Práctica 3: Sistemas Informáticos I

Programación Web y Bases de Datos:

Índice de contenido

Introducción:

En esta práctica se nos pide completar el funcionamiento de la página web implementada en la práctica anterior, pero para que ahora trabaje con **una base de datos** en SQL, en vez de en XML.

Ésto nos exigía conocer el funcionamiento de los *triggers,* así como saber crear procedimientos almacenados.

Además, nos requería refrescar la memoria para algunas *queries* pedidas.

Las herramientas de trabajo en esta práctica han sido:

- ***PostgreSQL***
- ***Servidor en Apache para probar la página web en PHP***
- ***Base de datos****(la cual había que modificar un poco)*

Diseño de la base de datos:

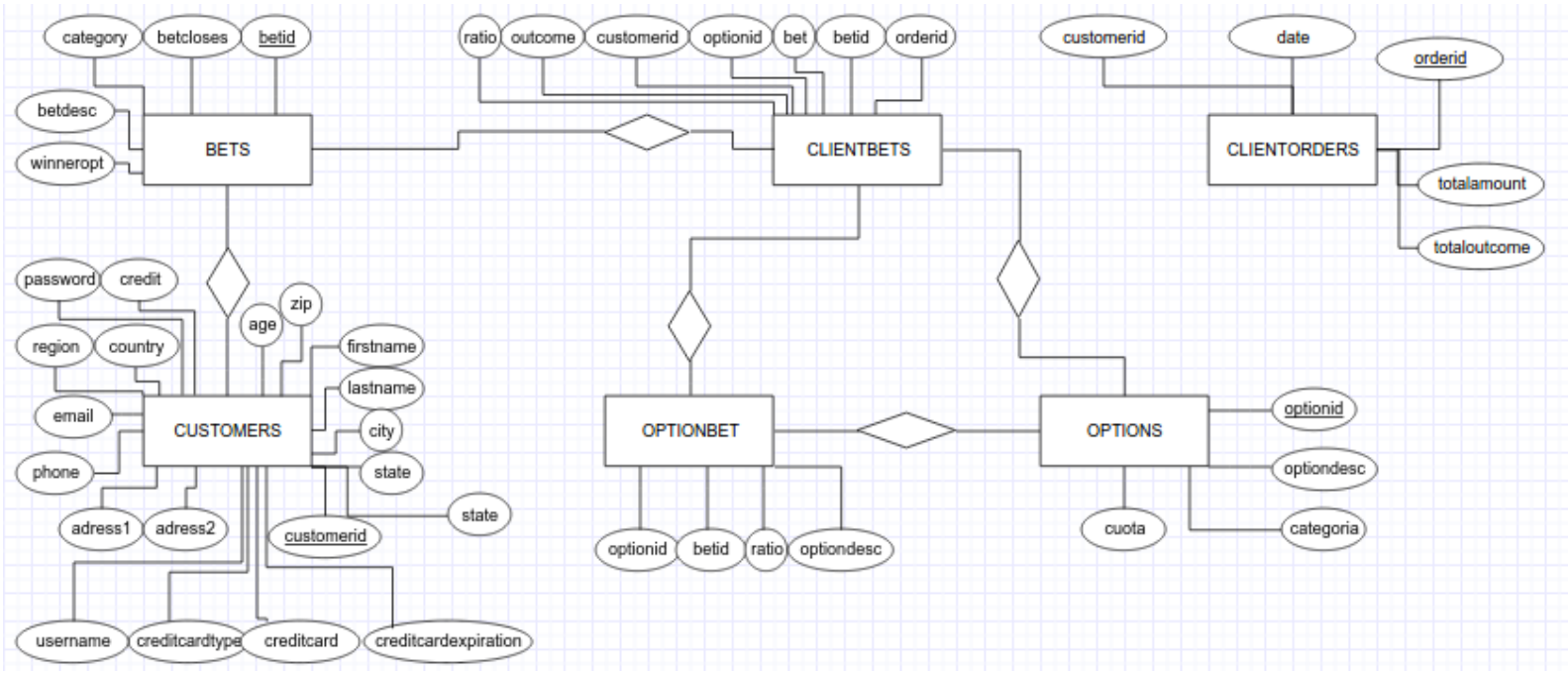
Una vez comprendido el funcionamiento de la base de datos, decidimos realizar cambios mínimos.

