual has both an `INVT_SEQ_NR` and an `APPLT_SEQ_NR` greater than zero for the same application.

Key Fields in the `TLS207_PERS_APPLN` Table

APPLN_ID

Identifier for the patent application, representing the **formal request** for patent protection. This field is a foreign key that references the `appl_id` in the `TSL201_APPLN` table, establishing a link between an application and its related persons.

In [3]:
```
q = db.query(
     TLS201_APPLN.appln_nr,
    TLS207_PERS_APPLN.appln_id
).join(
    TLS201_APPLN, TLS207_PERS_APPLN.appln_id == TLS201_APPLN.appl
n_id
)

res = patstat.df(q)

res
```

Out[3]:

	appln_nr	appln_id
0	2008078434	57025341
1	55538009	275544314
2	201113374528	381603477
3	19494176	9163373
4	3729008	55068766
...	...	...
1325922	67755903	52691972
1325923	26750488	49151418
1325924	89886792	54049442
1325925	201320168773	409563943
1325926	201010171525	323114641

1325927 rows × 2 columns

PERSON_ID

Unique identifier for each individual (whether an applicant or inventor). This identifier is used to link the person (who could be an inventor or an applicant) to a patent application. This field is a foreign key that references the `person_id` in the `TSL206_PERSON` table, identifying the person associated with the application.

- Inventors: Individuals who contributed to the invention.
- Applicants: Individuals or entities (e.g., companies) who apply for the patent. The applicant can be the same as the inventor or a separate entity, such as a corporation.

In [4]:

```
q = db.query(
    TLS201_APPLN.appln_nr,
    TLS207_PERS_APPLN.appln_id,
    TLS207_PERS_APPLN.person_id
).join(
    TLS201_APPLN, TLS207_PERS_APPLN.appln_id == TLS201_APPLN.appl
n_id
)

res = patstat.df(q)

res
```

Out[4]:

	appln_nr	appln_id	person_id
0	2008078434	57025341	40585895
1	55538009	275544314	45175768
2	201113374528	381603477	11243775
3	19494176	9163373	20649701
4	3729008	55068766	10779314
...	...	...	...
1325922	67755903	52691972	6552866
1325923	26750488	49151418	7243397
1325924	89886792	54049442	10261899
1325925	201320168773	409563943	43609631
1325926	201010171525	323114641	18745393

1325927 rows × 3 columns

We will get multiple rows where the same `APPLN_ID` (application ID) may appear multiple times with different `PERSON_ID` values. This occurs because one patent application can involve multiple persons (applicants or inventors). Each person linked to the application will have their own `PERSON_ID` .

The reverse scenario is also true. A single `PERSON_ID` (person) can be associated with multiple `APPLN_IDs` (patent applications), meaning one person can be involved in several different patent applications.

It can be concluded that a **many-to-many relationships** work both ways: a single person (`PERSON_ID`) can be linked to multiple applications (`APPLN_ID`), and a single application can involve multiple persons.

APPLT_SEQ_NR

This is the **applicant sequence number** and indicates the order of the applicants associated with the application. If the `APPLT_SEQ_NR` is non-zero, it means the person is listed as an applicant for the corresponding application. Multiple applicants may exist for the same application, each assigned a unique sequence number to indicate their order.

In [5]:

```
q = db.query(
    TLS201_APPLN.appln_nr,
    TLS207_PERS_APPLN.appln_id,
    TLS207_PERS_APPLN.person_id,
    TLS207_PERS_APPLN.applt_seq_nr
).join(
    TLS201_APPLN, TLS207_PERS_APPLN.appln_id == TLS201_APPLN.appln_id
).order_by(
    TLS207_PERS_APPLN.appln_id
)

res = patstat.df(q)

res
```

Out[5]:

	appln_nr	appln_id	person_id	applt_seq_nr
0	07015055	145	538	1
1	07015148	146	539	1
2	07015148	146	540	0
3	07015148	146	541	0
4	07015148	146	542	0
...	...	...	...	...
1325922	17826528	606428353	75789263	0
1325923	17826528	606428353	78036309	0
1325924	17826528	606428353	79739008	0
1325925	17826528	606428353	58675258	0
1325926	17826528	606428353	40901487	0

1325927 rows × 4 columns

INVT_SEQ_NR

This is the **inventor sequence number** and indicates the order of inventors associated with the application. If the `INVT_SEQ_NR` is non-zero, it means the person is listed as an inventor for the corresponding application. Similarly, multiple inventors can exist for a single application, and each inventor is assigned a unique sequence number to indicate their order.

