APLICACIÓN DE AHORROS

Sergio Ruiz Ortiz

Fecha:15/02/2026

Curso:1ºDAW

ÍNDICE

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.

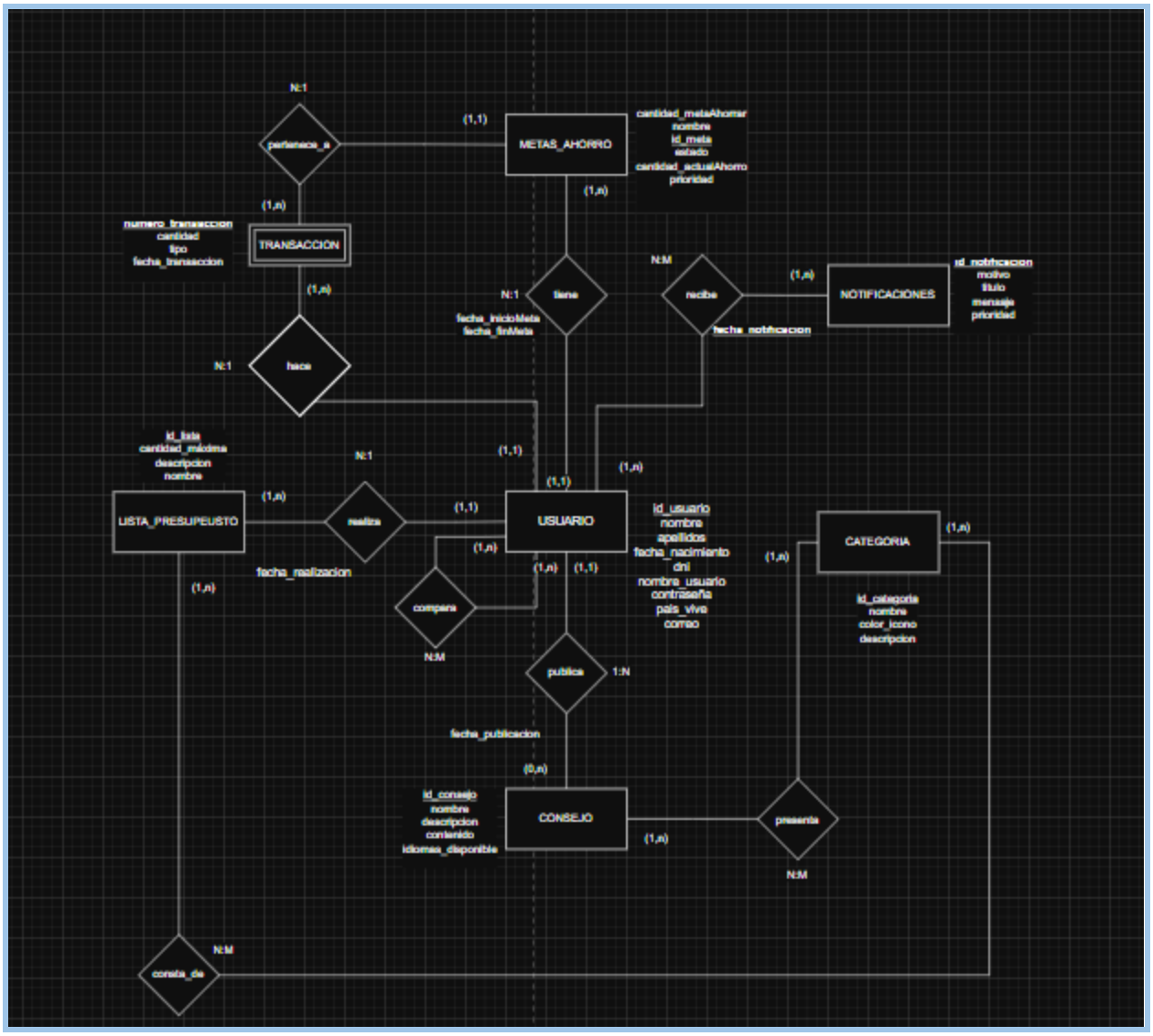
Por otro lado, los usuarios pueden publicar consejos relacionados con la gestión del dinero, con el objetivo de ayudar a otros usuarios. De cada consejo se guarda su identificador, el nombre, una descripción y el contenido, además de la fecha en la que se publica.

