MEM GRAPH

Life of a Distributed Query

Teon Banek

November 9, 2018

About Me

Teon Banek

- Graduated from University of Zagreb, Faculty of Electrical Engineering and Computing
- Lead query engine developer at Memgraph
- Loves fencing, lasagne and black tea
- `teon.banek@memgraph.com`

About Us

Memgraph Ltd.

- Startup, founded in 2016
- Building a graph database
  - In-memory
  - High-performance
  - Distributed
- `https://memgraph.com`

Outline

1. About
2. OpenCypher Query Language
3. Semantic Analysis
4. Query Planning and Optimization
5. Query Execution

SQL

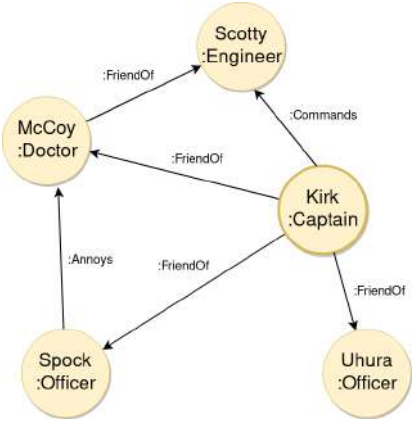
- An SQL query walks into a bar and sees two tables. It walks up to them and says ”May I join you?”

SQL

- An SQL query walks into a bar and sees two tables. It walks up to them and says ”May I join you?”
- ```
  SELECT * FROM a, (SELECT * FROM b, c
  WHERE b.rel_c = c.rel_b
  AND b.id NOT IN (SELECT id FROM d
  WHERE …
  )) WHERE …
  ```
- Joining tables produces very hard to read queries.
  ```

openCypher

```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```



Parsing

- antlr4
  - Generates a parser from BNF like grammar description.
- antrl4 AST → our custom AST
  - Allows for future support of other languages.
  - Makes potential antrl4 replacement easier.
- Literal and parameter stripping
  - Queries can be hashed and cached for reuse.

Semantic Analysis

- Various sanity checks:
  - trying to create the same element multiple times;
  - combining incompatible clauses (e.g. `UNION` and `UNION ALL`);
  - trying to use the same key twice to create a map;
  - etc.

Semantic Analysis

- Various sanity checks:
  - trying to create the same element multiple times;
  - combining incompatible clauses (e.g. `UNION` and `UNION ALL`);
  - trying to use the same key twice to create a map;
  - etc.
- Generating symbols for variables.
  - Validating variable scope and bindings.
  - Checking for type mismatches.

Symbol Generation

- For each variable a symbol is generated.
- Each symbols gets a space for its value on the *frame*.

Symbol Generation

- For each variable a symbol is generated.
- Each symbols gets a space for its value on the *frame*.
- Frame
  - Data structure (array) for storing values during execution.
  - Similar to a stack frame.
  - No dynamic allocation, so the size can be determined statically.

Symbol Generation

```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```

Symbol	Value
a	null

Symbol Generation

```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```

Symbol	Value
a	null
b	null

Symbol Generation

```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```

Symbol	Value
a	null
b	null
anon_edge	null

Symbol Generation

```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```

Symbol	Value
a	null
b	null
anon_edge	null
b (AS b)	null

Logical Operators

- Steps of work are described by *logical operators*.
- Similar, but different than operators of relational algebra found in SQL.

