1) .
2) .

Para elegir el valor más óptimo de bloque tenemos que tener en cuenta el valor de las tuplas.
Para calcular el valor de las tuplas lo hacemos de la siguiente manera, usando como ejemplo Item:

```
-- Tabla general de ítems
CREATE TABLE Items (
    nombre VARCHAR2(20) PRIMARY KEY,
    rareza VARCHAR2(10) NOT NULL
        CHECK (rareza IN ('Comun','Rara', 'Epica',
'Legendaria')),
    nivelMinUtilizacion NUMBER(3) NOT NULL
        CHECK (nivelMinUtilizacion BETWEEN 0 AND 342),
    intercambiable CHAR(1) NOT NULL
        CHECK (intercambiable IN ('S', 'N'))
);
```

En este caso sería el valor del espacio ocupado por cada campo “nombre” + “rareza” + “nivelMinUtilizacion” + “intercambiable”. Para pasarlo a bytes usaremos el largo de cada campo como valor en bytes, osea : 20 + 10 + 3 + 1 = 34 = t
Consideramos cada carácter como un byte

Pasando en limpio:

Tabla Jugador:

nombre	50
fecha_registro	3 sacado de google el tipo DATE
email	40
contrasena	40
nombrePais	20
cantHoras	5
nombreRegion	20
TOTAL	178

Tabla Personaje

email_Jugador	40
id	5

especie	10
fuerza	10
agilidad	10
inteligencia	10
vitalidad	10
resistencia	10
nivel	10
cantMonedas	10
TOTAL	125

Tabla Personaje_Posee_Habilidades

emailJugador	40
idPersonaje	5
nombreHabilidad	20
TOTAL	65

Tabla Habilidades

nombre	20
nivelMin	3
tipoEnergia	10
clasificacion	10
TOTAL	43

Tabla Personaje_Posee_Items

emailJugador	40
idPersonaje	5
nombreItem	20

equipado	1
TOTAL	67

Tabla Items

nombre	20
rareza	10
nivelMinUtilizacion	3
intercambiable	1
TOTAL	34

Tabla Item_Reliquia

nombre	20
TOTAL	20

Asumimos para seguir el hilo del práctico que el valor de los punteros es de 6 bytes.

Dado que la clase con la tupla más grande es Jugador con 178b podríamos usar cualquiera de los tamaños de los bloques ya que en ningún caso de los valores de bloques planteados la t más grande supera el valor del bloque. Pero por preferencias de que haya varios registros por bloque, pero que los bloques no sean innecesariamente grandes lo cual le sacaría un poco el sentido a la optimización por bloques usaremos el valor de 1024 bytes por bloque.

3) .
Supuestos:
* Valores no dados en la letra

Para la siguiente parte necesitaremos la informacion de que hay 30millones de usuarios

Calculo de archivo Jugadores:

n = 30.000.000
t = 178
B = 1024
f = $1024/178$ = (usando el piso) 5
b = 30.000.000 / 5 = (usando el techo) 6.000.000

Cálculo de archivo Personaje

n = 30.000.000
t = 125
B = 1024
f = $1024/125$ = (usando el piso) 8
b = $30.000.000/8$ = (usando el techo) 3.750.000

Calculo de archivo Items

n = 50 (supuesto)
t = 34
B = 1024
f = $1024/34$ = (usando el piso) 30
b = 50 / 30 = (usando el techo) 2

Calculo de archivo Habilidades:

n = 20 (supuesto)
t = 43
B = 1024
f = $1024/43$ = (usando el piso) 23
b = 20 / 23 = (usando el techo) 1

Calculo de archivo Items_Reliquia

n = 6 (supuesto)
t = 20
B = 1024
f = $1024/20$ = (usando el piso) 51
b = 6/51 = (usando el techo) 1

Cálculo de archivo Personaje_Posee_Habilidades

n = 90.000.000 (asumimos 3 habilidades por personaje promedio)
t = 65
B = 1024
f = $1024/65$ = (usando el piso) 15
b = $90.000.000/15$ = (usando el techo) 6.000.000

Cálculo de archivo Personaje_Posee_Items

n = 150.000.000 (asumimos items habilidades por personaje promedio)
t = 67
B = 1024

f = $1024/67$ = (usando el piso) 15
b = $150.000.000/15$ = (usando el techo) 10.000.000

4)

5) .
Supuestos:
1) En caso de que un subÁrbol tenga Select y Proyección, se asume el costo de la proyección dentro del select.

