

Trabajo de prácticas BDGE

LUIS ARDÉVOL MESA, MIGUEL MATO MARTÍNEZ

Usando una conjunto de datos de partidos de tenis, se usarán distintos tipos de base de datos para realizar consultas acerca de los mismos. En concreto, se tratarán bases de datos relaciones y con datos agregados en PostgreSQL, bases de datos distribuidas con SQL usando CITUS Data, y dos tipos de bases de datos noSQL: documentales con MongoDB y en grafo con Neo4j.

1. BASES DE DATOS RELACIONALES (POSTGRESQL)

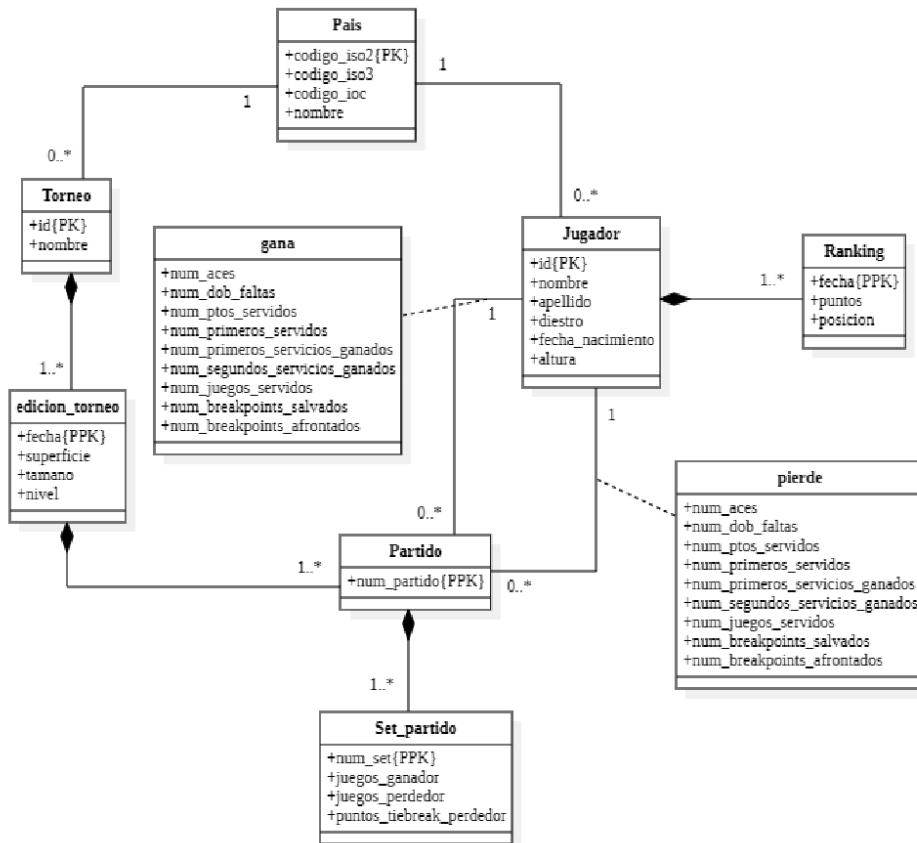


Fig. 1. Esquema de la base de datos de partidos de tenis.

En la figura 1 se muestra el esquema de la base de datos de partidos de tenis. Comenzaremos explicando este esquema de forma breve, pero haciendo especial hincapié en las relaciones entre las distintas tablas, ya que esto ayudará a comprender mejor las condiciones de *join* a usar en las consultas que usen este esquema. Destacamos que no se hará uso de la tabla ranking, por lo que no se comentará en el siguiente análisis.

La tabla pais contiene la información relativa a países, y se usará tanto para mostrar el país dónde se celebra un torneo como para referenciar el país de procedencia de un jugador; su clave primaria es `codigo_iso2`. La tabla jugador contiene la información de los jugadores, y su clave primaria es el `id` de cada jugador; el atributo `pais` referencia a `pais(codigo_iso2)`. La tabla torneo contiene la información de los torneos; su clave primaria es el `id` de cada torneo y el atributo `pais` referencia a `pais(codigo_iso2)`. La tabla `edicion_torneo` contiene la información de las ediciones de los torneos; su clave primaria es una combinación de los atributos `torneo` y `fecha`, donde el atributo `torneo` es una referencia a `torneo(id)`. La tabla `partido` contiene la información de los partidos disputados; su clave primaria es la combinación de los atributos `torneo`, `fecha` y `num_partido`, y tiene una clave foránea compuesta por los atributos `torneo` y `fecha`, que referencia a la clave primaria de `edicion_torneo`, `edicion_torneo(torneo, fecha)`. Además, los atributos `ganador` y `perdedor` son los `id` de los jugadores, por lo que puede enlazarse esta tabla con la de `jugador` mediante `jugador(id)`. Por último, la tabla `sets_partido` contiene la información de los sets de los partidos; su clave primaria es la combinación de los atributos `torneo`, `fecha`, `num_partido` y `num_set`, y tiene una clave foránea compuesta por los atributos `torneo`, `fecha` y `num_partido`, que referencia a la clave primaria de `partido`, `partido(torneo, fecha, num_partido)`.

Tras ver cómo se estructura la base de datos a usar, creamos la base de datos en PostgreSQL y cargamos los datos en ella. La estructura relacional se proporciona en el archivo `schema.sql`, y los datos en los archivos `pais.csv`, `jugador.csv`, `torneo.csv`, `edicion_torneo.csv`, `partido.csv`, `sets_partido.csv` y `ranking.csv`. Una vez creada la base de datos `tenis`, ejecutamos el archivo `schema.sql` con la definición de los esquemas de las tablas mediante el siguiente comando en la terminal:

```
psql -U alumnogreibd -d tenis -f
      /home/alumnogreibd/BDGE/datos/datos_tenis/schema.sql
```

Ahora, con los esquemas definidos (pero vacíos), cargamos los datos a partir de los archivos `.csv` con los siguientes comandos en terminal (es importante seguir el orden de carga de los datos debido a las dependencias entre las tablas):

```
psql -U alumnogreibd -d tenis -c "\copy pais from
      /home/alumnogreibd/BDGE/datos/datos_tenis/pais.csv csv"
psql -U alumnogreibd -d tenis -c "\copy jugador from
      /home/alumnogreibd/BDGE/datos/datos_tenis/jugador.csv csv"
psql -U alumnogreibd -d tenis -c "\copy torneo from
      /home/alumnogreibd/BDGE/datos/datos_tenis/torneo.csv csv"
psql -U alumnogreibd -d tenis -c "\copy edicion_torneo from
      /home/alumnogreibd/BDGE/datos/datos_tenis/edicion_torneo.csv csv"
psql -U alumnogreibd -d tenis -c "\copy partido from
      /home/alumnogreibd/BDGE/datos/datos_tenis/partido.csv csv"
psql -U alumnogreibd -d tenis -c "\copy sets_partido from
      /home/alumnogreibd/BDGE/datos/datos_tenis/sets_partido.csv csv"
psql -U alumnogreibd -d tenis -c "\copy ranking from
      /home/alumnogreibd/BDGE/datos/datos_tenis/ranking.csv csv"
```

Este orden es lógico por las dependencias entre tablas que comentamos anteriormente: `sets_partido` referencia a `partido`, por lo que es necesario cargar primero los datos de `partido` para poder cargar los de `sets_partido`. Este razonamiento aplica al resto de tablas: `partido` referencia a `edicion_torneo`, que referencia a `torneo`, que a su vez referencia a `pais`, que es referenciado por `jugador`.

En las siguientes consultas, los `join` se harán mediante *theta join* (*inner* por defecto). En caso de que se quiera

hacer otro tipo de *join*, se especificará de forma clara en la consulta. Esto considera el producto cartesiano de las tablas a unir y selecciona solo las tuplas que verifiquen el predicado (la condición o condiciones de *join*). Estos *join* se harán de forma general de forma implícita en la cláusula *where*, siguiendo los atributos referenciados entre las tablas. El *where* hace una selección de tuplas (filas) que cumplen el predicado o los predicados mencionados. Teniendo esto muy presente, abusaremos un poco del lenguaje para ser más concisos y frases como: “seleccionamos solo las tuplas en las que un jugador concreto es el ganador del partido”, reformularímos como “seleccionamos los partidos que ganó un jugador concreto”.

.1. Muestra todos los ganadores del torneo “Wimbledon” (Nombre, apellidos y año). Ordena el resultado por año.

Para esta consulta necesitamos la información de los jugadores, los partidos y los torneos, por lo que comenzamos en el *from* con el producto cartesiano de las tablas *jugador*, *partido* y *torneo* (especificando un alias para cada una de ellas). A continuación, especificamos las condiciones de *join* en el *where*, así como otras condiciones de selección:

- Condiciones de *join*:

- El predicado *j.id = p.ganador* sirve para unir la tabla *jugador* con la tabla *partido* mediante el atributo *id* de la tabla *jugador* y el atributo *ganador* de la tabla *partido*, haciendo una selección únicamente de las tuplas que cumplan esta condición. Esto nos permite, para cada jugador, seleccionar solo los partidos en los que ha ganado.
- El predicado *t.id = p.torneo* sirve para unir la tabla *torneo* con la tabla *partido* mediante el atributo *id* de la tabla *torneo* y el atributo *torneo* de la tabla *partido*, haciendo una selección únicamente de las tuplas que cumplan esta condición. Esto nos permite seleccionar solo los partidos que se han jugado en un torneo concreto.

- Condiciones de selección:

- La condición *t.nombre = ‘Wimbledon’* selecciona solo el torneo cuyo nombre es “Wimbledon”. Al hacer el *join* con la tabla *torneo*, estamos seleccionando solo los partidos que se han jugado en el torneo “Wimbledon”.
- La condición *p.ronda = ‘F’* selecciona solo los partidos que han sido finales.

Tras seleccionar las tuplas que nos interesan, usamos el *select* para obtener la proyección de las filas que nos interesan: en este caso, nos quedamos con los atributos *nombre* y *apellido* de la tabla *jugador* y el año de la fecha del partido, que denotaremos por el alias “*ano*”. Como *p.fecha* es del tipo *date*, este último atributo lo obtenemos mediante la función *extract()*, especificando que solo queremos el año de ese atributo (*year from p.fecha*). Tras esto, ordenamos el resultado por año con *order by ano*; el orden por defecto es ascendente. El código de la consulta se muestra a continuación, y el resultado se puede ver en la figura 2.

```
select j.nombre, j.apellido, extract(year from p.fecha) as ano
from jugador j, partido p, torneo t
where j.id = p.ganador
  and t.id = p.torneo
  and t.nombre = 'Wimbledon'
  and p.ronda = 'F'
order by ano
```

	A-Z nombre	A-Z apellido	123 año
1	Michael	Stich	1.991
2	Andre	Agassi	1.992
3	Pete	Sampras	1.993
4	Pete	Sampras	1.994
5	Pete	Sampras	1.995
6	Richard	Krajicek	1.996
7	Pete	Sampras	1.997
8	Pete	Sampras	1.998
9	Pete	Sampras	1.999
10	Pete	Sampras	2.000
11	Goran	Ivanisevic	2.001
12	Lleyton	Hewitt	2.002
13	Roger	Federer	2.003
14	Roger	Federer	2.004
15	Roger	Federer	2.005
16	Roger	Federer	2.006
17	Roger	Federer	2.007
18	Rafael	Nadal	2.008
19	Roger	Federer	2.009
20	Rafael	Nadal	2.010
21	Novak	Djokovic	2.011
22	Roger	Federer	2.012
23	Andy	Murray	2.013
24	Novak	Djokovic	2.014
25	Novak	Djokovic	2.015
26	Andy	Murray	2.016
27	Roger	Federer	2.017
28	Novak	Djokovic	2.018
29	Novak	Djokovic	2.019
30	Novak	Djokovic	2.021
31	Novak	Djokovic	2.022
32	Carlos	Alcaraz	2.023

Fig. 2. Modelo relacional Postgresql, consulta 1.

.2. Muestra los años en los que Roger Federer ganó algún torneo de nivel Gran Slam (G) o Master 1000 (M). Para cada año, muestra el número de torneos y lista sus nombres (ordenados por la fecha de celebración). Ordena el resultado por el año

Para esta consulta necesitamos información de los jugadores, los partidos, los torneos y las ediciones de los torneos. Comenzamos en el `from` con el producto cartesiano de las tablas `jugador`, `partido`, `torneo` y `edicion_torneo` (especificando un alias para cada una de ellas). A continuación, especificamos las condiciones de `join` en el `where`, así como otras condiciones de selección:

- Condiciones de `join` que, en global, nos permitirán seleccionar las tuplas de un jugador específico que ganó un torneo específico en un año específico:
 - El predicado `j.id = p.ganador` sirve para unir la tabla `jugador` con la tabla `partido` mediante el atributo `id` de la tabla `jugador` y el atributo `ganador` de la tabla `partido`, haciendo una selección únicamente de las tuplas que cumplan esta condición. Esto nos permite, para cada jugador, seleccionar solo los partidos en los que ha ganado.
 - El predicado `t.id = p.torneo` sirve para unir la tabla `torneo` con la tabla `partido` mediante el atributo `id` de la tabla `torneo` y el atributo `torneo` de la tabla `partido`, haciendo una selección únicamente de las tuplas que cumplan esta condición. Esto nos permite seleccionar solo los partidos que se han jugado en un torneo concreto.
 - El predicado `t.id = et.torneo` sirve para unir la tabla `torneo` con la tabla `edicion_torneo` mediante el atributo `id` de la tabla `torneo` y el atributo `torneo` de la tabla `edicion_torneo`, haciendo una selección únicamente de las tuplas que cumplan esta condición. Esto nos permite seleccionar ediciones de torneos concretos.
 - El predicado `p.fecha = et.fecha` sirve para unir la tabla `partido` con la tabla `edicion_torneo` mediante el atributo `fecha` de la tabla `partido` y el atributo `fecha` de la tabla `edicion_torneo`, haciendo una selección únicamente de las tuplas que cumplan esta condición. Esto nos permite

seleccionar solo los partidos que se han jugado en una edición concreta de un torneo (concreto, por la condición anterior).

- Estamos interesados específicamente en torneos de nivel Gran Slam (G) o Master 1000 (M) y en las finales ganadas por Roger Federer (lo que indica que ganó el torneo) durante varios años (es decir, varias ediciones de torneos). Por tanto, las condiciones de selección son:

- El predicado `p.ronda = 'F'` selecciona solo los partidos que han sido finales.
- El predicado `j.nombre = 'Roger' and j.apellido = 'Federer'` selecciona solo los partidos en los que ha ganado "Roger Federer".
- El predicado `et.nivel in ('G', 'M')` selecciona solo los torneos de nivel Gran Slam (G) o Master 1000 (M).

Con el `group by` agrupamos los resultados por el año en que se celebraron los torneos, para su posterior concatenación. Por último, obtenemos la proyección de los atributos que nos interesan: el año, que lo obtenemos extrayendo el año de la fecha de celebración del partido; el número de torneos ganados, que lo obtenemos contando el número de torneos distintos (usamos el `id` del torneo para diferenciar) en el resultado agrupado por año (`count(distinct t.id)`). Como última proyección, obtenemos una concatenación (con `string_agg()`) de esos torneos, separados por comas y ordenados por fecha de celebración de esa edición, en orden ascendente (`order by et.fecha`). El código de la consulta se muestra a continuación, y el resultado se pueden ver en la figura 3.

```
select extract(year from et.fecha) as ano, count(distinct t.id) as numero_torneos,
       string_agg(t.nombre, ', ' order by et.fecha) as torneos
from jugador j, partido p, torneo t, edicion_torneo et
where j.id = p.ganador
  and t.id = et.torneo
  and p.torneo = t.id
  and p.fecha = et.fecha
  and p.ronda = 'F'
  and j.nombre = 'Roger'
  and j.apellido = 'Federer'
  and et.nivel in ('G', 'M')
group by ano
order by ano
```

	ano	numero_torneos	torneos
1	2.002	1	Hamburg Masters
2	2.003	1	Wimbledon
3	2.004	6	Australian Open, Indian Wells Masters, Hamburg Masters, Wimbledon, Canada Masters, US Open
4	2.005	6	Miami Masters, Indian Wells Masters, Hamburg Masters, Wimbledon, US Open, Cincinnati Masters
5	2.006	7	Australian Open, Miami Masters, Indian Wells Masters, Wimbledon, US Open, Canada Masters, Madrid Masters
6	2.007	5	Australian Open, Hamburg Masters, Wimbledon, US Open, Cincinnati Masters
7	2.008	1	US Open
8	2.009	4	Roland Garros, Madrid Masters, Wimbledon, Cincinnati Masters
9	2.010	2	Australian Open, Cincinnati Masters
10	2.011	1	Paris Masters
11	2.012	4	Indian Wells Masters, Madrid Masters, Wimbledon, Cincinnati Masters
12	2.014	2	Cincinnati Masters, Shanghai Masters
13	2.015	1	Cincinnati Masters
14	2.017	5	Australian Open, Indian Wells Masters, Miami Masters, Wimbledon, Shanghai Masters
15	2.018	1	Australian Open
16	2.019	1	Miami Masters

Fig. 3. Modelo relacional Postgresql, consulta 2.

.3. Muestra los partidos de semifinales (ronda='SF') y final (ronda = 'F') del torneo de "Roland Garros" del 2018. Para cada partido muestra la ronda, el tipo de desenlace, el nombre y apellidos del ganador y el nombre y apellidos del perdedor y el resultado con el número de juegos del ganador y del perdedor en cada set, y opcionalmente en paréntesis el número de juegos del perdedor en el tie break

Para esta consulta necesitamos información de los partidos, los jugadores, los torneos y los sets de los partidos; como debemos distinguir entre el ganador y el perdedor, sin ningún tipo de ambigüedad, en el `from` usamos dos instancias de la tabla `jugador`, cada una con un alias distinto, por lo que estaremos haciendo el producto cartesiano de la tabla `jugador` consigo misma, así como con las tablas `partido`, `torneo` y `sets_partido` (especificando un alias para cada una de ellas). A continuación, especificamos las condiciones de `join` en el `where`, así como otras condiciones de selección:

- Condiciones de `join` que, en global, nos permitirán seleccionar las tuplas de partidos específicos de un torneo específico en una fecha específica:
 - El predicado `jg.id = p.ganador` sirve para unir la tabla `jugador` con la tabla `partido` mediante el atributo `id` de la tabla `jugador` y el atributo `ganador` de la tabla `partido`, haciendo una selección únicamente de las tuplas que cumplan esta condición. Esto nos permite, para cada jugador, seleccionar solo los partidos en los que ha ganado.
 - Del mismo modo, usamos el predicado `jp.id = p.perdedor` para eleccionar solo los partidos en los que ha perdido un determinado jugador.
 - El predicado `p.fecha = sp.fecha` sirve para unir la tabla `partido` con la tabla `sets_partido` mediante el atributo `fecha` de la tabla `partido` y el atributo `fecha` de la tabla `sets_partido`, haciendo una selección únicamente de las tuplas que cumplan esta condición. Esto nos permite seleccionar solo los sets de los partidos que se han jugado en una fecha concreta.
 - Como la fecha anterior no es única para cada partido, necesitamos una condición adicional para obtener los sets de un partido determinado. Para ello, usamos el predicado `p.num_partido = sp.num_partido`, que une la tabla `partido` con la tabla `sets_partido` mediante el atributo `num_partido` de la tabla `partido` y el atributo `num_partido` de la tabla `sets_partido`, haciendo una selección únicamente de las tuplas que cumplan esta condición.
 - El predicado `t.id = p.torneo` sirve para unir la tabla `torneo` con la tabla `partido` mediante el atributo `id` de la tabla `torneo` y el atributo `torneo` de la tabla `partido`, haciendo una selección únicamente de las tuplas que cumplan esta condición. Esto nos permite seleccionar solo los partidos que se han jugado en un torneo concreto.
- Concretamente, estamos interesados en las semifinales (`ronda='SF'`) y la final (`ronda='F'`) del torneo de "Roland Garros" del 2018. Por tanto, las condiciones de selección son:
 - El predicado `p.ronda in ('SF', 'F')` selecciona solo los partidos que han sido semifinales o finales.
 - El predicado `t.nombre = 'Roland Garros'` selecciona solo los partidos que se han jugado en el torneo "Roland Garros".
 - El predicado `extract(year from p.fecha) = '2018'` selecciona solo los partidos que se han jugado en el año 2018.

Ahora, para cada combinación de `p.ronda`, `p.desenlace`, `jg.nombre`, `jg.apellido`, `jp.nombre` y `jp.apellido`, tendremos una fila para cada set, donde solo variará la información de los juegos ganados y perdidos. Por tanto, agrupamos por los atributos mencionados anteriormente y, en la proyección, concatenamos los resultados

de los juegos ganados y perdidos en cada set, así como el resultado final del partido, para lo que usamos la función `string_agg()`. Como también debemos especificar la puntuación del *tiebreak* (en caso de haberla), usamos un `case` para comprobar si el atributo `puntos_tiebreak_perdedor` es null o no: en caso que sea nulo, concatenamos una cadena vacía, en caso de no ser nulo, concatenamos la puntuación del *tiebreak*, puesta entre paréntesis. Si bien esta es una parte de la proyección, el resto incluye simplemente la ronda, el desenlace, la fecha y los nombres y apellidos de los jugadores concatenados (un atributo para el ganador y otro para el perdedor). El código de la consulta se muestra a continuación, y el resultado se pueden ver en la figura 4.

```
select p.ronda, p.desenlace, jg.nombre || ' ' || jg.apellido AS ganador,
       jp.nombre || ' ' || jp.apellido AS perdedor,
       STRING_AGG(sp.juegos_ganador || '-' || sp.juegos_perdedor || 
                  case
                     when sp.puntos_tiebreak_perdedor is not null then
                         '(' || sp.puntos_tiebreak_perdedor || ')'
                     else
                         ''
                  end, ', ' order by sp.num_set) as resultado
from partido p, jugador jg, jugador jp, sets_partido sp, torneo t
where jg.id = p.ganador
  and jp.id = p.perdedor
  and p.fecha = sp.fecha
  and p.num_partido = sp.num_partido
  and t.id = p.torneo
  and p.ronda in ('SF', 'F')
  and t.nombre = 'Roland Garros'
  and extract(year from p.fecha) = '2018'
group by p.ronda, p.desenlace, jg.nombre, jg.apellido, jp.nombre, jp.apellido
```

	A-Z ronda	A-Z desenlace	A-Z ganador	A-Z perdedor	A-Z resultado
1	F	N	Rafael Nadal	Dominic Thiem	6-4, 6-3, 6-2
2	SF	N	Dominic Thiem	Marco Cecchinato	7-5, 7-6(10), 6-1
3	SF	N	Rafael Nadal	Juan Martin del Potro	6-4, 6-1, 6-2

Fig. 4. Modelo relacional Postgresql, consulta 3.

4. Muestra la lista de jugadores españoles (ES) que ganaron algún torneo de nivel Gran Slam (G). Para cada jugador muestra los siguientes datos resumen de todos sus partidos: número de partidos jugados, porcentaje de victorias, porcentaje de aces, porcentaje de dobles faltas, porcentaje de servicios ganados, porcentaje de restos ganados, porcentaje de break points salvados (de los sufridos en contra), porcentaje de break points ganados (de los provocados a favor)

Esta consulta la dividiremos en dos partes: primero, buscaremos los jugadores españoles que ganaron algún torneo de nivel Grand Slam (G), y, a continuación, calcularemos los datos resumen que se piden de todos sus partidos. Para la primera parte, necesitamos información de los jugadores, los partidos y las ediciones de los torneos; comenzamos en el `from` con el producto cartesiano de las tablas `jugador`, `partido` y `edicion_torneo` (especificando un alias para cada una de ellas). A continuación, especificamos las condiciones de `join` en el `where`, así como otras condiciones de selección:

- Condiciones de `join` que, en global, nos permitirán seleccionar las tuplas de un torneo específico:
 - El predicado `j.id = p.ganador` sirve para unir la tabla `jugador` con la tabla `partido` mediante el atributo `id` de la tabla `jugador` y el atributo `ganador` de la tabla `partido`, haciendo una selección únicamente de las tuplas que cumplan esta condición. Esto nos permite, para cada jugador, seleccionar solo los partidos en los que ha ganado.

- El predicado `p.torneo = et.torneo` sirve para unir la tabla `partido` con la tabla `edicion_torneo` mediante el atributo `torneo` de la tabla `partido` y el atributo `torneo` de la tabla `edicion_torneo`, haciendo una selección únicamente de las tuplas que cumplan esta condición. Esto nos permite seleccionar solo los partidos que se han jugado en una edición concreta de un torneo.
- El predicado `p.fecha = et.fecha` sirve para unir la tabla `partido` con la tabla `edicion_torneo` mediante el atributo `fecha` de la tabla `partido` y el atributo `fecha` de la tabla `edicion_torneo`, haciendo una selección únicamente de las tuplas que cumplan esta condición. Esto nos permite seleccionar solo los partidos que se han jugado en una fecha concreta.
- Como de todo lo anterior solo nos interesan los partidos finales (F) de torneos de nivel Grand Slam (G) y ganados por jugadores españoles (ES), las condiciones de selección son:
 - El predicado `p.ronda = 'F'` selecciona solo los partidos que han sido finales.
 - El predicado `et.nivel = 'G'` selecciona solo los torneos de nivel Grand Slam (G).
 - El predicado `j.pais = 'ES'` selecciona solo los jugadores españoles.

Con todo esto, obtenemos una tabla con los jugadores españoles que han ganado algún torneo de nivel Grand Slam (G), pero hay repetición ya que estamos considerando cada edición de cada Grand Slam. Para quedarnos solo con el subconjunto que nos interesa (sin repetición), usamos un `distinct` en la proyección. Concretamente, obtenemos la proyección del `id` y el nombre completo del jugador, que concatenamos con el operador `||`, y aplicamos el `distinct` sobre la proyección. Esta tabla resultado la guardamos como `jugadores_espanoles_ganadores`, y la usaremos en la segunda parte de la consulta.

Para obtener las estadísticas resumen de cada jugador español ganador de un Grand Slam, necesitamos solo información sobre quiénes son esos jugadores y sobre los partidos que han jugado. Por tanto, en el `from` usamos la tabla `jugadores_espanoles_ganadores` que acabamos de crear, y la tabla `partido` (especificando un alias para cada una de ellas). A continuación, especificamos las condiciones de `join` en el `where`, así como otras condiciones de selección:

- Condiciones de `join` que, en global, nos permitirán seleccionar las tuplas de partidos de un jugador específico:
 - El predicado `jeg.id_jugador = p.ganador or jeg.id_jugador = p.perdedor` nos permite seleccionar los partidos, tanto ganados como perdidos, en los que ha participado un jugador específico de la tabla `jugadores_espanoles_ganadores`.

Con esto, obtenemos una fila por cada partido de cada jugador, por lo que agrupamos por el jugador con `group by` antes de agregar valores. En la proyección, obtenemos varios atributos, que redondearemos a un decimal usando `round(..., 1)`. En general, excepto para los partidos y el porcentaje de victorias, todas las estadísticas se calculan de la misma forma considerando atributos distintos; por ello, daremos una explicación detallada de una de ellas, y el resto simplemente mencionaremos qué atributos hemos usado y si se siguió un proceso distinto.

- El nombre completo del jugador, que concatenamos previamente con el operador `||`.
- El número de partidos jugados como el conteo de los partidos en los que ha participado (`count(p.num_partido)`).
- El porcentaje de victorias, que calculamos como el número de partidos ganados entre el número total de partidos jugados, multiplicado por 100. Para obtener el número de partidos ganados, usamos un `case` que comprueba si el jugador es el ganador del partido o no. Esto genera una lista que contiene 1 si el jugador es el ganador y 0 si no lo es, y con `sum()` sumamos los valores de esta lista para obtener el número de partidos ganados.

- El porcentaje de aces, que calculamos como el número de aces del jugador entre el número total de puntos servidos, multiplicado por 100. Para obtener el número de aces, simplemente sumamos (`sum()`) los valores la columna `num_aces_ganador` o `num_aces_perdedor` para el jugador específico. El número de puntos servidos lo obtenemos de forma análoga usando la columna `num_ptos_servidos_ganador` o `num_ptos_servidos_perdedor` según corresponda. Para ver si corresponde usar las estadísticas del ganador o del perdedor, se considerará un `case` que distinga si nuestro jugador está en la posición de ganador o perdedor, y se usarán las estadísticas correspondientes. Además, se implementará la función `nullif()` para gestionar las divisiones por 0.
- El porcentaje de dobles faltas, que calculamos de forma análoga al porcentaje de aces, pero usando las columnas `num_dob_faltas_ganador` y `num_dob_faltas_perdedor` en lugar de las columnas de aces.
- El porcentaje de servicios ganados, que calculamos de forma análoga al porcentaje de aces, pero usando una suma de las columnas `num_primeros_servicios_ganados_ganador` y `num_segundos_servicios_ganados_ganador`, o de `num_primeros_servicios_ganados_perdedor` y `num_segundos_servicios_ganados_perdedor` en lugar de las columnas de aces.
- El porcentaje de restos ganados, que calculamos de forma análoga al porcentaje de aces. Sin embargo, está vez usaremos las estadísticas de rival para obtener la del jugador que nos interesa: si nuestro jugador ganó, nos fijaremos en `num_ptos_servidos_perdedor` y `num_restos_ganados_perdedor`, y si perdió, en `num_ptos_servidos_ganador`, `num_primeros_servicios_ganados_perdedor` y `p.num_segundos_servicios_ganados_perdedor`, y si perdió, en los mismos atributos pero del ganador. El porcentaje de restos ganados por nuestro jugador será entonces el cociente entre el número de saques que su rival no consiguió ganar el servicio, que lo obtenemos como la resta entre los puntos servidos y los primeros y segundos servicios ganados, y el número total de puntos servidos por el rival, multiplicado por 100.
- El porcentaje de *breaks* salvados, que calculamos de forma análoga al porcentaje de aces pero usando las columnas `num_break_salvados_ganador` o `num_break_salvados_perdedor` en lugar de las columnas de aces, y `num_break_afrontados_ganador` o `num_break_afrontados_perdedor` en lugar de las columnas de puntos servidos.
- El porcentaje de *breaks* ganados, que calculamos de forma análoga al porcentaje de aces pero usando las estadísticas del rival. Lo obtenemos como el cociente entre la el número de *breaks* que perdió el rival (calculado como la diferencia entre afrontados y ganados) y el número total de *breaks* que afrontó el rival, multiplicado por 100.

