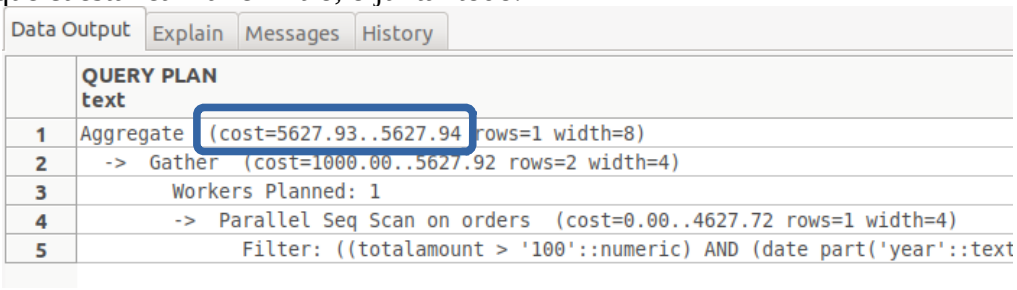
GENERAL

La practica entregada contiene los tres ficheros SQL, la aplicación con los ficheros database.py y routes.py, los templates html de las paginas dadas, y el start.wsgi para acceder a todas las paginas desde el localhost.

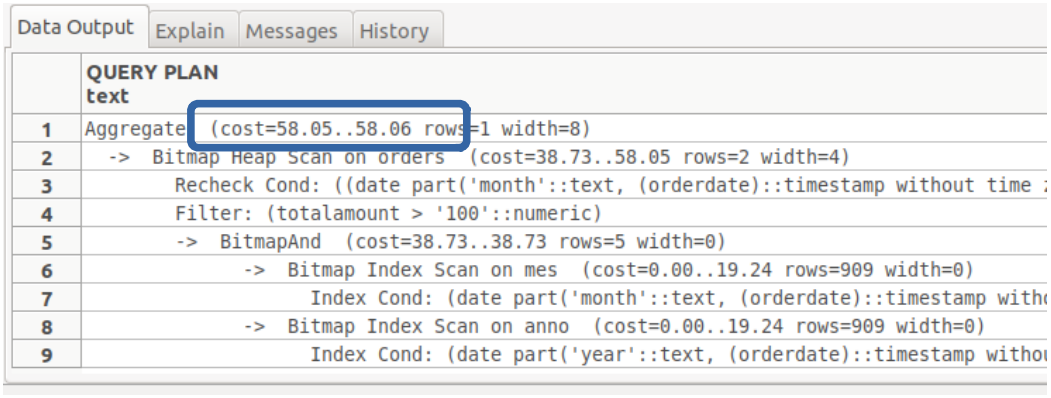
Abajo estan las respuestas a las preguntas del enunciado con capturas de pantalla y descripciones de los metodos que hemos utilizado.

OPTIMIZACIÓN

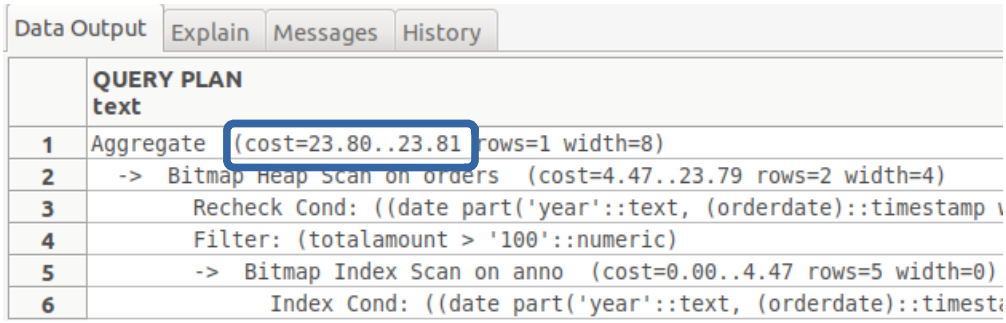
A) Si hacemos EXPLAIN de la consulta, nos indica el coste de esta en diferentes partes, como lo que cuesta realizar el filtro, o juntar todo.



