1 Queries

1.
```
MATCH (n :Person) // Node matching filtered on label
RETURN n.name     // property returned
```

2.
```
MATCH (n :Person :Outlaw) // Node matching filtered on labels
RETURN n.name             // property returned
```

When several node labels are mentioned in pattern matching, it's a conjunction : the node matched must possess all labels. That's why the result is different from query 1.

3.
```
MATCH (n {species : ‘dog’}) // Node matching filtered on properties
RETURN n                    // node returned (id, labels, properties)
```

If you don't want to deal with properties in pattern matching, you can generate instead

`MATCH (n) WHERE n.species = ‘dog’ RETURN n`

4.
```
MATCH (n)                // Node matching
WHERE n.bounty > 3000    // filtered by quantification on properties
RETURN COUNT (n.name)    // aggregation of results
```

I did not put a query for each possible operators (`<`, `>`, `<>`, `=`, `>=`, `<=`) but it would be good that you have 1 variant for each of them.

5.
```
MATCH (n)                       // Node matching
WHERE n.name STARTS WITH "J"    // filtered by string matching on properties
RETURN COUNT (n.name)           // aggregation of results
```

For constraining strings, `STARTS WITH`, `ENDS WITH` and `CONTAINS` should be enough. There is also regular expressions but it seems difficult to be expressed with natural language.

6.
```
MATCH (n)                     // Node matching
WHERE n.species IS NOT NULL   // filtered by properties existence checking
RETURN COUNT (n.name)         // aggregation of results
```

To know whether a node has a property, `IS NULL`, `IS NOT NULL` may be used.

7.
```
MATCH () -[:PARENTS] -> (n)      // relationship matching filtered on types
RETURN n.name ORDER BY (n.size) // ordering of results
```

The order is ascending by default but 'descending' (or just 'desc') may be specified if otherwise is preferable.

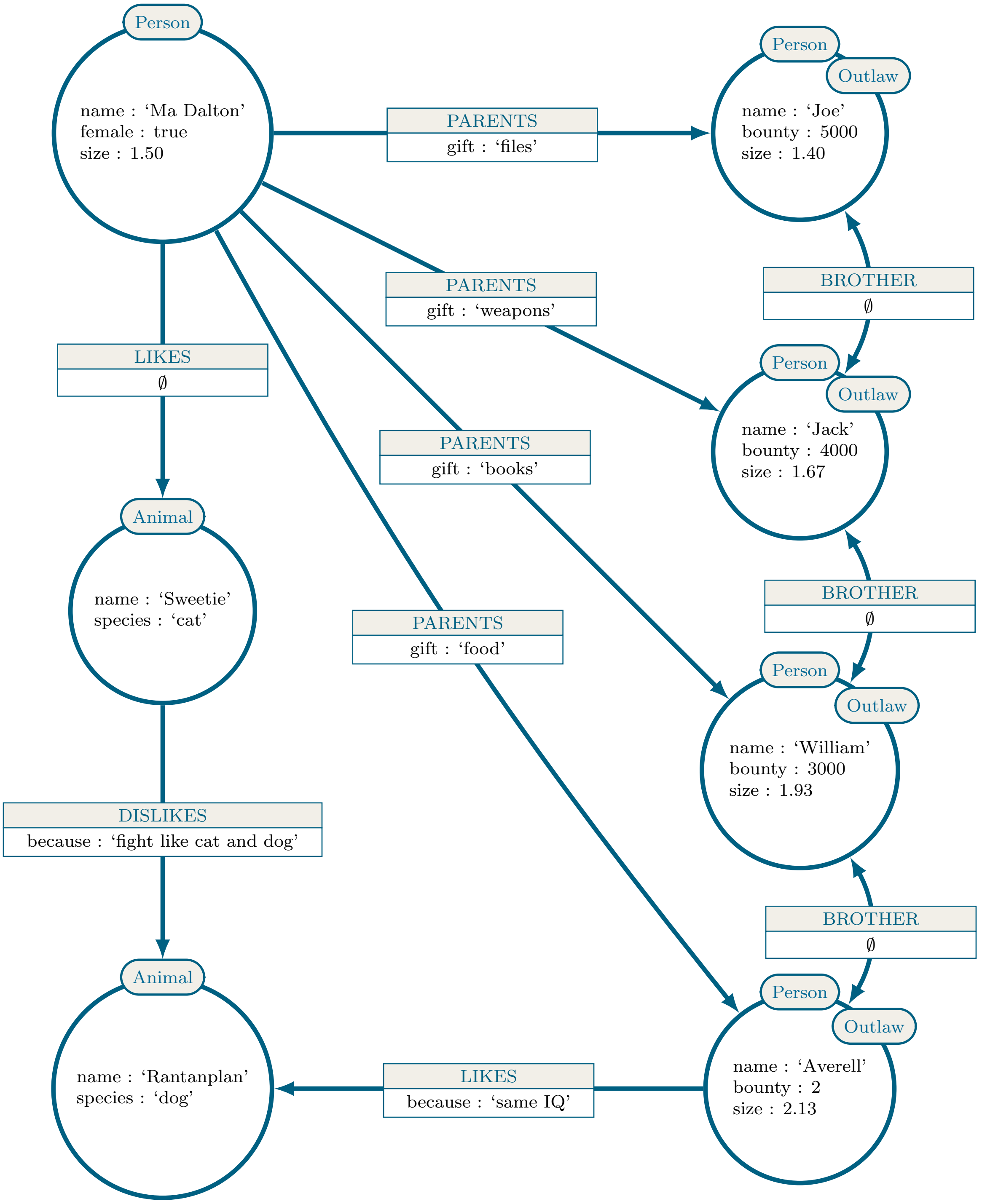
8.
```
MATCH ({name : ‘Joe’})
  -[:BROTHER *]-> (m)      // path matching on similar edge types
RETURN DISTINCT m.name     // removing duplicate
```