Uno de ellos fue el agregar una columna que almacene el total del *outcome* que teóricamente recibiría el **customer** cuando finalicen todas las apuestas de un pedido. Es decir, agregamos el campo **totaloutcome** en la tabla *clientorders.*

También tuvimos que utilizar un **ALTER SEQUENCE para** que el order_id empezara tras el último **order_id** creado.

Estos cambios han sido entregados en el fichero adjunto **actualizar.sql.**

El modelo E-R resultante es el siguiente:



Nuestra idea era hacer más cambios, sobre todo con respecto a las relaciones entre tablas.

Un cambio que nos gustaría haber realizado, por ejemplo, era el de relacionar *clientbets* con *clientsorder* **haciendo orderid PRIMARY KEY** en ésta última y siendo referenciada desde *clientbets.* Sin embargo, al intentar hacerlo, SQL lo impedía puesto que existían **apuestas con un orderid que no estaba en** clientorders.
Debido a esto, decidimos no tocar el contenido de la base de datos y dejar las relaciones como estaban.

Análisis de las tareas:

La práctica la podemos dividir en 3 partes:

Tareas referentes al diseño de la base de datos

Como hemos discutido anteriormente, realizamos en primer lugar **todos los cambios posibles en la base de datos y,** posteriormente, elaboramos el Modelo-ER.

Tareas referentes a las consultas y triggers

Antes de empezar a modificar la página, era necesario hacer algunas consultas para llenar unos campos que estaban vacíos en un principio, así como *triggers* que **automaticen** ciertos procesos necesarios tras la inserción / actualización / eliminación de una tupla.

Las consultas y triggers pedidas son:

- **SetTotalAmount.sql:**
  El funcionamiento es muy sencilo. Para cada **orderid** existente en la tabla *clientorders* busca **todas las apuestas con ese orderid.** En el caso de que no haya ninguna, **totalamount es 0.** En caso contrario, totalamount es la **suma** de todos los **bets.**

- **SetOutcomeBets.sql:**
  Para cada apuesta de *clientbets,* esta consulta calcula la ganancia multiplicando la **apuesta realizada por el ratio de ganancia <u>sólo si</u>** el optionID elegido ha sido el ganador del encuentro.

- **SetOrderTotalOutcome.sql:**
  Este procedimiento almacenado **calcula** la suma de los *outcome* con un **orderid** determinado. En el caso de que no exista ninguna apuesta con un **orderid** determinado, **totaloutcome** es 0. Al ser un procedure, puede ser llamado en cualquier momento.

- **UpdBets.sql:**
  El *trigger* simplemente es una llamada a la consulta de **setOutcomeBets** para cada actualización o inserción de una apuesta nueva, es decir, va actualizando el **outcome.**

- **UpdOrders.sql:**

Este otro era algo más complejo, pues había que hacer diferentes casos. Uno para el **UPDATE, otro para el INSERT y otro para el DELETE.**
Este trigger se lanza siempre que se hace algo en *clientbets.*
Hacemos uso de **TG_OP** para comprobar el caso a ejecutar.

- En caso de un UPDATE, comprobamos si el outcome es diferente.(**bet nunca va a cambiar).**
- En caso de un INSERT, actualizamos el totalamount y el totaloutcome si procede.
- En caso de un DELETE, actualizamos el totalamount y el totaloutcome si procede(pero con resta en este caso).

