B4M36DS2, BE4M36DS2: **Database Systems 2**

https://cw.fel.cvut.cz/b221/courses/b4m36ds2/

Practical Class 11

Neo4j from Python

Yuliia Prokop
prokoyul@fel.cvut.cz

9. 12. 2024

Czech Technical University in Prague, Faculty of Electrical Engineering

First Steps

Neo4j Python Driver

```
pip install neo4j
```

Using Neo4j from Python

https://neo4j.com/docs/getting-started/languages-guides/neo4j-python/

Basic program structure:

```
from neo4j import GraphDatabase

class MovieGraphApp:
    def __init__(self, uri="bolt://localhost:7687",  user="user",
password="password"):
        self.driver = GraphDatabase.driver(uri, auth=(user, password))


    def close(self):
        self.driver.close()
```

Database Connection

Database Connection Establishing a connection involves creating a driver instance and verifying connectivity.

```
def verify_connectivity(self):
    try:
        self.driver.verify_connectivity()
        print("✓ Connected to Neo4j")
        return True
    except Exception as e:
        print(f"✗ Connection error: {str(e)}")
        return False
```

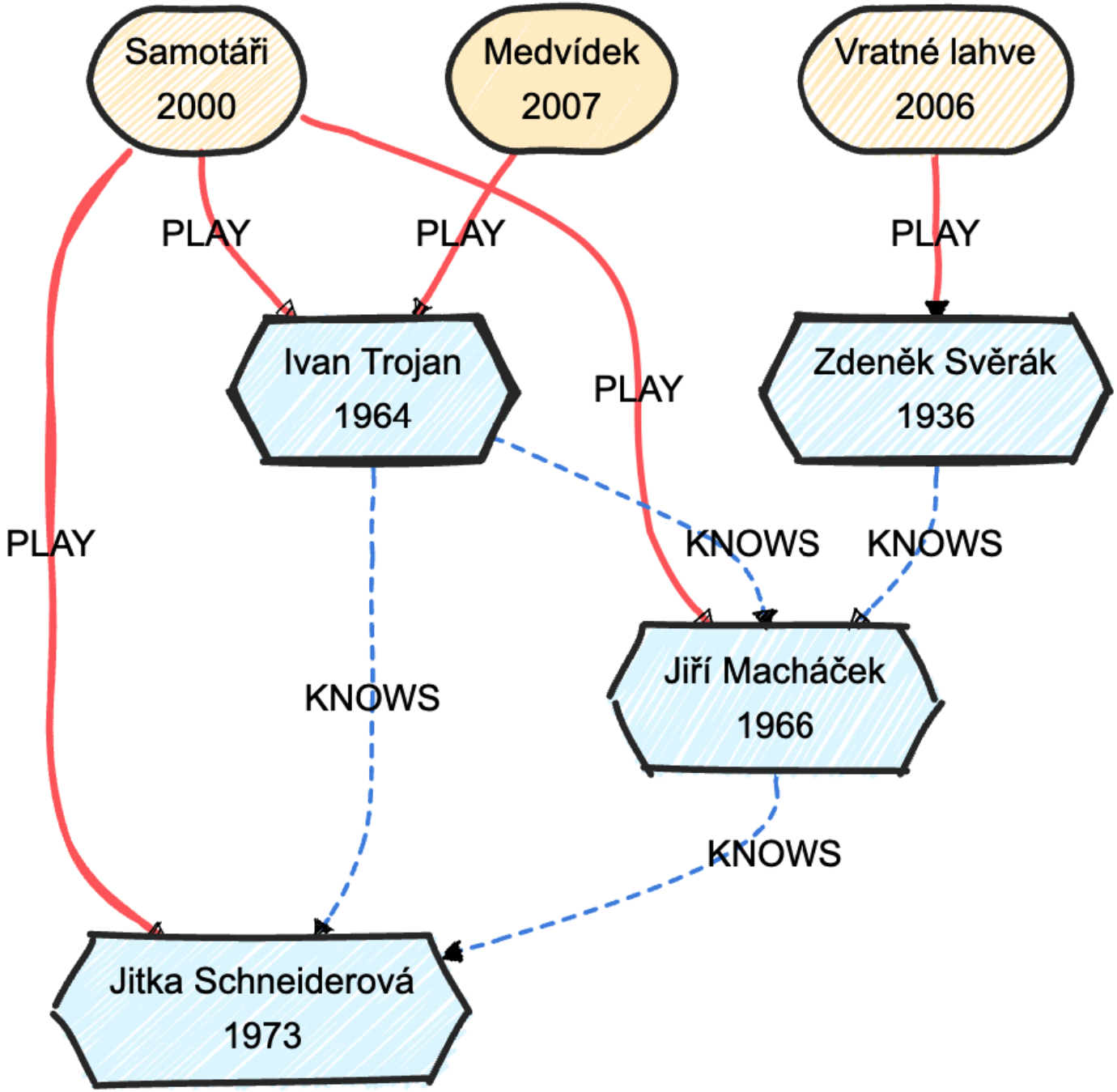
Creating Nodes

```
def create_node(self, label: str, properties: Dict):
    """Create a single node with label and properties"""
    with self.driver.session() as session:
        query = f"""
            CREATE (n:$label)
            SET n = $props
            RETURN n
        """
        result = session.run(query, label=label, props=properties)
```

