Mario López y Juan Moreno

# PRÁCTICA 4

## Enunciado: Optimización

## Apartado A:

Hemos creado un script llamado clientesDistintos.sql, el cual ejecuta la consulta que nos describe el enunciado. La ejecuta de varias maneras distintas, pero todas con explain. La primera no utiliza ningún índice, la segunda utiliza el índice 'month', la tercera el índice 'year', la cuarta el índice 'year-month', la quinta el índice 'month-year' y la última el índice 'customerid'.

| **Indice** | Sin índice | month | year | year-month | month-year | customerid |
|---|---|---|---|---|---|---|
| **Coste** | 5627 | 1509 | 1509 | 23 | 23 | 5627 |

A.
```
explain select count (distinct customerid)
from orders
where totalamount > 100 and
date_part('month', orders.orderdate) = 04 and
date_part('year', orders.orderdate) = 2015;
```

B. Ejecutamos la consulta añadiendo la sentencia EXPLAIN. En primer lugar se ejecuta una sentencia una sentencia de tipo 'Aggregate' que en este caso es un 'count' de nuestra consulta. Finalmente, el Filter elimina las filas que no cumplen la condición de la consulta. En la parte de Aggregate, podemos observar el campo cost. En este campo se muestran dos valores que van de un rango a otro. Esto significa que el primer valor será el coste de obtener la primera y fila y el segundo el coste final de la consulta(este valor es aproximado).

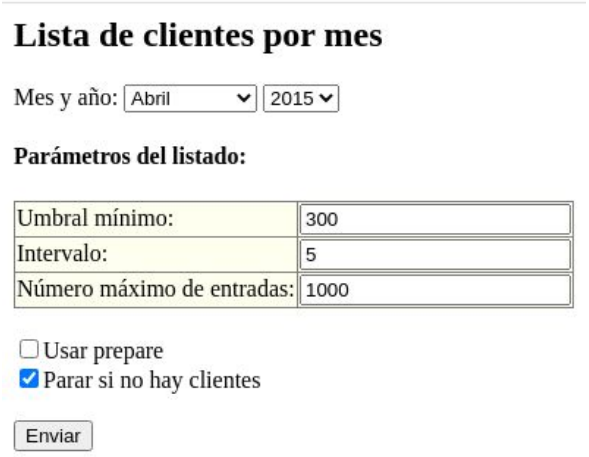
C. Hemos añadido a la consulta un índice para el campo 'month'. Como observamos en la tabla, el coste es bastante mejor que sin utilizar índice.

D. En el plan de la consulta podemos observar algunos campos que ya hemos reconocido en el apartado B. En primer lugar se ejecuta una sentencia una sentencia de tipo ‘Aggregate’ que ejecuta el ‘count’ de nuestra consulta. Realiza un Bitmap Heap Scan en la tabla orders porque es más eficiente el no tener que acceder a todas las filas de la tabla. Posteriormente, vuelve a comprobar la condición y filtra según lo indicado en la consulta. El Bitmap index scan ordena de forma secuencial los datos obtenidos por el índice. La condición del índice indica sobre qué campo estamos realizando el índice.

E. En la tabla, se indican los distintos índices que hemos utilizado. Podemos observar una mejora en los tiempos en los que el índice utiliza los campos ‘year-month’ y ‘month-year’ ya que se repiten bastante y nuestro acceso a la tabla es mucho más específico. Los índices no relevantes, tardan algo menos o un tiempo parecido a sin índice. La conclusión que sacamos es que el uso de índices suele mejorar el rendimiento de la consulta, y más si estos son compuestos y más relevantes con respecto a la lógica de la consulta.

## Apartado B:

A. Realizamos la página getListaCliMes del database.py siguiendo los pasos que se nos van indicando.
B. Tiempo sin prepare y sin índice: 85 ms.



## 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: 85 ms

C. Tiempo con prepare y sin índice: 85 ms.

## 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: 85 ms

**Usando prepare**

D. Tiempo sin prepare y con índice(‘year-month’): 21 ms.
E. Tiempo con prepare y con índice(‘year-month’): 25 ms.

Podemos observar que al ejecutar usando ‘prepare’ los tiempos pueden empeorar, aunque normalmente son parecidos a si no lo usamos. Esto se debe a que el uso del ‘prepare’ hace que la consulta se optimice sólo si ésta se ejecuta múltiples veces. Como podemos observar, el utilizar el índice compuesto(‘year-month’) hace que mejore el tiempo de ejecución de forma considerable, al igual que en el apartado anterior.

