

# Mercado de Valores

## **Grupo 4C**

Rodríguez Fernández, Carlos

Rozas Rodríguez, Carlos

Souto Soneira, Pablo

Ubeira Carramal, Pablo

Vidal Lorenzo, Adrián

Vilela Pérez, Nicolás



# Índice

<b>1.</b>	<b>Introducción .....</b>	<b>6</b>
1.1	Alcance del sistema .....	6
1.2	Tipo de usuarios .....	6
1.3	Funcionalidades básicas .....	6
1.3.1	Funcionalidades de usuario .....	6
1.3.2	Funcionalidades de comercio.....	7
1.4	Funcionalidades extra .....	8
<b>2</b>	<b>Modelo de Datos .....</b>	<b>10</b>
2.1	Modelo Entidad-Relación.....	10
2.2	Modelo Relacional.....	10
<b>3</b>	<b>Script de Creación de la Base de Datos.....</b>	<b>12</b>
3.1	Tabla usuario .....	12
3.2	Tabla empresa .....	12
3.3	Tabla inversor .....	12
3.4	Tabla regulador .....	12
3.5	Tabla prerregistro .....	13
3.6	Tabla datos .....	13
3.7	Tabla baja .....	13
3.8	Tabla mercado .....	13
3.9	Tabla tenerParticipaciones .....	13
3.10	Tabla informacionBeneficios .....	14
3.11	Tabla stacking .....	14
3.12	Tabla ofertaPacto .....	14
3.13	Tabla acuerdoPacto.....	15
3.14	Comentarios .....	15
<b>4</b>	<b>Interfaz.....</b>	<b>16</b>
4.1	Ventana de autenticación de usuarios .....	16
4.2	Ventana de solicitud de registro .....	16
4.3	Ventana del regulador .....	16
4.4	Ventana principal de empresa e inversor.....	19
4.5	Ventana de mercado .....	21
4.6	Ventana de gestión de pagos.....	22
4.7	Ventana de gestión de contratos de futuros.....	22
4.8	Ventana de gestión de stacking.....	23

4.9	Ventana de avisos/excepciones.....	24
5	Desarrollo de Funcionalidades.....	26
5.1	[FU_01] Acceso Sistema .....	26
5.2	[FU-02] Solicitud Registro .....	30
5.3	[FU_03] Confirmación Registro.....	33
5.4	[FU_04] Modificación Datos .....	35
5.5	[FU_05] Modificación Saldo Cuenta .....	36
5.6	[FU_06] Solicitud Baja .....	37
5.7	[FU_07] Confirmación Baja .....	38
5.8	[FC_01] Alta Participaciones .....	39
5.9	[FC_02] Baja Participaciones.....	40
5.10	[FC_03] Comisión Compra/Venta .....	41
5.11	[FC_04] Venta Participaciones .....	42
5.12	[FC_05] Baja Venta Participaciones .....	43
5.13	[FC_06] Compra Participaciones .....	43
5.14	[FC-07] Alta Pago Beneficios .....	46
5.15	[FC_08] Baja Pago Beneficios.....	49
5.16	[FC_09] Pago Beneficios .....	49
5.17	[FC_10] Stacking .....	54
5.18	[FC_11] Contratos de Futuros .....	57
	Apéndice A. Manual de Funcionamiento de la Aplicación .....	60





# 1. Introducción

## 1.1 Alcance del sistema

El objetivo de este trabajo es el de permitir que diferentes empresas puedan ofrecer participaciones sobre sus beneficios comerciales futuros. También debe permitir que diferentes inversores puedan adquirir las participaciones de las empresas participantes (las empresas también pueden adquirir participaciones de otras empresas). El mercado de valores contará con un regulador que autorizará, verificando diversas condiciones, la participación en el mercado de valores de empresas e inversores.

En este mercado de valores se podrán negociar (compra y venta) las participaciones de las empresas registradas. Además, cada empresa podrá pagar beneficios comerciales, previamente anunciados, a sus inversores.

## 1.2 Tipo de usuarios

\* **Regulador:** Usuario encargado de verificar el funcionamiento correcto del mercado de valores, especialmente en lo referido a la gestión de las cuentas de las empresas e inversores (modificaciones manuales de los saldos de las cuentas) y establecimiento de la comisión de compra/venta. No es necesario ningún dato de identificación personal ya que se trata de una persona jurídica. Accederá al sistema mediante identificación usuario/password.

\* **Empresas:** Organizaciones que desean ofrecer participaciones sobre sus beneficios comerciales futuros. Se necesitan los siguientes datos de identificación: CIF, nombre comercial, dirección y teléfono de contacto. Podrá realizar acciones de compra/venta sobre las participaciones de las empresas autorizadas en el sistema (propias o ajenas). Accederá al sistema mediante identificación usuario/password.

\* **Inversores:** Personas físicas que desean invertir mediante la negociación con participaciones de empresas. Se necesitan los siguientes datos de identificación: DNI, nombre y apellidos, dirección y teléfono de contacto. Podrá realizar acciones de compra/venta sobre las participaciones de las empresas autorizadas en el sistema. Accederá al sistema mediante identificación usuario/password.

## 1.3 Funcionalidades básicas

### 1.3.1 Funcionalidades de usuario

\* **[FU\_01] Acceso Sistema:** El sistema solicitará a los usuarios su identificador y su clave de acceso. Si la información suministrada es correcta se accederá a la información de usuario y este podrá realizar las operaciones asociadas a su tipo de usuario. En caso contrario, se informará de un error en las credenciales.