2) Juntamos los Selects ya que en la realidad el DBMS junta esas consultas, no recorrería el archivo de jugadores 3 veces para hacer 1 select cada vez (como ejemplo).

3) Asumimos que al hacer el select de items su resultado ocupa 1 solo bloque. Ya que coherentemente no deberían haber muchos items de rareza Legendaria

4) Asumimos que hay 2 reliquias de rareza legendaria por coherencia

5) Asumimos que un 60% de los personajes tienen al menos una habilidad de tipo “Magia”

6) De los select de personaje por la letra:
* El 30 % de los usuarios son de países de américa.
● El 10 % de los usuarios se crearon en 2024.
● El personaje del 85 % de los usuarios tiene una agilidad y resistencia menor a 50.

7) Cuando un calculo no da un numero entero redondeamos para arriba el resultado.

Selects y proyecciones:

Tabla Items it:

$\sigma_{rareza='Legendaria'}$ y $\pi_{(nombre)}$
Full scan: 2

Búsqueda Binaria: No es posible, ya que nuestro archivo no está ordenado.

Búsqueda con Índices: No es posible, ya que no tenemos índices implementados hasta el momento. En caso de implementarlo se usaría un **Índices secundarios por no clave**

En Personaje p1:
$\sigma_{especie!='Humano'}$, $\sigma_{agilidad>=50}$, $\sigma_{resistencia>=50}$ y $\pi_{(email\_Jugador)}$

Full scan: 3.750.000

Búsqueda Binaria: No es posible, ya que nuestro archivo no está ordenado.

Búsqueda con Índices: No es posible, ya que no tenemos índices implementados hasta el momento. En caso de implementarlo se usaría un **Índices secundarios por no clave**.

Tabla Habilidades h:

$\sigma_{clasificacion='Magia'}$ y $\pi_{(nombre)}$
Full scan: 1

Búsqueda Binaria: No es posible, ya que nuestro archivo no está ordenado.

Búsqueda con Índices: No es posible, ya que no tenemos índices implementados hasta el momento. En caso de implementarlo se usaría un **Índices secundarios por no clave**.

Tabla Jugador j1:

$\sigma_{region='AMERICA'}$, $\sigma_{EXTRACT(YEAR\ FROM\ fecha\_registro)=2024}$ y $\pi_{(nombre,\ email)}$
Full scan: 6.000.000

Búsqueda Binaria: No es posible, ya que nuestro archivo no está ordenado.

Búsqueda con Índices: No es posible, ya que no tenemos índices implementados hasta el momento. En caso de implementarlo se usaría un **Índices secundarios por no clave**.

Tabla Personaje_Posee_Items ppi:
$\pi_{(emailJugador,nombreItem)}$
Full scan: 10.000.000

Tabla Personaje_Posee_Habilidades pph:
$\pi_{(emailJugador,nombreHabilidad)}$
Full scan: 6.000.000

Joins:

Tablas Items_Reliquias y Items (solo legendarios):

$\bowtie$ ir.nombre = it.nombre

Loops anidados: Costo estimado : $1 + (\frac{1^* 1}{12-2}) \approx 2$

Index Join: No lo podemos aplicar por las mismas razones que en los selects, aunque si quisiéramos asumir el costo de crear el índice temporal sería un **Índices secundarios por clave**

Hash Join : Sí es posible ya que ambas tablas cumplen la pre condición de Equi Join de manera que tienen la misma clave. El costo estimado sería $3 * 1 + 3 * 1 = 6$

Sort-Merge Join: No es posible desde un inicio ya que nuestros archivos no están ordenados, pero si se aplica el algoritmo de ordenación el costo estimado sería la siguiente:
$2 * 1 * (1 + log2(1)) + 2 * 1 * (1 + log2(1)) + (1 + 1) = 2 + 2 + 2 = 6$
.