Si creamos dos índices, uno sobre la columna del año y el otro sobre la columna del mes (para los que se usara al hacer el filtro), podemos ver como los costes totales se reducen:

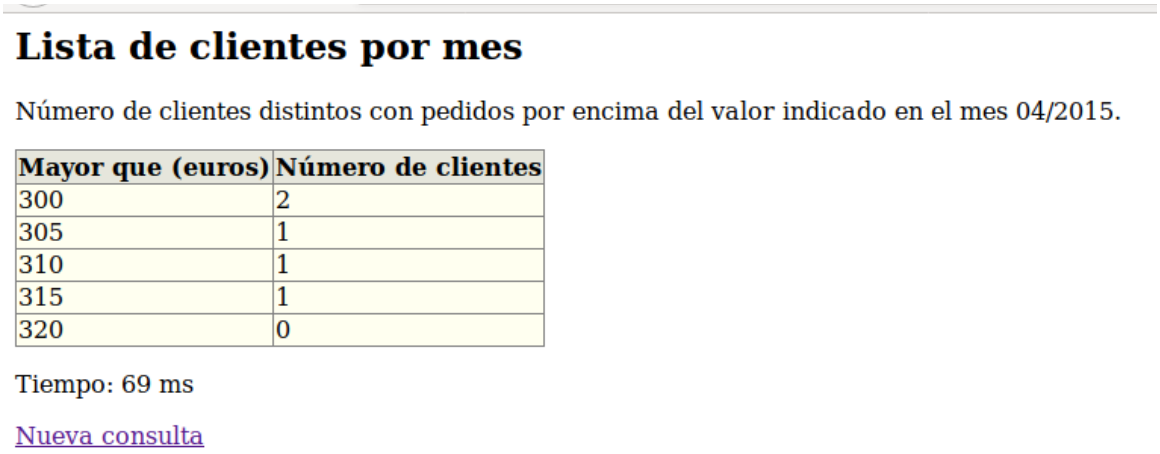


Pero creamos un único índice sobre ambas columnas del filtro, y volvemos a hacer un EXPLAIN sobre la consulta, podemos ver como los costes de esta se reducen aún más:



Por tanto, para esta consulta podemos concluir que lo mejor para reducir costes seria hacer un único índice sobre ambas columnas de la búsqueda.

B) Haciendo la consulta en apache, se ve que el rendimiento se cambia bastante. El tiempo de ejecución mejora con el uso de un índice, pero si el índice se tiene que crear de nuevo al ejecutar el programa, entonces el tiempo de rendimiento es peor.



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
300	2
305	1
310	1
315	1
320	0

Tiempo: 69 ms

Nueva consulta

La imagen de arriba es el tiempo de ejecución sin un índice.

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
300	2
305	1
310	1
315	1
320	0

Tiempo: 307 ms

Nueva consulta

La imagen de arriba es el tiempo de ejecución con la creación de un índice en 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
300	2
305	1
310	1
315	1
320	0

Tiempo: 17 ms

Nueva consulta

La imagen de arriba es el tiempo de ejecución con el índice ya creado.

Se ve que el mejor rendimiento es el que tiene un índice ya creado. El índice se hace sobre el año y mes. Otros índices se podrían hacer sobre el mes solo o el año solo, pero el mejor índice se hace juntando las dos cosas.

C)

i) Nada más ejecutarse, la primera consulta devuelve los resultados. Esto es porque se hace un sequential scan directamente.

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Seq Scan on customers (cost=3961.65..4490.81 rows=7046 width=4)
2	Filter: (NOT (hashed SubPlan 1))
3	SubPlan 1
4	-> Seq Scan on orders (cost=0.00..3959.38 rows=909 width=4)
5	Filter: ((status)::text = 'Paid'::text)

Data Output | Explain | Messages | History

	QUERY PLAN text
1	HashAggregate (cost=4537.41..4539.41 rows=200 width=4)
2	Group Key: customers.customerid
3	Filter: (count(*) = 1)
4	-> Append (cost=0.00..4462.40 rows=15002 width=4)
5	-> Seq Scan on customers (cost=0.00..493.93 rows=14093 width=4)
6	-> Seq Scan on orders (cost=0.00..3959.38 rows=909 width=4)
7	Filter: ((status)::text = 'Paid'::text)