Logical Operators

- Steps of work are described by *logical operators*.
- Similar, but different than operators of relational algebra found in SQL.
- ```
  MATCH (a :Captain {name:"Kirk"})-[:FriendOf]->(b)
  WHERE b :Officer RETURN b.name AS b
  ```
  ```

Logical Operators

- Steps of work are described by *logical operators*.
- Similar, but different than operators of relational algebra found in SQL.
- `MATCH (a :Captain {name:"Kirk"})-[:FriendOf]->(b)`
  `WHERE b :Officer RETURN b.name AS b`
  - `ScanAll` – find all nodes in the graph

Logical Operators

- Steps of work are described by *logical operators*.
- Similar, but different than operators of relational algebra found in SQL.
- `MATCH (a :Captain {name:"Kirk"})-[:FriendOf]->(b)`
  `WHERE b :Officer RETURN b.name AS b`
  - `ScanAll` – find all nodes in the graph
  - `Expand` – traverse all edges from a node

Logical Operators

- Steps of work are described by *logical operators*.
- Similar, but different than operators of relational algebra found in SQL.
- `MATCH (a :Captain {name:"Kirk"})-[:FriendOf]->(b)`
  `WHERE b :Officer RETURN b.name AS b`
  - `ScanAll` – find all nodes in the graph
  - `Expand` – traverse all edges from a node
  - `Filter` – apply a filter expression

Logical Operators

- Steps of work are described by *logical operators*.
- Similar, but different than operators of relational algebra found in SQL.
- `MATCH (a :Captain {name:"Kirk"})-[:FriendOf]->(b)`
  `WHERE b :Officer RETURN b.name AS b`
  - `ScanAll` – find all nodes in the graph
  - `Expand` – traverse all edges from a node
  - `Filter` – apply a filter expression
  - `Produce` – expressions to produce results

Extracting Filter Information

- Extracting filters into regular form.
- ```
  MATCH (a :Captain {name:"Kirk"})-[:FriendOf]->(b)
  WHERE b :Officer
  RETURN b.name AS b
  ```
  ```

Extracting Filter Information

- Extracting filters into regular form.
- ```
  MATCH (a) -[:FriendOf]-> (b)
  WHERE a :Captain AND a.name = "Kirk"
  AND b :Officer
  RETURN b.name AS b
  ```
  ```

Extracting Filter Information

- Extracting filters into regular form.
- ```
  MATCH (a) -[:FriendOf]-> (b)
  WHERE a :Captain AND a.name = "Kirk"
  AND b :Officer
  RETURN b.name AS b
  ```
  ```

Extracting Filter Information

- Extracting filters into regular form.
- ```
  MATCH (a) -[anon_edge]-> (b)
  WHERE a :Captain AND a.name = "Kirk"
  AND b :Officer
  AND anon_edge :FriendOf
  RETURN b.name AS b
  ```
  ```

Extracting Filter Information

- Extracting filters into regular form.
- ```
  MATCH (a) -[anon_edge]-> (b)
  WHERE a :Captain AND a.name = "Kirk"
  AND b :Officer
  AND anon_edge :FriendOf
  RETURN b.name AS b
  ```
- Collecting information on symbols used in expressions.
  - We want to apply filters as soon as possible.
  - Potentially replace with index lookup.
  ```

How to Match?

- We can alter the order of matching and estimate the best one.

How to Match?

- We can alter the order of matching and estimate the best one.
  - `MATCH (a) -[:FriendOf]-> (b)`

How to Match?

- We can alter the order of matching and estimate the best one.
  - `MATCH (a) -[:FriendOf]-> (b)`
  - `MATCH (b) <-[:FriendOf]- (a)`

How to Match?

- We can alter the order of matching and estimate the best one.
  - `MATCH (a) -[:FriendOf]-> (b)`
  - `MATCH (b) <-[:FriendOf]- (a)`
  - `MATCH (a), (b), (a) -[:FriendOf]-> (b)`

How to Match?

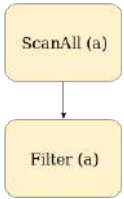
- We can alter the order of matching and estimate the best one.
  - `MATCH (a) -[:FriendOf]-> (b)`
  - `MATCH (b) <-[:FriendOf]- (a)`
  - `MATCH (a), (b), (a) -[:FriendOf]-> (b)`
  - `MATCH (a), (b), (b) <-[:FriendOf]- (a)`

`MATCH (a)-[:FriendOf]->(b)`

- `ScanAll` for `a`

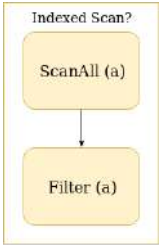
`MATCH (a)-[:FriendOf]->(b)`

- `ScanAll` for `a`
- Filter based on `a`



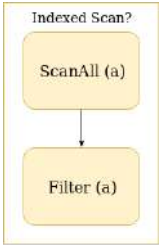
`MATCH (a)-[:FriendOf]->(b)`

- `ScanAll` for `a`
- Filter based on `a`
- Can we replace `ScanAll` + `Filter` with index?