Tablas Resultado del join anterior y Personaje_Posee_Items ppi:
Tabla intermedia hasta ahora

it.Nombre

recálculo del espacio consumido:

n = 2
t = 20
B = 1024
f = 1024/20 = 51
b = 1

⋈ it.nombre=ppi.nombreItem

Recalculo del espacio ocupado por PPI luego de la proyeccion:

n = 150.000.000 (sigue igual)
t = 60 (ya que no se eliminaron los otros datos)
B = 1024
f = $1024/60$ = (usando el piso) 17
b = $150.000.000/17$ = (usando el techo) 8.823.530

Loops anidados: Costo estimado : $1 + (\frac{1*8.823.530}{12-2}) = 882.354$

Index Join: No lo podemos aplicar por las mismas razones que en los selects, aunque si quisiéramos asumir el costo de crear el índice temporal sería un **Índices secundarios por clave.** Ya que en este caso el DBMS debería dar vuelta el JOIN para que se le haga el índice a la hoja de menor cantidad de bloques.

Hash Join : Sí es posible ya que ambas tablas cumplen la pre condición de Equi Join de manera que tienen la misma clave. El costo estimado sería
$3 * 1 + 3 * 8.823.529 = 8.823.532$

Sort-Merge Join: No es posible desde un inicio ya que nuestros archivos no están ordenados, pero si se aplica el algoritmo de ordenación el costo estimado sería
$2 * 1 * (1 + log2(1)) + 2 * 8.823.529 * (1 + log2(8.823.529)) + (1 + 8.823.529) \approx$
$= 2\,(1 + 0) + 17.647.058 * (1 + 23) + (8.823.530) =$
$= 2 + 17.647.058 * (24) + (8.823.530) = 432.352.924$
*el 23 es un aproximado al valor de $log2(8.823.529)$

Tablas Resultado del join anterior y Personaje p1 (luego de los select):
Tabla intermedia hasta ahora

it.Nombre	ppi.emailJugador

recálculo del espacio consumido:

n = 15.000.000
t = 60
B = 1024
f = 1024/60 = 17
b = 15.000.000/17 = 882.353

⋈ p1.email_Jugador =ppi.emailJugador

Recálculo de Personaje luego de los SELECT y proyección:

n = 30.000.000 * 0,15 (Por letra)= 4.500.000
t = 40 (por la proyección)
B = 1024
f = $1024/40$ = (usando el piso) 25
b = $4.500.000/25$ = (usando el techo) 180.000

Loops anidados: Costo estimado : $180.000 + (\frac{180.000*882.353}{12-2}) = 1.5882534 \times 10^{10}$

Index Join: No lo podemos aplicar por las mismas razones que en los selects, aunque si quisiéramos asumir el costo de crear el índice temporal sería un **Índices secundarios por clave.**

Hash Join : Sí es posible ya que ambas tablas cumplen la pre condición de Equi Join de manera que una misma columna. El costo estimado sería
$3 * 882.353 + 3 * 180.000 = 3.187.059$

Sort-Merge Join: No es posible desde un inicio ya que nuestros archivos no están ordenados, pero si se aplica el algoritmo de ordenación el costo estimado sería

$2 * 882.353 * (1 + log2(882.353)) + 2 * 180.000 * (1 + log2(180.000)) + (882.353 + 180.000) \approx$
$1.764.706 * (1 + 19) + 360.000 * (1 + 17) + 1.062.353 =$
$= 35.294.120 + 6.480.000 + 1.062.353 = 42.836.473$
*el 19 es un aproximado al valor de $log2(882.353)$
*el 19 es un aproximado al valor de $log2(180.000)$

Tablas Resultado del join anterior y Jugador j1 (luego de los select):

Tabla intermedia hasta ahora

it.Nombre	ppi.emailJugador

recálculo del espacio consumido:

n = 15.000.000 * 0,15 (por Letra) = 2.250.000
t = 60
B = 1024
f = 1024/60 = 17
b = 2.250.000/17 = 132.353
⋈ j1.email=p1.email_Jugador

Recalculo de Jugador luego de los SELECT y proyeccion:

n = 30.000.000×0,30×0,10=30.000.000×0,03=900.000 (restricciones de letra)
t = 90 (por la proyeccion)
B = 1024
f = $1024/90$ = (usando el piso) 11
b = $900.000/11$ = (usando el techo) 81.819

