Ejercicio de evaluación P101 plus

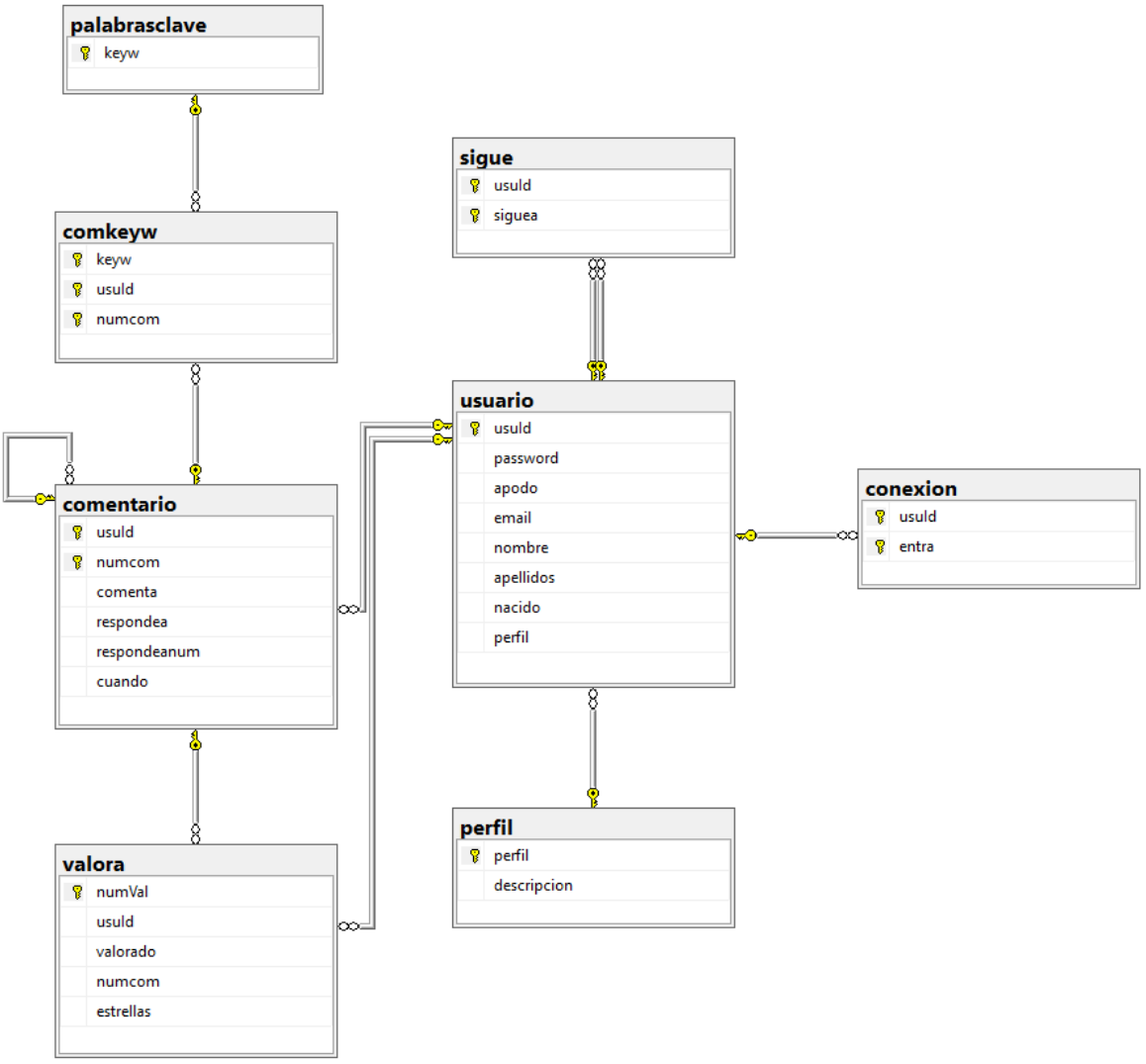
Población aleatoria de una base de datos

Se trata de rellenar unas tablas con datos aleatorio, algunos extraídos de otras tablas. Debes encapsularlo en procedimientos almacenados, con uno de ellos como el principal que llama a todos los demás. Aquí se describe cómo ha de ser la base de datos. Para los datos aleatorios puedes utilizar SQL dinámico.