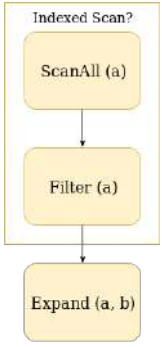
`MATCH (a)-[:FriendOf]->(b)`

- `ScanAll` for `a`
- Filter based on `a`
- Can we replace `ScanAll` + `Filter` with index?
  - Filter suitable for indexed lookup?
  - Index exists?



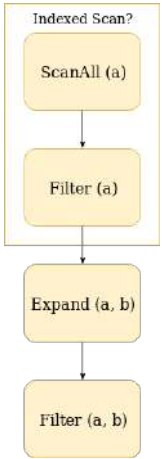
`MATCH (a)-[:FriendOf]->(b)`

- `ScanAll` for `a`
- Filter based on `a`
- Can we replace `ScanAll` + `Filter` with index?
  - Filter suitable for indexed lookup?
  - Index exists?
- `Expand` from `a` to `b`



`MATCH (a)-[:FriendOf]->(b)`

- `ScanAll` for `a`
- Filter based on `a`
- Can we replace `ScanAll` + `Filter` with index?
  - Filter suitable for indexed lookup?
  - Index exists?
- `Expand` from `a` to `b`
- Filter based on `a` and `b`

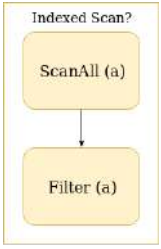


MATCH (a), (b), (a)-[:FriendOf]->(b)

- `ScanAll` for `a`

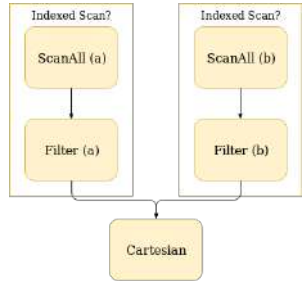
`MATCH (a), (b), (a)-[:FriendOf]->(b)`

- `ScanAll` for `a`
- Filter based on `a`, potentially index



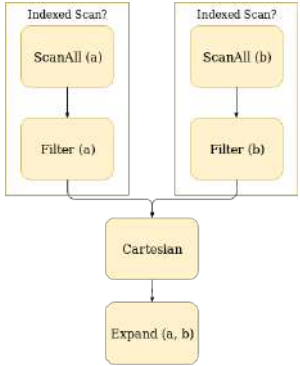
`MATCH (a), (b), (a)-[:FriendOf]->(b)`

- `ScanAll` for `a`
- Filter based on `a`, potentially index
- Same as above for finding `b`



`MATCH (a), (b), (a)-[:FriendOf]->(b)`

- `ScanAll` for `a`
- Filter based on `a`, potentially index
- Same as above for finding `b`
- `Expand` from `a` to `b`
  - Immediately produces edges connected from `a` to `b`



Plan Cost Estimation

- Estimate the cost of each operator and the total cost, based on:
  - cardinality increase/reduction
  - execution cost

Plan Cost Estimation

- Estimate the cost of each operator and the total cost, based on:
  - cardinality increase/reduction
  - execution cost
- Two sub-plans for matching:
  1. *ScanAll*(*indexed*) $\rightarrow$ *Expand* $\rightarrow$ *Filter*
  2. *ScanAll*(*indexed*) $\rightarrow$ *ScanAll*(*indexed*) $\rightarrow$ *Cartesian* $\rightarrow$ *Expand*

Plan Cost Estimation

- Estimate the cost of each operator and the total cost, based on:
  - cardinality increase/reduction
  - execution cost
- Two sub-plans for matching:
  1. *ScanAll*(*indexed*) $\rightarrow$ *Expand* $\rightarrow$ *Filter*
  2. *ScanAll*(*indexed*) $\rightarrow$ *ScanAll*(*indexed*) $\rightarrow$ *Cartesian* $\rightarrow$ *Expand*
- Scanned vertices degrees vs indexed lookup
  - If degree is low, 1st plan has lower cost.
  - Otherwise, the 2nd plan will be better.