Loops anidados: Costo estimado : $81.819 + (\frac{81.819*132.353}{12-2}) = 1.082.980.830$

Index Join: No lo podemos aplicar por las mismas razones que en los selects, aunque si quisiéramos asumir el costo de crear el índice temporal sería un **Índices secundarios por clave.**

Hash Join : Sí es posible ya que ambas tablas cumplen la pre condición de Equi Join de manera que una misma columna. El costo estimado sería
$3 * 81.819 + 3 * 132.353 = 642.516$

Sort-Merge Join: No es posible desde un inicio ya que nuestros archivos no están ordenados, pero si se aplica el algoritmo de ordenación el costo estimado sería

$2 * 81.819 * (1 + log2(81.819)) + 2 * 132.353 * (1 + log2(132.353))$
$+ (81.819 + 132.353) \approx$
$163.638 * (1 + 16) + 264.706 * (1 + 17) + 214.172 =$
$= 7.760.726$
*el 16 es un aproximado al valor de $log2(81.819)$
*el 17 es un aproximado al valor de $log2(132.353)$

Tablas Resultado del join anterior y Personaje_Posee_Habilidades pph (Luego de Proyecciones):

Tabla intermedia hasta ahora

it.Nombre	ppi.emailJugador	j1.Nombre

recálculo del espacio consumido:

n = 2.250.000 * 0,03 (Por letra) = 67.500
t = 110
B = 1024
f = 1024/110 = 9
b = 67.500/9 = 7.500

⋈p1.email_Jugador=pph.emailJugador

Recalculo de PPH luego de la proyeccion:

Cálculo de archivo Personaje_Posee_Habilidades

n = 90.000.000 (asumimos 3 habilidades por personaje promedio)
t = 60
B = 1024
f = $1024/65$ = (usando el piso) 17
b = $90.000.000/17$ = (usando el techo) 5.294.118

Loops anidados: Costo estimado : Loops anidados: Costo estimado :
$7.500 + (\frac{7.500*5.294.118}{12-2}) = 3.970.596.000$

Index Join: No lo podemos aplicar por las mismas razones que en los selects, aunque si quisiéramos asumir el costo de crear el índice temporal sería un **Índices secundarios por clave.**

Hash Join : Sí es posible ya que ambas tablas cumplen la pre condición de Equi Join de manera que una misma columna. El costo estimado sería
$3 * 7.500 + 3 * 5.294.118 = 15.904.854$

Sort-Merge Join: No es posible desde un inicio ya que nuestros archivos no están ordenados, pero si se aplica el algoritmo de ordenación el costo estimado sería
$2 * 7.500 * (1 + log2(7.500)) + 2 * 5.294.118 * (1 + log2(5.294.118)) + (7.500 + 5.294.118) \approx$
$15.000 * (1 + 13) + 10.588.236 * (1 + 22) + 5.301.618 =$
$= 249.041.046$
*el 13 es un aproximado al valor de $log2(7.500)$
*el 22 es un aproximado al valor de $log2(5.294.118)$

Tablas Resultado del join anterior y Habilidades h (luego de los select y proyecciones):
Tabla intermedia hasta ahora

it.Nombre	ppi.emailJugador	j1.Nombre	pph.NombreHabilidad

recálculo del espacio consumido:

n = 67.500
t = 130
B = 1024
f = 1024/130 = 7
b = 67.500/7 = 9.643

⋈ h.nombre = pph.nombreHabilidad

Recalculo de Habilidad luego del select y proyeccion:

n = 7 (supuesto ya que decían que eran en distribución uniforme)
t = 20
B = 1024
f = $1024/20$ = (usando el piso) 51
b = 7 / 51 = (usando el techo) 1

Loops anidados: Costo estimado : Loops anidados: Costo estimado :
$1 + (\frac{1*9.643}{12-2}) \approx 966$

Index Join: No lo podemos aplicar por las mismas razones que en los selects, aunque si quisiéramos asumir el costo de crear el índice temporal sería un **Índices secundarios por clave.**

Hash Join : Sí es posible ya que ambas tablas cumplen la pre condición de Equi Join de manera que una misma columna. El costo estimado sería
$3 * 1 + 3 * 9.643 = 28.932$