Data Output | Explain | Messages | History

	QUERY PLAN text
1	HashSetOp Except (cost=0.00..4640.83 rows=14093 width=8)
2	-> Append (cost=0.00..4603.32 rows=15002 width=8)
3	-> Subquery Scan on "*SELECT* 1" (cost=0.00..634.86 rows=14093 width=8)
4	-> Seq Scan on customers (cost=0.00..493.93 rows=14093 width=4)
5	-> Subquery Scan on "*SELECT* 2" (cost=0.00..3968.47 rows=909 width=8)
6	-> Seq Scan on orders (cost=0.00..3959.38 rows=909 width=4)
7	Filter: ((status)::text = 'Paid'::text)

ii) La segunda consulta, que usa consultas anidadas con uniones es la que más se beneficia de la ejecución en paralelo. La tercera tambien se beneficiaria, por el uso de subqueries, pero no tanto como la segunda.

D) El generador de estadísticas es un aspecto interno de la base de datos que calcula el mejor plan de ejecucion a base de pruebas internas. La planificación de las dos consultas es la misma al principio, usando un sequential scan en ambos casos. Despues de la creacion del indice, la planificacion de ambas consultas cambia a usar una busqueda de indice. El coste de rendimiento usando esta planificacion es mejor en ambas consultas. Al ejecutar el comando ANALYZE, se cambia de nuevo la planificacion, mejorando el rendimiento en el caso de la primera consulta (respecto a las primeras ejecuciones), y empeorando el rendimiento de la segunda consulta (respecto a las primeras ejecuciones).

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Aggregate (cost=3507.17..3507.18 rows=1 width=8)
2	-> Seq Scan on orders (cost=0.00..3504.90 rows=909 width=0)
3	Filter: (status IS NULL)

Primera consulta, sin indice.

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Aggregate (cost=3961.65..3961.66 rows=1 width=8)
2	-> Seq Scan on orders (cost=0.00..3959.38 rows=909 width=0)
3	Filter: ((status)::text = 'Shipped'::text)

Segunda consulta, sin indice.

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Aggregate (cost=1496.52..1496.53 rows=1 width=8)
2	-> Bitmap Heap Scan on orders (cost=19.46..1494.25 rows=909 width=0)
3	Recheck Cond: (status IS NULL)
4	-> Bitmap Index Scan on estado (cost=0.00..19.24 rows=909 width=0)
5	Index Cond: (status IS NULL)

Primera consulta con indice.

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Aggregate (cost=1498.79..1498.80 rows=1 width=8)
2	-> Bitmap Heap Scan on orders (cost=19.46..1496.52 rows=909 width=0)
3	Recheck Cond: ((status)::text = 'Shipped'::text)
4	-> Bitmap Index Scan on estado (cost=0.00..19.24 rows=909 width=0)
5	Index Cond: ((status)::text = 'Shipped'::text)

Segunda consulta con indice.

Output pane

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Aggregate (cost=7.28..7.29 rows=1 width=8)
2	-> Index Only Scan using estado on orders (cost=0.42..7.28 rows=
3	Index Cond: (status IS NULL)

Primera consulta despues de ejecutar ANALYZE.

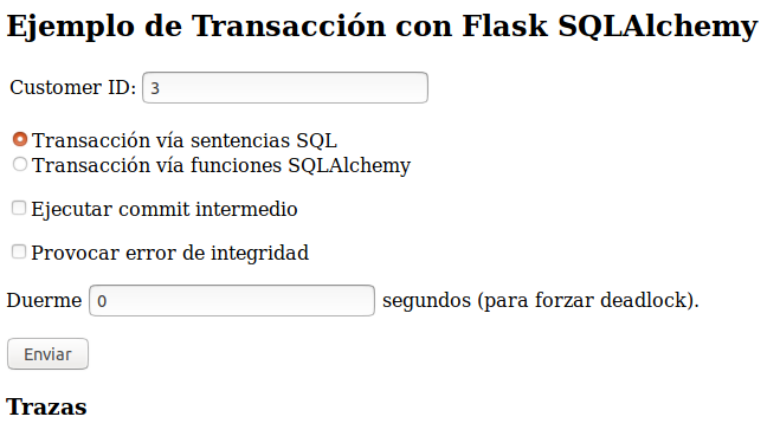
Data Output | Explain | Messages | History

