Okay, great! The Event Sourcing / Ledger Model (Approach 2) is indeed well-suited for your goals, especially for tracing history and enabling visualizations like tree graphs. Since you're using Python and currently JSON, let's refine this approach.

**1. Data Structure (Python/JSON Implementation)**

We can represent the core entities using nested dictionaries and lists in JSON, which maps easily to Python dictionaries and lists.

* **properties.json:** A dictionary mapping PropertyID to property details.  
  {  
   "PROP123": {  
   "Address": "123 Main St, Cityville",  
   "ParcelNumber": "PN-ABC-001",  
   "Description": "Lot 5, Block B"  
   }  
  }
* **owners.json:** A dictionary mapping OwnerID to owner details.  
  {  
   "OWNER\_M": {"Name": "Marco"},  
   "OWNER\_C": {"Name": "Carlos"},  
   "OWNER\_L": {"Name": "Laura"},  
   "OWNER\_Ma": {"Name": "Maria"},  
   "OWNER\_INIT": {"Name": "Original Seller Corp."} // Example initial owner  
  }
* **deeds.json:** A dictionary mapping DeedID to deed metadata.  
  {  
   "DEED001": {"DocRef": "Matricula 999 / R.1", "EffectiveDate": "2010-01-15"},  
   "DEED002": {"DocRef": "Matricula 999 / R.2", "EffectiveDate": "2015-05-20"},  
   "DEED003": {"DocRef": "Matricula 999 / R.3", "EffectiveDate": "2017-08-10"}  
  }
* **ownership\_ledger.json:** A list containing each transfer event entry. *Crucially, each entry represents a single owner's change in share for a specific deed.*  
  [  
   // DEED001: Initial Acquisition - Marco gets 100% from Initial Seller  
   {  
   "LedgerEntryID": 1, "DeedID": "DEED001", "PropertyID": "PROP123",  
   "OwnerID": "OWNER\_INIT", "ChangeInSharePercentage": -1.00,  
   "EffectiveDate": "2010-01-15", "Notes": "Initial Sale"  
   },  
   {  
   "LedgerEntryID": 2, "DeedID": "DEED001", "PropertyID": "PROP123",  
   "OwnerID": "OWNER\_M", "ChangeInSharePercentage": 1.00,  
   "EffectiveDate": "2010-01-15", "Notes": "Initial Purchase"  
   },  
    
   // DEED002: Marco (currently 100%) sells 60% to Carlos. Marco keeps 40%.  
   {  
   "LedgerEntryID": 3, "DeedID": "DEED002", "PropertyID": "PROP123",  
   "OwnerID": "OWNER\_M", "ChangeInSharePercentage": -0.60,  
   "EffectiveDate": "2015-05-20", "Notes": "Partial Sale to Carlos"  
   },  
   {  
   "LedgerEntryID": 4, "DeedID": "DEED002", "PropertyID": "PROP123",  
   "OwnerID": "OWNER\_C", "ChangeInSharePercentage": 0.60,  
   "EffectiveDate": "2015-05-20", "Notes": "Purchase from Marco"  
   },  
    
   // DEED003: Carlos (currently 60%) sells 40% (absolute) - 20% to Laura, 20% to Maria. Carlos keeps 20%.  
   {  
   "LedgerEntryID": 5, "DeedID": "DEED003", "PropertyID": "PROP123",  
   "OwnerID": "OWNER\_C", "ChangeInSharePercentage": -0.40, // Selling 40% absolute share  
   "EffectiveDate": "2017-08-10", "Notes": "Partial Sale to Laura & Maria"  
   },  
   {  
   "LedgerEntryID": 6, "DeedID": "DEED003", "PropertyID": "PROP123",  
   "OwnerID": "OWNER\_L", "ChangeInSharePercentage": 0.20,  
   "EffectiveDate": "2017-08-10", "Notes": "Purchase from Carlos"  
   },  
   {  
   "LedgerEntryID": 7, "DeedID": "DEED003", "PropertyID": "PROP123",  
   "OwnerID": "OWNER\_Ma", "ChangeInSharePercentage": 0.20,  
   "EffectiveDate": "2017-08-10", "Notes": "Purchase from Carlos"  
   }  
  ]  
  *Key points:*
  + Every transfer involves at least one negative entry (grantor) and at least one positive entry (grantee).
  + For a given DeedID and PropertyID, the sum of ChangeInSharePercentage across all related ledger entries *must* be zero.

