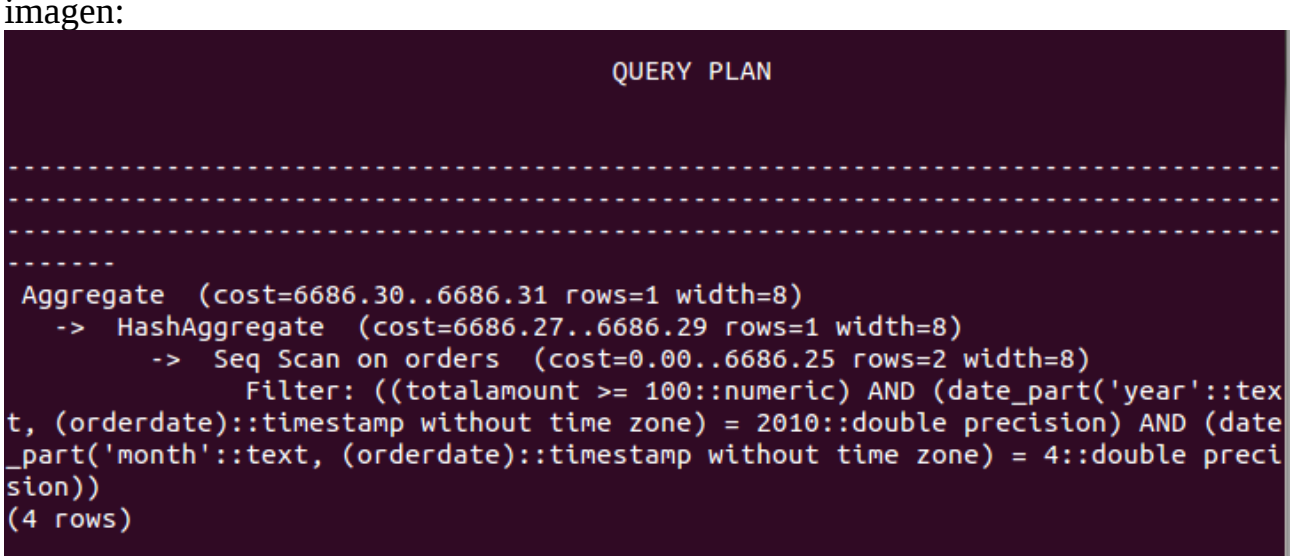
SISTEMAS INFORMATICOS

PRACTICA 4

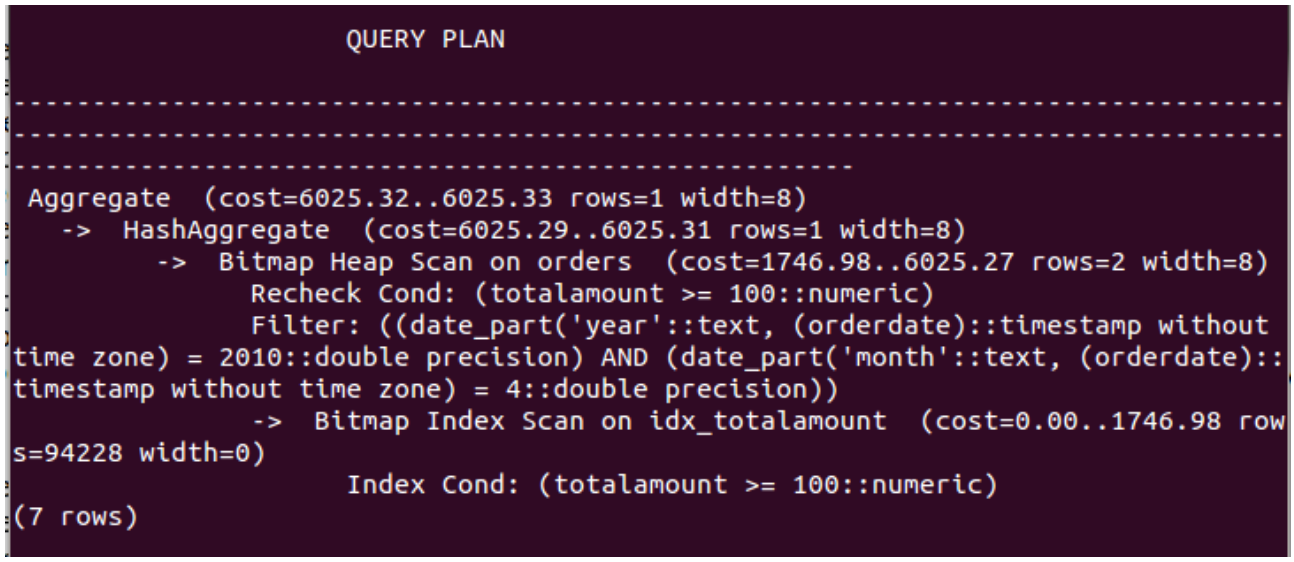
GR:1311
Daniel Garduño Hernandez
Mario Valdemaro, Garcia Roque

Apartado A)
La query solicitada es: select count(num)from (select count(*)as num
from orders
where totalamount >=100
and extract (year from orderdate)=2010 and extract (month from orderdate) =4
group by extract (year from orderdate),extract (month from orderdate) ,customerid)
as a;

El resultado de hacer EXPLAIN sobre esta consulta es el mostrado en la siguiente imagen:



Al aplicar el índice: “CREATE INDEX idx_totalamount on orders(totalamount);” podemos ver que el rendimiento mejora, como se puede apreciar en la siguiente imagen:



Creando índices sobre orderdate y customerid no se mejora el rendimiento.