	QUERY PLAN text
1	Finalize Aggregate (cost=4210.57..4210.58 rows=1 width=8)
2	-> Gather (cost=4210.45..4210.56 rows=1 width=8)
3	Workers Planned: 1
4	-> Partial Aggregate (cost=3210.45..3210.46 rows=1 width=8)
5	-> Parallel Seq Scan on orders (cost=0.00..3023.69 rows=74705 wid
6	Filter: ((status)::text = 'Shipped'::text)

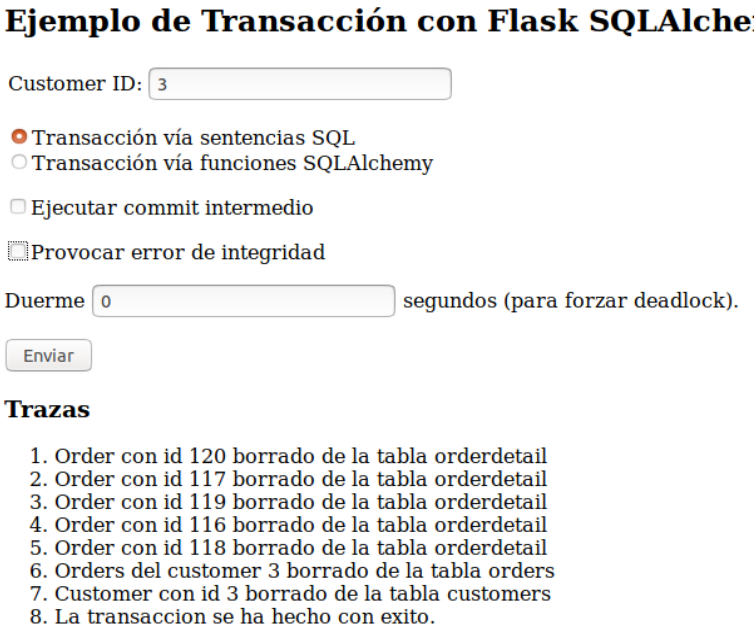
Segunda consulta despues de ejecutar ANALYZE.

TRANSACCIONES Y DEADLOCKS

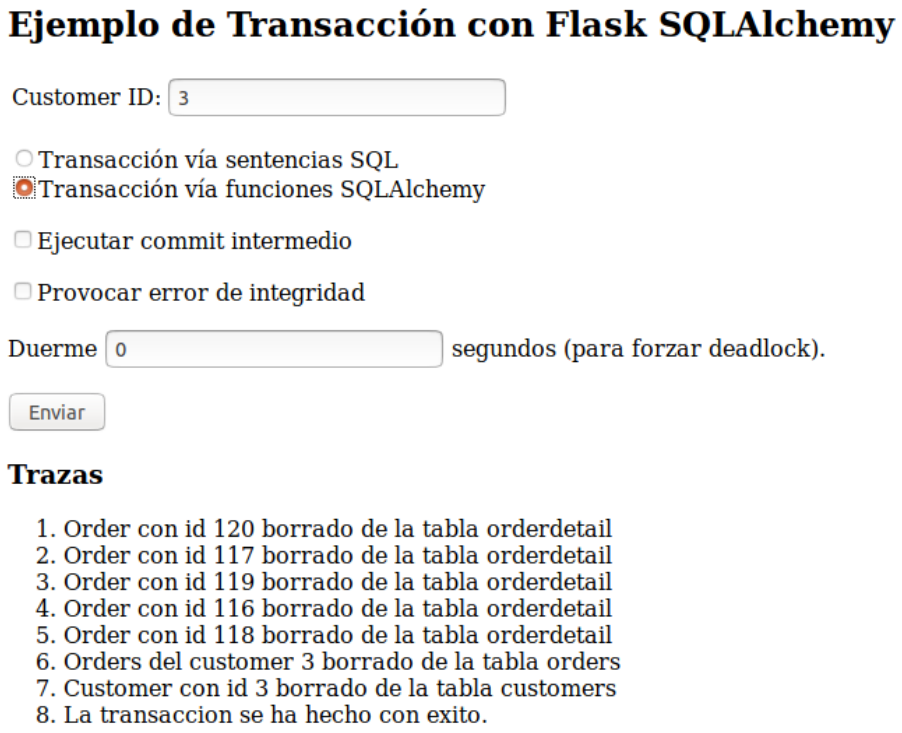
E) Ejemplo de la pagina principal de borraCliente.