**2. Calculating Current Ownership Status (Python)**

This function aggregates the ledger entries for a specific property up to a certain date (or the latest if no date is specified).

from collections import defaultdict  
from datetime import datetime  
  
# Assume you load the JSON data into these variables:  
# properties\_data, owners\_data, deeds\_data, ledger\_data  
  
def get\_ownership\_status(ledger, owners, property\_id, target\_date\_str=None):  
 """Calculates ownership percentages for a property at a specific date."""  
  
 if target\_date\_str:  
 target\_date = datetime.strptime(target\_date\_str, '%Y-%m-%d').date()  
 else:  
 target\_date = datetime.now().date() # Use current date for latest status  
  
 ownership = defaultdict(float)  
 relevant\_entries = [  
 entry for entry in ledger  
 if entry['PropertyID'] == property\_id and  
 datetime.strptime(entry['EffectiveDate'], '%Y-%m-%d').date() <= target\_date  
 ]  
  
 # Sort by date to process in order (though sum order doesn't strictly matter)  
 relevant\_entries.sort(key=lambda x: datetime.strptime(x['EffectiveDate'], '%Y-%m-%d').date())  
  
 for entry in relevant\_entries:  
 ownership[entry['OwnerID']] += entry['ChangeInSharePercentage']  
  
 # Format output with names and percentages, filtering out zero shares  
 current\_status = {}  
 for owner\_id, share in ownership.items():  
 if abs(share) > 1e-9: # Check if share is effectively non-zero  
 owner\_name = owners.get(owner\_id, {}).get('Name', owner\_id) # Fallback to ID if name not found  
 current\_status[owner\_name] = round(share \* 100, 2) # Percentage  
  
 return current\_status  
  
# Example Usage:  
# Assuming you've loaded your JSON files into the respective variables  
# latest\_status = get\_ownership\_status(ledger\_data, owners\_data, "PROP123")  
# print(f"Latest Ownership for PROP123: {latest\_status}")  
# Output: Latest Ownership for PROP123: {'Marco': 40.0, 'Carlos': 20.0, 'Laura': 20.0, 'Maria': 20.0}  
  
# status\_as\_of\_2016 = get\_ownership\_status(ledger\_data, owners\_data, "PROP123", "2016-12-31")  
# print(f"Ownership for PROP123 as of 2016-12-31: {status\_as\_of\_2016}")  
# Output: Ownership for PROP123 as of 2016-12-31: {'Marco': 40.0, 'Carlos': 60.0}

**3. Tracing History & Visualization (Tree Graph)**

This requires identifying the source of each *positive* change in the ledger for a current owner.

**Logic for Tracing:**

1. Calculate the current ownership status for the property.
2. For a specific owner (e.g., Laura, 20%), find all *positive* ledger entries for her and this property.
3. For each positive entry (e.g., LedgerEntryID: 6, +0.20, DeedID: DEED003):
   * Find the corresponding *negative* entry(ies) with the same DeedID and PropertyID (e.g., LedgerEntryID: 5, OwnerID: OWNER\_C, -0.40). This tells you Laura got her share (or part of it) from Carlos via DEED003.
   * Note the grantor (Carlos), the share amount (+0.20), and the deed/date.
4. Now, recursively trace the origin of the *grantor's* (Carlos's) share *before* that transfer (DEED003). How did Carlos have the share he transferred? Find positive entries for Carlos *before* DEED003's effective date. (e.g., LedgerEntryID: 4, +0.60, DeedID: DEED002).
5. Find the grantor for *that* transfer (e.g., LedgerEntryID: 3, OwnerID: OWNER\_M, -0.60). So, Carlos got his initial share from Marco via DEED002.
6. Continue tracing Marco's share back until you hit the initial acquisition (ChangeInSharePercentage > 0 with no corresponding negative entry for a non-initial owner in the *same* deed, or a designated "initial" deed).

**Visualization Tool Recommendation: Graphviz**

Graphviz is excellent for creating these kinds of hierarchical/tree diagrams from Python.

# You'll need to install the library: pip install graphviz  
import graphviz  
  
# Assume owners\_data and deeds\_data are loaded  
  
def build\_ownership\_trace\_graph(ledger, owners, deeds, property\_id, target\_owner\_id):  
 """Builds a Graphviz graph tracing the origin of an owner's share."""  
  
 dot = graphviz.Digraph(comment=f'Ownership Trace for {owners.get(target\_owner\_id, {}).get("Name", target\_owner\_id)} on {property\_id}')  
 dot.attr('node', shape='box', style='filled', color='lightgrey')  
 dot.attr(rankdir='RL') # Rank direction Right to Left for a nice tree  
  
 processed\_entries = set() # To avoid infinite loops in complex scenarios (though unlikely here)  
 nodes\_added = set()  
  
 def add\_node(owner\_id, share\_pct, date\_str):  
 node\_id = f"{owner\_id}\_{date\_str}\_{share\_pct:.2f}"  
 if node\_id not in nodes\_added:  
 owner\_name = owners.get(owner\_id, {}).get("Name", owner\_id)  
 label = f"{owner\_name}\n{share\_pct:.2f}%\n({date\_str})"  
 dot.node(node\_id, label)  
 nodes\_added.add(node\_id)  
 return node\_id  
  
 def trace\_recursive(current\_owner\_id, effective\_date\_limit):  
 # Find positive entries for this owner up to the date limit  
 owner\_entries = [  
 e for e in ledger  
 if e['OwnerID'] == current\_owner\_id and  
 e['PropertyID'] == property\_id and  
 e['ChangeInSharePercentage'] > 1e-9 and # Positive change  
 datetime.strptime(e['EffectiveDate'], '%Y-%m-%d').date() <= effective\_date\_limit and  
 e['LedgerEntryID'] not in processed\_entries  
 ]  
 owner\_entries.sort(key=lambda x: datetime.strptime(x['EffectiveDate'], '%Y-%m-%d').date(), reverse=True)  
  
