Base de datos no relacionales

We Love-Love Tennis

Felipe Almeida, 4.955.986-4
Santiago Guridi, 5.505.660-6

Instituto de Computación
Udelar
Uruguay
03/07/2023

Índice

1. Introducción

El tenis es uno de los deportes más populares y apasionantes a nivel mundial, capturando la atención de millones de fanáticos en cada competencia. Con una larga historia que se remonta a siglos atrás, este deporte ha evolucionado hasta convertirse en una disciplina altamente competitiva y emocionante.

El calendario tenístico está compuesto por una amplia variedad de torneos, desde los prestigiosos Grand Slam hasta los eventos de menor escala, ofreciendo a los jugadores la oportunidad de competir en diferentes superficies como césped, arcilla y pista dura. Cada torneo otorga puntos a los jugadores en función de su desempeño, y al final de la temporada, se determina un ranking mundial que refleja el éxito y la consistencia de los competidores.

Los nombres de los grandes jugadores de tenis han dejado una huella imborrable en la historia del deporte, como Roger Federer, Rafael Nadal, Novak Djokovic, Serena Williams y Naomi Osaka, entre muchos otros. Estos atletas han elevado el nivel de juego a nuevas alturas, rompiendo récords y cautivando a audiencias de todo el mundo con su destreza y pasión por el tenis.

En este trabajo, se utilizó una base de datos de grafos que contiene información detallada sobre tenistas, partidos y torneos de tenis. El objetivo principal fue explorar y evaluar la eficacia de la herramienta Neo4j para la extracción de datos en este contexto.

El artículo está estructurado de la siguiente manera. En la sección II se describe el proceso de recopilación de datos históricos de los principales torneos de tenis, así como el modelado y la carga de la base de datos.

En la sección III se detallan los diversos experimentos realizados sobre la base de datos. Estos experimentos abarcan desde consultas específicas hasta análisis de datos estadísticos haciendo comparaciones con otras estadísticas ya existentes. Se intenta revelar patrones o relaciones interesantes.

Finalmente, en la sección IV se presentan las conclusiones obtenidas a partir de los experimentos realizados, destacando los hallazgos más relevantes. Además, se sugieren posibles trabajos futuros que podrían realizarse para aprovechar al máximo la base de datos y continuar investigando en este campo de estudio.

2. Desarrollo

2.1. Obtención de datos

Se obtuvieron datos de la plataforma kaggle (*ATP/WTA Tennis Data*, s.f.) que contienen estadísticas de partidos y jugadores de tenis de la WTA (Women's Tennis Association) y la ATP (Association of Tennis Professionals).

Estos datos incluyen partidos entre 1949 y 2021, lo cual constituye un gran volumen de datos para cargar y analizar a posteriori.

Los datos obtenidos se presentan en formato csv, por lo que hubo que definir un proceso para poder cargarlos a una base de datos de grafos.

2.2. Purga del dataset

A la hora de llevar la carga de los datos del dataset a la base de datos en formato de grafo, se encontraron varios problemas listados a continuación:

1. El dataset contiene mas de 450.000 filas de datos, lo cual excede el limite de la instancia remota gratuita que provee Neo4J.
2. Algunos atributos que aparentan ser ids primarios (*player_id*) se repiten.
3. Datos como el id de torneos no representan el torneo sino el partido

Para reducir el numero de datos y de relaciones que es necesario crear en la base de datos de grafos, se limita los datos a aquellos posteriores a 1992 (Cabe aclarar que en una instancia local se pudieron cargar todos los datos y analizarlos cuando fue necesario). Para remediar los atributos que se consideraban primarios se porcede de la siguiente forma. En total se hallan 9975 ids repetidas dentro del data set de players. Se eliminan de aqui los registros que presentan estos errores. Se toma nota de los identificadores que tienen en común los torneos, y se procede a sustituir todos los que son parte del mismo torneo por el identificador común (En particular Davis Cup y Fed Cup)

2.3. Carga del dataset

Para llevar a cabo la carga del dataset en la instancia de la base de datos de grafos se procede de la siguiente manera:

1. Se carga un nodo *Country* por cada valor único que existe dentro de las nacionalidades de los jugadores.
2. Se carga un nodo *Tourney* por cada valor único que existe dentro de los nombres de los torneos de los partidos.
3. Se carga un nodo *Player* por cada fila del csv de jugadores. Se crea a su vez una relación **BORN_IN** hacia el país en el cual el jugador nació. Debido a limitaciones de memoria de la instancia de la base de datos de grafos, esto se debe llevar a cabo en transacciones. En este caso se hizo de a 1000 jugadores por transacción.
4. Se crea un índice sobre el atributo *player_id* de los jugadores. Esto es necesario, ya que de lo contrario la carga de los partidos y partidos lleva un tiempo excesivo.
5. Se carga un nodo *Match* por cada fila del csv de partidos. Se debe llevar a cabo una busqueda del jugador ganador y perdedor del partido mediante un MATCH, y se crea la relación **WON** y **LOST**. También se crea la relación **PART_OF** para hacer pertenecer el partido a un torneo en particular.