A continuación se muestra la consulta completa y el resultado se puede ver en la figura 5.

```
with jugadores_espanoles_ganadores as (
  select distinct j.id as id_jugador, j.nombre || ' ' || j.apellido as jugador
  from partido p, jugador j, edicion_torneo et
  where p.ganador = j.id
    and p.torneo = et.torneo
    and p.fecha = et.fecha
    and j.pais = 'ES'
    and p.ronda = 'F'
    and et.nivel = 'G'
)

select jeg.jugador, count(p.num_partido) as partidos,
  round(100.0 * sum(case when jeg.id_jugador = p.ganador then 1 else 0 end) /
    count(p.num_partido), 1) as pcje_victorias,
  round(100.0 * sum(case when jeg.id_jugador = p.ganador then
    p.num_aces_ganador else p.num_aces_perdedor end) /
    nullif(sum(case when jeg.id_jugador = p.ganador then p.num_ptos_servidos_ganador
      else p.num_ptos_servidos_perdedor end), 0), 1) as pcje_aces,
```

```

round(100.0 * sum(case when jeg.id_jugador = p.ganador then p.num_dob_faltas_ganador
                     else p.num_dob_faltas_perdedor end) /
nullif(sum(case when jeg.id_jugador = p.ganador then p.num_ptos_servidos_ganador
                     else p.num_ptos_servidos_perdedor end), 0), 1) as pcje_dobles_faltas,
round(100.0 * sum(case when jeg.id_jugador = p.ganador then
                     p.num_primeros_servicios_ganados_ganador + p.num_segundos_servicios_ganados_ganador
                     else p.num_primeros_servicios_ganados_perdedor + p.num_segundos_servicios_ganados_perdedor end) /
nullif(sum(case when jeg.id_jugador = p.ganador then p.num_ptos_servidos_ganador
                     else p.num_ptos_servidos_perdedor end), 0), 1) as pcje_servicios_ganados,
round(100.0 * sum(case when jeg.id_jugador = p.ganador then
                     p.num_ptos_servidos_perdedor - p.num_primeros_servicios_ganados_perdedor -
                     p.num_segundos_servicios_ganados_perdedor
                     else p.num_ptos_servidos_ganador - p.num_primeros_servicios_ganados_ganador -
                     p.num_segundos_servicios_ganados_ganador end) /
nullif(sum(case when jeg.id_jugador = p.ganador then p.num_ptos_servidos_perdedor
                     else p.num_ptos_servidos_ganador end), 0), 1) as pcje_restos_ganados,
round(100.0 * sum(case when jeg.id_jugador = p.ganador then p.num_break_salvados_ganador
                     else p.num_break_salvados_perdedor end) /
nullif(sum(case when jeg.id_jugador = p.ganador then p.num_break_afrontados_ganador
                     else p.num_break_afrontados_perdedor end), 0), 1) as pcje_breaks_salvados,
round(100.0 * sum(case when jeg.id_jugador = p.ganador then
                     p.num_break_afrontados_perdedor - p.num_break_salvados_perdedor
                     else p.num_break_afrontados_ganador - p.num_break_salvados_ganador end) /
nullif(sum(case when jeg.id_jugador = p.ganador then p.num_break_afrontados_perdedor
                     else p.num_break_afrontados_ganador end), 0), 1) as pcje_breaks_ganados
from jugadores_espanoles_ganadores jeg, partido p
where jeg.id_jugador = p.ganador
      or jeg.id_jugador = p.perdedor
group by jeg.jugador

```

	Az jugador	123 par1	123 pcje_vici1	123 pcje_ace	123 pcje_dobles_fal	123 pcje_servicios_gan	123 pcje_restos_gan	123 pcje_breaks_salv	123 pcje_breaks_gan
1	Albert Costa	648	58,5	4,5	2,6	62,5	39,5	60,3	41
2	Carlos Alcaraz	183	79,8	4,6	3	65,4	41,6	63,8	41,1
3	Carlos Moya	873	63,9	6,4	3,2	63,4	39	62,8	40,5
4	Juan Carlos Ferrero	717	64,3	5,1	2,8	63,7	39,7	62,5	40,1
5	Rafael Nadal	1.276	82,3	4,3	2,3	67,3	42,4	66,1	44,9
6	Sergi Bruguera	601	62,9	5,2	2,5	61,9	41,6	59,2	44,9

Fig. 5. Modelo relacional Postgresql, consulta 4.

5. Lista los jugadores que fueron derrotados (en algún partido del 2018) por el rival de Rafael Nadal de la primera ronda (R128) de Roland Garros de 2018

Esta consulta la dividiremos de nuevo en dos partes. Primero, buscaremos el rival de Rafael Nadal en la primera ronda de Roland Garros de 2018, y, a continuación, buscaremos los jugadores que fueron derrotados por ese rival en algún partido de 2018. Para la primera parte, necesitamos información de los partidos, los jugadores, los torneos y sus ediciones; comenzamos en el `from` con el producto cartesiano de las tablas jugador, partido, torneo y edición_torneo (especificando un alias para cada una de ellas). Aquí, de nuevo, usamos la tabla jugador dos veces, una para identificar a un jugador (que será Rafael Nadal) y otra para identificar al rival sin ambigüedad. A continuación, especificamos las condiciones de `join` en el `where`, así como otras condiciones de selección:

- Condiciones de `join` que, en global, nos permitirán seleccionar las tuplas de partidos específicos:
 - El predicado `jg.id = p.ganador` sirve para unir la tabla jugador con la tabla partido mediante el atributo `id` de la tabla jugador y el atributo `ganador` de la tabla partido, haciendo una selección únicamente de las tuplas que cumplen esta condición. Esto nos permite seleccionar solo los partidos en los que ganó un jugador concreto.

- El predicado `jp.id = p.perdedor` sirve para unir la tabla `jugador` con la tabla `partido` mediante el atributo `id` de la tabla `jugador` y el atributo `perdedor` de la tabla `partido`, haciendo una selección únicamente de las tuplas que cumplan esta condición. Esto nos permite seleccionar solo los partidos en los que un jugador concreto es el perdedor del partido.
 - El predicado `p.torneo = et.torneo` sirve para unir la tabla `partido` con la tabla `edicion_torneo` mediante el atributo `torneo` de la tabla `partido` y el atributo `torneo` de la tabla `edicion_torneo`, haciendo una selección únicamente de las tuplas que cumplan esta condición. Esto nos permite seleccionar solo los partidos que se han jugado en una edición concreta de un torneo.
 - El predicado `p.fecha = et.fecha` sirve para unir la tabla `partido` con la tabla `edicion_torneo` mediante el atributo `fecha` de la tabla `partido` y el atributo `fecha` de la tabla `edicion_torneo`, haciendo una selección únicamente de las tuplas que cumplan esta condición. Esto nos permite seleccionar solo los partidos que se han jugado en una fecha concreta.
 - El predicado `t.id = et.torneo` sirve para unir la tabla `torneo` con la tabla `edicion_torneo` mediante el atributo `id` de la tabla `torneo` y el atributo `torneo` de la tabla `edicion_torneo`, haciendo una selección únicamente de las tuplas que cumplan esta condición. Esto nos permite seleccionar solo los torneos que se han jugado en una edición concreta.
- Ahora, queremos especificar que ese rival sea Rafael Nadal (no sabemos si ganó o perdió), que el torneo sea Roland Garros de 2018 y que la ronda sea la primera ronda (R128). Por tanto, las condiciones de selección son:
 - El predicado `p.ronda = 'R128'` selecciona solo los partidos que han sido de la primera ronda.
 - El predicado `t.nombre = 'Roland Garros'` selecciona solo los partidos que se han jugado en el torneo "Roland Garros".
 - El predicado `extract(year from p.fecha) = '2018'` selecciona solo los partidos que se han jugado en el año 2018.
 - El predicado `jp.nombre = 'Rafael' and jp.apellido = 'Nadal' or jp.nombre = 'Rafael' and jp.apellido = 'Nadal'` selecciona solo los partidos en los que ha participado Rafael Nadal, ya sea como ganador o como perdedor.

El resultado es entonces una única tupla, con ese partido de primera ronda de Roland Garros 2018. Como queremos la información del rival, obtenemos como proyección su `id` y su nombre completo, que concatenamos con el operador `||`. Al no conocer de antemano si Rafael Nadal ganó o perdió, no podemos saber si el rival fue el ganador o el perdedor, por lo que debemos usar un `case` para seleccionar al ganador si Nadal fue el perdedor y viceversa. Guardamos esta información en una tabla llamada `rival_nadal`, que usaremos en la segunda parte de la consulta.

La segunda parte de la consulta es más simple: queremos ver qué jugadores perdieron contra este rival en el año 2018. Por tanto, necesitamos información de los jugadores, los partidos y la tabla `rival_nadal`; comenzamos en el `from` con el producto cartesiano de las tablas `jugador`, `partido` y `rival_nadal` (especificando un alias para cada una de ellas). A continuación, especificamos las condiciones de `join` en el `where`, así como otras condiciones de selección:

- Condiciones de `join` que, en global, nos permitirán seleccionar las tuplas de partidos específicos:
 - El predicado `rn.id_jugador = p.ganador` nos permite seleccionar solo los partidos en los que el rival de Rafael Nadal anterior fue el ganador.

- El predicado `p.perdedor = j.id` nos permite seleccionar solo los partidos en los que el perdedor fue un jugador específico.
- Ya con esto tenemos tuplas de los partidos ganados por el rival, pero falta especificar el año con la siguiente condición de selección:
 - El predicado `extract(year from p.fecha) = '2018'` selecciona solo los partidos que se han jugado en el año 2018.

Esto nos devuelve las tuplas de partidos de 2018 con el rival de Nadal como ganador, y al no haber repetición, no es necesario usar un `distinct`. En la proyección, obtenemos el nombre completo del jugador perdedor y su país, que mostraremos como el código iso 2 correspondiente. El código de la consulta se muestra a continuación, y el resultado se puede ver en la figura 6.

```
with rival_nadal as (
    select case when jg.nombre = 'Rafael' then jp.id else jg.id end as id_jugador,
           case when jg.nombre = 'Rafael' then jp.nombre || ' ' || jp.apellido
                 else jg.nombre || ' ' || jg.apellido end as jugador
      from partido p, jugador jg, jugador jp, edicion_torneo et, torneo t
     where p.ganador = jg.id
           and p.perdedor = jp.id
           and p.torneo = et.torneo
           and p.fecha = et.fecha
           and et.torneo = t.id
           and t.nombre = 'Roland Garros'
           and p.ronda = 'R128'
           and extract(year from p.fecha) = '2018'
           and (jg.nombre = 'Rafael' and jg.apellido = 'Nadal'
                 or jp.nombre = 'Rafael' and jp.apellido = 'Nadal')
)
select j.nombre || ' ' || j.apellido as jugador, j.pais as pais
  from rival_nadal rn, partido p, jugador j
 where rn.id_jugador = p.ganador
   and p.perdedor = j.id
   and extract(year from p.fecha) = '2018'
```

	A-Z jugador	A-Z pais
1	Joao Domingues	PT
2	Federico Delbonis	AR
3	Pablo Cuevas	UY
4	Roberto Carballes Baena	ES
5	Diego Schwartzman	AR
6	Denis Istomin	UZ

Fig. 6. Modelo relacional Postgresql, consulta 5.

2. DATOS AGREGADOS EN SQL (POSTGRESQL)

A continuación, queremos crear una tabla que contenga todos los datos de los partidos (países, torneos, ediciones, partidos, sets y jugadores), pero con datos agregados. Concretamente, debemos definir estructuras que permitan resolver de forma eficiente las consultas Q1 y Q3 de la sección anterior. Para mantener orden en nuestra base de datos, crearemos un nuevo esquema agregados para alojar esta nueva tabla.

El sistema de tipos de PostgreSQL tiene constructores de tipo ROW, para tipos compuestos, y ARRAY para colecciones (en nuestro caso usaremos solo arrays unidimensionales). Para definir una tabla con tipos compuestos que resuelva eficientemente las consultas deseadas, el esquema debe ir orientado a los partidos en lugar de a los jugadores. Buscaremos definir entonces un esquema donde cada fila de la tabla represente un partido. Para ello, definimos los siguientes tipos:

A. Tipos compuestos

- pais_info: contiene información sobre un país, con los campos originales de dicha tabla: codigo_iso2, codigo_iso3, codigo_ioc y nombre.

```
create type pais_info as (
    codigo_iso2 char(2),
    codigo_iso3 char(3),
    codigo_ioc char(3),
    nombre varchar(100)
)
```

- torneo_info: contiene información sobre un torneo, con los campos originales de dicha tabla: id, nombre y pais. El campo pais será de tipo pais_info.

```
create type torneo_info as (
    id integer,
    nombre varchar(100),
    pais pais_info
)
```

- edicion_torneo_info: contiene información sobre una edición de un torneo, con los campos originales de dicha tabla: torneo, fecha, superficie, tamano y nivel. El campo torneo será de tipo torneo_info.

```
create type edicion_torneo_info as (
    torneo torneo_info,
    fecha date,
    superficie varchar(20),
    tamano integer,
    nivel char(1)
)
```

- set_info: contiene información sobre un set de un partido, con los campos originales de su tabla correspondiente: torneo, fecha, num_partido, num_set, juegos_ganador, juegos_perdedor y puntos_tiebreak_perdedor. El campo torneo será de tipo torneo_info.

```
create type set_info as (
    torneo torneo_info,
    fecha date,
    num_partido integer,
    num_set integer,
    juegos_ganador integer,
```

```

        juegos_perdedor integer,
        puntos_tiebreak_perdedor integer
)

```

- La tabla original de partidos tenía mucha información, ya que incluía estadísticas del ganador y del perdedor. Para simplificar, definimos un tipo jugador_stats que contenga solo las estadísticas de un jugador en un partido, con las estadísticas que se daban en la tabla partido: num_aces, num_dob_faltas, num_ptos_servidos, num_primeros_servicios, num_primeros_servicios_ganados, num_segundos_servicios_ganados, num_juegos_servidos, num_break_salvados y num_break_afrontados.

```

create type jugador_stats as (
    num_aces integer,
    num_dob_faltas integer,
    num_ptos_servidos integer,
    num_primeros_servicios integer,
    num_primeros_servicios_ganados integer,
    num_segundos_servicios_ganados integer,
    num_juegos_servidos integer,
    num_break_salvados integer,
    num_break_afrontados integer
)

```

- jugador_info: contiene información sobre un jugador, con los campos originales de dicha tabla: id, nombre, apellido, diestro, fecha_nacimiento, pais y altura. El campo pais será de tipo pais_info.

```

create type jugador_info as (
    id integer,
    nombre varchar(100),
    apellido varchar(100),
    diestro boolean,
    fecha_nacimiento date,
    pais pais_info,
    altura integer
)

```

- La tabla que unifique todo será partidos y, como comentamos, tendrá una fila por cada partido. Los campos serán los originales de la tabla partido, pero con los tipos compuestos que acabamos de definir: torneo será de tipo torneo_info, ganador y perdedor serán de tipo jugador_info, y ganador_stats y perdedor_stats serán de tipo jugador_stats (almacenamos las estadísticas de una forma mucho más limpia). Además, el campo info_sets será un array de elementos de tipo set_info (al ser el array homogéneo y declarar que todos los elementos sean del mismo tipo, aunque sea un tipo compuesto, podemos definir el array).

```

create table partidos (
    torneo edicion_torneo_info,
    fecha date,
    num_partido integer,
    num_sets integer,
    info_sets set_info array, -- array de elementos de tipo set_info
    ronda varchar(5),
    desenlace char(1),
    ganador jugador_info,
    perdedor jugador_info,
    ganador_stats jugador_stats,
    perdedor_stats jugador_stats
)

```

Una vez construida la tabla, queremos construir sus filas (insertar datos) a partir de las tablas del modelo relacional antes usado. Antes de introducir datos en un tipo compuesto, verificaremos que la clave primaria que tenía ese registro en la tabla relacional no sea nula y, en caso de serlo, insertaremos un null en lugar del registro. Para ello, usaremos un *case*. La función *cast()* nos permitirá agrupar valores individuales dentro de un tipo compuesto. Esto deberemos usarlo cada vez que queramos insertar un país, un torneo, la información de un set, las estadísticas de los jugadores o los propios jugadores.

Lo primero es seleccionar las tablas a usar en el *from*. Partimos del producto cartesiano entre *partido*, *edicion_torneo*, *torneo*, *pais* (3 veces, una para el país del torneo, otra para el país del ganador y otra para el país del perdedor) y *jugador* (2 veces, una para el ganador y otra para el perdedor). En este caso, las condiciones de *join*, así como el tipo, son muy importantes, ya que podemos perder información por el camino.

- Usamos un *inner join* para unir las tablas *partido* y *edicion_torneo* mediante los campos *fecha* y *torneo*. Esto nos permite seleccionar solo los partidos que se han jugado en una edición concreta de un torneo.
- Usamos un *inner join* para unir las tablas *edicion_torneo* y *torneo* mediante el campo *torneo*. Esto nos permite seleccionar ediciones de un torneo concreto.
- Usamos un *left join* para unir las tablas *torneo* y *pais* mediante el campo *pais*. Esto nos permite seleccionar torneos que se han jugado en un país concreto. Importante usar aquí un *left join* para no perder información de los torneos que no tienen país registrado, como son *ATP Cup*, *Tour Finals* y *Masters Cup* (NULL).
- Usamos un *left join* para unir las tablas *partido* y *jugador* mediante el campo *ganador*. Esto nos permite seleccionar los jugadores que han ganado un partido concreto. Importante usar aquí un *left join* para no perder información de los partidos que no tienen ganador registrado (NULL).
- Usamos un *left join* para unir las tablas *jugador* y *pais* mediante el campo *pais* (para el ganador). Esto nos permite seleccionar jugadores que son de un país concreto. Aquí usamos un *left join* para no perder información de los jugadores que no tienen país registrado (NULL) o cuyo país no está en la tabla *pais*. Con unas consultas sencillas se puede comprobar que actualmente no existe este problema, pero haciendo un *inner join* perdemos 2 registros del total, por lo que un *left join* asegura que no perdamos información.
- Usamos un *left join* para unir las tablas *partido* y *jugador* mediante el campo *perdedor*. Esto nos permite seleccionar los jugadores que han perdido un partido concreto. Importante usar aquí un *left join* para no perder información de los partidos que no tienen perdedor registrado (NULL).
- Usamos un *left join* para unir las tablas *jugador* y *pais* mediante el campo *pais* (para el perdedor). Esto nos permite seleccionar jugadores que son de un país concreto. Aquí usamos un *left join* por la misma razón que antes: aparentemente no hay inconsistencias, pero si cambiamos este por un *inner join*, perdemos un total de 5 registros.

Explicamos ahora brevemente el proceso de inserción de datos en la tabla *partidos*, una vez definido el producto del cual haremos una proyección:

- Comenzamos con el atributo *torneo*, de tipo *edicion_torneo_info*. Usamos un *case* para comprobar si el torneo existe en la tabla *torneo*. Si no existe, insertamos un null. Si existe, insertamos valores en los atributos que marca este tipo compuesto: *fecha*, *superficie*, *tamaño* y *nivel* vendrán de la tabla *edicion_torneo*, mientras que *torneo* debemos “castearlo” al tipo *torneo_info*, introduciendo sus atributos. Los atributos de este tipo vienen todos de la tabla *torneo* excepto el país, que se debe “castear” al tipo *pais_info* y tomará sus atributos de la tabla *pais*.
- Los atributos *fecha*, *num_partido*, *num_sets*, *ronda* y *desenlace* se insertan directamente de la tabla *partido*.

- El atributo `info_sets` es un array de elementos de tipo `set_info`. Para construirlo, usamos un array con una subconsulta que selecciona los sets de un partido concreto. Cada set se construye con un `cast` que agrupa los valores de la tabla `sets_partido` en un tipo compuesto `set_info`.

En dicha subconsulta, en la que solo usamos la tabla `sets_partido`, fijamos condiciones de *join* para unir esta tabla con la tabla `partido`, mediante los campos `torneo`, `fecha` y `num_partido`. Así, obtenemos la información relativa a los sets de un partido concreto. Ordenamos en orden ascendente según el número del set y, finalmente, obtenemos la proyección de la información de los sets de ese partido.

- Los atributos `ganador` y `perdedor` son de tipo `jugador_info` y los obtenemos de forma completamente análoga. Usamos un `case` para comprobar si el ganador o el perdedor existen en la tabla `jugador`. Si no existen, insertamos un `null`. Si existen, insertamos valores en los atributos que marca este tipo compuesto: nombre, apellido, diestro, fecha de nacimiento y altura vendrán de la tabla `jugador`. El país se debe “castear” al tipo `pais_info` y tomará sus atributos de la tabla `pais`.
- Los atributos `ganador_stats` y `perdedor_stats` son de tipo `jugador_stats` y los obtenemos de forma completamente análoga. Usamos un `case` para comprobar si el ganador o el perdedor existen en la tabla `jugador`. Si no existen, insertamos un `null`. Si existen, insertamos valores en los atributos que marca este tipo compuesto: número de aces, dobles faltas, puntos servidos, primeros servicios, primeros servicios ganados, segundos servicios ganados, juegos servidos, breaks salvados y breaks afrontados vendrán todos dados por la tabla `partido`, para el jugador que se esté usando.

A continuación, la consulta para insertar los datos en nuestra nueva tabla desde la base de datos relacional.

```

insert into partidos(
torneo, fecha, num_partido, num_sets, info_sets, ronda, desenlace, ganador, perdedor,
ganador_stats, perdedor_stats)
select
    -- campo 'torneo' (tipo 'edicion_torneo_info')
    case
        when t.id is null then null
        else cast((cast((t.id, t.nombre, cast((pa.codigo_iso2, pa.codigo_iso3, pa.codigo_ioc,
            pa.nombre) as pais_info)) as torneo_info),
            et.fecha, et.superficie, et.tamano, et.nivel) as edicion_torneo_info)
    end as torneo,

    p.fecha as fecha,
    p.num_partido as num_partido,
    p.num_sets as num_sets,

    -- campo 'info_set' (array de 'set_info')
    array(select cast((cast((t.id, t.nombre, cast((pa.codigo_iso2, pa.codigo_iso3, pa.codigo_ioc,
        pa.nombre) as pais_info)) as torneo_info),
            sp.fecha, sp.num_partido, sp.num_set, sp.juegos_ganador, sp.juegos_perdedor,
            sp.puntos_tiebreak_perdedor) as set_info)
        from public.sets_partido sp
        where sp.torneo = p.torneo
        and sp.fecha = p.fecha
        and sp.num_partido = p.num_partido
        order by sp.num_set) as info_sets,

    p.ronda as ronda,
    p.desenlace as desenlace,

    -- campo 'ganador' (tipo 'jugador_info')
    case
        when jg.id is null then null
        else cast((jg.id, jg.nombre, jg.apellido, jg.diestro, jg.fecha_nacimiento,

```

```

        cast((pg.codigo_iso2, pg.codigo_iso3, pg.codigo_ioc, pg.nombre) as pais_info),
        jg.altura) as jugador_info)
end as ganador,

-- campo 'perdedor' (tipo 'jugador_info')
case
    when jp.id is null then null
    else cast((jp.id, jp.nombre, jp.apellido, jp.diestro, jp.fecha_nacimiento,
               cast((pp.codigo_iso2, pp.codigo_iso3, pp.codigo_ioc, pp.nombre) as pais_info),
               jp.altura) as jugador_info)
end as perdedor,

-- campo 'ganador_stats' (tipo 'jugador_stats')
case
    when jg.id is null and jp.id is null then null
    else cast((p.num_aces_ganador, p.num_dob_faltas_ganador,
               p.num_ptos_servidos_ganador, p.num_primeros_servicios_ganador,
               p.num_primeros_servicios_ganados_ganador, p.num_segundos_servicios_ganados_ganador,
               p.num_juegos_servidos_ganador, p.num_break_salvados_ganador,
               p.num_break_afrontados_ganador) as jugador_stats)
end as ganador_stats,

-- campo 'perdedor_stats' (tipo 'jugador_stats')
case
    when jg.id is null and jp.id is null then null
    else cast((p.num_aces_perdedor, p.num_dob_faltas_perdedor,
               p.num_ptos_servidos_perdedor, p.num_primeros_servicios_perdedor,
               p.num_primeros_servicios_ganados_perdedor, p.num_segundos_servicios_ganados_perdedor,
               p.num_juegos_servidos_perdedor, p.num_break_salvados_perdedor,
               p.num_break_afrontados_perdedor) as jugador_stats)
end as perdedor_stats
from public.partido p join public.edicion_torneo et on et.fecha = p.fecha and et.torneo = p.torneo
    join public.torneo t on t.id = et.torneo
    left join public.pais pa on t.pais = pa.codigo_iso2
    left join public.jugador jg on p.ganador = jg.id
    left join public.pais pg on jg.pais = pg.codigo_iso2
    left join public.jugador jp on p.perdedor = jp.id
    left join public.pais pp on jp.pais = pp.codigo_iso2

```

A.1. Muestra todos los ganadores del torneo “Wimbledon” (Nombre apellido y año). Ordena el resultado por año.

Para reescribir esta consulta, debemos tener en cuenta que ahora solo existe una tabla, por lo que no necesitamos hacer productos cartesianos ni *joins*. Simplemente, especificamos condiciones sobre los campos y seleccionamos los campos que queremos mostrar. Para acceder a un campo dentro de un tipo compuesto que sea columna de la tabla, será necesario usar paréntesis; veámoslo en la siguiente consulta.

Partiendo de la tabla partidos, seleccionamos aquellos cuyo torneo sea “Wimbledon” y la ronda sea “F” (final). A la ronda se accede de forma fácil, al ser un atributo de la propia tabla partidos. Sin embargo, para acceder al nombre del torneo, debemos acceder a nombre, dentro de torneo (de tipo *torneo_info*), dentro de torneo (de tipo *edicion_torneo*). Para ello, debemos usar paréntesis para acceder al primer torneo: (*torneo*).*torneo.nombre*. Equivalentemente, podríamos haber usado ((*torneo*).*torneo.nombre*, pero al acceder primero al tipo compuesto, el gestor ya tiene claro que esa columna se trata de un tipo compuesto y no requiere una segunda aclaración con el doble paréntesis.

Tras la selección, tenemos un conjunto de tuplas con los partidos finales de Wimbledon. Obtenemos como proyección el nombre y apellido del ganador (campos a los cuales debemos acceder mediante (*ganador*), al ser un tipo compuesto), y el año de la edición del torneo, que lo obtenemos con un simple *extract*. Para terminar, ordenamos el resultado por año. La consulta se muestra a continuación y el resultado, que coincide con lo

esperado, se puede ver en la figura 7.

```
select (ganador).nombre, (ganador).apellido, extract(year from fecha) as ano
from partidos
where (torneo).torneo.nombre = 'Wimbledon'
    and ronda = 'F'
order by ano
```

	A-Z nombre	A-Z apellido	123 ano
1	Michael	Stich	1.991
2	Andre	Agassi	1.992
3	Pete	Sampras	1.993
4	Pete	Sampras	1.994
5	Pete	Sampras	1.995
6	Richard	Krajicek	1.996
7	Pete	Sampras	1.997
8	Pete	Sampras	1.998
9	Pete	Sampras	1.999
10	Pete	Sampras	2.000
11	Goran	Ivanisevic	2.001
12	Lleyton	Hewitt	2.002
13	Roger	Federer	2.003
14	Roger	Federer	2.004
15	Roger	Federer	2.005
16	Roger	Federer	2.006
17	Roger	Federer	2.007
18	Rafael	Nadal	2.008
19	Roger	Federer	2.009
20	Rafael	Nadal	2.010
21	Novak	Djokovic	2.011
22	Roger	Federer	2.012
23	Andy	Murray	2.013
24	Novak	Djokovic	2.014
25	Novak	Djokovic	2.015
26	Andy	Murray	2.016
27	Roger	Federer	2.017
28	Novak	Djokovic	2.018
29	Novak	Djokovic	2.019
30	Novak	Djokovic	2.021
31	Novak	Djokovic	2.022
32	Carlos	Alcaraz	2.023

Fig. 7. Datos agregados en SQL. Tipos compuestos, consulta 1.

A.2. Muestra los años en los que Roger Federer ganó algún torneo de nivel Gran Slam (G) o Master 1000 (M). Para cada año, muestra el número de torneos y lista sus nombres (ordenados por la fecha de celebración). Ordena el resultado por el año

Para reescribir esta consulta, usamos la única tabla de la que disponemos. Fijamos como condiciones de selección que el nombre del ganador sea Roger y el apellido Federer (campos a los que accedemos mediante ganador, que es un tipo compuesto), que el nivel del torneo sea G o M (accedemos a nivel mediante (torneo).nivel, ya que nivel es un campo del atributo torneo, de tipo edicion_torneo) y que la ronda sea final (F). Esto nos devuelve una tupla para cada partido final de un Grand Slam o Master 1000 que Roger Federer ha ganado. Agrupamos esto por año, que lo obtenemos desde la fecha del torneo, a la cual accedemos a través del tipo torneo como (torneo).fecha, para obtener una única fila para cada año. Como proyección nos quedamos con el año del torneo (de los torneos concretamente), el número de torneos que hay en esa agrupación (los podemos contar por el número de id distintos que haya, accediendo a (torneo).torneo.id), y una concatenación del nombre de los torneos ganados ese año por Federer ordenada por fecha (el campo nombre lo encontramos, de nuevo, accediendo a (torneo).torneo). A continuación, mostramos la consulta y el resultado, que coincide con lo esperado, se puede ver en la figura 8.

```
select extract(year from (torneo).fecha) as ano, count(distinct (torneo).torneo.id) as numero_torneos,
       string_agg((torneo).torneo.nombre, ', ' order by (torneo).fecha) as torneos
from partidos
where (torneo).nivel in ('G', 'M')
```

```

    and (ganador).nombre = 'Roger'
    and (ganador).apellido = 'Federer'
    and ronda = 'F'
group by ano

```

	ano	numero_torneos	torneos
1	2.002	1	Hamburg Masters
2	2.003	1	Wimbledon
3	2.004	6	Australian Open, Indian Wells Masters, Hamburg Masters, Wimbledon, Canada Masters, US Open
4	2.005	6	Miami Masters, Indian Wells Masters, Hamburg Masters, Wimbledon, US Open, Cincinnati Masters
5	2.006	7	Australian Open, Miami Masters, Indian Wells Masters, Wimbledon, US Open, Canada Masters, Madrid Masters
6	2.007	5	Australian Open, Hamburg Masters, Wimbledon, US Open, Cincinnati Masters
7	2.008	1	US Open
8	2.009	4	Roland Garros, Madrid Masters, Wimbledon, Cincinnati Masters
9	2.010	2	Australian Open, Cincinnati Masters
10	2.011	1	Paris Masters
11	2.012	4	Indian Wells Masters, Madrid Masters, Wimbledon, Cincinnati Masters
12	2.014	2	Cincinnati Masters, Shanghai Masters
13	2.015	1	Cincinnati Masters
14	2.017	5	Australian Open, Indian Wells Masters, Miami Masters, Wimbledon, Shanghai Masters
15	2.018	1	Australian Open

Fig. 8. Datos agregados en SQL. Tipos compuestos, consulta 2.