Puedes necesitar conocer:

1. TOP (select)
2. Función random
3. Función replicate
4. Funciones de cadena
5. Función newid
6. Ordenación aleatoria de una consulta; también aquí; en tablas grandes aquí
7. SQL dinámico: orden EXECUTE; también aquí

La base de datos



La base de datos representa, más o menos, una red social, sin necesitar ser fiel a la realidad. Salvo las tablas PERFIL y PALABRASCLAVE, las demás estarán vacías y tendréis que rellenarlas de datos.

La creación de las tablas y los datos iniciales los encontrarás en este fichero de creación de la base de datos.

Programación adicional

Los requisitos

1. Se asignará perfil 1 al 20 % de los usuarios, y perfil 2 al 5%.
2. Los usuarios pueden seguir a entre 0 y 10 otros usuarios.
3. Un 25% de los usuarios realizan entre 1 y 5 comentarios; el texto del comentario puede ser una constante para todos ellos. Los dos valores anteriores (0.25 y 5) se pasarán como parámetro.
4. Un 30% de los comentarios son respuestas a otros comentarios.
5. Un 25% de los comentarios estarán valorados; el 5% de los usuarios hace esas valoraciones.
6. Cada comentario tendrá entre 0 y 4 palabras clave.
7. Se almacenarán entre 1 y 3 conexiones por usuario.

Desarrollarás el código en lotes a continuación del código del ejercicio P101 anterior, en un único fichero. Eso permitirá reejecutar como si fuera la primera vez cuando se necesite hacer un cambio y comprobar su éxito. Más tarde se puede encapsular ese lote en el procedimiento almacenado.

Modificar perfiles

Debemos asignar perfil 1 al 20 % de los usuarios y perfil 2 al 5%. Esto se puede hacer de muchas maneras, integrándolo en el PA anterior o haciendo uno independiente que se ejecute a continuación. Nosotros lo haremos de esta segunda forma.

Examina la tabla y constata que la columna perfil tiene valor cero por defecto. En toda inserción, si no se proporciona un valor diferente, el sistema asignará el valor por defecto.

```
perfil char(3) default '0'
```

Determinar la cantidad de usuarios actual

La función count(*) cuenta las filas de una tabla, o del resultado de una consulta en general. Utilízala para guardar el valor en una variable *cantidad*.

```
select count(*) from usuario;
```

Modificar perfiles

Queremos modificar el 20% y el 5% de los usuarios, respectivamente, y de forma aleatoria. Si queremos obtener ordenaciones aleatorias:

```
select usuid,perfil from usuario order by newid();
```

Si, de estos, queremos el 20% —de *cantidad*—, usaremos top().

```
select top(@cantidad*.20) usuid,perfil from usuario order by newid();
```

Tendremos un problema con esta consulta por culpa del parámetro de top(), que debe ser un número entero. Soluciónalo.

Para hacer la modificación de datos utilizaremos expresiones de tabla común (*Common Table Expression*, CTE). Simplificando, construiremos una consulta cuyo resultado no se destruya inmediatamente, y que podamos usar en una segunda consulta.

```
WITH cons_temp AS consulta
UPDATE cons_temp…;
```

Si consulta es la orden select anterior, el efecto final es que modificaremos únicamente las filas obtenidas en *consulta*.Fíjate que consulta contiene la clave primaria de USUARIO, con lo que no habrá problema en realizar la actualización.

Por ejemplo, `update usuario set perfil=1` daría el mismo resultado que

```
WITH temp_cons AS select usuid,perfil from usuario
UPDATE temp_usu SET perfil=1
```

Puedes ver más ejemplos en este documento.

Usando CTE y la consulta anterior para restringir la operación al 20% de los usuarios, modifica el perfil. De forma idéntica, por valor 2 a otro 5% de los usuarios, pero sin tener en cuenta al 20% recién modificado.

Esqueleto de la modificación de perfiles

```
crear PA -- sin parámetros
declaración de variables
contar usuarios: (@cantidad)
poner perfil 1 al 20% de @cantidad de usuarios
poner perfil 2 al 5% de @cantidad de usuarios restantes
```

Podemos ahorrarnos el contar filas en la tabla usuario ya que, desde el principio, estamos trabajando con un parámetro *cantusu*, la cantidad de usuarios. Pero, previendo que