Bulk Node Creation

```
def create_multiple_actors(self):
    with self.driver.session() as session:
        session.run("""
            UNWIND $actors as actor
            CREATE (a:ACTOR)
            SET a = actor
        """, actors=[
            {"id": "newman", "name": "Paul Newman", "year": 1925},
            {"id": "cruise", "name": "Tom Cruise", "year": 1962}
        ])
```

Graph


Samotáři
2000
Medvídek
2007
Vratné lahve
2006
PLAY
PLAY
PLAY
Ivan Trojan
1964
Zdeněk Svěrák
1936
PLAY
PLAY
KNOWS
KNOWS
KNOWS
Jiří Macháček
1966
KNOWS
Jitka Schneiderová
1973

Nodes

Create graph nodes for a few actors

- Create nodes, add ACTOR labels, add properties
  - trojan, Ivan Trojan, 1964
  - machacek, Jiří Macháček, 1966
  - schneiderova, Jitka Schneiderová, 1973
  - sverak, Zdeněk Svěrák, 1936
- Remember node references

Creating Actor nodes:

```
def create_actors(self):
    with self.driver.session() as session:
        session.run("""
            CREATE (a:ACTOR {id: 'trojan', name: 'Ivan Trojan', year:
1964})
            CREATE (a:ACTOR {id: 'machacek', name: 'Jiří Macháček',
year: 1966})
            CREATE (a:ACTOR {id: 'schneiderova', name: 'Jitka
Schneiderová', year: 1973})
            CREATE (a:ACTOR {id: 'sverak', name: 'Zdeněk Svěrák',
year: 1936})
        """)
```

Creating Movie nodes:

```
def create_movies(self):
    with self.driver.session() as session:
        session.run("""
            CREATE (m:MOVIE {id: 'samotari', title: 'Samotáři',
year: 2000})
            CREATE (m:MOVIE {id: 'medvidek', title:
'Medvídek', year: 2007})
            CREATE (m:MOVIE {id: 'vratnelahve', title: 'Vratné
lahve', year: 2006})
        """)
```

Relationships

Create relationships between our actors

- Create relationships of KNOWS type
  - trojan → machacek
  - trojan → schneiderova
  - machacek → trojan
  - machacek → schneiderova
  - sverak → machacek
- Consider these relationships as symmetric

Creating Relationships:

```
def create_relationships(self):
    with self.driver.session() as session:
        # Create KNOWS relationships between actors
        session.run("""
            MATCH (a1:ACTOR {id: 'trojan'}), (a2:ACTOR {id: 'machacek'})
            CREATE (a1)-[:KNOWS]->(a2)
            CREATE (a2)-[:KNOWS]->(a1)
        """)

        # Create PLAY relationships between movies and actors
        session.run("""
            MATCH (m:MOVIE {id: 'samotari'}), (a:ACTOR {id: 'trojan'})
            CREATE (m)-[:PLAY]->(a)
        """)
```

Creating Relationships:

```
def create_movie_relationships(self):
    with self.driver.session() as session:
        # Create all PLAY relationships in one query
        session.run("""
            MATCH (m:MOVIE), (a:ACTOR)
            WHERE (m.id = 'samotari' AND a.id IN
                        ['trojan', 'machacek', 'schneiderova'])
              OR (m.id = 'medvidek' AND a.id = 'trojan')
              OR (m.id = 'vratnelahve' AND a.id = 'sverak')
            CREATE (m)-[:PLAY]->(a)
        """)
```

Update Node Properties

```
def update_node(self, label, node_id, updates):
    with self.driver.session() as session:
        session.run("""
            MATCH (n:$label {id: $id})
            SET n += $updates
            RETURN n
        """, label=label, id=node_id, updates=updates)

# Example:
updates = {"year": 1963, "awards": ["Oscar", "Golden Globe"]}
update_node("ACTOR", "cruise", updates)
```

Add/Remove Labels

```
def modify_node_labels(self, node_id, add_labels=None,
remove_labels=None):
    with self.driver.session() as session:
        if add_labels:
            session.run("""
                MATCH (n {id: $id})
                SET n:$labels
            """, id=node_id, labels=add_labels)

        if remove_labels:
            session.run("""
                MATCH (n {id: $id})
                REMOVE n:$labels
            """, id=node_id, labels=remove_labels)
```

Modify Relationship Properties:

```
def update_relationship(self, from_id, to_id, rel_type,
updates):
    with self.driver.session() as session:
        session.run("""
            MATCH (a {id: $from})-[r:$type]->(b {id: $to})
            SET r += $updates
        """, from=from_id, to=to_id, type=rel_type,
updates=updates)
```

