Kathmandu, Nepal 14–16 November
REGIONAL CONFERENCE 2024
Climate Responsive Land Governance and Disaster Resilience: Safeguarding Land Rights

A Python Library for Detection of Inconsistencies between Parcel Database and Plot Register

Mahesh Thapa
M.Sc. Geospatial Technologies

ORGANISED BY FIG NICS
PLATINUM SPONSOR Trimble

Parcel Database – The Core of Land Administration

Should be -

- Correct – All data in the database should be correct.
- Complete – All data in the area-of-interest should be present.

Errors in Parcel Database developed by Digitization of Paper Maps

Errors can be introduced when cadastral databases are created from the digitization of paper maps, primarily due to illegibility caused by extensive use

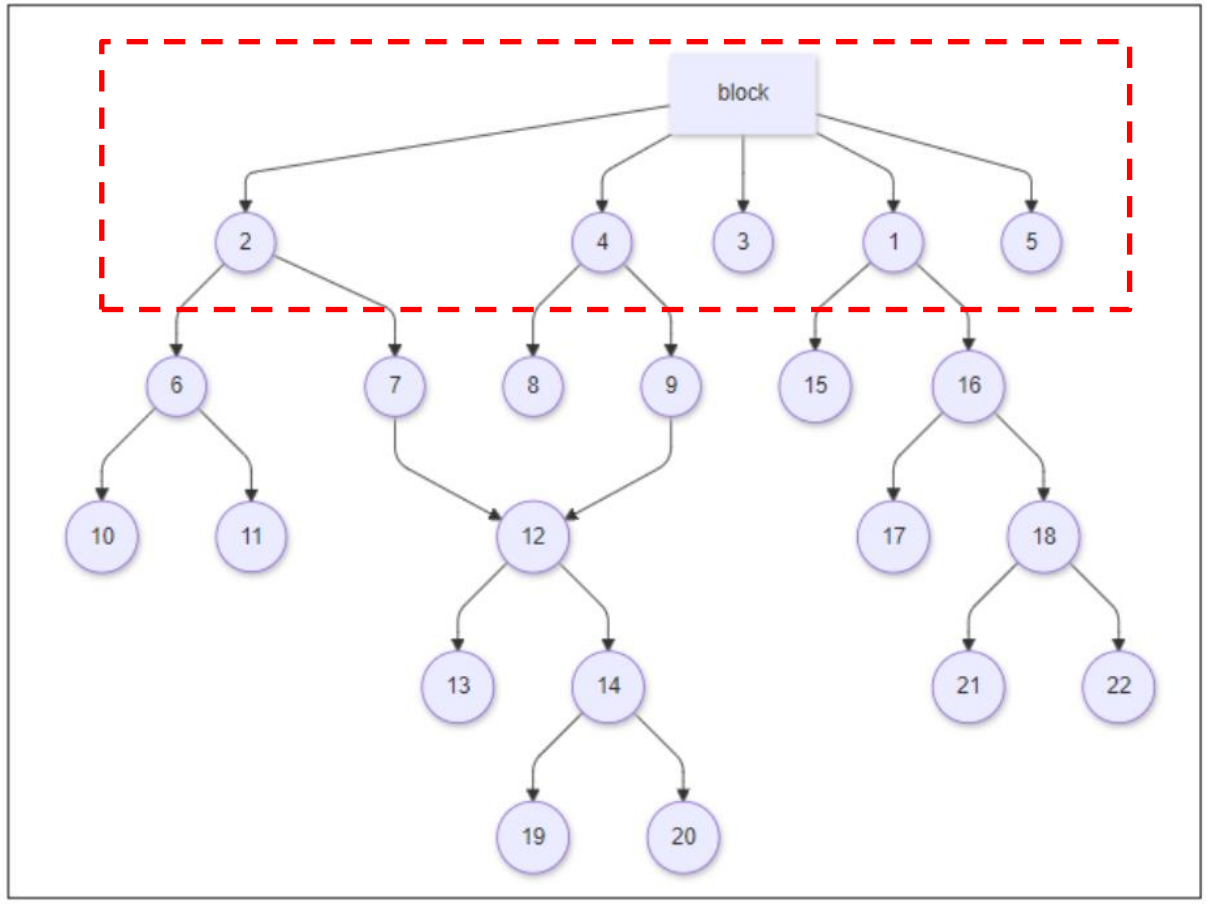
How can errors in a parcel-database be identified?

