



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.

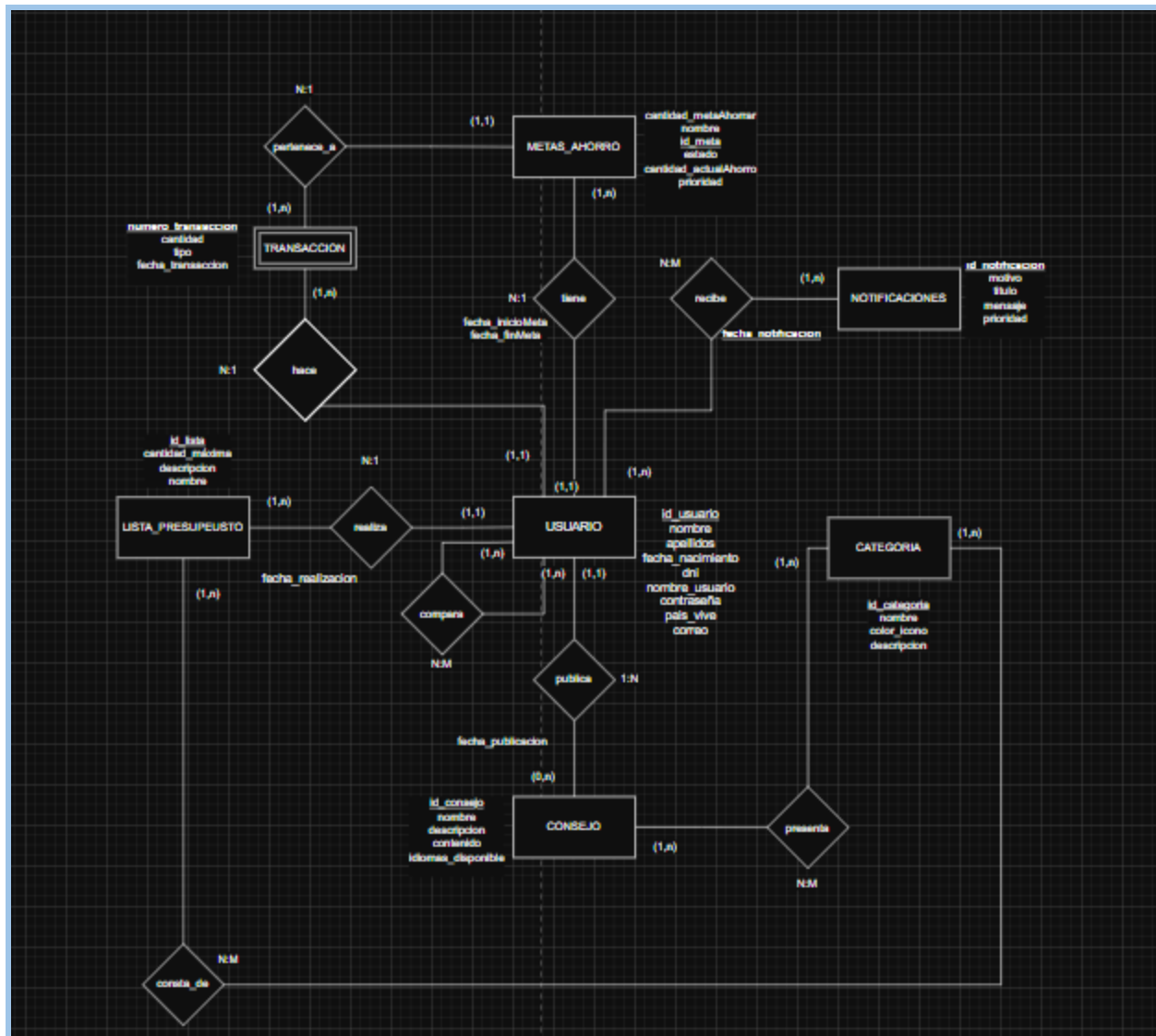
Por otro lado, los usuarios pueden publicar consejos relacionados con la gestión del dinero, con el objetivo de ayudar a otros usuarios. De cada consejo se guarda su identificador, el nombre, una descripción y el contenido, además de la fecha en la que se publica.

