Tool: Neo4j

Data Definition Language (DDL)

CARDS (NODES)

```
LOAD CSV FROM 'file:///cards.csv' AS row FIELDTERMINATOR "|"

WITH toInteger(row[2]) AS convertedManaCost,

case row[3] when "\N" then null else toInteger(row[3]) end AS edhrecRank,

case row[7] when "\N" then null else toBoolean(toInteger(row[7])) end AS isReserved,

row[8] AS layout,

case row[9] when "\N" then null else row[9] end AS loyalty,

case row[10] when "\N" then null else row[10] end AS manaCost,

row[12] AS name,

case row[13] when "\N" then null else row[13] end AS power,

case row[14] when "\N" then null else row[14] end AS text,

case row[15] when "\N" then null else row[15] end AS toughness

CREATE (c:Card)

SET c.convertedManaCost = convertedManaCost, c.edhrecRank = edhrecRank,
c.isReserved = isReserved, c.layout = layout, c.loyalty = loyalty, c.manaCost = manaCost,
c.name = name, c.power = power, c.text = text, c.toughness = toughness;
```

COLORS (NODES)

```
LOAD CSV FROM 'file:///colors.csv' AS row FIELDTERMINATOR "|"

WITH row[2] AS color

MERGE (:Color {color: color});
```

COLORS (RELATIONSHIPS)

```
LOAD CSV FROM 'file:///colors.csv' AS row FIELDTERMINATOR "|"

WITH row[1] AS name, row[2] AS color

MATCH (c:Card {name: name})

MATCH (col:Color {color: color})

MERGE (c)-[:IS_OF_COLOR]->(col);
```

KEYWORDS (NODES)

```
LOAD CSV FROM 'file:///keywords.csv' AS row FIELDTERMINATOR "|"

WITH row[2] AS keyword

MERGE (:Keyword {keyword: keyword});
```

KEYWORDS (RELATIONSHIPS)

```
LOAD CSV FROM 'file:///keywords.csv' AS row FIELDTERMINATOR "|"

WITH row[1] AS name, row[2] AS keyword

MATCH (c:Card {name: name})

MATCH (key:Keyword {keyword: keyword})

MERGE (c)-[:HAS_KEYWORD]->(key);
```

SUBTYPES (NODES)

```
LOAD CSV FROM 'file:///subtypes.csv' AS row FIELDTERMINATOR "|"

WITH row[2] AS subtype

MERGE (:Subtype {subtype: subtype});
```

SUBTYPES (RELATIONSHIPS)

```
LOAD CSV FROM 'file:///subtypes.csv' AS row FIELDTERMINATOR "|"

WITH row[1] AS name, row[2] AS subtype

MATCH (c:Card {name: name})

MATCH (subt:Subtype {subtype: subtype})

MERGE (c)-[:HAS_SUBTYPE]->(subt);
```

SUPERTYPES (NODES)

```
LOAD CSV FROM 'file:///supertypes.csv' AS row FIELDTERMINATOR "|"

WITH row[2] AS supertype

MERGE (:Supertype {supertype: supertype});
```

SUPERTYPES (RELATIONSHIPS)

```
LOAD CSV FROM 'file:///supertypes.csv' AS row FIELDTERMINATOR "|"

WITH row[1] AS name, row[2] AS supertype

MATCH (c:Card {name: name})

MATCH (supert:Supertype {supertype: supertype})

MERGE (c)-[:HAS_SUPERTYPE]->(supert);
```

TYPES (NODES)

```
LOAD CSV FROM 'file:///types.csv' AS row FIELDTERMINATOR "|"

WITH row[2] AS type

MERGE (:Type {type: type});
```

TYPES (RELATIONSHIPS)

```
LOAD CSV FROM 'file:///types.csv' AS row FIELDTERMINATOR "|"

WITH row[1] AS name, row[2] AS type

MATCH (c:Card {name: name})

MATCH (t:Type {type: type})

MERGE (c)-[:HAS_TYPE]->(t);
```

LEGALITIES (NODES)

```
MERGE (f:Format {format: "Commander"});

MERGE (f:Format {format: "Historic"});

MERGE (f:Format {format: "Legacy"});

MERGE (f:Format {format: "Modern"});

MERGE (f:Format {format: "Pauper"});

MERGE (f:Format {format: "Standard"});

MERGE (f:Format {format: "Vintage"});
```