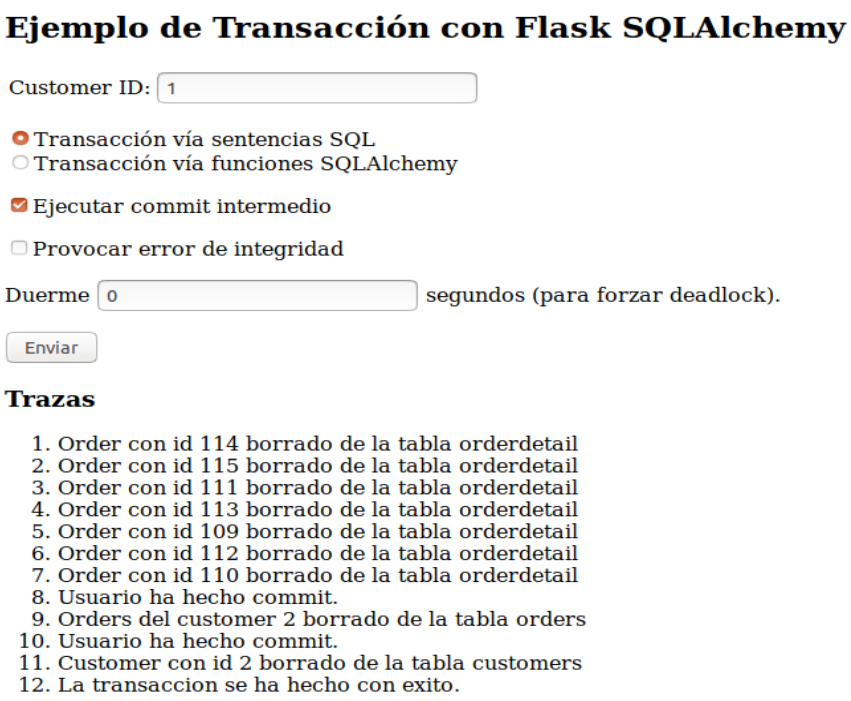
El programa permite el borrado de un cliente por su ID. Da la opcion de hacerlos usando transacciones de SQL o usando SQLAlchemy. Tambien se puede elegir si se hace commits intermedios o si se quiere provocar un fallo para comprobar el rollback. Como se ve abajo, se hace primero el borrado de los registros asociados, y luego se borra el cliente.



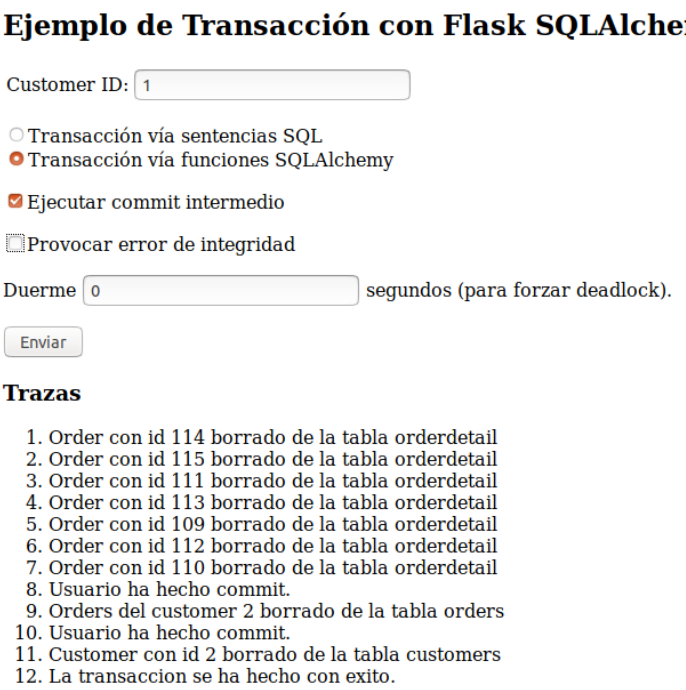
Borrado con solo SQL.