Sort-Merge Join: No es posible desde un inicio ya que nuestros archivos no están ordenados, pero si se aplica el algoritmo de ordenación el costo estimado sería
$2 * 1 * (1 + log2(1)) + 2 * 9.643 * (1 + log2(9.643))$
$+ (1 + 9.643) \approx$
$2 * (1 + 0) + 19.286 * (1 + 13) + 1 + 9.644 =$
$= 279.651$
*el **13** es un aproximado al valor de $log2(9.643)$

Ultima proyeccion:

Tabla intermedia hasta ahora

it.Nombre	ppi.emailJugador	j1.Nombre	pph.Nombre Habilidad

recálculo del espacio consumido:

n = 67.500 * 0,6 (Supuesto) = 40.500
t = 130
B = 1024
f = 1024/130 = 7
b = 40.500/7 = 5.786

Planteando el Costo de un POSIBLE plan lógico tenemos lo siguiente:

Selects (todos full scan):
2 + 3.750.000 + 1 + 6.000.000 + 10.000.000 = 19.750.003

Joins(todos loops anidados):
2 + 882.354 + 1.5882534×10^{10} + 1.082.980.830 + 3.970.596.000 + 966 = 20.936.994.152

Última proyección:
5.786

Resultado final: 19.750.003 + 20.936.994.152 + 5.786 = 20.956.755.789

6)

Selects:
Selects y proyecciones:

Todos los índices usados son

Tabla Jugador j1:

$\sigma_{region='AMERICA'}$, $\sigma_{EXTRACT(YEAR\ FROM\ fecha\_registro)=2024}$ y $\pi_{(nombre,\ email)}$
Full scan: 6.000.000

Cálculo del inidice
n = 30.000.000
t = 6 + 23 = 29
B = 1024
f = 1024/29 = 35
b = 30.000.000/35 = 857.143

Eso + # registros que cumplen la condición (3% de 30.000.000) = 1.937.143

1.937.143 + Proyección (full scan) = 3.874.286

En este caso, al usar el índice el costo es menor que con el full scan.

Joins:
Tercer join:
Resultado con Personaje:
$loop\ anidados$: 1.5882534×10^{10}

resultado del segundo join

- Número de tuplas: n=15 000 000

- Bloques ocupados: bo=882 353
  (*tras proyectar y filtrar en pasos anteriores*)

Index join:
Ya que es por clave no ordenado se calcula asì:

n = 4,500,000
t = 46
B = 1024
f = 1024/46 = 22
b = 30.000.000/22 = 204546
X = ⌈logf(Nhojas)⌉
X = 4

C = X + CS (si es por clave CS = 1)
C = X + 1
C = 5

bR + (2.250.000 * C)
1000000 + (2.250.000 * 5) = 12.250.000 < loop anidado por lo tanto este índice sirve

⋈ p1.email_Jugador = pph.emailJugador

Loop anidados:

1.8409632×10^10

n=900000
t=40 +6B=46
f=⌊1024/46⌋=22
#hojas=⌈900000/22⌉=40909
X=⌈log22(40909)⌉=4
CS=1(clave u´nica)
C=X+CS=4+1=5
Entonces:
Coste Index Join = bo + no×C = 132 353+2 250 000×5 = 11 382 353 **<** **1.8409632×10^{10}**, por lo tanto **este índice sirve**.

Quinto join:
Resultado con Personaje_Posee_Habilidades:
Loop anidados: 3.5760816×10^10

n=90000000
t=40 +6 =46
B=1024
f=⌊1024/46⌋=22
#hojas=⌈90000000/22⌉=4090909
X=⌈log22(4090909)⌉=5
CS=1(clave u´nica)
C=X+CS=5+1=6

Entonces:

Coste Index Join = bo + no×C= 5 786 + 67 500 × 6 = **412 500** ≪ **3.5760816×10^{10}**, por lo tanto **este índice sirve**.

7) .