LEGALITIES (RELATIONSHIPS)

```
LOAD CSV FROM 'file:///legalities.csv' AS row FIELDTERMINATOR "|"

WITH row[1] AS name, row[2] AS commander

MATCH (c:Card {name: name})

MATCH (f:Format {format: "Commander"})

MERGE (c)-[:IS_LEGAL_IN {legality: commander} ]->(f);

LOAD CSV FROM 'file:///legalities.csv' AS row FIELDTERMINATOR "|"

WITH row[1] AS name, row[3] AS historic

MATCH (c:Card {name: name})

MATCH (f:Format {format: "Historic"})
```

```
MERGE (c)-[:IS_LEGAL_IN {legality: historic} ]->(f);

LOAD CSV FROM 'file:///legalities.csv' AS row FIELDTERMINATOR "|"

WITH row[1] AS name, row[4] AS legacy

MATCH (c:Card {name: name})

MATCH (f:Format {format: "Legacy"})

MERGE (c)-[:IS_LEGAL_IN {legality: legacy} ]->(f);

LOAD CSV FROM 'file:///legalities.csv' AS row FIELDTERMINATOR "|"

WITH row[1] AS name, row[5] AS modern

MATCH (c:Card {name: name})

MATCH (f:Format {format: "Modern"})

MERGE (c)-[:IS_LEGAL_IN {legality: modern} ]->(f);

LOAD CSV FROM 'file:///legalities.csv' AS row FIELDTERMINATOR "|"

WITH row[1] AS name, row[6] AS pauper

MATCH (c:Card {name: name})

MATCH (f:Format {format: "Pauper"})

MERGE (c)-[:IS_LEGAL_IN {legality: pauper} ]->(f);

LOAD CSV FROM 'file:///legalities.csv' AS row FIELDTERMINATOR "|"

WITH row[1] AS name, row[7] AS standard

MATCH (c:Card {name: name})

MATCH (f:Format {format: "Standard"})

MERGE (c)-[:IS_LEGAL_IN {legality: standard} ]->(f);

LOAD CSV FROM 'file:///legalities.csv' AS row FIELDTERMINATOR "|"

WITH row[1] AS name, row[8] AS vintage

MATCH (c:Card {name: name})

MATCH (f:Format {format: "Vintage"})

MERGE (c)-[:IS_LEGAL_IN {legality: vintage} ]->(f);
```

RULINGS (NODES)

```
LOAD CSV FROM 'file:///rulings.csv' AS row FIELDTERMINATOR "|"

WITH row[3] AS ruling

MERGE (:Ruling {ruling: ruling});
```

RULINGS (RELATIONSHIPS)

```
LOAD CSV FROM 'file:///rulings.csv' AS row FIELDTERMINATOR "|"

WITH row[1] AS name, date(row[2]) AS date, row[3] AS ruling, toInteger(row[4]) AS
dailycount

MATCH (c:Card {name: name})

MATCH (r:Ruling {ruling: ruling})

MERGE (c)-[:REFERRED_TO_BY {date: date, dailycount: dailycount}]->(r);
```

PRINTINGS (NODES)

```
LOAD CSV FROM 'file:///printings.csv' AS row FIELDTERMINATOR "|"

WITH row[2] AS frameVersion, toBoolean(toInteger(row[3])) AS isFullArt,
toBoolean(toInteger(row[4])) AS isPromo, toBoolean(toInteger(row[5])) AS isTextless, row[7]
AS rarity, row[9] AS uuid

CREATE (p:Printing)

SET p.frameVersion = frameVersion, p.isFullArt = isFullArt, p.isPromo = isPromo,
p.isTextless = isTextless, p.rarity = rarity, p.uuid = uuid;
```

ARTISTS (NODES)

```
LOAD CSV FROM 'file:///printings.csv' AS row FIELDTERMINATOR "|"

WITH case row[1] when "\N" then "NoArtist" else row[1] end AS name

MERGE (a:Artist {name: name});

MATCH (a:Artist {name: "NoArtist"}) DETACH DELETE a;
```

ARTISTS (RELATIONSHIPS)

```
LOAD CSV FROM 'file:///printings.csv' AS row FIELDTERMINATOR "|"

WITH row[1] AS artist, row[9] AS uuid

MATCH (a:Artist {name: artist})

MATCH (p:Printing {uuid: uuid})

MERGE (a)-[:REALIZED]->(p);
```

PRICES (PROPERTIES)

```
LOAD CSV FROM 'file:///prices.csv' AS row FIELDTERMINATOR "|"

WITH row[1] AS uuid,

case row[2] when "\N" then null else toFloat(row[2]) end AS normalPrice,

case row[3] when "\N" then null else toFloat(row[3]) end AS foilPrice

MATCH (p:Printing {uuid: uuid})

SET p.normalPrice = normalPrice, p.foilPrice = foilPrice;
```

BLOCKS (NODES)

```
LOAD CSV FROM 'file:///sets.csv' AS row FIELDTERMINATOR "|"

WITH case row[2] when "\N" then "NoBlock" else row[2] end AS block

MERGE (b:Block {name: block});

MATCH (b:Block {name: "NoBlock"}) DETACH DELETE b;
```