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:

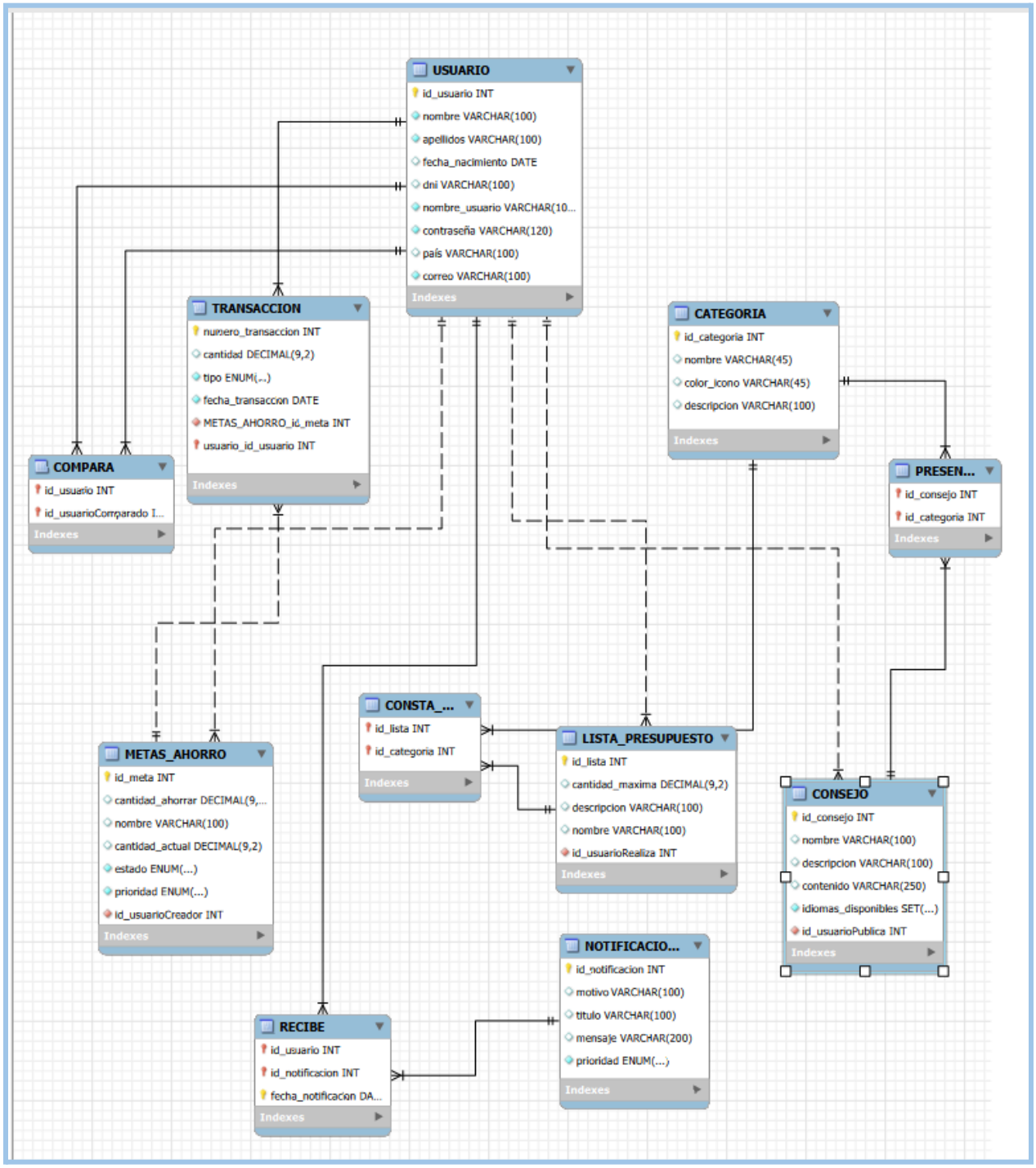
- **Usuario**: Aquí guardo los datos básicos de la persona, como su nombre, correo y país. Sin el usuario, nada de lo demás tendría sentido.
- **Metas de Ahorro**: Son los objetivos que se propone cada uno, cómo ahorrar para un viaje. Guardo cuánto quieren conseguir y qué prioridad tiene cada meta.
-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


N:1
pertenece_a
(1,1)
METAS_AHORRO
cantidad_metaAhorrar
nombre
id_meta
estado
cantidad_actualAhorro
prioridad
(1,n)
(1,n)
numero_transaccion
cantidad
tipo
fecha_transaccion
TRANSACCION
(1,n)
N:1
hace
N:M
tiene
N:1
fecha_inicioMeta
fecha_finMeta
recibe
(1,n)
NOTIFICACIONES
id_notificacion
motivo
titulo
mensaje
prioridad
fecha_notificacion
id_lista
cantidad_maxima
descripcion
nombre
N:1
(1,1)
(1,n)
(1,n)
LISTA_PRESUPUESTO
realiza
(1,1)
USUARIO
id_usuario
nombre
apellidos
fecha_nacimiento
dni
nombre_usuario
contraseña
pais_vive
correo
(1,n)
fecha_realizacion
(1,n)
(1,n)
(1,1)
(1,n)
CATEGORIA
(1,n)
id_categoria
nombre
color_icono
descripcion
compra
N:M
publica
1:N
fecha_publicacion
(0,n)
id_consejo
nombre
descripcion
contenido
idiomas_disponible
CONSEJO
(1,n)
presenta
N:M
N:M
consta_de

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.