Se adjunta el archivo BDNR_Obligatorio_final_grupo_20 que contiene los comandos para la carga del dataset en cualquier base de datos de grafos de Neo4j. Los csv's utilizados como entrada son obtenidos del (*Tennis Dataset*, s.f.), el cual es un repositorio público en Github creado para este trabajo.

A partir de la carga de datos acorde a la descripción previa, y a los comandos adjuntos en el Jupyter Notebook publicado en el repositorio () se obtiene un modelo de la base de datos como se muestra en la Figura 1

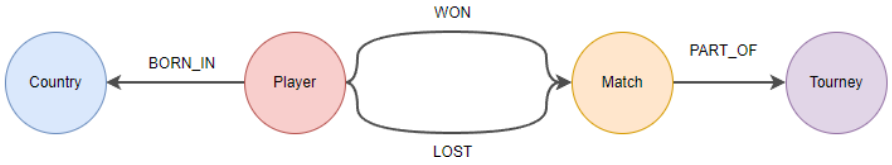

Figura 1: Modelo de la base de datos

2.4. Herramientas

La ventaja que se obtiene mediante el uso de una base de datos de grafos es más evidente cuando se evaluan las cadenas de victorias de jugadores. Es entonces que se buscan opciones dentro de Neo4j para poder recorrer estas relaciones de victorias lo mejor posible. En particular se cuenta con las siguientes operaciones:

1. shortestPath: Es una función que obtiene el camino más corto entre dos nodos dados. También se la puede configurar para que solamente utilice ciertas relaciones y no cualquier arista que una nodos.

2. APOC: Es una biblioteca oficialmente apoyada por Neo4j que cuenta con una gran variedad de funciones que no son simples de implementar en Cypher. La misma implementa las funciones en Java. (Neo4j, s.f.). Esta viene por defecto instalada en las instancias provistas por AuraDB, pero debe ser manualmente instalada en otro tipo de instancias

3. Experimentación

Se presenta a continuación multiples estadísticas que se consideraron interesantes obtener mediante consultas a la base de datos creada. Además se presenta un breve análisis sobre cada una de ellas (Se encuentran en el archivo BDNR_Obligatorio_final_grupo_20 las mismas, y se pueden ejecutar cada una de ellas)

3.1. Camino de menor longitud

Objetivo: Obtener el camino de menor longitud que une mediante partidos jugados/ganados Rod Laver y Roger Federer.

A la hora de buscar el camino de menor longitud entre dos jugadores se puede hacer uso de la función *shortestPath* provista por Neo4j, la cual determina el camino de menor longitud entre dos nodos utilizando solamente relaciones de un tipo especificado. En este caso se utiliza solamente las relaciones **WON** y **LOST** de los jugadores a los partidos jugados. Es entonces que la query resulta la siguiente:

```
MATCH (p:Player {player_id:'103819'}), // Federer
      (n:Player {player_id:'100029'})  // Rod Laver
RETURN shortestPath((p)-[:WON|LOST*]-(n))
```

En particular para obtener la distancia entre toda dupla de jugadores se puede llevar a cabo esta formulación para todos los nodos del producto cartesiano de los nodos de label *Player* con si mismos. La misma no computará para datasets de gran tamaño, pero si lo hace para esta versión reducida.

```
MATCH (p:Player),
      (n:Player)
RETURN p.player_id,
  n.player_id,
  length(shortestPath((p)-[:WON|LOST*]-(n)))
```

La idea detrás de buscar el camino mas corto entre Rod Laver y Roger Federer fue que el primero es considerado uno de los mejores tenistas de la década del 60 mientras que el último es uno de los mejores de la última década. De esta manera, es evidente que no existen partidos oficiales entre ellos pero sí es interesante obtener el camino más corto que los une.

