



APLICACIÓN DE AHORROS



Sergio Ruiz Ortiz

Fecha:15/02/2026

Curso:1ºDAW

ÍNDICE

ÍNDICE.....	1
DESCRIPCIÓN DEL PROYECTO.....	2
MODELO ENTIDAD RELACIÓN.....	4
MODELO RELACIONAL EN WORKBENCH.....	6
CAMBIOS CON SET Y ENUM.....	6
CONSULTAS SQL.....	7
CONCLUSIÓN.....	13

DESCRIPCIÓN DEL PROYECTO

Este proyecto trata sobre una aplicación para gestionar el dinero personal, pensada para ayudar a las personas a organizar sus ahorros y controlar sus gastos de una forma sencilla e intuitiva. He elegido este proyecto porque considero importante organizar el dinero de forma clara y responsable.


Cada usuario puede registrarse en la aplicación y se guardan sus datos personales y de acceso, como su identificador, nombre, apellidos, fecha de nacimiento, nombre de usuario, contraseña, correo electrónico y el país en el que vive. El usuario es el elemento principal del sistema, ya que es quien realiza todas las acciones dentro de la aplicación.

Una vez registrado, el usuario puede crear distintas metas de ahorro, por ejemplo ahorrar para un viaje o para una compra importante. De cada meta se guarda su identificador, el nombre de la meta, el estado en el que se encuentra, la prioridad, la cantidad total que se desea ahorrar y la cantidad que se ha conseguido hasta el momento. Además, se registran las fechas de inicio y de fin de cada meta.

Para gestionar el dinero dentro de las metas de ahorro, el usuario puede realizar transacciones, que permiten añadir o retirar dinero. Cada transacción queda asociada a un usuario y a una meta concreta, y se guarda el número de transacción, la cantidad, el tipo de movimiento (ingreso o retirada) y la fecha en la que se realiza.

Además, el usuario puede crear distintas listas de presupuesto, que sirven para establecer límites de gasto. Por ejemplo, se puede definir un presupuesto para comida, ocio o compras. De cada lista se guarda su identificador, el nombre, una descripción y la cantidad máxima que se puede gastar .

Las listas de presupuesto se organizan mediante categorías, que permiten identificar fácilmente el tipo de gasto. De cada categoría se guarda su identificador, el nombre, una breve descripción y el color del icono que la representa. La aplicación también incluye un sistema de notificaciones, que informa al usuario sobre distintos eventos



importantes, como el estado de una meta de ahorro o avisos relacionados con sus presupuestos.

De cada notificación se guarda su identificador, el motivo, el título, el mensaje y su prioridad, así como la fecha en la que se recibe.

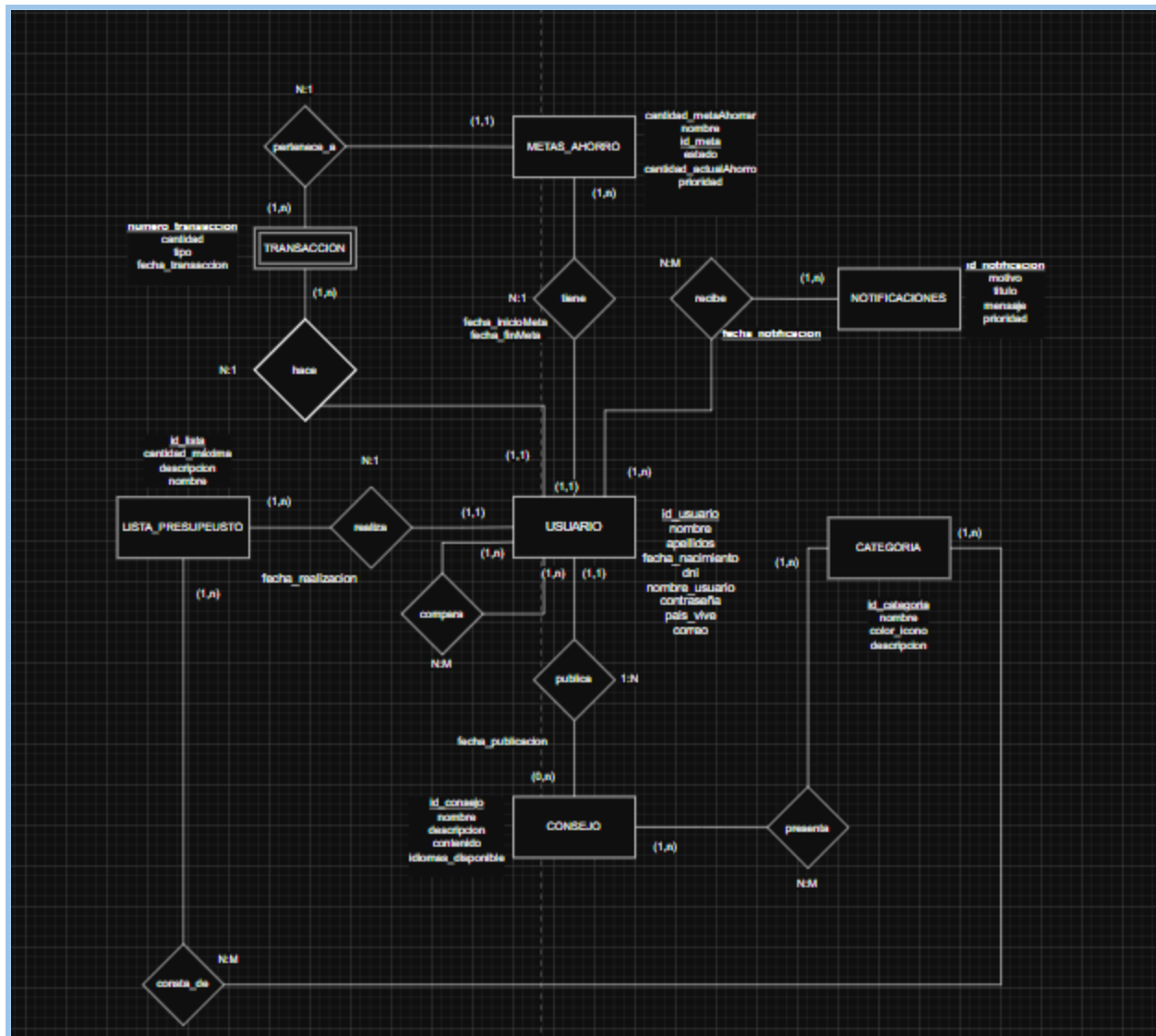
Por otro lado, los usuarios pueden publicar consejos relacionados con la gestión del dinero, con el objetivo de ayudar a otros usuarios. De cada consejo se guarda su identificador, el nombre, una descripción y el contenido, además de la fecha en la que se publica.

Por último, la aplicación permite que los usuarios se comparen entre sí, de manera que puedan ver su situación económica en relación con otros usuarios, fomentando un la competencia entre usuarios, lo que los va a llevar a tomárselo más en serio.

MODELO ENTIDAD RELACIÓN

Para que la aplicación funcione bien, lo primero que he hecho es organizar cómo se va a guardar toda la información. El centro de todo es el usuario, porque es quien hace todo en la app. A partir de ahí, he creado varias entidades que se conectan entre sí:

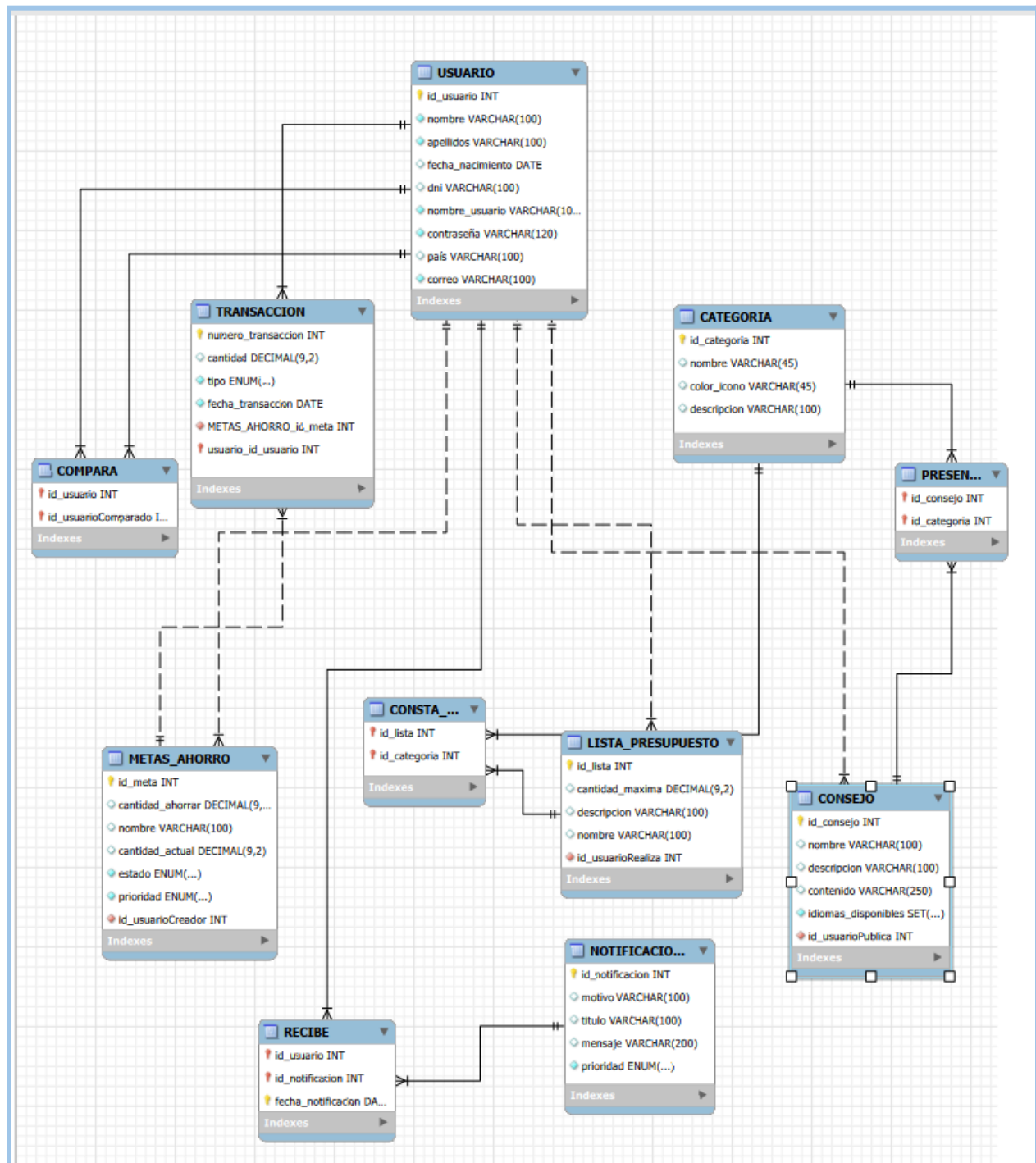
- **Usuario:** Aquí guardo los datos básicos de la persona, como su nombre, correo y país. Sin el usuario, nada de lo demás tendría sentido.
- **Metas de Ahorro:** Son los objetivos que se propone cada uno, cómo ahorrar para un viaje. Guardo cuánto quieren conseguir y qué prioridad tiene cada meta.
- **Transacciones:** Cada vez que alguien mete o saca dinero de una meta, se registra aquí. Así sabemos exactamente cuánto llevan ahorrado en cada momento.
- **Listas de Presupuesto:** Sirven para poner límites y no gastar de más en cosas como comida u ocio.
- **Categorías:** Las uso para organizar esos presupuestos. Por ejemplo, para saber si un gasto es de ocio o de comida.
- **Notificaciones:** Son los avisos que le llegan al usuario para decirle cómo van sus ahorros o si se está pasando de su presupuesto o cualquier otro aviso importante.
- **Consejos:** Es una parte donde los usuarios pueden escribir recomendaciones para ayudar a otros a ahorrar dinero.



MODELO RELACIONAL EN WORKBENCH

Cuando ya tenía el esquema claro, me pasé a MySQL Workbench para montar la base de datos de verdad. Aquí es donde creé las tablas que luego se usan en el servidor, intentando que todo fuera lo más fiel posible al modelo Entidad-Relación que hice al principio.

En el Workbench fui copiando lo que ya tenía hecho, pero aquí ya tenía que elegir bien el tipo de dato de cada atributo para que no me diese fallos. También hay que fijarse en las relaciones: las líneas continuas son para las relaciones fuertes y las discontinuas para las débiles, para que el diagrama se entienda perfectamente y no dé errores luego al conectarlas.



CAMBIOS CON SET Y ENUM

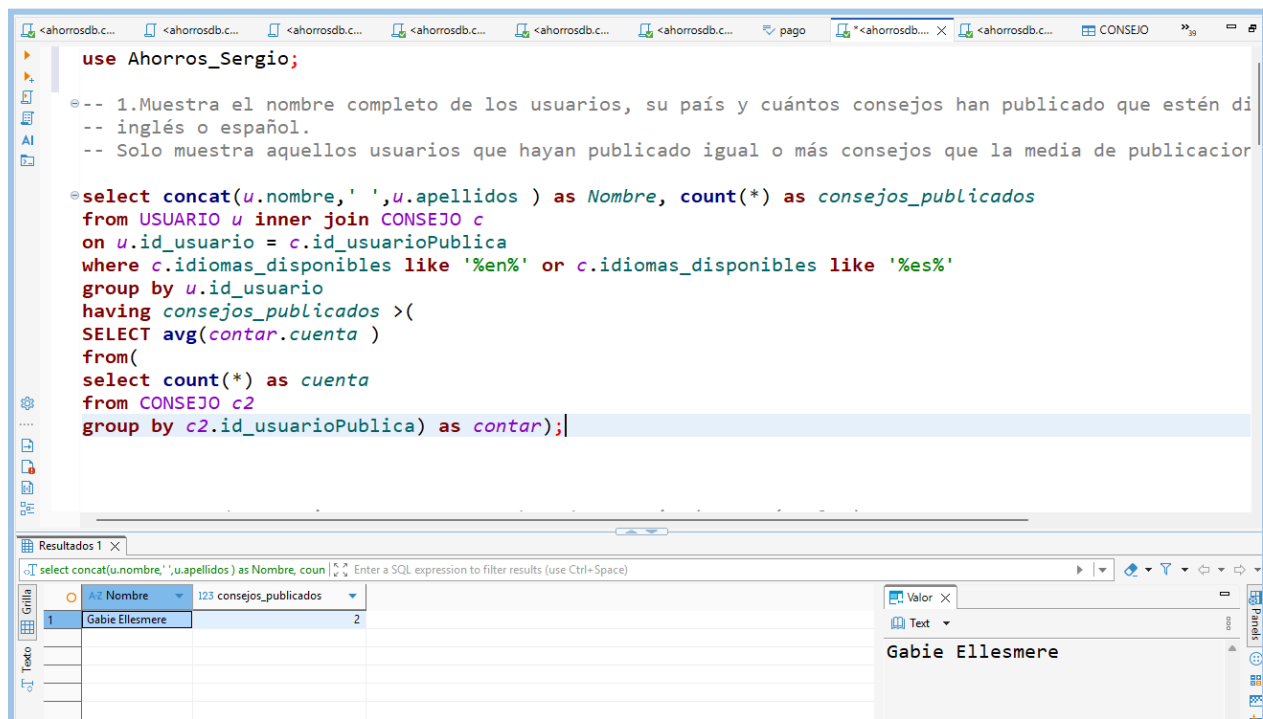
En esta parte del proyecto, decidí hacer unos ajustes en las tablas para que la base de datos fuera más profesional y no permitiera errores al meter los datos. En lugar de dejar que se pudiera escribir cualquier cosa, usé los tipos ENUM y SET para limitar las opciones:

- **ENUM:** Lo he aplicado en cosas como el estado de las metas o la prioridad. Así, solo se puede elegir entre opciones fijas como activo, pendiente o inactivo lo que puede servir para evitar que alguien escriba mal una palabra y luego las consultas no funcionen.
- **SET:** Este lo he usado para los Idiomas disponibles en los consejos, agregando la columna directamente a esa tabla. La diferencia con el ENUM es que aquí un mismo consejo puede estar en varios idiomas a la vez, como español e inglés, y el tipo SET me deja marcar varias opciones en una sola columna.

```
ALTER TABLE METAS_AHORRO MODIFY COLUMN estado ENUM('activo', 'pendiente', 'inactivo') NOT NULL DEFAULT 'inactivo';
ALTER TABLE METAS_AHORRO MODIFY COLUMN prioridad ENUM('baja', 'media', 'alta') not null default 'baja';
ALTER TABLE NOTIFICACIONES modify column prioridad ENUM('baja', 'media', 'alta') not null default 'baja';
ALTER TABLE TRANSACCION modify column tipo ENUM('compra', 'venta', 'transferencia') not null default 'compra';
alter table PRESENTA CHANGE id_coonsejo id_consejo INT;
ALTER TABLE CONSEJO CHANGE id_coonsejo id_consejo INT;
ALTER TABLE CONSEJO
ADD COLUMN idiomas_disponibles SET('es', 'en', 'fr', 'pt', 'de', 'it')
NOT NULL DEFAULT 'es' AFTER contenido;
```

CONSULTAS SQL

1. Muestra el nombre completo de los usuarios, su país y cuántos consejos han publicado que estén disponibles en al menos inglés o español. Solo muestra aquellos usuarios que hayan publicado igual o más consejos que la media de publicaciones global.



The screenshot shows a SQL IDE with a query editor and a results panel. The query is as follows:

```
use Ahorros_Sergio;

-- 1. Muestra el nombre completo de los usuarios, su país y cuántos consejos han publicado que estén di
-- inglés o español.
-- Solo muestra aquellos usuarios que hayan publicado igual o más consejos que la media de publicacione

select concat(u.nombre, ' ', u.apellidos) as Nombre, count(*) as consejos_publicados
from USUARIO u inner join CONSEJO c
on u.id_usuario = c.id_usuarioPublica
where c.idiomas_disponibles like '%en%' or c.idiomas_disponibles like '%es%'
group by u.id_usuario
having consejos_publicados > (
SELECT avg(contar.cuenta)
from(
select count(*) as cuenta
from CONSEJO c2
group by c2.id_usuarioPublica) as contar);
```

The results panel shows the following data:

Nombre	consejos_publicados
Gabie Ellesmere	2

2. Para cada usuario, muestra su nombre de usuario en mayúsculas y el nombre de su lista de presupuesto. Indica también el nombre de la categoría asociada a esa lista. Solo muestra aquellas listas que superen la media de gasto máximo.

The screenshot shows a database management tool interface. The top pane displays a SQL query with comments in Spanish. The bottom pane shows the results of the query in a table format.

SQL Query:

```
-- 2. Para cada usuario, muestra su nombre de usuario (en mayúsculas) y
-- el nombre de su lista de presupuesto. Indica también el nombre de la categoría
-- asociada a esa lista. Solo muestra aquellas listas que
-- superen la media de gasto máximo.

select UPPER(u.nombre ) as nombre, lp.nombre as nombre_lista, c.id_categoria
from USUARIO u inner join LISTA_PRESUPUESTO lp
on u.id_usuario = lp.id_usuarioRealiza
inner join CONSTA_DE cd
on lp.id_lista = cd.id_lista
inner join CATEGORIA c
on cd.id_categoria = c.id_categoria
where lp.cantidad_maxima > (
select avg(lp2.cantidad_maxima )
from LISTA_PRESUPUESTO lp2);
```

Query Results:

	AZ nombre	AZ nombre_lista	123 id_categoria
1	MILTY	Control Financiero	3
2	VIVYAN	Presupuesto Familiar	4
3	ARDRA	Presupuesto Básico	5
4	GUILLAUME	Resumen Financiero	6
5	BEVERLEY	Presupuesto General	1
6	LILLIE	Presupuesto Mensual	3
7	RUTHERFORD	Control Mensual	4
8	JANOT	Plan de Gastos	6
9	ANDI	Gastos Personales	2
10	VIRGIL	Presupuesto Personal	3
11	ELLI	Plan Financiero	4
12	EMMET	Presupuesto Familiar	7
13	CLARITA	Control Financiero	1
14	HARTLEY	Presupuesto Mensual	2
15	CONNOR	Resumen Financiero	4
16	WINSLOW	Control Mensual	5
17	PERKIN	Presupuesto del Hogar	7
18	QUINTUS	Mi Presupuesto	3
19	GERALD	Mi Presupuesto	4
20	TADIO	Presupuesto General	1
21	TARYN	Presupuesto General	3
22	ELVINA	Lista de Gastos	4
23	BRIA	Control Mensual	6

3. Calcula cuántos días han pasado desde que cada usuario recibió su última notificación. Muestra el nombre del usuario, la fecha de la última notificación y los días transcurridos. Si nunca ha recibido una, usamos IFNULL para poner "0".

```
-- 3. Calcula cuántos días han pasado desde que cada usuario recibió su última notificación.
-- Muestra el nombre del usuario, la fecha de la última notificación y los días transcurridos. Si nunca ha recibido una, usa IFNULL para poner "0".
create view consulta_3 as
select u.nombre as nombre_usuario,
ifnull(max(r.fecha_notificacion), 'no le ha llegado') as ultima_fecha,
ifnull(DATEDIFF(NOW(), r.fecha_notificacion), 'no tiene') as dias_diferencia
from USUARIO u left join RECIBE r
on u.id_usuario = r.id_usuario
group by u.id_usuario ;

-- 4. Compara lo que un usuario ha gastado de verdad con su presupuesto. Muestra el
-- su presupuesto máximo, el total de sus transacciones y su saldo restante.
create view consulta_4 as
select concat(left(u.apellidos, LENGTH(u.apellidos)-1), UPPER(right(u.apellidos, 1))) as nombre_usuario,
(select sum(t2.cantidad) from transacciones t2 where t2.usuario = u.id_usuario) as cantidad_gastada,
(select max(presupuesto) from usuarios u where u.id_usuario = u.id_usuario) as presupuesto_maximo,
(select sum(t2.cantidad) from transacciones t2 where t2.usuario = u.id_usuario) as cantidad_gastada
```

SUARIO 1 X

select u.nombre as nombre_usuario, ifnull(max(r.fecha_n | Enter a SQL expression to filter results (use Ctrl+Space)

	A-Z nombre_usuario	A-Z ultima_fecha	A-Z dias_diferencia
1	Gabie	no le ha llegado	no tiene
2	Milty	no le ha llegado	no tiene
3	Vivyan	no le ha llegado	no tiene
4	Ardra	no le ha llegado	no tiene

4. Compara lo que un usuario ha gastado de verdad con su presupuesto. Muestra el apellido con su última letra en mayúscula, su presupuesto máximo, el total de sus transacciones y su saldo restante.

The screenshot shows a SQL IDE with a query editor and a results pane. The query editor contains the following SQL code:

```
group by u.id_usuario ;

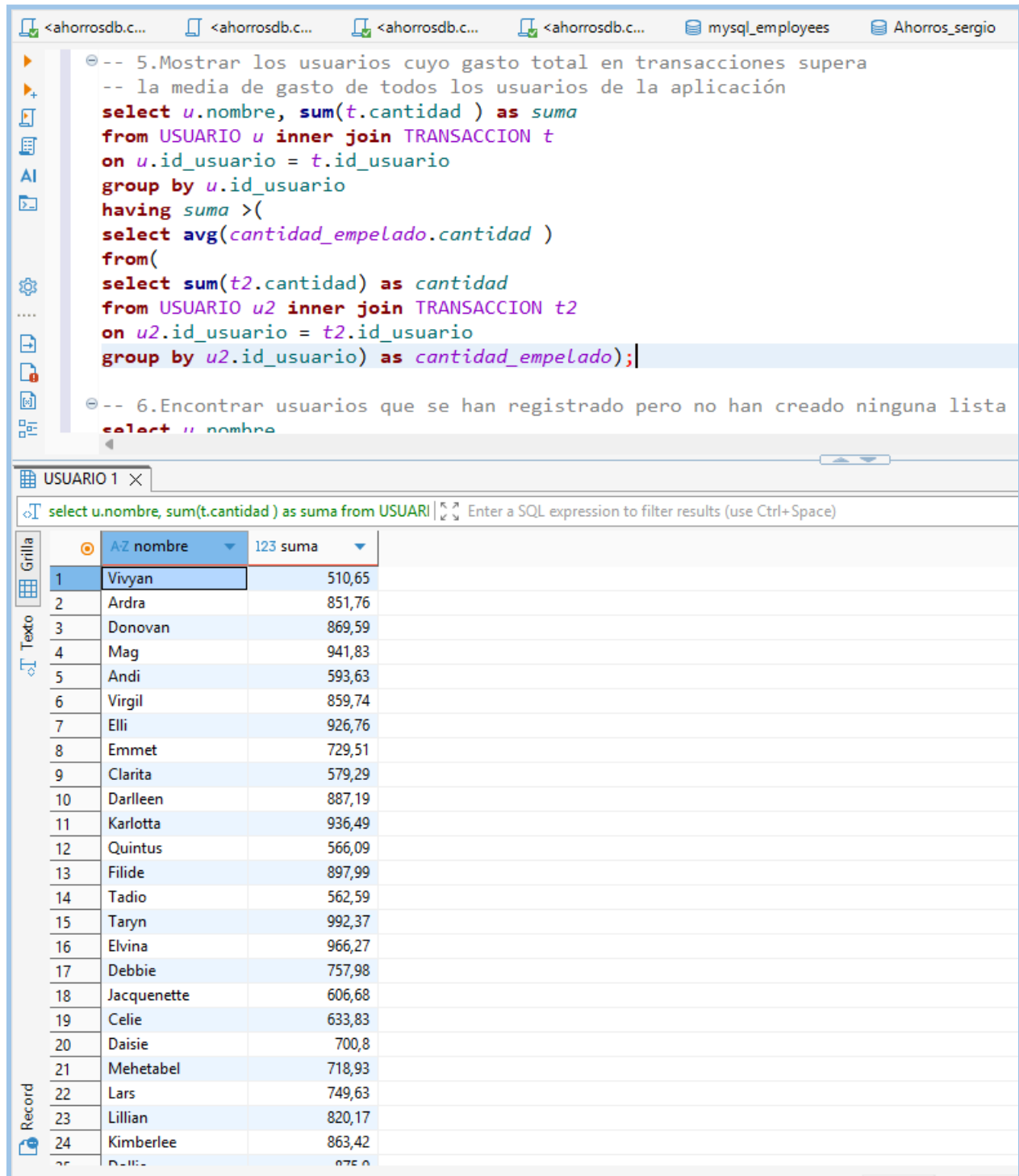
-- 4. Compara lo que un usuario ha gastado de verdad con su presupuesto
-- su presupuesto máximo, el total de sus transacciones y su saldo
create view consulta_4 as
select concat(left(u.apellidos, LENGTH(u.apellidos)-1), UPPER(right(
, (lp.cantidad_maxima - (SELECT sum(t2.cantidad )
from USUARIO u2 inner join TRANSACCION t2
on u2.id_usuario = t2.id_usuario
where u.id_usuario = u2.id_usuario )) as comparación, lp.cantidad_m
count(distinct t.numero_transaccion ) as numero_transacciones
from TRANSACCION t INNER JOIN USUARIO u
ON t.id_usuario = u.id_usuario
inner join LISTA_PRESUPUESTO lp
on u.id_usuario = lp.id_usuarioRealiza
group by u.id_usuario;

-- 5. Mostrar los usuarios cuyo gasto total en transacciones supera
-- la media de gasto de todos los usuarios de la aplicación
select u.nombre, sum(t.cantidad ) as suma
```

The results pane shows a table with the following data:

	AZ concat(left(u.apellidos, LENGTH(u.apellidos)-1), UP	123 comparación	123 cantidad_maxima	123 numero_transacciones
1	EllesmerE	3.916,6	4.392,63	1
2	SumptoN	5.748,54	6.002,29	1
3	MulvaneY	5.572,16	6.082,81	1
4	KynseY	6.382,81	7.234,57	1
5	GentreaU	6.998,58	7.242,45	1
6	De BeauchemP	1.282,32	1.324,07	1
7	Suddell	7.343,63	7.387,93	1
8	MinoR	3.666,6	4.536,19	1
9	RhymeR	6.672,79	6.831,45	1
10	McClintoN	7.532,89	7.688,24	1
11	BirkwoodD	-560,95	380,88	1
12	JoselsoN	4.984,25	5.003,91	1
13	MellY	2.046,06	2.397,3	1
14	MoxsoN	1.426,01	1.739,22	1
15	WillAV	5.100,70	5.701,01	1

5. Mostrar los usuarios cuyo gasto total en transacciones supera la media de gasto de todos los usuarios de la aplicación



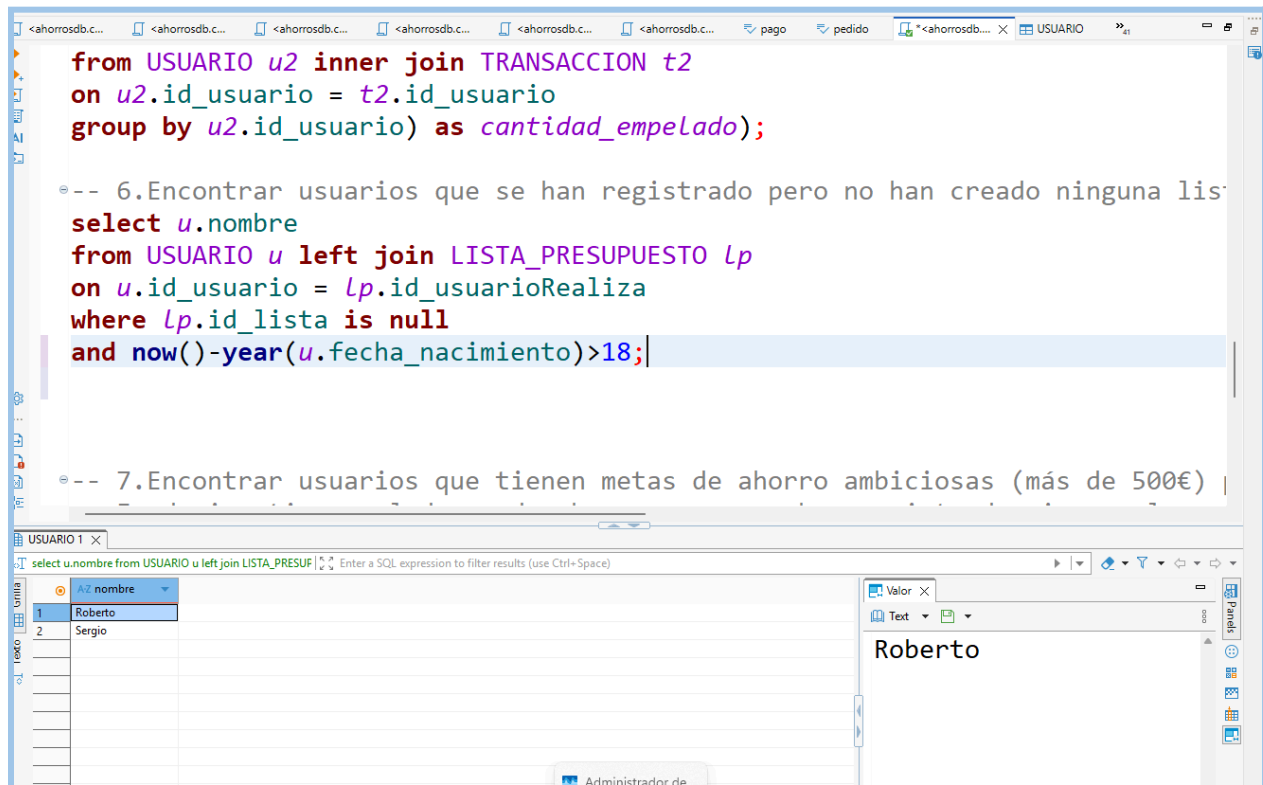
The screenshot shows a MySQL IDE window with a SQL query editor and a results grid. The query is designed to find users whose total transaction amount exceeds the average transaction amount of all users.

```
-- 5. Mostrar los usuarios cuyo gasto total en transacciones supera
-- la media de gasto de todos los usuarios de la aplicación
select u.nombre, sum(t.cantidad) as suma
from USUARIO u inner join TRANSACCION t
on u.id_usuario = t.id_usuario
group by u.id_usuario
having suma > (
select avg(cantidad_emplado.cantidad)
from(
select sum(t2.cantidad) as cantidad
from USUARIO u2 inner join TRANSACCION t2
on u2.id_usuario = t2.id_usuario
group by u2.id_usuario) as cantidad_emplado);
```

Below the query editor, the results are displayed in a table grid. The table has two columns: 'A-Z nombre' and 'suma'. The results show 24 rows of data, with the first row highlighted in blue.

	A-Z nombre	suma
1	Vivyan	510,65
2	Ardra	851,76
3	Donovan	869,59
4	Mag	941,83
5	Andi	593,63
6	Virgil	859,74
7	Elli	926,76
8	Emmet	729,51
9	Clarita	579,29
10	Darlleen	887,19
11	Karlotta	936,49
12	Quintus	566,09
13	Filide	897,99
14	Tadio	562,59
15	Taryn	992,37
16	Elvina	966,27
17	Debbie	757,98
18	Jacquenette	606,68
19	Celie	633,83
20	Daisie	700,8
21	Mehetabel	718,93
22	Lars	749,63
23	Lillian	820,17
24	Kimberlee	863,42

6. Encontrar usuarios que se han registrado pero no han creado ninguna lista de presupuesto y que tengan más de 18 años. (usuarios que no usan la app).



The screenshot shows a SQL IDE with a query editor and a results pane. The query editor contains the following SQL code:

```
from USUARIO u2 inner join TRANSACCION t2
on u2.id_usuario = t2.id_usuario
group by u2.id_usuario) as cantidad_employed);

-- 6. Encontrar usuarios que se han registrado pero no han creado ninguna lis
select u.nombre
from USUARIO u left join LISTA_PRESUPUESTO lp
on u.id_usuario = lp.id_usuarioRealiza
where lp.id_lista is null
and now()-year(u.fecha_nacimiento)>18;

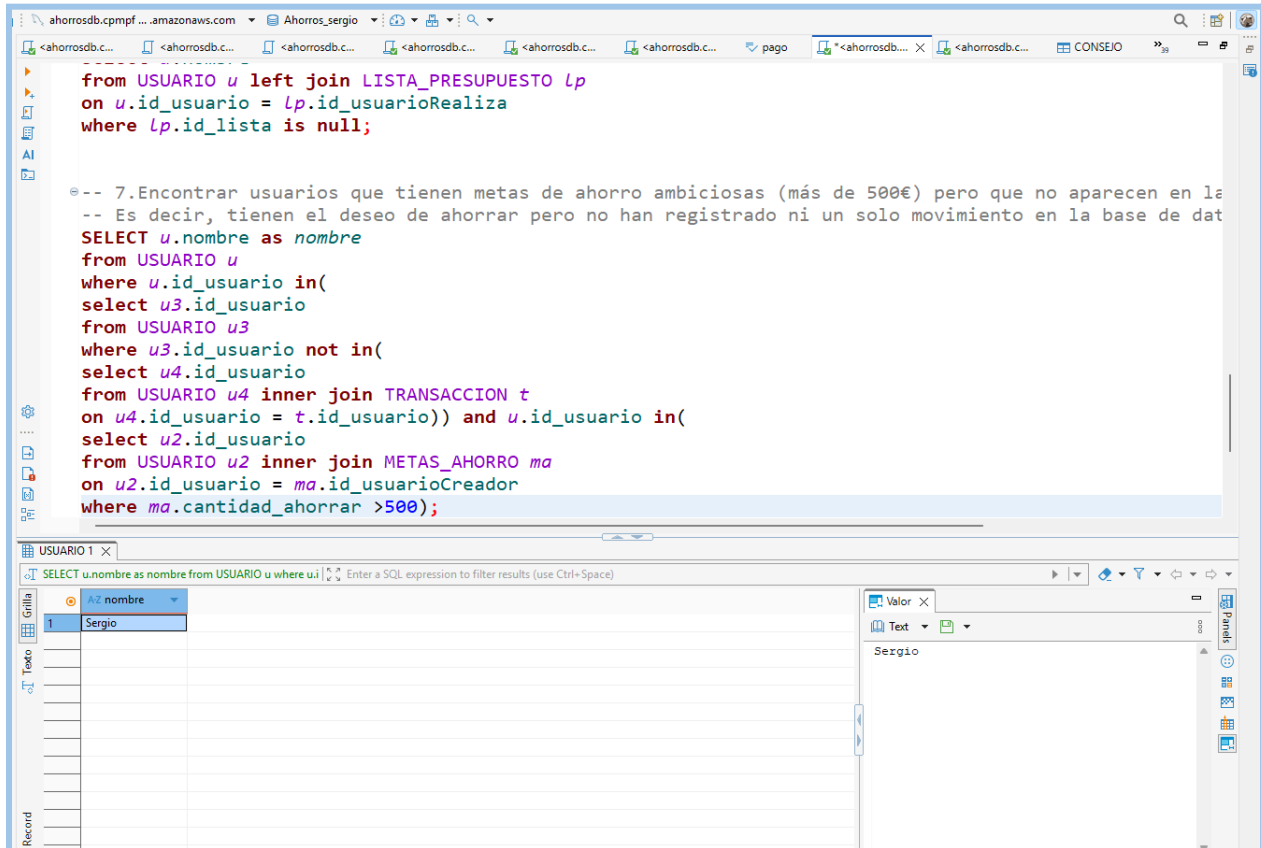
-- 7. Encontrar usuarios que tienen metas de ahorro ambiciosas (más de 500€)
```

The results pane shows the output of the query, which is a table with two columns: 'id' and 'nombre'. The table contains two rows:

id	nombre
1	Roberto
2	Sergio

The 'Valor' pane on the right shows the value 'Roberto'.

7. Encontrar usuarios que tienen metas de ahorro ambiciosas (más de 500€) pero que no aparecen en la tabla de transacciones. Es decir, tienen el deseo de ahorrar pero no han registrado ni un solo movimiento en la base de datos.



The screenshot shows a database management tool interface. The top pane displays two SQL queries. The first query is a simple join: `from USUARIO u left join LISTA_PRESUPUESTO lp on u.id_usuario = lp.id_usuarioRealiza where lp.id_lista is null;`. The second query is more complex, identifying users with high savings goals but no transactions: `-- 7. Encontrar usuarios que tienen metas de ahorro ambiciosas (más de 500€) pero que no aparecen en la -- Es decir, tienen el deseo de ahorrar pero no han registrado ni un solo movimiento en la base de datos SELECT u.nombre as nombre from USUARIO u where u.id_usuario in(select u3.id_usuario from USUARIO u3 where u3.id_usuario not in(select u4.id_usuario from USUARIO u4 inner join TRANSACCION t on u4.id_usuario = t.id_usuario)) and u.id_usuario in(select u2.id_usuario from USUARIO u2 inner join METAS_AHORRO ma on u2.id_usuario = ma.id_usuarioCreador where ma.cantidad_ahorrar >500);`. The bottom pane shows the results of the second query in a table with one row:

Grilla	Nombre
1	Sergio

. To the right, a 'Valor' pane shows the text 'Sergio'.



CONCLUSIÓN

Con este proyecto he aprendido lo importante que es darle el valor correcto a cada dato al definir las tablas; si te equivocas con el tipo, luego las consultas fallan o te truncan la información. También me ha servido para ver cómo se monta una base de datos real en AWS y cómo manejar un volumen de 1000 registros sin que se me escape nada. A futuro me gustaría mejorar la precisión a la hora de poner valor a los atributos, y me gustaría aumentar el nivel de mis consultas.



Enlace a GitHub: https://github.com/sergioruiz28282/Ahorros_ProyectoBD_SRO.git