In [6]:

```
q = db.query(
    TLS201_APPLN.appln_nr,
    TLS207_PERS_APPLN.appln_id,
    TLS207_PERS_APPLN.person_id,
    TLS207_PERS_APPLN.applt_seq_nr,
    TLS207_PERS_APPLN.invt_seq_nr
).join(
    TLS201_APPLN, TLS207_PERS_APPLN.appln_id == TLS201_APPLN.appl
n_id
).order_by(
    TLS207_PERS_APPLN.appln_id
)

res = patstat.df(q)

res
```

Out[6]:

	appln_nr	appln_id	person_id	applt_seq_nr	invt_seq_nr
0	07015055	145	538	1	1
1	07015148	146	539	1	0
2	07015148	146	540	0	1
3	07015148	146	541	0	2
4	07015148	146	542	0	3
...	...	...	...	...	...
1325922	17826528	606428353	75789263	0	1
1325923	17826528	606428353	78036309	0	2
1325924	17826528	606428353	79739008	0	3
1325925	17826528	606428353	58675258	0	4
1325926	17826528	606428353	40901487	0	5

1325927 rows × 5 columns

Persons Connected to an Application

Now, let’s proceed with some exercises to explore the key aspects of this table. Suppose we want to determine the number of applicants and inventors for each application. To achieve this, we will utilise the **case** statement, which allows us to set conditions within the same query. This approach enables us to count both applicants and inventors simultaneously, providing a comprehensive overview of each application’s contributors.

In [7]:

```
count_query = db.query(
    TLS207_PERS_APPLN.appln_id,
    func.count(case([(TLS207_PERS_APPLN.applt_seq_nr != 0, TLS207_PERS_APPLN.person_id)])).label('num_applicants'),  # Count applicants,  'case' enables us to categorise counts based on specific criteria within the same query
    func.count(case([(TLS207_PERS_APPLN.invt_seq_nr != 0, TLS207_PERS_APPLN.person_id)])).label('num_inventors')   # Count inventors
).group_by(
    TLS207_PERS_APPLN.appln_id
)

count_res = patstat.df(count_query)

sorted_count_res = count_res.sort_values(by='num_applicants', ascending=False).reset_index(drop=True)
sorted_count_res
```

```
/tmp/ipykernel_23740/3506280912.py:3: RemovedIn20Warning: Depreca
ted API features detected! These feature(s) are not compatible wi
th SQLAlchemy 2.0. To prevent incompatible upgrades prior to upda
ting applications, ensure requirements files are pinned to "sqlal
chemy<2.0". Set environment variable SQLALCHEMY_WARN_20=1 to show
all deprecation warnings.  Set environment variable SQLALCHEMY_SI
LENCE_UBER_WARNING=1 to silence this message. (Background on SQLA
lchemy 2.0 at: https://sqlalche.me/e/b8d9)
  func.count(case([(TLS207_PERS_APPLN.applt_seq_nr != 0, TLS207_P
ERS_APPLN.person_id)])).label('num_applicants'),  # Count applica
nts,  'case' enables us to categorise counts based on specific cr
iteria within the same query
```

Out[7]:

	appln_id	num_applicants	num_inventors
0	353146079	24	23
1	405239188	23	0
2	3897780	22	20
3	54681446	20	19
4	332865143	20	18
...	...	...	...
318073	54408470	0	9
318074	51528990	0	9
318075	57137667	0	9
318076	458937104	0	9
318077	560850158	0	9

318078 rows × 3 columns

This query is designed to count the number of applicants and inventors associated with each patent application in the `TLS207_PERS_APPLN` table. It retrieves the application ID (`ÀPPLN_ID`) and performs conditional counts of both applicants and inventors by examining the applicant sequence number (`APPLT_SEQ_NR`) and inventor sequence number (`INVT_SEQ_NR`). Specifically, a person is counted as an applicant if the applicant sequence number is non-zero, and as an inventor if the inventor sequence number is non-zero.

Count of Applications by Role

Turning our attention to individuals, we can analyse the **frequency** with which each person is listed as an **applicant** and as an **inventor**. It is important to note that, typically, inventors are individuals rather than legal entities. In contrast, companies, organisations, and institutions are usually designated as applicants, and they may be listed alongside individual inventors in applications, who will also be considered applicants. This distinction is crucial for understanding the dynamics of patent applications and the roles that different entities play in the innovation process.

```
In [8]:
combined_query = db.query(
    TLS207_PERS_APPLN.person_id,
    func.count(case([(TLS207_PERS_APPLN.applt_seq_nr != 0, TLS207_PERS_APPLN.appln_id)])).label('num_applications_as_applicant'),
    func.count(case([(TLS207_PERS_APPLN.invt_seq_nr != 0, TLS207_PERS_APPLN.appln_id)])).label('num_applications_as_inventor')
).group_by(
    TLS207_PERS_APPLN.person_id
).order_by(
    func.count(case([(TLS207_PERS_APPLN.applt_seq_nr != 0, TLS207_PERS_APPLN.appln_id)])).desc()
)

combined_res = patstat.df(combined_query)

combined_res
```

Out[8]:

	person_id	num_applications_as_applicant	num_applications_as_inventor
0	5210554	2886	0
1	41936141	2272	0
2	5393568	2194	0
3	60028824	2047	0
4	40095798	1304	0
...	...	...	...
646338	53761116	0	1
646339	58214288	0	1
646340	58438277	0	1
646341	7243397	0	1
646342	10261899	0	1

646343 rows × 3 columns

This query counts the number of applications associated with each person, distinguishing between their roles as applicants and inventors.