Construccion del arbol B+ del ultimo join:
Sabiendo que B es 1024 si le saco el puntero obligatorio me queda que
B = 1018
n=90000000
t=40 +6 =46
f=⌊1024/46⌋=22 aunque si consideramos la regla de que no puede estar lleno y lo hacemos al 70% entonces f = 15

Nivel	# nodos	# entradas	# hijos (#punteros)
1 (raíz)	1	15	16
2	16	16 nodos, cada uno con 15 entradas => 15*16 = **240 (no llega a albergar** 4.090.909**)**	16 nodos, cada uno con 16 punteros => 16*16 = **256**
3	256	256 nodos, cada uno con 15 entradas => 256*15 = **3850 (no llega a albergar** 4.090.909**)**	256 nodos, cada uno con 16 punteros => 256 *16 = **4.096**
4	4.096	4.096 nodos, cada uno con 15 entradas => 4.096*15 = **61.440**	4.096 nodos, cada uno con 16 punteros => 4.096*16 = **65.536**
5	**65.536**	65.536 nodos, cada uno con 15 entradas => 65.536*15 = **983.040**	65.536 nodos, cada uno con 16 punteros =>65.536*16 = **1.048.576**
6 (nivel hoja)	**1.048.576**	1.048.576 nodos, cada uno con 15 entradas => 1.048.576*15 = **15.728.640**	

Comparando si sirve Con el nuevo número de niveles debido a la regla del 70%

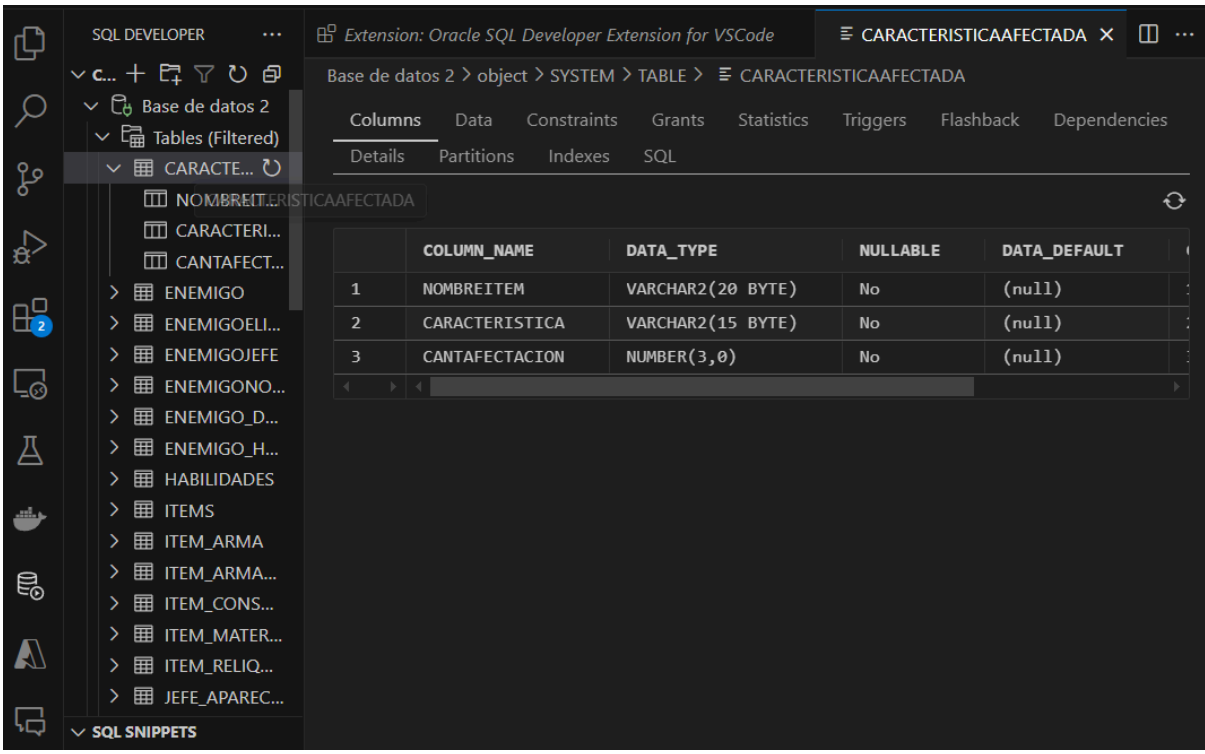
X=6
CS=1(clave u´nica)
C=X+CS=6+1=7

Entonces:

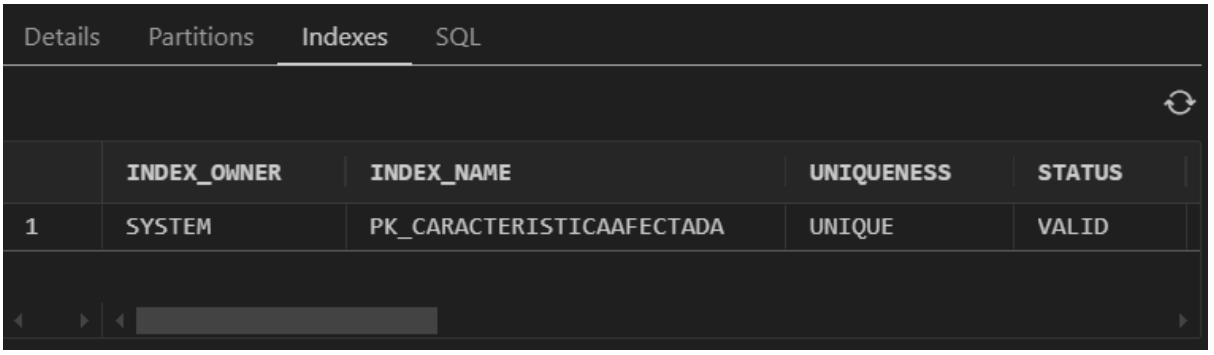
Coste Index Join = bo + no×C= 5 786 + 67 500 × 7 = **478.286** ≪ **3.5760816×10^{10}**, por lo tanto **este índice sirve**.

8) Nosotros para el alcance de este obligatorio utilizamos la extension disponible en VS code de SQL developer (https://marketplace.visualstudio.com/items?itemName=Oracle.sql-developer)

Por lo que para poder cumplir con esta parte podríamos seguir el siguiente procedimiento:
1) Ejecutar el DDL en la carpeta SCRIPTS/Parte1/Parte1.3.sql (con la conexión a la bd previamente establecida con un usuario con permisos)
2) Ir a la extensión -> desplegar la información de la conexión que hayamos establecido -> Tables -> Buscar alguna de las tablas



3) Indexes y ahí ver toda la información pertinente sobre el/los índices creados por el DBMS para cada tabla crear esta.



Pero para no terminar con un eterno documento lleno de capturas de pantallas elegimos realizar un script que nos muestre todo esto de una forma más nemotécnica.

Los pasos para esto serían :
1) Ejecutar el DDL en la carpeta SCRIPTS/Parte1/Parte1.3.sql (con la conexión a la bd previamente establecida con un usuario con permisos)
2) ir al script presente en SCRIPTS/Parte3/Parte3.8.sql y ejecutarlo. Obteniendo el siguiente resultado

	TABLE_NAME	INDEX_NAME	COLUMN_NA
26	MISION_DA_RECOMPENSA	PK_DAREC	CODMISIOI
27	MISION_DA_RECOMPENSA	PK_DAREC	IDRECOMPI
28	MISION_ES_PREVIA_DE_MISION	PK_PREVIAMISION	CODMISIOI
29	MISION_ES_PREVIA_DE_MISION	PK_PREVIAMISION	CODMISIOI
30	MISION_ES_PREVIA_DE_ZONA	PK_PREVIAZONA	CODMISIOI
31	MISION_ES_PREVIA_DE_ZONA	PK_PREVIAZONA	NOMBREZOI
32	PERSONAJE	PK_PERSONAJE	EMAIL_JU
33	PERSONAJE	UQ_PERSONAJE_ID	ID
34	PERSONAJE_POSEE_HABILIDADES	PK_POSEEHAB	EMAILJUG

Ej cual es mucho más fácil de obtener para todas las tablas y con nombre nemotecnicos.

