Week 10 Querying Databases: The Non-Relational Ways (2)

Assignment Solutions

Question 1

In Cypher, some of the SQL clauses are apparently "missing", such as SELECT, FROM, GROUP BY, and HAVING. However, Cypher is expressive enough to offer the functionalities of all these clauses. From what you have learned, what are the Cypher clauses that provide the corresponding functionalities of SELECT, FROM, GROUP BY, and HAVING, respectively?

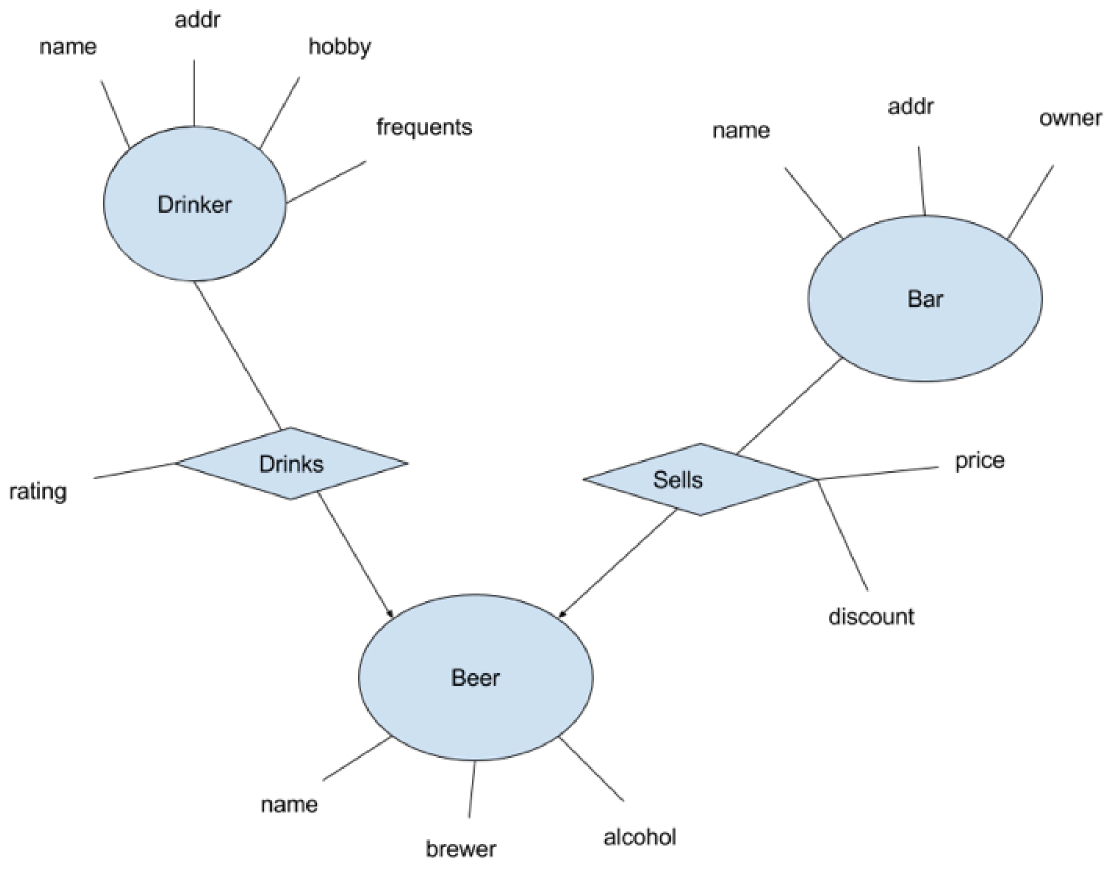
```
*A: RETURN, MATCH, RETURN or WITH, WITH...WHERE

B: MATCH, WITH, RETURN or WITH, MATCH

C: RETURN, MATCH, MATCH, WITH
```

Solution: A

Explanation: RETURN not only performs projection (like SELECT), but also specifies the grouping key as all the returned expressions that are not aggregate functions. WITH is similar to RETURN in that it does grouping as well as aggregation, but instead of terminating the query, it passes the result to the next stage in the pipeline. FROM is handled as part of the MATCH clause (actually MATCH is more like FROM...WHERE). Also note that in Cypher, WHERE cannot be used as a standalone clause, instead, it must be used as part of MATCH, OPTIONAL MATCH, or WITH. Since in SQL, the HAVING clause operates on the grouping result, like the pipeline stage $match in MongoDB, the corresponding Cypher clause is WITH...WHERE, in which WITH passes down the expressions (including aggregate functions), and WHERE does the filtering.