## Apartado C:

A.

| QUERY PLAN |
|---|
| Seq Scan on customers  (cost=4004.68..4533.85 rows=7046 width=4) |
| Filter: (NOT (hashed SubPlan 1)) |
| SubPlan 1 |
| -> Seq Scan on orders  (cost=0.00..3959.38 rows=18124 width=4) |
| Filter: ((status)::text = 'Paid'::text) |

B.

| QUERY PLAN |
| --- |
| HashAggregate (cost=4795.63..4797.63 rows=200 width=4) |
| Group Key: customers.customerid |
| Filter: (count(*) = 1) |
| -> Append (cost=0.00..4634.55 rows=32217 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=18124 width=4) |
| Filter: ((status)::text = 'Paid'::text) |

C.

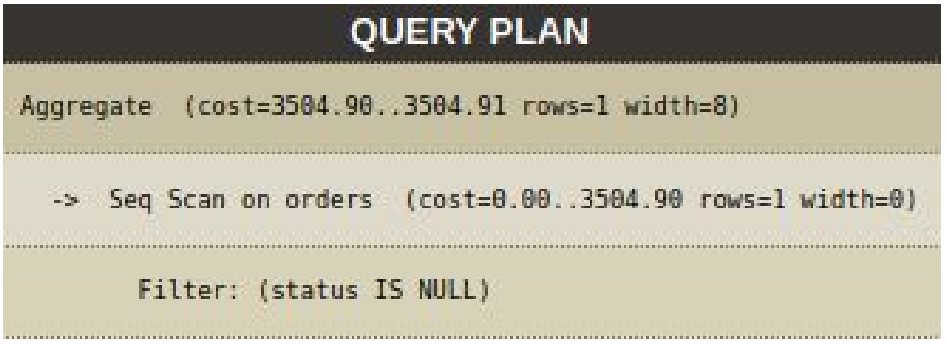
| QUERY PLAN |
| --- |
| HashSetOp Except (cost=0.00..4715.84 rows=14093 width=8) |
| -> Append (cost=0.00..4678.34 rows=15002 width=8) |
| -> Subquery Scan on "*SELECT* 1" (cost=0.00..634.86 rows=14093 width=8) |
| -> Seq Scan on customers (cost=0.00..493.93 rows=14093 width=4) |
| -> Subquery Scan on "*SELECT* 2" (cost=0.00..3968.47 rows=909 width=8) |
| -> Seq Scan on orders (cost=0.00..3959.38 rows=909 width=4) |
| Filter: ((status)::text = 'Paid'::text) |

Al observar los tres planes de consulta, podemos ver que la tercera consulta es la que devuelve algún resultado nada más comenzar su ejecución debido a que el coste comienza en 0. Esto significa que el primer resultado se obtiene en ese tiempo. Podemos observar que la segunda y la tercera consulta son las que se benefician sobre la ejecución en paralelo. El Seq Scan está a la misma altura en estas dos consultas, solo que en la tercera cada uno está dividido en subconsultas distintas.

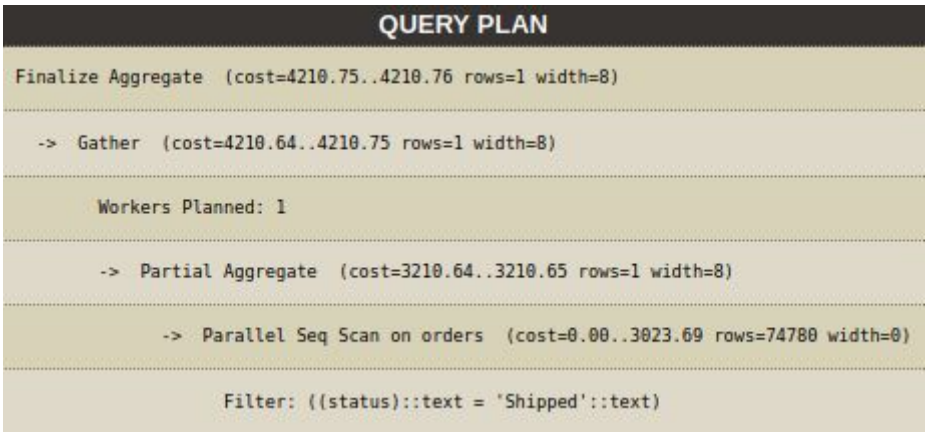
## **Apartado D:**

A. -
B. Consultas

a. Consulta 1 sin indice:




b. Consulta 2 sin indice:




Se realiza un escaneo secuencial sobre la tabla orders hasta encontrar las filas que cumplen sus condiciones correspondientes.

La segunda consulta tiene un tiempo de ejecución más alto que la primera. Se debe a que como podemos observar en el Filter del plan de ejecución, éste cambia. En la primera consulta no filtra por texto, sino que solo tiene que comprobar el valor del campo. En cambio, la segunda consulta, a parte de que obtiene un resultado mayor que 0, tiene que filtrar el texto de cada fila.

C. Consultas tras crear índice status

a. Consulta 1 con índice:

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

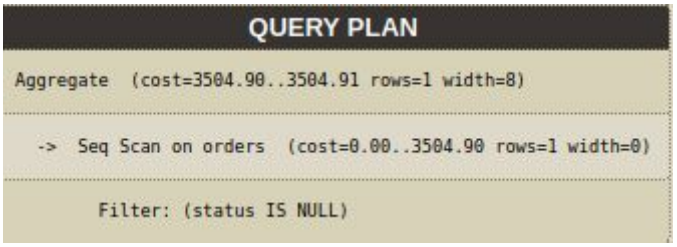
b. Consulta 2 con índice:

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

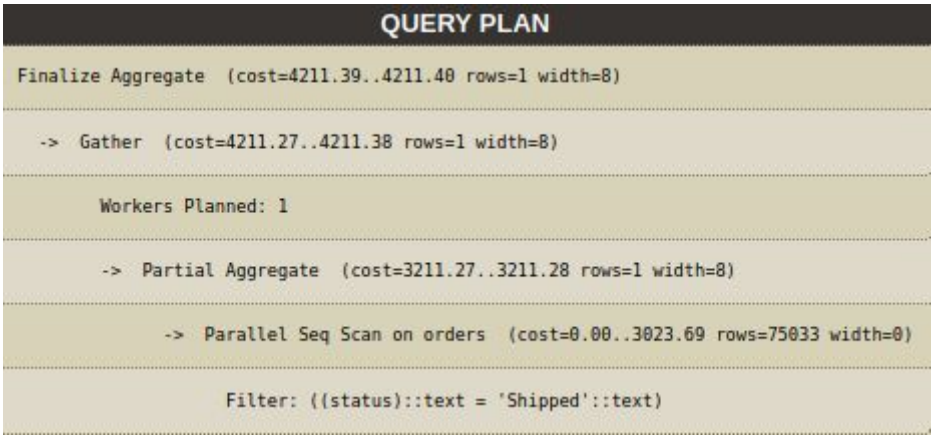
D. Podemos observar que el tiempo de ejecución con el índice en la primera consulta mejora más que en la segunda consulta. De todas formas, la segunda consulta se optimiza notablemente. Al crear el índice sobre la columna status, el plan de acceso cambia a un Index Scan sobre ese índice.

E.