- **UpdCredit:**

  El trigger comprueba si un **orderid** pasa de tener una fecha NULL a una establecida. **Si es asi, descuenta del crédito del customer el totalamount.** Además, le suma el totaloutcome en caso de que corresponda.

Tareas referentes a la página web

Sólo se nos pedia dos implementaciones de la página web con base de datos, **aunque hemos integrado la base de datos totalmente al final**.

La funcionalidad que se nos pedía desde un principio con base de datos era la de **Login** y la del **carrito.**

- La implementación de Log In es **muy sencilla.** Tras rellenar el formulario de Log In se comprueba si **algún username coincide con el username introducido en el form. Si es así, se comprueba la contraseña.** Si también coincide, entoonces almacenamos en **cookies** el nombre del usuario y su id de sesión iniciada.

  Para ésta tarea no era necesario hacer muchos cambios con respecto a la práctica anterior.

- La implementación del carrito era algo más costosa. Además, implementamos algo nuevo con respecto a la práctica 2. Ahora, al rellenar un carrito como **usuario invitado** (al cual vamos a poder acceder desde diferentes páginas) podemos guardarlo en sesión para confirmarlo **una vez hagamos LogIn.** Por otra parte, una vez hecho el LogIn, si existe la variable de sesión **carrito**, se nos muestra en el **div** de la esquina superior derecha. Si no, se crea en caso contrario. **Además,** ahora tras realizar una apuesta,

volvemos al **index, no al carrito(al que vamos a poder acceder en cualquier momento).**

El funcionamiento del carrito, al no venir explícitamente especificado en el enunciado, lo hemos planteado como en la gran mayoría de las páginas web. El carrito se mantiene como variable de sesión, y **una vez que confirmamos el pedido,** entonces escribe en la base de datos. ¿Qué escribe? Muy sencillo, primero crea un una tupla en *clientorders.* Posteriormente recogemos el *orderid* de dicha tabla para asignárselo a cada una de las **apuestas** que vamos a insertar en clientbets. Una vez hecho esto, actualizamos el campo **fecha** del pedido total en la tabla *clientorders,* lo cual lanza un trigger que descuenta del crédito del cliente el **totalamount.**

Hemos comprobado que cada una de las consultas y triggers funcionan correctamente insertando tuplas en la tabla correspondiente desde *postgreSQL.*

Mejoras propuestas por el alumno

- Una de las mejoras que podríamos considerar es el hecho de que el **crédito** del cliente no se actualice cuando se cierra **todo el pedido,** sino(como ocurre en las páginas reales) cuando cada apuesta por separado termina.

- Otra sería el trabajar con **todos** los campos que aparecen en la base de datos, pues algunos de ellos, como **betcloses** no los utilizamos para mucho.

- Otra, la cual no hemos podido probar(puesto que no hemos implementado una forma de “finalizar una apuesta y establecer un winneropt global”) sería la de **comprobar que la ganancia** se obtiene correctamente. Sin embargo, esto si lo hemos comprobado desde *PostgreSQL* y **funciona correctamente**.

Tareas extra que hemos llevado a cabo

Como trabajo adicional, **hemos hecho una implementación completa de la base de datos en la página web.** Es decir, no trabajamos en ningún lugar con XML, y se puede **tanto registrar un usuario(**darlo de alta en la base de datos**), buscar apuestas(**las cuales las ordena por orden de finalización de la apuesta y solo muestra las primeras 20 para no saturar la página**), mostrar el historial de cada usuario(**últimas apuestas realizadas con información desplegable de ellas como el

outcome...**),** etcétera.

Para ellos hemos tenido que adaptar todos los módulos para trabajar con la **base de datos, como podemos ver en las siguientes capturas:**

Pantalla de registro:

Pantalla de apuesta:

Pantalla del carrito de compra(con un elemento):

Pantalla del historial(mostrando la última apuesta hecha):

Pantalla de búsqueda(que busca apuestas 'similares', no tiene por qué ser el nombre exacto):