SETS (NODES)

```
LOAD CSV FROM 'file:///sets.csv' AS row FIELDTERMINATOR "|"

WITH row[3] AS code, toBoolean(toInteger(row[4])) AS isFoilOnly,
toBoolean(toInteger(row[5])) AS isOnlineOnly, row[6] AS name, date(row[7]) AS
releaseDate, toInteger(row[8]) AS totalSetSize, row[9] AS type

CREATE (s:Set)

SET s.code = code, s.isFoilOnly = isFoilOnly, s.isOnlineOnly = isOnlineOnly, s.name =
name, s.releaseDate = releaseDate, s.totalSetSize = totalSetSize, s.type = type;
```

SETS (RELATIONSHIPS)

```
LOAD CSV FROM 'file:///sets.csv' AS row FIELDTERMINATOR "|"

WITH row[2] AS block, row[3] AS code

MATCH (b:Block {name: block})

MATCH (s:Set {code: code})

MERGE (b)-[:CONTAINS]->(s);
```

PRINTINGS (RELATIONSHIPS)

```
LOAD CSV FROM 'file:///printings.csv' AS row FIELDTERMINATOR "|"

WITH row[8] AS setCode, row[9] AS uuid

MATCH (s:Set {code: setCode})

MATCH (p:Printing {uuid: uuid})

MERGE (p)-[:PRINTED_IN]->(s);

LOAD CSV FROM 'file:///printings.csv' AS row FIELDTERMINATOR "|"

WITH row[6] AS name, row[9] AS uuid

MATCH (c:Card {name: name})

MATCH (p:Printing {uuid: uuid})

MERGE (c)-[:PRINTED_AS]->(p);
```

Queries

1) Return the **cards** appearing in the **top 100** of the **EDHRec.com Ranking**, having at least one **printing** whose **normal price** is **less than one tenth** of the **average normal price** (order the result by **price**, in ascending order)

```
MATCH (p:Printing)
WITH avg(p.normalPrice) AS avgPrice
MATCH (c:Card)-[:PRINTED_AS]->(p)
WHERE c.edhrecRank <= 100 AND p.normalPrice < avgPrice/10
RETURN c.name AS name, min(p.normalPrice) AS minPrice
ORDER BY minPrice;
```

2) Return the **average text length** of the cards in each **block** of sets (order the result by **average text length**, in descending order)

```
MATCH
(c:Card)-[:PRINTED_AS]->(p:Printing)-[:PRINTED_IN]->(s:Set)<-[:CONTAINS]-(b:Block)
RETURN b.name AS Block, avg(size(c.text)) AS AvgTextLength
ORDER BY AvgTextLength DESC;
```

2a) Example: *size()* function in **Neo4j** VS *length()* function in **SQL**

```
// U+2014 EM DASH: —

// UTF-8: E2 80 94          (3 bytes)


// Neo4j:

// The function size() returns the number of Unicode characters in a string

RETURN size("—");

// Output: 1


// SQL:

// The function length() returns the number of bytes of data in character data

SELECT length("—");

// Output: 3
```

3) Return the **artists** that realized the **maximum number of printings** for each card **type** of cards having **common** or **uncommon** rarity (return **all the artists** in case of ties)

```
MATCH
(a:Artist)-[:REALIZED]->(p:Printing)<-[:PRINTED_AS]-(c:Card)-[:HAS_TYPE]->(t:Type)
WHERE p.rarity="common" OR p.rarity="uncommon"
WITH a.name AS Artist, t.type AS Type, count(p) AS count
WITH Type, max(count) AS Count
MATCH
(a:Artist)-[:REALIZED]->(p:Printing)<-[:PRINTED_AS]-(c:Card)-[:HAS_TYPE]->(t:Type)
WHERE p.rarity="common" OR p.rarity="uncommon"
WITH a.name AS Artist, t.type AS Type2, count(p) AS count, Type, Count
WHERE Type2 = Type AND count = Count
RETURN Type, Artist, Count
ORDER BY Count DESC;
```

4) Return the number of **new cards** for each **set**

```
MATCH (c:Card)-[:PRINTED_AS]->(p:Printing)-[:PRINTED_IN]->(s:Set)
WITH c.name AS Card, min(s.releaseDate) AS FirstReleaseDate
MATCH (c:Card {name: Card})-[:PRINTED_AS]->(p:Printing)-[:PRINTED_IN]->(s:Set
{releaseDate: FirstReleaseDate})
RETURN s.name AS setName, count(c) AS count
ORDER BY count DESC;
```

5) Return the **rulings** of the **cards** with at least a **supertype**, containing one or more of the cards **keywords**

```
MATCH (key:Keyword)<-[:HAS_KEYWORD]-(c:Card)-[:REFERRED_TO_BY]->(r:Ruling)
WHERE r.ruling =~ '(?i).*'+key.keyword+'.*' AND EXISTS
((c)-[:HAS_SUPERTYPE]->(:Supertype))
RETURN c.name AS card, key.keyword AS keyword, r.ruling AS ruling
ORDER BY c.name;
```