A.3. Muestra los partidos de semifinales (ronda='SF') y final (ronda = 'F') del torneo de "Roland Garros" del 2018. Para cada partido muestra la ronda, el tipo de desenlace, el nombre y apellidos del ganador y el nombre y apellidos del perdedor y el resultado con el número de juegos del ganador y del perdedor en cada set, y opcionalmente en paréntesis el número de juegos del perdedor en el tie break

En este caso, las condiciones de selección que deben ir en el where son la ronda (semifinal o final), el nombre del torneo (Roland Garros) y el año (2018). Para cada partido, queremos mostrar la ronda, el desenlace, el nombre y apellidos del ganador y del perdedor, y el resultado con el número de juegos del ganador y del perdedor en cada set. Para ello, debemos acceder a los campos de los tipos compuestos que hemos definido. Todo esto se obtiene de forma fácil y directa accediendo a las columnas necesarios, usando paréntesis en caso de que esa columna sea un tipo compuesto, a excepción de la información de los sets. Veamos esto en concreto con un poco más de detalle.

Para obtener la información de los sets, debemos acceder a la columna info_sets, que es un array con elementos de tipo compuesto. Para acceder a sus elementos, debemos hacer una subconsulta donde desagreguemos el array en el from con la función unnest. Tras desagregar el array, se trata como una consulta estándar y accedemos a los campos de los elementos del array como si fueran columnas de una tabla. En este caso, accedemos a los campos juegos_ganador, juegos_perdedor y puntos.tiebreak_perdedor de cada set. Para mostrar estos datos de forma más clara, concatenamos los resultados en un solo string con la función string_agg, mostrando la puntuación del tiebreak solo si existe. Además, esta concatenación se hace por orden ascendente según el número del set. Así, obtenemos una única tupla con los información de los sets de un partido. La consulta y el resultado, que coincide con lo esperado, se pueden ver en la figura 9.

```

select ronda, desenlace,
       (ganador).nombre || ' ' || (ganador).apellido as ganador,
       (perdedor).nombre || ' ' || (perdedor).apellido as perdedor,
       (select string_agg(iset.juegos_ganador || '-' || iset.juegos_perdedor || 
case
        when iset.puntos.tiebreak_perdedor is not null then
          '(' || iset.puntos.tiebreak_perdedor || ')'
        else ''
      end, ', ' order by iset.num_set)
       from unnest(info_sets) as iset) as resultado
from partidos
where ronda in ('SF', 'F')

```

```
and (torneo).torneo.nombre = 'Roland Garros'
and extract(year from fecha) = '2018'
```

	O	A-Z ronda	A-Z desenlace	A-Z ganador	A-Z perdedor	A-Z resultado
1		SF	N	Rafael Nadal	Juan Martin del Potro	6-4, 6-1, 6-2
2		SF	N	Dominic Thiem	Marco Cecchinato	7-5, 7-6(10), 6-1
3		F	N	Rafael Nadal	Dominic Thiem	6-4, 6-3, 6-2

Fig. 9. Datos agregados en SQL. Tipos compuestos, consulta 3.

A.4. Muestra la lista de jugadores españoles (ES) que ganaron algún torneo de nivel Gran Slam (G). Para cada jugador muestra los siguientes datos resumen de todos sus partidos: número de partidos jugados, porcentaje de victorias, porcentaje de aces, porcentaje de dobles faltas, porcentaje de servicios ganados, porcentaje de restos ganados, porcentaje de break points salvados (de los sufridos en contra), porcentaje de break points ganados (de los provocados a favor)

Esta consulta, al igual que en el caso del modelo relacional, la dividiremos en dos partes, una para obtener los jugadores españoles que han ganado algún torneo de nivel Gran Slam y otra para obtener los datos resumen de todos sus partidos. Para la primera parte, usando la tabla partidos en el from, fijamos como condiciones de selección que el país del ganador sea España (accedemos a la columna de ganador y seleccionamos, del campo pais, el código iso 2), el nivel del torneo sea G y la ronda sea final. Esto nos devuelve una tupla por cada edición del torneo ganado por un jugador español, por lo que en la proyección usaremos un distinct para quedarnos con una lista de jugadores únicos: queremos su id y su nombre completo, como concatenación de nombre y apellido. Todos estos campos son accesibles desde la columna (ganador). El resultado será una tabla que llamaremos jugadores_espanoles_ganadores.

Para la segunda parte usaremos el producto cartesiano de jugadores_espanoles_ganadores y partidos. Como condiciones de join usaremos los id de los jugadores, tanto del ganador como del perdedor. Estos son campos dentro de las columnas de tipo compuesto p.ganador y p.perdedor. Para obtener los datos resumen del partido en la proyección, usaremos un razonamiento análogo al que se usó en la consulta 3 del modelo relacional, por lo que no entreamos a ese nivel de detalle. Sin embargo, comentamos los detalles técnicos que diferencian ambas consultas en este aspecto concreto:

- Para verificar si el jugador español es el ganador o el perdedor, necesitamos recurrir a id_jugador, de la tabla jugadores_espanoles_ganadores y el id del jugador, que encontramos como atributo del ganador o perdedor en la tabla partidos, que son columnas de tipo compuesto.
- Para acceder a las estadísticas, al ser columnas de tipo compuesto, debemos acceder usando también paréntesis, al igual que para los atributos del ganador o el perdedor, esto es (p.ganador_stats) . o (p.perdedor_stats) .

Teniendo en cuenta esto, la consulta se muestra a continuación y el resultado, que coincide con lo esperado, se pueden ver en la figura 10.

```
with jugadores_espanoles_ganadores as (
    select distinct (ganador).id as id_jugador, (ganador).nombre || ' ' || (ganador).apellido AS jugador
    from partidos
    where (ganador).pais.codigo_iso2 = 'ES'
        and ronda = 'F'
        and (torneo).nivel = 'G'
)
select
    jeg.jugador,
    count(p.num_partido) as partidos,
```

```

round(100.0 * sum(case when jeg.id_jugador = (p.ganador).id then 1 else 0 end) /
      count(p.num_partido), 1) as pcje_victorias,
round(100.0 * sum(case when jeg.id_jugador = (p.ganador).id then (p.ganador_stats).num_aces
      else (p.perdedor_stats).num_aces end) /
      nullif(sum(case when jeg.id_jugador = (p.ganador).id then (p.ganador_stats).num_ptos_servidos
                  else (p.perdedor_stats).num_ptos_servidos end), 0), 1) as pcje_aces,
round(100.0 * sum(case when jeg.id_jugador = (p.ganador).id then (p.ganador_stats).num_dob_faltas
      else (p.perdedor_stats).num_dob_faltas end) /
      nullif(sum(case when jeg.id_jugador = (p.ganador).id then (p.ganador_stats).num_ptos_servidos
                  else (p.perdedor_stats).num_ptos_servidos end), 0), 1) as pcje_dobles_faltas,
round(100.0 * sum(case when jeg.id_jugador = (p.ganador).id
      then (p.ganador_stats).num_primeros_servicios_ganados +
            (p.ganador_stats).num_segundos_servicios_ganados
      else (p.perdedor_stats).num_primeros_servicios_ganados +
            (p.perdedor_stats).num_segundos_servicios_ganados end) /
      nullif(sum(case when jeg.id_jugador = (p.ganador).id then (p.ganador_stats).num_ptos_servidos
                  else (p.perdedor_stats).num_ptos_servidos end), 0), 1) as pcje_servicios_ganados,
round(100.0 * sum(case when jeg.id_jugador = (p.ganador).id
      then (p.perdedor_stats).num_ptos_servidos - (p.perdedor_stats).num_primeros_servicios_ganados -
            (p.perdedor_stats).num_segundos_servicios_ganados
      else (p.ganador_stats).num_ptos_servidos - (p.ganador_stats).num_primeros_servicios_ganados -
            (p.ganador_stats).num_segundos_servicios_ganados end) /
      nullif(sum(case when jeg.id_jugador = (p.ganador).id then (p.perdedor_stats).num_ptos_servidos
                  else (p.ganador_stats).num_ptos_servidos end), 0), 1) as pcje_restos_ganados,
round(100.0 * sum(case when jeg.id_jugador = (p.ganador).id then (p.ganador_stats).num_break_salvados
      else (p.perdedor_stats).num_break_salvados end) /
      nullif(sum(case when jeg.id_jugador = (p.ganador).id then (p.ganador_stats).num_break_afrontados
                  else (p.perdedor_stats).num_break_afrontados end), 0), 1) as pcje_breaks_salvados,
round(100.0 * sum(case when jeg.id_jugador = (p.ganador).id then (p.perdedor_stats).num_break_afrontados
      - (p.perdedor_stats).num_break_salvados
      else (p.ganador_stats).num_break_afrontados - (p.ganador_stats).num_break_salvados end) /
      nullif(sum(case when jeg.id_jugador = (p.ganador).id then (p.perdedor_stats).num_break_afrontados
                  else (p.ganador_stats).num_break_afrontados end), 0), 1) as pcje_breaks_ganados
from jugadores_espanoles_ganadores jeg, partidos p
where jeg.id_jugador = (p.ganador).id
  or jeg.id_jugador = (p.perdedor).id
group by jeg.jugador

```

	A-Z jugador	123 part	123 pcje_vict	123 pcje_ace	123 pcje_dobles_fa	123 pcje_servicios_gan	123 pcje_restos_gan	123 pcje_breaks_salv	123 pcje_breaks_gana
1	Albert Costa	648	58,5	4,5	2,6	62,5	39,5	60,3	41
2	Carlos Alcaraz	183	79,8	4,6	3	65,4	41,6	63,8	41,1
3	Carlos Moya	873	63,9	6,4	3,2	63,4	39	62,8	40,5
4	Juan Carlos Ferrero	717	64,3	5,1	2,8	63,7	39,7	62,5	40,1
5	Rafael Nadal	1.276	82,3	4,3	2,3	67,3	42,4	66,1	44,9
6	Sergi Bruguera	601	62,9	5,2	2,5	61,9	41,6	59,2	44,9

Fig. 10. Datos agregados en SQL. Tipos compuestos, consulta 4.

A.5. Lista los jugadores que fueron derrotados (en algún partido del 2018) por el rival de Rafael Nadal de la primera ronda (R128) de Roland Garros de 2018

Esta consulta la dividiremos en dos subconsultas, una para obtener el rival de Rafael Nadal en la primera ronda de Roland Garros de 2018 y otra para obtener los jugadores que fueron derrotados por ese rival. Para la primera parte, que solo requiere la tabla partidos, fijamos como condiciones de selección que la ronda sea R128, el nombre del torneo sea Roland Garros y el año sea 2018. Además, el nombre del ganador o perdedor debe ser Rafael, y su apellido Nadal. No explicamos cómo acceder a estos campos porque ya se hizo con anterioridad. Esto devuelve una única fila con el rival de Nadal en este partido concreto, por lo que nos quedamos únicamente con la proyección de su nombre completo (concatenación de nombre y apellido). La tabla resultante la llamaremos `rival_nadal`.

Para la segunda parte, usaremos el producto cartesiano de `rival_nadal` y `partidos`. Como condiciones de `join` usaremos que el nombre y apellido del rival de Nadal coincida con el nombre y apellidos del ganador de ese partido, que es una columna de tipo compuesto (`ganador`) a la cual accedemos como en las consultas anteriores. Además, si seleccionamos los partidos disputados en 2018 a través de la fecha, obtenemos un conjunto de tuplas con los partidos ganados por el rival de Nadal en 2018. En la proyección, nos quedamos simplemente con una concatenación del nombre y apellido del perdedor y su país, al que accedemos a través de `(perdedor).pais.codigo_iso2`. No haría falta, pero ordenamos el resultado por el nombre del jugador. La consulta y el resultado, que coincide con el esperado, se pueden ver en la figura 11.

```
with rival_nadal as (
    select case when (ganador).nombre = 'Rafael' then (perdedor).nombre || ' ' || (perdedor).apellido
                else (ganador).nombre || ' ' || (ganador).apellido end as jugador
    from partidos
    where (((ganador).nombre = 'Rafael' and (ganador).apellido = 'Nadal') or
           ((perdedor).nombre = 'Rafael' and (perdedor).apellido = 'Nadal'))
         and ronda = 'R128'
         and (torneo).torneo.nombre = 'Roland Garros'
         and extract(year from fecha) = '2018'
)
select (p.perdedor).nombre || ' ' || (p.perdedor).apellido as jugador, (p.perdedor).pais.codigo_iso2 as pais
from partidos p, rival_nadal rn
where (p.ganador).nombre || ' ' || (p.ganador).apellido = rn.jugador
      and extract(year from p.fecha) = '2018'
order by jugador
```

	A-Z jugador	A-Z país
1	Denis Istomin	UZ
2	Diego Schwartzman	AR
3	Federico Delbonis	AR
4	Joaõ Domingues	PT
5	Pablo Cuevas	UY
6	Roberto Carballes Baena	ES

Fig. 11. Datos agregados en SQL. Tipos compuestos, consulta 5.

B. JSON

A continuación, queremos crear de nuevo una tabla que contenga todos los datos de los partidos (países, torneos, ediciones, partidos, sets y jugadores). Concretamente, esta vez debe resolver de forma eficiente las consultas Q2 y Q4. Para mantener orden en nuestra base de datos, crearemos un nuevo esquema agregadosJSON para alojar esta nueva tabla.

PostgreSQL proporciona un tipo de datos “json” y un conjunto de funciones y operadores que permiten tanto generar documentos JSON a partir de datos relacionales como consultar partes de un documento JSON. Además del tipo “json”, también se proporciona un tipo más eficiente “jsonb”, que almacena los documentos en formato binario. Para resolver de forma eficiente las consultas deseadas, nuestra tabla deberá estar orientada a los jugadores, es decir, una tupla para cada jugador con todos sus datos y los partidos que ha jugado.

En el caso de JSON (usaremos JSONB), tenemos objetos JSON y agregados de objetos JSON. La creación del esquema de la tabla y la inserción de los datos se realiza mediante la misma consulta, que explicamos a continuación.

Comenzaremos con la tabla *jugador*, y haremos un *left join* con la tabla *país* para obtener los datos del país del jugador. El motivo del *left join* en lugar de un *inner join* lo comentamos en la sección de agregados de tipos compuestos. En el *where* realizamos un paso crítico para la eficiencia de la consulta: filtramos los jugadores que han participado en algún partido. Para ello, seleccionamos los jugadores cuyo id está en la lista de perdedores o ganadores de algún partido. Debemos hacer esto ya que, de los 60 mil jugadores existentes en la base de datos, unos 58 mil no ganaron ni perdieron ningún partido (según los datos recogidos en nuestra base de datos relacional).

Con esto tenemos la tuplas de los partidos jugadores que han participado en algún partido. Al obtener la proyección, haremos que cada tupla tenga los siguientes atributos:

- *jugador*: objeto jsonb con los datos del jugador: id, nombre, apellido, diestro, fecha de nacimiento y altura.
- *país*: objeto jsonb con los datos del país del jugador: código ISO2, código ISO3, código IOC y nombre.
- *partidos_ganados*: agregación de objetos jsonb con los datos de los partidos ganados por el jugador. Cada objeto jsonb será un partido concreto y contendrá los datos del torneo, la fecha, la ronda, el desenlace, el número de partido, el rival, los sets y las estadísticas del jugador y las del rival.

La agregación de objetos jsonb se hace mediante subconsultas, por lo que debemos pararnos a explicar la construcción de este agregado más en detalle. Para esta subconsulta partimos de la tabla *partido*, que denotaremos por el alias pg al estar tratando con los partidos ganados, y realizamos un *left join* con la tabla *edicion_torneo* mediante los atributos que definen la FK de la tabla *partido* y la PK de la tabla *edicion_torneo*. A continuación, realizamos un *left join* con la tabla *torneo* mediante los atributos que definen la FK de la tabla *edicion_torneo* y la PK de la tabla *torneo*. Finalmente, hacemos otro *left join* con la tabla *país* mediante el código ISO2. Seleccionamos solo los partidos ganados por el jugador que estamos tratando y construimos un objeto jsonb con los datos del torneo, la fecha, la ronda, el desenlace, el número de partido, el rival, los sets y las estadísticas del jugador y las del rival.

De estos atributos, fecha, ronda, desenlace, num_partido y perdedor son directamente obtenidos de la tabla *partido* pg. Además, torneo es simplemente un objeto jsonb con los datos relativos a la edición del torneo: nombre, país (que será un objeto jsonb con los atributos del país que se reflejan en la tabla relacional de país), fecha, superficie, tamaño y nivel. Las estadísticas de el jugador y del rival son

también objetos jsonb que toman los datos estadísticos de la tabla `partido_pg`.

Por último, los sets son un agregado de objetos jsonb que se construye mediante una subconsulta que selecciona los sets de un partido concreto. Cada objeto jsonb de este agregado contiene el número de set, los juegos ganados por el jugador, los juegos ganados por el rival y los puntos del tiebreak en caso de que se haya llegado a este. Para hacer la subconsulta descrita nos centramos en la tabla `sets_partido`. La unimos con condiciones de *join* a `partido`, igualando los atributos que componen la clave foránea de `sets_partido` con los atributos que componen la clave primaria de `partido`, seleccionando así solo los sets del partido que estamos tratando.

- `partidos_perdidos`: este agregado de objetos jsonb lo obtenemos de forma completamente análoga a `partidos_ganados`, pero con los partidos perdidos por el jugador.

```
create table tenisjson as
select
    -- columna 'jugador' como objeto json
    jsonb_build_object(
        'id', j.id,
        'nombre', j.nombre,
        'apellido', j.apellido,
        'diestro', j.diestro,
        'fecha_nacimiento', j.fecha_nacimiento,
        'altura', j.altura
    ) as jugador,
    -- columna 'pais' como objeto json
    jsonb_build_object(
        'codigo_iso2', p.codigo_iso2,
        'codigo_iso3', p.codigo_iso3,
        'codigo_ioc', p.codigo_ioc,
        'nombre', p.nombre
    ) as pais,
    -- columna 'partidos_ganados' como agregado json
    (select jsonb_agg(jsonb_build_object(
        'torneo', jsonb_build_object(
            'nombre', t.nombre,
            'pais', jsonb_build_object(
                'codigo_iso2', pa.codigo_iso2,
                'codigo_iso3', pa.codigo_iso3,
                'codigo_ioc', pa.codigo_ioc,
                'nombre', pa.nombre),
            'fecha', et.fecha,
            'superficie', et.superficie,
            'tamano', et.tamano,
            'nivel', et.nivel),
        'fecha', pg.fecha,
        'ronda', pg.ronda,
        'desenlace', pg.desenlace,
        'num_partido', pg.num_partido,
        'rival', pg.perdedor,
        'sets', (select jsonb_agg(
            jsonb_build_object(
                'num_set', sp.num_set,
                'juegos_ganador', sp.juegos_ganador,
                'juegos_perdedor', sp.juegos_perdedor,
                'puntos_tiebreak_perdedor', sp.puntos_tiebreak_perdedor
            )
        )
    )
    from sets_partido sp
    where sp.torneo = pg.torneo
        and sp.fecha = pg.fecha
)
```

```
        and sp.num_partido = pg.num_partido),
    'stats', jsonb_build_object(
        'num_aces', pg.num_aces_ganador,
        'num_dob_faltas', pg.num_dob_faltas_ganador,
        'num_puntos_servidos', pg.num_ptos_servidos_ganador,
        'num_primeros_servicios', pg.num_primeros_servicios_ganador,
        'num_primeros_servicios_ganados', pg.num_primeros_servicios_ganados_ganador,
        'num_segundos_servicios_ganados', pg.num_segundos_servicios_ganados_ganador,
        'num_juegos_servidos', pg.num_juegos_servidos_ganador,
        'num_break_salvados', pg.num_break_salvados_ganador,
        'num_break_afrontados', pg.num_break_afrontados_ganador
    ),
    'stats_rival', jsonb_build_object(
        'num_aces', pg.num_aces_perdedor,
        'num_dob_faltas', pg.num_dob_faltas_perdedor,
        'num_puntos_servidos', pg.num_ptos_servidos_perdedor,
        'num_primeros_servicios', pg.num_primeros_servicios_perdedor,
        'num_primeros_servicios_ganados', pg.num_primeros_servicios_ganados_perdedor,
        'num_segundos_servicios_ganados', pg.num_segundos_servicios_ganados_perdedor,
        'num_juegos_servidos', pg.num_juegos_servidos_perdedor,
        'num_break_salvados', pg.num_break_salvados_perdedor,
        'num_break_afrontados', pg.num_break_afrontados_perdedor
    )
))
from public.partido pg
    left join public.edicion_torneo et on pg.torneo = et.torneo and pg.fecha = et.fecha
    left join public.torneo t on et.torneo = t.id
    left join public.pais pa on t.pais = pa.codigo_iso2
    where pg.ganador = j.id) as partidos_ganados,
-- columna 'partidos_perdidos' como json
(select jsonb_agg(jsonb_build_object(
    'torneo', jsonb_build_object(
        'nombre', t.nombre,
        'pais', jsonb_build_object(
            'codigo_iso2', pa.codigo_iso2,
            'codigo_iso3', pa.codigo_iso3,
            'codigo_ioc', pa.codigo_ioc,
            'nombre', pa.nombre
        ),
        'fecha', et.fecha,
        'superficie', et.superficie,
        'tamano', et.tamano,
        'nivel', et.nivel
    ),
    'fecha', pp.fecha,
    'ronda', pp.ronda,
    'desenlace', pp.desenlace,
    'num_partido', pp.num_partido,
    'rival', pp.ganador,
    'sets', (select jsonb_agg(
        jsonb_build_object(
            'num_set', sp.num_set,
            'juegos_ganador', sp.juegos_ganador,
            'juegos_perdedor', sp.juegos_perdedor,
            'puntos_tiebreak_perdedor', sp.puntos_tiebreak_perdedor
        )
    )
)
from sets_partido sp
where sp.torneo = pp.torneo
    and sp.fecha = pp.fecha
    and sp.num_partido = pp.num_partido),
    'stats', jsonb_build_object(
```

```

        'num_aces', pp.num_aces_perdedor,
        'num_dob_faltas', pp.num_dob_faltas_perdedor,
        'num_puntos_servidos', pp.num_ptos_servidos_perdedor,
        'num_primeros_servicios', pp.num_primeros_servicios_perdedor,
        'num_primeros_servicios_ganados', pp.num_primeros_servicios_ganados_perdedor,
        'num_segundos_servicios_ganados', pp.num_segundos_servicios_ganados_perdedor,
        'num_juegos_servidos', pp.num_juegos_servidos_perdedor,
        'num_break_salvados', pp.num_break_salvados_perdedor,
        'num_break_afrontados', pp.num_break_afrontados_perdedor
    ),
    'stats_rival', jsonb_build_object(
        'num_aces', pp.num_aces_ganador,
        'num_dob_faltas', pp.num_dob_faltas_ganador,
        'num_puntos_servidos', pp.num_ptos_servidos_ganador,
        'num_primeros_servicios', pp.num_primeros_servicios_ganador,
        'num_primeros_servicios_ganados', pp.num_primeros_servicios_ganados_ganador,
        'num_segundos_servicios_ganados', pp.num_segundos_servicios_ganados_ganador,
        'num_juegos_servidos', pp.num_juegos_servidos_ganador,
        'num_break_salvados', pp.num_break_salvados_ganador,
        'num_break_afrontados', pp.num_break_afrontados_ganador
    )
))
from public.partido pp
    left join public.edicion_torneo et on pp.torneo = et.torneo and pp.fecha = et.fecha
    left join public.torneo t on et.torneo = t.id
    left join public.pais pa on t.pais = pa.codigo_iso2
    where pp.perdedor = j.id) as partidos_perdidos
from public.jugador j
    left join public.pais p on j.pais = p.codigo_iso2
where j.id in (select perdedor from public.partido)
    or j.id in (select ganador from public.partido)

```

Antes de comenzar con las consultas, comentamos los aspectos clave:

- Para acceder a un valor como un objeto jsonb completo, usamos la notación `->`.
- Para acceder a un valor como un texto, usamos la notación `-»`.
- Para acceder a elementos de un array jsonb, debemos usar la función `jsonb_array_elements(nombre_array)`. Esto extrae los elementos y los asigna a registros independientes. Esto es similar a la función `unnest` que usamos en los arrays de PostgreSQL y, del mismo modo, se puede asignar un alias a cada elemento del array.

Así, por ejemplo, para acceder al nombre de un torneo, con la estructura que creamos, debemos desagregar al agregado jsonb ‘partidos ganados’ con `jsonb_array_elements(partidos_ganados)` y acceder al campo ‘torneo’ con `->` (ya que es un objeto jsonb) y al campo ‘nombre’ con `-»`. Así, si previamente asignamos a `jsonb_array_elements(partidos_ganados)` el alias `partidos(pg)`, cada elemento del array se denominará por “`pg`”, y podremos acceder al nombre del torneo como `pg -> ‘torneo’ -» ‘nombre’`.

B.1. Muestra todos los ganadores del torneo “Wimbledon” (Nombre apellido y año). Ordena el resultado por año.

El ejemplo anterior nos resulta útil para resolver esta consulta (curiosamente usa uno de los atributos que necesitamos...). Para esta consulta partimos del producto cartesiano de la tabla `tenisjson` con la función `jsonb_array_elements(partidos_ganados)` para extraer los elementos jsonb de partidos ganados. Con esto, estamos generando tuplas que no nos sirven de utilidad para la consulta, así que debemos hacer una selección en el `where` para quedarnos solo con los partidos finales de Wimbledon. Para ello, seleccionamos los partidos finales (a ronda accedemos desde “`pg`” con `-»`, ya que queremos el resultado como texto) cuyo torneo tenga nombre “Wimbledon” (aquí tenemos el ejemplo anterior). A continuación, seleccionamos el nombre y apellido del jugador y el año del torneo como atributos de texto. Aquí hay una sutileza a tener en cuenta: como estamos

obteniendo la fecha en formato texto al usar `->`, debemos convertirla a tipo date con `::date` antes de usar el `extract(year from ...)` que venimos usando durante todo el trabajo. Finalmente, ordenamos el resultado por año. La consulta se muestra a continuación, y el resultado es el esperado (figura 12)

```
select tj.jugador ->> 'nombre' as nombre,
       tj.jugador ->> 'apellido' as apellido,
       extract(year from (pg ->> 'fecha')::date) as ano
  from tenisjson tj, jsonb_array_elements(partidos_ganados) as partidos(pg)
 where pg ->> 'ronda' = 'F'
   and pg -> 'torneo' ->> 'nombre' = 'Wimbledon'
  order by ano
```

	O	A-Z nombre	A-Z apellido	123 ano
1		Michael	Stich	1.991
2		Andre	Agassi	1.992
3		Pete	Sampras	1.993
4		Pete	Sampras	1.994
5		Pete	Sampras	1.995
6		Richard	Krajicek	1.996
7		Pete	Sampras	1.997
8		Pete	Sampras	1.998
9		Pete	Sampras	1.999
10		Pete	Sampras	2.000
11		Goran	Ivanisevic	2.001
12		Lleyton	Hewitt	2.002
13		Roger	Federer	2.003
14		Roger	Federer	2.004
15		Roger	Federer	2.005
16		Roger	Federer	2.006
17		Roger	Federer	2.007
18		Rafael	Nadal	2.008
19		Roger	Federer	2.009
20		Rafael	Nadal	2.010
21		Novak	Djokovic	2.011
22		Roger	Federer	2.012
23		Andy	Murray	2.013
24		Novak	Djokovic	2.014
25		Novak	Djokovic	2.015
26		Andy	Murray	2.016
27		Roger	Federer	2.017
28		Novak	Djokovic	2.018
29		Novak	Djokovic	2.019
30		Novak	Djokovic	2.021
31		Novak	Djokovic	2.022
32		Carlos	Alcaraz	2.023

Fig. 12. Datos agregados en SQL. Tipo JSON, consulta 1.

B.2. Muestra los años en los que Roger Federer ganó algún torneo de nivel Gran Slam (G) o Master 1000 (M). Para cada año, muestra el número de torneos y lista sus nombres (ordenados por la fecha de celebración). Ordena el resultado por el año

Para esta consulta partimos del mismo punto que para la anterior: el producto cartesiano de la tabla `tenisjson` con la función `jsonb_array_elements(partidos_ganados)`. Esta vez, la selección es más específica: queremos los partidos finales (accedemos en ronda como texto) de torneos Grand Slam o Master 1000 (accedemos como texto con `'torneo' ->> 'nivel'`), donde el ganador sea Roger Federer (accedemos al nombre y apellido como texto del jugador desde `jugador`). Esto nos da una tupla para cada final ganada por Federer, por lo que para nuestro resultado debemos hacer un `group by` por año. Así, tenemos una tupla para cada año.