- Existing approach involves **overlaying digitized data onto scanned copies** of paper maps and reviewing each parcel individually.
- While this method **theoretically works**, it is too laborious and **not practical** enough.
- We require a more automated solution that can efficiently identify errors in the parcel database.

So, what can be done?

- Utilize **the relationship between the parcel-database and records in a plot-register.**
- A plot-register contains the record of all parcel subdivision and consolidation.
- Parcel database should be such that it should be consistent with the records in plot-register.

Records in a Plot-Register and its corresponding Parcel Tree

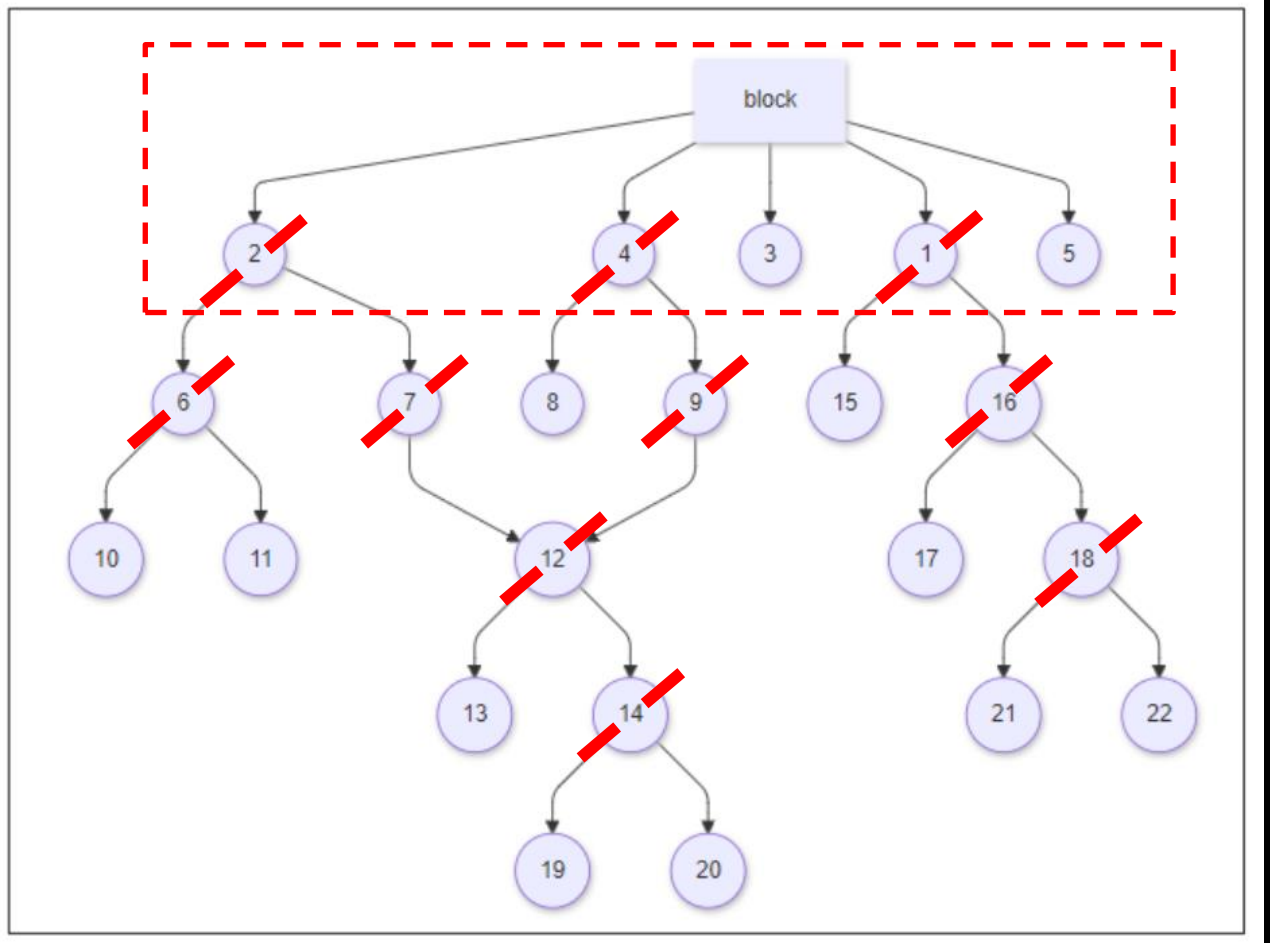


Parcel Tree

child_parcel	parent_parcel
1	block
2	block
3	block
4	block
5	block
6	2
7	2
8	4
9	4
10	6
11	6
12	7, 9
13	12
14	12
15	1
16	1
17	16
18	16
19	14
20	14
21	18
22	18