Luego de ejecutar la consulta se obtuvo que el camino más corto es el grafo que tiene los partidos: Rod Laver VS Jimmy Connors, Jimmy Connors VS Byron Black y Byron Black vs Roger Federer.

Estos partidos se disputaron en Agosto de 1975, Abril de 1992, Junio de 1999 respectivamente, resaltando claramente la gran cantidad de años entre partido y partido.

El resultado de la consulta se visualiza mejor en la Figura 2

3.2. Duración promedio

Objetivo: Obtener el jugador cuyo promedio de duración de partidos ganados fue el mas rápido.

Para obtener esta estadística se ejecutó la siguiente consulta:

```
MATCH (p:Player)-[:WON]->(m:Match)
WHERE m.minutes IS NOT NULL
RETURN
  p.player_id AS id,
  p.name_first + ' ' + p.name_last as name,
  avg(toInteger(m.minutes)) AS minutes_avg, COUNT(m) AS matches
ORDER BY minutes_avg ASC
```

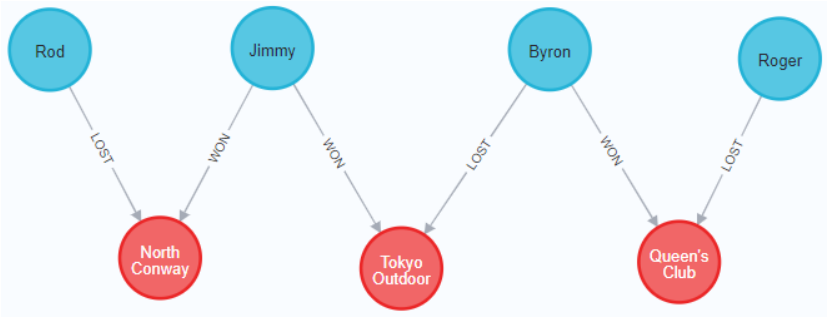

Figura 2: Camino entre Rod Laver y Roger Federer

De esta manera se obtuvo que Gianni Mina es el que tiene el promedio de tiempo de victoria mas bajo (26 minutos) desde 1992. Sin embargo, notamos que solo hay registros de 1 partido para este tenista, con lo cual se decidió modificar de la siguienta manera la consulta para tener en cuenta la cantidad de partidos disputados.

```
MATCH (p:Player)-[:WON]->(m:Match)
WHERE m.minutes IS NOT NULL
RETURN
  p.player_id as id,
  p.name_first + ' ' + p.name_last AS name,
  avg(toInteger(m.minutes)) AS minutes_avg,
  COUNT(m) AS matches,
  1/(avg(toInteger(m.minutes))*sqrt(COUNT(m))) as weighted_avg
ORDER BY weighted_avg ASC
```

Esta nueva conuslta pondera los partidos jugados por cada tenista, para ello se definió:

$$weighted\_avg = \frac{1}{\text{promedio\_minutos} \times \sqrt{\text{total\_partidos}}}$$

Finalmente se ordenan los resultados de manera ascendente por weighted_avg. Ahora obtenemos otro jugador como resultado de esta consulta, Rafael Nadal. Si bien el promedio en minutos de victoria es de 115, este tenista ha ganado 955 partidos, lo cual lo convierte en uno de los jugadores con tiempo de victoria más rapido en promedio y uno de los mejores de la historia.

3.3. Ganadores según superficie

Objetivo: Obtener el pais con tenistas mas ganadores en cada tipo de cancha

Para esto se debe basar la query en los nodos de Country, luego recorrer las relaciones **BORN_IN** hacia los nodos Player, y luego las partidas ganadas mediante la relación **WON** a los nodos Match. Luego alcanza con retornar el conteo por pais y superficie del Match.

```
MATCH (c:Country)<-[:BORN_IN]-(p:Player)-[:WON]->(m:Match)
RETURN c.country,m.surface,COUNT(m) AS Wins
```

Con esto se consigue la cantidad de victorias en cada superficie para cada país. Solo hallar el pais con más victorias para cada superficie. Esto se consigue usando la función COLLECT.

```
MATCH (c:Country)<-[:BORN_IN]-(p:Player)-[:WON]->(m:Match)
WITH m.surface AS surface, c.country AS country, COUNT(m) AS Wins
ORDER BY surface, Wins DESC
RETURN surface,
  COLLECT({country: country, wins: Wins})[0] AS maxWins
```

Este brinda como resultado que el país con mayor victorias en todas las superficies resulta ser Estados Unidos, a excepción de los partidos en canchas cerradas, donde aquí es Gran Bretaña quien tiene un mayur número de victorias con 9 victorias totales.