Finalmente, obtenemos la proyección usando el año del torneo (al cual accedemos como en la anterior consulta), el conteo de los torneos (usamos `count(distinct)` sobre el nombre de los torneos para evitar contar repetidos) y los nombres de los torneos (usamos `string_agg` para concatenar los nombres de los torneos, ordenados por fecha de celebración). Ordenamos el resultado final por año en orden ascendente. La consulta se muestra a continuación, y el resultado es el esperado (figura 13)

```

select extract(year from (pg -> 'torneo' ->> 'fecha')::date) as ano,
       count(distinct pg -> 'torneo' ->>'nombre') as numero_torneos,
       string_agg(pg -> 'torneo' ->>'nombre', ', ' order by pg -> 'torneo' ->> 'fecha') as torneos
from tenisjson tj, jsonb_array_elements(partidos_ganados) as partidos(pg)
where tj.jugador ->> 'nombre' = 'Roger'
      and tj.jugador ->> 'apellido' = 'Federer'
      and pg ->> 'ronda' = 'F'
      and pg -> 'torneo'->>'nivel' in ('G', 'M')
group by ano
order by ano

```

	123 ano	123 numero_torneos	AZ torneos
1	2.002	1	Hamburg Masters
2	2.003	1	Wimbledon
3	2.004	6	Australian Open, Indian Wells Masters, Hamburg Masters, Wimbledon, Canada Masters, US Open
4	2.005	6	Miami Masters, Indian Wells Masters, Hamburg Masters, Wimbledon, Cincinnati Masters, US Open
5	2.006	7	Australian Open, Miami Masters, Indian Wells Masters, Wimbledon, Canada Masters, US Open, Madrid Masters
6	2.007	5	Australian Open, Hamburg Masters, Wimbledon, Cincinnati Masters, US Open
7	2.008	1	US Open
8	2.009	4	Madrid Masters, Roland Garros, Wimbledon, Cincinnati Masters
9	2.010	2	Australian Open, Cincinnati Masters
10	2.011	1	Paris Masters
11	2.012	4	Indian Wells Masters, Madrid Masters, Wimbledon, Cincinnati Masters
12	2.014	2	Cincinnati Masters, Shanghai Masters
13	2.015	1	Cincinnati Masters
14	2.017	5	Australian Open, Indian Wells Masters, Miami Masters, Wimbledon, Shanghai Masters
15	2.018	1	Australian Open
16	2.019	1	Miami Masters

Fig. 13. Datos agregados en SQL. Tipo JSON, consulta 2.

B.3. Muestra los partidos de semifinales (ronda='SF') y final (ronda = 'F') del torneo de "Roland Garros" del 2018. Para cada partido muestra la ronda, el tipo de desenlace, el nombre y apellidos del ganador y el nombre y apellidos del perdedor y el resultado con el número de juegos del ganador y del perdedor en cada set, y opcionalmente en paréntesis el número de juegos del perdedor en el tie break

Para esta consulta, partimos del producto cartesiano de la tabla tenisjson con las funciones jsonb_array_elements(partidos_ganados) y jsonb_array_elements(pg -> 'sets') para extraer los elementos jsonb de partidos ganados y extraer los elementos del objeto jsonb 'sets', que designaremos por un alias. Esta vez estamos considerando dos veces la tabla tenisjson, ya que necesitamos estadísticas del ganador y del perdedor. Con esto tenemos de nuevo un exceso de filas debido al producto cartesiano, por lo que seleccionamos los partidos de semifinales y finales de Roland Garros 2018 (accedemos a la ronda, la fecha y al nombre del torneo como texto). Además, seleccionamos solo las tuplas donde el id del jugador rival coincide con el id del jugador de la segunda tabla tenisjson que consideramos. Con todo esto, tenemos una tupla para cada set de cada partido de semifinales y finales de Roland Garros 2018. Como esto no es lo que queremos, agrupamos por ronda, desenlace y nombre y apellidos del ganador y del perdedor.

Con todo esto tenemos simplemente una tupla por cada partido (3, 2 para semifinales y una para final). Para la proyección, seleccionamos la ronda, el desenlace, el nombre y apellidos del ganador y del perdedor como texto (nótese que al concatenar nombre y apellidos debemos hacer la conversión explícita de los atributos a texto) y el resultado, que se construye concatenando el número de juegos del ganador y del perdedor en cada set, y opcionalmente el número de juegos del perdedor en el tie break (si este existe). La consulta se muestra a continuación, y el resultado es el esperado (figura 14)

```

select pg ->> 'ronda' as ronda, pg ->> 'desenlace' as desenlace,
       (tj.jugador ->> 'nombre')::text || ' ' || (tj.jugador ->> 'apellido')::text as ganador,
       (tjr.jugador ->> 'nombre')::text || ' ' || (tjr.jugador ->> 'apellido')::text as perdedor,
       string_agg((s ->> 'juegos_ganador')::text || '-' || (s ->> 'juegos_perdedor')::text ||
                  case when s ->> 'puntos.tiebreak_perdedor' is not null

```

```

        then '(' || (s -> 'puntos_tiebreak_perdedor')::text || ')'
        else '' end, ', ' order by s -> 'num_set') as resultado
from tenisjson tj, jsonb_array_elements(tj.partidos_ganados) as partidos(pg),
      jsonb_array_elements(pg -> 'sets') as setss(s), tenisjson tjr
where pg ->'torneo' -> 'nombre' = 'Roland Garros'
  and extract(year from (pg -> 'fecha')::date) = 2018
  and pg -> 'ronda' in ('SF', 'F')
  and tjr.jugador->>'id' = pg ->> 'rival'
group by pg ->> 'ronda', pg ->> 'desenlace',
         tj.jugador ->> 'nombre', tj.jugador ->> 'apellido',
         tjr.jugador ->> 'nombre', tjr.jugador ->> 'apellido'

```

	A-Z ronda	A-Z desenlace	A-Z ganador	A-Z perdedor	A-Z resultado
1	F	N	Rafael Nadal	Dominic Thiem	6-4, 6-3, 6-2
2	SF	N	Dominic Thiem	Marco Cecchinato	7-5, 7-6(10), 6-1
3	SF	N	Rafael Nadal	Juan Martin del Potro	6-4, 6-1, 6-2

Fig. 14. Datos agregados en SQL. Tipo JSON, consulta 3.

B.4. Muestra la lista de jugadores españoles (ES) que ganaron algún torneo de nivel Gran Slam (G). Para cada jugador muestra los siguientes datos resumen de todos sus partidos: número de partidos jugados, porcentaje de victorias, porcentaje de aces, porcentaje de dobles faltas, porcentaje de servicios ganados, porcentaje de restos ganados, porcentaje de break points salvados (de los sufridos en contra), porcentaje de break points ganados (de los provocados a favor)

Como anteriormente, se dividirá esta consulta en dos: una primera parte que selecciona los jugadores españoles que ganaron algún torneo de nivel Grand Slam y una segunda parte que calcula los datos resumen de todos sus partidos.

Para la primera parte, partimos del producto cartesiano de la tabla tenisjson con la función jsonb_array_elements(partidos_ganados) para extraer los elementos jsonb de partidos ganados. Seleccionamos los jugadores españoles (accedemos al código ISO2 del país como texto) que ganaron algún torneo de nivel Gran Slam (accedemos a la ronda y al nivel del torneo como texto). Con esto, tenemos una tupla para cada partido final ganado por un jugador español en un torneo de nivel Grand Slam. En la proyección, nos quedamos con el id y el nombre y apellido del jugador, usando distinct para evitar duplicados.

Para la segunda parte, partimos del producto cartesiano de la tabla de jugadores españoles con tenisjson y con la función jsonb_array_elements(tj.partidos_ganados) para extraer los elementos jsonb de partidos ganados. Seleccionamos en el where los partidos en los que los jugadores españoles antes diferenciados aparecen, tanto como ganadores como rivales. Con esto, tenemos una tupla para cada partido jugado por un jugador español (que haya ganado un Grand Slam). Antes de hacer la proyección, agrupamos por jugador para tener una tupla para cada jugador.

La proyección para obtener las estadísticas del jugador es conceptualmente igual a lo que ya hicimos en esta consulta para el modelo relacional y para el de tipos compuestos. Solo tenemos que tener en cuenta que los valores que queremos están en el objeto jsonb 'stats' o en 'stats_rival' de cada partido (es decir, accedemos con ->). Una vez dentro de estos, accedemos con normalidad a las estadísticas que queremos en formato texto con ->>. La consulta se muestra a continuación, y el resultado es el esperado (figura 15)

```

with jugadores_espanoles_ganadores as (
  select distinct (tj.jugador ->> 'id')::integer as id_jugador,
         (tj.jugador ->> 'nombre')::text || ' ' || (tj.jugador ->> 'apellido')::text as jugador
  from tenisjson tj, jsonb_array_elements(partidos_ganados) as partidos(pg)
  where tj.pais ->> 'codigo_iso2' = 'ES'
    and pg ->> 'ronda' = 'F'

```

```

        and pg -> 'torneo' ->> 'nivel' = 'G'
    )

select jeg.jugador,
       count(*) as partidos,
       round(100.0 * count(case when pg ->> 'rival' = jeg.id_jugador::text then null else 1 end)::numeric /
             count(*), 1) as pcje_victorias,
       round(100.0 * sum(case when pg ->> 'rival' = jeg.id_jugador::text
                           then (pg -> 'stats_rival' ->> 'num_aces')::numeric
                           else (pg -> 'stats' ->> 'num_aces')::numeric end) /
             nullif(sum(case when pg ->> 'rival' = jeg.id_jugador::text
                           then (pg -> 'stats_rival' ->> 'num_puntos_servidos')::numeric
                           else (pg->'stats'->>'num_puntos_servidos')::numeric end), 0), 1) as pcje_aces,
       round(100.0 * sum(case when pg ->> 'rival' = jeg.id_jugador::text
                           then (pg -> 'stats_rival' ->> 'num_dob_faltas')::numeric
                           else (pg -> 'stats' ->> 'num_dob_faltas')::numeric end) /
             nullif(sum(case when pg ->> 'rival' = jeg.id_jugador::text
                           then (pg -> 'stats_rival' ->> 'num_puntos_servidos')::numeric
                           else (pg -> 'stats' ->> 'num_puntos_servidos')::numeric end), 0), 1)
                           as pcje_dobles_faltas,
       round(100.0 * sum(case when pg ->> 'rival' = jeg.id_jugador::text
                           then (pg -> 'stats_rival' ->> 'num_primeros_servicios_ganados')::numeric +
                               (pg -> 'stats_rival' ->> 'num_segundos_servicios_ganados')::numeric
                           else (pg -> 'stats' ->> 'num_primeros_servicios_ganados')::numeric +
                               (pg -> 'stats' ->> 'num_segundos_servicios_ganados')::numeric end) /
             nullif(sum(case when pg ->> 'rival' = jeg.id_jugador::text
                           then (pg -> 'stats_rival' ->> 'num_puntos_servidos')::numeric
                           else (pg -> 'stats' ->> 'num_puntos_servidos')::numeric end), 0), 1)
                           as pcje_servicios_ganados,
       round(100.0 * sum(case when pg ->> 'rival' = jeg.id_jugador::text
                           then (pg -> 'stats' ->> 'num_puntos_servidos')::numeric -
                               (pg -> 'stats' ->> 'num_primeros_servicios_ganados')::numeric -
                               (pg -> 'stats' ->> 'num_segundos_servicios_ganados')::numeric
                           else (pg -> 'stats_rival' ->> 'num_puntos_servidos')::numeric -
                               (pg -> 'stats_rival' ->> 'num_primeros_servicios_ganados')::numeric -
                               (pg -> 'stats_rival' ->> 'num_segundos_servicios_ganados')::numeric end) /
             nullif(sum(case when pg ->> 'rival' = jeg.id_jugador::text
                           then (pg -> 'stats' ->> 'num_puntos_servidos')::numeric
                           else (pg -> 'stats_rival' ->> 'num_puntos_servidos')::numeric end), 0), 1)
                           as pcje_restos_ganados,
       round(100.0 * sum(case when pg ->> 'rival' = jeg.id_jugador::text
                           then (pg->'stats_rival'->>'num_break_salvados')::numeric
                           else (pg -> 'stats' ->> 'num_break_salvados')::numeric end) /
             nullif(sum(case when pg ->> 'rival' = jeg.id_jugador::text
                           then (pg -> 'stats_rival' ->> 'num_break_afrontados')::numeric
                           else (pg -> 'stats' ->> 'num_break_afrontados')::numeric end), 0), 1)
                           as pcje_breaks_salvados,
       round(100.0 * sum(case when pg ->> 'rival' = jeg.id_jugador::text
                           then (pg -> 'stats' ->> 'num_break_afrontados')::numeric -
                               (pg -> 'stats' ->> 'num_break_salvados')::numeric
                           else (pg -> 'stats_rival' ->> 'num_break_afrontados')::numeric -
                               (pg -> 'stats_rival' ->> 'num_break_salvados')::numeric end) /
             nullif(sum(case when pg ->> 'rival' = jeg.id_jugador::text
                           then (pg -> 'stats' ->> 'num_break_afrontados')::numeric
                           else (pg -> 'stats_rival' ->> 'num_break_afrontados')::numeric end), 0), 1)
                           as pcje_breaks_ganados
from jugadores_espanoles_ganadores jeg, tenisjson tj,
     jsonb_array_elements(tj.partidos_ganados) as partidos(pg)
where (tj.jugador ->> 'id')::integer = jeg.id_jugador
      or pg ->> 'rival' = jeg.id_jugador::text
group by jeg.jugador

```

	A-Z jugad	123 parti	123 pcje_vict	123 pcje_ace	123 pcje_dobles_fal	123 pcje_servicios_gana	123 pcje_restos_gan	123 pcje_breaks_salv	123 pcje_breaks_ganai
1	Albert Costa	648	58,5	4,5	2,6	62,5	39,5	60,3	41
2	Rafael Nadal	1.276	82,3	4,3	2,3	67,3	42,4	66,1	44,9
3	Carlos Moya	873	63,9	6,4	3,2	63,4	39	62,8	40,5
4	Carlos Alcaraz	183	79,8	4,6	3	65,4	41,6	63,8	41,1
5	Juan Carlos Ferrero	717	64,3	5,1	2,8	63,7	39,7	62,5	40,1
6	Sergi Bruguera	601	62,9	5,2	2,5	61,9	41,6	59,2	44,9

Fig. 15. Datos agregados en SQL. Tipo JSON, consulta 4.

B.5. Lista los jugadores que fueron derrotados (en algún partido del 2018) por el rival de Rafael Nadal de la primera ronda (R128) de Roland Garros de 2018

Esta consulta la dividiremos en dos subconsultas: una que selecciona el rival de Rafael Nadal en la primera ronda de Roland Garros 2018 y otra que selecciona los jugadores que fueron derrotados por este rival en algún partido del 2018.

Para la primera parte, partimos del producto cartesiano de la tabla tenisjson (por duplicado, para discernir entre ganador y perdedor) con la función jsonb_array_elements(partidos_ganados) para extraer los elementos jsonb de partidos ganados. Seleccionamos solo las tuplas que nos interesan:

- El nombre torneo debe ser Roland Garros.
- La ronda debe ser R128.
- El año de la fecha debe ser del 2018.
- El nombre y apellido de un jugador debe ser Rafael Nadal, respectivamente. Como no sabemos si ganó o perdió, debemos considerar ambos casos: si es ganador (buscando en la tabla tenisjson tj), o si es perdedor (buscando en la tabla tenisjson con el alias tjr).

Con esto, tenemos un única tupla con el rival de Rafael Nadal en la primera ronda de Roland Garros 2018. Proyectamos para obtener el id del jugador rival y el nombre y apellido del jugador, considerando que, cuando el ganador sea Nadal, debemos obtener el rival desde tjr, mientras que en caso contrario, debemos obtener al rival desde tj. No explicamos como acceder a los atributos concretos ya que ya se ha hecho con anterioridad para estos parámetros concretos. El resultado lo guardamos en una tabla temporal rival_nadal.

Para la segunda parte, partimos del producto cartesiano de la tabla anterior con tenisjson y con la función jsonb_array_elements(tj.partidos_perdidos) para extraer los elementos jsonb de partidos perdidos. En el where hacemos una selección de las tuplas para las cuales el id del rival (es decir, el ganador) figure como el del rival de Nadal que encontramos antes. Además, seleccionamos solo los partidos del 2018. Con esto, tenemos una tupla para cada partido ganado por este jugador en el año 2018 (no hay repetidos). En la proyección, seleccionamos simplemente el nombre y apellido del jugador (que concatenamos al seleccionar como texto con -») y el código ISO2 del país. La consulta se muestra a continuación, y el resultado es el esperado (figura 16)

```
with rival_nadal as (
    select case when tj.jugador ->> 'nombre' = 'Rafael'
                then (pg ->> 'rival')::integer
                else (tj.jugador ->> 'id')::integer end as id_jugador,
    case when tj.jugador ->> 'nombre' = 'Rafael'
                then (tjr.jugador ->> 'nombre')::text || ' ' || (tjr.jugador ->> 'apellido')::text
                else (tj.jugador ->> 'nombre')::text || ' ' || (tj.jugador ->> 'apellido')::text
                end as jugador
    from tenisjson tj, jsonb_array_elements(tj.partidos_ganados) as partidos(pg), tenisjson tjr
    where pg -> 'torneo' ->> 'nombre' = 'Roland Garros'
        and pg ->> 'ronda' = 'R128'
        and extract(year from (pg ->> 'fecha')::date) = 2018
        and tjr.jugador->>'id' = pg ->> 'rival'
        and ((tj.jugador ->> 'nombre' = 'Rafael' and tj.jugador ->> 'apellido' = 'Nadal'))
```

```
        or (tjr.jugador ->> 'nombre' = 'Rafael' and tjr.jugador ->> 'apellido' = 'Nadal'))  
    )  
  
select tj.jugador->>'nombre' || ' ' || (tj.jugador->>'apellido')::text as jugador,  
      tj.pais->>'codigo_iso2' as pais  
from rival_nadal rn, tenisjson tj, jsonb_array_elements(tj.partidos_perdidos) as partidos(pg)  
where rn.id_jugador = (pg->>'rival')::integer  
      and extract(year from (pg->>'fecha')::date) = 2018
```

	A-Z jugador	A-Z pais
1	Pablo Cuevas	UY
2	Denis Istomin	UZ
3	Federico Delbonis	AR
4	Diego Schwartzman	AR
5	Roberto Carballes Baena	ES
6	Joaõ Domingues	PT

Fig. 16. Datos agregados en SQL. Tipo JSON, consulta 5.

3. BASES DE DATOS DISTRIBUIDAS CON SQL (CITUS DATA)

Para esta parte de la práctica decidimos crear el clúster sobre máquinas reales y no sobre la máquina virtual, para profundizar en detalles acerca de cómo configurar PostgreSQL en un entorno real, tanto a la hora de permitir conexiones desde el exterior como habilitar la autenticación y la conexión entre nodos de un clúster.

Preparación

Las máquinas que alquilamos en la nube de OVH son servidores con 1 vCore, 2 GB de RAM y 20 GB de almacenamiento SSD corriendo Ubuntu 22.04, su oferta más económica. Para la conexión a estas máquinas optamos por la autenticación mediante clave pública/privada. Para ello debemos generar un par de claves dentro del directorio `$HOME/.ssh` de nuestra máquina local, esto lo hacemos mediante el comando:

```
ssh-keygen -b 4096 -f ~/.ssh/keys/ovh
```

Esto nos genera la clave privada que guardaremos en nuestra máquina y una clave pública que usaremos a la hora de configurar las máquinas en el portal de OVH. Para facilitar el trabajo contra estas máquinas, definimos en nuestra máquina local en la ruta `$HOME/.ssh/config` el siguiente archivo de configuración:

```
Host ovh*
  User      ubuntu
  IdentityFile ~/.ssh/keys/ovh

Host ovh1
  HostName 51.210.241.66

Host ovh2
  HostName 51.210.241.239

Host ovh3
  HostName 51.210.241.207
```

Gracias a este, podremos conectarnos mediante `ssh` a las máquinas sin tener que escribir el usuario o la dirección IP pública de las máquinas, ya que podremos usar `ssh ovh1`, por ejemplo. Finalmente, dentro de cada máquina, cambiamos su nombre mediante el siguiente comando:

```
sudo hostnamectl hostname <nuevo_nombre>
```

Instalación

Para instalar PostgreSQL y su extensión Citus seguimos el [guion](#) de los propios creadores de Citus. Comenzamos creando el script bash `all_nodes_terraform.sh` que se deberá ejecutar en todas las máquinas:

```
#!/bin/bash

# ejecutar desde local: ssh user@host 'bash -s' < <path_to_this_file>

POSTGRES_PASSWORD="***"

# Añadir el repositorio de Citus al empaquetador de Ubuntu
curl https://install.citusdata.com/community/deb.sh | sudo bash

# Instalar Postgres y Citus
sudo apt-get -y install postgresql-16-citus-12.1

# Precargar la extensión Citus
sudo pg_conftool 16 main set shared_preload_libraries citus

# Iniciar el servicio de Postgres y habilitarlo para arrancar al iniciar
```

```

sudo systemctl start postgresql
sudo systemctl enable postgresql

# Modificamos la contraseña del usuario postgres
sudo -i -u postgres psql -c "ALTER USER postgres WITH PASSWORD '${POSTGRES_PASSWORD}';"

# Modificamos el archivo /etc/postgres/16/main/postgresql.conf mediante comando
sudo pg_conftool 16 main set listen_addresses '*'

# Al archivo pg_hba.conf de nuestra versión instalada añadimos la siguiente linea
# que otorga acceso completo a usuarios autenticados
echo -e "\n# Allow access from all IP addresses\nhost    all      all      0.0.0.0/0      md5"
| sudo tee -a /etc/postgresql/16/main/pg_hba.conf

# Reiniciamos el servicio de postgres
sudo systemctl restart postgresql

# Los diferentes nodos del clúster deben de poder comunicarse entre sí
# Para ello, creamos el archivo .pgpass en el directorio $HOME del usuario postgres
# Postgres leerá de ese archivo las contraseñas de distintas conexiones
# Usamos * para indicarle que use la misma contraseña para todas las conexiones, desde cada nodo a los otros 2
# https://docs.citusdata.com/en/v8.3/admin_guide/cluster_management.html#increasing-worker-security
sudo bash -c "echo '*:*:*:$POSTGRES_PASSWORD' > /var/lib/postgresql/.pgpass"

# Cambiamos permisos de lectura y escritura únicamente para el usuario postgres
sudo chown postgres:postgres /var/lib/postgresql/.pgpass
sudo chmod 600 /var/lib/postgresql/.pgpass

```

Con este script, instalamos PostgreSQL y Citus, configuramos PostgreSQL para escuchar tráfico en todas sus interfaces de red para poder enviar solicitudes desde internet y añadimos un nuevo tipo de conexión aceptada, desde cualquier dirección IP, pero que requiere de autenticación por contraseña.

Finalmente, acorde a la [guía](#) de Citus acerca de cómo manejar la conexión entre nodos de un clúster, definimos el archivo .pgpass en la carpeta \$HOME del usuario postgres, creado cuando instalamos PostgreSQL. Este archivo es necesario para que los nodos se puedan comunicar entre sí, ya que, al no poder configurar las conexiones entre nodos en la misma red por limitaciones de nuestro proveedor cloud, estas conexiones también deberán ser autenticadas.

Para ejecutar este script en cada máquina, desde nuestra máquina local podemos hacer:

```
ssh username@host 'bash -s' <path_script.sh>
```

Configurar Citus

Una vez que tenemos PostgreSQL y Citus instalados y configurados a nivel de sistema, debemos configurar Citus. Las extensiones en PostgreSQL se asocian a una base de datos, por lo que podemos tener una base de datos llamada citus donde instalaremos la extensión mientras que el resto de bases de datos se comportarán como bases de datos PostgreSQL estándar. Para esto, creamos otro script, all_nodes_citus.sh, que se ejecutará en todas las máquinas del clúster:

```

# Eliminar extensión y base de datos citus
sudo -i -u postgres psql -d citus -c "DROP EXTENSION citus CASCADE;"
sudo -i -u postgres psql -c "DROP DATABASE citus;"

# Recrear base de datos citus
sudo -i -u postgres psql -c "CREATE DATABASE citus;"
```

```
# Añadir la extensión citus a la base de datos citus
sudo -i -u postgres psql -d citus -c "CREATE EXTENSION citus;"

# Verificamos que citus se ha instalado correctamente
sudo -i -u postgres psql -d citus -c "SELECT * FROM citus_version();"
```

Finalmente, escogeremos una sola máquina para actuar como nodo coordinador, en nuestro caso, ovh1. En ella, deberemos ejecutar los comandos SQL para proclamar el nodo coordinador y añadir los demás nodos worker. Estos comandos SQL los tenemos recogidos en otro script, `coordinador_citus.sh`, que solo se ejecutara en el nodo coordinador:

```
# Registrar la dirección del nodo coordinador
sudo -i -u postgres psql -d citus -c "SELECT citus_set_coordinator_host('51.210.241.66', 5432);"

# Añadir los nodos worker
sudo -i -u postgres psql -d citus -c "SELECT * from citus_add_node('51.210.241.239', 5432);"
sudo -i -u postgres psql -d citus -c "SELECT * from citus_add_node('51.210.241.207', 5432);"

# Verificar clúster
sudo -i -u postgres psql -d citus -c "SELECT * FROM citus_get_active_worker_nodes();"
```

Después de ejecutar los scripts, obtenemos la salida mostrada en la figura 17, donde podemos observar que el clúster se ha creado correctamente.

```
citus_set_coordinator_host
-----
(1 row)

citus_add_node
-----
2
(1 row)

citus_add_node
-----
3
(1 row)

node_name      | node_port
-----+-----
51.210.241.239 |      5432
51.210.241.207 |      5432
(2 rows)
```

Fig. 17. Citus, cluster montado.

Trabajar contra el clúster

En primer lugar se nos pide que, suponiendo que la base de datos crecerá al incorporar nuevos partidos de nuevas ediciones de torneos, distribuyamos los datos. La tabla de partido es la elección obvia para ser distribuida, particionando por el atributo `torneo`, lo cual nos garantiza que toda la información relacionada con los partidos en un torneo esté localizada en un único nodo. Esto minimiza las comunicaciones entre nodos y mejora el rendimiento de las consultas relacionadas con un torneo específico. Por extensión, la tabla `set_partidos` irá co-localizada con `partido`, así como la tabla `edicion_torneo`. Ambas también particionadas por el atributo `torneo`.

Las tablas pais y torneo las pondremos como tablas de referencia, esto es, tablas que son replicadas entre todos los nodos worker ya que son tablas pequeñas que no cambian con frecuencia.

Hacemos un inciso para mencionar que los nodos que almacenan datos son, principalmente, los workers, encargados de almacenar tablas distribuidas y de referencia. El nodo coordinador es el punto de entrada al clúster y se encarga de organizar las consultas y lanzarlas a los workers adecuados. No obstante, existen un tipo de tablas que sí pueden guardarse en el nodo coordinador: las tablas locales. Estas son tablas de PostgreSQL *vanilla* que no han sido declaradas como distribuidas o de referencia.

Una vez que tenemos definido cómo vamos a distribuir los datos, cargamos el esquema de la primera parte del trabajo y le añadimos el siguiente código SQL para definir cómo manejaremos las tablas dentro de Citus:

```
-- Tablas de referencia (copiadas a todos los worker)
SELECT create_reference_table('pais');
SELECT create_reference_table('torneo');
SELECT create_reference_table('jugador');

-- Tablas distribuidas
SELECT create_distributed_table('partido', 'torneo');
SELECT create_distributed_table('edicion_torneo', 'torneo', colocate_with => 'partido');
SELECT create_distributed_table('sets_partido', 'torneo', colocate_with => 'partido');
SELECT create_distributed_table('ranking', 'jugador');
```

```
psql:3.sql-citus/1.load/schema_citus.sql:102: ERROR: cannot create foreign key constraint since relations are not colocated or not referencing a reference table
DETAIL: A distributed table can only have foreign keys if it is referencing another colocated hash distributed table or a reference table
psql:3.sql-citus/1.load/schema_citus.sql:103: ERROR: cannot distribute relation
DETAIL: Currently, colocate_with option is not supported with append / range distributed tables and local tables added to metadata.
psql:3.sql-citus/1.load/schema_citus.sql:104: ERROR: cannot distribute relation
DETAIL: Currently, colocate_with option is not supported with append / range distributed tables and local tables added to metadata.
create_distributed_table
-----
(1 row)

psql:3.sql-citus/1.load/schema_citus.sql:113: ERROR: constraint "partido_torneo_fecha_fkey" for relation "partido" already exists
psql:3.sql-citus/1.load/schema_citus.sql:119: ERROR: constraint "sets_partido_torneo_fecha_num_partido_fkey" for relation "sets_partido" already exists
   table_name | citus_table_type | distribution_column | colocation_id | table_size | shard_count | table_owner | access_method
-----+-----+-----+-----+-----+-----+-----+-----+-----+
  edicion_torneo | local      | <none>           | 0 | 8192 bytes | 1 | postgres | heap
  jugador         | reference   | <none>           | 1 | 24 kB     | 1 | postgres | heap
  pais            | reference   | <none>           | 1 | 96 kB     | 1 | postgres | heap
  partido          | local      | <none>           | 0 | 8192 bytes | 1 | postgres | heap
  ranking          | distributed | jugador           | 3 | 256 kB    | 32 | postgres | heap
  sets_partido    | local      | <none>           | 0 | 8192 bytes | 1 | postgres | heap
  torneo           | reference   | <none>           | 1 | 24 kB     | 1 | postgres | heap
(7 rows)
```

Fig. 18. Citus, error en claves foráneas.

No obstante, de esta forma obtenemos el error mostrado en la figura 18, en el que Citus nos indica que no puede añadir claves foráneas a la tabla distribuida de partido, porque las tablas distribuidas solo pueden tener claves foráneas que apunten a tablas de referencia o a una tabla distribuida co-localizada con la tabla que declara la clave foránea.