Apartado B)

Tomaremos los siguientes datos como base para todos los valores de la tabla
Umbral mínimo 100
Intervalo 5
Numero máximo de entradas 1000
Parar si no hay clientes activado
Fecha de inicio enero 2009
Índice es:
CREATE INDEX idx_totalamount on orders(totalamount);

Versiones	Sin índices	Con índice
Prepare	1869,01 ms	154,34 ms
Sin prepare	1844,34 ms	199,92 ms

Como veíamos en el apartado anterior utilizar índices mejora notablemente el tiempo de consulta. También se observa que las sentencias preparadas mejoran con el uso de índices, y, si no los usamos, en general la diferencia entre usar sentencias preparadas y no usarlas es muy leve, teniendo las sentencias preparadas un peor rendimiento en general.

Apartado C)

```
SI_P4=# explain select customerid  from customers where customerid not in
(select customerid from orders where status = 'Paid');
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on customers  (cost=3961.65..4490.81 rows=7046 width=4)
   Filter: (NOT (hashed SubPlan 1))
   SubPlan 1
     ->  Seq Scan on orders  (cost=0.00..3959.38 rows=909 width=4)
           Filter: ((status)::text = 'Paid'::text)
(5 rows)

SI_P4=# explain select customerid from (select customerid from customers
 union all select customerid from orders where status ='Paid') as A
 group by customerid having count(*)=1;
                             QUERY PLAN
---------------------------------------------------------------------
 HashAggregate  (cost=4537.41..4539.91 rows=200 width=4)
   Filter: (count(*) = 1)
   ->  Append  (cost=0.00..4462.40 rows=15002 width=4)
         ->  Seq Scan on customers  (cost=0.00..493.93 rows=14093 width=4)
         ->  Seq Scan on orders  (cost=0.00..3959.38 rows=909 width=4)
               Filter: ((status)::text = 'Paid'::text)
(6 rows)

SI_P4=# explain select customerid from customers  except
select customerid from orders
where status ='Paid';
                                  QUERY PLAN
------------------------------------------------------------------------------
 HashSetOp Except  (cost=0.00..4640.83 rows=14093 width=4)
   ->  Append  (cost=0.00..4603.32 rows=15002 width=4)
         ->  Subquery Scan on "*SELECT* 1"  (cost=0.00..634.86 rows=14093 width=4)
               ->  Seq Scan on customers  (cost=0.00..493.93 rows=14093 width=4)
         ->  Subquery Scan on "*SELECT* 2"  (cost=0.00..3968.47 rows=909 width=4)
               ->  Seq Scan on orders  (cost=0.00..3959.38 rows=909 width=4)
                     Filter: ((status)::text = 'Paid'::text)
(7 rows)
```

En cuanto a tiempos las 3 tienen tiempos muy parecidos, siendo la primera la más rápida y la tercera la más lenta.

Apartado D)

Mediante la sentencia “EXPLAIN” vemos el coste de ejecución de las dos consultas:

```
si1=# explain select count(*)
from orders
where status is null;
                          QUERY PLAN
---------------------------------------------------------------
 Aggregate  (cost=3507.17..3507.18 rows=1 width=0)
   ->  Seq Scan on orders  (cost=0.00..3504.90 rows=909 width=0)
         Filter: (status IS NULL)
(3 rows)
```

```
si1=# explain select count(*)
from orders
where status ='Shipped';
                          QUERY PLAN
---------------------------------------------------------------
 Aggregate  (cost=3961.65..3961.66 rows=1 width=0)
   ->  Seq Scan on orders  (cost=0.00..3959.38 rows=909 width=0)
         Filter: ((status)::text = 'Shipped'::text)
(3 rows)
```