Counts of Individuals as Both Applicants and Inventors

If we wanted to narrow our investigation, we could focus on individuals who serve as both inventors and applicants. To achieve this, we would build upon the previous query by adding conditions to filter the results. Specifically, we would identify those individuals in the database who are listed **as both inventors for certain applications and as applicants for others**, or even for the same applications. This approach allows us to examine the dual roles these individuals play within the context of patent applications.

In [9]:

```
applicant_inventor_query = db.query(
    TLS207_PERS_APPLN.person_id,
    func.count(case([(TLS207_PERS_APPLN.applt_seq_nr != 0, TLS207_PERS_APPLN.appln_id)])).label('num_applications_as_applicant'),
    func.count(case([(TLS207_PERS_APPLN.invt_seq_nr != 0, TLS207_PERS_APPLN.appln_id)])).label('num_applications_as_inventor')
).group_by(
    TLS207_PERS_APPLN.person_id
).having(
    (func.count(case([(TLS207_PERS_APPLN.applt_seq_nr != 0, TLS207_PERS_APPLN.appln_id)])) > 0) &
    (func.count(case([(TLS207_PERS_APPLN.invt_seq_nr != 0, TLS207_PERS_APPLN.appln_id)])) > 0)
).order_by(
    func.count(case([(TLS207_PERS_APPLN.invt_seq_nr != 0, TLS207_PERS_APPLN.appln_id)])).desc()  # Order by inventor count first
)

applicant_inventor_res = patstat.df(applicant_inventor_query)

applicant_inventor_res
```

Out[9]:

	person_id	num_applications_as_applicant	num_applications_as_inventor
0	60133150	9	378
1	65106873	3	369
2	65931398	14	337
3	59325860	1	311
4	59408599	3	311
...	...	...	...
74420	9258777	1	1
74421	86555810	1	1
74422	16483451	1	1
74423	12097446	1	1
74424	43609631	1	1

74425 rows × 3 columns

In [10]:

```
applicant_inventor_query = db.query(
    TLS207_PERS_APPLN.person_id,
    func.count(case([(TLS207_PERS_APPLN.applt_seq_nr != 0, TLS207
_PERS_APPLN.appln_id)])).label('num_applications_as_applicant'),
    func.count(case([(TLS207_PERS_APPLN.invt_seq_nr != 0, TLS207_
PERS_APPLN.appln_id)])).label('num_applications_as_inventor')
).group_by(
    TLS207_PERS_APPLN.person_id
).having(
    (func.count(case([(TLS207_PERS_APPLN.applt_seq_nr != 0, TLS20
7_PERS_APPLN.appln_id)])) > 0) &
    (func.count(case([(TLS207_PERS_APPLN.invt_seq_nr != 0, TLS207
_PERS_APPLN.appln_id)])) > 0)
).order_by(
    func.count(case([(TLS207_PERS_APPLN.applt_seq_nr != 0, TLS207
_PERS_APPLN.appln_id)])).desc()  # Then order by applicant count
)

applicant_inventor_res = patstat.df(applicant_inventor_query)

applicant_inventor_res
```

Out[10]:

	person_id	num_applications_as_applicant	num_applications_as_inventor
0	51208860	643	1
1	12691803	394	273
2	7390862	98	116
3	12640630	87	94
4	12691802	66	63
...	...	...	...
74420	9258777	1	1
74421	86555810	1	1
74422	16483451	1	1
74423	12097446	1	1
74424	43609631	1	1

74425 rows × 3 columns

These tables indicate that these individuals are associated with at least one role as either an applicant or an inventor in their respective patent applications. This could highlight individuals with diverse contributions to innovation. By ordering first by inventors and then by applicants, we can identify the most active individuals contributing to the innovation process, showcasing those who are heavily involved in both applying for and inventing patents.

This type of analysis could also be performed without using `CASE` statements by constructing separate DataFrames, which are stored in memory and then merged. By saving them into a DataFrame, they can be reused throughout the analysis, rather than rewriting the query with the `CASE` condition each time. This approach might make the process more flexible and allow for easier manipulation of the data during further analysis.