Como podemos observar, las tablas distribuidas relativas a los torneos no se han distribuido y se han quedado como tablas locales, su valor por defecto; este es el motivo por el cual no hemos podido añadir las claves foráneas. Para solventar esto y mantener las claves foráneas en nuestro esquema, debemos modificar el código de la siguiente manera:

```
DROP TABLE IF EXISTS pais CASCADE;
DROP TABLE IF EXISTS jugador CASCADE;
DROP TABLE IF EXISTS torneo CASCADE;
DROP TABLE IF EXISTS edicion_torneo CASCADE;
DROP TABLE IF EXISTS partido CASCADE;
DROP TABLE IF EXISTS sets_partido CASCADE;
```

```

DROP TABLE IF EXISTS ranking CASCADE;

CREATE TABLE pais (...);

CREATE TABLE jugador (...);

CREATE TABLE torneo (...);

CREATE TABLE edicion_torneo (...);

CREATE TABLE partido (...);

CREATE TABLE sets_partido (...);

CREATE TABLE ranking (...);

-- Tablas de referencia (copiadas a todos los worker)
SELECT create_reference_table('pais');
SELECT create_reference_table('torneo');
SELECT create_reference_table('jugador');

-- Tablas distribuidas
SELECT create_distributed_table('partido', 'torneo');
SELECT create_distributed_table('edicion_torneo', 'torneo', colocate_with => 'partido');
SELECT create_distributed_table('sets_partido', 'torneo', colocate_with => 'partido');
SELECT create_distributed_table('ranking', 'jugador');

-- Después de distribuir las tablas, podemos crear las claves foráneas de tablas distribuidas, no antes

-- Clave foránea para partido
ALTER TABLE partido
ADD CONSTRAINT partido_torneo_fecha_fkey
FOREIGN KEY (torneo, fecha)
REFERENCES edicion_torneo (torneo, fecha);

-- Clave foránea para sets_partido
ALTER TABLE sets_partido
ADD CONSTRAINT sets_partido_torneo_fecha_num_partido_fkey
FOREIGN KEY (torneo, fecha, num_partido)
REFERENCES partido (torneo, fecha, num_partido);

```

En este código definimos las claves foráneas una vez que las tablas se han distribuido correctamente. De esta forma, no tenemos ningún fallo. Podemos observar cómo quedan nuestras tablas (figura 19) mediante la instrucción:

```
SELECT * FROM citus_tables;
```

table_name	citus_table_type	distribution_column	colocation_id	table_size	shard_count	table_owner	access_method
edicion_torneo	distributed	torneo	2	800 kB	32	postgres	heap
jugador	reference	<none>	1	15 MB	1	postgres	heap
pais	reference	<none>	1	312 kB	1	postgres	heap
partido	distributed	torneo	2	21 MB	32	postgres	heap
ranking	distributed	jugador	2	51 MB	32	postgres	heap
sets_partido	distributed	torneo	2	27 MB	32	postgres	heap
torneo	reference	<none>	1	168 kB	1	postgres	heap
(7 rows)							

Fig. 19. Distribución final de tablas en Citus.

En la figura 19 podemos observar cómo nuestras elecciones para las tablas distribuidas parecen acertadas si comparamos el tamaño en megas de las tablas distribuidas y en kilobytes de las de referencia. La única

excepción a esto es la tabla jugador, que hemos dejado como tabla de referencia ya que, si la distribuyéramos, no podríamos co-localizarla con partido, y las consultas que unan jugador y partido (la mayoría), requerirían de transferencias entre nodos. Así, a pesar de que jugador es una tabla que previsiblemente aumentará a medida que nuevos jugadores entren en el circuito, es más eficiente no distribuirla y dejarla como tabla de referencia, ya que presuponemos que la tabla de partido crecerá más rápido que la de jugadores porque, por cada nuevo jugador, estos jugarán uno o más partidos.

Lanzar consultas al clúster

Para enviar consultas al clúster desde nuestra máquina local podemos usar un cliente gráfico como DBeaver, ya que Citus es transparente a los clientes de PostgreSQL, dejándose ver como una base de datos PostgreSQL estándar.

También podemos utilizar psql, el cliente de PostgreSQL para la terminal. Para mejorar la ergonomía al usar esta herramienta, podemos definir dos ficheros en el directorio \$HOME de nuestra máquina local: .pgpass, que al igual que hicimos en los servidores, almacenará conjuntos de servidores:puerto:base_datos:usuario:contraseña, y el archivo .pg_service.conf, que nos permitirá poner un alias a las conexiones de PostgreSQL:

```
[tenis]
host=51.210.241.66
port=5432
dbname=tenis
user=postgres

[citus]
host=51.210.241.66
port=5432
dbname=citus
user=postgres

[citus-worker-1]
host=51.210.241.239
port=5432
dbname=citus
user=postgres

[citus-worker-2]
host=51.210.241.207
port=5432
dbname=citus
user=postgres
```

Así, desde nuestra máquina local, podemos lanzar consultas SQL definidas en archivos .sql de forma muy cómoda con el siguiente comando:

```
psql service=[service_name] -f <path_to_sql_file>
```

No comentaremos la semántica del código de las propias consultas, al ser idéntico al descrito en el primer apartado (modelo relacional). Los resultados de las mismas también coinciden con los mostrados en ese apartado, y pueden verse en las figuras 20, 21, 22, 23 y 24.

.1. Muestra todos los ganadores del torneo “Wimbledon” (Nombre apellido y año). Ordena el resultado por año.

```
select j.nombre, j.apellido, extract(year from p.fecha) as año
from jugador j, partido p, torneo t
where j.id = p.ganador
and t.id = p.torneo
```

```
and t.nombre = 'Wimbledon'
and p.ronda = 'F'
order by ano
```

nombre	apellido	ano
Michael	Stich	1991
Andre	Agassi	1992
Pete	Sampras	1993
Pete	Sampras	1994
Pete	Sampras	1995
Richard	Krajicek	1996
Pete	Sampras	1997
Pete	Sampras	1998
Pete	Sampras	1999
Pete	Sampras	2000
Goran	Ivanisevic	2001
Lleyton	Hewitt	2002
Roger	Federer	2003
Roger	Federer	2004
Roger	Federer	2005
Roger	Federer	2006
Roger	Federer	2007
Rafael	Nadal	2008
Roger	Federer	2009
Rafael	Nadal	2010
Novak	Djokovic	2011
Roger	Federer	2012
Andy	Murray	2013
Novak	Djokovic	2014
Novak	Djokovic	2015
Andy	Murray	2016
Roger	Federer	2017
Novak	Djokovic	2018
Novak	Djokovic	2019
Novak	Djokovic	2021
Novak	Djokovic	2022
Carlos	Alcaraz	2023

Fig. 20. Bases de datos distribuidas. Citus, consulta 1.

.2. Muestra los años en los que Roger Federer ganó algún torneo de nivel Gran Slam (G) o Master 1000 (M). Para cada año, muestra el número de torneos y lista sus nombres (ordenados por la fecha de celebración). Ordena el resultado por el año

```
select extract(year from et.fecha) as ano, count(distinct t.id) as numero_torneos,
       string_agg(t.nombre, ', ' order by et.fecha) as torneos
from jugador j, partido p, torneo t, edicion_torneo et
where j.id = p.ganador
  and t.id = et.torneo
  and p.torneo = t.id
  and p.fecha = et.fecha
  and p.ronda = 'F'
  and j.nombre = 'Roger'
  and j.apellido = 'Federer'
  and et.nivel in ('G', 'M')
group by ano
order by ano
```

```
miguel@macmini Trabajo % psql service=citus -f 1.sql/q2.sql
ano | torneos |                                            torneos
-----+-----+
2002 | 1 | Hamburg Masters
2003 | 1 | Wimbledon
2004 | 6 | Australian Open, Indian Wells Masters, Hamburg Masters, Wimbledon, Canada Masters, US Open
2005 | 6 | Miami Masters, Indian Wells Masters, Hamburg Masters, Wimbledon, Cincinnati Masters, US Open
2006 | 7 | Australian Open, Indian Wells Masters, Miami Masters, Wimbledon, US Open, Canada Masters, Madrid Masters
2007 | 5 | Australian Open, Hamburg Masters, Wimbledon, Cincinnati Masters, US Open
2008 | 1 | US Open
2009 | 4 | Madrid Masters, Roland Garros, Wimbledon, Cincinnati Masters
2010 | 2 | Australian Open, Cincinnati Masters
2011 | 1 | Paris Masters
2012 | 4 | Indian Wells Masters, Madrid Masters, Wimbledon, Cincinnati Masters
2014 | 2 | Cincinnati Masters, Shanghai Masters
2015 | 1 | Cincinnati Masters
2017 | 5 | Australian Open, Indian Wells Masters, Miami Masters, Wimbledon, Shanghai Masters
2018 | 1 | Australian Open
2019 | 1 | Miami Masters
(16 rows)
```

Fig. 21. Bases de datos distribuidas. Citus, consulta 2.

- .3. Muestra los partidos de semifinales (ronda='SF') y final (ronda = 'F') del torneo de "Roland Garros" del 2018. Para cada partido muestra la ronda, el tipo de desenlace, el nombre y apellidos del ganador y el nombre y apellidos del perdedor y el resultado con el número de juegos del ganador y del perdedor en cada set, y opcionalmente en paréntesis el número de juegos del perdedor en el tie break

```
select p.ronda, p.desenlace, jg.nombre || ' ' || jg.apellido AS ganador,
       jp.nombre || ' ' || jp.apellido AS perdedor,
       STRING_AGG(sp.juegos_ganador || '-' || sp.juegos_perdedor || ''
                  case
                    when sp.puntos.tiebreak_perdedor is not null then
                      '(' || sp.puntos.tiebreak_perdedor || ')'
                    else
                      ''
                  end, ', ' order by sp.num_set) as resultado
from partido p, jugador jg, jugador jp, sets_partido sp, torneo t
where jg.id = p.ganador
  and jp.id = p.perdedor
  and p.fecha = sp.fecha
  and p.num_partido = sp.num_partido
  and t.id = p.torneo
  and p.ronda in ('SF', 'F')
  and t.nombre = 'Roland Garros'
  and extract(year from p.fecha) = '2018'
group by p.ronda, p.desenlace, jg.nombre, jg.apellido, jp.nombre, jp.apellido
```

```
miguel@macmini Trabajo % psql service=citus -f 1.sql/q3.sql
ronda | desenlace | ganador | perdedor | resultado
-----+-----+-----+-----+
SF   | N         | Rafael Nadal | Juan Martin del Potro | 6-4 6-1 6-2
SF   | N         | Dominic Thiem | Marco Cecchinato    | 7-5 7-6(10) 6-1
F    | N         | Rafael Nadal | Dominic Thiem      | 6-4 6-3 6-2
(3 rows)
```

Fig. 22. Bases de datos distribuidas. Citus, consulta 3.

- .4. Muestra la lista de jugadores españoles (ES) que ganaron algún torneo de nivel Gran Slam (G). Para cada jugador muestra los siguientes datos resumen de todos sus partidos: número de partidos jugados, porcentaje de victorias, porcentaje de aces, porcentaje de dobles faltas, porcentaje de servicios ganados, porcentaje de restos ganados, porcentaje de break points salvados (de los sufridos en contra), porcentaje de break points ganados (de los provocados a favor)

```
with jugadores_espanoles_ganadores as (
  select distinct j.id as id_jugador, j.nombre || ' ' || j.apellido as jugador
  from partido p, jugador j, edicion_torneo et
  where p.ganador = j.id
    and p.torneo = et.torneo
```

```

        and p.fecha = et.fecha
        and j.pais = 'ES'
        and p.ronda = 'F'
        and et.nivel = 'G'
    )

select jeg.jugador, count(p.num_partido) as partidos,
    round(100.0 * sum(case when jeg.id_jugador = p.ganador then 1 else 0 end) /
        count(p.num_partido), 1) as pcje_victorias,
    round(100.0 * sum(case when jeg.id_jugador = p.ganador then
        p.num_aces_ganador else p.num_aces_perdedor end) /
        nullif(sum(case when jeg.id_jugador = p.ganador then p.num_ptos_servidos_ganador
            else p.num_ptos_servidos_perdedor end), 0), 1) as pcje_aces,
    round(100.0 * sum(case when jeg.id_jugador = p.ganador then p.num_dob_faltas_ganador
        else p.num_dob_faltas_perdedor end) /
        nullif(sum(case when jeg.id_jugador = p.ganador then p.num_ptos_servidos_ganador
            else p.num_ptos_servidos_perdedor end), 0), 1) as pcje_dobles_faltas,
    round(100.0 * sum(case when jeg.id_jugador = p.ganador then
        p.num_primeros_servicios_ganados_ganador + p.num_segundos_servicios_ganados_ganador
        else p.num_primeros_servicios_ganados_perdedor + p.num_segundos_servicios_ganados_perdedor end) /
        nullif(sum(case when jeg.id_jugador = p.ganador then p.num_ptos_servidos_ganador
            else p.num_ptos_servidos_perdedor end), 0), 1) as pcje_servicios_ganados,
    round(100.0 * sum(case when jeg.id_jugador = p.ganador then
        p.num_ptos_servidos_perdedor - p.num_primeros_servicios_ganados_perdedor -
        p.num_segundos_servicios_ganados_perdedor
        else p.num_ptos_servidos_ganador - p.num_primeros_servicios_ganados_ganador -
        p.num_segundos_servicios_ganados_ganador end) /
        nullif(sum(case when jeg.id_jugador = p.ganador then p.num_ptos_servidos_perdedor
            else p.num_ptos_servidos_ganador end), 0), 1) as pcje_restos_ganados,
    round(100.0 * sum(case when jeg.id_jugador = p.ganador then p.num_break_salvados_ganador
        else p.num_break_salvados_perdedor end) /
        nullif(sum(case when jeg.id_jugador = p.ganador then p.num_break_afrontados_ganador
            else p.num_break_afrontados_perdedor end), 0), 1) as pcje_breaks_salvados,
    round(100.0 * sum(case when jeg.id_jugador = p.ganador then
        p.num_break_afrontados_perdedor - p.num_break_salvados_perdedor
        else p.num_break_afrontados_ganador - p.num_break_salvados_ganador end) /
        nullif(sum(case when jeg.id_jugador = p.ganador then p.num_break_afrontados_perdedor
            else p.num_break_afrontados_ganador end), 0), 1) as pcje_breaks_ganados
from jugadores_espanoles_ganadores jeg, partido p
where jeg.id_jugador = p.ganador
    or jeg.id_jugador = p.perdedor
group by jeg.jugador

```

	jugador	partidos	pcje_victorias	pcje_aces	pcje_dobles_faltas	pcje_servicios_ganados	pcje_restos_ganados	pcje_breaks_salvados	pcje_breaks_ganados
Albert Costa	648	58.5	4.5	2.6	62.5	39.5	60.3	41.0	
Juan Carlos Ferrero	717	64.3	5.1	2.8	63.7	39.7	62.5	40.1	
Carlos Moya	873	63.9	6.4	3.2	63.4	39.0	62.8	40.5	
Carlos Alcaraz	183	79.8	4.6	3.0	65.4	41.6	63.8	41.1	
Rafael Nadal	1276	82.3	4.3	2.3	67.3	42.4	66.1	44.9	
Sergi Bruguera	601	62.9	5.2	2.5	61.9	41.6	59.2	44.9	
(6 rows)									

Fig. 23. Bases de datos distribuidas. Citus, consulta 4.

.5. Lista los jugadores que fueron derrotados (en algún partido del 2018) por el rival de Rafael Nadal de la primera ronda (R128) de Roland Garros de 2018

```

with rival_nadal as (
    select case when jg.nombre = 'Rafael' then jp.id else jg.id end as id_jugador,
        case when jg.nombre = 'Rafael' then jp.nombre || ' ' || jp.apellido
            else jg.nombre || ' ' || jg.apellido end as jugador
    from partido p, jugador jg, jugador jp, edicion_torneo et, torneo t
    where p.ganador = jg.id

```

```

        and p.perdedor = jp.id
        and p.torneo = et.torneo
        and p.fecha = et.fecha
        and et.torneo = t.id
        and t.nombre = 'Roland Garros'
        and p.ronda = 'R128'
        and extract(year from p.fecha) = '2018'
        and (jg.nombre = 'Rafael' and jg.apellido = 'Nadal'
        or jp.nombre = 'Rafael' and jp.apellido = 'Nadal')
)

select j.nombre || ' ' || j.apellido as jugador, j.pais as pais
from rival_nadal rn, partido p, jugador j
where rn.id_jugador = p.ganador
    and p.perdedor = j.id
    and extract(year from p.fecha) = '2018'

```

```

● miguel@macmini Trabajo % psql service=citus -f 1.sql/q5.sql
                jugador      | pais
-----+-----
 Roberto Carballes Baena | ES
 Diego Schwartzman        | AR
 Joao Domingues           | PT
 Federico Delbonis       | AR
 Pablo Cuevas             | UY
 Denis Istomin           | UZ
(6 rows)

```

Fig. 24. Bases de datos distribuidas. Citus, consulta 5.

Factor de replicación

En esta parte de la práctica se nos pide eliminar uno de los nodos para comprobar si Citus es capaz de ejecutar consultas con normalidad. Podemos parar un nodo *worker* deteniendo el servicio de postgres mediante:

```
sudo systemctl stop postgres
```

Probamos ahora a enviar consultas y, como podemos ver en la figura 25 para una consulta a una tabla distribuida, el nodo coordinador detecta que un nodo *worker* no responde y la consulta falla. Pero en la figura 26 podemos observar como consultas a tablas de referencia siguen funcionando, ya que se pueden recuperar los datos.

```

psql:1.sql/q1.sql:8: ERROR: connection to the remote node postgres@51.210.241.207:5432 failed with the following error: serv
er closed the connection unexpectedly
This probably means the server terminated abnormally
before or while processing the request.

```

Fig. 25. Citus, error sin un nodo worker a tabla distribuida.

```

psql:1.sql/q1.sql:8: ERROR: connection to the remote node postgres@51.210.241.207:5432 failed with the following error: serv
er closed the connection unexpectedly
This probably means the server terminated abnormally
before or while processing the request.
miguel@macmini Trabajo % psql service=citus -c "select * from pais limit 1"
codigo_iso2 | codigo_iso3 | codigo_ioc | nombre
-----+-----+-----+-----
 AF     | AFG      | AFG      | Afghanistan
(1 row)

```

Fig. 26. Citus, consulta sin un nodo worker a tabla de referencia.

Para conseguir que las consultas a tablas distribuidas funcionen con nodos del *cluster* no operativos, podemos modificar la variable `citus.shard_replication_factor`, que controla cuántas réplicas de cada fragmento o *shard* se crean en diferentes nodos, asegurando que los datos estén disponibles incluso si un nodo falla. Para cambiar esta variable, debemos *setear* en nuestro esquema esta variable de configuración antes de definir las tablas:

```
DROP TABLE IF EXISTS pais CASCADE;
DROP TABLE IF EXISTS jugador CASCADE;
DROP TABLE IF EXISTS torneo CASCADE;
DROP TABLE IF EXISTS edicion_torneo CASCADE;
DROP TABLE IF EXISTS partido CASCADE;
DROP TABLE IF EXISTS sets_partido CASCADE;
DROP TABLE IF EXISTS ranking CASCADE;

SET citus.shard_replication_factor = 2;

CREATE TABLE pais (
    codigo_iso2      CHAR(2)      PRIMARY KEY,
    codigo_iso3      CHAR(3)      UNIQUE NOT NULL,
    codigo_ioc       CHAR(3)      UNIQUE,
    nombre           VARCHAR(100) UNIQUE NOT NULL
);

...
```

```
psql:3.sql-citus/1.load/schema_replication_2.sql:113: ERROR: cannot create foreign key constraint
DETAIL: Citus currently supports foreign key constraints only for "citus.shard_replication_factor = 1".
HINT: Please change "citus.shard_replication_factor to 1". To learn more about using foreign keys with other replication factors, please contact us at https://citusdata.com/about/contact_us.
psql:3.sql-citus/1.load/schema_replication_2.sql:118: ERROR: cannot create foreign key constraint
DETAIL: Citus currently supports foreign key constraints only for "citus.shard_replication_factor = 1".
HINT: Please change "citus.shard_replication_factor to 1". To learn more about using foreign keys with other replication factors, please contact us at https://citusdata.com/about/contact_us.
```

Fig. 27. Citus, error de clave foránea con replicación.

En la figura 27, podemos observar como Citus vuelve a quejarse de claves foráneas. Antes era capaz de manejar claves foráneas en una tabla distribuida siempre y cuando estas hicieran referencia a atributos de tablas de referencia o de tablas distribuidas y co-localizadas, una limitación que surge de la necesidad de los datos de claves foráneas a residir en el mismo nodo; sin embargo, al añadir la replicación de *shards*, esta imposición es imposible de garantizar.

No obstante, no necesitamos modificar el código ya que las tablas se han creado correctamente, como indica la figura 28. Únicamente la clave foránea no se ha creado, pero no existe manera de solucionar este inconveniente con la versión de Citus instalada, la última disponible.

table_name	citus_table_type	distribution_column	colocation_id	table_size	shard_count	table_owner	access_method
edicion_torneo	distributed	torneo	2	1600 kB	32	postgres	heap
jugador	reference	<none>	1	15 MB	1	postgres	heap
pais	reference	<none>	1	312 kB	1	postgres	heap
partido	distributed	torneo	2	42 MB	32	postgres	heap
ranking	distributed	jugador	2	182 MB	32	postgres	heap
sets_partido	distributed	torneo	2	54 MB	32	postgres	heap
torneo	reference	<none>	1	168 kB	1	postgres	heap

Fig. 28. Citus, tablas con replicación.

Procedemos ahora a parar uno de los nodos y comprobamos que sí podemos ejecutar las consultas (figura 29), aunque Citus nos avisa de que no ha podido comunicarse con un nodo.

```
miguel@macmini Trabajo % psql service=citus -f 1.sql/q3.sql
psql:1.sql:26: WARNING: connection to the remote node postgres@51.210.241.207:5432 failed with the following error: s
erver closed the connection unexpectedly
      This probably means the server terminated abnormally
      before or while processing the request.
ronda | desenlace | ganador | perdedor | resultado
-----+-----+-----+-----+-----+
SF   | N       | Rafael Nadal | Juan Martin del Potro | 6-4 6-1 6-2
SF   | N       | Dominic Thiem | Marco Cecchinato | 7-5 7-6(18) 6-1
F    | N       | Rafael Nadal | Dominic Thiem | 6-4 6-3 6-2
(3 rows)
```

Fig. 29. Citus, consulta a tabla distribuida con un nodo apagado.

Citus Columnar

El formato columnar de Citus introduce soporte para almacenamiento de datos en un formato columnar, diseñado para optimizar el rendimiento en cargas de trabajo analíticas, donde las consultas suelen procesar grandes volúmenes de datos pero acceden a unas pocas columnas específicas. El modo columnar permite una compresión más eficiente y reduce la cantidad de datos leídos. No obstante, las tablas con acceso columnar se convierten en `append only`, es decir, solo se pueden añadir nuevos datos, no se pueden hacer operaciones `UPDATE` o `DELETE`.

Para configurar una tabla con formato columnar, lo indicamos de la siguiente forma:

```
CREATE TABLE partido (
    torneo          INT,
    fecha           DATE,
    ...
    PRIMARY KEY (torneo, fecha, num_partido)
) USING columnar;
```

Una vez cargado el esquema, podemos inspeccionar la tabla `partido` y en la figura 30 observamos que el método de acceso se ha modificado correctamente de `heap`, el valor por defecto, a `columnar`.

Column	Type	Table "public.partido"						
		Collation	Nullable	Default	Storage	Compression	Stats target	Description
torneo	integer		not null		plain			
fecha	date		not null		plain			
num_partido	integer		not null		plain			
num_sets	integer				plain			
ronda	character varying(4)				extended			
desenlace	character(1)				extended			
ganador	integer				plain			
perdedor	integer				plain			
num_aces_ganador	integer				plain			
num_dob_faltas_ganador	integer				plain			
num_ptos_servidos_ganador	integer				plain			
num_primeros_servicios_ganador	integer				plain			
num_primeros_servicios_ganados_ganador	integer				plain			
num_segundos_servicios_ganados_ganador	integer				plain			
num_juegos_servidos_ganador	integer				plain			
num_break_salvados_ganador	integer				plain			
num_break_afrontados_ganador	integer				plain			
num_aces_perdedor	integer				plain			
num_dob_faltas_perdedor	integer				plain			
num_ptos_servidos_perdedor	integer				plain			
num_primeros_servicios_perdedor	integer				plain			
num_primeros_servicios_ganados_perdedor	integer				plain			
num_segundos_servicios_ganados_perdedor	integer				plain			
num_juegos_servidos_perdedor	integer				plain			
num_break_salvados_perdedor	integer				plain			
num_break_afrontados_perdedor	integer				plain			

Indexes:

- "partido_pkey" PRIMARY KEY, btree (torneo, fecha, num_partido)

Access method: columnar

Fig. 30. Citus, tabla `partido` con acceso columnar.

Comparación de tiempos

Para medir el tiempo de ejecución de las consultas, se ha empleado el comando *timing* en `psql` para obtener el tiempo de ejecución en milisegundos. Esto lo haremos desde el nodo coordinador del *cluster* para reducir la latencia.

	Q1 [ms]	Q2 [ms]	Q3 [ms]	Q4 [ms]	Q5 [ms]
Row					
Ejecución 1	29.354	42.19	32.146	169.904	90.101
Ejecución 2	29.708	35.563	37.633	167.64	97.767
Ejecución 3	33.938	38.538	36.29	134.863	131.97
Ejecución 4	18.23	32.685	48.048	149.431	84.634
Ejecución 5	21.41	33.993	29.891	146.508	72.72
Media $\pm \sigma$	26.928 \pm 5.88	36.994 \pm 3.78	36.802 \pm 6.59	153.669 \pm 14.39	95.438 \pm 20.93
Columnar					
Ejecución 1	21.682	28.574	30.581	350.034	80.681
Ejecución 2	13.919	22.264	31.815	358.8	48.361
Ejecución 3	19.801	18.089	32.237	340.87	58.983
Ejecución 4	13.919	20.912	32.962	358.214	92.353
Ejecución 5	16.355	17.047	27.685	375.217	50.133
Media $\pm \sigma$	17.135 \pm 3.133	21.377 \pm 4.059	31.056 \pm 1.854	356.627 \pm 11.354	66.102 \pm 17.448

Table 1. Citus, comparación de tiempos de ejecución (en milisegundos) de las diferentes consultas en tablas con acceso columnar frente a acceso por filas. Cada consulta se ejecutó 5 veces para sacar una estadística fiable.

Podemos observar en la tabla 1 como el formato de acceso columnar mejora ligeramente los tiempos de ejecución en las consultas Q1, Q2, Q3 y Q5 frente al acceso por filas, pero empeora significativamente los de la consulta Q4.

Hacer un *benchmark* de bases de datos no es tarea fácil, y los resultados pueden no reflejar los tiempos de ejecución reales en producción. Intentamos explicar los resultados obtenidos destacando que la consulta Q4 accede a muchas columnas de la tabla partidos para calcular las estadísticas de los jugadores, y este tipo de consultas, el tener que recuperar muchas columnas dentro de una fila, es uno de los puntos débiles de las tablas columnares, que destacan en tareas de agregación o en lecturas de pocas columnas de tablas grandes.

4. BASES DE DATOS NOSQL: DOCUMENTALES (MONGODB)

Para crear el *cluster* de MongoDB propuesto, también emplearemos las máquinas reales del apartado de Citus. Para ello, necesitaremos 3 nodos; en cada uno se ejecutarán 4 procesos `mongod` y un proceso `mongos`. Primero, es necesario instalar y configurar MongoDB.

Instalación y Configuración

Mediante el siguiente *script Bash*, podemos instalar la versión 6.0 del paquete `mongo-org` en nuestras máquinas. Un detalle importante de esta versión con respecto a la versión instalada en las máquinas virtuales es que el comando `mongo` ha sido renombrado a `mongosh`.

```
#!/bin/bash

# run from local with: ssh user@host 'bash -s' < <path_to_this_file>

# Importar la clave de mongo al llavero del sistema operativo
wget -qO - https://www.mongodb.org/static/pgp/server-6.0.asc | gpg --dearmor | \
sudo tee /usr/share/keyrings/mongodb.gpg > /dev/null

# Actualizar el repositorio de apt para incluir mongo-org
echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb.gpg ]" \
"https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/6.0 multiverse" | \
sudo tee /etc/apt/sources.list.d/mongodb-org-6.0.list

# Actualizar el controlador de paquetes del sistema
sudo apt update

# Instalar mongo
sudo apt install -y mongodb-org

# Mostrar versiones instaladas
mongod --version
mongosh --version

# Reiniciar el sistema
sudo reboot
```

Para facilitar el manejo de direcciones IP, podemos añadir un alias para las distintas IP de las máquinas en el archivo `$HOME/etc/hosts`, lo cual nos permitirá hacer referencia a otros nodos de una forma mucho más cómoda.

```
#!/bin/bash

HOSTS=(
    "51.210.241.66    ovh1"
    "51.210.241.239   ovh2"
    "51.210.241.207   ovh3"
)

# Añadir pareja {ip: alias} a /etc/hosts si no existen
for ENTRY in "${HOSTS[@]}"; do
    IP=$(echo "${ENTRY}" | awk '{print $1}')
    HOSTNAME=$(echo "${ENTRY}" | awk '{print $2}')

    if grep -q "${IP}" /etc/hosts; then
        echo "Entry '${ENTRY}' already exists in /etc/hosts. Skipping."
    else
        echo "Adding '${ENTRY}' to /etc/hosts"
        echo "${ENTRY}" | sudo tee -a /etc/hosts > /dev/null
    fi
done
```

```
    fi
done

tail /etc/hosts -n 5
```

Lanzamiento de los Procesos de MongoDB

Para crear nuestro *cluster*, es necesario lanzar un total de 4 procesos `mongod` y un proceso `mongos` en cada nodo. Cada proceso tendrá una configuración específica según su rol en el *cluster*. A continuación, se describen los detalles de estos procesos:

- **1 proceso de configuración (configsvr):** Este proceso `mongod` actúa como el servidor de configuración del *cluster*. Su rol es almacenar los metadatos del *cluster*, como el mapeo de fragmentos y la información de distribución de datos.
- **3 procesos de fragmentos (shardsvr):** Estos procesos `mongod` se encargan de almacenar los datos reales del *cluster*. Cada uno representa un fragmento diferente del conjunto de datos. Cada fragmento se configurará como un conjunto de réplicas para mejorar la redundancia y disponibilidad.
- **1 proceso de enrutamiento (mongos):** Este proceso actúa como el enrutador del *cluster*. Es el encargado de recibir las solicitudes de los clientes y dirigirlas a los fragmentos correspondientes. Debe configurarse para conectarse a los servidores de configuración (`configsvr`) y proporciona un punto de entrada único para las operaciones del cliente, ocultando la complejidad interna del *cluster*.