USUARIO
id_usuario INT
nombre VARCHAR(100)
apellidos VARCHAR(100)
fecha_nacimiento DATE
dni VARCHAR(100)
nombre_usuario VARCHAR(10...
contraseña VARCHAR(120)
país VARCHAR(100)
correo VARCHAR(100)
Indexes
TRANSACCION
numero_transaccion INT
cantidad DECIMAL(9,2)
tipo ENUM(...)
fecha_transaccion DATE
METAS_AHORRO_id_meta INT
usuario_id_usuario INT
Indexes
CATEGORIA
id_categoria INT
nombre VARCHAR(45)
color_icono VARCHAR(45)
descripcion VARCHAR(100)
Indexes
COMPARA
id_usuario INT
id_usuarioComparado I...
Indexes
PRESEN...
id_consejo INT
id_categoria INT
Indexes
CONSTA_...
id_lista INT
id_categoria INT
Indexes
LISTA_PRESUPUESTO
id_lista INT
cantidad_maxima DECIMAL(9,2)
descripcion VARCHAR(100)
nombre VARCHAR(100)
id_usuarioRealiza INT
Indexes
METAS_AHORRO
id_meta INT
cantidad_ahorrar DECIMAL(9,...
nombre VARCHAR(100)
cantidad_actual DECIMAL(9,2)
estado ENUM(...)
prioridad ENUM(...)
id_usuarioCreador INT
Indexes
CONSEJO
id_consejo INT
nombre VARCHAR(100)
descripcion VARCHAR(100)
contenido VARCHAR(250)
idiomas_disponibles SET(...)
id_usuarioPublica INT
Indexes
NOTIFICACIO...
id_notificacion INT
motivo VARCHAR(100)
titulo VARCHAR(100)
mensaje VARCHAR(200)
prioridad ENUM(...)
Indexes
RECIBE
id_usuario INT
id_notificacion INT
fecha_notificacion DA...
Indexes

CAMBIOS CON SET Y ENUM

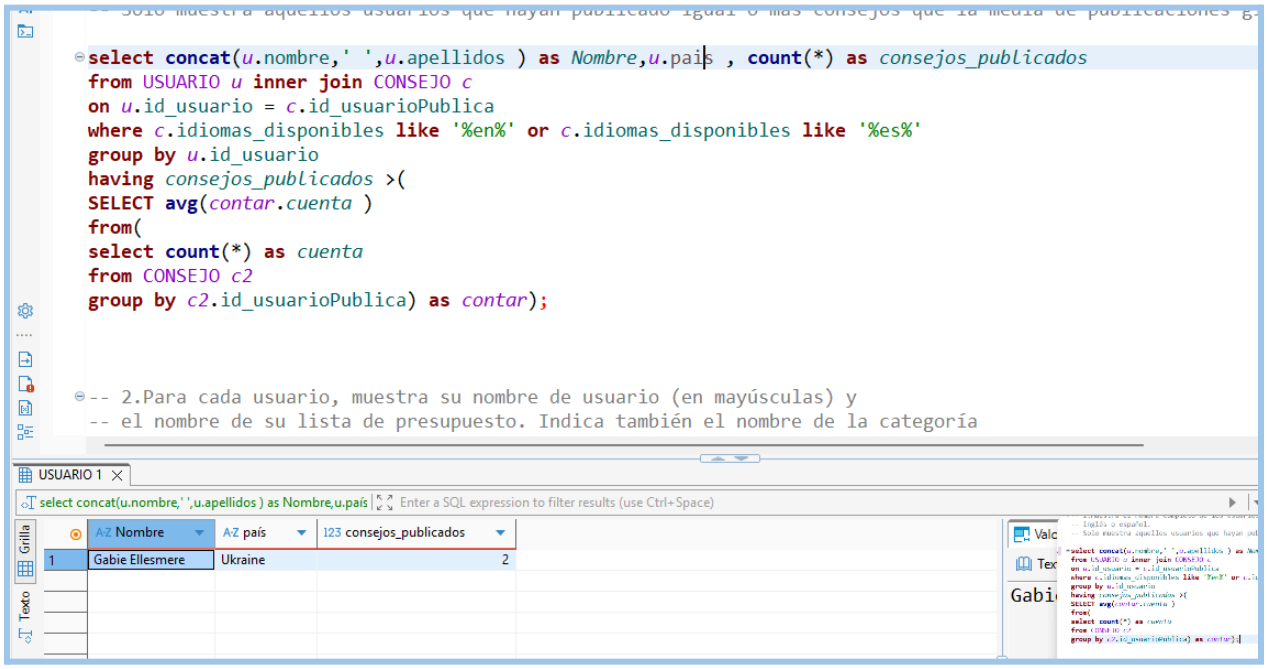
En esta parte del proyecto, decidí hacer unos ajustes en las tablas para que la base de datos fuera más profesional y no permitiera errores al meter los datos. En lugar de dejar que se pudiera escribir cualquier cosa, usé los tipos ENUM y SET para limitar las opciones:

- **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;
```

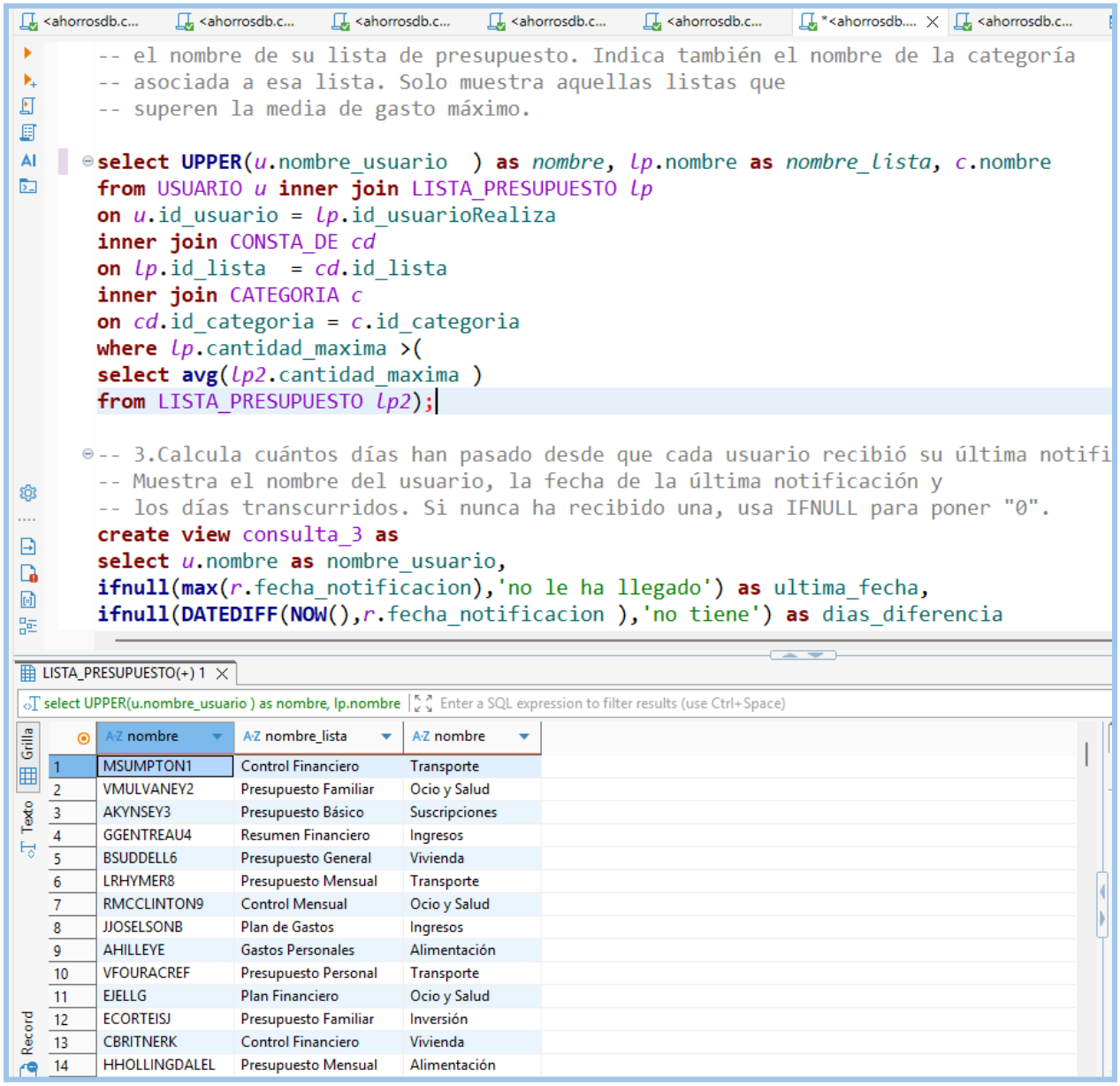
CONSULTAS SQL

1.Muestra el nombre completo de los usuarios, su país y cuántos consejos han publicado siempre que estos consejos están en inglés o español. Solo mostraremos aquellos usuarios que hayan publicado igual o más consejos que la media de publicaciones global.



