Neo4j Cypher 고급 패턴 매칭

1. 가변 길이 경로(Variable-Length Paths)

가변 길이 경로는 그래프 내에서 노드 간의 간접적인 연결을 탐색할 때 매우 유용합니다. 다음과 같은 형태로 사용됩니다:

```
-[관계타입*최소길이..최대길이]->
```

구문 설명

- `*` : 가변 길이 경로임을 나타내는 연산자
- `최소길이` : 탐색할 관계의 최소 개수 (생략 시 1)
- `최대길이` : 탐색할 관계의 최대 개수 (생략 시 무제한)

가변 길이 경로 예시 해설

```
MATCH (p1:Person {name: '홍길동'})-[r:KNOWS*1..3]->(p2:Person)
RETURN p1.name AS 시작점, [rel IN r | type(rel) + '(' + rel.since + ')'] AS 관계, p2.na
```

이 쿼리는:

1. 이름이 '홍길동'인 Person 노드(p1)를 찾습니다.
2. p1에서 시작하여 1~3단계 KNOWS 관계로 연결된 다른 Person 노드(p2)를 찾습니다.
3. 결과로 시작점(홍길동), 관계 정보(관계 타입과 since 속성), 도착점(연결된 Person)을 반환합니다.

관계 변수 처리 설명: `[rel IN r | ...]` 구문은 리스트 컴프리헨션(List Comprehension)이라고 하며, 경로에 포함된 모든 관계 r에 대해 각 관계 정보를 배열로 변환합니다.

2. 최단 경로(Shortest Path)

최단 경로 함수는 두 노드 간의 가장 짧은 경로를 찾는 Neo4j의 내장 함수입니다.

구문 설명

```
shortestPath((시작노드)-[관계패턴]-(도착노드))
```

- `shortestPath()` : 최단 경로를 찾는 함수
- `관계패턴` : 탐색할 관계 유형과 방향 (없으면 모든 관계 유형 탐색)

최단 경로 예시 해설

```
MATCH p=shortestPath((p1:Person {name: '홍길동'})-[*]-(p2:Person {name: '최민수'}))
RETURN [node IN nodes(p) | node.name] AS 경로
```

이 쿼리는:

1. 홍길동과 최민수 사이의 최단 경로를 찾습니다.
2. 모든 유형의 관계(`[*]`)를 고려합니다.
3. `nodes(p)` 함수를 사용하여 경로에 포함된 모든 노드를 배열로 추출합니다.
4. 각 노드의 name 속성만 추출하여 '경로'라는 이름으로 반환합니다.

참고: `shortestPath()` 함수는 기본적으로 BFS(너비 우선 탐색) 알고리즘을 사용하므로 가장 적은 수의 관계를 통과하는 경로를 찾습니다.

3. 집계 함수(Aggregation Functions)

집계 함수는 데이터의 통계적 분석을 위해 사용됩니다. Neo4j는 다양한 집계 함수를 제공합니다.

주요 집계 함수

- `COUNT()` : 레코드 수 계산
- `SUM()` : 숫자 값 합계
- `AVG()` : 숫자 값 평균
- `MIN()` : 최소값
- `MAX()` : 최대값
- `COLLECT()` : 여러 값을 배열로 수집

집계 함수 예시 해설

```
MATCH (p:Person)-[:LIKES]->(i:Interest)
RETURN i.name AS 관심분야, COUNT(p) AS 사람수
ORDER BY 사람수 DESC
```

이 쿼리는:

1. Person 노드와 LIKES 관계로 연결된 Interest 노드를 찾습니다.
2. 각 Interest 노드에 대해 LIKES 관계로 연결된 Person 노드 수를 계산합니다.
3. 결과를 사람수 기준으로 내림차순 정렬합니다.

이 집계 패턴은 "각 그룹별 통계"를 계산할 때 자주 사용됩니다.

4. WITH 절

WITH 절은 쿼리를 여러 부분으로 나누고 중간 결과를 다음 부분으로 전달할 수 있게 해줍니다. SQL의 서브쿼리와 유사한 역할을 합니다.

구문 설명

```
MATCH (노드 패턴)
WITH 결과1, 결과2, ... [WHERE 조건]
MATCH (새로운 노드 패턴)
RETURN 최종결과
```

- WITH 절 다음에는 다음 단계로 전달할 변수들을 지정합니다.
- WITH 절 내에서 WHERE 절을 사용해 중간 결과를 필터링할 수 있습니다.

WITH 절 예시 해설

```
MATCH (p:Person)
WITH p.city AS 도시, AVG(p.age) AS 평균나이
WHERE 평균나이 >= 35
RETURN 도시, 평균나이
ORDER BY 평균나이 DESC
```

이 쿼리는:

1. 모든 Person 노드를 찾습니다.
2. WITH 절에서 도시별로 그룹화하여 평균 나이를 계산합니다.
3. 평균 나이가 35 이상인 도시만 필터링합니다.
4. 최종 결과로 도시와 평균 나이를 반환하고, 평균 나이 기준으로 내림차순 정렬합니다.

WITH 절은 복잡한 쿼리를 여러 논리적 단계로 나눌 수 있게 해주어 가독성을 높이고 쿼리 실행의 중간 결과를 처리할 수 있게 해줍니다.

고급 패턴 매칭 활용 팁

1. 경로 변수의 활용

경로 전체를 변수에 할당하면 해당 경로에 포함된 노드와 관계에 접근할 수 있는 함수를 활용할 수 있습니다:

- `nodes(path)` : 경로의 모든 노드 반환
- `relationships(path)` : 경로의 모든 관계 반환
- `length(path)` : 경로의 길이(관계 수) 반환

2. 패턴 조합 활용

여러 패턴을 조합해 복잡한 쿼리를 작성할 수 있습니다:

```
// 공통 관심사가 있고, 서로 2단계 이내로 연결된 사람들 찾기
MATCH (p1:Person)-[:LIKES]->(i:Interest)<-[:LIKES]-(p2:Person),
      path=shortestPath((p1)-[:KNOWS*1..2]-(p2))
WHERE p1.name < p2.name  // 중복 결과 방지
RETURN p1.name, p2.name, i.name AS 공통관심사, length(path) AS 거리
```

3. 서브쿼리와 유사한 패턴

WITH 절과 OPTIONAL MATCH를 활용해 SQL의 서브쿼리와 유사한 패턴을 구현할 수 있습니다:

```
// 각 사람의 친구 수와 방문한 장소 수 계산
MATCH (p:Person)
OPTIONAL MATCH (p)-[:KNOWS]->(friend)
WITH p, COUNT(friend) AS 친구수
OPTIONAL MATCH (p)-[:VISITED]->(place)
RETURN p.name AS 이름, 친구수, COUNT(place) AS 방문장소수
```

이러한 고급 패턴 매칭 기능들을 적절히 조합하면 복잡한 그래프 쿼리를 효율적으로 작성할 수 있습니다.