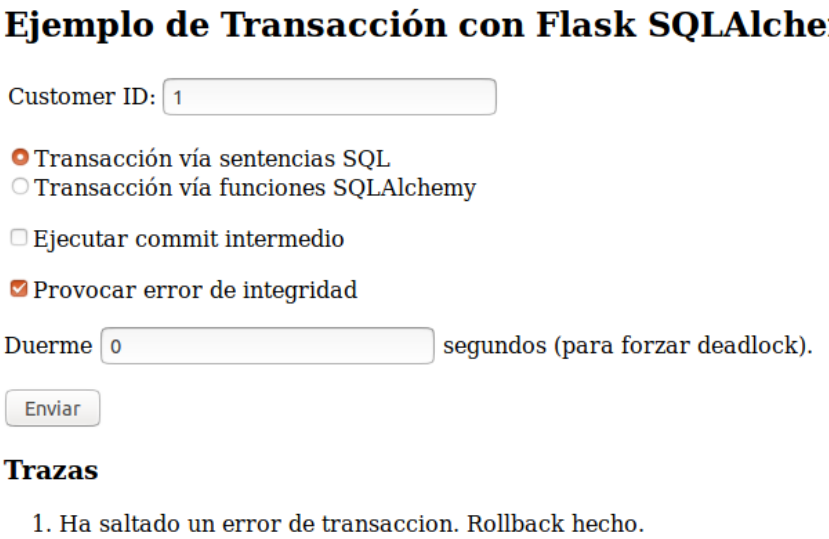
Borrado con solo SQLAlchemy.



Borrado con SQL y commits intermedios.



Borrado con SQLAlchemy y commits intermedios.



Intento de borrado, provocando fallos.

F) Los datos alterados por la página o por el trigger no son visibles porque se hace el rollback antes de acabar.

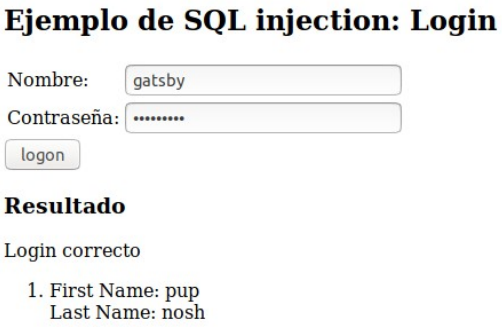
El deadlock se produce porque cuando se hace un sleep, la base de datos se queda esperando una respuesta de los extremos del servidor, pero como pasa suficiente tiempo, se hace un timeout sin avisar y la pagina se queda esperando para siempre.

Para poder evitar esto, se puede crear una condicions de excepcion que avisa si una transaccion se ha hecho timeout, y entonces se manda un aviso y se deshace la transaccion.

SEGURIDAD

G) Aqui se trabaja con SQL injections. Usando el nombre gatsby';--- se consigue hacer un select de solo el nombre de usuario, porque el '---' convierte el resto del commando en commentario.

En la siguiente imagen, con el uso de la contrasena ‘or’1’=’1, se consigue acceder a la primera cuenta en la base de datos.



Si se prohibe el uso de apostrofes, punticomas, y otros caracteres que se pueden asociar a SQL, se puede evitar el SQL injection, pero al coste de un poco de seguridad por tener contrasenas no tan seguros.

H) La consulta que consigue mostrar todas las tablas es la siguente:

‘; SELECT table_name
FROM information_schema.tables
WHERE table_type='BASE TABLE'
AND table_schema='public';---

Porque se accede con conocimiente universal del funcionamiento de la base de datos postgres.

Para poder evitar eso, se podria usar un combobox, pero deja libre a ataques el protocolo GET. Entonces, cambiando el protocolo a POST y haciendo uso de un combo box se podria llegar a mejorar la proteccion contra ataques de SQL injection.