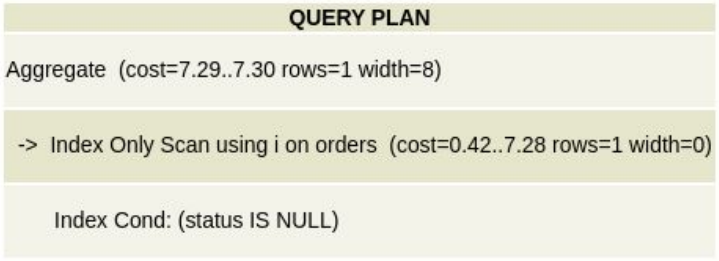
a. Consulta 1 sin índice y con analyze



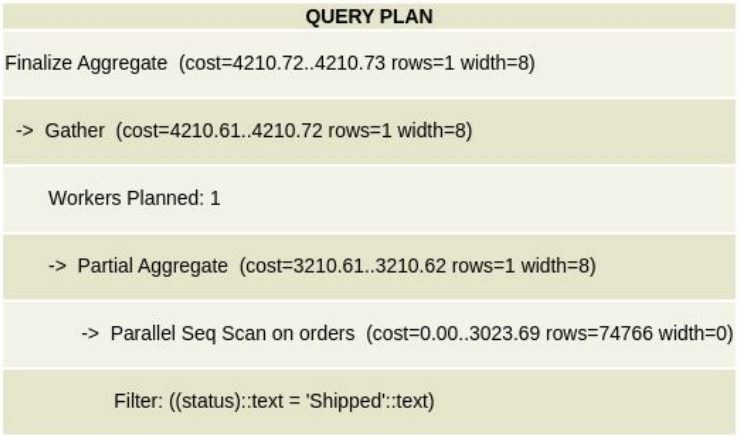

b. Consulta 2 sin índice y con analyze




c. Consulta 1 con índice y con analyze




d. Consulta 2 con índice y con analyze




Podemos observar que al ejecutar las dos consultas utilizando el analyze en la tabla orders y sin index, no cambia el plan de la consulta y tampoco varía el coste. Sin embargo, al utilizar índices, los planes de ejecución difieren al utilizar el analyze. Se

reduce bastante el coste mediante el uso de estadísticas y con índices. Lo que realiza el analyze es invocar el optimizador y posteriormente producir el explain.

F. Podemos observar que al ejecutar las dos consultas utilizando el analyze en la tabla orders y sin index, no cambia el plan de la consulta y tampoco varía el coste. Sin embargo, al utilizar índices, los planes de ejecución difieren al utilizar el analyze. Se reduce bastante el coste mediante el uso de estadísticas y con índices. Lo que realiza el analyze es invocar el optimizador y posteriormente producir el explain.
    a. Coste consulta 1 con analyze y sin índice: 3504
    b. Coste consulta 2 con analyze y sin índice: 4211
    c. Coste consulta 1 con analyze y con índice status: 7.30
    d. Coste consulta 2 con analyze y con índice status: 4210
G. Gracias al comando analyze sobre la tabla orders, generamos estadísticas sobre los estados que son más frecuentes en las tablas. Nos permite realizar un planteamiento eficiente sobre las consultas dos y tres.
    a. Consulta 3 con analyze y con índice:

| QUERY PLAN |
| --- |
| Aggregate (cost=2320.45..2320.46 rows=1 width=8) |
| -> Bitmap Heap Scan on orders (cost=361.12..2275.06 rows=18155 width=0) |
| Recheck Cond: ((status)::text = 'Paid'::text) |
| -> Bitmap Index Scan on i (cost=0.00..356.58 rows=18155 width=0) |
| Index Cond: ((status)::text = 'Paid'::text) |

    b. Consulta 4 con analyze y con índice:

| QUERY PLAN |
| --- |
| Aggregate (cost=2954.57..2954.58 rows=1 width=8) |
| -> Bitmap Heap Scan on orders (cost=719.56..2863.23 rows=36534 width=0) |
| Recheck Cond: ((status)::text = 'Processed'::text) |
| -> Bitmap Index Scan on i (cost=0.00..710.42 rows=36534 width=0) |
| Index Cond: ((status)::text = 'Processed'::text) |

H. Adjuntamos en la carpeta ‘SQL’ el script countStatus.sql

# Enunciado: Transacciones y Deadlocks

## Apartado E:

Apartados A-J:

Para empezar, en el routes.py, ya que se nos pide realizar la página utilizando ‘GET’, cambiamos la recepción de argumentos de ‘form’ a ‘args’.

