

Práctica 3

Sistemas **Informáticos**

Patricia Anza Mateos
Óscar García de Lara Parreño
Grupo: 1312
Pareja: 1

Diagrama entidad-relación de la base de datos inicial

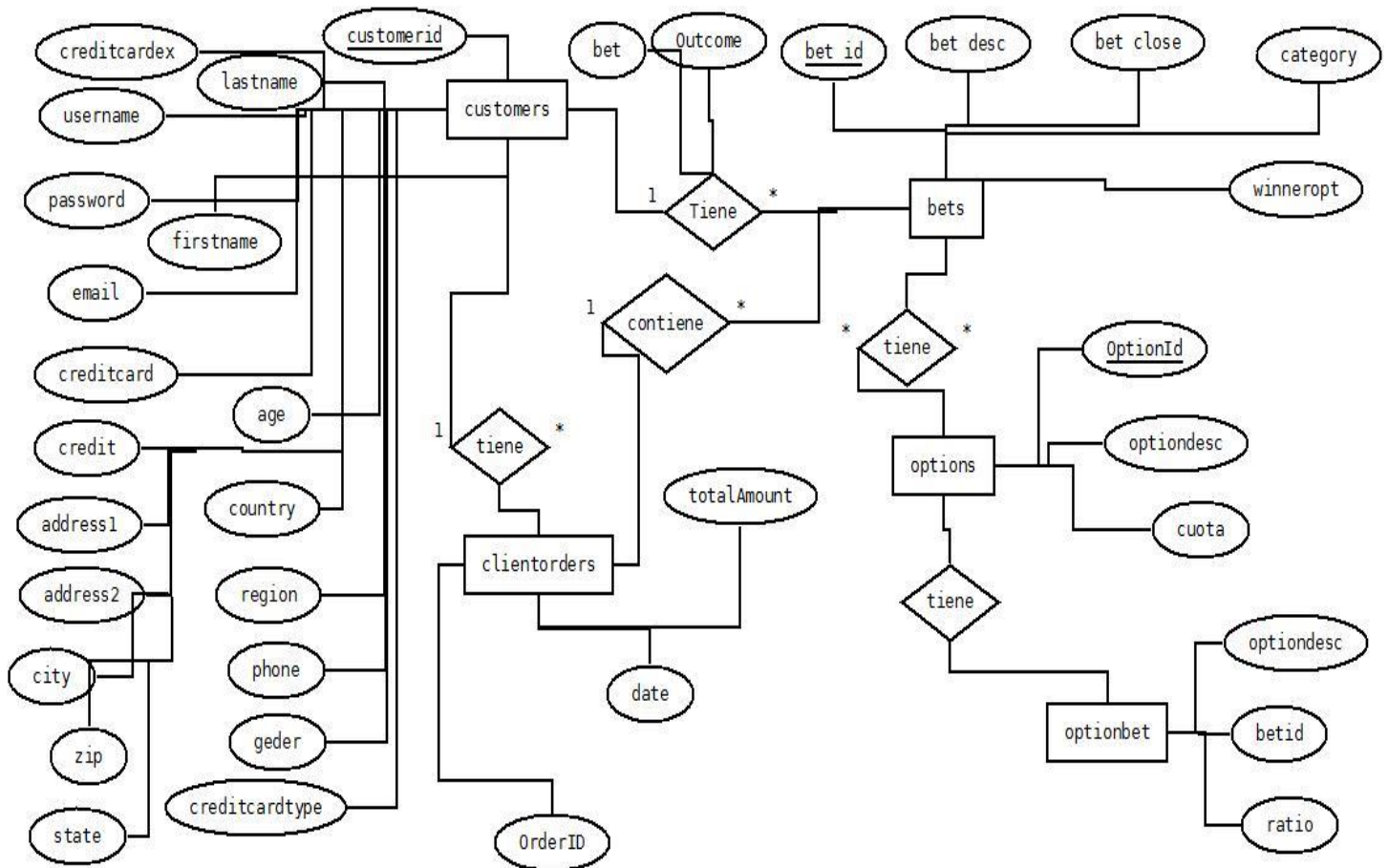
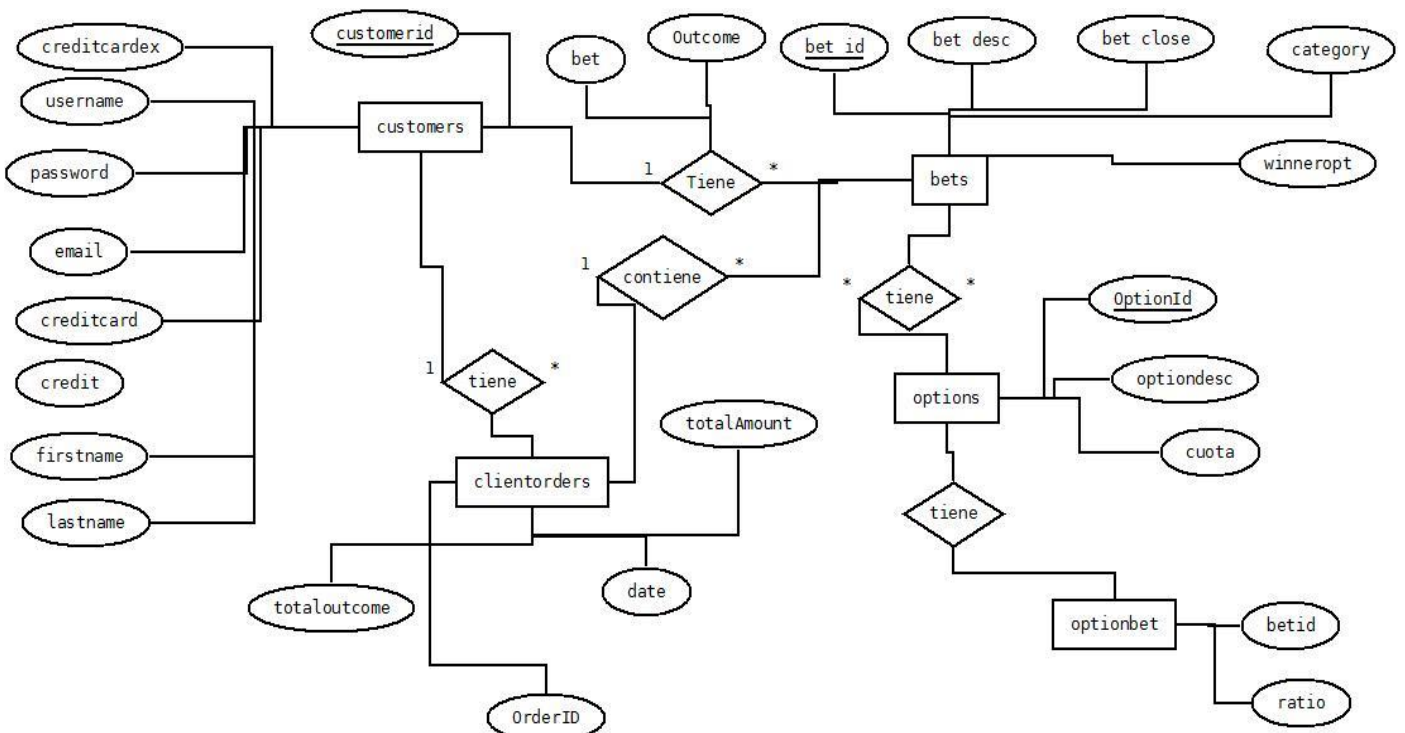


Diagrama entidad-relación de la base de datos tras los cambios



Hemos actualizado los campos de la tabla customer, eliminando los que no creíamos necesarios en la base de datos. Los campos eliminados de la tabla customer son: address1, address2, city, zip, state, country, region, pone, creditcardtype, age y gender. En esta tabla hemos decidido que el email sea no nulo, ya que para hacer nuestro log in el usuario debe acceder poniendo su email. Por otro lado, en la tabla options hemos eliminado el atributo category ya que este podía ser tomado de la tabla bets.

De la tabla clientbets hemos eliminado el campo customerid ya que podía ser cogido de la tabla customers.

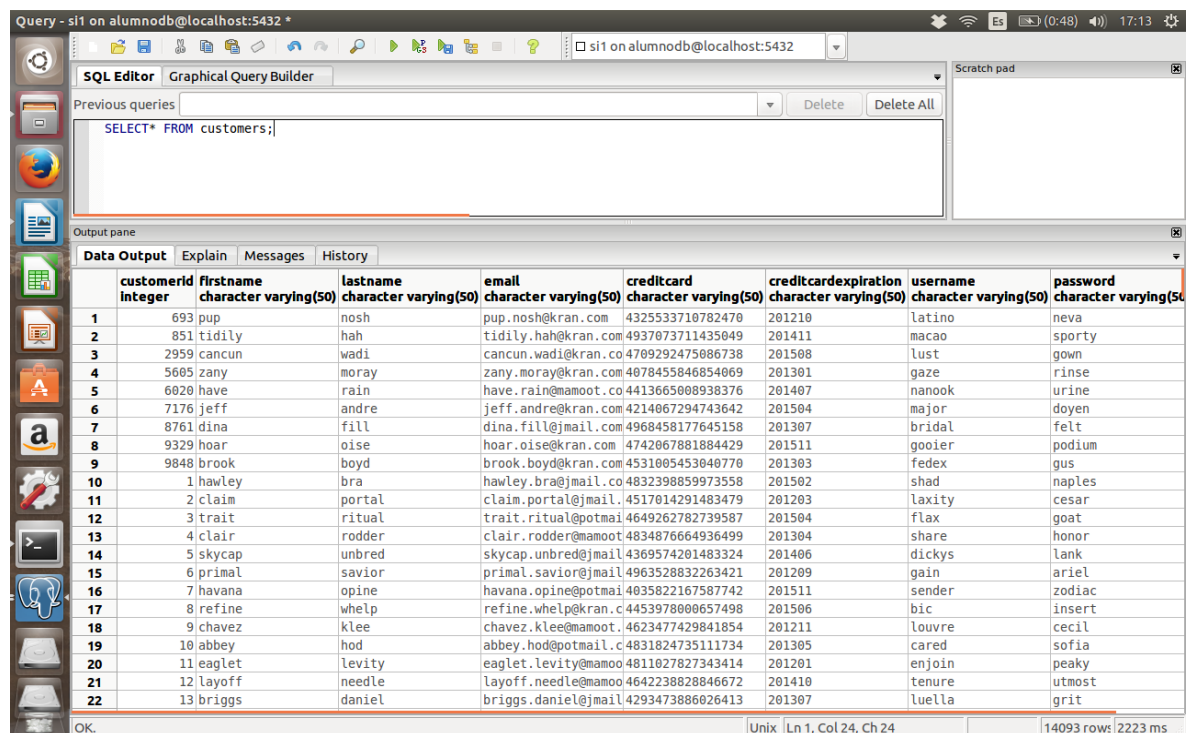
Lo mismo ocurre con el campo optiondesc de optionbet que podía ser cogido de la tabla options.

En la tabla clientorders hemos añadido un nuevo atributo, totalotcome, que calcula el total de dinero ganado. Este campo se actualiza en la consulta que está en setOutcomebets.sql.

Por último hemos añadido un índice en la tabla customers en el atributo email para que la búsqueda en la base de datos resulte más rápida y sencilla.

Todas estas actualizaciones se encuentran en el archivo actualiza.sql.

Tras actualizar la tabla de customers nos queda lo siguiente:



	customerid integer	firstname character varying(50)	lastname character varying(50)	email character varying(50)	creditcard character varying(50)	creditcardexpiration character varying(50)	username character varying(50)	password character varying(50)
1	693	pup	nosh	pup.nosh@kran.com	4325533710782470	201210	latino	neva
2	851	tidily	hah	tidily.hah@kran.com	4937073711435049	201411	macao	sporty
3	2959	cancun	wadi	cancun.wadi@kran.com	4709292475086738	201508	lust	gown
4	5605	zany	moray	zany.moray@kran.com	4078455846854069	201301	gaze	rinse
5	6020	have	rain	have.rain@mamoot.co	4413665008938376	201407	nanook	urine
6	7176	jeff	andre	jeff.andre@kran.com	4214067294743642	201504	major	doyen
7	8761	dina	fill	dina.fill@gmail.com	4968458177645158	201307	bridal	felt
8	9329	hoar	oise	hoar.oise@kran.com	4742067881884429	201511	gooier	podium
9	9848	brook	boyd	brook.boyd@kran.com	4531005453840770	201303	fedex	gus
10	1	hawley	bra	hawley.bra@gmail.com	4832398859973558	201502	shad	naples
11	2	claim	portal	claim.portal@gmail.com	4517014291483479	201203	laxity	cesar
12	3	trait	ritual	trait.ritual@potmai	4649262782739587	201504	flax	goat
13	4	clair	rodder	clair.rodder@mamoot	4834876664936499	201304	share	honor
14	5	skycap	unbred	skycap.unbred@gmail.com	4369574201483324	201406	dickys	lank
15	6	primal	savior	primal.savior@gmail.com	4963528832263421	201209	gain	ariel
16	7	havana	opine	havana.opine@potmai	4035822167587742	201511	sender	zodiac
17	8	refine	whelp	refine.whelp@kran.com	4453978000657498	201506	bic	insert
18	9	chavez	kleee	chavez.kleee@mamoot	4623477429841854	201211	louvre	cecil
19	10	abbey	hod	abbey.hod@potmai	4831824735111734	201305	cared	sofia
20	11	eaglet	levity	eaglet.levity@mamoo	4811027827343414	201201	enjoin	peaky
21	12	layoff	needle	layoff.needle@mamoo	4642238828466672	201410	tenure	utmost
22	13	briggs	daniel	briggs.daniel@gmail.com	4293473886026413	201307	luella	grit

Query - si1 on alumnodb@localhost:5432 *

SQL Editor Graphical Query Builder

Previous queries Delete Delete All

```
SELECT * FROM customers;
SELECT * FROM clientorders;
SELECT * FROM clientbets;
```

Output pane

Data Output Explain Messages History

	optionid Integer	bet numeric	ratio numeric	outcome numeric	betid Integer	orderid Integer
1	68	94.58	2.52		1137	35645
2	58	29.35	9.11		1131	35645
3	87	25.19	4.67		1099	29500
4	88	101.01	3.65		1097	29500
5	87	73.18	6.71		1099	126402
6	72	20.45	6.03		1129	126402
7	60	95.42	1.40		1135	126402
8	54	73.94	1.13		1123	126402
9	85	20.18	5.83		1103	126402
10	39	77.67	9.97		1093	126402
11	41	82.03	10.00		1097	126402
12	50	47.77	10.00		1115	126402
13	71	29.42	10.96		1131	94923
14	81	80.80	6.02		1111	94923
15	49	81.63	2.05		1113	94923
16	44	65.94	8.05		1103	94923
17	57	61.05	2.96		1129	94923
18	66	73.86	5.48		1141	94923
19	61	56.73	1.78		1137	94923
20	81	49.78	7.89		1111	105802
21	48	37.25	5.66		1111	21206
22	63	55.01	10.35		1141	21206

OK. Unix Ln 3, Col 1, Ch 54 25 chars 669638 row 44934 ms

Tras actualizar la tabla de clientbets nos queda esto.

Hemos eliminado el id del cliente y el campo outcome sale a null y no se actualizará hasta que se realice una de las queries,

Tras actualizar la tabla de optionbet nos queda lo siguiente:

Query - si1 on alumnodb@localhost:5432 *

SQL Editor Graphical Query Builder

Previous queries Delete Delete All

```
SELECT * FROM customers;
SELECT * FROM clientorders;
SELECT * FROM clientbets;
SELECT * FROM optionbet;
```

Output pane

Data Output Explain Messages History

	optionid Integer	betid Integer	ratio numeric
1	108	1	1.871088
2	109	1	12.244744
3	1	1	12.049418
4	108	2	1.871088
5	109	2	12.244744
6	1	2	12.049418
7	107	3	1.600600
8	110	3	2.805263
9	1	3	2.118980
10	107	4	1.600600
11	110	4	2.805263
12	1	4	2.118980
13	122	5	3.519047
14	111	5	1.480961
15	1	5	2.282888
16	122	6	3.519047
17	111	6	1.480961
18	1	6	2.282888
19	121	7	3.135133
20	112	7	1.567561
21	1	7	2.216875
22	121	8	3.135133

OK. Unix Ln 4, Col 1, Ch 80 24 chars 16068 rows 865 ms

Como se puede observar se elimina el campo optiondesc.

Tras actualizar la tabla de clientorders nos queda lo siguiente:

The screenshot shows a SQL client window titled "Query - si1 on alumnodb@localhost:5432". The "SQL Editor" tab is active, displaying the following SQL query:

```
SELECT * FROM customers;  
SELECT * FROM clientorders;
```

The "Output pane" is visible below the editor, showing the results of the query. The "Data Output" tab is selected, displaying a table with the following columns: customerid (Integer), date (timestamp with time zone), orderid (Integer), totalamount (numeric), and totaloutcome (Integer). The table contains 22 rows of data, with the first row having a customerid of 693 and a date of 2012-12-02 17:23:39+01. The last row has a customerid of 851 and a date of 2012-10-01 00:50:42+02. The totalamount and totaloutcome columns are currently empty for all rows.

customerid Integer	date timestamp with time zone	orderid Integer	totalamount numeric	totaloutcome Integer
1	693 2012-12-02 17:23:39+01	1		
2	693 2013-04-20 10:52:59+02	2		
3	693 2012-10-29 00:14:59+01	3		
4	693 2013-02-14 08:13:26+01	4		
5	693 2012-11-08 07:51:08+01	5		
6	851 2012-11-03 08:11:02+01	6		
7	851 2013-01-20 04:53:57+01	7		
8	851 2012-10-28 22:10:32+01	8		
9	851 2012-12-21 16:44:32+01	9		
10	851 2013-01-21 21:25:07+01	10		
11	851 2013-05-06 12:33:52+02	11		
12	851 2012-11-26 06:32:29+01	12		
13	851 2013-05-02 07:07:31+02	13		
14	851 2012-11-15 01:06:20+01	14		
15	851 2013-02-25 21:47:24+01	15		
16	851 2013-02-10 17:27:46+01	16		
17	851 2012-12-12 23:56:43+01	17		
18	851 2012-10-17 02:43:43+02	18		
19	851 2013-01-24 16:35:05+01	19		
20	851 2013-02-09 19:22:00+01	20		
21	851 2013-04-22 12:59:16+02	21		
22	851 2012-10-01 00:50:42+02	22		

The status bar at the bottom indicates "OK", "Unix", "Ln 2, Col 1, Ch 26", "27 chars", "149030 rows", and "11375 ms".

Como se puede observar el campo totaloutcome y totalamount se inicializan como null y hasta que no realicemos el procedimiento adecuado no se van a actualizar.

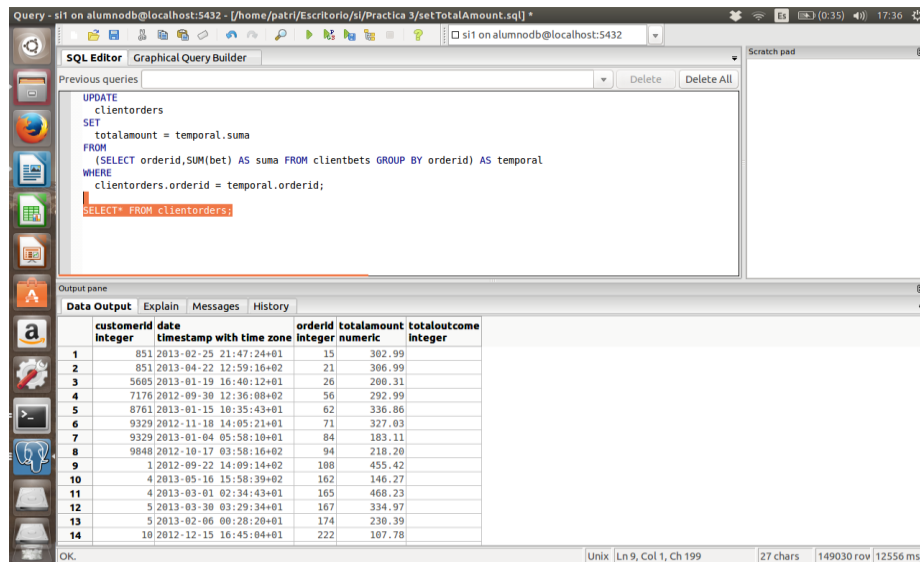
Análisis de las soluciones dadas para las diferentes tareas

1. Elaboración de la consulta setTotalAmount:

```
UPDATE
  clientorders
SET
  totalamount = temporal.suma
FROM
  (SELECT orderid,SUM(bet) AS suma FROM clientbets GROUP
  BY orderid) AS temporal
WHERE
  clientorders.orderid = temporal.orderid;
```

En esta consulta hemos ido actualizando el atributo totalamount de la tabla de clientorders sumando las apuestas cuyo orderid era igual al de la tabla de clientorders. De esta manera nos quedaremos con el dinero de las apuestas por las que el cliente ha apostado.

El resultado es el siguiente:



The screenshot shows a SQL IDE window with the following components:

- SQL Editor:** Contains the SQL query for updating the totalamount in the clientorders table.
- Output pane:** Displays the results of the query execution, including a table of updated data and a status bar.

Data Output Table:

	customerid integer	date timestamp with time zone	orderid integer	totalamount numeric	totaloutcome integer
1	851	2013-02-25 21:47:24+01	15	302.99	
2	851	2013-04-22 12:59:16+02	21	306.99	
3	5605	2013-01-19 16:40:12+01	26	200.31	
4	7176	2012-09-30 12:36:00+02	56	292.99	
5	8761	2013-01-15 10:35:43+01	62	336.86	
6	9329	2012-11-18 14:05:21+01	71	327.03	
7	9329	2013-01-04 05:50:10+01	84	103.11	
8	9848	2012-10-17 03:58:16+02	94	218.20	
9	1	2012-09-22 14:09:14+02	108	455.42	
10	4	2013-05-16 15:58:39+02	162	146.27	
11	4	2013-03-01 02:34:43+01	165	468.23	
12	5	2013-03-30 03:29:34+01	167	334.97	
13	5	2013-02-06 00:28:20+01	174	230.39	
14	10	2012-12-15 16:45:04+01	222	107.78	

Status Bar: OK. | Unix | Ln 9, Col 1, Ch 199 | 27 chars | 149030 rov | 12556 ms

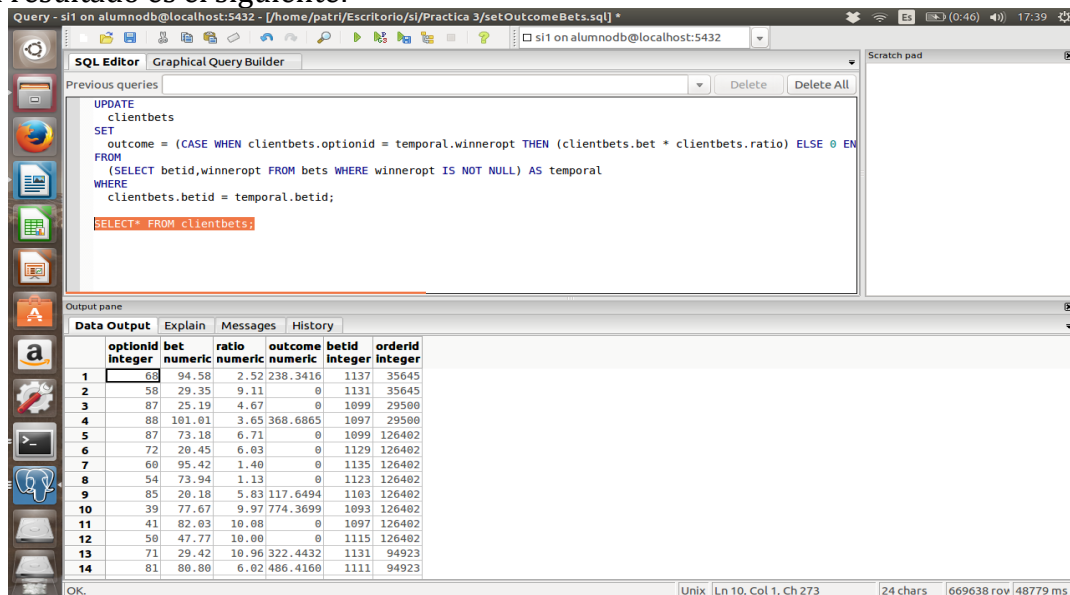
El campo totalamount actualizado.

2. Elaboración de la consulta setOutcomeBets.sql:

```
UPDATE
  clientbets
SET
  outcome = (CASE WHEN clientbets.optionid =
temporal.winneropt THEN (clientbets.bet * clientbets.ratio)
ELSE 0 END)
FROM
  (SELECT betid,winneropt FROM bets WHERE winneropt IS
NOT NULL) AS temporal
WHERE
  clientbets.betid = temporal.betid;
```

En esta consulta actualizamos el atributo outcome de la tabla clientbets. Para ello debemos comprobar primero que la apuesta está cerrada (comprobamos que winneropt no es nulo) y que el id de la apuesta sea el mismo que la apuesta del cliente. Una vez tenemos eso actualizamos el valor en el que comprobamos si la opción marcada por el cliente coincide con la opción ganadora. Si es así multiplicaremos la apuesta por el ratio y lo asignamos. En caso contrario se asignará cero.

El resultado es el siguiente:



	optionid Integer	bet numeric	ratio numeric	outcome numeric	betid Integer	orderid Integer
1	68	94.58	2.52	238.3416	1137	35645
2	58	29.35	9.11	0	1131	35645
3	87	25.19	4.67	0	1099	29500
4	88	101.01	3.65	368.6865	1097	29500
5	87	73.18	6.71	0	1099	126402
6	72	20.45	6.03	0	1129	126402
7	60	95.42	1.40	0	1135	126402
8	54	73.94	1.13	0	1123	126402
9	85	20.18	5.83	117.6494	1103	126402
10	39	77.67	9.97	774.3699	1093	126402
11	41	82.03	10.08	0	1097	126402
12	50	47.77	10.00	0	1115	126402
13	71	29.42	10.96	322.4432	1131	94923
14	81	80.80	6.02	486.4160	1111	94923

El campo outcome se ha actualizado.

3. Elaborar el procedimiento almacenado

setOrderTotalOutcome.sql:

```
CREATE OR REPLACE FUNCTION finalRes (integer)
RETURNS VOID AS $$
DECLARE
id ALIAS FOR $1;
BEGIN
IF(id = -1) THEN
    UPDATE clientorders SET totaloutcome = res.suma FROM
    (SELECT orderid, SUM(outcome) AS suma FROM clientbets
    GROUP BY orderid) AS res WHERE
    clientorders.orderid=res.orderid;
ELSE
    UPDATE clientorders SET totaloutcome = (SELECT
    SUM(outcome) FROM clientbets WHERE orderid=id GROUP
    BY orderid ) WHERE clientorders.orderid=id;
END IF;
END;
$$ LANGUAGE plpgsql;

SELECT finalRes(-1);
```

Para este procedimiento almacenado hemos creado una función a la que le pasamos un entero que será el id del pedido del cuál queremos actualizar el totalOutcome. Si le pasamos -1 deberá actualizar toda la tabla. El procedimiento almacenado crea o actualiza esta función y como se puede ver si el entero es -1 se deberá actualizar toda la tabla de clientOrders y si no solo la que el id sea igual al oderid de la tabla de clientorders. Para hacer la consulta de este procedimiento nos hemos fijado en la de setTotalAmount.sql, pero en este caso sumamos el outcome de las apuestas del pedido.

El resultado es el siguiente:

The screenshot shows a SQL IDE window titled "Query - si1 on alumnodb@localhost:5432 - [/home/patrl/Escritorio/si/Practica 3/setOrderTotaloutcome.sql] *". The SQL Editor contains the following PL/SQL code:

```
DECLARE
    id ALIAS FOR $1;
BEGIN
    IF(id = -1) THEN
        UPDATE clientorders SET totaloutcome = res.suma FROM (SELECT orderid, SUM(outcome) AS suma FROM clientbets GP
    ELSE
        UPDATE clientorders SET totaloutcome = (SELECT SUM(outcome) FROM clientbets WHERE orderid=id GROUP BY orderid )
    END IF;
END;
$$ LANGUAGE plpgsql;

SELECT finalRes(-1);

SELECT* FROM clientorders;
```

The Output pane shows the results of the query execution. The "Data Output" tab is selected, displaying a table with the following columns: customerid, date, orderid, totalamount, and totaloutcome. The data is as follows:

customerid	date	orderid	totalamount	totaloutcome
851	2013-02-25 21:47:24+01	15	302.99	285
851	2013-04-22 12:59:16+02	21	306.99	1993
5685	2013-01-19 16:40:12+01	26	200.31	599
7176	2012-09-30 12:36:08+02	56	292.99	251
8761	2013-01-15 10:35:43+01	62	336.86	635
9329	2012-11-18 14:05:21+01	71	327.03	1535
9329	2013-01-04 05:58:10+01	84	183.11	751
9848	2012-10-17 03:58:16+02	94	218.20	1874
1	2012-09-22 14:09:14+02	108	455.42	1388
4	2013-05-16 15:58:39+02	162	146.27	572
4	2013-03-01 02:34:43+01	165	468.23	867
5	2013-03-30 03:29:34+01	167	334.97	1101
5	2013-02-06 00:28:20+01	174	230.39	809
10	2012-12-15 16:45:04+01	222	107.78	0

The status bar at the bottom indicates "OK", "Unix", "Ln 16, Col 1, Ch 500", "26 chars", "149030 rows", and "13089 ms".

El campo totaloutcome se ha actualizado.

4. Realizar un trigger updBets:

```
CREATE OR REPLACE FUNCTION finalResTrigger() RETURNS
trigger AS $$
BEGIN
IF (NEW.winneropt!=OLD.winneropt) THEN
  UPDATE CLIENTBETS SET outcome=
CASE WHEN clientbets.optionid = NEW.winneropt THEN
clientbets.bet*clientbets.ratio

  ELSE 0
END WHERE NEW.betid = clientbets.betid;
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS actualizaInfo on bets;
CREATE TRIGGER actualizaInfo
AFTER INSERT OR UPDATE OF winneropt
ON bets
FOR EACH ROW
EXECUTE PROCEDURE finalResTrigger();
```

Creamos un trigger llamado actualizaInfo que se dispara después de insertar o actualizar el atributo winneropt de la tabla bets. Si se actualiza llamará a la función finalResTrigger(). En esta función comprobamos que el valor nuevo de winneropt es distinto al antiguo. En el caso de que sea así actualizamos el outcome por el nuevo valor. Para la realización de la consulta nos hemos basado en la de setOutcomeBets.sql.

5. Realizar el trigger udtOrders.sql:

```
CREATE OR REPLACE FUNCTION updOrders() RETURNS
trigger AS $$
BEGIN
    IF (TG_OP = 'DELETE' ) THEN

        UPDATE clientorders SET totalamount = (SELECT
SUM(bet) FROM clientbets WHERE orderid= OLD.orderid
GROUP BY orderid)WHERE clientorders.orderid =
OLD.orderid;

        EXECUTE finalRes(OLD.orderid);

        RETURN NULL;
    ELSE

        UPDATE clientorders SET totalamount = (SELECT
SUM(bet) FROM clientbets WHERE orderid= NEW.orderid
GROUP BY orderid)WHERE clientorders.orderid =
NEW.orderid;
        EXECUTE finalRes(NEW.orderid);
        RETURN NEW;
    END IF;

END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS actualizaClientOrders on clientbets;

CREATE TRIGGER actualizaClientOrders
AFTER INSERT OR UPDATE OR DELETE
ON clientbets
FOR EACH ROW
EXECUTE PROCEDURE updOrders();
```

Hemos creado un trigger que cada vez que se inserte, elimine o actualice la tabla clientbets llame a la función updOrders.

Esta función comprobará primero si lo que se realiza es una eliminación. Si este es el caso se hará la consulta para actualizar el totalamount del carrito. Después de llamará al procedimiento almacenado con el id antiguo. En caso de que sea una inserción o una actualización se hará lo mismo pero se tomará el nuevo id del pedido. Nos hemos basado en la consulta del procedimiento almacenado.

6. Realizar el trigger updCredit:

```
CREATE OR REPLACE FUNCTION updCredit() RETURNS
trigger AS $$
BEGIN

    IF (NEW.date IS NOT NULL AND OLD.date IS NULL) THEN

        UPDATE customers SET credit = credit - NEW.totalamount
        WHERE customerid = NEW.customerid;

    ELSEIF (NEW.date IS NOT NULL AND 0=(SELECT
COUNT(winneropt) FROM bets WHERE winneropt IS NULL
AND betid IN (SELECT betid FROM clientbets WHERE
orderid=NEW.orderid))) THEN
        UPDATE customers SET credit = credit +
NEW.totaloutcome WHERE customerid = NEW.customerid;

    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS actualizaCreditCustomers on
clientorders;
CREATE TRIGGER actualizaCreditCustomers
AFTER UPDATE
ON clientorders
FOR EACH ROW
EXECUTE PROCEDURE updCredit();
```

Creamos un trigger que actualiza en crédito del cliente cada vez que se actualice la tabla clientorders. Cuando esto ocurre se llama a la función updCredit() que lo primero que te comprueba es que el carrito esté cerrado (la fecha es no nula) y que la fecha antigua fuera nula. Si esto se cumple se debe de actualizar el crédito quitando el dinero que finalmente se ha apostado. Si la apuesta ya se ha cerrado y todas sus apuestas ya se han jugado se suma a crédito el dinero ganado en las apuestas.

7. Implementar el login:

Hemos actualizado el login.php para que se pueda leer los datos de la base de datos. Para ello hemos conectado la base de datos en el php y hemos ido realizando consultas para acceder a los campos de customer de la base de datos y comprobar que estos son válidos (email y contraseña correctos). En nuestro caso nos hemos dado cuenta de que el email debe ser único, por tanto hacemos el login introduciendo el correo y la contraseña.

```
try
{
    $db= pg_connect("host=localhost dbname=si1
user=alumnodb password=alumnodb");
}
catch(Exception $e)
{
    echo "<script type=\"text/javascript\">alert(\"\".$e-
>getMessage().\".\");
window.location.href='login.php';</script>";
}
$result = pg_query_params($db, 'SELECT firstname
FROM customers WHERE email = $1 and password = $2',
array($mail,$psw));
if(!$result){

pg_close($db);
    echo "<script
type=\"text/javascript\">alert(\"Datos incorrectos\");
window.location.href='login.php';</script>";
}
$row = pg_fetch_row($result);
```

8. Implementar el carrito:

Para implementar el carrito hemos tenido que modificar la web para que esta funcionara a través de la base de datos. Por ello hemos cambiado el index.php para que leyera las apuestas de la base de datos, apuesta.php donde tenemos el detalle de la apuesta, el carrito.php en el que en esta práctica será obligatorio estar autenticado para usarlo. En esta parte indicaremos es que líneas de carrito.php esta las cosas.

Líneas 273-321:

Lo primeros que hacemos es obtener el id del carrito abierto del cliente, en el caso que no tenga ninguno se creara uno, después comprobamos que la apuesta que quiere realizar no esté ya en el carrito, si lo está abortamos, sino la insertamos.

Líneas 353-384:

En este momento imprimimos el carrito abierto que tenga el cliente, en caso de no tener ninguno aparece un mensaje indicándolo. La estructura del carrito sigue siendo el de la práctica anterior con unos botones para cambiar y eliminar cada apuesta y otro para procesar el carrito.

Líneas 239-258:

En caso de haber confirmado el cambio del dinero de la apuesta, se hace un update de esa apuesta, añadiendo el nuevo valor.

Líneas 209-227:

Este caso es cuando pide eliminar una apuesta.

Líneas 145-177:

Acá procesamos el carrito para cerrarlo, comprobamos que el cliente tenga suficiente crédito para pagar la apuesta, si es así cambiamos la fecha del carrito a la actual, un cambio importante en este momento con lo anterior es que no cambiamos el crédito ya que eso se encarga automáticamente un trigger.

9. Otros cambios

Se ha cambiado las barras laterales, para que sean dinámicas según las apuestas abiertas en ese momento ya que son las que se muestran en el índice. También se cambia buscar tanto la cabecera para que muestre las categorías dinámicamente como en la barra lateral, como el código de buscar.php para que obviamente trabaje con la base de datos.

10. Futuros cambios

Estaría bien cambiar la funcionalidad del trigger updCredit o crear otro aparte para cuando cierre la fecha, compruebe que las apuestas del carrito no se hayan cerrado mientras estaban en el carrito o cuando se cierre una apuesta se recorra los carritos abiertos y se elimine de los que la contenga.

También habría que cambiar el registro en la web ya que no usa la base de datos.