The star means an undefined number of hop with relationships matched (here, all of them must have type `BROTHER`). You can specify the number of hop (`[:BROTHER *2]`) or a range (`[:BROTHER *2..4]`).

9.
```
MATCH ({name : ‘Joe’})
  <-[:BROTHER | PARENTS *]-> (m)  // path matching on similar edge types
RETURN DISTINCT m.name            // removing duplicate
```

When several edge types are mentioned in pattern matching, it's a disjunction : the edge matched must possess one type in the list. It is the opposite for nodes (see query 2). This comes from the fact that a node have a list of labels but an edge must have one and only one type.

2 Graph


Person
name : ‘Ma Dalton’
female : true
size : 1.50
PARENTS
gift : ‘files’
Person
Outlaw
name : ‘Joe’
bounty : 5000
size : 1.40
PARENTS
gift : ‘weapons’
BROTHER
∅
LIKES
∅
Person
Outlaw
name : ‘Jack’
bounty : 4000
size : 1.67
PARENTS
gift : ‘books’
Animal
name : ‘Sweetie’
species : ‘cat’
PARENTS
gift : ‘food’
BROTHER
∅
Person
Outlaw
name : ‘William’
bounty : 3000
size : 1.93
DISLIKES
because : ‘fight like cat and dog’
BROTHER
∅
Animal
name : ‘Rantanplan’
species : ‘dog’
LIKES
because : ‘same IQ’
Person
Outlaw
name : ‘Averell’
bounty : 2
size : 2.13

3 Graph creation

If you want to create the graph above to test the queries, here is what you need to input.

```
CREATE
(Ma :Person {name : 'Ma Dalton', female : true, size : 1.50}),
(Sweetie : Animal {name : 'Sweetie', species : 'cat'}),
(Rantanplan : Animal {name : 'Rantanplan', species : 'dog'}),
(Joe : Person : Outlaw  {name : 'Joe', bounty : 5000, size : 1.40}),
(Jack : Person : Outlaw  {name : 'Jack', bounty : 4000, size : 1.67}),
(William :Person :Outlaw  {name : 'William', bounty : 3000, size : 1.93}),
(Averell :Person :Outlaw  {name : 'Averell', bounty : 2, size : 2.13}),
(Ma) -[:LIKES]-> (Sweetie),
(Sweetie) -[:DISLIKES {because : 'fight like cat and dog'}]-> (Rantanplan),
(Averell) -[:LIKES {because : 'same IQ'}]-> (Rantanplan),
(Ma) -[:PARENTS {gift : 'files'}]->  (Joe),
(Ma) -[:PARENTS {gift : 'weapons'}]-> (Jack),
(Ma) -[:PARENTS {gift : 'books'}]->  (William),
(Ma) -[:PARENTS {gift : 'food'}]->  (Averell),
(Joe) -[:BROTHER]-> (Jack),
(Jack) -[:BROTHER]-> (William),
(William) -[:BROTHER]-> (Averell),
(Joe) <-[:BROTHER]- (Jack),
(Jack) <-[:BROTHER]- (William),
(William) <-[:BROTHER]- (Averell)
```