```sql
select concat(u.nombre,' ',u.apellidos ) as Nombre,u.pais , 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);



-- 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
```

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.



```
-- 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_usuario  ) as nombre, lp.nombre as nombre_lista, c.nombre
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);

-- 3.Calcula cuántos días han pasado desde que cada usuario recibió su última notifi
-- 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
```

LISTA_PRESUPUESTO(+) 1

```
select UPPER(u.nombre_usuario ) as nombre, lp.nombre
```

Enter a SQL expression to filter results (use Ctrl+Space)

	nombre	nombre_lista	nombre
1	MSUMPTON1	Control Financiero	Transporte
2	VMULVANEY2	Presupuesto Familiar	Ocio y Salud
3	AKYNSEY3	Presupuesto Básico	Suscripciones
4	GGENTREAU4	Resumen Financiero	Ingresos
5	BSUDDELL6	Presupuesto General	Vivienda
6	LRHYMER8	Presupuesto Mensual	Transporte
7	RMCCLINTON9	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.

```sql
-- 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 co
-- 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)))
```

UARIO 1

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

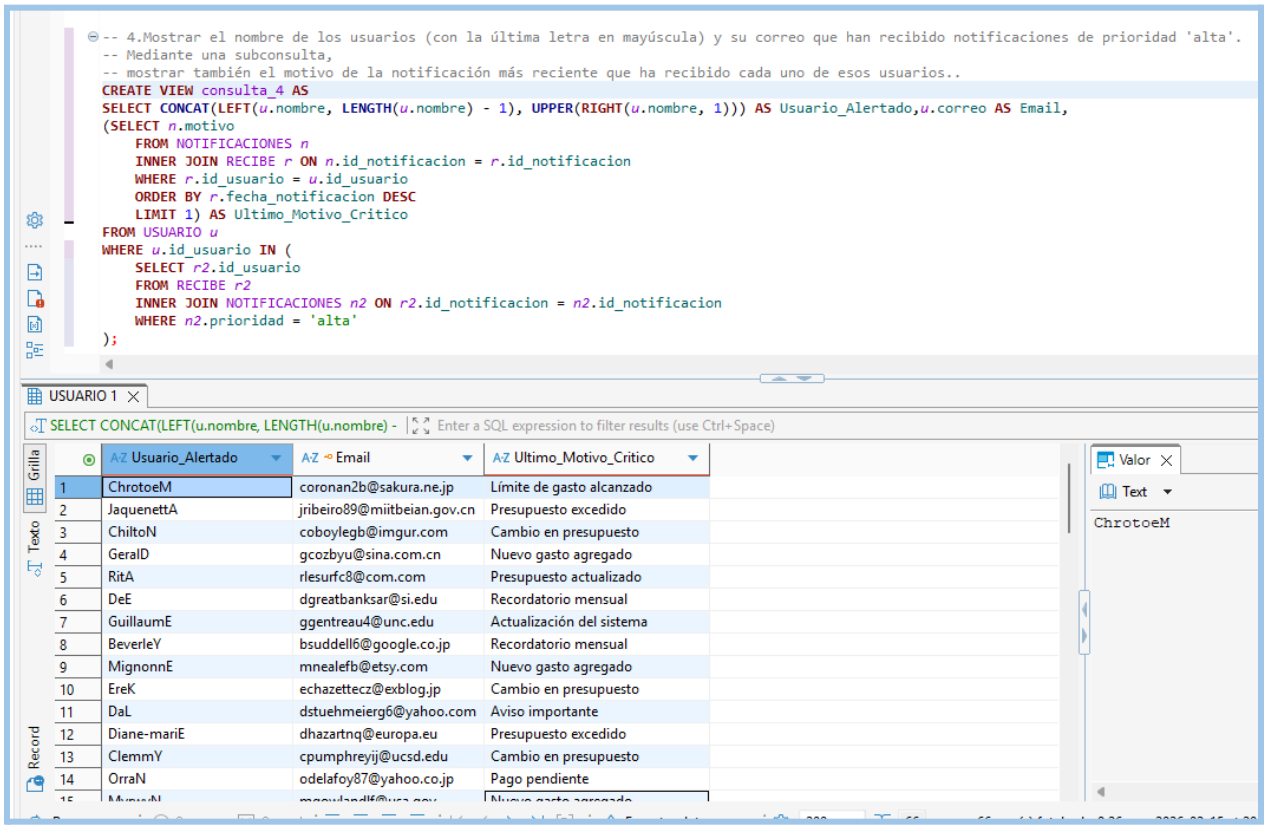
	nombre_usuario	ultima_fecha	dias_diferencia
	Gabie	no le ha llegado	no tiene
	Milty	no le ha llegado	no tiene
	Vivyan	no le ha llegado	no tiene
	Ardra	no le ha llegado	no tiene
	Guillaume	2021-12-30	1508
	Aindrea	no le ha llegado	no tiene
	Beverley	2022-01-02	1505
	Donovan	no le ha llegado	no tiene
	Lilllie	no le ha llegado	no tiene
0	Rutherford	no le ha llegado	no tiene
1	Mag	no le ha llegado	no tiene
2	Janot	no le ha llegado	no tiene

Valor

Text

Gabie

4 Mostrar el nombre de los usuarios (con la última letra en mayúscula) y su correo que han recibido notificaciones de prioridad 'alta'. Mediante una subconsulta, mostrar también el motivo de la notificación más reciente que ha recibido cada uno de esos usuarios.