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:

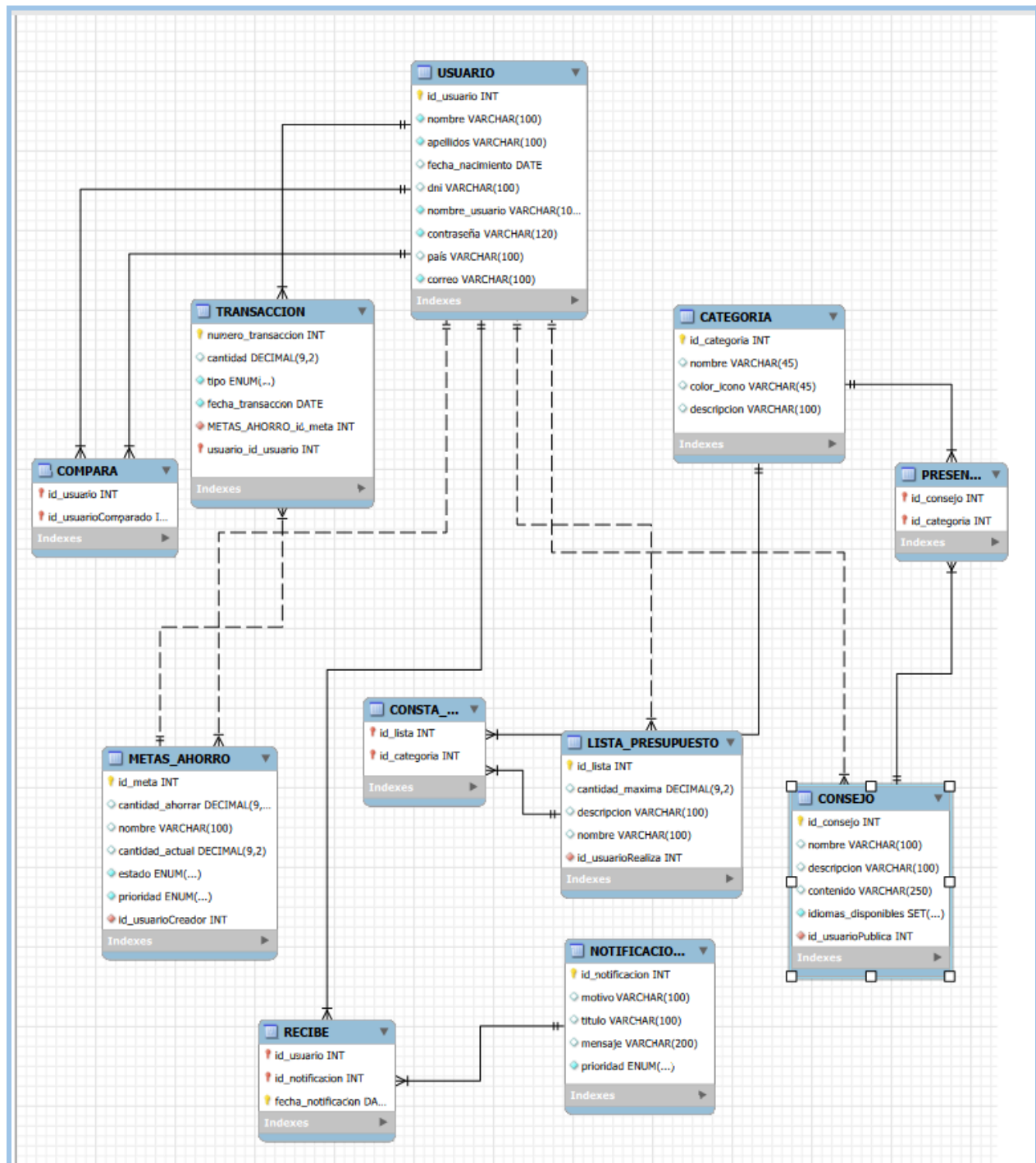
- **Usuario:** Aquí guardo los datos básicos de la persona, como su nombre, correo y país. Sin el usuario, nada de lo demás tendría sentido.
- **Metas de Ahorro:** Son los objetivos que se propone cada uno, cómo ahorrar para un viaje. Guardo cuánto quieren conseguir y qué prioridad tiene cada meta.
- **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:

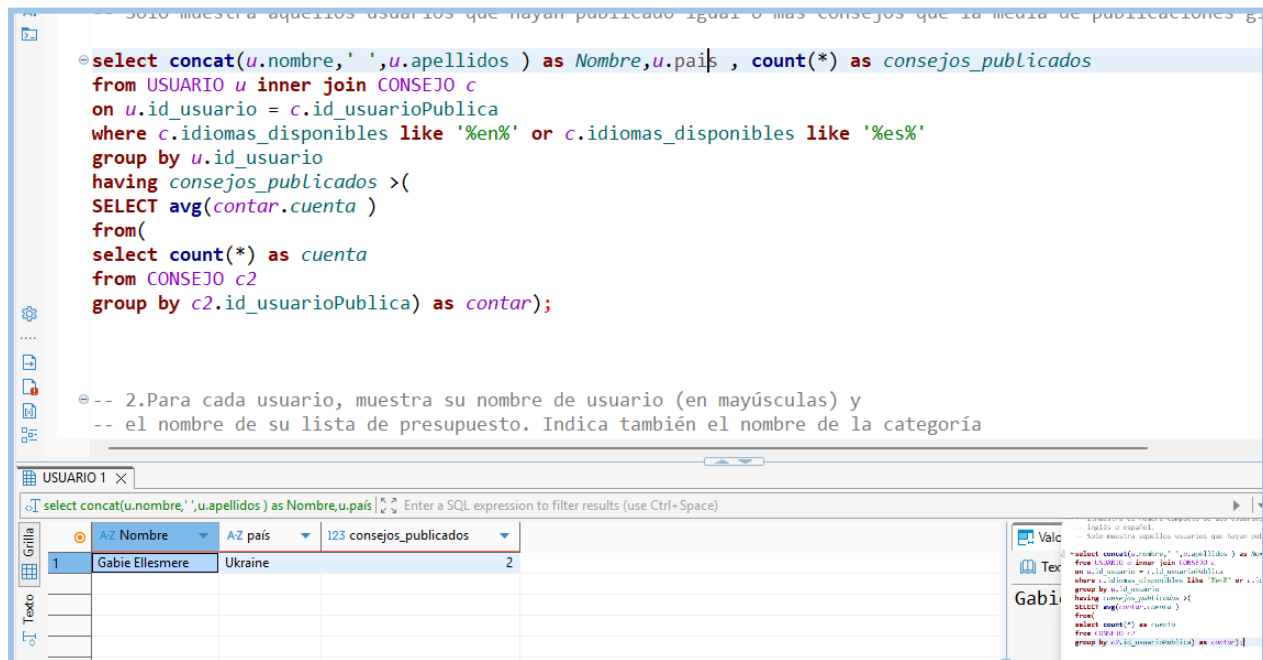
- **ENUM:** Lo he aplicado en cosas como el estado de las metas o la prioridad. Así, solo se puede elegir entre opciones fijas como activo, pendiente o inactivo lo que puede servir para evitar que alguien escriba mal una palabra y luego las consultas no funcionen.
- **SET:** Este lo he usado para los 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('ingreso', 'retirada') not null;

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 solo de aquellos usuarios que han publicado algún consejo en inglés o español y 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 grid. The query is as follows:

```
-- Solo muestra aquellos usuarios que hayan publicado igual o más consejos que la media de publicaciones g-
select concat(u.nombre, ' ', u.apellidos) as Nombre, u.país, 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);
```

Below the query, there are two comments:

```
-- 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
```

The results grid shows the following data:

	AZ Nombre	AZ país	123 consejos_publicados
1	Gabie Ellesmere	Ukraine	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 with multiple tabs. The active tab displays two SQL queries. The first query is a SELECT statement that joins the USUARIO, LISTA_PRESUPUESTO, CONSTA_DE, and CATEGORIA tables. It filters for lists where the maximum quantity is greater than the average maximum quantity of all lists. The second query is a CREATE VIEW statement that creates a view named consulta_3, which selects the user name, the last notification date, and the number of days since the last notification.

Below the queries, the results of the first query are displayed in a table. The table has four columns: ID, nombre, nombre_lista, and nombre_categoria. The results show 14 rows of data.

ID	nombre	nombre_lista	nombre_categoria
1	MSUMPTON1	Control Financiero	Transporte
2	VMULVANEY2	Presupuesto Familiar	Ocio y Salud
3	AKYNSEY3	Presupuesto Básico	Suscripciones
4	GGENTREAU4	Resumen Financiero	Ingresos
5	BSUDELLE6	Presupuesto General	Vivienda
6	LRHYMER8	Presupuesto Mensual	Transporte
7	RMCLINTON9	Control Mensual	Ocio y Salud
8	JJOSELSONB	Plan de Gastos	Ingresos
9	AHILLEYE	Gastos Personales	Alimentación
10	VFOURACREF	Presupuesto Personal	Transporte
11	EJELLG	Plan Financiero	Ocio y Salud
12	ECORTEISJ	Presupuesto Familiar	Inversión
13	CBRITNERK	Control Financiero	Vivienda
14	HHOLLINGDALEL	Presupuesto Mensual	Alimentación

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 "no ha llegado" en la columna de las fechas, y no tienen en la de los días transcurridos.

```
-- 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(), max(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 apellido con
-- 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)))
```

UUARIO 1 X

select u.nombre as nombre_usuario, ifnull(max(r.fecha_n...

AZ nombre_usuario	AZ ultima_fecha	AZ dias_diferencia
Gabie	no le ha llegado	no tiene
Milty	no le ha llegado	no tiene
Vivyan	no le ha llegado	no tiene
Ardra	no le ha llegado	no tiene
Guillaume	2021-12-30	1508
Aindrea	no le ha llegado	no tiene
Beverley	2022-01-02	1505
Donovan	no le ha llegado	no tiene
Lillie	no le ha llegado	no tiene
Rutherford	no le ha llegado	no tiene
Mag	no le ha llegado	no tiene
Janot	no le ha llegado	no tiene
K...	2021-06-11	1787

Valor X

Text

Gabie

4. Mostrar un informe detallado del progreso de las metas de ahorro de cada usuario. Para ello, se debe visualizar el apellido del usuario transformando su última letra a mayúsculas, el nombre de la meta y la cantidad objetivo a ahorrar. Además, se debe calcular el saldo actual real de la meta (restando las retiradas a los ingresos) y mostrar la cantidad pendiente para alcanzar dicho objetivo. Se deben controlar los valores nulos para que, en caso de que no existan transacciones, el saldo aparezca como cero

```
-- 4. Mostrar un informe detallado del progreso de las metas de ahorro de cada usuario.
-- Para ello, se debe visualizar el apellido del usuario transformando su última letra a mayúsculas,
-- el nombre de la meta y la cantidad objetivo a ahorrar.
-- Además, se debe calcular el saldo actual real de la meta (restando las retiradas a los ingresos) y
-- mostrar la cantidad pendiente para alcanzar dicho objetivo. Se deben controlar los valores nulos para que, en caso de que
-- no existan transacciones, el saldo aparezca como cero
CREATE VIEW consulta_4 AS;
SELECT
  CONCAT(LEFT(u.apellidos, LENGTH(u.apellidos) - 1), UPPER(RIGHT(u.apellidos, 1))) AS Apellido_Usuario,
  m.nombre AS Nombre_Meta,
  m.cantidad_ahorrar AS Objetivo_Meta,
  (
    IFNULL((SELECT SUM(t1.cantidad) FROM TRANSACCION t1 WHERE t1.METAS_AHORRO_id_meta = m.id_meta AND t1.tipo = 'ingreso'), 0) -
    IFNULL((SELECT SUM(t2.cantidad) FROM TRANSACCION t2 WHERE t2.METAS_AHORRO_id_meta = m.id_meta AND t2.tipo = 'retirada'), 0)
  ) AS Saldo_Actual_Real,
  (
    m.cantidad_ahorrar -
    (IFNULL((SELECT SUM(t3.cantidad) FROM TRANSACCION t3 WHERE t3.METAS_AHORRO_id_meta = m.id_meta AND t3.tipo = 'ingreso'), 0) -
    IFNULL((SELECT SUM(t4.cantidad) FROM TRANSACCION t4 WHERE t4.METAS_AHORRO_id_meta = m.id_meta AND t4.tipo = 'retirada'), 0))
  ) AS Pendiente_para_Objetoivo
FROM USUARIO u
INNER JOIN METAS_AHORRO m ON u.id_usuario = m.id_usuarioCreador;
```

TAS_AHORRO 1 X

ECT CONCAT(LEFT(u.apellidos, LENGTH(u.apellidos) - 1), UPPER(RIGHT(u.apellidos, 1))) Enter a SQL expression to filter results (use Ctrl+Space)

AZ Apellido_Usuario	AZ Nombre_Meta	123 Objetivo_Meta	123 Saldo_Actual_Real	123 Pendiente_para_Objetoivo
EllesmerE	Ahorro salud	37.561,44	-4.977,76	42.539,20
SumptoN	Ahorro ropa	73.966,42	-1.542,74	75.509,16
MulvaneY	Ahorro imprevistos	27.756,76	-2.301,66	30.058,42
Kynsey	Ahorro mudanza	6.171,93	-947,31	7.119,24
GentreaU	Ahorro emergencia	13.699,4	-3.006,77	16.706,17
De BeauchemP	Ahorro navidad	19.550,38	-393,57	19.943,95
Suddell	Ahorro viaje	17.280,56	-3.137,6	20.418,16
MinoR	Ahorro ahorro_familiar	13.305,09	3.227,02	10.078,07
RhymeR	Ahorro vacaciones	23.942,89	1.266,99	22.675,90
McClintoN	Ahorro mascota	1.135,9	-2.473,95	3.609,85
BirkwooD	Ahorro regalos	43.042,62	-2.501,02	45.543,64
JoselsoN	Ahorro viaje	94.791,03	-1.749,05	96.540,08
Melly	Ahorro vacaciones	97.862,39	-154,27	98.016,66
MoscoN	Ahorro navidad	52.037,02	-1.868,85	54.905,87