Generar seguidores

La tabla SIGUE almacena parejas de identificadores de usuario. La idea es que el primero "sigue" al segundo en la red social.

```
create table sigue (
	usuId int,
	siguea int,
	constraint PK_sigue primary key (usuId,siguea),
	foreign key (usuId) references usuario(usuId),
	foreign key (siguea) references usuario(usuId)
);
```

	usuId	siguea
1	3	28
2	3	52
3	3	91
4	4	42

Lo más sencillo es plantear un bucle que calcule una cantidad aleatoria entre 0 y 10 identificadores de usuario, recordando que *usuId* es un entero.

```
create procedure gen_sigue
as
declare @cuantos int, @usu1 int, @usu2 int;

select @cuantos = count(*) from usuario;
while (@cuantos > 0)
begin
	set @usu1=1;
	set @usu2=1;
	while (@usu1 = @usu2)
	begin
		select @usu1=floor(rand()*@cuantos)+1;
		select @usu2=floor(rand()*@cuantos)+1;
	end

	insert into sigue (usuId,siguea) values(@usu1,@usu2);
	set @cuantos=@cuantos-1;
end
go
```

Sin embargo, no es la forma más correcta de hacerlo, no podemos garantizar por completo que los identificadores estén entre 1 y 100. Además, la probabilidad de provocar un duplicado es alta. Modifica el procedimiento anterior para asegurar que los valores de @usu1 y @usu2 sean consistentes:

1. Que @usu1 y @usu2 almacenen *usuId* reales

2. Que no se produzcan violaciones de clave primaria.

Generación de comentarios

La tabla comentario

```
create table comentario (
	usuId int,
	numcom smallint,
	comenta varchar(1000) not null,
	respondea int,
	respondeanum smallint,
	cuando datetime2,
	constraint PK_comentario primary key (usuId,numcom),
	constraint FK_comentario_usuario foreign key (usuId) references usuario(usuId),
	constraint FK_comentario_comentario foreign key (respondea,respondeanum) references comentario(usuId,numcom)
);
```

	usuId	numcom	comenta	respondea	respondeanum	cuando
1	9	1	Esto es un comentario	NULL	NULL	2017-08-26 00:00:00.0000000
2	9	2	Esto es un comentario	NULL	NULL	2017-12-29 00:00:00.0000000
3	9	3	Esto es un comentario	NULL	NULL	2017-02-04 00:00:00.0000000
4	9	4	Esto es un comentario	NULL	NULL	2017-01-21 00:00:00.0000000
5	9	5	Esto es un comentario	NULL	NULL	2017-12-18 00:00:00.0000000
6	9	6	Esto es un comentario	NULL	NULL	2017-05-08 00:00:00.0000000
7	13	1	Esto es un comentario	NULL	NULL	2017-12-13 00:00:00.0000000
8	13	2	Esto es un comentario	NULL	NULL	2017-10-15 00:00:00.0000000
9	13	3	Esto es un comentario	NULL	NULL	2017-08-21 00:00:00.0000000
10	13	4	Esto es un comentario	NULL	NULL	2017-06-24 00:00:00.0000000
11	17	1	Esto es un comentario	NULL	NULL	2017-11-03 00:00:00.0000000
12	17	2	Esto es un comentario	NULL	NULL	2017-05-28 00:00:00.0000000
13	17	3	Esto es un comentario	NULL	NULL	2017-07-27 00:00:00.0000000
14	17	4	Esto es un comentario	NULL	NULL	2017-04-09 00:00:00.0000000

Esta creación de comentarios la vamos a hacer en dos partes. Primero insertaremos comentarios y después estableceremos que algunos son respuesta a otros comentarios previos.

Parte 1: creación de comentarios

Aquí vamos a utilizar cursores. Este tipo de objetos pertenecen al estándar de SQL y surgieron ante la necesidad del procesamiento de resultados fila a fila. Los cursores hay que declararlos, abrirlos, recorrerlos y, finalmente, cerrarlos y destruirlos.

```
DECLARE micursor CURSOR FOR consulta
OPEN micursor
FETCH NEXT FROM micursor INTO variables
WHILE @@FETCH_STATUS = 0
BEGIN
	...(operaciones con el cursor)
```

```
            FETCH NEXT FROM micursor INTO variables
      END
      CLOSE micursor
      DEALLOCATE micursor
```

En este documento verás un ejemplo.

Esqueleto de la creación de comentarios

```
crear PA gen_comenta(parámetros @cantusu, @maxPorUsu)
declarar variables
declarar cursor sobre @cantusu identificadores de usuario
abrir cursor
Por cada fila del cursor:
      calcular la cantidad de comentarios para el usuario (@cuantos)
      insertar @cuantos comentarios para el usuario
            usuId: el identificador de usuario recuperado del cursor
            numcom: número de comentario, de la variable contador de un bucle
            comenta: siempre 'Esto es un comentario'
            cuando: fecha aleatoria entre '2017-01-01' y '2019-10-10'
cerrar cursor
destruir cursor
```

Antes de ejecutar el PA, recuerda calcular el valor que vas a pasar en *@cantusu*.