Planning Distributed Query

- Worker machines store a sub-graph.
- Each worker can produce a subset of results.

Planning Distributed Query

- Worker machines store a sub-graph.
- Each worker can produce a subset of results.
- *ScanAll*(*indexed*) → *Expand* → *Filter*
  - Final results are merged on master machine.
  - `Expand` may need to communicate with other workers.

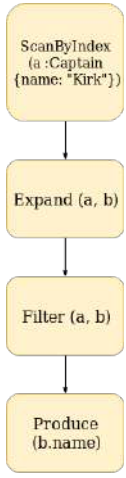
Planning Distributed Query

- Worker machines store a sub-graph.
- Each worker can produce a subset of results.
- *ScanAll*(*indexed*) → *Expand* → *Filter*
  - Final results are merged on master machine.
  - `Expand` may need to communicate with other workers.
- *ScanAll*(*indexed*) → *ScanAll*(*indexed*) → *Cartesian* → *Expand*
  - Master needs to get the Cartesian of `ScanAll` to execute `Expand`.
  - May cause potentially high memory consumption or workload.
  - No need for communication between worker machines.

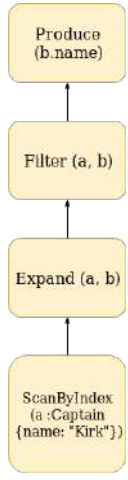
Planning Distributed Query

- Worker machines store a sub-graph.
- Each worker can produce a subset of results.
- *ScanAll*(*indexed*) → *Expand* → *Filter*
  - Final results are merged on master machine.
  - `Expand` may need to communicate with other workers.
- *ScanAll*(*indexed*) → *ScanAll*(*indexed*) → *Cartesian* → *Expand*
  - Master needs to get the Cartesian of `ScanAll` to execute `Expand`.
  - May cause potentially high memory consumption or workload.
  - No need for communication between worker machines.
- Cost estimator will need to estimate communication overhead.

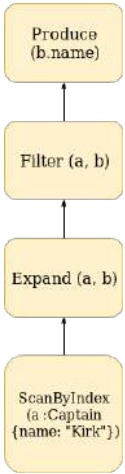
Final Plan


ScanByIndex
(a :Captain
{name: "Kirk"})
Expand (a, b)
Filter (a, b)
Produce
(b.name)

Final Plan


Produce
(b.name)
Filter (a, b)
Expand (a, b)
ScanByIndex
(a :Captain
{name: "Kirk"})

Final Plan



But how do we execute it?

Pull Mechanism

- Iterative approach
  - Each operator produces a `Cursor` (iterator).
  - Calling `Pull` on top of the plan cursor produces a single result.

Pull Mechanism

- Iterative approach
  - Each operator produces a `Cursor` (iterator).
  - Calling `Pull` on top of the plan cursor produces a single result.
- Lazy evaluation saves memory.
- Limiting or skipping results is natural.

Pull Mechanism

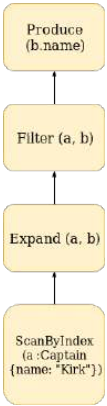
- Iterative approach
  - Each operator produces a `Cursor` (iterator).
  - Calling `Pull` on top of the plan cursor produces a single result.
- Lazy evaluation saves memory.
- Limiting or skipping results is natural.
- But some operations don't play nice:
  - ordering results and
  - CRUD operations.

