# Neo4j UHNW Knowledge Graph – Data Model & Ingestion Script

## 1. Graph Schema Design

### Node Labels

| Label | Primary Key | Mandatory Properties | Optional Properties (examples) |
| --- | --- | --- | --- |
| Person | id | name (string) | dateOfBirth, citizenship, netWorth, occupation, alias, lastUpdated |
| Company | id | name (string) | registrationNumber, industry, website, country, lastUpdated |
| Dataset | name | – | description, ingestDate |

### Relationship Types

| Type | Start → End | Key Properties | Notes |
| --- | --- | --- | --- |
|  |  |  |  |

| **HAS\_ROLE\_AT** | Person → Company | role, startDate, endDate, source | Ingested from MAS workflow & company disclosures |
| --- | --- | --- | --- |
| **FAMILY** | Person ↔ Person | relation (e.g. father, spouse) , source | Derived from Wikidata-style JSON |
| **OWNS** | Person → Company | sharePct, valuation, source | (optional; placeholder for future enrichment) |
| **BELONGS\_TO** | *Any* → Dataset | – | Tag original source rows for lineage |

**Constraints & Indexes**

CREATE CONSTRAINT person\_id IF NOT EXISTS ON (p:Person) ASSERT p.id IS UNIQUE;  
CREATE CONSTRAINT company\_id IF NOT EXISTS ON (c:Company) ASSERT c.id IS UNIQUE;  
CREATE INDEX person\_name IF NOT EXISTS FOR (p:Person) ON (p.name);  
CREATE INDEX company\_name IF NOT EXISTS FOR (c:Company) ON (c.name);

## 2. Python Ingestion Script – load\_graph.py

#!/usr/bin/env python3  
"""  
Load three UHNW‑related data sources into a Neo4j Aura instance using the schema above.  
  
Usage:  
 export NEO4J\_URI='neo4j+s://<your-aura-endpoint>'  
 export NEO4J\_USER='neo4j'  
 export NEO4J\_PASSWORD='<password>'  
 python load\_graph.py \  
 --neo4j\_export neo4j\_query\_table\_data\_2025-6-26.json \  
 --wikidata\_json data.json \  
 --mas\_workflow MAS\_first\_1500.json  
  
n8n v1.98.1 tip: wrap this script with an \*\*Execute Command\*\* node (Image: `python:3.12-alpine`) and mount the dataset volume for fully automated nightly refresh.  
"""  
  
import os, json, csv, argparse  
from pathlib import Path  
from typing import Dict, Any, List, Optional  
from neo4j import GraphDatabase  
  
# ──────────────────────────────────────────────────────────────────────────────  
# Neo4j connection helpers  
# ──────────────────────────────────────────────────────────────────────────────  
NEO4J\_URI = os.getenv("NEO4J\_URI")  
NEO4J\_USER = os.getenv("NEO4J\_USER")  
NEO4J\_PASSWORD = os.getenv("NEO4J\_PASSWORD")  
  
driver = GraphDatabase.driver(NEO4J\_URI, auth=(NEO4J\_USER, NEO4J\_PASSWORD))  
  
MERGE\_PERSON = """  
MERGE (p:Person {id:$id})  
SET p.name = COALESCE(p.name, $name),  
 p.lastUpdated = datetime($lastUpdated),  
 p += $props  
"""  
  
MERGE\_COMPANY = """  
MERGE (c:Company {id:$id})  
SET c.name = COALESCE(c.name, $name),  
 c.lastUpdated = datetime($lastUpdated),  
 c += $props  
"""  
  
MERGE\_ROLE = """  
MATCH (p:Person {id:$person\_id})  
MATCH (c:Company {id:$company\_id})  
MERGE (p)-[r:HAS\_ROLE\_AT {role:$role}]->(c)  
SET r.startDate = $startDate,  
 r.endDate = $endDate,  
 r.source = $source  
"""  
  
MERGE\_FAMILY = """  
MATCH (a:Person {id:$src\_id})  
MATCH (b:Person {id:$dst\_id})  
MERGE (a)-[r:FAMILY {relation:$relation}]->(b)  
SET r.source = $source  
"""  
  
def merge\_person(tx, id:str, name:str, props:Dict[str,Any]):  
 tx.run(MERGE\_PERSON,  
 id=id,  
 name=name,  
 props={k:v for k,v in props.items() if v is not None},  
 lastUpdated=os.getenv('LOAD\_TS','2025-07-08'))  
  
def merge\_company(tx, id:str, name:str, props:Dict[str,Any]):  
 tx.run(MERGE\_COMPANY,  
 id=id,  
 name=name,  
 props={k:v for k,v in props.items() if v is not None},  
 lastUpdated=os.getenv('LOAD\_TS','2025-07-08'))  
  
def merge\_role(tx, person\_id:str, company\_id:str, role:str, start:Optional[int], end:Optional[int], source:str):  
 tx.run(MERGE\_ROLE,  
 person\_id=person\_id,  
 company\_id=company\_id,  
 role=role,  
 startDate=start,  
 endDate=end,  
 source=source)  
  
def merge\_family(tx, src\_id:str, dst\_id:str, relation:str, source:str):  
 tx.run(MERGE\_FAMILY,  
 src\_id=src\_id,  
 dst\_id=dst\_id,  
 relation=relation,  
 source=source)  
  