```sql
-- 4.Mostrar el nombre de los usuarios (con la última letra en mayúscula) y su correo que han recibido notificaciones de prioridad 'alta'.
-- Mediante una subconsulta,
-- mostrar también el motivo de la notificación más reciente que ha recibido cada uno de esos usuarios..
CREATE VIEW consulta_4 AS
SELECT CONCAT(LEFT(u.nombre, LENGTH(u.nombre) - 1), UPPER(RIGHT(u.nombre, 1))) AS Usuario_Alertado,u.correo AS Email,
(SELECT n.motivo
    FROM NOTIFICACIONES n
    INNER JOIN RECIBE r ON n.id_notificacion = r.id_notificacion
    WHERE r.id_usuario = u.id_usuario
    ORDER BY r.fecha_notificacion DESC
    LIMIT 1) AS Ultimo_Motivo_Critico
FROM USUARIO u
WHERE u.id_usuario IN (
    SELECT r2.id_usuario
    FROM RECIBE r2
    INNER JOIN NOTIFICACIONES n2 ON r2.id_notificacion = n2.id_notificacion
    WHERE n2.prioridad = 'alta'
);
```

USUARIO 1

SELECT CONCAT(LEFT(u.nombre, LENGTH(u.nombre) - | Enter a SQL expression to filter results (use Ctrl+Space)

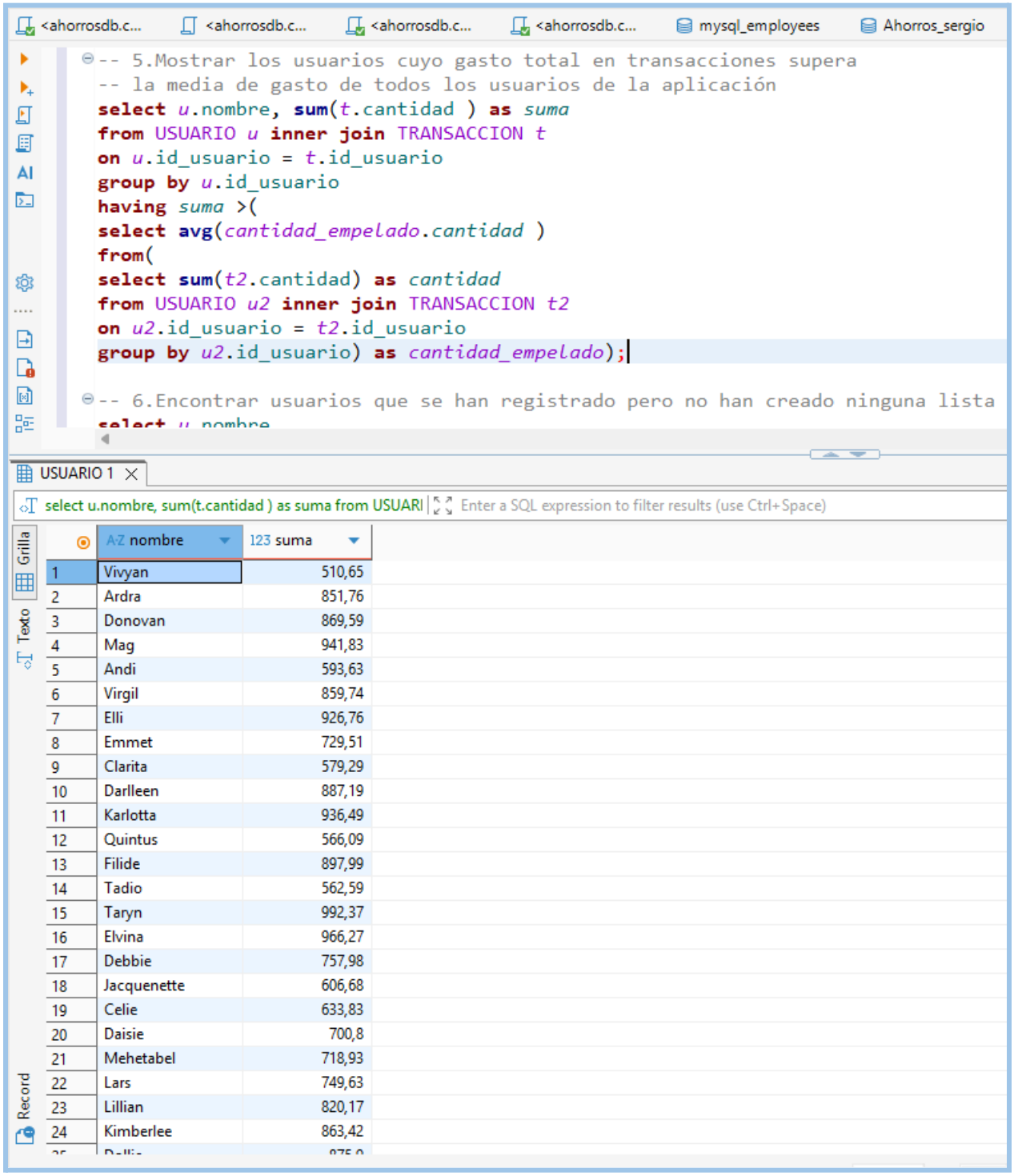
	Usuario_Alertado	Email	Ultimo_Motivo_Critico
1	ChrotoeM	coronan2b@sakura.ne.jp	Límite de gasto alcanzado
2	JaquenettA	jribeiro89@miitbeian.gov.cn	Presupuesto excedido
3	ChiltoN	coboylegb@imgur.com	Cambio en presupuesto
4	GeralD	gcozbyu@sina.com.cn	Nuevo gasto agregado
5	RitA	rlesurfc8@com.com	Presupuesto actualizado
6	DeE	dgreatbanksar@si.edu	Recordatorio mensual
7	GuillaumE	ggentreau4@unc.edu	Actualización del sistema
8	BeverleY	bsuddell6@google.co.jp	Recordatorio mensual