Parte 2: modificar comentarios como respuesta

Del resultado anterior, vamos a elegir algunos comentarios y modificarlos dando valor a `(respondea,respondeanum)`.

	usuId	numcom	comenta	respondea	respondeanum	cuando
1	9	1	Esto es un comentario	9	6	2017-08-26 00:00:00.0000000
2	9	2	Esto es un comentario	9	1	2017-12-29 00:00:00.0000000
3	9	3	Esto es un comentario	NULL	NULL	2017-02-04 00:00:00.0000000
4	9	4	Esto es un comentario	13	4	2017-01-21 00:00:00.0000000
5	9	5	Esto es un comentario	NULL	NULL	2017-12-18 00:00:00.0000000
6	9	6	Esto es un comentario	9	4	2017-05-08 00:00:00.0000000
7	13	1	Esto es un comentario	NULL	NULL	2017-12-13 00:00:00.0000000
8	13	2	Esto es un comentario	17	1	2017-10-15 00:00:00.0000000
9	13	3	Esto es un comentario	NULL	NULL	2017-08-21 00:00:00.0000000
10	13	4	Esto es un comentario	17	2	2017-06-24 00:00:00.0000000
11	17	1	Esto es un comentario	43	1	2017-11-03 00:00:00.0000000
12	17	2	Esto es un comentario	43	4	2017-05-28 00:00:00.0000000

Lo que está claro es que los comentarios a los que se puede responder ya están en la tabla COMENTARIO. Podemos volver a usar CTE para crear 2 listas:

1. comentarios que son respuesta
2. comentarios a los que se responde

Lo vamos a hacer sobre el 30% de los comentarios:

```
declare @c int;
select @c=count(*)*.30 from comentario;
```

Ejecuta lo siguiente; suponemos que ese 30% es, en realidad, 15 comentarios.

```
select top(15) usuId,numcom from comentario order by newid()
```

Obtienes 15 comentarios seleccionados aleatoriamente. Para lo que queremos hacer necesitamos numerar las filas. Reutilizamos la expresión anterior como subconsulta:

```
select usuId,numcom,row_number() over (order by usuId) rn
from (select top(15) usuId,numcom from comentario order by newid()) t
```

Si ejecutamos 2 veces obtendríamos 2 listas de comentarios ordenados por un entero secuencial:

usuId	numcom	rn
5	2	1
9	4	2
13	2	3
13	5	4
13	3	5
15	2	6
17	2	7
31	1	8
36	2	9
62	3	10
66	1	11
69	4	12
74	1	13
93	1	14
93	2	15

usuId	numcom	rn
1	2	1
2	1	2
3	3	3
15	1	4
17	1	5
17	2	6
28	5	7
36	5	8
58	3	9
62	4	10
66	1	11
69	1	12
86	1	13
93	3	14
93	2	15

La idea es que, igualando la columna *rn* de cada tabla, convertimos un comentario a respuesta a otro comentario.

usuId	numcom	ra	ran
5	2	1	2
9	4	2	1
13	2	3	3
13	5	15	1
13	3	17	1
15	2	17	2
17	2	28	5
31	1	36	5
36	2	58	3
62	3	62	4
69	4	69	1
74	1	86	1
93	1	93	3

```
select t1.usuId,t1.numcom,t2.usuId ra,t2.numcom ran
from t1, t2
where t1.rn=t2.rn -- emparejar comentarios por rn
--evitar el caso de que un comentario que sea respuesta de sí mismo
and (t1.usuId<>t2.usuId or t1.numcom<>t2.numcom);
```

El problema reside en que, en realidad, no queremos la consulta sino actualizar COMENTARIO con esa información. Para eso necesitamos un update algo más sofisticado.

```
UPDATE comentario set respondea=t2.usuId, respondeanum=t2.numcom
from comentario c
      --comentario que es respuesta
      join  t1 on (c.usuId=t1.usuId and c.numcom=t1.numcom)
      --comentario al que responde
      join  t2 on (t1.rn=t2.rn)
where t1.usuId<>t2.usuId or t1.numcom<>t2.numcom;--que no sean el mismo
```

Adapta lo explicado para su correcta ejecución. Puede hacerse con CTE —preferentemente— o con tablas temporales.