Para ver cómo varía el tiempo, creamos un índice en la tabla orders por la columna status y volvemos a estudiar de nuevo la planificación de las consultas. Como se puede comprobar, el tiempo se reduce más de la mitad.

```
si1=# create index idx_status on orders(status);
CREATE INDEX
si1=# explain select count(*)
from orders
where status is null;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=1496.37..1496.38 rows=1 width=0)
   ->  Bitmap Heap Scan on orders  (cost=19.32..1494.10 rows=909 width=0)
         Recheck Cond: (status IS NULL)
         ->  Bitmap Index Scan on idx_status  (cost=0.00..19.09 rows=909 width=0)
               Index Cond: (status IS NULL)
(5 rows)

si1=# explain select count(*)
from orders
where status ='Shipped';
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=1498.65..1498.66 rows=1 width=0)
   ->  Bitmap Heap Scan on orders  (cost=19.32..1496.37 rows=909 width=0)
         Recheck Cond: ((status)::text = 'Shipped'::text)
         ->  Bitmap Index Scan on idx_status  (cost=0.00..19.09 rows=909 width=0)
               Index Cond: ((status)::text = 'Shipped'::text)
(5 rows)
```

Ahora ejecutamos “ANALYZE” para generar las estadísticas sobre la tabla orders para permitir una mejor optimización de las queries:

```
si1=# ANALYZE orders;
ANALYZE
```

Vemos de nuevo el coste de las consultas, y comprobamos que el tiempo se reduce mucho.

```
si1=# explain select count(*)
from orders
where status is null;
                                QUERY PLAN
---------------------------------------------------------------------------
 Aggregate  (cost=8.29..8.30 rows=1 width=0)
   ->  Index Scan using idx_status on orders  (cost=0.00..8.29 rows=1 width=0)
         Index Cond: (status IS NULL)
(3 rows)

si1=# explain select count(*)
from orders
where status ='Shipped';
                            QUERY PLAN
------------------------------------------------------------------
 Aggregate  (cost=4277.27..4277.28 rows=1 width=0)
   ->  Seq Scan on orders  (cost=0.00..3959.38 rows=127156 width=0)
         Filter: ((status)::text = 'Shipped'::text)
(3 rows)
```

Viendo el resultado de la ejecución de las otras dos consultas:

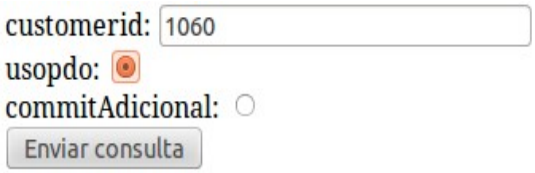
```
si1=# explain select count(*)
from orders
where status ='Paid';
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=2320.30..2320.31 rows=1 width=0)
   ->  Bitmap Heap Scan on orders  (cost=360.97..2274.91 rows=18155 width=0)
         Recheck Cond: ((status)::text = 'Paid'::text)
         ->  Bitmap Index Scan on idx_status  (cost=0.00..356.43 rows=18155 width=0)
               Index Cond: ((status)::text = 'Paid'::text)
(5 rows)

si1=# explain select count(*)
from orders
where status ='Processed';
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=2953.17..2953.18 rows=1 width=0)
   ->  Bitmap Heap Scan on orders  (cost=718.98..2861.97 rows=36479 width=0)
         Recheck Cond: ((status)::text = 'Processed'::text)
         ->  Bitmap Index Scan on idx_status  (cost=0.00..709.86 rows=36479 width=0)
               Index Cond: ((status)::text = 'Processed'::text)
(5 rows)
```

Apartado E):

borraCliente.php: Versión correcta del programa que, mediante una transacción, borra de la base de datos un cliente y toda su información asociada, esto es, sus pedidos y los detalles de los mismos. Una vez que empieza la transacción se ejecutan tres queries de borrado. La primera borra los detalles de los pedidos, de la tabla orderdetail; la segunda borra los pedidos realizados por el cliente; la tercera borra el cliente.

borraClienteMal.php: Versión con fallos del programa. La segunda query que se ejecuta es la de borrar el cliente, por lo que el programa falla, al intentar borrar el cliente sin borrar los pedidos asociados, referenciados por una Foreign Key. Opcionalmente se podrá dar opción al uso de un "commit" intermedio para que, aunque el programa falle, el resultado de la ejecución de la primera query se mantenga, y empiece otra transacción después de este commit, y antes de la ejecución de la segunda query. Al finalizar con error, se ejecuta un "rollback", que devuelve la tabla al estado inicial.

a) Si no usamos el commit intermedio:



Estado de la tabla orderdetail antes de la ejecucion de la query
14077-1444-1060
14077-5717-1060
14075-4306-1060
14076-3783-1060
14076-362-1060

Estado de la tabla orderdetail despues de la ejecucion de la query

SQLSTATE[23503]: Foreign key violation: 7 ERROR: update or delete on table "customers" violates foreign key constraint "orders_customerid_fkey" on table "orders"
DETAIL: Key (customerid)=(1060) is still referenced from table "orders".