Plot-Register

Parent-parcels are annulled(cancelled) and no more exists in parcel-database

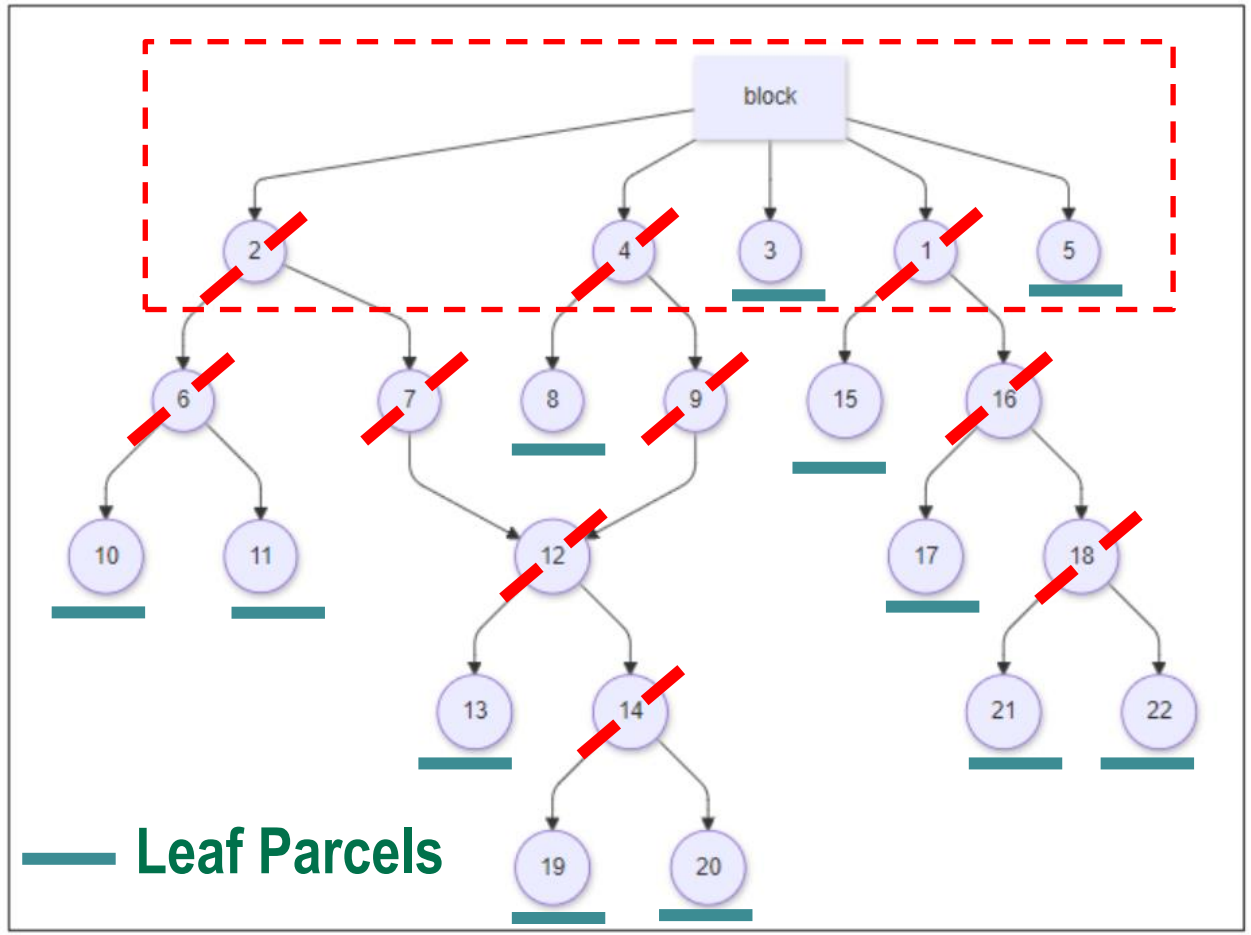


Parcel Tree

child_parcel	parent_parcel
1	block
2	block
3	block
4	block
5	block
6	2
7	2
8	4
9	4
10	6
11	6
12	7, 9
13	12
14	12
15	1
16	1
17	16
18	16