Crear valoraciones

Pide el ejercicio que "hasta un 25% de los comentarios estarán valorados; no más del 5% de los usuarios hace esas valoraciones; el resultado final será la mitad de todos esos usuarios valorando todos esos comentarios". Supongamos que son 10 comentarios y 3 usuarios. Lo primero es seleccionarlos:

	elusuario
1	69
2	92
3	38

	valora	a
1	100	4
2	79	2
3	24	2
4	73	2
5	89	1
6	64	4
7	1	4
8	80	1
9	79	3
10	68	4

Esos 3 usuarios serán los responsables de las valoraciones de esos 10 comentarios. La valoración consistirá en un valor entre 1 y 5 —"estrellas", se supone—. De todas las posibles combinaciones de esas filas, solo queremos la mitad, elegidas aleatoriamente.

	elusuario	valora_a_usu	a_num	con
1	38	68	4	2
2	38	1	4	5
3	92	68	4	3
4	38	24	2	3
5	69	80	1	2
6	38	64	4	5
7	92	1	4	4
8	38	100	4	4
9	38	80	1	4
10	69	79	2	3
11	69	24	2	3
12	92	80	1	4
13	69	79	3	2
14	69	73	2	2
15	92	73	2	5

La columna *con* es la valoración de *elusuario* al comentario (*valora_a_usu,a_num*).

```
crear PA (parámetros @nusu,@ncom)
seleccionar @nusu usuarios
seleccionar @ncom comentarios a ser valorados
insertar en VALORA @nusu*@ncom/2 filas del producto cartesiano de las dos
consultas anteriores.
```

Asignar palabras clave

Se pide asociar a cada comentario entre 0 y 4 palabras clave elegidas de entre la lista almacenada en PALABRASCLAVE. Esta asociación se guarda, a su vez, en la tabla COMKEYW. El máximo de palabras clave será parámetro del PA.

	keyw	usuId	numcom
1	almeria	1	1
2	libro	1	1
3	porno	1	1
4	presentacion	1	1
5	mov	1	2
6	practicas	1	2
7	shopping	1	2
8	.gif	1	4
9	blog	1	4
10	lit	1	4
11	pages	1	4
12	.azw3	8	1
13	dosier	8	1
14	.azw3	10	1
15	proyecto	10	1

La formas más inmediata de conseguirlo es con un cursor sobre los identificadores de comentario y, para cada fila, insertar en COMKEYW el resultado de una consulta que combine el identificador del comentario (usuId,numcom) con los primeros resultados (entre 0 y 4) de un ordenación aleatoria de las palabras clave.

```
crear PA (parámetro @maxpals)
declarar variables
declarar cursor sobre comentario
abrir cursor
leer primera fila del cursor (@usuid,@numcom)
mientras queden filas
insertar (rand()*(@maxpals+1) filas en comkeyw (@usuid,@numcom,keyw) desde
       palabrasclave
leer siguiente fila
cerrar cursor
destruir cursor
```

Crear conexiones

La tabla CONEXION almacena pares (usuario,fecha_hora). Lo único que necesitamos es generar una fecha más o menos aleatoria.

Prueba esto:

```
declare @hoy datetime;
set @hoy=dbo.getRandomDate('20150101','20190930');
select @hoy,
dateadd(day,1+floor(rand()*30),@hoy),
dateadd(month,1+floor(rand()*12),@hoy),
dateadd(year,1+floor(rand()*2),@hoy),
dateadd(hour,1+floor(rand()*24),@hoy),
dateadd(minute,1+floor(rand()*60),@hoy),
dateadd(second,1+floor(rand()*60),@hoy),
dateadd(millisecond,1+floor(rand()*1000),@hoy)
```

Recuerda que `dbo.getRandomDate()` es una función que hemos creado al preparar los datos. `DateAdd()` es una función de T-SQL que añade una cantidad a una parte de un `datetime`.

Solo tienes que combinarlo todo para generar una fecha completa aleatoria. Después, utilízalo para generar las entre 1 y 3 conexiones por usuario.

Combinar todo en un único PA

Para finalizar, queremos un PA único para lanzarlo todo, por ejemplo con 100 usuarios:

exec generar_bd 100

Obviamente, este PA contiene llamadas a los PA previos:

```
create procedure generar_bd (@numusus int)
as
exec generar_usuarios @numusus;
exec cambiar perfiles;
...
```

Puedes comprobar, aparte de otras consultas que tú hagas, la cantidad de filas en cada tabla:

```
select
     (select count(*) from usuario) usuarios,
     (select count(*) from sigue) sigue,
     (select count(*) from comentario) comentarios,
     (select count(*) from comentario where respondea is not null) respuestas,
     (select count(*) from valora) valoraciones,
     (select count(*) from comkeyw) palabras_clave;
      (select count(*) from comkeyw) conexiones;
```