Change Relationship Type:

```
def change_relationship_type(self, from_id, to_id, old_type,
new_type):
    with self.driver.session() as session:
        # Neo4j doesn't support direct relationship type changes
        # We need to create new and delete old
        session.run("""
            MATCH (a {id: $from})-[r:$old_type]->(b {id: $to})
            WITH a, b, properties(r) as props
            DELETE r
            CREATE (a)-[new:$new_type]->(b)
            SET new = props
        """, from=from_id, to=to_id,
            old_type=old_type, new_type=new_type)
```

Deleting Nodes and Relationships:

```
def delete_node(self, node_id):
    with self.driver.session() as session:
        session.run("""
            MATCH (n {id: $id})
            DETACH DELETE n
        """, id=node_id)

def delete_relationship(self, from_id, to_id, rel_type):
    with self.driver.session() as session:
        session.run("""
            MATCH (a {id: $from})-[r:$type]->(b {id: $to})
            DELETE r
        """, from=from_id, to=to_id, type=rel_type)
```

Graph Traversal in Neo4j with Python

1. Basic Graph Traversal Concepts:

- MATCH Patterns - Core of Neo4j traversal in Python
- Path Finding - Ways to navigate through the graph
- Direction Control - Incoming, outgoing, or both directions
- Filtering - Conditions for including nodes and relationships

2. Key Components of Graph Traversal (Using Cypher with Python):

```
def traverse_graph(self, start_node_id):
    with self.driver.session() as session:
        result = session.run("""
            MATCH path = (start:ACTOR {id: $actor_id})-[*1..3]->(connected)
            WHERE start <> connected
            RETURN path
        """, actor_id=start_node_id)
```

Graph Traversals

Finding friends of an actor:

```
def find_friends(self, actor_id):
    with self.driver.session() as session:
        result = session.run("""
            MATCH (start:ACTOR {id: $actor_id})
            MATCH (start)-[:KNOWS*1..]->(friend:ACTOR)
            RETURN DISTINCT friend.name as name
        """, actor_id=actor_id)
        return [record["name"] for record in result]
```

Graph Traversals

Finding actor's movies:

```
def find_actor_movies(self, actor_id):
    with self.driver.session() as session:
        result = session.run("""
            MATCH (a:ACTOR {id: $actor_id})<-[:PLAY]-
(m:MOVIE)
            RETURN m.title as movie, m.year as year
        """, actor_id=actor_id)
        return [(record["movie"], record["year"]) for record in
result]
```

Advanced Cypher Queries

Finding co-actors:

```
def find_coactors(self, movie_id):
    with self.driver.session() as session:
        result = session.run("""
            MATCH (m:MOVIE {id: $movie_id})-[:PLAY]->(a:ACTOR)
            RETURN a.name as name, a.year as year
            ORDER BY a.year
        """, movie_id=movie_id)
        return [(record["name"], record["year"]) for record in result]
```

Advanced Cypher Queries

Finding friends who played in the same movies:

```
def find_friends_in_movies(self):
    with self.driver.session() as session:
        result = session.run("""
            MATCH (a1:ACTOR)-[:KNOWS]-(a2:ACTOR)
            MATCH (m:MOVIE)-[:PLAY]->(a1)
            MATCH (m)-[:PLAY]->(a2)
            RETURN a1.name as actor1, a2.name as actor2, m.title as movie
        """)
        return [(r["actor1"], r["actor2"], r["movie"]) for r in result]
```

Advanced Cypher Queries

Movie statistics:

```
def get_movie_stats(self):
    with self.driver.session() as session:
        result = session.run("""
            MATCH (m:MOVIE)-[:PLAY]->(a:ACTOR)
            WITH m, COUNT(a) as actor_count
            RETURN m.title as title, m.year as year,
                   actor_count as actors
            ORDER BY m.year DESC
        """)
        return [(r["title"], r["year"], r["actors"]) for r in result]
```

Usage Example:

```
def main():
    app = MovieGraphApp()
    try:
        if app.verify_connectivity():
            # Create graph
            app.create_actors()
            app.create_movies()
            app.create_relationships()

            # Query examples
            print("\nMovie Statistics:")
            stats = app.get_movie_stats()
            for title, year, actors in stats:
                print(f"{title} ({year}): {actors} actors")
```

Usage Example (cont):

```
            print("\nFriends who played together:")
            collaborations = app.find_friends_in_movies()
            for actor1, actor2, movie in collaborations:
                print(f"{actor1} and {actor2} in {movie}")
    finally:
        app.close()


if __name__ == "__main__":
    main()
```

Create indexes

```
def create_indexes(self):
    """Create indexes for better query performance."""
    with self.driver.session() as session:
        # Create constraint that serves as index
        session.run("""
            CREATE CONSTRAINT actor_id IF NOT EXISTS
            FOR (a:ACTOR) REQUIRE a.id IS UNIQUE
        """)
        session.run("""
            CREATE CONSTRAINT movie_id IF NOT EXISTS
            FOR (m:MOVIE) REQUIRE m.id IS UNIQUE
        """)
        # Create index for frequently queried property
        session.run("""
            CREATE INDEX actor_name IF NOT EXISTS
            FOR (a:ACTOR) ON (a.name)
        """)
```