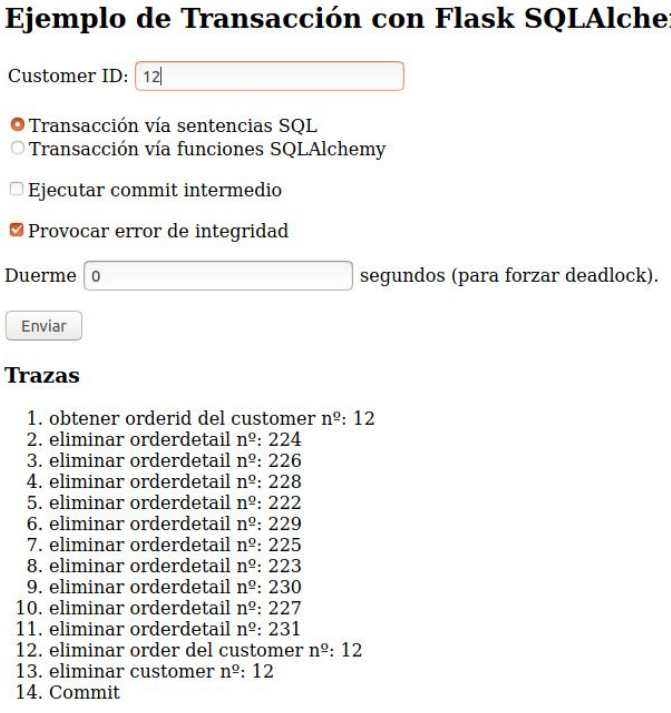
Posteriormente, nos disponemos a implementar la funcionalidad de la página en el métodod ‘delCustomers’ del database.py.

Ejecutaremos las consultas que nos permitirán borrar el customer. Para esto, seguiremos los siguientes pasos:

1. Seleccionamos el orderid de la tabla orders para el customer
2. Borramos los orderdetail del pedido que hemos seleccionado previamente
3. Borramos el order con el orderid que hemos obtenido en la primera consulta
4. Finalmente eliminamos el customer y se realiza el commit.

De esta forma, evitamos las restricciones en las claves foráneas del customer. Sin embargo, si queremos forzar que de fallo al borrar el customer, realizaremos el paso 4 anterior antes de tiempo. Si salta la excepción(ha habido algún fallo), ejecturamos el rollback para deshacer la transaccion.

Borramos customer con id  = 12 mediante sentencias SQL



Borramos customer con id  = 13, vía funciones SQLALCHEMY

**Trazas**

1. obtener orderid del customer nº: 13
2. eliminar orderdetail nº: 244
3. eliminar orderdetail nº: 252
4. eliminar orderdetail nº: 246
5. eliminar orderdetail nº: 232
6. eliminar orderdetail nº: 253
7. eliminar orderdetail nº: 238
8. eliminar orderdetail nº: 255
9. eliminar orderdetail nº: 249
10. eliminar orderdetail nº: 234
11. eliminar orderdetail nº: 241
12. eliminar orderdetail nº: 250
13. eliminar orderdetail nº: 248
14. eliminar orderdetail nº: 236
15. eliminar orderdetail nº: 256
16. eliminar orderdetail nº: 247
17. eliminar orderdetail nº: 233
18. eliminar orderdetail nº: 245
19. eliminar orderdetail nº: 240
20. eliminar orderdetail nº: 235
21. eliminar orderdetail nº: 237
22. eliminar orderdetail nº: 254
23. eliminar orderdetail nº: 243
24. eliminar orderdetail nº: 239
25. eliminar orderdetail nº: 242
26. eliminar orderdetail nº: 251
27. eliminar order del customer nº: 13
28. eliminar customer nº: 13
29. Commit

Fallo de integridad con customerid = 14

**Ejemplo de Transacción con Flask SQLAlchemy**

Customer ID: 1

- (•) Transacción vía sentencias SQL
- ( ) Transacción vía funciones SQLAlchemy

- [ ] Ejecutar commit intermedio
- [x] Provocar error de integridad

Duerme 0 segundos (para forzar deadlock).