19	14
20	14
21	18
22	18

Plot-Register

Concept of Leaf Parcels

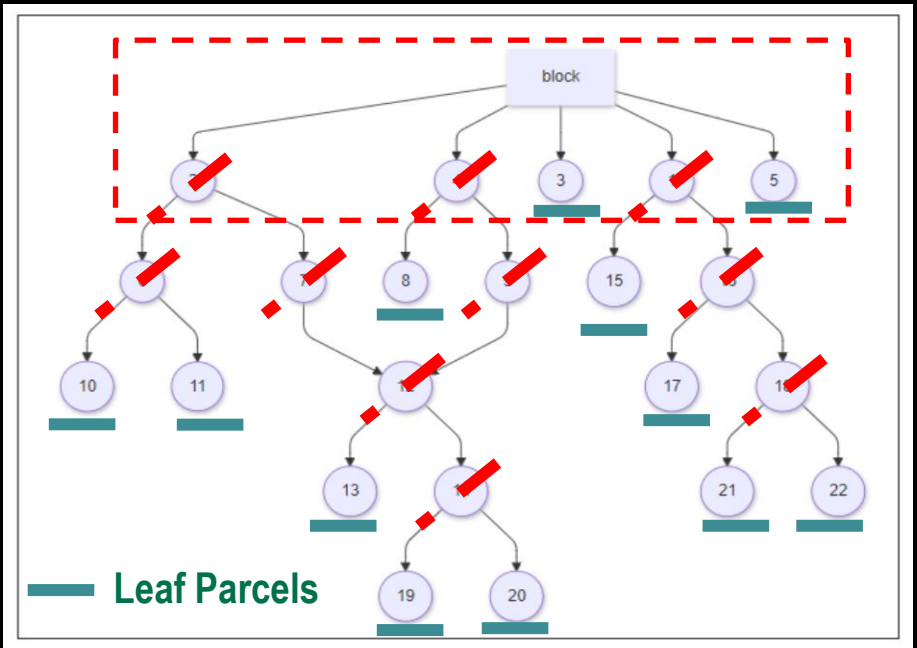


Parcel Tree

child_parcel	parent_parcel
1	block
2	block
3	block
4	block
5	block
6	2
7	2
8	4
9	4
10	6
11	6
12	7, 9
13	12
14	12
15	1
16	1
17	16
18	16
19	14
20	14
21	18
22	18