Execution

```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```

Execution

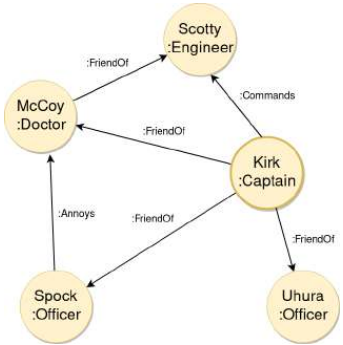
```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```

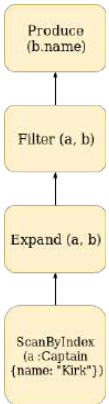


Symbol	Value
a	null
b	null
anon_edge	null
b (AS b)	null

Execution

```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```

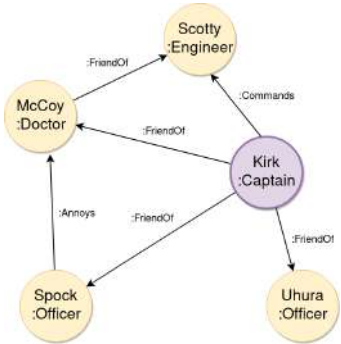


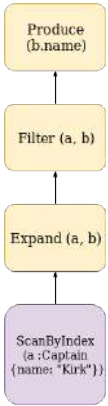


Symbol	Value
a	null
b	null
anon_edge	null
b (AS b)	null

Execution

```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```

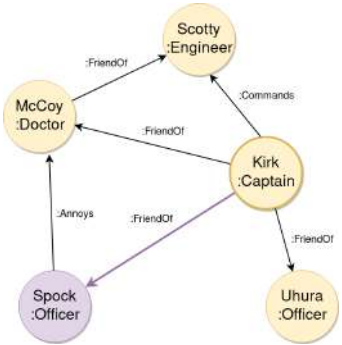


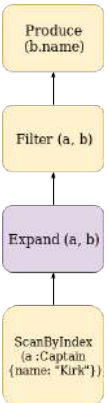


Symbol	Value
a	Kirk
b	null
anon_edge	null
b (AS b)	null

Execution

```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```

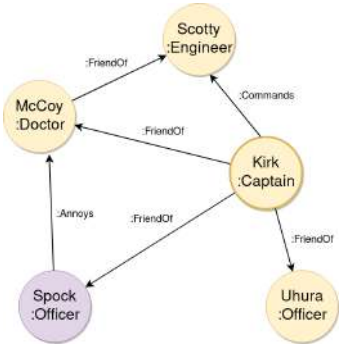


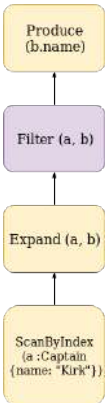


Symbol	Value
a	Kirk
b	Spock
anon_edge	Kirk, Spock
b (AS b)	null

Execution

```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```

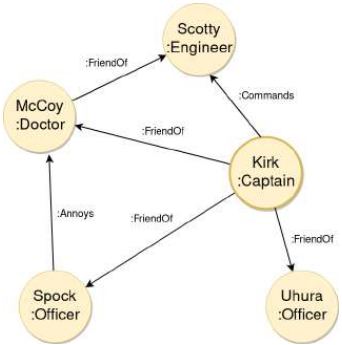


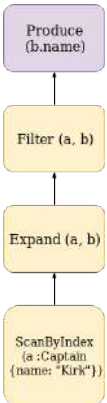


Symbol	Value
a	Kirk
b	Spock
anon_edge	Kirk, Spock
b (AS b)	null

Execution

```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```

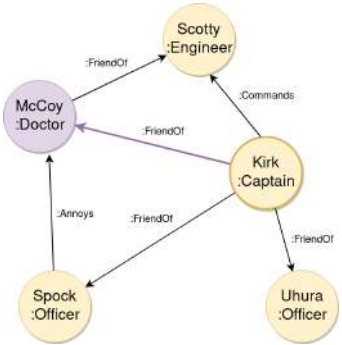


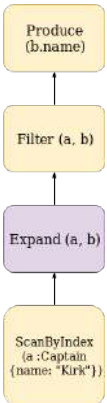


Symbol	Value
a	Kirk
b	Spock
anon_edge	Kirk, Spock
b (AS b)	"Spock"

Execution