Estado de la tabla orderdetail despues del rollback
14076-3783-1060
14076-362-1060
14076-1121-1060
14076-355-1060
14076-1562-1060

b) Si usamos el commit intermedio:

Estado de la tabla orderdetail antes de la ejecucion de la query
14076-3783-1060
14076-362-1060
14076-1121-1060
14076-355-1060
14076-1562-1060

Estado de la tabla orderdetail despues de la ejecucion de la query

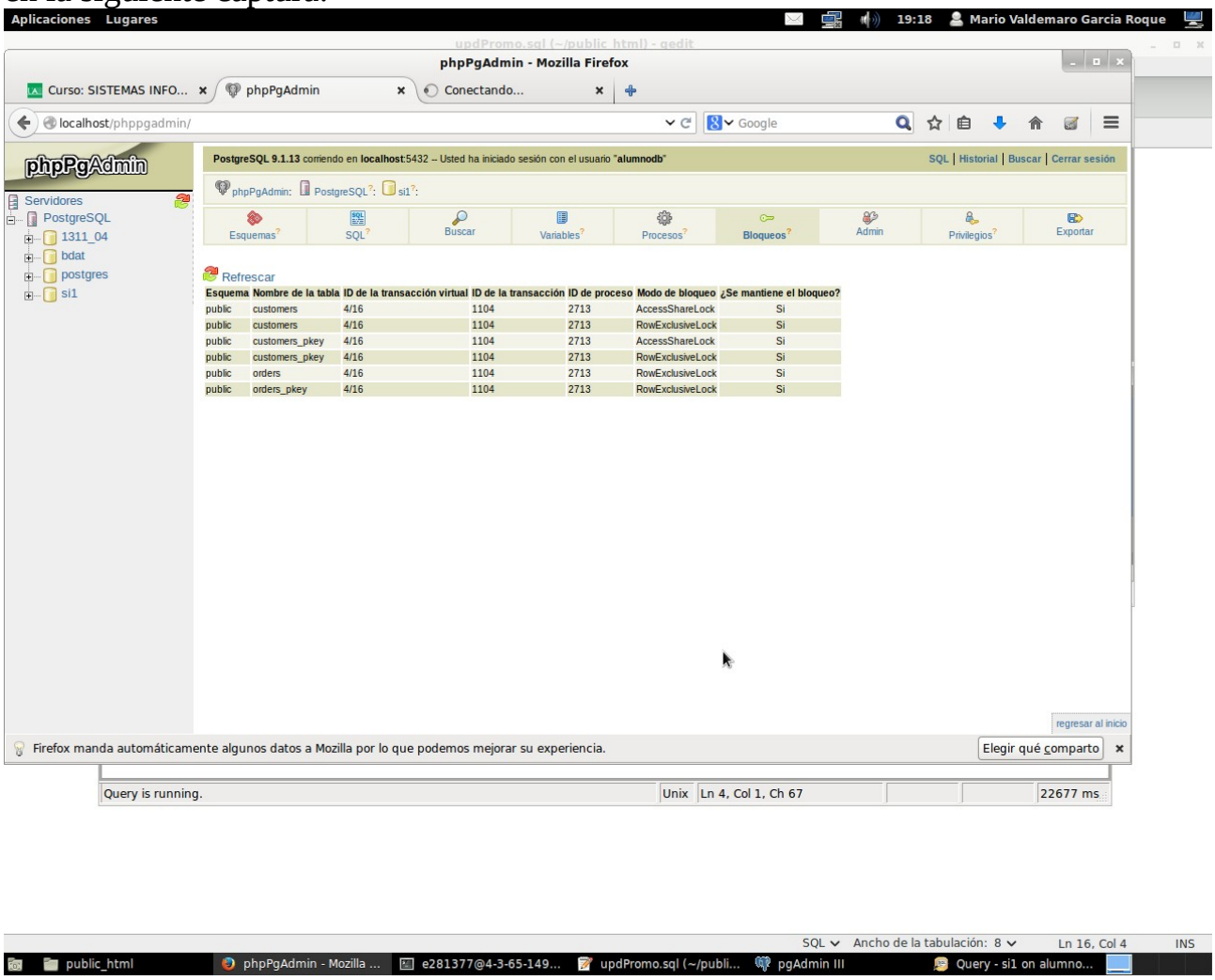
SQLSTATE[23503]: Foreign key violation: 7 ERROR: update or delete on table "customers" violates foreign key constraint "orders_customerid_fkey" on table "orders"
DETAIL: Key (customerid)=(1060) is still referenced from table "orders".

Estado de la tabla orderdetail despues del rollback

Commit: Sentencia cuyo efecto es que el resultado de la ejecución de la transacción sea permanente una vez que haya finalizado correctamente.

Apartado F):

El primer bloqueo que producimos es en la tabla customers como se puede observar en la siguiente captura:



Esto ocurre porque mientras la consulta para actualizar el carrito se queda bloqueando tanto customers como orders. La transacción de borra cliente esta esperando para usar la tabla customers pero no puede porque ya la están modificando. Por eso se produce un bloqueo, pero no es definitivo ya que una vez pase el tiempo que hemos programado en el trigger la tabla customers se podrá volver a usar.

Sin embargo si se pueden consultar datos mientras.