Plot-Register

Concept of Leaf Parcels



Parcel Tree

child_parcel	parent_parcel
1	block
2	block
3	block
4	block
5	block
6	2
7	2
8	4
9	4
10	6
11	6
12	7, 9
13	12
14	12
15	1
16	1
17	16
18	16
19	14
20	14
21	18
22	18

Plot-Register

Leaf Parcels = child_parcel – parent_parcel

Leaf Parcels = Child Parcels - Parent Parcels
= [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22] – [2,4,6,7,9,12,1,16,14,18]
= [3,5,10,11,13,15,17,19,20,21,22]

Principle of Consistency between Parcel Database and Plot-Register

"The plot-register and parcel-database are consistent if an only if parcel-database contains exclusively leaf-parcels, no-more no-less."

Parcel Database

Plot-Register

TAXONOMY OF INCONSISTENCIES

a. EXTRA PARCELS

Parcels in parcel-database but not in entire parcel-tree.

b. DUPLICATE PARCELS

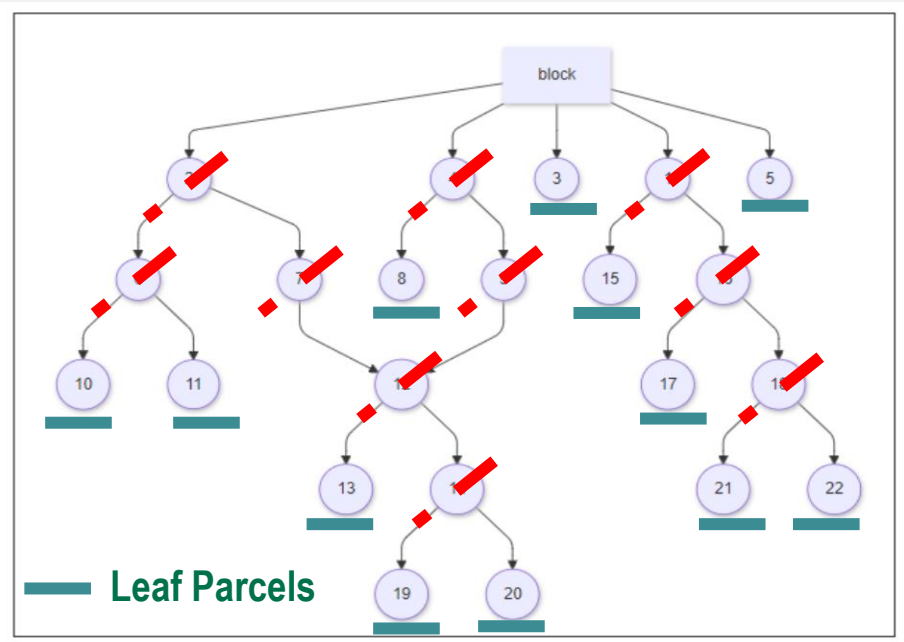
Parcels in leaf-parcels whose count is greater in parcel-database than their count in leaf-parcels.

c. DEAD PARCELS

Parcels that should have been annulled from parcel-database as per plot-register but still exist in parcel-database.

d. MISSING PARCELS

If a parcel in the leaf-parcels is missing in parcel-database and all of its ancestor parcels are also absent, these parcels are classified as missing parcels.