Para lanzar estos procesos, podemos hacerlo desde la terminal especificando las distintas opciones como argumentos, o podemos crear archivos de configuración específicos para cada proceso. Optamos por la segunda vía, definiendo los siguientes archivos de configuración:

Archivo de Configuración para el Config Server

```
sharding:
  clusterRole: configsvr

replication:
  replSetName: rsConfServer

storage:
  dbPath: /home/ubuntu/dbMongo/rsConfServer

systemLog:
  destination: file
  logAppend: true
  path: /home/ubuntu/dbMongo/rsConfServer/mongod.log

net:
  port: 27010
  bindIp: 0.0.0.0

processManagement:
  timeZoneInfo: /usr/share/zoneinfo
  fork: true

security:
  keyFile: /home/ubuntu/mongo_keyfile
  authorization: enabled
```

Archivo de Configuración para los Shards: para cada uno de los 3 *shards*, el archivo de configuración sería el siguiente. Solo cambia el nombre del conjunto de réplicas (`replicaSetName`), las rutas a los directorios donde se guardarán los datos, los *logs* y el puerto (`net.port`) para cada *shard*.

```

sharding:
  clusterRole: shardsvr

replication:
  replicaSetName: rsShard{1|2|3}

storage:
  dbPath: /home/ubuntu/dbMongo/rsShard{1|2|3}

systemLog:
  destination: file
  logAppend: true
  path: /home/ubuntu/dbMongo/rsShard{1|2|3}/mongod.log

net:
  port: {27011 | 27012 | 27013}
  bindIp: 0.0.0.0

processManagement:
  timeZoneInfo: /usr/share/zoneinfo
  fork: true

security:
  keyFile: /home/ubuntu/mongo_keyfile
  authorization: enabled

```

Archivo de Configuración para el Proceso Mongos

```

sharding:
  configDB: ovh1:27010,ovh2:27010,ovh3:27010

processManagement:
  timeZoneInfo: /usr/share/zoneinfo
  fork: true

security:
  keyFile: /home/ubuntu/mongo_keyfile

systemLog:
  destination: file
  logAppend: true
  path: /home/ubuntu/dbMongo/mongos/mongod.log

net:
  port: 27017
  bindIp: 0.0.0.0

```

A continuación, se explican las diferentes opciones de configuración:

1. sharding.clusterRole:

- Define el rol de cada nodo dentro del *cluster* de *sharding*. Puede ser `configsvr` (servidor de configuración) o `shardsvr` (shard). El proceso `mongos` no tiene esta opción.

2. replication.replicaSetName:

- Especifica el nombre del conjunto de réplicas para los nodos de replicación. Los nodos de los *shards* y el *config server* estarán en conjuntos de réplicas, lo que garantiza la disponibilidad y la redundancia de los datos.

3. **storage.dbPath:**

- Define la ruta en el sistema de archivos donde MongoDB almacenará los datos de la base de datos. Cada componente tiene su propia ruta específica (por ejemplo, *rsConfServer* para el *config server* y *rsShard1*, *rsShard2*, *rsShard3* para los *shards*).

4. **systemLog.destination y systemLog.path:**

- Configuran donde se almacenarán los *logs* del sistema de MongoDB. En este caso, se almacenan en archivos de *log* dentro de un directorio específico.
- *logAppend* asegura que los *logs* se agreguen al archivo existente, evitando sobrescribir los *logs* anteriores.

5. **net.port y net.bindIp:**

- Configuran el puerto y las interfaces de red en las que MongoDB escuchará las conexiones. *bindIp*: 0.0.0.0 permite que el servicio escuche en todas las interfaces de red, abriendo el servidor al tráfico externo.

6. **processManagement.fork:**

- *fork* indica que el proceso de MongoDB se ejecutará en segundo plano como un *daemon* después de iniciarse correctamente y nos devolverá la terminal. Para habilitar esta opción es necesario configurar el guardado de *logs*.

7. **security.keyFile y security.authorization:**

- *keyFile*: Define el archivo de clave compartida que MongoDB usará para autenticar y asegurar la comunicación entre los diferentes nodos del *cluster*.
- *authorization*: Activa la autenticación de usuarios, lo que significa que los clientes también deben autenticarse para acceder a las bases de datos de MongoDB.

Ahora debemos distribuir los archivos de configuración y el archivo *keyFile* usado para autenticar la comunicación entre los nodos a las tres máquinas. Para ello, utilizamos el siguiente *script bash*:

```
#!/bin/bash

# Rutas de archivos y carpetas
MONGO_KEYFILE_NAME="mongo_keyfile"
MONGO_KEYFILE=". ./4.mongo/setup/${MONGO_KEYFILE_NAME}"
MONGO_CONFIGS=". ./4.mongo/setup/configs/*"
MONGO_CONFIGS_PATH="/home/ubuntu/mongo/config/"
MONGO_DB_PATH="/home/ubuntu/dbMongo/"

# Hosts
REMOTE_HOSTS=( "ovh1" "ovh2" "ovh3" )

for REMOTE_HOST in "${REMOTE_HOSTS[@]}"; do
```

```

ssh "$REMOTE_HOST" "sudo rm -r $MONGO_DB_PATH"

# Crear los directorios necesarios en cada máquina
ssh "$REMOTE_HOST" "mkdir -p ${MONGO_CONFIGS_PATH}"
ssh "$REMOTE_HOST" "mkdir -p ${MONGO_DB_PATH}/rsConfServer"
ssh "$REMOTE_HOST" "mkdir -p ${MONGO_DB_PATH}/rsShard1"
ssh "$REMOTE_HOST" "mkdir -p ${MONGO_DB_PATH}/rsShard2"
ssh "$REMOTE_HOST" "mkdir -p ${MONGO_DB_PATH}/rsShard3"
ssh "$REMOTE_HOST" "mkdir -p ${MONGO_DB_PATH}/mongos"

# Copiar el archivo keyFile
scp "$MONGO_KEYFILE" "${REMOTE_HOST}:~/"

# Copiar los archivos de configuración
scp $MONGO_CONFIGS "${REMOTE_HOST}: ${MONGO_CONFIGS_PATH}"

# Dar permisos de lectura y escritura para el archivo keyFile
ssh "$REMOTE_HOST" "sudo chmod 600 ~/ ${MONGO_KEYFILE_NAME}"

done

```

Este script realiza los siguientes pasos:

1. Elimina el directorio de base de datos existente en cada máquina, si existe.
2. Crea los directorios necesarios en cada máquina para almacenar las configuraciones y los datos de MongoDB.
3. Copia el archivo keyFile a cada máquina para garantizar la autenticación segura entre los nodos.
4. Copia los archivos de configuración de MongoDB a cada máquina.
5. Asigna los permisos adecuados al archivo keyFile.

Levantamiento de los Procesos en el Cluster

Ahora que los archivos de configuración y el archivo keyFile están distribuidos, podemos proceder a levantar los procesos necesarios para el *cluster* de MongoDB. El primer paso es iniciar el *replicaset* del servidor de configuración.

Iniciar el ReplicaSet de Configuración

En cada máquina, ejecutamos el siguiente comando para iniciar el proceso mongod con el archivo de configuración para el servidor de configuración:

```
mongod --config mongo/config/configsvr.conf
```

En solo una máquina, nos conectamos a la *shell* de MongoDB usando mongosh:

```
mongosh --port 27010
```

Dentro de la shell de MongoDB, iniciamos el *replicaset* del servidor de configuración:

```

rs.initiate(
{
  _id: "rsConfServer",
  configsvr: true,
  members: [
    { _id : 0, host : "ovh1:27010" },
  ]
}
)

```

```

        { _id : 1, host : "ovh2:27010" },
        { _id : 2, host : "ovh3:27010" }
    ]
}
)

```

Una vez iniciado el servidor de configuración, podemos proceder a lanzar los procesos de los *shards*.

Iniciar los Procesos de los Shards

En cada máquina, ejecutamos los siguientes comandos para iniciar los procesos de los tres *shards*:

```

mongod --config mongo/config/shard1.conf
mongod --config mongo/config/shard2.conf
mongod --config mongo/config/shard3.conf

```

Luego, en solo una máquina, nos conectamos a cada uno de los *replicates* para inicializarlos.

```
mongosh --port 27011
```

Dentro de la *shell* de MongoDB, inicializamos el *replicaset* para el primer *shard*:

```

rs.initiate(
{
  _id : "rsShard1",
  members: [
    { _id : 0, host : "ovh1:27011" },
    { _id : 1, host : "ovh2:27011" },
    { _id : 2, host : "ovh3:27011" }
  ]
}
)

```

Realizamos el mismo proceso para los otros dos *shards*:

```

mongosh --port 27012
rs.initiate(
{
  _id : "rsShard2",
  members: [
    { _id : 0, host : "ovh1:27012" },
    { _id : 1, host : "ovh2:27012" },
    { _id : 2, host : "ovh3:27012" }
  ]
}
)

```

```

mongosh --port 27013
rs.initiate(
{
  _id : "rsShard3",
  members: [
    { _id : 0, host : "ovh1:27013" },
    { _id : 1, host : "ovh2:27013" },
    { _id : 2, host : "ovh3:27013" }
  ]
}
)

```

Lanzar el Proceso Mongos

Finalmente, lanzamos el proceso `mongos` en una máquina con la siguiente configuración:

```
mongos --config mongo/config/mongos.conf
```

Nos conectamos al enrutador y configuramos los diferentes *shards* del *cluster*:

```
mongosh --port 27017
sh.addShard( "rsShard1/ovh1:27011,ovh2:27011,ovh3:27011")
sh.addShard( "rsShard2/ovh1:27012,ovh2:27012,ovh3:27012")
sh.addShard( "rsShard3/ovh1:27013,ovh2:27013,ovh3:27013")
```

Habilitar Autenticación y Crear un Usuario

Un detalle importante es que la autenticación tuvo que deshabilitarse para permitir la conexión con `mongosh` durante la configuración del *cluster*. Una vez que el *cluster* esté creado, nos conectamos y creamos un usuario privilegiado con los siguientes comandos:

```
db.createUser({
  user: "admin",
  pwd: "***",
  roles: [
    { role: "userAdminAnyDatabase", db: "admin" },
    { role: "clusterAdmin", db: "admin" },
    { role: "readWriteAnyDatabase", db: "admin" }
  ]
});
```

Finalmente, debemos parar todos los procesos `mongod` y `mongos` en todas las máquinas utilizando el siguiente comando:

```
pkill -9 mongo
```

En la figura 31 mostramos la verificación de que no existen procesos `mongod` o `mongos` en ejecución con el siguiente comando:

```
pgrep mongo
```

```
ubuntu@ovh1:~$ pgrep mongo
100887
101131
101247
101364
101515
ubuntu@ovh1:~$ pkill -9 mongo
ubuntu@ovh1:~$ pgrep mongo
ubuntu@ovh1:~$
ubuntu@ovh1:~$
ubuntu@ovh1:~$
ubuntu@ovh1:~$
```

Fig. 31. Verificación de procesos `mongod` en ejecución

Reabilitamos la autenticación, y luego volvemos a lanzar los procesos `mongod` y `mongos`. Esta vez no es necesario inicializar los *replicaSets* o agregar los *shards*, ya que el *cluster* mantiene los datos de configuración. Esto lo vemos en la figura 32.

```

ubuntu@ovh1:~$ mongod --config mongo/config/shard1.conf
about to fork child process, waiting until server is ready for connections.
forked process: 101131
child process started successfully, parent exiting
ubuntu@ovh1:~$ mongod --config mongo/config/shard2.conf
about to fork child process, waiting until server is ready for connections.
forked process: 101247
child process started successfully, parent exiting
ubuntu@ovh1:~$ mongod --config mongo/config/shard3.conf
about to fork child process, waiting until server is ready for connections.
forked process: 101364
child process started successfully, parent exiting
ubuntu@ovh1:~$ mongos --config mongo/config/mongos.conf
about to fork child process, waiting until server is ready for connections.
forked process: 101515
child process started successfully, parent exiting
ubuntu@ovh1:~$ pgrep mongo
100887
101131
101247
101364
101515
ubuntu@ovh1:~$ █

```

Fig. 32. Lanzamiento de los procesos *mongod* y *mongos*.

A. Extracción de datos para consultas Q1 y Q3

En la práctica se nos pide diseñar una colección que permita resolver de forma eficiente las consultas Q1 y Q3, consultas centradas en partidos de torneos específicos. Diseñamos, pues, esta colección partiendo de la tabla de partidos, y agregando a cada partido todas las demás tablas, excepto la tabla de *ranking*.

En nuestra base de datos relacional, ejecutamos la siguiente instrucción SQL para generar un archivo que contenga los diferentes documentos JSON que importaremos a MongoDB:

```

COPY (
  SELECT
    jsonb_build_object(
      'torneo', jsonb_build_object(
        'id', t.id,
        'nombre', t.nombre,
        'fecha', et.fecha,
        'pais', CASE
          WHEN tp.codigo_iso2 IS NOT NULL THEN jsonb_build_object(
            'codigo_iso2', tp.codigo_iso2,
            'codigo_iso3', tp.codigo_iso3,
            'codigo_ioc', tp.codigo_ioc,
            'nombre', tp.nombre
          )
          ELSE NULL
        END,
        'superficie', et.superficie,
        'tamano', et.tamano,
        'nivel', et.nivel
      ),
      'num_sets', p.num_sets,
      'ronda', p.ronda,
      'desenlace', p.desenlace,
      'ganador', CASE
        WHEN jg.id IS NOT NULL THEN jsonb_build_object(
          'id', jg.id,
          'nombre', jg.nombre,
          'apellido', jg.apellido,
          'diestro', jg.diestro,
          'fecha_nacimiento', jg.fecha_nacimiento,
        )
      END
    )
  FROM partido p
  LEFT JOIN torneo t ON p.torneo_id = t.id
  LEFT JOIN estadio et ON p.estadio_id = et.id
  LEFT JOIN pais tp ON p.pais_id = tp.id
  LEFT JOIN jugadora jg ON p.ganador_id = jg.id
)
INTO partido_json;

```

```

'altura', jg.altura,
'pais', CASE
    WHEN pg.codigo_iso2 IS NOT NULL THEN jsonb_build_object(
        'codigo_iso2', pg.codigo_iso2,
        'codigo_iso3', pg.codigo_iso3,
        'codigo_ioc', pg.codigo_ioc,
        'nombre', pg.nombre
    )
    ELSE NULL
END,
'stats', jsonb_build_object(
    'aces', p.num_aces_ganador,
    'dobles_faltas', p.num_dob_faltas_ganador,
    'puntos_servidos', p.num_ptos_servidos_ganador,
    'primeros_servicios', p.num_primeros_servicios_ganador,
    'primeros_servicios_ganados', p.num_primeros_servicios_ganados_ganador,
    'segundos_servicios_ganados', p.num_segundos_servicios_ganados_ganador,
    'juegos_servidos', p.num_juegos_servidos_ganador,
    'breaks_salvados', p.num_break_salvados_ganador,
    'breaks_afrontados', p.num_break_afrontados_ganador
)
)
ELSE NULL
END,
'perdedor', CASE
    WHEN jp.id IS NOT NULL THEN jsonb_build_object(
        'id', jp.id,
        'nombre', jp.nombre,
        'apellido', jp.apellido,
        'diestro', jp.diestro,
        'fecha_nacimiento', jp.fecha_nacimiento,
        'altura', jp.altura,
        'pais', CASE
            WHEN pp.codigo_iso2 IS NOT NULL THEN jsonb_build_object(
                'codigo_iso2', pp.codigo_iso2,
                'codigo_iso3', pp.codigo_iso3,
                'codigo_ioc', pp.codigo_ioc,
                'nombre', pp.nombre
            )
            ELSE NULL
        END,
        'stats', jsonb_build_object(
            'aces', p.num_aces_perdedor,
            'dobles_faltas', p.num_dob_faltas_perdedor,
            'puntos_servidos', p.num_ptos_servidos_perdedor,
            'primeros_servicios', p.num_primeros_servicios_perdedor,
            'primeros_servicios_ganados', p.num_primeros_servicios_ganados_perdedor,
            'segundos_servicios_ganados', p.num_segundos_servicios_ganados_perdedor,
            'juegos_servidos', p.num_juegos_servidos_perdedor,
            'breaks_salvados', p.num_break_salvados_perdedor,
            'breaks_afrontados', p.num_break_afrontados_perdedor
        )
    )
    ELSE NULL
)
ELSE NULL
END,
'sets', (
    SELECT jsonb_agg(
        jsonb_build_object(
            'num_set', sp.num_set,
            'juegos_ganador', sp.juegos_ganador,
            'juegos_perdedor', sp.juegos_perdedor,
            'puntos_tiebreak_perdedor', sp.puntos_tiebreak_perdedor
        )
    )
)

```

```

        )
    )
    FROM sets_partido sp
    WHERE sp.torneo = p.torneo AND sp.fecha = p.fecha AND sp.num_partido = p.num_partido
)
)
FROM
partido p
JOIN edicion_torneo et ON p.torneo = et.torneo AND p.fecha = et.fecha
JOIN torneo t ON et.torneo = t.id
LEFT JOIN pais tp ON t.pais = tp.codigo_iso2
LEFT JOIN jugador jg ON p.ganador = jg.id
LEFT JOIN pais pg ON jg.pais = pg.codigo_iso2
LEFT JOIN jugador jp ON p.perdedor = jp.id
LEFT JOIN pais pp ON jp.pais = pp.codigo_iso2
) TO '/tmp/tenis.json';

```

Al ejecutarla, nos genera el archivo tenis.json, que contiene documentos JSON con el siguiente formato:

```
{
  "sets": [
    {
      "num_set": 1,
      "juegos_ganador": 6,
      "juegos_perdedor": 4,
      "puntos_tiebreak_perdedor": null
    },
    {
      "num_set": 2,
      "juegos_ganador": 5,
      "juegos_perdedor": 7,
      "puntos_tiebreak_perdedor": null
    },
    {
      "num_set": 3,
      "juegos_ganador": 6,
      "juegos_perdedor": 4,
      "puntos_tiebreak_perdedor": null
    }
  ],
  "ronda": "R32",
  "torneo": {
    "id": 112,
    "pais": {
      "nombre": "United Kingdom",
      "codigo_ioc": "GBR",
      "codigo_iso2": "GB",
      "codigo_iso3": "GBR"
    },
    "fecha": "2019-07-07",
    "nivel": "A",
    "nombre": "Newport",
    "tamano": 32,
    "superficie": "Hierba"
  },
  "ganador": {
    "id": 105815,
    "pais": {
      "nombre": "United States of America",
      "codigo_ioc": "USA",
      "codigo_iso2": "US",
      "codigo_iso3": "USA"
    }
  }
}
```

```
"codigo_iso3": "USA"
},
"stats": {
  "aces": 8,
  "dobles_faltas": 4,
  "breaks_salvados": 4,
  "juegos_servidos": 16,
  "puntos_servidos": 113,
  "breaks_afrontados": 6,
  "primeros_servicios": 57,
  "primeros_servicios_ganados": 46,
  "segundos_servicios_ganados": 25
},
"altura": 188,
"nombre": "Tennys",
"diestro": true,
"apellido": "Sandgren",
"fecha_nacimiento": "1991-07-07"
},
"num_sets": 3,
"perdedor": {
  "id": 104797,
  "pais": {
    "nombre": "Uzbekistan",
    "codigo_ioc": "UZB",
    "codigo_iso2": "UZ",
    "codigo_iso3": "UZB"
  },
  "stats": {
    "aces": 1,
    "dobles_faltas": 4,
    "breaks_salvados": 5,
    "juegos_servidos": 16,
    "puntos_servidos": 101,
    "breaks_afrontados": 8,
    "primeros_servicios": 60,
    "primeros_servicios_ganados": 43,
    "segundos_servicios_ganados": 20
  },
  "altura": 188,
  "nombre": "Denis",
  "diestro": true,
  "apellido": "Istomin",
  "fecha_nacimiento": "1986-09-09"
},
"desenlace": "N"
}
```

B. Indexación, sharding y carga de datos

A la hora de distribuir los datos, debemos también indexar el atributo que vayamos a utilizar como clave de partición. Elegimos indexar y particionar por el atributo torneo.nombre mediante *hash*, ya que es empleado en casi todas las consultas y presenta una alta cardinalidad. Deberemos comprobar, después de importar los documentos, cuántos *chunks* guarda cada *shard* para verificar que existe un reparto equitativo. Ajustamos el valor del tamaño de *chunks*, por defecto 128 megabytes en MongoDB 6.0, a un tamaño de 8 megas, para forzar la distribución de los datos entre todos los *shards*:

```
use tenis

db.createCollection('partidos')

db.settings.updateOne(
  { _id: "chunksize" },
  { $set: { _id: "chunksize", value: 8 } },
  { upsert: true }
)

db.partidos.createIndex({ "torneo.nombre": "hashed" })
sh.shardCollection("tenis.partidos", { "torneo.nombre": "hashed" })
```

Generamos y cargamos los datos desde ovh1:

```
psql service=citus -f 4.mongo/sql_to_json.sql
mongoimport --host ovh1 --port 27017 --db tenis2 --collection partidos --file /tmp/tenis.json --authenticationDatabase
```

Debemos verificar que nuestro *cluster* ha reparticionado los datos de forma equitativa entre todos los nodos. Esto lo vemos en la figura 33 y lo hacemos con el siguiente comando:

```
sh.status()
```

```
collections: [
  'tenis.partidos': {
    shardKey: {
      'torneo.nombre': 'hashed'
    },
    unique: false,
    balancing: true,
    chunkMetadata: [
      {
        shard: 'rsShard1',
        nChunks: 2
      },
      {
        shard: 'rsShard2',
        nChunks: 3
      },
      {
        shard: 'rsShard3',
        nChunks: 2
      }
    ],
  }
]
```

Fig. 33. Verificación del reparto equitativo de los datos en los nodos del cluster en MongoDB.

C. Consultas MongoDB

C.1. Muestra todos los ganadores del torneo “Wimbledon” (Nombre apellidos y año). Ordena el resultado por año.

Esta consulta (cuyo resultado vemos en la figura 34) está compuesta por tres etapas en un pipeline de agregación:

1. **\$match:** Filtramos los documentos para incluir solo aquellos del torneo “Wimbledon” y de la ronda final (‘F’).
2. **\$project:** Seleccionamos los campos que se mostrarán en el resultado. En este caso:
 - Excluye el campo `_id`.
 - Muestra el nombre y apellido del ganador.
 - Extrae y muestra el año de la fecha del torneo.
3. **\$sort:** Ordenamos los documentos por el año de forma ascendente.

```
db.partidos.aggregate([
  {
    $match: {
      "torneo.nombre": "Wimbledon",
      ronda: "F",
    },
  },
  {
    $project: {
      _id: 0,
      nombre: "$ganador.nombre",
      apellido: "$ganador.apellido",
      año: {
        $year: {
          $toDate: "$torneo.fecha",
        },
      },
    },
  },
  {
    $sort: {
      año: 1,
    },
  },
]);

```

• { "nombre": "Michael", "apellido": "Stich", "año": 1991 }
• { "nombre": "Andre", "apellido": "Agassi", "año": 1992 }
• { "nombre": "Pete", "apellido": "Sampras", "año": 1993 }
• { "nombre": "Pete", "apellido": "Sampras", "año": 1994 }
• { "nombre": "Pete", "apellido": "Sampras", "año": 1995 }

Fig. 34. Bases de datos NoSQL. MongoDB, consulta 1. Al no caber en la propia pantalla, mostramos parte del resultado.

C.2. Muestra los años en los que Roger Federer ganó algún torneo de nivel Gran Slam (G) o Master 1000 (M). Para cada año, muestra el número de torneos y lista sus nombres (ordenados por la fecha de celebración). Ordena el resultado por el año

Esta consulta (cuyo resultado mostramos en la figura 35) consta de seis etapas en el pipeline de agregación:

1. **\$match:** Filtramos los documentos para incluir solo aquellos en los que el apellido del ganador es "Federer", el nivel del torneo está en los valores "G" o "M", y la ronda es la final ("F").
2. **\$addFields:** Agregamos un nuevo campo año, que contiene el año extraído de la fecha del torneo.
3. **\$group:** Agrupamos los documentos por el campo año, contando el total de torneos por año (total), y construimos un *array* de torneos (torneos) con el nombre y la fecha de cada torneo.
4. **\$addFields:** Dentro de la etapa de agregar campos, ordenamos el *array* de torneos por la fecha de celebración del torneo de manera ascendente.
5. **\$project:** Proyectamos los campos del resultado final. Excluimos el campo _id, pero mostramos el año y el total de torneos por año. Además, convertimos el *array* de torneos en una cadena de texto separada por comas, listando los nombres de los torneos en orden.
6. **\$sort:** Ordenamos los resultados por el campo año de manera ascendente.

```
db.partidos.aggregate([
  {
    $match: {
      "ganador.apellido": "Federer",
      "torneo.nivel": { $in: ["G", "M"] },
      ronda: "F"
    }
  },
  {
    $addFields: {
      año: {
        $year: { $toDate: "$torneo.fecha" }
      }
    }
  },
  {
    $group: {
      _id: "$año",
      total: { $sum: 1 },
      torneos: {
        $push: {
          nombre: "$torneo.nombre",
          fecha: { $toDate: "$torneo.fecha" }
        }
      }
    }
  },
  {
    $addFields: {
      torneos: {
        $sortArray: {
          input: "$torneos",
          sortBy: { fecha: 1 }
        }
      }
    }
  }
])
```

```

$project: {
    _id: 0,
    año: "$_id",
    total: "$total",

    torneos: {
        $reduce: {
            input: "$torneos.nombre",
            initialValue: "",
            in: {
                $cond: {
                    if: { $eq: ["$$value", ""] },
                    then: "$$this",
                    else: {
                        $concat: ["$$value", ", ", "$$this"],
                    },
                },
            },
        },
    },
},
{
    $sort: { año: 1 },
},
]);

```

año : 2002
total : 1
torneos : "Hamburg Masters"
año : 2003
total : 1
torneos : "Wimbledon"
año : 2004
total : 6
torneos : "Australian Open, Indian Wells Masters, Hamburg Masters, Wimbledon, Can..."
año : 2005
total : 6
torneos : "Indian Wells Masters, Miami Masters, Hamburg Masters, Wimbledon, Cinci..."
año : 2006
total : 7
torneos : "Australian Open, Indian Wells Masters, Miami Masters, Wimbledon, Canad..."
▶ año : 2007
total : 5
torneos : "Australian Open, Hamburg Masters, Wimbledon, Cincinnati Masters, US Op..."
año : 2008
total : 1
torneos : "US Open"
año : 2009
total : 4
torneos : "Madrid Masters, Roland Garros, Wimbledon, Cincinnati Masters"
año : 2010
total : 2
torneos : "Australian Open, Cincinnati Masters"
año : 2011
total : 1
torneos : "Paris Masters"
año : 2012
total : 4
torneos : "Indian Wells Masters, Madrid Masters, Wimbledon, Cincinnati Masters"
año : 2014
total : 2
torneos : "Cincinnati Masters, Shanghai Masters"
año : 2015

Fig. 35. Bases de datos NoSQL. MongoDB, consulta 2.

C.3. Muestra los partidos de semifinales (ronda='SF') y final (ronda = 'F') del torneo de "Roland Garros" del 2018. Para cada partido muestra la ronda, el tipo de desenlace, el nombre y apellidos del ganador y el nombre y apellidos del perdedor y el resultado con el número de juegos del ganador y del perdedor en cada set, y opcionalmente en paréntesis el número de juegos del perdedor en el tie break

Esta consulta (cuyo resultado vemos en la figura 36) consta de tres etapas en el pipeline de agregación:

1. **\$match:** Filtramos los documentos para obtener los partidos del torneo "Roland Garros" que se jugaron en el año 2018. Además, seleccionamos solo los partidos que se juegan en las rondas SF (semifinal) o F (final).
2. **\$addFields:** Agregamos un campo nuevo llamado `resultado` que concatena los resultados de los sets del partido en el siguiente formato: `juegos_ganador-juegos_perdedor`. Si el set contiene puntos de *tiebreak* para el perdedor, los incluimos entre paréntesis después del marcador, y si no hay *tiebreak*, se omite. Este campo contiene la cadena de resultados de todos los sets jugados en el partido.
3. **\$project:** Proyectamos los campos seleccionados en el resultado final. Excluyimos el campo `_id`, y muestramos la ronda, el desenlace, el nombre completo del ganador (concatenando el nombre y apellido), el nombre completo del perdedor (concatenando nombre y apellido), y el campo `resultado` que fue generado en la etapa anterior.

```
db.partidos.aggregate([
  {
    $match: {
      "torneo.nombre": "Roland Garros",
      $expr: {
        $eq: [{ $year: { $toDate: "$torneo.fecha" } }, 2018],
      },
      ronda: { $in: ["SF", "F"] },
    },
  },
  {
    $addFields: {
      resultado: {
        $reduce: {
          input: "$sets",
          initialValue: "",
          in: {
            $concat: [
              "$$value",
              {
                $cond: {
                  if: { $eq: ["$$value", "" ] },
                  then: "",
                  else: " ",
                },
              },
            ],
          },
        },
        "-",
        {
          $toString: "$$this.juegos_ganador",
        },
        "-",
        {
          $toString: "$$this.juegos_perdedor",
        },
        {
          $cond: {
            if: {
              $ne: ["$$this.puntos_tiebreak_perdedor", null],
            },
            then: {

```

```
$concat: [
    "(",
    {
        $toString: $$this.puntos_tiebreak_perdedor
    },
    ")",
],
},
else: "",
},
},
],
},
},
},
},
},
},
},
},
$project: {
    _id: 0,
    ronda: 1,
    desenlace: 1,
    ganador: {
        $concat: ["$ganador.nombre", " ", "$ganador.apellido"],
    },
    perdedor: {
        $concat: ["$perdedor.nombre", " ", "$perdedor.apellido"],
    },
    resultado: 1,
},
},
]);
});
```

```
ronda : "SF"  
desenlace : "N"  
resultado : "6-4 6-1 6-2"  
ganador : "Rafael Nadal"  
perdedor : "Juan Martin del Potro"
```

```
ronda : "SF"  
desenlace : "N"  
resultado : "7-5 7-6(10) 6-1"  
ganador : "Dominic Thiem"  
perdedor : "Marco Cecchinato"
```

```
ronda : "F"  
desenlace : "N"  
resultado : "6-4 6-3 6-2"  
ganador : "Rafael Nadal"  
perdedor : "Dominic Thiem"
```

Fig. 36. Bases de datos NoSQL. MongoDB, consulta 3.