9) .
Explicacion de como obtener el plan lógico:

1) Agregar al incio del codigo presente en Scripts/Parte3/Parte3.1 la siguiente linea:
   EXPLAIN PLAN FOR

```
EXPLAIN PLAN FOR
SELECT j1.nombre, j1.email
FROM JUGADOR j1, PERSONAJE
WHERE j1.email = p1.email_J
AND EXTRACT(YEAR FROM j1.fe
AND j1.nombreRegion = 'AMEF
```

2) Luego ejecutar la consulta
3) finalizada ejecutaremos la siguiente consulta:
   SELECT *
   FROM TABLE(DBMS_XPLAN.DISPLAY (FORMAT=>'ALL +OUTLINE'))

Lo cual nos dará el plan, ponemos la primera foto ya que es largo:

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	125	7 (0)	00:00:
* 1	FILTER					
2	NESTED LOOPS		1	125	3 (0)	00:00:
3	NESTED LOOPS		1	125	3 (0)	00:00:
* 4	TABLE ACCESS FULL	JUGADOR	1	70	2 (0)	00:00:
* 5	INDEX UNIQUE SCAN	PK_PERSONAJE	1		0 (0)	00:00:
* 6	TABLE ACCESS BY INDEX ROWID	PERSONAJE	1	55	1 (0)	00:00:
7	NESTED LOOPS SEMI		1	53	2 (0)	00:00:
* 8	INDEX RANGE SCAN	PK_POSEEHAB	1	34	2 (0)	00:00:
* 9	TABLE ACCESS BY INDEX ROWID	HABILIDADES	1	19	1 (0)	00:00:
* 10	INDEX UNIQUE SCAN	PK_HABILIDADES	1		0 (0)	00:00
11	NESTED LOOPS SEMI		1	65	2 (0)	00:00:
12	NESTED LOOPS		1	53	2 (0)	00:00:

Explicación del plan y comparación con el nuestro:
Evidentemente los datos que tenía este DBMS contra los que teníamos nosotros por letra tienen un mundo de diferencia por lo que el resultado es abrumadoramente distinto al nuestro. A eso hay que sumarle que nosotros no fuimos por el plan más óptimo.

Como se lee:
Por la indentacion de Operation sabemos cual rama es hija de cual
Las Id que tienen asterisco se puede bajar a la tabla de Predicate information donde se explica que hace en esa funcion

```
Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter( EXISTS (SELECT 0 FROM "PERSONAJE_POSEE_HABILID
               WHERE "PPH"."NOMBREHABILIDAD"="H"."NOMBRE" AND
               "PPH"."EMAILJUGADOR"=:B1) AND  EXISTS (SELECT 0
               "IR","PERSONAJE_POSEE_ITEMS" "PPI","ITEMS" "IT"
               "IT"."RAREZA"='Legendaria' AND "PPI"."EMAILJUGA
   4 - filter("J1"."NOMBREREGION"='AMERICA' AND EXTRACT(YEAR
               INTERNAL_FUNCTION("J1"."FECHA_REGISTRO"))=2024)
   5 - access("J1"."EMAIL"="P1"."EMAIL_JUGADOR")
   6 - filter("P1"."ESPECIE"<>'Humano' AND "P1"."AGILIDAD">=5
   8 - access("PPH"."EMAILJUGADOR"=:B1)
```

Poniendo en Orden lo que hace el:
Explicación de su plan lógico.
Para empezar Oracle hace un nested loops entre el resultado de los select en Jugador y el resultado de buscar con un indice como PK de personaje con lo resultante de los select en jugador y a esos jugadores tambien le aplica sus selects.

Luego con las tuplas resultantes (haciendo semi join osea que si encuentra un resultado ya para) de esto usa los datos de personaje para buscar en PPH, con el dato de la habilidad la busca en su índice y con eso va a la tabla Habilidades, y para cada habilidad encontrada de esta manera le aplica los selects

Y por último repite el mismo procedimiento utilizado en PPH y Habilidades pero con PPI , Item Reliquia e Ítem.

Como expliqué previamente el plan de Oracle fue TAN distinto al nuestro debido también a la diferencia entre la información sobre la ocupación de la BD. Oracle empezó con personaje y Jugadores ya que solo tenemos 2 de cada uno cargados en la BD mientras que en la letra eran 30 millones.

La otra diferencia más marcada es la de uso de índices para acelerar la búsqueda en Habilidades e Items una vez que PPI ya otorgaba la PK de estos.