Question 2

The figure below shows the graph model for "Friday Night". Using Cypher, how can you find the names of all the bars where at least 3 different kinds of beers are available and a drinker called "Bob" frequents?



*A:

```
 MATCH (drinker:Drinker {name: "Bob", frequents: bar.name}),
(bar:Bar)-[s:Sells]->(beer:Beer) WITH bar.name AS bar_name,
count(beer) AS beer_count WHERE beer_count >= 3 RETURN bar_name
```

B:

```
 MATCH (drinker:Drinker {name:
"Bob"})-[:Drinks]->(beer:Beer)<-[:Sells]-(bar:Bar) WITH count(beer)
AS beer_count WHERE beer_count >= 3 RETURN bar.name
```

C:

```
 MATCH (:Drinker)-[:Drinks]->(beer:Beer)<-[:Sells]-(bar:Bar) WITH
bar.name AS bar_name, count(beer) AS beer_count WHERE beer_count >=
3 RETURN bar_name
```

Solution: A

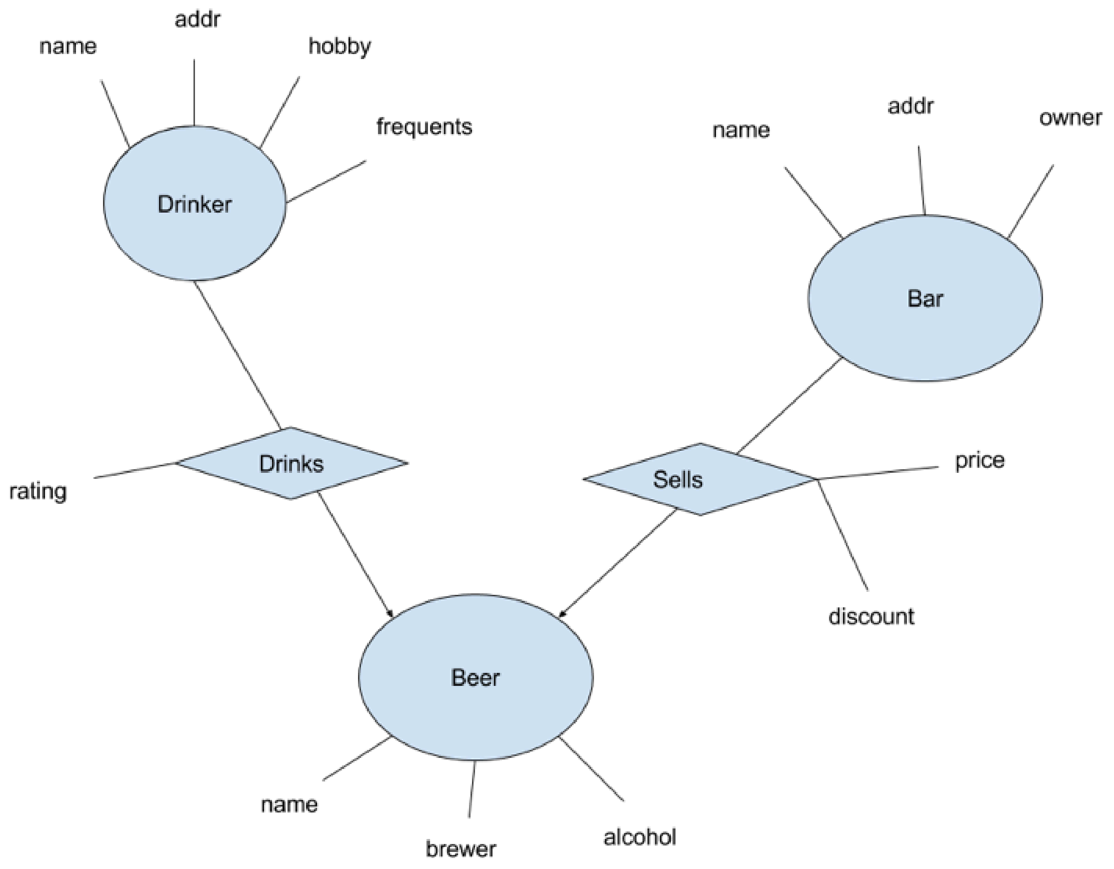
Explanation: For A, we first select the pattern `(:Bar)-[:Sells]->(:Beer)` to find the relation between bar and beer for condition: at least 3 different beers. Meantime, we select the bars where "Bob" frequents using `(drinker:Drinker {name: "Bob", frequents: bar.name}),` `(bar:Bar)` for condition: a drinker called "Bob" frequents. Then we group by bar_name and get the total number of different beers in each bar. Finally we return the names of the bars satisfying the two conditions.