3.4. Ganador de torneo

Objetivo: Obtener el ganador de cada torneo para cada fecha en la que se jugó.

Los datos utilizados tienen una representación un tanto obtusa de los torneos, pero los mismos pueden ser identificados mediante su nombre y su fecha. Es asi que se busca encontrar el ganador de un torneo en una fecha. Para esto se encuentran todos los jugadores que hayan participado en partidas de dicho torneo en dicha fecha y luego se cuentan la cantidad de victorias de cada uno:

```
MATCH (t:Tourney {name:'Lyon'})<-[:PART_OF]-
  (m:Match{tourney_date: date('2018-05-21')})
  <-[wl:WON|LOST]-(p:Player)
RETURN p, COUNT(CASE WHEN TYPE(wl) = 'WON' THEN 1 ELSE NULL END)
  AS won_count
ORDER BY won_count DESC
LIMIT 1
```

Para poder conseguir el ganador de cada uno se eliminan las clausulas de *toruney_date* y *name*, y se pasa a usar un collect sobre players y su cantidad de victorias, tomando solamente el elemento 0 de esa collección:

```
MATCH (t:Tourney)<-[:PART_OF]-(m:Match)<-[wl:WON|LOST]-(p:Player)
WITH t.name AS tourney_name, m.tourney_date AS tourney_date, p,
COUNT(CASE WHEN TYPE(wl) = 'WON' THEN 1 ELSE NULL END)
  AS won_count
ORDER BY won_count DESC
WITH tourney_name, tourney_date,
  COLLECT({player: p, won_count: won_count}) AS players
RETURN tourney_name, tourney_date,
```

```
  players[0].player AS player,
  players[0].won_count AS max_wins
```

Ejecutando esta query se encuentran los ganadores de los torneos para multiples fechas como por ejemplo Andy Murray para el torneo de Wimbledon en el año 2013

3.5. Victorias transitivas

Objetivo: Dado un jugador A y un año, obtener todos los jugadores a los que le ha ganado, y que le ha "ganado"transitivamente. Esto es, si A derrotó a B y a su vez B derrotó a C anteriormente en el año, B y C son incluidos en el resultado final.

Definimos esta relación como ganar transitivamente.

```
MATCH (pw:Player)-[w1:WON]->(m1:Match)<-[l1:LOST]-(pl:Player)
  -[w2:WON]->(m2:Match)<-[l2:LOST]-(pll:Player)
WHERE date({year:$year}) <= m1.tourney_date < date({year:$year+1})
AND date({year:$year}) <= m2.tourney_date < date({year:$year+1})
AND m2.tourney_date <= m1.tourney_date
AND pw.player_id <> pll.player_id
WITH pw, COLLECT(DISTINCT pl) + COLLECT(DISTINCT pll) AS
  distinctLostPlayers
RETURN pw.name_first + " " + pw.name_last as winner_name,
  SIZE(apoc.coll.toSet(distinctLostPlayers)) AS
  distinctCount
ORDER BY distinctCount DESC
```

Podemos de esta manera obtener varias estadísticas y comparaciones. Por ejemplo, podriamos definir al mejor tenista del año como aquel que le ganó transitivamente a mas tenistas. Tomando como ejemplo el año 2019 (último año pre-pandemia) se obtuvo que el tenista griego, Stefanos Tsitsipas fue el que tuvo mas victorias transitivas (194).

Otro dato interesante es comparar si los tenistas que le ganaron a mas tenistas directamente en ese año son los mismos a los obtenidos con esta consulta. Además podemos analizar el puesto en el ranking ATP que tenían estos tenistas en ese año.

La siguiente tabla muestra los datos de los 10 tenistas con mas victorias transitivas y los otros datos recién mencionados

Nombre	Victorias transitivas	Victorias directas	Posición ranking 2019
Stefanos Tsitsipas	194	43	6
Dominic Thiem	190	39	4
Roger Federer	186	42	3
Novak Djokovic	182	43	2
Andrey Rublev	181	33	23
Denis Shapovalov	180	36	15
Daniil Medvedev	180	49	5
Rafael Nadal	180	47	1
Matteo Berrettini	178	34	8
Alexander Zverev	173	35	7

Se observa que, a priori, esta nueva medida de victorias transitivas tiene sentido y podría ser utilizada para definiar un nuevo ranking entre jugadores.