 acquisition\_nodes = []  
  
 for entry in owner\_entries:  
 processed\_entries.add(entry['LedgerEntryID'])  
 deed\_id = entry['DeedID']  
 share\_gained = entry['ChangeInSharePercentage'] \* 100  
 effective\_date = datetime.strptime(entry['EffectiveDate'], '%Y-%m-%d').date()  
 effective\_date\_str = effective\_date.strftime('%Y-%m-%d')  
  
 # Create node for this acquisition point  
 current\_node\_id = add\_node(current\_owner\_id, share\_gained, effective\_date\_str)  
 acquisition\_nodes.append(current\_node\_id)  
  
  
 # Find the grantor(s) for this specific deed/transfer  
 grantor\_entries = [  
 g for g in ledger  
 if g['DeedID'] == deed\_id and  
 g['PropertyID'] == property\_id and  
 g['ChangeInSharePercentage'] < -1e-9 # Negative change  
 ]  
  
 if not grantor\_entries:  
 # Likely initial acquisition or data issue  
 dot.edge("SOURCE", current\_node\_id, label=f"Initial Acq?\n({effective\_date\_str})")  
 continue # Stop tracing this branch  
  
 deed\_info = deeds.get(deed\_id, {})  
 deed\_label = f"{deed\_info.get('DocRef', deed\_id)}\n({effective\_date\_str})"  
  
 # Trace back each grantor \*before\* this transfer date  
 for grantor\_entry in grantor\_entries:  
 grantor\_id = grantor\_entry['OwnerID']  
 # Important: Trace grantor's history \*before\* this transfer  
 grantor\_nodes = trace\_recursive(grantor\_id, effective\_date - timedelta(days=1)) # Look just before the transfer  
  
 # Link grantor nodes to the current acquisition node  
 for grantor\_node\_id in grantor\_nodes:  
 dot.edge(grantor\_node\_id, current\_node\_id, label=deed\_label)  
  
  
 return acquisition\_nodes # Return nodes created at this level  
  
  
 # --- Calculate current share of target owner first ---  
 status\_now = get\_ownership\_status(ledger, owners, property\_id)  
 target\_owner\_name = owners.get(target\_owner\_id, {}).get("Name", target\_owner\_id)  
 current\_share = status\_now.get(target\_owner\_name, 0)  
  
 if current\_share > 0:  
 final\_node\_id = add\_node(target\_owner\_id, current\_share, "Current")  
 # Start tracing back from today  
 source\_nodes = trace\_recursive(target\_owner\_id, datetime.now().date())  
 # Link the final traced acquisitions to the "Current" node (optional visual step)  
 for node\_id in source\_nodes:  
 # Maybe just rely on the recursive calls to build the full tree ending at dated nodes  
 pass # The recursive calls build the tree structure already  
 else:  
 print(f"{target\_owner\_name} has no current ownership in {property\_id}.")  
  
  
 return dot  
  
  
# --- Example Usage (Needs timedelta from datetime) ---  
from datetime import datetime, timedelta  
  
# Assuming you have loaded ledger\_data, owners\_data, deeds\_data  
  
# Trace Laura's share  
# graph\_laura = build\_ownership\_trace\_graph(ledger\_data, owners\_data, deeds\_data, "PROP123", "OWNER\_L")  
# if graph\_laura:  
# graph\_laura.render('ownership\_trace\_laura', view=True, format='png') # Saves file and tries to open it  
  
# Trace Marco's share  
# graph\_marco = build\_ownership\_trace\_graph(ledger\_data, owners\_data, deeds\_data, "PROP123", "OWNER\_M")  
# if graph\_marco:  
# graph\_marco.render('ownership\_trace\_marco', view=True, format='png')

*Self-correction:* The tracing logic needs refinement. It should focus on tracing the *source* of positive entries. When a positive entry is found, identify the negative entries in the *same deed*. Then, recursively trace the origin of the share held by those *grantors* *before* the transfer date. The provided code sketch needs adjustment to correctly implement this recursive linking. Added timedelta. Refined node naming and labels. Added rank direction. The recursive logic linking needs careful testing.

**4. Data Storage Consideration**

While JSON works initially, as your data grows, querying and maintaining integrity becomes harder. Consider migrating to **SQLite**:

* Create tables matching your JSON structures (Properties, Owners, Deeds, OwnershipLedger).
* Use Python's sqlite3 module to interact with the database.
* Benefits: Faster queries (especially with indexes on PropertyID, OwnerID, EffectiveDate), data integrity (types, constraints), easier updates.

This refined approach using the Ledger model in Python/JSON, coupled with Graphviz for visualization, should give you the clear current status and the historical trace you need. Remember to thoroughly test the tracing logic.