Por ultimo para hacer un deadlock permanente seria tan sencillo como que ocurriese algo así:

T1:	T2:
XLOCK Y	XLOCK X
XLOCK X	XLOCK Y

El problema es que no hemos llegado a poder coordinarlo bien para que salga, ya que en parte se necesita algo de suerte.

Apartado G):

1. Insertamos en username lo siguiente : gatsby' or password like '%
   esta inserción de código hace que para el usuario gatsby no
   haga falta tener la contraseña para conectarnos ya que la query
   nos quedaría algo como esto: select fistname, lastname from customers
   where username like 'gatsby' or password like '%' and password like '';
   por tanto no nos hace falta la contraseña para conectarnos.
2. Con insertar esto en el campo usuario funcionaria:
   ' or username like '%' or password like '%
   El problema es que haremos login como todos los usuarios
   siendo el único válido el ultimo, si la página mantiene información de usuarios,
   que no es el caso.
3. En principio esto se podría evitar haciendo que cada vez que detecte que
   se han introducido valores como ('), (%), etc. insertase a su izquierda
   un slash, es decir, '/'. Esto lo soluciona, ya que nos impediría que lo
   que se inserta tenga efecto ya que todo quedaría dentro de un quote
   y por tanto solo se compararía eso con un username, contraseña, etc. Esto
   funciona en MySQL, en postgres no estamos seguros. Si no funciona hay
   muchas mas maneras. Podríamos usar, por ejemplo, sentencias preparadas o
   limitar las tablas a las que puede acceder un usuario. En general cualquier
   mecanismo que revise la inserción de datos sería una buena solución.

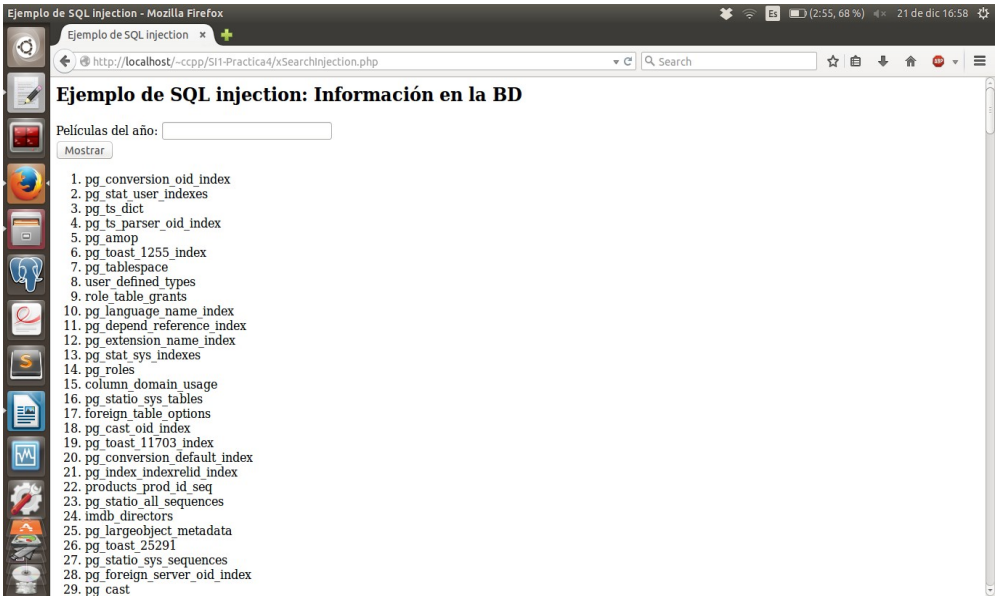
Apartado h):

Antes de nada vamos a mostrar 2 métodos para hacer lo que nos pedís, principalmente porque el método que nos proponéis nos parece demasiado complicado y encontramos una forma mucho mas sencilla de hacerlo.

Obtener la tabla del sistema:

-1' union select relname from pg_class where relname like '%

Resultado:

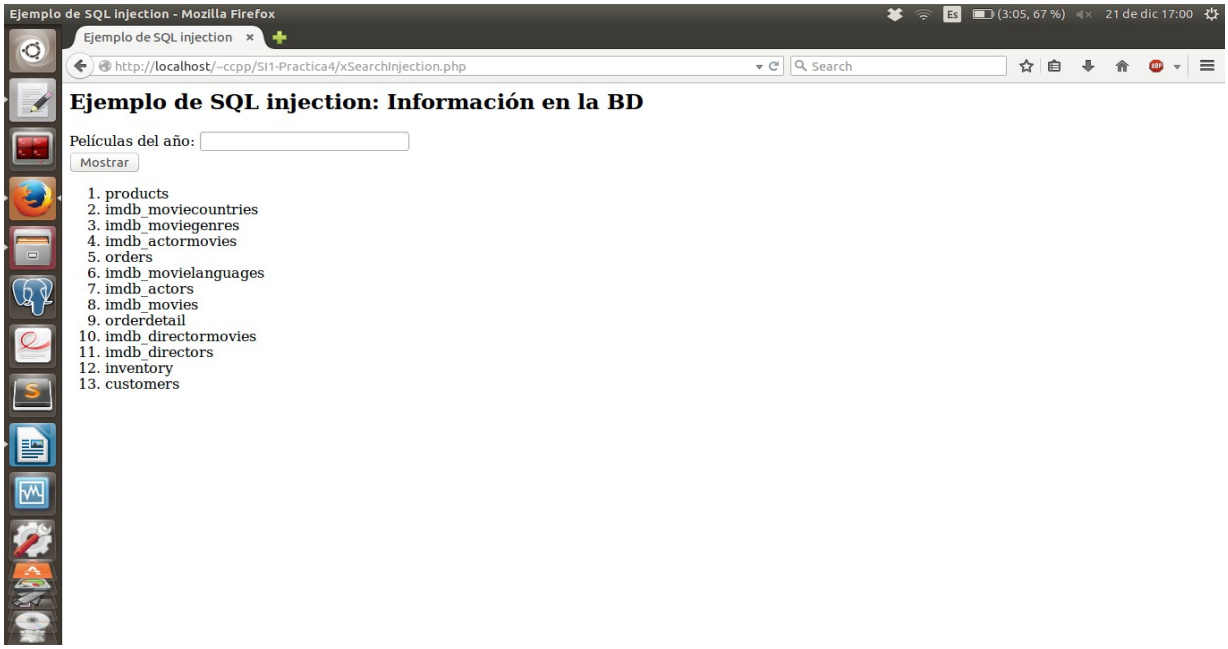


Obtener las tablas public de la tabla sistema:

Forma1:

-1' union select table_name from information_schema.tables where table_schema like 'public

Resultado:



Forma 2:

primero esto

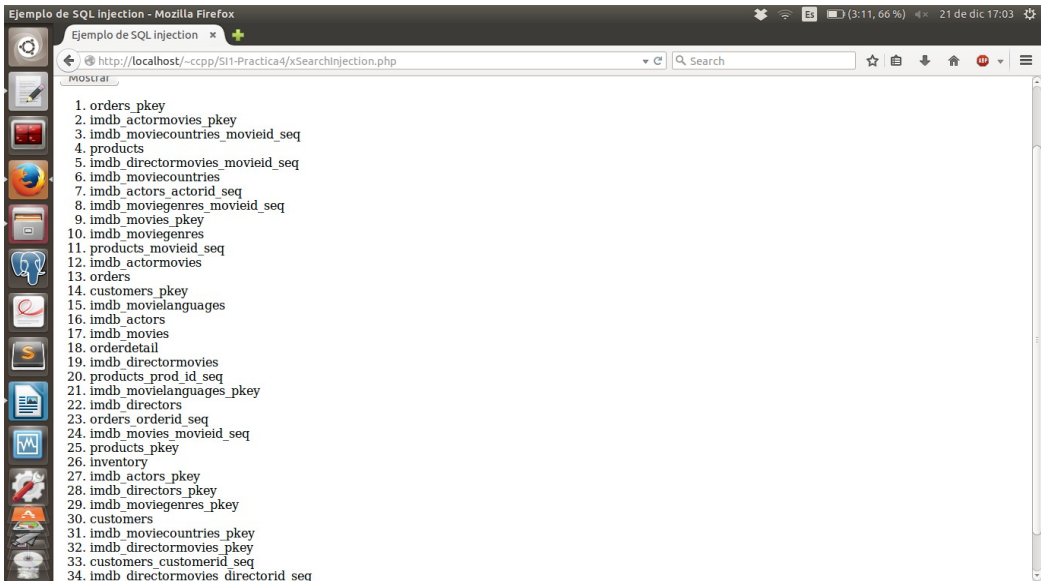
-1' union select cast(oid as varchar) from pg_namespace where nspname like 'public

que nos de volvera como resultado 2200 que es el tipo de las tablas publicas.

Y después:

-1' union select relname from pg_class where relnamespace = 2200 or relname like '-1

Resultado:



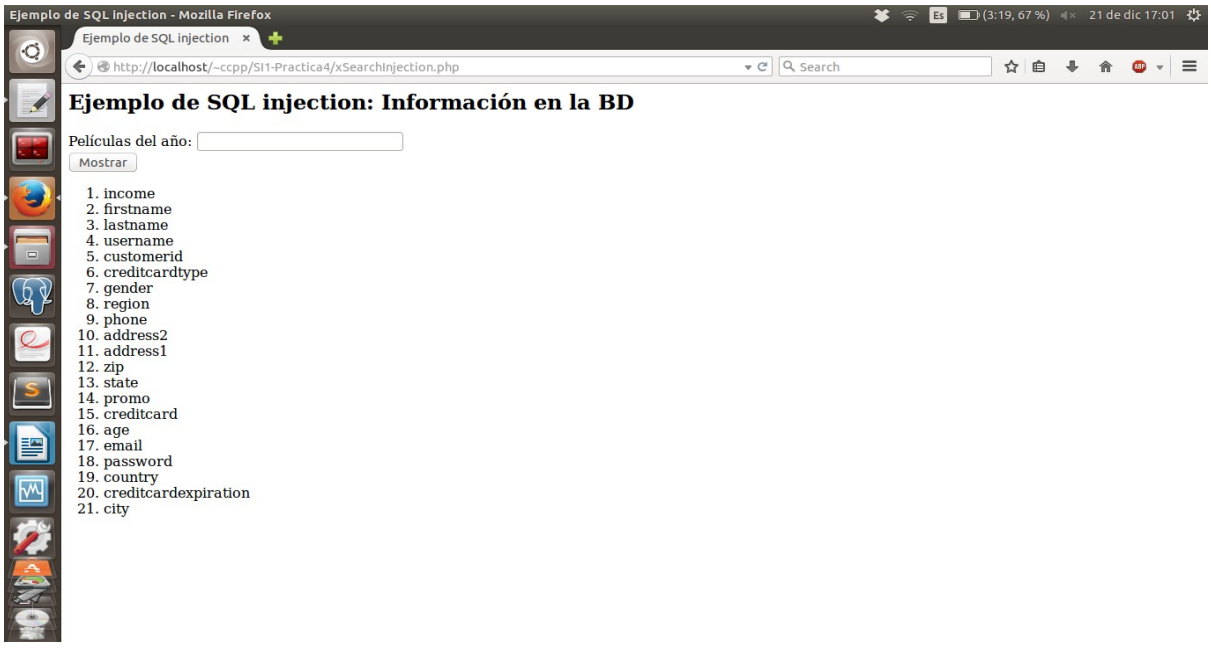
Evidentemente usaremos la tabla customers.

Obtener información de la tabla customers:

Forma 1:

-1' union select column_name from information_schema.columns where table_name like 'customers

Resultado:



Forma 2:

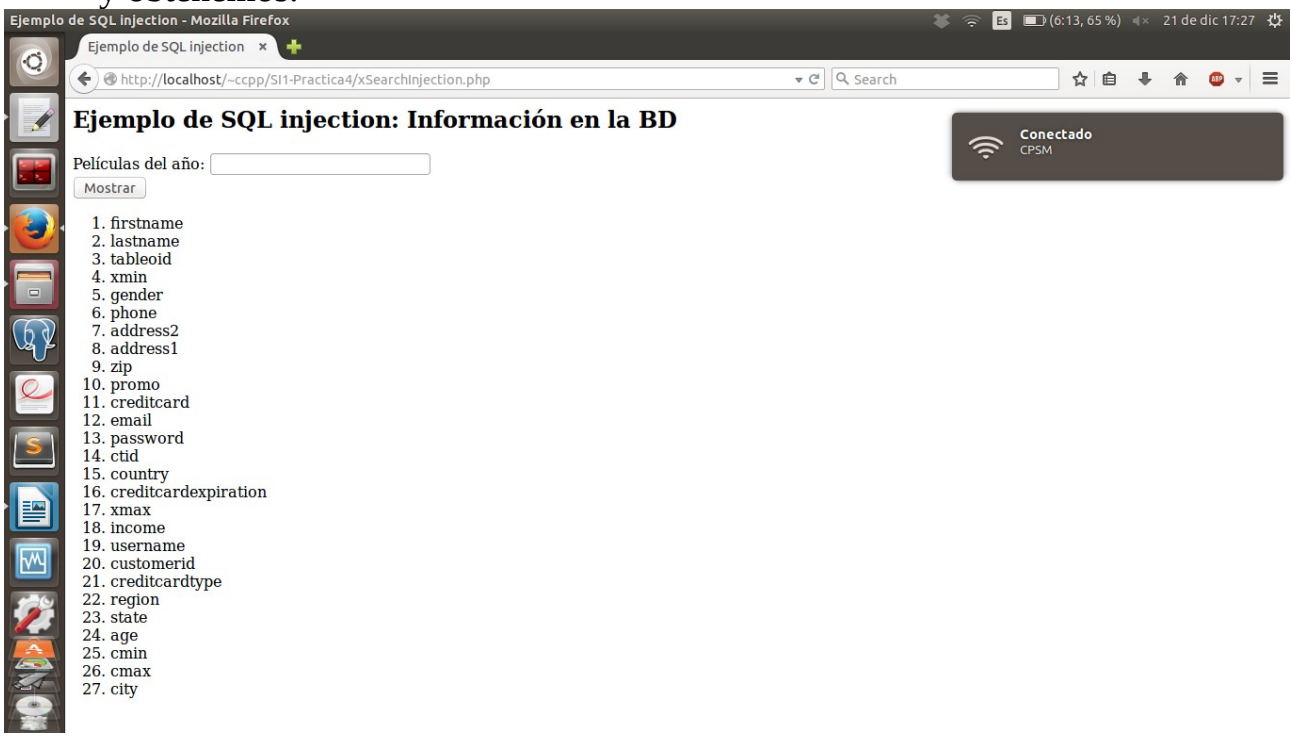
Primero tenemos que ver el oid de la tabal customers que es 25283 ya que lo obtenemos con la consulta