3.6. Mejor racha de victorias

Objetivo: Dado un año o un rango de años, obtener los tenistas que tuvieron la mayor cantidad de partidos sin perder. Es decir, la racha de victorias mas larga.

La siguiente consulta obtiene lo buscado. Primero se obtienen todos los partidos que se jugaron en el rango de tiempo definido y se ordenan por tourney_date para cada jugador. Luego el primer REDUCE obtiene todas las rachas de victorias consecutivas obtenidas por los tenistas agregandon un nuevo array vacío cada vez que encuentra un LOST. Finalmente el segundo REDUCE obtiene el array de maximo tamaño, el cual indica la mayor racha de victorias consecutivas de ese tenista.

```
MATCH (pw:Player)-[r:WON|LOST]->(m1:Match)<-[:WON|LOST]->
  (pl:Player)
WHERE date({year:$year_start}) <= m1.tourney_date <
  date({year:$year_end})
WITH pw, m1, type(r) as results
ORDER BY m1.tourney_date ASC

// Obtener todas las rachas de victorias
WITH pw, REDUCE(s = [], result IN COLLECT(results) |
  CASE
    WHEN result = 'LOST' THEN s + [[]]
    ELSE s[0..size(s)-1] + [s[size(s)-1] + [result]]
  END
) AS all_winning_streaks

// Obtener todas la mejor racha de victorias
WITH pw, [arr IN all_winning_streaks WHERE size(arr) =
  REDUCE(maxSize = 0, a IN all_winning_streaks |
    CASE
      WHEN size(a) > maxSize THEN size(a)
```

```
      ELSE maxSize
    END
)][0] AS maximum_winning_streak

WHERE maximum_winning_streak IS NOT NULL
WITH pw, maximum_winning_streak, SIZE(maximum_winning_streak)
  AS streak_number
ORDER BY streak_number DESC
RETURN pw.name_first + ' ' + pw.name_last as player_name,
  streak_number
```

Tomando como ejemplo nuevamente el 2019, se puede observar que los tenistas con racha mas larga fueron los mostrados en esta tabla:

Nombre	**Racha de victorias**	**Posición ranking 2019**
Rafael Nadal	20	1
Daniil Medvedev	16	5
Roger Federer	11	3
Novak Djokovic	9	2
Christian Garin	9	33
David Goffin	8	11
Albert Ramos	8	41
Benoit Paire	8	24
Sam Querrey	8	44
Alexander Zverev	8	7

Se puede observar claramente que no necesariamente los mejores tenistas del ranking son los que tienen rachas de victorias mas grandes, ya que esto depende mucho de la calidad y dificultad de los partidos disputados por cada uno de ellos.

4. Conclusiones

En este proyecto, se desarrolló una base de datos de grafos utilizando Neo4j para almacenar datos históricos del mundo del tenis. El diseño implementado y la posterior carga de datos resultaron ser apropiados para la realización de las consultas buscadas.

Además, mediante la ejecución de distintas consultas a la base de datos, se evidenció la capacidad de la herramienta y de las bases de datos de grafos para obtener información relevante. En particular se destaca la capacidad de hallar caminos de longitud no determinada de antemano (Como los caminos de WON y LOST), los cuales, en una base relacional modelada tal cual los datos de carga, requeriria una cantidad indefinida de JOINS. Esta capacidad viene con un costo al uso de memoria, lo cual era notorio a la hora de llevar a cabo queries que dependian en el producto cartesiano de grandes colecciones de datos.

No obstante, cabe destacar que aún existe un potencial sin explotar en los datos recopilados, ya que no se utilizaron todos sus elementos. Aspectos como los puntos ganados, los sets disputados o incluso la cantidad de Aces realizados por partido podrían brindar la posibilidad de realizar consultas más específicas de gran interés.

En resumen, este proyecto ha demostrado la importancia y utilidad de las bases de datos de grafos en el análisis de datos en el ámbito del tenis, proporcionando información valiosa y abriendo nuevas perspectivas para comprender este deporte.

Referencias

ATP/WTA Tennis Data. (s.f.). https://www.kaggle.com/datasets/taylorbrownlow/atpwta-tennis-data. (Accessed: June 30, 2023)

Neo4j. (s.f.). *Neo4j APOC Library Documentation.* https://neo4j.com/labs/apoc/5/. (Accessed: June 30, 2023)

Tennis Dataset. (s.f.). https://github.com/flacoski/tennis_data_BDNR. (Accessed: July 2, 2023)