# ──────────────────────────────────────────────────────────────────────────────  
# Parsers for each dataset  
# ──────────────────────────────────────────────────────────────────────────────  
  
def ingest\_neo4j\_query\_table(path:Path):  
 """Ingest `neo4j\_query\_table\_data\_2025-6-26.json`."""  
 data = json.loads(path.read\_text())  
 with driver.session() as session:  
 for record in data:  
 p = record['n']  
 c = record['m']  
 rel = record['r']  
 pid = f"src1:{p['identity']}"  
 cid = f"src1:{c['identity']}"  
 session.write\_transaction(merge\_person, pid, p['properties']['name'], p['properties'])  
 session.write\_transaction(merge\_company, cid, c['properties']['name'], c['properties'])  
 if rel:  
 rp = rel['properties']  
 session.write\_transaction(merge\_role,  
 pid,  
 cid,  
 rp.get('role'),  
 rp.get('startDate'),  
 rp.get('endDate'),  
 'neo4j\_query\_table\_data')  
  
def ingest\_wikidata\_json(path:Path):  
 """Ingest simplified Wikidata‑style JSON (`data.json`)."""  
 persons = json.loads(path.read\_text())['persons']  
 with driver.session() as session:  
 # pass 1 – nodes  
 for p in persons:  
 pid = f"wd:{p['id']}"  
 session.write\_transaction(merge\_person, pid, p['props']['name'], p['props'])  
 # pass 2 – family relationships  
 for p in persons:  
 src\_id = f"wd:{p['id']}"  
 attrs = p.get('attributes', {})  
 for tag,label in attrs.items():  
 if not tag.endswith('Label'):  
 continue  
 relation = tag.replace('Label','').lower() # fatherLabel → father  
 targets = label if isinstance(label, list) else [label]  
 for name in targets:  
 dst\_id = f"wd:label:{name}"  
 session.write\_transaction(merge\_person, dst\_id, name, {})  
 session.write\_transaction(merge\_family, src\_id, dst\_id, relation, 'wikidata\_json')  
  
# NOTE: MAS\_first\_1500.json is an n8n workflow \*definition\* rather than flat data.  
# In production we recommend exporting the scraped personnel rows into CSV/JSON.  
# Below is a stub that expects such a CSV (company\_name,person\_name,person\_title).  
  
def ingest\_mas\_csv(csv\_path:Path):  
 with driver.session() as session, csv\_path.open() as f:  
 reader = csv.DictReader(f)  
 for row in reader:  
 pid = f"mas:{row['person\_name']}".lower().replace(' ','\_')  
 cid = f"mas:{row['company\_name']}".lower().replace(' ','\_')  
 session.write\_transaction(merge\_person, pid, row['person\_name'], {})  
 session.write\_transaction(merge\_company, cid, row['company\_name'], {})  
 session.write\_transaction(merge\_role, pid, cid, row['person\_title'], None, None, 'MAS\_scrape')  
  
# ──────────────────────────────────────────────────────────────────────────────  
# Entrypoint  
# ──────────────────────────────────────────────────────────────────────────────  
  
def main(args):  
 ingest\_neo4j\_query\_table(Path(args.neo4j\_export))  
 ingest\_wikidata\_json(Path(args.wikidata\_json))  
 ingest\_mas\_csv(Path(args.mas\_personnel\_csv))  
 driver.close()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 p = argparse.ArgumentParser(description="Load UHNW datasets into Neo4j Aura")  
 p.add\_argument("--neo4j\_export", required=True)  
 p.add\_argument("--wikidata\_json", required=True)  
 p.add\_argument("--mas\_personnel\_csv",required=True,  
 help="Flattened MAS personnel file (generated by your n8n workflow)")  
 main(p.parse\_args())

## 3. Suggested ETL Automation in n8n (v1.98.1)

1. **Cron** → **Execute Command** (python load\_graph.py …) → **IF / Slack** for alerts.
2. Store raw exports in S3 / MinIO, then trigger the script for lineage.
3. Use n8n’s **Wait / Continue** to chain incremental MAS scraping before ingest.

### Next Steps & Enhancements

* **Deduplication** – Use APOC apoc.periodic.iterate with Levenshtein to collapse homonyms.
* **UHNW Scoring Layer** – Compute centrality + attach wealth tiers as :UHNW {tier} nodes.
* **Visualization** – Integrate GraphXR or D3.js network with Neo4j GraphQL for your RM dashboard.
* **Checks & Tests** – Add pytest + GitHub Actions to unit‑test parsers and Cypher constraints before deployment.

Let me know which part you’d like to dive deeper into! :rocket: