

Sistemas Informáticos I: Práctica 3



Sergio Fuentes de Uña — Daniel Perdices Burrero

29 de noviembre de 2016

Índice

1. Análisis y resultados de la base de datos proporcionada	3
1.1. Esquema de las tablas proporcionadas	3
1.2. Organización de las tablas	4
1.3. Diseño inicial de las tablas	4
1.4. Propuesta de diseño de las tablas	5
2. Consultas SQL	6
2.1. Análisis de las consultas realizadas	6
2.1.1. setTotalAmount	6
2.1.2. setOutcomeBets	7
2.1.3. setOrderTotalOutcome	8
2.2. <i>Triggers</i>	9
3. Adaptación a la aplicación Web	9
3.1. Cambios realizados	9
3.2. Funcionalidad de la página web implementada	10

1. Análisis y resultados de la base de datos proporcionada

1.1. Esquema de las tablas proporcionadas

Leyenda: primary key ↑:foreign key

■ customers

- customerid
- firstname
- lastname
- address1
- address2
- city
- state
- zip
- country
- region
- email
- phone
- creditcardtype
- creditcard
- creditcardexpiration
- username
- password
- age
- credit
- gender

■ clientorders

- customerid
- date
- orderid

■ clientbets

- customerid↑
- optionid↑
- betid↑
- orderid
- bet
- ratio
- outcome

■ bets

- betid
- betcloses
- category
- betdesc
- winneropt

■ options

- optionid
- optiondesc
- categoria
- cuota

■ optionbet

- optionid↑
- betid↑
- ratio
- optiondesc

1.2. Organización de las tablas

La base de datos consta de las siguientes tablas:

- **customers** Clientes de la página de apuestas.
- **clientorders** Carritos de los clientes.
- **clientbets** Apuestas realizadas por los clientes.
- **bets** Apuestas disponibles.
- **options** Entidades deportivas.
- **optionbet** Entidades concretas que se involucran en cada apuesta

Estas tablas se encuentran relacionadas a través del siguiente diagrama Entidad-Relación

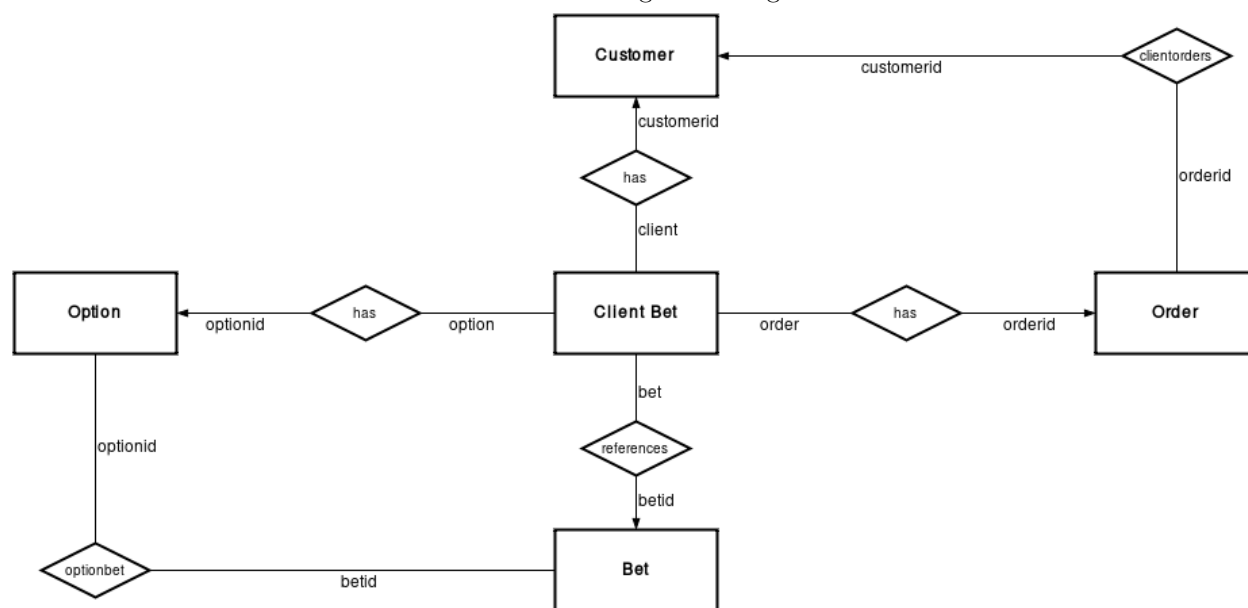


Figura 1: Diagrama E-R (sin atributos) de la base de datos proporcionada

1.3. Diseño inicial de las tablas

Con lo proporcionado, se pueden ver varias claras decisiones de diseño tomadas a la hora de elaborar la base de datos que se ha proporcionado:

1. El atributo **customerid** de la tabla **clientbets** es redundante con el atributo de igual nombre de la tabla **clientorders**.
2. El atributo **category** de la tabla **bets** y el atributo **categoría** de la tabla **options** son redundantes de igual manera, además, en la tabla las categorías se suelen repetir, por tanto se gasta mucho espacio de almacenamiento en guardar la misma cadena varias ocasiones.
3. El atributo **optiondesc** del mismo modo aparece en dos tablas, **options** y **optionbet**, aunque se podría ver como el mismo.

Muchas de estas repeticiones, especialmente la del **customerid** son bastante discutibles pues la repetición del dato mejora el rendimiento de la base de datos.

1.4. Propuesta de diseño de las tablas

Se han realizado los siguientes cambios en las tablas para normalizar la base de datos evitando futuras inconsistencias. Se han añadido restricciones para evitar registros erróneos.

■ `customers:`

- Se agrega la restricción de email válido.
- Se agrega la restricción de edad válida.
- Se agrega la restricción de código postal válida.
- Se agrega la restricción de tarjeta de crédito válida.
- Se agrega la restricción de teléfono válido.

■ `clientorders:`

- Se agrega la restricción de `orderid` como primary key.
- Se agrega la restricción de `customerid` como foreign key.
- Se activa el borrado en cascada si se elimina el cliente (`customerid`).

■ `clientbets:`

- Se agrega la restricción de `orderid` como foreign key.
- Se elimina la columna `customerid`.
- Se activa el borrado en cascada si se elimina el carrito (`orderid`).

■ `bets:`

- Se elimina el campo `category`.
- Se agrega el campo `categoryid` como foreign key.
- Se agrega la restricción de `winneropt` como foreign key.

■ `options`

- Se elimina el campo `categoria`.
- Se agrega el campo `categoryid` como foreign key.

■ `optionbet`

- Se elimina la columna `optiondesc`.

■ `categories`

- Se agrega la tabla.
- Se agrega el campo `categoryid` como primary key.
- Se agrega el campo `categorystring`.

A continuación se muestra el diagrama E-R (sin atributos por legibilidad) del estado final.

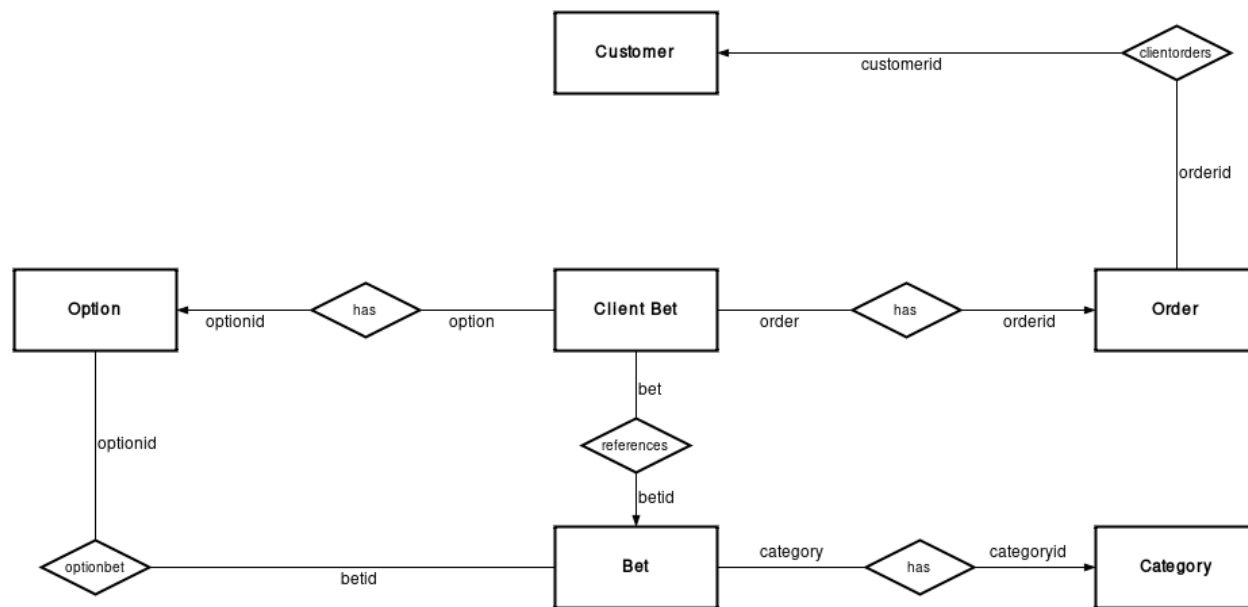


Figura 2: Diagrama E-R de la base de datos final

Todos estos cambios los realiza el fichero `actualiza.sql`

2. Consultas SQL

El objetivo de esta práctica ha sido implementar un conjunto de funciones y *triggers* que faciliten el mantenimiento y manejo de la base de datos y de la aplicación web.

2.1. Análisis de las consultas realizadas

A continuación se relatan los resultados obtenidos así como un resumen del rendimiento de las consultas.

2.1.1. setTotalAmount

Comprobación de los resultados

Se muestra el primer pedido previa la ejecución del ejercicio

customerid	date	orderid	totalamount
693	2012-12-02 17:23:39+01	1	

optionid	bet	ratio	outcome	betid	orderid
117	79.71	9.01		164	1
47	23.30	8.42		2372	1
51	74.58	5.92		2418	1
86	94.57	9.69		2438	1

Tras ejecutar el ejercicio resulta:

customerid	date	orderid	totalamount
693	2012-12-02 17:23:39+01	1	272.16

Por lo que se observa que la consulta se realiza correctamente

Rendimiento y optimización

El rendimiento original de la consulta

```
Update on clientorders (cost=97847.15..112110.70 rows=118766 width=146)
-> Hash Join (cost=97847.15..112110.70 rows=118766 width=146)
    Hash Cond: (aux.id = clientorders.orderid)
-> Subquery Scan on aux (cost=90991.97..98686.49 rows=118766
    width=96)
-> GroupAggregate (cost=90991.97..97498.83 rows=118766
    width=10)
    Group Key: clientbets.orderid
-> Sort (cost=90991.97..92666.07 rows=669638 width=10)
    Sort Key: clientbets.orderid
-> Seq Scan on clientbets (cost=0.00..14749.38
    rows=669638 width=10)
-> Hash (cost=3536.30..3536.30 rows=149030 width=54)
-> Seq Scan on clientorders (cost=0.00..3536.30 rows=149030
    width=54)
```

Para optimizar esta consulta, que pese a no tardar un tiempo excesivo, se detectan lecturas secuenciales, se crean índices.

Rendimiento tras la creación del índice sobre `clientbets(orderid)`

```
Update on clientorders (cost=6855.60..69056.69 rows=118766 width=146)
-> Hash Join (cost=6855.60..69056.69 rows=118766 width=146)
    Hash Cond: (aux.id = clientorders.orderid)
-> Subquery Scan on aux (cost=0.42..55632.49 rows=118766 width=96)
-> GroupAggregate (cost=0.42..54444.83 rows=118766 width=10)
    Group Key: clientbets.orderid
-> Index Scan using clbets_oid_idx on clientbets (
    cost=0.42..49612.05 rows=669640 width=10)
-> Hash (cost=3536.30..3536.30 rows=149030 width=54)
-> Seq Scan on clientorders (cost=0.00..3536.30 rows=149030
    width=54)
```

Rendimiento tras la creación del índice sobre `clientorders(orderid)`

```
Update on clientorders (cost=0.84..69557.52 rows=118766 width=146)
-> Merge Join (cost=0.84..69557.52 rows=118766 width=146)
    Merge Cond: (aux.id = clientorders.orderid)
-> Subquery Scan on aux (cost=0.42..55632.49 rows=118766 width=96)
-> GroupAggregate (cost=0.42..54444.83 rows=118766 width=10)
    Group Key: clientbets.orderid
-> Index Scan using clbets_oid_idx on clientbets (
    cost=0.42..49612.05 rows=669640 width=10)
-> Index Scan using clo_oid_idx on clientorders (cost=0.42..12067.88
    rows=149031 width=54)
```

Se observa una mejoría en el coste

2.1.2. setOutcomeBets

Comprobación de los resultados

Miramos de nuevo el mismo pedido y los winneropt

bet	outcome	winneropt	optionid
79.71		120	117
23.30		47	47
74.58		51	51
94.57		86	86

Tras ejecutar el ejercicio debería resultar que si winneropt=optionid, entonces el outcome debería ser ratio-bet, en caso de que no sea null pero sean distintos los valores, el outcome debería ser 0

bet	ratio	outcome	winneropt	optionid
79.71	9.01	0	120	117
23.30	8.42	196.1860	47	47
74.58	5.92	441.5136	51	51
94.57	9.69	916.3833	86	86