Valor X

Text

EllesmerE

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

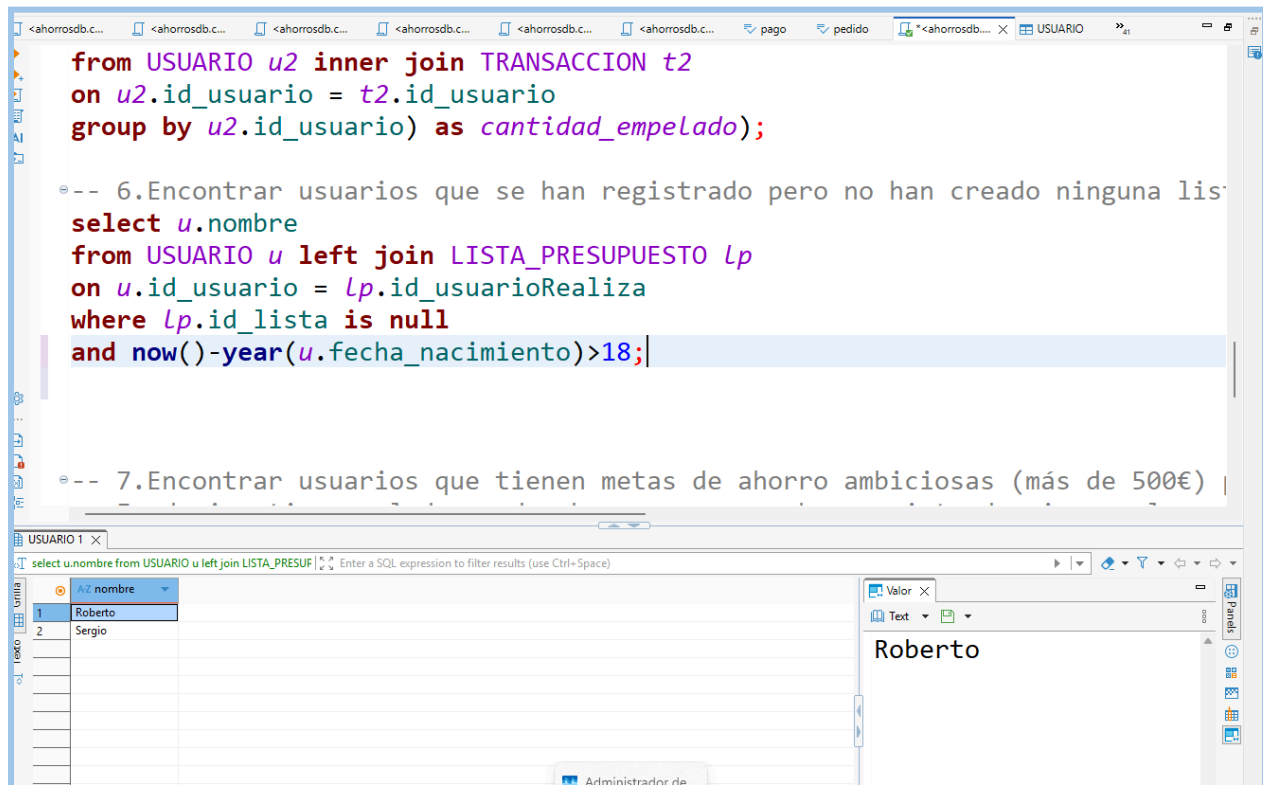
The screenshot shows a MySQL IDE window with a SQL query and its results. The query is designed to find users whose total transaction amount exceeds the average transaction amount of all users.

```
-- 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)
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, the results are displayed in a table with columns 'nombre' and 'suma'.

	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	Darleen	887,19
11	Karlotta	936,49
12	Quintus	566,09
13	Fillide	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 database management tool interface. The top pane displays SQL queries. The second query is highlighted, which corresponds to the task: finding users who have registered but have not created any budget list and are older than 18.

```
-- 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;
```

The bottom pane shows the results of the query in a table:

	nombre
1	Roberto
2	Sergio

On the right, a 'Valor' pane displays the name '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