C.4. Muestra la lista de jugadores españoles (ES) que ganaron algún torneo de nivel Gran Slam (G). Para cada jugador muestra los siguientes datos resumen de todos sus partidos: número de partidos jugados, porcentaje de victorias, porcentaje de aces, porcentaje de dobles faltas, porcentaje de servicios ganados, porcentaje de restos ganados, porcentaje de break points salvados (de los sufridos en contra), porcentaje de break points ganados (de los provocados a favor)

Esta consulta (cuyo resultado mostramos en la figura 37 consta de varias etapas en el pipeline de agregación:

1. **\$match:** Filtramos los partidos donde el torneo es de nivel "G" (Grand Slam), la ronda es "F" (final), y el ganador es de España (codigo_iso2 = "ES").
2. **\$group:** Agrupamos los documentos por el id del ganador, y para cada grupo obtenemos el nombre completo del jugador (concatenando el nombre y el apellido del ganador) utilizando \$first.
3. **\$lookup:** Realizamos una operación de join con la colección partidos, buscando todos los partidos en los que el jugador (basado en su id) haya sido ganador o perdedor. Los resultados los almacenamos en un array llamado all_matches.
4. **\$project:** Proyectamos varios campos:
 - jugador: El nombre completo del ganador.
 - partidos: El número de partidos en los que el jugador ha participado, calculado con \$size.
 - pcje_victorias: El porcentaje de victorias calculado como la proporción de victorias del jugador sobre el total de partidos jugados, multiplicado por 100.
 - pcje_aces: El porcentaje de aces ganados, calculado como el total de aces en los partidos ganados dividido por el total de puntos servidos.
 - pcje_dobles_faltas: El porcentaje de dobles faltas cometidas, calculado de manera similar a pcje_aces.
 - pcje_servicios_ganados: El porcentaje de puntos de servicio ganados, basado en los primeros y segundos servicios, calculado por la misma lógica.
 - pcje_restos_ganados: El porcentaje de puntos ganados en los restos de servicio, calculado de manera similar a pcje_servicios_ganados.
 - pcje_breaks_salvados: El porcentaje de breaks salvados en comparación con los breaks enfrentados, calculado con la misma lógica.
 - pcje_breaks_ganados: El porcentaje de breaks ganados comparado con los breaks enfrentados.
5. **\$project (redondeo):** Redondeamos todos los porcentajes calculados en la etapa anterior a un decimal.

```
db.partidos.aggregate([
  {
    $match: {
      "torneo.nivel": "G",
      ronda: "F",
      "ganador.pais.codigo_iso2": "ES",
    },
  },
  {
    $group: {
      _id: "$ganador.id",
    }
  }
])
```

```
jugador: {
    $first: {
        $concat: ["$ganador.nombre", " ", "$ganador.apellido"],
    },
},
},
$lookup: {
    from: "partidos",
    let: { playerId: "$_id" },
    pipeline: [
        {
            $match: {
                $expr: {
                    $or: [
                        {
                            $eq: ["$ganador.id", "$$playerId"],
                        },
                        {
                            $eq: ["$perdedor.id", "$$playerId"],
                        },
                    ],
                },
            },
        },
        {
            as: "all_matches",
        },
    ],
},
{
    $project: {
        _id: 0,
        jugador: 1,
        partidos: { $size: "$all_matches" },
        pcje_victorias: {
            $multiply: [
                {
                    $divide: [
                        {
                            $size: {
                                $filter: {
                                    input: "$all_matches",
                                    cond: {
                                        $eq: ["$$this.ganador.id", "$_id"],
                                    },
                                },
                            },
                            { $size: "$all_matches" },
                        ],
                    },
                    100,
                ],
            },
            pcje_aces: {
                $multiply: [
                    {
                        $divide: [
                            {
                                $size: {
                                    $filter: {
                                        input: "$all_matches",
                                        cond: {
                                            $eq: ["$$this.perdedor.id", "$_id"],
                                        },
                                    },
                                },
                            },
                            { $size: "$all_matches" },
                        ],
                    },
                ],
            },
        },
    },
},
```

```
$sum: {
    $map: {
        input: "$all_matches",
        as: "match",
        in: {
            $cond: [
                {
                    $eq: ["$$match.ganador.id", "$_id"],
                },
                "$$match.ganador.stats.aces",
                "$$match.perdedor.stats.aces",
            ],
        },
    },
    $sum: {
        $map: {
            input: "$all_matches",
            as: "match",
            in: {
                $cond: [
                    {
                        $eq: ["$$match.ganador.id", "$_id"],
                    },
                    "$$match.ganador.stats.puntos_servidos",
                    "$$match.perdedor.stats.puntos_servidos",
                ],
            },
        },
    },
    100,
],
},
pcje_dobles_faltas: {
    $multiply: [
        {
            $divide: [
                {
                    $sum: {
                        $map: {
                            input: "$all_matches",
                            as: "match",
                            in: {
                                $cond: [
                                    {
                                        $eq: ["$$match.ganador.id", "$_id"],
                                    },
                                    "$$match.ganador.stats.dobles_faltas",
                                    "$$match.perdedor.stats.dobles_faltas",
                                ],
                            },
                        },
                    },
                },
            ],
        },
    ],
},
```

```
        input: "$all_matches",
        as: "match",
        in: {
          $cond: [
            {
              $eq: ["$$match.ganador.id", "$_id"],
            },
            "$$match.ganador.stats.puntos_servidos",
            "$$match.perdedor.stats.puntos_servidos",
          ],
        },
      },
    ],
  },
  100,
],
},
pcje_servicios_ganados: {
  $multiply: [
  {
    $divide: [
    {
      $sum: {
        $map: {
          input: "$all_matches",
          as: "match",
          in: {
            $cond: [
              {
                $eq: ["$$match.ganador.id", "$_id"],
              },
              {
                $add: [
                  "$$match.ganador.stats.primeros_servicios_ganados",
                  "$$match.ganador.stats.segundos_servicios_ganados",
                ],
              },
              {
                $add: [
                  "$$match.perdedor.stats.primeros_servicios_ganados",
                  "$$match.perdedor.stats.segundos_servicios_ganados",
                ],
              },
            ],
          },
        },
      },
    ],
  },
  ],
},
{
  $sum: {
    $map: {
      input: "$all_matches",
      as: "match",
      in: {
        $cond: [
          {
            $eq: ["$$match.ganador.id", "$_id"],
          },
          "$$match.ganador.stats.puntos_servidos",
          "$$match.perdedor.stats.puntos_servidos",
        ],
      },
    },
  },
}
```

```
        ],
      },
    },
  },
],
},
100,
],
},
pcje_restos_ganados: {
  $multiply: [
    {
      $divide: [
        {
          $sum: {
            $map: {
              input: "$all_matches",
              as: "match",
              in: {
                $cond: [
                  {
                    $eq: ["$$match.ganador.id", "$_id"],
                  },
                  {
                    $subtract: [
                      "$$match.perdedor.stats.puntos_servidos",
                      {
                        $add: [
                          "$$match.perdedor.stats.primeros_servicios_ganados",
                          "$$match.perdedor.stats.segundos_servicios_ganados",
                        ],
                      },
                    ],
                  },
                  {
                    $subtract: [
                      "$$match.ganador.stats.puntos_servidos",
                      {
                        $add: [
                          "$$match.ganador.stats.primeros_servicios_ganados",
                          "$$match.ganador.stats.segundos_servicios_ganados",
                        ],
                      },
                    ],
                  },
                  ],
                },
              },
            },
            $sum: {
              $map: {
                input: "$all_matches",
                as: "match",
                in: {
                  $cond: [
                    {
                      $eq: ["$$match.ganador.id", "$_id"],
                      "$$match.perdedor.stats.puntos_servidos",
                    },
                  ],
                },
              },
            },
          },
        ],
      },
    },
  ],
}
```

```
        "$$match.ganador.stats.puntos_servidos",
    ],
},
},
],
},
],
},
100,
],
},
pcje_breaks_salvados: {
$multiply: [
{
$divide: [
{
$sum: {
$map: {
input: "$all_matches",
as: "match",
in: {
$cond: [
{
$eq: ["$$match.ganador.id", "$_id"],
},
"$$match.ganador.stats.breaks_salvados",
"$$match.perdedor.stats.breaks_salvados",
],
},
},
},
},
],
},
},
},
{
$sum: {
$map: {
input: "$all_matches",
as: "match",
in: {
$cond: [
{
$eq: ["$$match.ganador.id", "$_id"],
},
"$$match.ganador.stats.breaks_afrontados",
"$$match.perdedor.stats.breaks_afrontados",
],
},
},
},
},
},
],
},
100,
],
},
pcje_breaks_ganados: {
$multiply: [
{
$divide: [
{
$sum: {
$map: {
input: "$all_matches",

```

```
        as: "match",
        in: {
          $cond: [
            {
              $eq: ["$$match.ganador.id", "$_id"],
            },
            {
              $subtract: [
                "$$match.perdedor.stats.breaks_afrontados",
                "$$match.perdedor.stats.breaks_salvados",
              ],
            },
            {
              $subtract: [
                "$$match.ganador.stats.breaks_afrontados",
                "$$match.ganador.stats.breaks_salvados",
              ],
            },
            ],
          },
        },
      },
      $sum: {
        $map: {
          input: "$all_matches",
          as: "match",
          in: {
            $cond: [
              {
                $eq: ["$$match.ganador.id", "$_id"],
              },
              "$$match.perdedor.stats.breaks_afrontados",
              "$$match.ganador.stats.breaks_afrontados",
            ],
          },
        },
        100,
      },
    },
  },
}

// Round all percentages to 1 decimal place
{
  $project: {
    jugador: 1,
    partidos: 1,
    pcje_victorias: {
      $round: ["$pcje_victorias", 1],
    },
    pcje_aces: { $round: ["$pcje_aces", 1] },
    pcje_dobles_faltas: {
      $round: ["$pcje_dobles_faltas", 1],
    },
    pcje_servicios_ganados: {
      $round: ["$pcje_servicios_ganados", 1],
    }
  }
}
```

```

    },
    pcje_restos_ganados: {
      $round: ["$pcje_restos_ganados", 1],
    },
    pcje_breaks_salvados: {
      $round: ["$pcje_breaks_salvados", 1],
    },
    pcje_breaks_ganados: {
      $round: ["$pcje_breaks_ganados", 1],
    },
  },
]);

```

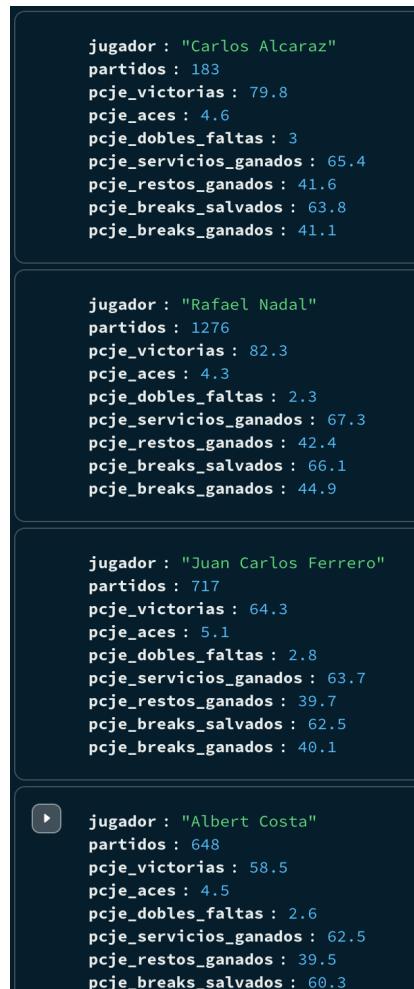


Fig. 37. Bases de datos NoSql. MongoDB, consulta 4. Resultado parcial debido al tamaño del mismo.

C.5. *Lista los jugadores que fueron derrotados (en algún partido del 2018) por el rival de Rafael Nadal de la primera ronda (R128) de Roland Garros de 2018*

Esta consulta (cuyo resultado mostramos en la figura 38) consta de varias etapas en el pipeline de agregación:

1. **\$match:** Filtramos los partidos del torneo "Roland Garros" en el año 2018, específicamente de la ronda R128, y donde al menos uno de los jugadores es Rafael Nadal, ya sea como ganador o perdedor.
2. **\$project:** Agregamos un campo `rival_id` que determina el `id` del rival del jugador Rafael Nadal. Si

Rafael Nadal es el ganador del partido, se asigna el id del perdedor; de lo contrario, se asigna el id del ganador.

3. **\$lookup:** Realizamos una operación de join con la colección partidos, buscando los partidos donde el id del rival aparece como ganador. El resultado de esta operación se almacena en el campo partidos_rival.
4. **\$unwind:** Descomponemos el array partidos_rival para obtener un único documento por cada partido encontrado. Esto convierte los elementos del array en documentos separados dentro del flujo.
5. **\$match:** Filtramos los documentos para asegurarse de que los partidos de la colección partidos_rival sean del año 2018, usando el operador \$year en la fecha del torneo.
6. **\$project:** Proyectamos el nombre y apellido del perdedor de cada partido del rival, concatenando ambos en un solo campo jugador. También proyectamos el código de país (codigo_iso2) del perdedor del partido del rival.

```
db.partidos.aggregate([
  {
    $match: {
      "torneo.nombre": "Roland Garros",
      $expr: {
        $eq: [{ $year: { $toDate: "$torneo.fecha" } }, 2018],
      },
      ronda: "R128",
      $or: [
        {
          "ganador.nombre": "Rafael",
          "ganador.apellido": "Nadal",
        },
        {
          "perdedor.nombre": "Rafael",
          "perdedor.apellido": "Nadal",
        },
      ],
    },
  },
  {
    $project: {
      rival_id: {
        $cond: [
          {
            $and: [
              {
                $eq: ["$ganador.nombre", "Rafael"],
              },
              {
                $eq: ["$ganador.apellido", "Nadal"],
              },
            ],
          },
          {
            "$perdedor.id",
            "$ganador.id",
          },
        ],
      },
    },
  },
  {
    $lookup: {
      from: "partidos",
    }
  }
])
```

```
localField: "rival_id",
foreignField: "ganador.id",
as: "partidos_rival",
},
{
$unwind: "$partidos_rival",
},
{
$match: {
$expr: {
$eq: [
{
$year: {
$toDate: "$partidos_rival.torneo.fecha",
},
},
2018,
],
},
},
},
{
$project: {
_id: 0,
jugador: {
$concat: [
"$partidos_rival.perdedor.nombre",
" ",
"$partidos_rival.perdedor.apellido",
],
},
pais: "$partidos_rival.perdedor.pais.codigo_iso2",
},
},
],
});
```

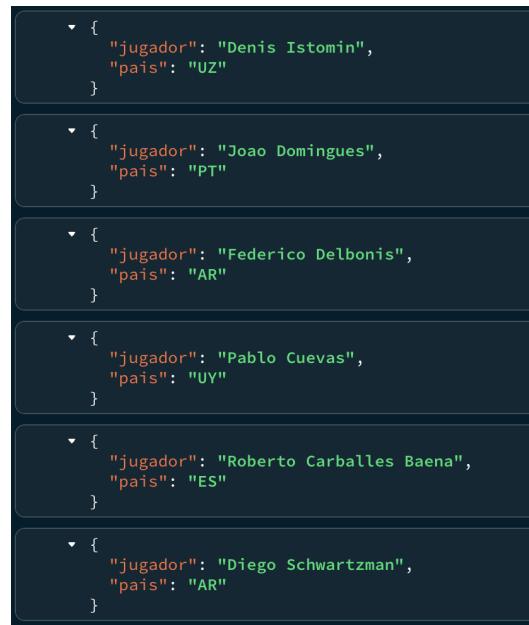


Fig. 38. Bases de datos NoSQL. MongoDB, consulta 5.

D. Preferencias y compromiso de lectura

Procedemos ahora a comprobar si el *cluster* sigue funcionando al retirar nodos. Para probar esto, creamos una consulta de prueba que fuerce a MongoDB a emplear datos de todos los *shards*. La consulta que planteamos agrega el total de partidos con desenlace normal:

```
db.partidos.aggregate([
  {$match: { desenlace: "N" } },
  {$count: "partidos_con_desenlace_normal"},
]);
```

Procedemos a parar todos los procesos de MongoDB en un nodo como ya hemos visto anteriormente, y comprobamos que la consulta sigue ejecutándose correctamente. Al dejar un solo nodo activo, la consulta ya no puede ejecutarse y nos devuelve un error (ver figura 39).

```
> db.partidos.aggregate([
  {$match: { desenlace: "N" } },
  {$count: "partidos_con_desenlace_normal"},
]);
✖ > MongoServerError[FailedToSatisfyReadPreference]: Could not find host matching read preference { mode: "primary" } for set rsShard1
[mongos] tenis>
```

Fig. 39. Error en MongoDB al retirar nodos.

Comprobamos cuál es la preferencia de lectura, que por defecto está configurada para leer desde el primario, y el compromiso de lectura, por defecto, lectura local sin garantía de leer los datos más recientes (figura 40).

```
> db.getMongo().getReadPref();
< ReadPreference {
  mode: 'primary',
  tags: undefined,
  hedge: undefined,
  maxStalenessSeconds: undefined,
  minWireVersion: undefined
}
```

(a) Preferencias de lectura en nuestro *cluster*.

```
> db.adminCommand(
  {
    getDefaultRWConcern: 1
  }
)
< {
  defaultReadConcern: { level: 'local' },
  defaultWriteConcern: { w: 'majority', wtimeout: 0 },
  defaultWriteConcernSource: 'implicit',
  defaultReadConcernSource: 'implicit',
  localUpdateWallClockTime: 2024-12-22T17:58:38.537Z,
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1734890318, i: 1 }),
    signature: {
      hash: Binary.createFromBase64('p75Fz1KurkADw6CVCx8urGAKF08='),
      keyId: Long('7450268241231872025')
    },
    operationTime: Timestamp({ t: 1734890318, i: 1 })
  }
}
[mongos] tenis>
```

(b) Compromiso de lectura en nuestro *cluster*.

Fig. 40. Verificación de nodos del *cluster*.

Podemos concluir que uno de los nodos eliminados era el primario, y que el nodo que queda vivo es un secundario que no ha sido capaz de promocionarse a primario, ya que no quedan nodos vivos con los cuales comunicarse para establecer una votación.

Si cambiamos las preferencias de lectura a cualquiera que no obligue a leer del primario, las consultas volverán a funcionar (figura 41).

```

> db.getMongo().setReadPref('primaryPreferred');
db.partidos.aggregate([
  { $match: { desenlace: "N" } },
  { $count: "partidos_con_desenlace_normal" }
]);
< {
  partidos_con_desenlace_normal: 92151
}
> db.getMongo().setReadPref('secondary');
db.partidos.aggregate([
  { $match: { desenlace: "N" } },
  { $count: "partidos_con_desenlace_normal" }
]);
< {
  partidos_con_desenlace_normal: 92151
}
> db.getMongo().setReadPref('secondaryPreferred');
db.partidos.aggregate([
  { $match: { desenlace: "N" } },
  { $count: "partidos_con_desenlace_normal" }
]);
< {
  partidos_con_desenlace_normal: 92151
}
> db.getMongo().setReadPref('nearest');
db.partidos.aggregate([
  { $match: { desenlace: "N" } },
  { $count: "partidos_con_desenlace_normal" }
]);
< {
  partidos_con_desenlace_normal: 92151
}
> db.getMongo().setReadPref('primary');
db.partidos.aggregate([
  { $match: { desenlace: "N" } },
  { $count: "partidos_con_desenlace_normal" }
]);
<
  partidos_con_desenlace_normal: 92151
}

MongoServerError[FailedToSatisfyReadPreference]: Could not find host matching read preference { mode: "primary" } for set rsShard2

```

Fig. 41. Resultados de consulta de prueba.

Fijando la preferencia de lectura en **primaryPreferred**, procedemos ahora a ver cómo afecta el compromiso de lectura a la posibilidad de ejecutar la consulta.

```

> db.partidos.aggregate([
  { $match: { desenlace: "N" } },
  { $count: "partidos_con_desenlace_normal" }
], { readConcern: { level: "majority" } });
< {
  partidos_con_desenlace_normal: 92151
}
> db.partidos.aggregate([
  { $match: { desenlace: "N" } },
  { $count: "partidos_con_desenlace_normal" }
], { readConcern: { level: "linearizable" } });
MongoServerError[HostUnreachable]: cannot satisfy linearizable read concern on non-primary node
> db.partidos.aggregate([
  { $match: { desenlace: "N" } },
  { $count: "partidos_con_desenlace_normal" }
], { readConcern: { level: "available" } });
< {
  partidos_con_desenlace_normal: 92151
}
> db.partidos.aggregate([
  { $match: { desenlace: "N" } },
  { $count: "partidos_con_desenlace_normal" }
], { readConcern: { level: "snapshot" } });
Error: Async script execution was interrupted
> db.partidos.aggregate([
  { $match: { desenlace: "N" } },
  { $count: "partidos_con_desenlace_normal" }
], { readConcern: { level: "local" } });
< {
  partidos_con_desenlace_normal: 92151
}

```

Fig. 42. Efecto del compromiso de lectura al ejecutar la consulta.

El compromiso de lectura linearizable, que nos garantiza leer los datos más recientes, falla, ya que este solo puede especificarse para operaciones de lectura desde el primario. También falla el compromiso de lectura snapshot, que requiere confirmar que los datos están en un estado consistente, cosa imposible con un solo nodo secundario, dado que no existe el *quorum* necesario para garantizar esta consistencia.

5. BASES DE DATOS NOSQL: GRAFOS (NEO4J)

Neo4J viene ya instalado en la máquina virtual que estamos usando, por lo que simplemente debemos arrancar el servicio con el comando `sudo systemctl start neo4j`. Trabajaremos desde la interfaz *web* en `http://localhost:7474`. Esta nos da celdas para introducir consultas en Cypher, el lenguaje de Neo4J, y proporciona una visualización gráfica de los resultados (si lo permiten) y en forma de tablas.

A. Creación de índices

Lo primero es crear la estructura de nodos y enlaces adecuada para las consultas. Iremos describiéndola poco a poco, pero antes introduciremos los índices necesarios para que las consultas pensadas sean eficientes:

```
CREATE INDEX IF NOT EXISTS FOR (j:Jugador) ON (j.id);
CREATE INDEX IF NOT EXISTS FOR (et:EdicionTorneo) ON (et.torneo, et.fecha);
CREATE INDEX IF NOT EXISTS FOR (p:Partido) ON (p.num_partido, p.fecha);
CREATE INDEX IF NOT EXISTS FOR (p:País) ON (p.codigo_iso2);
CREATE INDEX IF NOT EXISTS FOR (t:Torneo) ON (t.id);
```

La indexación de las originales claves primarias en las tablas de relacionales resulta en consultas mucho más eficientes. Esto lo hacemos para Jugador, EdicionTorneo, País y Torneo. Para facilitar la búsqueda de partidos específicos, se indexan simplemente el número de partido y la fecha.

B. Carga de datos con JDBC

Al estar los datos almacenados dentro de un SGBD relacional, usaremos JDBC para acceder directamente al gestor y recuperar los datos. A continuación, vamos creando la base de datos de grafos a la vez que cargamos los datos de la base de datos relacional. Iremos describiendo poco a poco los nodos y relaciones que compondrán la nueva BD. Comenzamos creando nodos para los países, con sus correspondientes atributos de la tabla relacional.

```
WITH "jdbc:postgresql://localhost:5432/tenis?user=alumnogreibd&password=greibd2021" AS url
CALL apoc.load.jdbc(url, "SELECT codigo_iso2, codigo_iso3, codigo_ioc, nombre FROM pais") YIELD row
CREATE (p:País {
    codigo_iso2: row.codigo_iso2,
    codigo_iso3: row.codigo_iso3,
    codigo_ioc: row.codigo_ioc,
    nombre: row.nombre
});
```

Creamos nodos para los jugadores, con sus correspondientes atributos de la tabla relacional. Se crea una relación `REPRESENTA_A` entre el nodo del país y el nodo del jugador, para indicar que el jugador representa a su país. Para ello, se usa `MATCH` para encontrar el nodo del país correspondiente al jugador, se crea el nodo del jugador y, posteriormente, se crea la relación (saliente desde el nodo jugador hacia el nodo país).

```
WITH "jdbc:postgresql://localhost:5432/tenis?user=alumnogreibd&password=greibd2021" AS url
CALL apoc.load.jdbc(url,
    "SELECT id, nombre, apellido, diestro, fecha_nacimiento, pais, altura FROM jugador") YIELD row
MATCH (pa:País {codigo_iso2: row.pais})
CREATE (j:Jugador {
    id: row.id,
    nombre: row.nombre,
    apellido: row.apellido,
    diestro: row.diestro,
    fecha_nacimiento: row.fecha_nacimiento,
    altura: row.altura
})
CREATE (j)-[:REPRESENTA_A]->(pa);
```

Nodos para los torneos, con sus correspondientes atributos de la tabla relacional. Se crea una relación SE_CELEBRA_EN entre el nodo del país y el nodo del torneo, para indicar que el torneo se celebra en un país. Para ello, se usa MATCH para encontrar el nodo del país correspondiente al torneo, se crea el nodo del torneo y, posteriormente, se crea la relación (saliente desde el nodo torneo hacia el nodo país).

```
// cargamos los torneos
WITH "jdbc:postgresql://localhost:5432/tenis?user=alumnogreibd&password=greibd2021" AS url
CALL apoc.load.jdbc(url, "SELECT id, nombre, pais FROM torneo") YIELD row
CREATE (t:Torneo {
    id: row.id,
    nombre: row.nombre
})
WITH t, row
OPTIONAL MATCH (pa:Pais {codigo_iso2: row.pais})
WITH t, pa
WHERE pa IS NOT NULL
CREATE (t)-[:SE_CELEBRA_EN]->(pa);
```

Nodos para las ediciones de los torneos, con sus correspondientes atributos de la tabla relacional. Se crea una relación EDICION_DE entre el nodo de la edición del torneo y el nodo del torneo, para indicar que la edición del torneo pertenece a un cierto torneo. Para ello, se usa MATCH para encontrar el nodo del torneo correspondiente a la edición del torneo, se crea el nodo de la edición del torneo y, posteriormente, se crea la relación (saliente desde el nodo edición del torneo hacia el nodo torneo).

```
// cargamos las ediciones de los torneos
WITH "jdbc:postgresql://localhost:5432/tenis?user=alumnogreibd&password=greibd2021" AS url
CALL apoc.load.jdbc(url, "SELECT torneo, fecha, superficie, tamano, nivel FROM edicion_torneo") YIELD row
MATCH (t:Torneo {id: row.torneo})
CREATE (et:EdicionTorneo {
    fecha: row.fecha,
    superficie: row.superficie,
    tamano: row.tamano,
    nivel: row.nivel,
    torneo: row.torneo
})
CREATE (et)-[:EDICION_DE]->(t);
```

A continuación, cargamos los partidos creando nodos con sus correspondientes atributos de la tabla relacional. Esta creación dio varios problemas debido al conflicto con el *left join* que comentamos en la sección de agregados; con una carga estándar se omitían partidos, y por tanto la consulta 4 no devolvía los resultados esperados. Para solucionar esto, se ha optado por crear primero el nodo del partido y, posteriormente, se crean las relaciones con los jugadores, torneos y ganadores/perdedores. La creación de las mismas la haremos con un OPTIONAL MATCH para que se cree la relación si existe el nodo al que se quiere enlazar, pero en caso negativo, se incluya el resto de datos del partido igualmente. Además, se ha añadido un LIMIT para evitar problemas de rendimiento: la carga se hizo de 10000 en 10000, cambiando el OFFSET en cada carga.

Vamos un poco más en detalle con las relaciones:

- SE_JUEGA_EN: Relación desde el nodo del partido hacia el nodo del torneo, para indicar que el partido se juega en un cierto torneo. Esta relación se crea con un OPTIONAL MATCH para que se cree la relación si existe el nodo del torneo al que se quiere enlazar, pero en caso negativo, se incluya el resto de datos del partido igualmente.
- GANADO_POR: Relación desde el nodo del partido hacia el nodo del jugador, para indicar que el partido ha sido ganado por un cierto jugador. Esta relación se crea con un OPTIONAL MATCH para que se cree la

relación si existe el nodo del jugador al que se quiere enlazar, pero en caso negativo, se incluya el resto de datos del partido igualmente. Además, esta relación contiene como atributos las estadísticas del ganador en ese partido.