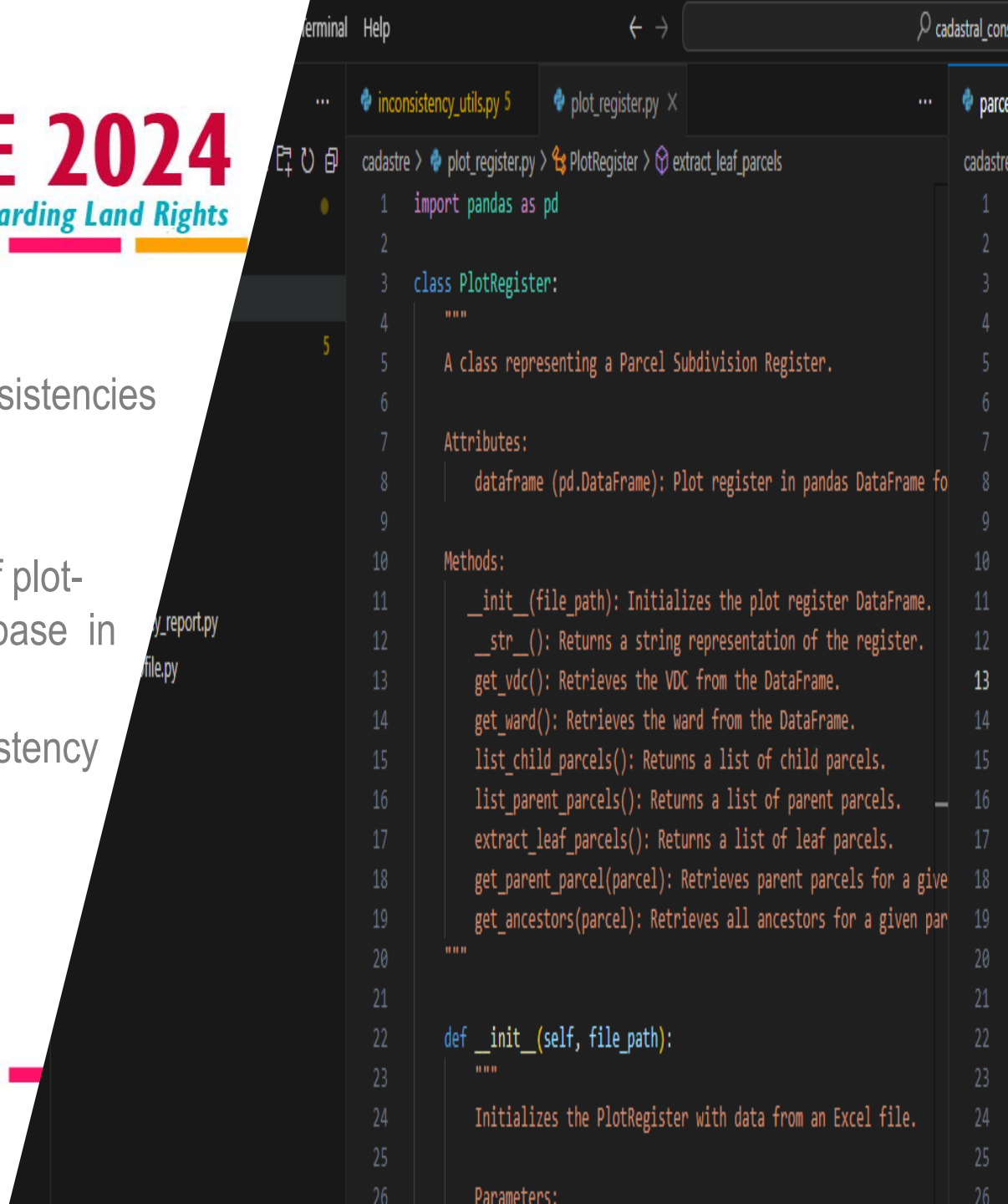


child_parcel	parent_parcel
1	block
2	block
3	block
4	block
5	block
6	2
7	2
8	4
9	4
10	6
11	6

child_parcel	parent_parcel
12	7, 9
13	12
14	12
15	1
16	1
17	16
18	16
19	14
20	14
21	18
22	18

Python Library

- A python library is developed for identifying categories of inconsistencies between parcel_database and plot_register.
- These inconsistencies point out to errors in parcel database
- Pandas and GeoPandas library are used for reading records of plot-register in excel format as DataFrame and reading parcel database in shapefile format as GeoDataFrame
- Accessible at https://github.com/maheshthapa/cadastral_consistency