```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```

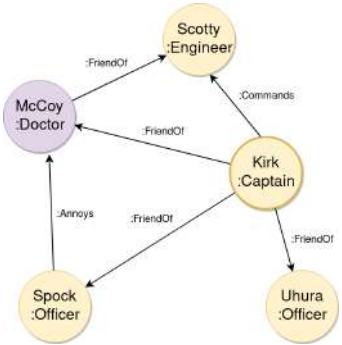


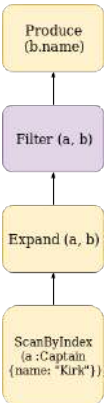


Symbol	Value
a	Kirk
b	McCoy
anon_edge	Kirk, McCoy
b (AS b)	"Spock"

Execution

```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```

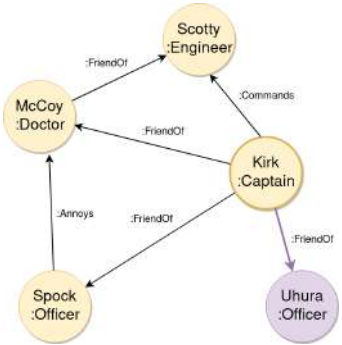


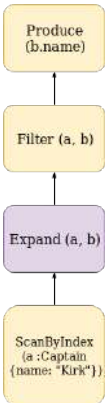


Symbol	Value
a	Kirk
b	McCoy
anon_edge	Kirk, McCoy
b (AS b)	"Spock"

Execution

```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```

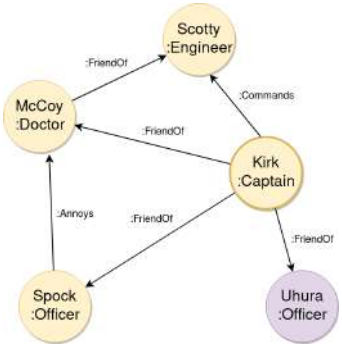


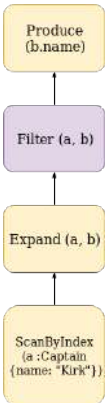


Symbol	Value
a	Kirk
b	Uhura
anon_edge	Kirk, Uhura
b (AS b)	"Spock"

Execution

```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```

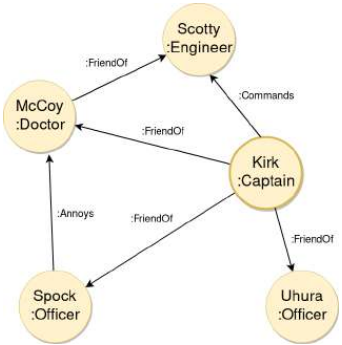


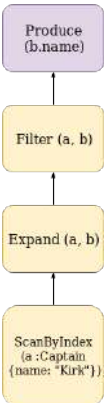


Symbol	Value
a	Kirk
b	Uhura
anon_edge	Kirk, Uhura
b (AS b)	"Spock"

Execution

```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```

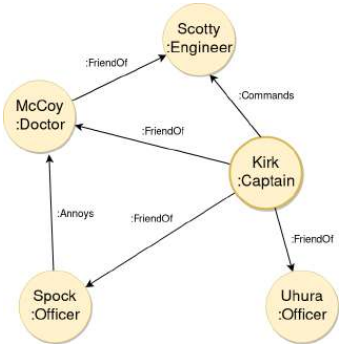


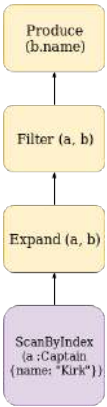


Symbol	Value
a	Kirk
b	Uhura
anon_edge	Kirk, Uhura
b (AS b)	"Uhura"

Execution

```
MATCH (a :Captain {name: "Kirk"}) -[:FriendOf]-> (b)
WHERE b :Officer RETURN b.name AS b
```