6) For each **color**, return the number of **cards** of type “**creature**” having both **power** and **toughness** larger or equal to **10** that are legal in “**Legacy**”

```
MATCH (t:Type {type: "Creature"})<-[:HAS_TYPE]-(c:Card)-[:IS_LEGAL_IN {legality:
"Legal"}]->(f:Format {format: "Legacy"}), (c)-[:IS_OF_COLOR]->(col:Color)
WHERE toInteger(c.power) >=10 AND toInteger(c.toughness) >=10
RETURN col.color AS color, count(c) AS count
ORDER BY count DESC;
```

7) For each **non-foil-only set**, return **name**, **size**, the **sum** of both the **normal** and **foil prices** (when **NOT NULL**) of each of its printings and the “**Foil Markup**” (**ratio** between **total foil** and **total normal price**)

```
MATCH (p:Printing)-[:PRINTED_IN]->(s:Set)
WHERE s.isFoilOnly = false
WITH s.name AS name, sum(p.normalPrice) AS totalNormalPrice, sum(p.foilPrice) AS
totalFoilPrice, CASE sum(p.normalPrice) WHEN 0 THEN NULL ELSE
sum(p.foilPrice)/sum(p.normalPrice) END AS FoilMarkup, s.totalSetSize AS setSize
WHERE FoilMarkup IS NOT NULL AND FoilMarkup <> 0
RETURN name, setSize, round(totalNormalPrice,2) AS totalNormalPrice,
round(totalFoilPrice,2) AS totalFoilPrice, round(FoilMarkup,2) AS FoilMarkup
ORDER BY FoilMarkup DESC;
```

8) Return the **uuid** and the **normal price** of all the **printings** released in the 21st Century, whose **price** is less than or equal to **10€**, of **cards** that have **any color symbol** in their **mana cost** but are **colorless** (order the result by **normal price**, in descending order)

```
MATCH (c:Card)-[:PRINTED_AS]->(p:Printing)-[:PRINTED_IN]->(s:Set)
WHERE s.releaseDate >= date("2000-01-01") AND p.normalPrice <= 10 AND c.manaCost
=~ '.*[WUBRG].*' AND NOT EXISTS ((c)-[:IS_OF_COLOR]->(:Color))
RETURN p.uuid AS uuid, p.normalPrice AS normalPrice
ORDER BY normalPrice;
```

9) For each set, return the names of the **previous** set by **release date** (break ties by **alphabetical** order)

```
MATCH (s:Set)
OPTIONAL MATCH (prev:Set)
WHERE prev.releaseDate < s.releaseDate OR (prev.releaseDate = s.releaseDate AND
prev.name < s.name)
WITH s.name AS currSet, date(max(prev.releaseDate)) AS prevReleaseDate
MATCH (s:Set {name: currSet})
OPTIONAL MATCH (prev:Set {releaseDate: prevReleaseDate})
WHERE prev.releaseDate < s.releaseDate OR (prev.releaseDate = s.releaseDate AND
prev.name < s.name)
WITH s.name AS currSet, max(prev.name) AS prevSet
RETURN currSet, prevSet
ORDER BY currSet;
```

10) For each set, return the names of the **previous** and the **next** set by **release date** (break ties by **alphabetical** order)

```
MATCH (s:Set)
OPTIONAL MATCH (prev:Set)
WHERE prev.releaseDate < s.releaseDate OR (prev.releaseDate = s.releaseDate AND
prev.name < s.name)
WITH s.name AS currSet, date(max(prev.releaseDate)) AS prevReleaseDate
MATCH (s:Set {name: currSet})
OPTIONAL MATCH (prev:Set {releaseDate: prevReleaseDate})
WHERE prev.releaseDate < s.releaseDate OR (prev.releaseDate = s.releaseDate AND
prev.name < s.name)
WITH s.name AS currSet1, max(prev.name) AS prevSet

MATCH (s:Set)
OPTIONAL MATCH (next:Set)
WHERE next.releaseDate > s.releaseDate OR (next.releaseDate = s.releaseDate AND
next.name > s.name)
WITH s.name AS currSet, date(min(next.releaseDate)) AS nextReleaseDate, currSet1,
prevSet
MATCH (s:Set {name: currSet})
OPTIONAL MATCH (next:Set {releaseDate: nextReleaseDate})
WHERE next.releaseDate > s.releaseDate OR (next.releaseDate = s.releaseDate AND
next.name > s.name)
WITH s.name AS currSet2, min(next.name) AS nextSet, currSet1, prevSet

WHERE currSet2 = currSet1
RETURN currSet1 AS name, prevSet, nextSet;
```