For B, Variable `bar` becomes undefined after WITH and thus cannot be used in the RETURN clause.

For C, Missing the drinker name "Bob"

Question 3

The figure below shows the graph model for "Friday Night". Using Cypher, how can you find the beers (by name) sold at Sober Bar that have the highest alcohol content among all the beers sold at Sober Bar?



*A:

```
 MATCH (beer:Beer)<-[:Sells]-(:Bar {name: "Sober Bar"}) WITH
MAX(beer.alcohol) as max_alcohol MATCH (beer:Beer)<-[:Sells]-(:Bar
{name: "Sober Bar"}) WHERE beer.alcohol = max_alcohol RETURN
beer.name
```

B:

```
 MATCH (beer:Beer) WITH MAX(beer.alcohol) as max_alcohol MATCH
(beer:Beer)<-[:Sells]-(:Bar {name: "Sober Bar"}) WHERE beer.alcohol
= max_alcohol RETURN beer.name
```

C:

```
 MATCH (beer:Beer)<-[:Sells]-(:Bar {name: "Sober Bar"}) WITH
MAX(beer.alcohol) as max_alcohol MATCH (beer:Beer) WHERE
beer.alcohol = max_alcohol RETURN beer.name
```

Solution: A

Explanation: For A, First we find the max_alcohol among all the beers sold at Sober Bar. Then among the same set of Sober Bar beers, we find the beer whose alcohol level matches the maximum that we found in the previous stage. Note that we need to use WITH to pass the max_alcohol value down the pipeline. Side note: to avoid the two duplicate MATCH clauses, we can use collect() to "save" the list of beers sold at Sober Bar, and then UNWIND them in the second stage. In this case, using COLLECT() might look like an overkill, but can serve as a good excersise

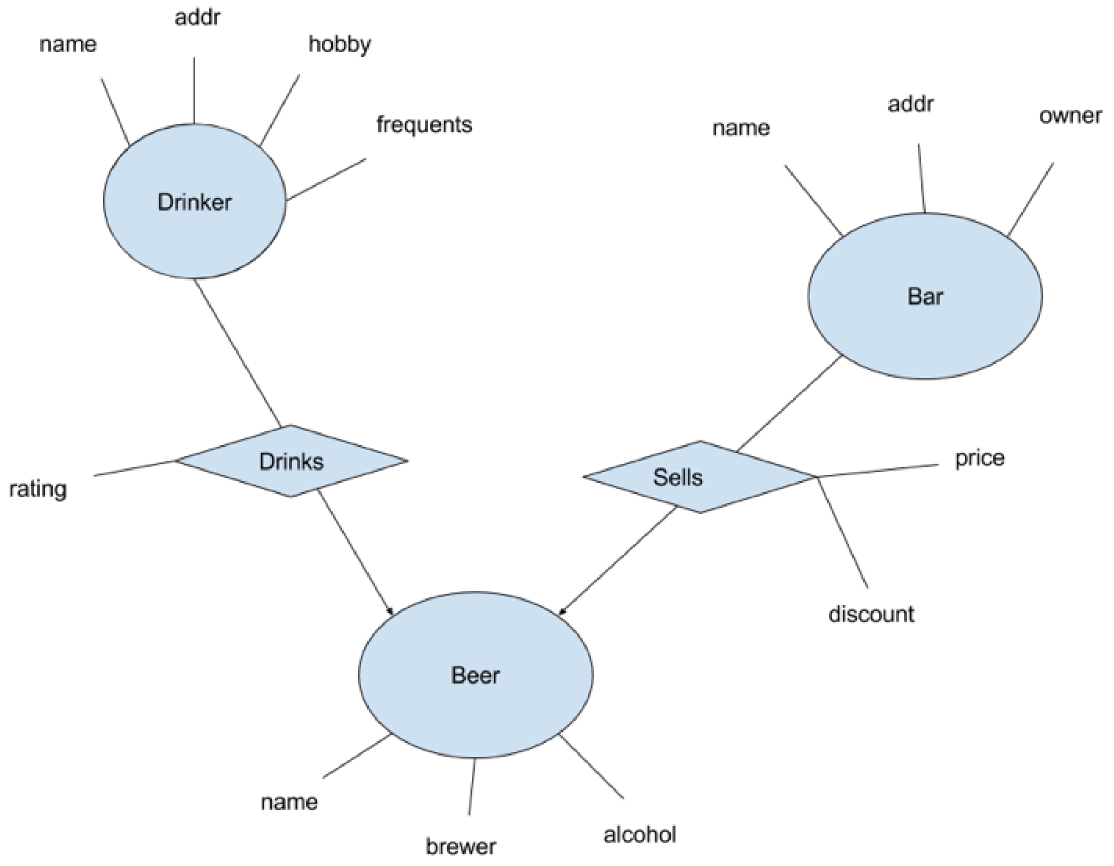
For B, The beers in the first MATCH clause are not necessarily from Sober Bar, and hence the max_alcohol value we get is the global max. If this global max is greater than the local max within Sober Bar, none of the beers sold at Sober Bar will match that level of alcohol and, therefore, nothing will be returned.