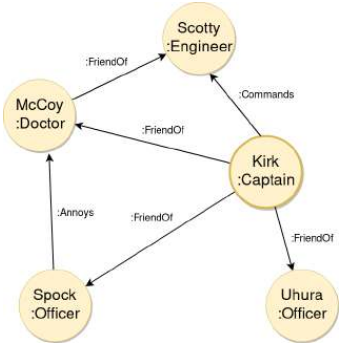




Symbol	Value
a	Kirk
b	Uhura
anon_edge	Kirk, Uhura
b (AS b)	"Uhura"

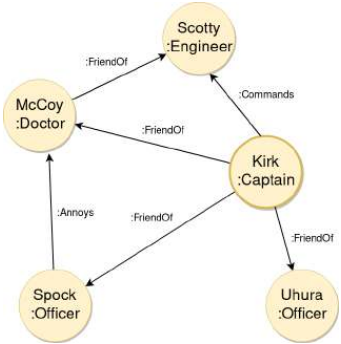
CRUD Execution

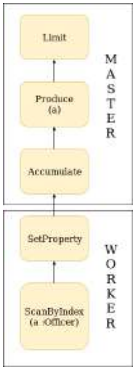
```
MATCH (a :Officer) SET a.arrogant = true
RETURN a LIMIT 1
```



CRUD Execution

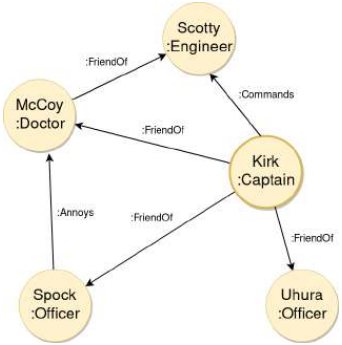
```
MATCH (a :Officer) SET a.arrogant = true
RETURN a LIMIT 1
```

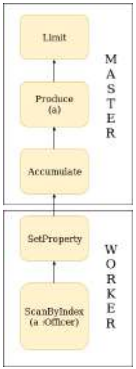




CRUD Execution

```
MATCH (a :Officer) SET a.arrogant = true
RETURN a LIMIT 1
```

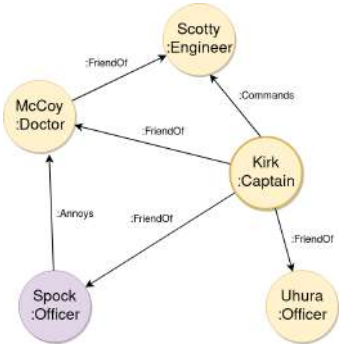




Symbol	Value
a	null

CRUD Execution

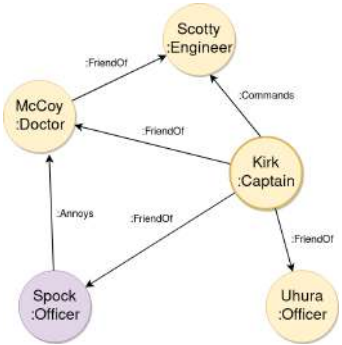
```
MATCH (a :Officer) SET a.arrogant = true
RETURN a LIMIT 1
```



Symbol	Value
a	Spock

CRUD Execution

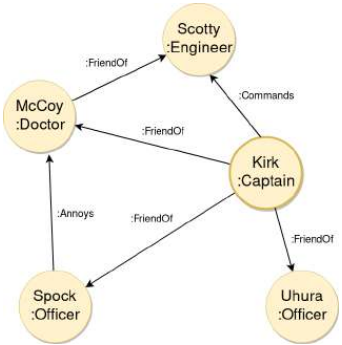
```
MATCH (a :Officer) SET a.arrogant = true
RETURN a LIMIT 1
```

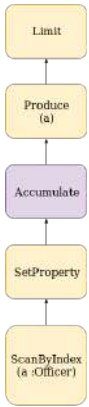


Symbol	Value
a	Spock

CRUD Execution

```
MATCH (a :Officer) SET a.arrogant = true
RETURN a LIMIT 1
```