Rendimiento y optimización

En este caso la única lectura secuencial de las tablas es necesaria para la actualización de todos los valores. Se observa otra lectura secuencial pero de coste mínimo.

```
Update on clientbets (cost=333.20..20155.54 rows=4051 width=36)
-> Hash Join (cost=333.20..20155.54 rows=4051 width=36)
    Hash Cond: ((clientbets.optionid = bets.winneropt) AND (
        clientbets.betid = bets.betid))
-> Seq Scan on clientbets (cost=0.00..14749.40 rows=669640 width=30)
-> Hash (cost=220.88..220.88 rows=7488 width=14)
    -> Seq Scan on bets (cost=0.00..220.88 rows=7488 width=14)
```

2.1.3. setOrderTotalOutcome

Comprobación de los resultados De nuevo se observa el primer pedido sin outcome y simplemente hace falta fijarse en los outcome del ejercicio anterior y sumarlos para saber el resultado.

customerid	date	orderid	totalamount	totaloutcome
693	2012-12-02 17:23:39+01	1	272.16	

Se comprueba que efectivamente el resultado coincide.

customerid	date	orderid	totalamount	totaloutcome
693	2012-12-02 17:23:39+01	1	272.16	1554.0829

Rendimiento y optimización Se muestra ya el coste tras construir los índices oportunos.

```
Update on clientbets (cost=333.20..20155.54 rows=4051 width=36)
-> Hash Join (cost=333.20..20155.54 rows=4051 width=36)
    Hash Cond: ((clientbets.optionid = bets.winneropt) AND (
        clientbets.betid = bets.betid))
```



```
-> Seq Scan on clientbets (cost=0.00..14749.40 rows=669640 width=30)
-> Hash (cost=220.88..220.88 rows=7488 width=14)
    -> Seq Scan on bets (cost=0.00..220.88 rows=7488 width=14)
```

2.2. Triggers

Se incluyen los tres triggers solicitados:

1. updBets: actualiza el campo outcome de las apuestas realizadas por un cliente una vez se ha designado un ganador.
2. updOrders: actualiza el campo totalamount y totaloutcome del carrito cuando se modifica por el *trigger* anterior.
3. updCredit: actualiza el campo credit del cliente cuando un carrito se cierra o se modifica uno cerrado (asignación de ganadores).

El correcto funcionamiento de estos *triggers* se puede observar mediante el fichero `tests.sql`.

Parte del código se ha reutilizado del apartado anterior.

Nota: Debido a que parte de la funcionalidad de los apartados anteriores es realizada por los *triggers*, no es nada recomendable ejecutar de nuevo las actualizaciones, pues de hacerlo se actualizarán todas las filas de las tablas generando tantas ejecuciones de los *triggers* como filas tienen las tablas.

3. Adaptación a la aplicación Web

Para esta tarea se ha realizado un fichero SQL llamado `adapta.sql` que realiza ciertas adaptaciones sobre la base de datos proporcionada para dar menos relevancia a los atributos de la base de datos que no tienen relación con la aplicación o una función dentro de la misma.

3.1. Cambios realizados

Los cambios realizados son:

- Quitar restricción `not null` en atributo `firstname` de la tabla `customers`
- Quitar restricción `not null` en atributo `lastname` de la tabla `customers`
- Quitar restricción `not null` en atributo `address1` de la tabla `customers`
- Quitar restricción `not null` en atributo `city` de la tabla `customers`
- Quitar restricción `not null` en atributo `country` de la tabla `customers`
- Quitar restricción `not null` en atributo `region` de la tabla `customers`
- Quitar restricción `not null` en atributo `creditcardtype` de la tabla `customers`
- Cambiar los atributos de actualización secuencial de las ids de todas las tablas, de manera que al insertar se inserten automáticamente con `id = max(ids)`

3.2. Funcionalidad de la página web implementada

Se ha implementado una interacción de la web con la base de datos casi completa:

- Se ha integrado con la parte de gestión y creación de usuarios
- Se ha integrado con la parte de visualización de las apuestas y el sistema creación y apuestas.
- Se ha integrado la gestión del carrito con la base de datos, de manera que el carrito se guarda en la base de datos.
- Se ha implementado un sistema de carga incremental mediante AJAX para evitar el envío de un gran volumen de datos en la carga de la página principal.

El objetivo final logrado ha sido integrar totalmente la página desarrollada con la base de datos, de tal manera que la página es totalmente funcional y todo el manejo se realiza usando la base de datos.