For C, The beers in the second MATCH clause are not necessarily from Sober Bar, so if a beer from another bar happens to have the same alcohol content as max_alcohol in Sober Bar, it will be returned as well, which violates the requirement.

Question 4

[Food for Thought] The figure below shows the graph model for "Friday Night". Using Cypher, how can you find the beers (by name) with the highest *average* rating?

Hint: learning how to use the COLLECT() function will be useful.



```
*A:

 MATCH (:Drinker)-[drinks:Drinks]->(beer:Beer) WITH beer.name AS
beer_name, AVG(drinks.rating) as avg_rating WITH COLLECT([beer_name,
avg_rating]) AS beer_avg_list, MAX(avg_rating) AS max_avg_rating
UNWIND beer_avg_list AS beer_avg WITH beer_avg[0] AS beer_name,
beer_avg[1] AS avg_rating, max_avg_rating WHERE avg_rating =
max_avg_rating RETURN beer_name

B:
 MATCH (:Drinker)-[drinks:Drinks]->(beer:Beer) WITH beer.name AS
beer_name, AVG(drinks.rating) as avg_rating WITH COLLECT([beer_name,
avg_rating]) AS beer_avg, MAX(avg_rating) AS max_avg_rating WITH
beer_avg[0] AS beer_name, beer_avg[1] AS avg_rating, max_avg_rating
WHERE avg_rating = max_avg_rating RETURN beer_name
```

C:

```
 MATCH (:Drinker)-[drinks:Drinks]->(beer:Beer) WITH beer.name AS
beer_name, AVG(drinks.rating) as avg_rating WITH beer_name,
avg_rating, MAX(avg_rating) AS max_avg_rating WHERE avg_rating =
max_avg_rating RETURN beer_name
```

Solution: A

Explanation: For A, In the first stage, we compute the average rating for each beer. In the second stage, we compute the max among all the average ratings, obtaining the highest average rating, and, at the same time, we save the "beer & avg_rating" relation obtained in stage one using COLLECT(). Finally, in the third stage, we find the beer(s) whose average rating matches the highest average rating. Note that before using the saved relation (as a single list), we must UNWIND the list first (similar to unwinding an array in MongoDB).

For B, Here the stored list resulting from COLLECT() is not unwound before being used in the next stage, which is illegal.

For C, The second WITH clause intends to obtain the highest average rating among all the average ratings, however, because beer_name and avg_rating appear in the same clause, they will be used as the grouping key, and MAX() is called on each group, which is just a single beer-rating pair, hence this does not work as intended.

Question 5

Translate the following SQL query into Neo4j Cypher. We are trying to figure out the names and prices of the coffee that Starbucks sold. Assume Neo4j has the following schema: node Shop(name), node Coffee(name), and link Sales(price).

SELECT c.name, s.price FROM Coffee c, Sales s WHERE c.name = s.coffee AND c.shop = 'Starbucks' AND s.price > 4

```
*A:

 MATCH (shop:Shop)-[s:Sales]->(c:Coffee) WHERE s.price > 4 AND
shop.name = 'Starbucks' RETURN c.name, s.price s

B:

 RETURN s.price, c.name MATCH (:Shop)-[s:Sales]->(c:Coffee) WHERE
s.price > 4

C:

 MATCH (c:Coffee {shop:'Starbucks'})-[s:Sales] WHERE s.price > 4
RETURN s.price, c.name
```

Solution: A

Explanation: For B, Here the Cypher syntax is wrong-- should be MATCH-WHERE-RETURN-- although most components from the SQL query are still there.

For C, Although we do not know what the Neo4j graph looks like, we can imagine that Sales would be a relationship between a coffee and a shop.