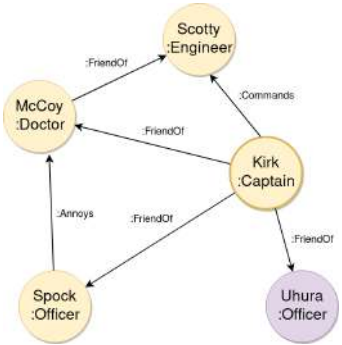




Symbol	Value
a	Spock

CRUD Execution

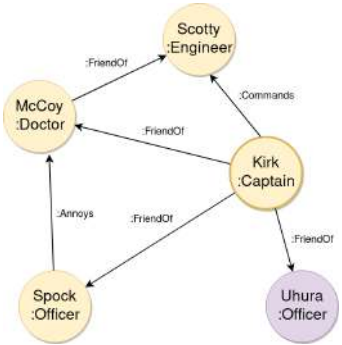
```
MATCH (a :Officer) SET a.arrogant = true
RETURN a LIMIT 1
```



Symbol	Value
a	Uhura

CRUD Execution

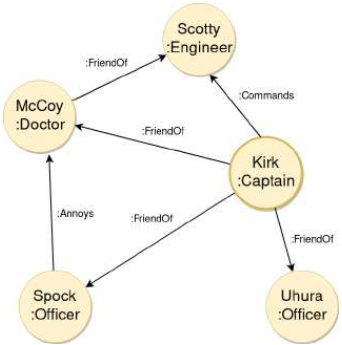
```
MATCH (a :Officer) SET a.arrogant = true
RETURN a LIMIT 1
```

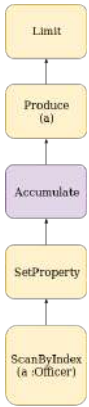


Symbol	Value
a	Uhura

CRUD Execution

```
MATCH (a :Officer) SET a.arrogant = true
RETURN a LIMIT 1
```

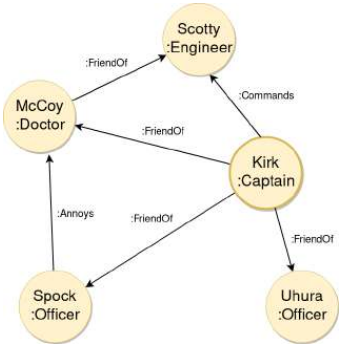


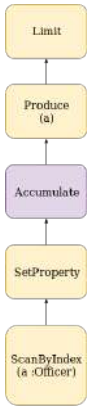


Symbol	Value
a	Uhura

CRUD Execution

```
MATCH (a :Officer) SET a.arrogant = true
RETURN a LIMIT 1
```

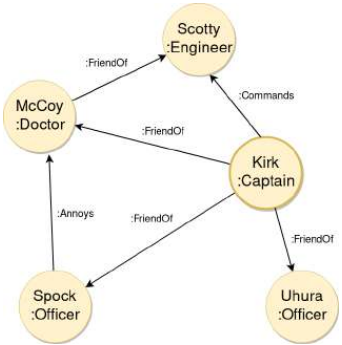




Symbol	Value
a	Spock

CRUD Execution

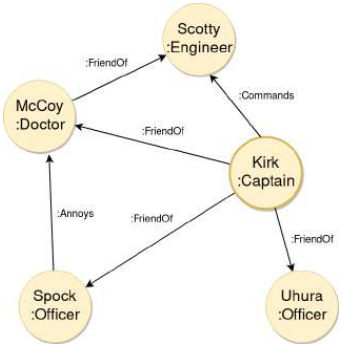
```
MATCH (a :Officer) SET a.arrogant = true
RETURN a LIMIT 1
```



Symbol	Value
a	Spock

CRUD Execution

```
MATCH (a :Officer) SET a.arrogant = true
RETURN a LIMIT 1
```



Symbol	Value
a	Spock

The End

- Thank you for your attention!

The End

- Thank you for your attention!
- Do you have any questions?

The End

- Thank you for your attention!
- Do you have any questions?
- We are hiring: `careers@memgraph.com`