-1' union select cast (oid as varchar) from pg_class where relname like 'customers' or relname like '-1

que es básicamente la misma que la del apartado anterior. Después hacemos esta otra consulta para obtener los atributos

-1' union select attname from pg_attribute where attrelid =25283 or attname like '-1

y obtenemos:

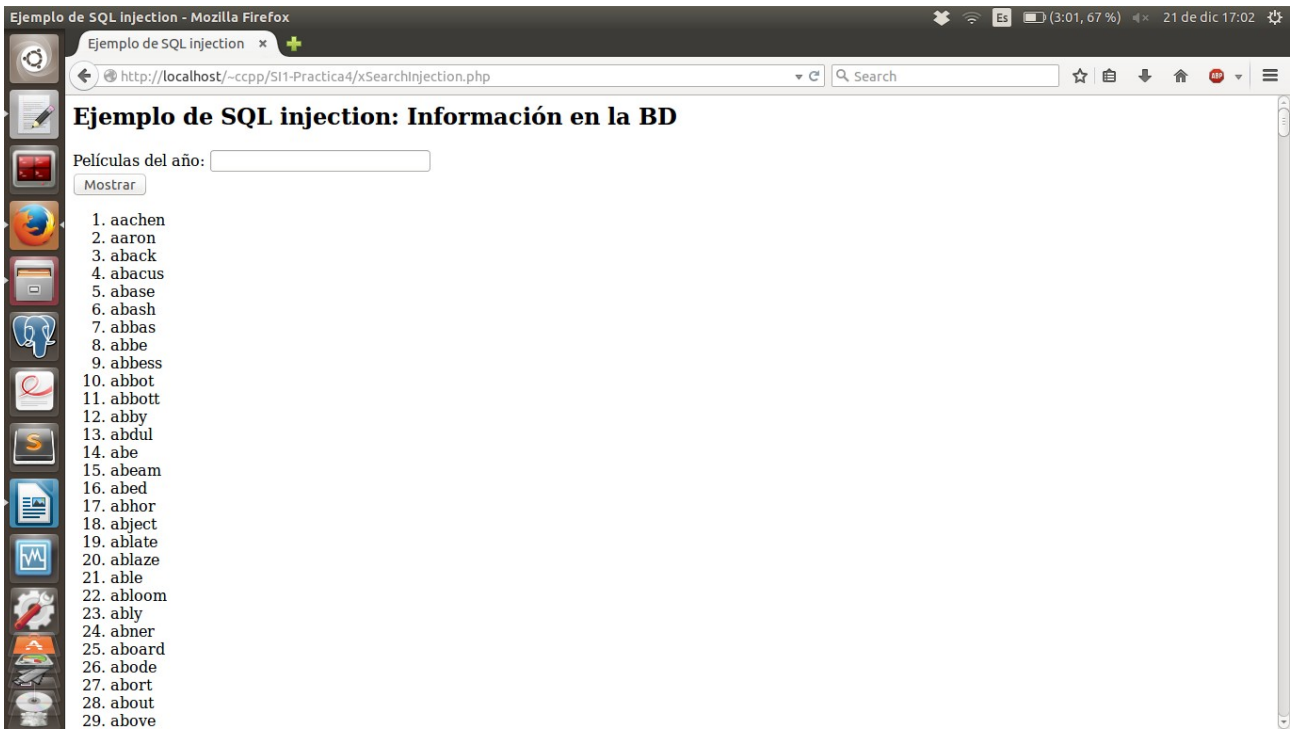


Aquí el mejor candidato para contener username evidentemente es username.

Ya que sabemos todo podemos hacer la ultima consulta:

-1' union select username from customers where username like '%

Que nos devuelve todos los usuarios:



Por ultimo ninguna de las formas que proponéis para mejorar la seguridad de la página funciona ya que: el POST va a dar igual porque podemos seguir metiendo código como lo hemos hecho hasta ahora, por el formulario. Un desplegable también es una muy mala idea ya que si tenemos GET podemos seguir metiendo información por la url de la pagina y si tiene un POST da completamente igual porque podemos ir a la pagina, ver el código html de la pagina con F12, copiar el formulario de la página a otra página en nuestro ordenador y cambiar el tipo de input para poder escribir lo que queramos en el formulario mientras mantenemos los nombres de las variables. Por último le mandamos lo que queramos y la página no nota la diferencia de si le están mandando cosas desde la misma pagina o desde otra.

El problema de los métodos que nos dais para asegurar la pagina es que están ideados para “limitar” las formas en las que se pueden hacer las cosas, lo que es una mala aproximación ya que quita accesibilidad y no aporta nada de seguridad.

Lo que hay que hacer es analizar lo que entra.