Functions to Detect Inconsistencies

Extra Parcels

```python
# Returns list of extra parcels
def extract_extra_parcels(register, database):
    """
    Extracts parcels from the database that are not present in the register's child parcels.

    Parameters:
        register (PlotRegister): The register containing child parcels.
        database (Database): The database containing all parcels.

    Returns:
        list: A list of extra parcels not found in the child parcels.
    """
    # Retrieve lists of parcels from the database and register
    database_parcels = database.list_parcels()
    child_parcels = register.list_child_parcels()

    # Identify extra parcels that are in the database but not in the register's child parcels
    extra_parcels = [parcel for parcel in database_parcels if parcel not in child_parcels]

    return extra_parcels
```

Missing Parcels

```python
def extract_missing_parcels(register, database):
    # Extract leaf parcels and convert to sets for efficient lookups
    leaf_parcels = set(register.extract_leaf_parcels())
    database_parcels = set(database.list_parcels())

    missing_parcels = []

    # Iterate through each leaf parcel
    for parcel in leaf_parcels:
        if parcel not in database_parcels:
            ancestors = register.find_ancestors(parcel)

            # Check if none of the ancestors are present in the database parcels
            if not any(ancestor in database_parcels for ancestor in ancestors):
                missing_parcels.append(parcel)
    return missing_parcels
```

Dead Parcels

```python
def extract_dead_parcels(register, database):
    # Extract leaf parcels and convert to sets for efficient lookups
    leaf_parcels = set(register.extract_leaf_parcels())
    database_parcels = set(database.list_parcels())

    dead_parcels = []

    # Iterate through each leaf parcel
    for parcel in leaf_parcels:
        if parcel not in database_parcels:
            ancestors = register.find_ancestors(parcel)

            # Add ancestors that are present in the database parcels
            dead_parcels.extend(ancestor for ancestor in ancestors if ancestor in database_parcels)

    return dead_parcels
```

Duplicate Parcels

```python
def extract_duplicate_parcels(register, database):
    # Extract leaf parcels from the register
    leaf_parcels = register.extract_leaf_parcels()

    # List all parcels in the database
    database_parcels = database.list_parcels()
    print(database_parcels)

    # Count occurrences of each parcel in the leaf parcels
    leaf_count = {}
    for parcel in leaf_parcels:
        leaf_count[parcel] = leaf_count.get(parcel, 0) + 1

    # Count occurrences of each parcel in the database
    database_count = {}
    for parcel in database_parcels:
        database_count[parcel] = database_count.get(parcel, 0) + 1

    # Identify duplicates by comparing counts
    duplicate_parcels = [
        parcel for parcel in leaf_count
        if database_count.get(parcel, 0) > leaf_count[parcel]
    ]

    return duplicate_parcels
```

Sample Implementation

Parcels in Parcel Database

1, 3, 4, 6, 7, 8, 9, 11, 12, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 22, 22, 23, 24, 25, 26, 27

child_parcel	parent_parcel
1	block
2	block
3	block
4	block
5	block
6	2
7	2
8	4
9	4
10	6
11	6

child_parcel	parent_parcel
12	7, 9
13	12
14	12
15	1
16	1
17	16
18	16
19	14
20	14
21	18
22	18

Records as per Plot-register

Python Library

Sn	Inconsistency Categories	Parcels
1	Extra Parcels	['24', '25', '27', '26', '23']
2	Duplicate Parcels	['22']
3	Missing Parcels	['5']
4	Dead Parcels	['6']

Implications and Further Development

- Can be used to detect errors in cadastral database
- Can be also used as quality check tool during digitization process
- Can boost the confidence over maps generated using digital parcel-databases
- Lots of room for improvements with additional functionality such as graphic parcel tree generator, etc.

Thank You