9	MignonnE	mnealefb@etsy.com	Nuevo gasto agregado
10	EreK	echazettecz@exblog.jp	Cambio en presupuesto
11	DaL	dstuehmeierg6@yahoo.com	Aviso importante
12	Diane-mariE	dhazartnq@europa.eu	Presupuesto excedido
13	ClemmY	cpumphreyij@ucsd.edu	Cambio en presupuesto
14	OrraN	odelafoy87@yahoo.co.jp	Pago pendiente

Valor

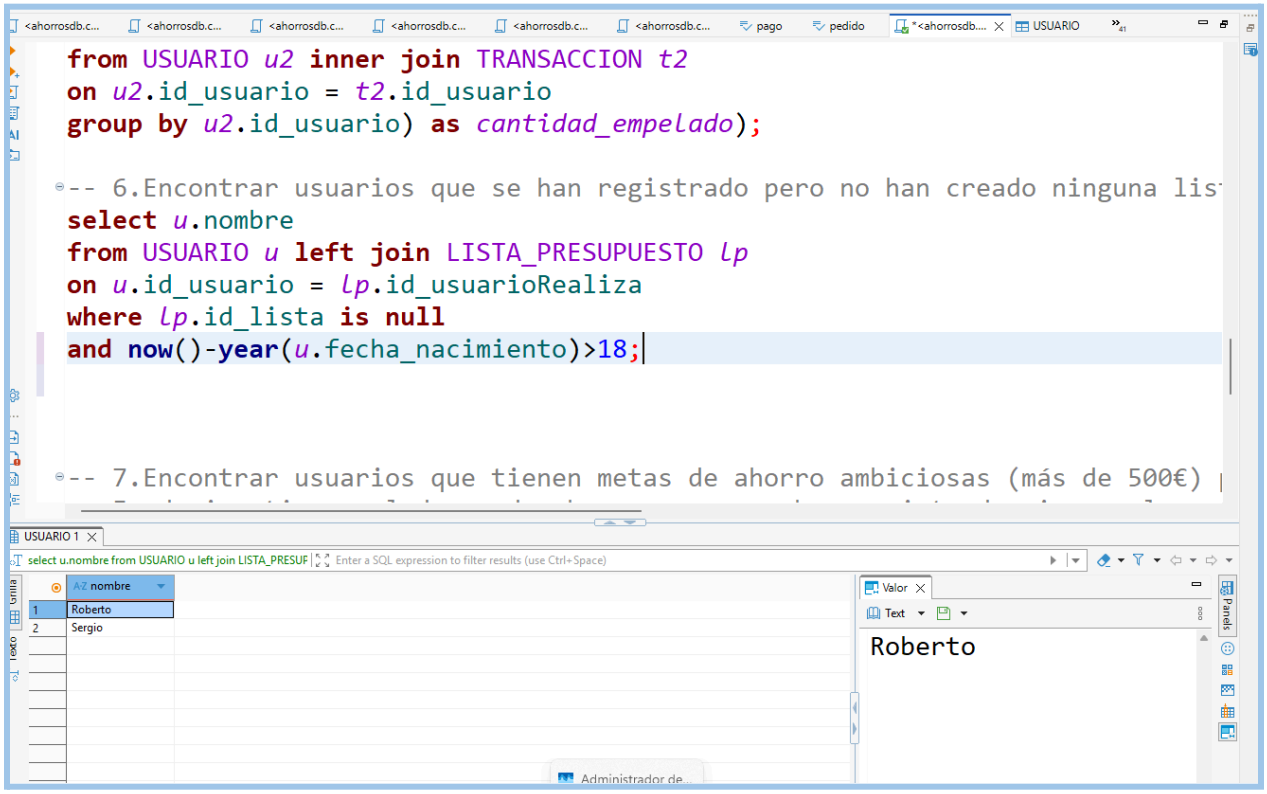
Text

ChrotoeM

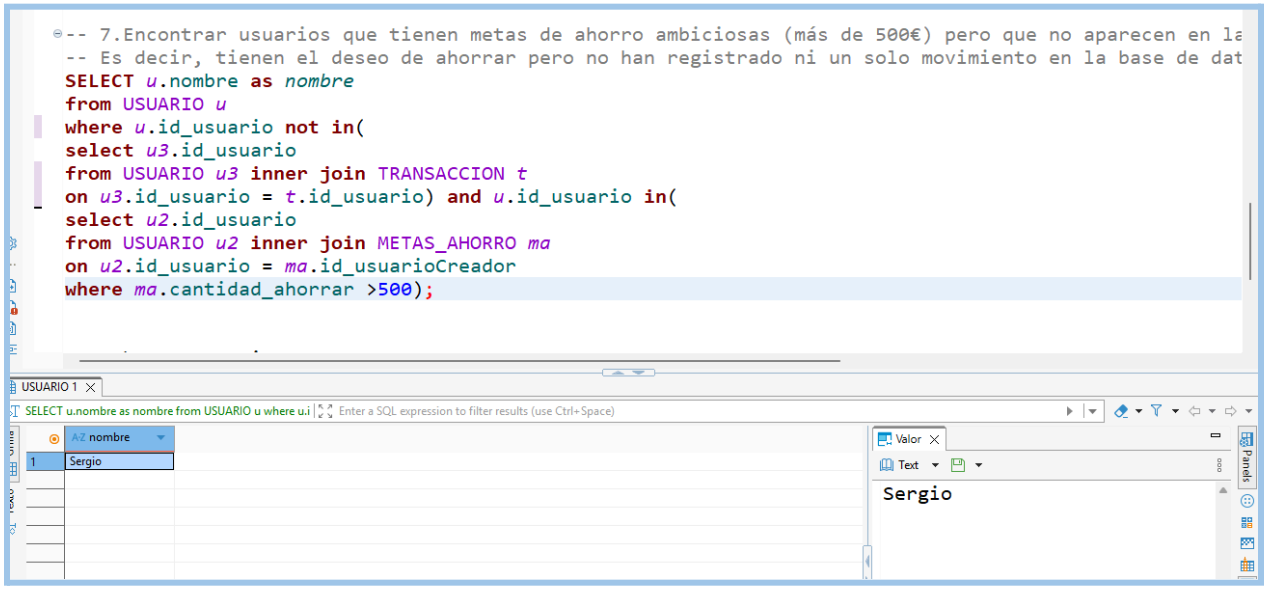
5.Mostrar los usuarios cuyo gasto total en transacciones supera la media de gasto de todos los usuarios de la aplicación



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).



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.



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