- PERDIDO_POR: Relación completamente análoga a la anterior, pero para el perdedor del partido.

```
// cargamos los partidos
WITH "jdbc:postgresql://localhost:5432/tenis?user=alumnogreibr&password=greibd2021" AS url
CALL apoc.load.jdbc(url,
"SELECT * FROM partido p LIMIT 10000 OFFSET 0") YIELD row

CREATE (p:Partido {
num_partido: row.num_partido,
fecha: row.fecha,
num_sets: row.num_sets,
ronda: row.ronda,
desenlace: row.desenlace,
torneo_id: row.torneo
})

// vinculamos con el torneo si existe
WITH p, row
OPTIONAL MATCH (t:Torneo {id: row.torneo})
WITH p, row, t
WHERE t IS NOT NULL
CREATE (p)-[:SE_JUEGA_EN]->(t)

// vinculamos con el ganador si existe
WITH p, row
OPTIONAL MATCH (ganador:Jugador {id: toInteger(row.ganador)})
WITH p, row, ganador
WHERE ganador IS NOT NULL
CREATE (p)-[:GANADO POR {
num_aces: row.num_aces_ganador,
num_dob_faltas: row.num_dob_faltas_ganador,
num_ptos_servidos: row.num_ptos_servidos_ganador,
num_primeros_servicios: row.num_primeros_servicios_ganador,
num_primeros_servicios_ganados: row.num_primeros_servicios_ganados_ganador,
num_segundos_servicios_ganados: row.num_segundos_servicios_ganados_ganador,
num_juegos_servidos: row.num_juegos_servidos_ganador,
num_break_salvados: row.num_break_salvados_ganador,
num_break_afrontados: row.num_break_afrontados_ganador
}]->(ganador)

// vinculamos con el perdedor si existe
WITH p, row
OPTIONAL MATCH (perdedor:Jugador {id: toInteger(row.perdedor)})
WITH p, row, perdedor
WHERE perdedor IS NOT NULL
CREATE (p)-[:PERDIDO POR {
num_aces: row.num_aces_perdedor,
num_dob_faltas: row.num_dob_faltas_perdedor,
num_ptos_servidos: row.num_ptos_servidos_perdedor,
num_primeros_servicios: row.num_primeros_servicios_perdedor,
num_primeros_servicios_ganados: row.num_primeros_servicios_ganados_perdedor,
num_segundos_servicios_ganados: row.num_segundos_servicios_ganados_perdedor,
num_juegos_servidos: row.num_juegos_servidos_perdedor,
num_break_salvados: row.num_break_salvados_perdedor,
num_break_afrontados: row.num_break_afrontados_perdedor
}]->(perdedor);
```

Por último, cargamos los sets de los partidos con los atributos de la tabla relacional homónima. Se crea una relación PERTENECE_A desde el nodo del set del partido hacia el nodo del partido, para indicar que el set pertenece a un cierto partido. Para ello, se usa OPTIONAL MATCH para encontrar el nodo del partido correspondiente al set del partido, y, si se encuentra, crea el nodo del set del partido y, posteriormente, se crea la relación. La carga de estos datos también la haremos de 10000 en 10000 para evitar problemas de rendimiento.

```
// cargamos los sets de los partidos
WITH "jdbc:postgresql://localhost:5432/tenis?user=alumnogreibd&password=greibd2021" AS url
CALL apoc.load.jdbc(url,
"SELECT * FROM sets_partido LIMIT 10000 OFFSET 0") YIELD row
OPTIONAL MATCH (p:Partido {num_partido: row.num_partido, fecha: row.fecha})
WITH row, p
WHERE p IS NOT NULL
CREATE (sp:SetPartido {
num_set: row.num_set,
juegos_ganador: row.juegos_ganador,
juegos_perdedor: row.juegos_perdedor,
puntos_tiebreak_perdedor: row.puntos_tiebreak_perdedor
})
CREATE (sp)-[:PERTENECE_A]->(p);
```

C. Consultas

Con la base de datos de grafos ya creada, podemos realizar las mismas consultas que llevamos realizando durante todo el documento. Para aligerar la cantidad de texto en las explicaciones posteriores, vamos a comentar algunos detalles de la sintaxis del código, comunes a todas las consultas. De este modo, en la descripción de cada consulta podremos dar un enfoque menos técnico (al conocer ya de antemano esos detalles) y más semántico.

- Para indicar una relación entre dos nodos, usamos `->`. Con el signo `->` indicamos la dirección de la relación, es decir, con `->` indicamos que la relación va desde el nodo de la izquierda hacia el de la derecha, y con `<-` indicamos que la relación va desde el nodo de la derecha hacia el de la izquierda.
- La relación entre ambos nodos la especificaremos entre corchetes, como por ejemplo `-[:RELACION]->`, y podemos agregar un alias en caso de que queramos referirnos a la relación, por ejemplo `-[r:RELACION]->`. En caso de ser una relación con atributos, podemos añadirlos entre llaves, como por ejemplo `-[:RELACION {atributo1: valor1, atributo2: valor2}]->`.
- Los nodos los especificamos entre paréntesis, por ejemplo `(n)`. Si queremos hacer referencia a ello, podemos añadir un alias, por ejemplo `(n)`. En caso de querer referirnos a un tipo de nodo concreto, añadimos el tipo, por ejemplo `(n:TipoNodo)`.
- La estructura básica de las consultas comenzará con un MATCH para buscar patrones en el grafo, seguirá con un WHERE para filtrar los resultados (como hacíamos en SQL), y finalmente un RETURN para mostrar los resultados. En varias consultas queremos hacer cálculos sobre los resultados; para ello usamos WITH, que nos permite pasar los resultados de una cláusula a otra. Al igual que en SQL, podemos usar ORDER BY para ordenar los resultados.

Con todo esto, si queremos, por ejemplo, buscar patrones de partidos ganados por un jugador en cierto torneo, usaremos un patrón de búsqueda en el MATCH de tipo

$$(j:Jugador)<-[:GANADO POR]-(p:Partido)-[:SE JUEGA EN]->(t:Torneo)$$

Usamos las relaciones ya creadas, un partido es ganado por un jugador (relación desde el partido hacia el jugador) y se juega en un torneo (relación desde el partido hacia el torneo). Especificamos un alias para cada

nodo e indicamos el tipo de nodo entre los paréntesis.

Tras esta breve introducción de los detalles sintácticos más básicos de este lenguaje, mostramos las consultas en Cypher que hemos realizado en Neo4J para obtener los mismos resultados que en las consultas SQL.

C.1. Muestra todos los ganadores del torneo “Wimbledon” (Nombre apellido y año). Ordena el resultado por año.

Para esta consulta, curiosamente, nos sirve el ejemplo que pusimos antes. Queremos buscar jugadores que hayan ganado el partido de la ronda final de un torneo concreto, Wimbledon. Para esto, partimos de nodos tipo Partido y buscamos los nodos de tipo Jugador y de tipo Torneo que se relacionen con ese partido mediante las relaciones GANADO_POR y SE_JUEGA_EN, respectivamente. Al buscar este patrón, debemos filtrar o seleccionar solo los nodos de partidos que correspondan a rondas finales, y el nodo de torneo Wimbledon; esto lo hacemos con el where, como en SQL.

Con el RETURN mostramos el nombre y apellido de los jugadores que cumplen con el patrón, así como el año de la edición del torneo en el que ganaron. Para obtener el año, convertimos la fecha a string, con `toString()`, y nos quedamos con los 4 primeros caracteres de esa cadena de texto, que son el año; esto lo conseguimos con `substring`, fijando como carácter inicial el 0 y final el 3 (4 no incluido). Finalmente, ordenamos el resultado de forma ascendente con el `ORDER BY` año. A continuación se muestra la consulta en Cypher y el resultado en la figura 43.

```

MATCH (j:Jugador)-[gp:GANADO_POR]-(p:Partido)-[:SE_JUEGA_EN]->(t:Torneo)
WHERE t.nombre = 'Wimbledon'
    AND p.ronda = 'F'
RETURN j.nombre AS nombre, j.apellido AS apellido, substring(toString(p.fecha), 0, 4) AS año
ORDER BY año;

```

nombre	apellido	año	Roger	Federer	2006
"Michael"	"Stich"	"1991"	"Roger"	"Federer"	"2007"
"Andre"	"Agassi"	"1992"	"Rafael"	"Nadal"	"2008"
"Pete"	"Sampras"	"1993"	"Roger"	"Federer"	"2009"
"Pete"	"Sampras"	"1994"	"Rafael"	"Nadal"	"2010"
"Pete"	"Sampras"	"1995"	"Novak"	"Djokovic"	"2011"
"Richard"	"Krajicek"	"1996"	"Roger"	"Federer"	"2012"
"Pete"	"Sampras"	"1997"	"Andy"	"Murray"	"2013"
"Pete"	"Sampras"	"1998"	"Novak"	"Djokovic"	"2014"
"Pete"	"Sampras"	"1999"	"Novak"	"Djokovic"	"2015"
"Pete"	"Sampras"	"2000"	"Andy"	"Murray"	"2016"
"Goran"	"Ivanisevic"	"2001"	"Roger"	"Federer"	"2017"
"Lleyton"	"Hewitt"	"2002"	"Novak"	"Djokovic"	"2018"
"Roger"	"Federer"	"2003"	"Novak"	"Djokovic"	"2019"
"Roger"	"Federer"	"2004"	"Novak"	"Djokovic"	"2021"
"Roger"	"Federer"	"2005"	"Carlos"	"Alcaraz"	"2023"

Fig. 43. Modelo de grafos en Neo4J, consulta 1. Al no caber la visualización de la tabla completa en la pantalla, se recortó la imagen por la mitad y se compone en dos partes para mostrar la tabla completa.

C.2. Muestra los años en los que Roger Federer ganó algún torneo de nivel Gran Slam (G) o Master 1000 (M). Para cada año, muestra el número de torneos y lista sus nombres (ordenados por la fecha de celebración). Ordena el resultado por el año

Comenzamos buscando los nodos de tipo Partido que se relacionen con los nodos de tipo Jugador y Torneo mediante las relaciones GANADO_POR y SE_JUEGA_EN, respectivamente. Además, estamos interesados en las distintas ediciones de un mismo torneo, por lo que buscamos los nodos de tipo EdicionTorneo que se relacionen con los nodos de tipo Torneo mediante la relación EDICION_DE. Concretamente, filtraremos los nodos de partidos que correspondan a rondas finales, y los nodos de ediciones de torneos que tengan un nivel de 'G' o 'M'. Además, nos aseguramos de que la fecha de la edición del torneo sea la misma que la fecha del partido.

Hasta aquí tendríamos ya todo lo necesario para devolver los datos que se piden, pero necesitamos agrupar los resultados por año y ordenar, dentro de un mismo año, los torneos según su fecha de celebración. Para esto vamos fijando resultados intermedios:

- La agrupación por año se hace con el primer WITH, donde obtenemos el año a partir de la fecha de celebración del partido (como hicimos en la consulta anterior). Tras esto, collect() nos permite agrupar los torneos de un mismo año (al haber realizado el WITH antes). Para ordenarlos por fecha, guardamos tanto los nombres como las fechas de los partidos.
- El conjunto de torneos de cada año se almacena en una colección y, para operar con ella, debemos deshacerla en filas; esto lo conseguimos con UNWIND. Como paso intermedio, nos quedamos con el año, el nombre y la fecha de cada torneo, y ahora sí podemos hacer un ORDER BY sobre la fecha de celebración de cada torneo. Decidimos usar UNWIND ya que, en este caso, simplificaba la declaración de los campos a devolver en el RETURN, pero se podría haber prescindido de él. Veremos un ejemplo en la consulta 3.

Tras este ordenamiento usamos RETURN para mostrar el año, el número de torneos (como un conteo de los nombres distintos) y los nombres de los torneos ganados por Federer ese año, que se concatenan en una sola cadena con reduce().

Para la concatenación usamos una función de reducción (sí, un poco avanzado para el nivel que estamos manejando en estas consultas, pero es la herramienta más adecuada que encontramos para sustituir al STRING_AGG de SQL). Daremos una explicación detallada de su funcionamiento, ya que la usaremos en otras consultas. Por hacer una analogía, esta función sería como usar una función lambda en Python para condensar un bucle for: agrupamos los nombres de los torneos de un mismo año, ya ordenados, con collect(nombres). Cogemos el primer nombre con head(), lo guardamos en una variable s y guardamos el resto de nombres en una lista tail(). Luego, concatenamos s con el siguiente elemento de x. Esto se repite de forma iterativa gracias al reduce() (que hace el papel del for) que rodea todo este bloque de código. A continuación se muestra la consulta en Cypher y el resultado en la figura 44.

```

MATCH (j:Jugador)-[:GANADO_POR]-(p:Partido)-[:SE_JUEGA_EN]->(t:Torneo)
MATCH (et:EdicionTorneo)-[:EDICION_DE]->(t)
WHERE j.nombre = 'Roger'
    AND j.apellido = 'Federer'
    AND p.ronda = 'F'
    AND et.nivel IN ['G', 'M']
    AND et.fecha = p.fecha
    AND et.torneo = t.id
WITH substring(toString(p.fecha), 0, 4) AS ano,
    collect({nombre: t.nombre, fecha: p.fecha}) AS torneos
UNWIND torneos AS torneo
WITH ano, torneo.nombre AS nombre, torneo.fecha AS fecha
ORDER BY fecha
RETURN ano, count(DISTINCT nombre) AS num_torneos,
    reduce(s = head(collect(nombre)), x IN tail(collect(nombre)) | s + ', ' + x) AS torneos
ORDER BY ano;

```

"ano"	"num_torneos"	"torneos"
"2002"	1	"Hamburg Masters"
"2003"	1	"Wimbledon"
"2004"	6	"Australian Open, Indian Wells Masters, Hamburg Masters, Wimbledon, Canada Masters, US Open"
"2005"	6	"Miami Masters, Indian Wells Masters, Hamburg Masters, Wimbledon, Cincinnati Masters, US Open"
"2006"	7	"Australian Open, Miami Masters, Indian Wells Masters, Wimbledon, Canada Masters, US Open, Madrid Masters"
"2007"	5	"Australian Open, Hamburg Masters, Wimbledon, Cincinnati Masters, US Open"
"2008"	1	"US Open"
"2009"	4	"Madrid Masters, Roland Garros, Wimbledon, Cincinnati Masters"
"2010"	2	"Australian Open, Cincinnati Masters"
"2011"	1	"Paris Masters"
"2012"	4	"Indian Wells Masters, Madrid Masters, Wimbledon, Cincinnati Masters"
"2014"	2	"Cincinnati Masters, Shanghai Masters"
"2015"	1	"Cincinnati Masters"
"2017"	5	"Australian Open, Miami Masters, Indian Wells Masters, Wimbledon, Shanghai Masters"
"2018"	1	"Australian Open"
"2019"	1	"Miami Masters"

Fig. 44. Modelo de grafos en Neo4J, consulta 2.

C.3. Muestra los partidos de semifinales (ronda='SF') y final (ronda = 'F') del torneo de "Roland Garros" del 2018. Para cada partido muestra la ronda, el tipo de desenlace, el nombre y apellidos del ganador y el nombre y apellidos del perdedor y el resultado con el número de juegos del ganador y del perdedor en cada set, y opcionalmente en paréntesis el número de juegos del perdedor en el tie break

Para esta consulta, comenzamos con los nodos de tipo Partido, desde los cuales buscamos relaciones de tipo GANADO_POR y PERDIDO_POR con los nodos de tipo Jugador, ya que queremos la información de ambos participantes. Además, desde estos nodos de partidos buscamos nodos de tipo Torneo mediante la relación SE_JUEGA_EN, y nodos de tipo SetPartido mediante la relación PERTENECE_A (dirigida desde los SetPartido hacia los Partido). Este es el patrón inicial de búsqueda en nuestro grafo. Ahora debemos filtrar los resultados:

- Del tipo Torneo queremos el nodo correspondiente a Roland Garros.
- De los nodos de tipos Partido queremos aquellos que correspondan a las rondas de semifinales y final, y que sean del año 2018.

Con esto tenemos los nodos que queremos, así que con una cláusula WITH nos quedamos con ellos a partir de sus respectivos alias, los ordenamos por el número de set y, en otro WITH, concatenamos los resultados de los juegos de cada set en una sola cadena con el formato adecuado e incluyendo la puntuación del tie break en caso de existir. Finalmente, con el RETURN mostramos la ronda y el desenlace, el nombre y apellidos de ambos jugadores (ganador y perdedor) que, al ser una concatenación de dos valores, podemos hacer de forma más cómoda que con el reduce() de la consulta anterior. No obstante, volvemos a recurrir a la función de reducción para concatenar los resultados de los sets de cada partido.

Para esta consulta podríamos haber recurrido de nuevo a UNWIND, pero esta vez optamos por una forma más directa. Esto nos permite explorar varias formas de realizar una misma tarea, la agrupación ordenada. A continuación se muestra la consulta en Cypher y el resultado en la figura 45.

```

MATCH (jg:Jugador)-> [:GANADO POR] ->(p:Partido) -[:PERDIDO POR] -(jp:Jugador),
(p)-[:SE_JUEGA_EN]->(t:Torneo),
(sp:SetPartido)-[:PERTENECE_A]->(p)
WHERE t.nombre = 'Roland Garros'
    AND p.ronda IN ['SF', 'F']
    AND substring(toString(p.fecha), 0, 4) = '2018'
WITH p, jg, jp, sp
ORDER BY sp.num_set
WITH p, jg, jp,
    collect(sp.juegos_ganador + '-' + sp.juegos_perdedor +
CASE sp.puntos_tiebreak_perdedor
WHEN null THEN ''
ELSE '(' + toString(sp.puntos_tiebreak_perdedor) + ')'
END) as sets
RETURN p.ronda as ronda, p.desenlace as desenlace,
    jg.nombre + ' ' + jg.apellido as ganador,
    jp.nombre + ' ' + jp.apellido as perdedor,
    reduce(s = head(sets), x IN tail(sets) | s + ', ' + x) as resultado
ORDER BY p.fecha;

```

ronda	desenlace	ganador	perdedor	resultado
"SF"	"N"	"Rafael Nadal"	"Juan Martin del Potro"	"6-4, 6-1, 6-2"
"SF"	"N"	"Dominic Thiem"	"Marco Cecchinato"	"7-5, 7-6(10), 6-1"
"F"	"N"	"Rafael Nadal"	"Dominic Thiem"	"6-4, 6-3, 6-2"

Fig. 45. Modelo de grafos en Neo4J, consulta 3.

C.4. Muestra la lista de jugadores españoles (ES) que ganaron algún torneo de nivel Gran Slam (G). Para cada jugador muestra los siguientes datos resumen de todos sus partidos: número de partidos jugados, porcentaje de victorias, porcentaje de aces, porcentaje de dobles faltas, porcentaje de servicios ganados, porcentaje de restos ganados, porcentaje de break points salvados (de los sufridos en contra), porcentaje de break points ganados (de los provocados a favor)

Comenzamos buscando jugadores españoles ganadores de un Grand Slam. Para ello, partimos de los nodos de tipo Jugador y buscamos los nodos de tipo País que se relacionen con ellos mediante la relación REPRESENTA_A. A partir de estos nodos de jugadores, buscamos los nodos de tipo Partido que se relacionen con ellos mediante las relaciones GANADO_POR (relación dirigida desde Partido hacia Jugador). Además, desde estos nodos de partidos buscamos los nodos de tipo Torneo mediante la relación SE_JUEGA_EN, y los nodos de tipo EdicionTorneo que se relacionen con los nodos de Torneo mediante la relación EDICION_DE. Tras marcar el patrón de búsqueda, simplemente filtramos los nodos de partidos correspondientes a rondas finales y de ediciones de torneos de nivel 'G', asegurando la coincidencia de las fechas de los partidos y las ediciones de los torneos. Con esto tenemos los jugadores que queríamos, que guardamos con un WITH para usar ahora en el cálculo de estadísticas.

Para calcular las estadísticas de estos jugadores, comenzamos por buscar el nodo de tipo Jugador correspondiente a cada jugador que hemos encontrado mediante un patrón de búsqueda sencillo donde especificamos el tipo de nodo en el MATCH y filtramos por el id del jugador. Luego buscamos los nodos de partido que se relacionen con el nodo del jugador que estemos considerando mediante la relación GANADO_POR o PERDIDO_POR (ya que queremos todos sus partidos). Necesitamos también las estadísticas de sus rivales, para calcular el número de restos de nuestro jugador, por lo que encajamos un patrón de búsqueda para los nodos de nuestros partidos que se relacionen con los nodos de tipo Jugador mediante las relaciones PERDIDO_POR o GANADO_POR, y los denotaremos por un alias distinto, para mantener las estadísticas de nuestro jugador y del rival separadas.

Esto lo hacemos un OPTIONAL MATCH ya que puede darse el caso de que el rival no esté en la base de datos y sea NULL.

Sacaremos primero como resultado intermedio los valores que nos interesan para las estadísticas, y luego calcularemos las estadísticas que se piden. En el primer WITH nos quedamos con el nombre y apellidos de nuestro jugador, ya concatenado de antes, los nodos de partidos y jugador, la relación de nuestro jugador con el partido y el tipo de esta relación (ya que nos permitirá obtener el porcentaje de victorias), y la relación del rival con el partido. También nos quedamos con algunos valores estadísticos:

- Con el tipo de relación podemos obtener una lista de 1 para los partidos ganados y 0 para los perdidos por nuestro jugador, como hicimos en SQL.
- Nos quedamos con num_aces, num_ptos_servidos, num_dob_faltas, num_primeros_servicios_ganados y num_segundos_servicios_ganados (concretamente la suma de estos dos últimos). Todo esto lo obtenemos fácilmente de la relación de nuestro jugador con cada partido, que denotamos con el alias r.
- Para el resto de valores que necesitamos (que no vamos a explicar, ya que el cálculo de las estadísticas es el mismo que en SQL), usamos un CASE sobre el tipo de la relación de nuestro jugador con el partido, para saber si accedemos a los valores disponibles en la relación de nuestro rival con el partido de tipo PERDIDO_POR o de tipo GANADO_POR.

Con todo esto en el primer WITH, calculamos las estadísticas en el segundo WITH, donde nos quedamos con el nombre y apellido, el conteo de partidos (a partir del número de nodos de tipo Partido que tenga asociados nuestro jugador), y los porcentajes, que calculamos usando sum() sobre los valores que hemos obtenido en el primer WITH y dividiendo por la cantidad adecuada (de nuevo, esto son detalles de cálculo ya explicados, no de la propia consulta en Cypher).

Finalmente, con el RETURN mostramos el nombre y apellidos del jugador, el número de partidos jugados y las estadísticas, redondeadas a un decimal. A continuación se muestra la consulta en Cypher y el resultado en la figura 46.

```
// buscamos jugadores españoles que han ganado finales de Grand Slam (para eliminar duplicados,
sino lo hacemos directamente con las estadísticas)
MATCH (j:Jugador)-[:REPRESENTA_A]->(p:País {codigo_iso2: 'ES'})
MATCH (j)<-[:GANADO_POR]-(partido:Partido)-[:SE_JUEGA_EN]->(t:Torneo)
MATCH (et:EdiciónTorneo)-[:EDICION_DE]->(t)
WHERE partido.ronda = 'F'
    AND et.nivel = 'G'
    AND et.fecha = partido.fecha
    AND et.torneo = t.id
WITH DISTINCT j.id AS id_jugador, j.nombre + ' ' + j.apellido AS jugador

// calculamos las estadísticas de estos jugadores
MATCH (j:Jugador)
WHERE j.id = id_jugador
MATCH (p:Partido)
MATCH (j)<-[:GANADO_POR|PERDIDO_POR]-(p)
// necesitamos las estadísticas de sus rivales
OPTIONAL MATCH (p)-[rp:PERDIDO_POR]->(rival_j:Jugador)
OPTIONAL MATCH (p)-[rg:GANADO_POR]->(rival_g:Jugador)
WITH jugador, p, j, r, type(r) AS tipo, rp, rg,
CASE type(r) WHEN 'GANADO_POR' THEN 1 ELSE 0 END AS es_ganador,
r.num_aces AS aces,
r.num_ptos_servidos AS ptos_servidos,
r.num_dob_faltas AS dobles_faltas,
r.num_primeros_servicios_ganados + r.num_segundos_servicios_ganados AS servicios_ganados,
// restos ganados
```

```

CASE type(r)
WHEN 'GANADO_POR' THEN rp.num_ptos_servidos - rp.num_primeros_servicios_ganados -
rp.num_segundos_servicios_ganados
ELSE rg.num_ptos_servidos - rg.num_primeros_servicios_ganados -
rg.num_segundos_servicios_ganados END AS restos_ganados,
CASE type(r)
WHEN 'GANADO_POR' THEN rp.num_ptos_servidos
ELSE rg.num_ptos_servidos END AS ptos_servidos_rival,
r.num_break_salvados AS breaks_salvados,
r.num_break_afrontados AS breaks_afrontados,
CASE type(r)
WHEN 'GANADO_POR' THEN rp.num_break_afrontados - rp.num_break_salvados
ELSE rg.num_break_afrontados - rg.num_break_salvados END AS breaks_ganados,
CASE type(r)
WHEN 'GANADO_POR' THEN rp.num_break_afrontados
ELSE rg.num_break_afrontados END AS breaks_rival

WITH jugador, count(p) AS partidos,
100.0 * sum(es_ganador) / count(p) AS pcje_victorias,
CASE WHEN sum(ptos_servidos) = 0 THEN 0
ELSE 100.0 * sum(aces) / sum(ptos_servidos) END AS pcje_aces,
CASE WHEN sum(ptos_servidos) = 0 THEN 0
ELSE 100.0 * sum(dobles_faltas) / sum(ptos_servidos) END AS pcje_dobles_faltas,
CASE WHEN sum(ptos_servidos) = 0 THEN 0
ELSE 100.0 * sum(servicios_ganados) / sum(ptos_servidos) END AS pcje_servicios_ganados,
CASE WHEN sum(ptos_servidos_rival) = 0 THEN 0
ELSE 100.0 * sum(restos_ganados) / sum(ptos_servidos_rival) END AS pcje_restos_ganados,
CASE WHEN sum(breaks_afrontados) = 0 THEN 0
ELSE 100.0 * sum(breaks_salvados) / sum(breaks_afrontados) END AS pcje_breaks_salvados,
CASE WHEN sum(breaks_rival) = 0 THEN 0
ELSE 100.0 * sum(breaks_ganados) / sum(breaks_rival) END AS pcje_breaks_ganados
RETURN jugador, partidos,
round(pcje_victorias, 1) AS pcje_victorias,
round(pcje_aces, 1) AS pcje_aces,
round(pcje_dobles_faltas, 1) AS pcje_dobles_faltas,
round(pcje_servicios_ganados, 1) AS pcje_servicios_ganados,
round(pcje_restos_ganados, 1) AS pcje_restos_ganados,
round(pcje_breaks_salvados, 1) AS pcje_breaks_salvados,
round(pcje_breaks_ganados, 1) AS pcje_breaks_ganados
ORDER BY jugador;

```

"jugador"	"partidos"	"pcje_victorias"	"pcje_aces"	"pcje_dobles_faltas"	"pcje_servicios_ganados"	"pcje_restos_ganados"	"pcje_breaks_salvados"	"pcje_breaks_ganados"
"Albert Costa"	648	58.5	4.5	2.6	62.5	39.5	60.3	41.0
"Carlos Alcaraz"	183	79.8	4.6	3.0	65.4	41.6	63.8	41.1
"Carlos Moya"	873	63.9	6.4	3.2	63.4	39.0	62.8	40.5
"Juan Carlos Ferrero"	717	64.3	5.1	2.8	63.7	39.7	62.5	40.1
"Rafael Nadal"	1276	82.3	4.3	2.3	67.3	42.4	66.1	44.9
"Sergi Bruguera"	601	62.9	5.2	2.5	61.9	41.6	59.2	44.9

Fig. 46. Modelo de grafos en Neo4J, consulta 4.

C.5. Lista los jugadores que fueron derrotados (en algún partido del 2018) por el rival de Rafael Nadal de la primera ronda (R128) de Roland Garros de 2018

Comenzamos la consulta buscando el nodo de tipo Jugador correspondiente a Rafael Nadal. Tomamos el resto de nodos de tipo Jugador y buscamos un patrón de nodos de tipo Partido que se relacionen con los nodos de tipo Torneo mediante la relación SE_JUEGA_EN. Filtramos los nodos de partidos correspondientes a la ronda R128 y al año 2018. Además, queremos quedarnos con los nodos de tipo Partido que se relacionen con el nodo de Rafael Nadal mediante las relaciones GANADO_POR o PERDIDO_POR, y con otro nodo de tipo Jugador

mediante la relación opuesta. Nos quedamos con el nodo de este jugador rival.

A continuación, buscamos un patrón de nodos de tipo Partido que se relacionen con el nodo del rival mediante las relaciones GANADO_POR, y con nodos de tipo Jugador mediante la relación PERDIDO_POR. Filtramos los nodos de partidos correspondientes al año 2018. Nos quedamos con el nodo de este jugador perdedor y buscamos el nodo de tipo País con el que se relacione el perdedor mediante la relación REPRESENTA_A.

Finalmente, con el RETURN mostramos el nombre y apellidos concatenados de los jugadores que encajen con el patrón de búsqueda y los filtros anteriores, junto con el código ISO2 del país al que representa. A continuación se muestra la consulta en Cypher y el resultado en la figura 47.

```
// encontramos al rival de Nadal en Roland Garros 2018 R128
MATCH (nadal:Jugador {nombre: 'Rafael', apellido: 'Nadal'})
MATCH (rival:Jugador)
MATCH (p:Partido)-[:SE_JUEGA_EN]->(t:Torneo {nombre: 'Roland Garros'})
WHERE p.ronda = 'R128'
AND substring(toString(p.fecha), 0, 4) = '2018'
AND ((nadal)<-[:GANADO_POR]-(p)-[:PERDIDO_POR]->(rival) OR
(nadal)<-[:PERDIDO_POR]-(p)-[:GANADO_POR]->(rival))

// buscamos los partidos donde este rival perdió en 2018
WITH rival
MATCH (rival)<-[:GANADO_POR]-(derrotas:Partido)-[:PERDIDO_POR]->(perdedor:Jugador)
WHERE substring(toString(derrotas.fecha), 0, 4) = '2018'
MATCH (perdedor)-[:REPRESENTA_A]->(pais:Pais)
RETURN DISTINCT perdedor.nombre + ' ' + perdedor.apellido AS jugador, pais.codigo_iso2 AS pais
ORDER BY jugador;
```

"jugador"	"pais"
"Denis Istomin"	"UZ"
"Diego Schwartzman"	"AR"
"Federico Delbonis"	"AR"
"Joao Domingues"	"PT"
"Pablo Cuevas"	"UY"
"Roberto Carballes Baena"	"ES"

Fig. 47. Modelo de grafos en Neo4J, consulta 5.