Enviar

**Trazas**

1. obtener orderid del customer nº: 14
2. eliminar orderdetail nº: 264
3. eliminar orderdetail nº: 261
4. eliminar orderdetail nº: 263
5. eliminar orderdetail nº: 257
6. eliminar orderdetail nº: 258
7. eliminar orderdetail nº: 260
8. eliminar orderdetail nº: 259
9. eliminar orderdetail nº: 262
10. eliminar customer nº: 14
11. Rollback

Borramos customer con id  = 14, vía funciones SQLALCHEMY y con commit intermedio

**Trazas**

1. obtener orderid del customer nº: 17
2. eliminar orderdetail nº: 315
3. eliminar orderdetail nº: 313
4. eliminar orderdetail nº: 307
5. eliminar orderdetail nº: 306
6. eliminar orderdetail nº: 304
7. eliminar orderdetail nº: 309
8. eliminar orderdetail nº: 308
9. eliminar orderdetail nº: 312
10. eliminar orderdetail nº: 310
11. eliminar orderdetail nº: 316
12. eliminar orderdetail nº: 314
13. eliminar orderdetail nº: 311
14. eliminar orderdetail nº: 305
15. Commit
16. Begin
17. eliminar order del customer nº: 17
18. eliminar customer nº: 17
19. Commit

## Apartado F:

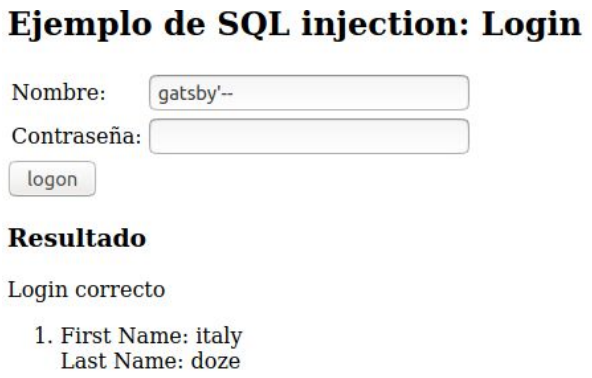
A. -
B. Hemos creado el script updPromo.sql en el directorio “SQL”, que empieza creando una nueva columna llamada “promo” en la tabla customers.
C. Posteriormente, creamos una función que devuelve un trigger, el cual al realizar un update en la tabla customers, cambiará el “totalamount” de los pedidos de ese customer dependiendo del valor que se haya asignado a la columna promo.
D. Introducimos la sentencia “perform pg_sleep(10);” para que duerma 10 segundos antes de finalizar.
E. Insertamos la sentencia “time.sleep(duerme)” en la función “delCustomer” del fichero database.py antes de finalizar el borrado del usuario.
F. -
G. -

## Enunciado: Seguridad

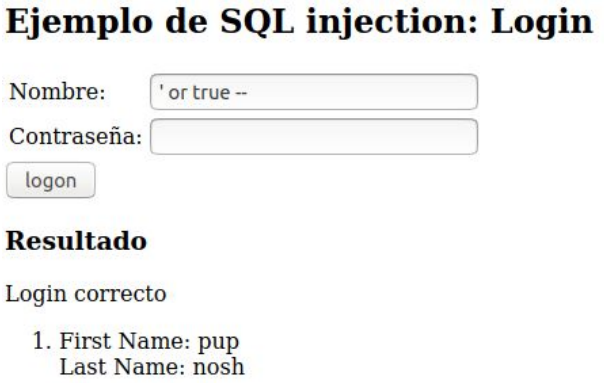
## Apartado G:

A. Para conseguir iniciar sesión utilizando sólo el nombre del usuario, lo que realizamos es inyectar un comentario tras el nombre de usuario, es decir ‘--.

La comilla sirve para cerrar el campo del nombre en la consulta y el -- hacemos que lo posterior quede comentado y no se ejecute. La consulta quedaría de tal forma: select * from customers where username= ‘gatsby’--’ and password = ‘  ’, la parte de password quedaŕa comentada, haciendo que iniciemos sesión exitosamente y devolviendo el firstname y lastname del usuario.




B. Lo que realizamos para poder iniciar sesión sin usuario ni contraseña, es meter una condición que siempre se cumpla en la consulta: “where true”. De esta forma, la consulta devolverá el primer resultado de la base de datos de la siguiente forma: "select * from customers where username=' ‘ or true -- and password=' ‘. Obtenemos el siguiente resultado:




C. Para evitar estos accesos indebidos habría que hacer lo siguiente:
    a. Usar sentencias preparadas para no poder inyectar código en éstas.
    b. Comprobar los argumentos introducidos en el formulario, es decir:
        i. Símbolos de comentarios
        ii. Símbolos de ;
        iii. Espacios, tabulaciones o saltos de línea