\* **[FU\_02] Solicitud Registro:** Cualquier empresa/inversor podrá solicitar el alta en el mercado de valores mediante el suministro de sus datos de identificación. En este proceso de solicitud de registro, el usuario propondrá un identificador y una clave de acceso. En el caso de que el identificador ya exista en el sistema se informará al usuario y se volverá a solicitar. Si todos los datos son correctos se registrará la solicitud. Quedará pendiente de confirmar por parte del regulador.

\* **[FU\_03] Confirmación Registro:** Una solicitud de registro se convertirá en un registro efectivo cuando el regulador lo autorice. Una vez confirmado el registro, el usuario podrá acceder a una cuenta dónde se almacenarán sus fondos disponibles y a una cartera de participaciones dónde se almacenarán las participaciones que posee de cada empresa.

\* **[FU\_04] Modificación Datos:** Cualquier empresa/inversor podrá modificar sus datos de identificación en cualquier momento. También podrá modificar su identificador (siempre que siga siendo único en el sistema) y su clave de acceso. En el caso de que el identificador propuesto ya exista en el sistema se informará de un error en la modificación de datos.

\* **[FU\_05] Modificación Saldo Cuenta:** Las modificaciones en el saldo de las cuentas de los usuarios, consecuencia de transferencias económicas recibidas y/o emitidas por la administración del mercado de valores, serán realizadas por el regulador tras la comprobación de la correspondiente transferencia (en la base de datos no se almacenará ninguna información sobre dichas transferencias). Los usuarios no podrán realizar modificaciones manuales en los saldos de sus cuentas (las modificaciones serán consecuencia de la realización de operaciones).

\* **[FU\_06] Solicitud Baja:** Cualquier empresa/inversor podrá solicitar su baja en el sistema en cualquier momento. Quedará pendiente de confirmar por parte del regulador.

\* **[FU\_07] Confirmación Baja:** La baja de un usuario debe ser confirmada por el regulador, una vez verifique que el usuario no posee participaciones y que el saldo de su cuenta está a cero. En el caso de que el usuario tenga participaciones se rechazará la baja. En el caso de que el saldo no sea cero se pondrá a cero (suponiendo la realización de la correspondiente transferencia) y se tramitará la baja.

### 1.3.2 Funcionalidades de comercio

\* **[FC\_01] Alta Participaciones:** Las empresas registradas podrán dar de alta el número de participaciones que deseen. Las participaciones dadas de alta por una empresa se incorporarán a su propia cartera de participaciones. Por simplicidad del sistema, las participaciones no se identificarán de forma unitaria. Sólo se registrará la cantidad total de participaciones, de cada empresa, que posee cada empresa/inversor y el número total de participaciones involucradas en cada operación de compra/venta.

\* **[FC\_02] Baja Participaciones:** Las empresas registradas podrán dar de baja el número de participaciones que deseen, siempre y cuando dispongan de ellas en su propia cartera. Las participaciones dadas de baja por una empresa se eliminarán del sistema disminuyendo el número total de participaciones existentes.

\* **[FC\_03] Comisión Compra/Venta:** El regulador del mercado fijará una comisión sobre las operaciones de compra/venta. El importe de la comisión de compra/venta deberá ser público y se informará al vendedor en el momento de realizar la oferta de venta. El importe de dicha comisión, a deducir del ingreso al vendedor, será ingresado en la cuenta especial del regulador. La comisión a aplicar a una operación de compra/venta será la disponible de forma pública en el sistema en el momento de poner en el mercado la oferta de venta.

\* **[FC\_04] Venta Participaciones:** Cualquier usuario, inversor y/o empresa, poseedor de participaciones de alguna empresa podrá ponerlas a la venta. El total de participaciones puestas a la venta por un usuario no puede ser superior al número de participaciones que posee dicho



usuario. Para ello deberá indicar la empresa, el número global de participaciones y el precio de venta. Cuando las participaciones se vendan, recibirá en su cuenta el importe de la venta menos la comisión fijada por el mercado de valores. Las ofertas de venta permanecerán en el mercado de forma permanente salvo que sean retiradas por el usuario que las realizó.

\* **[FC\_05] Baja Venta Participaciones:** El usuario que ha realizado una oferta de venta podrá eliminarla en cualquier momento. En el caso de ejecuciones parciales de la oferta de venta, la baja afectará a la parte de la oferta no ejecutada.

\* **[FC\_06] Compra Participaciones:** Cualquier usuario podrá realizar ofertas de compra, siempre y cuando disponga de saldo suficiente en su cuenta. Para ello deberá indicar la empresa, el número global de participaciones y el precio máximo de compra. La compra se realizará de forma completa si hay suficientes participaciones a la venta al precio indicado o inferior (una compra puede completarse a partir de varias ofertas de venta; en ese caso, se seleccionarán las ofertas de venta por orden de precio y de antigüedad) o de forma parcial si hay menos participaciones a la venta, al precio fijado o inferior, de las indicadas en la orden de compra. Las órdenes de compra no quedarán en el mercado una vez cruzadas con las ofertas de venta existentes, tanto se hayan realizado de forma parcial o, incluso, no se hayan podido realizar por no existir oferta adecuada.

\* **[FC\_07] Alta Pago Beneficios:** Las empresas registradas podrán publicar información sobre sus pagos futuros de beneficios asociados a las participaciones que están en negociación en el mercado. Para ello deberá indicar la fecha de pago y el beneficio a abonar por cada participación (puede ser saldo en efectivo, participaciones o una combinación de ambas). Para completar el alta deberá tener en su cuenta el importe suficiente (en saldo en efectivo y/o en participaciones) para hacer frente al pago de beneficios a publicar. El importe de dicho pago se bloqueará (no estará disponible para operaciones) hasta la eliminación del anuncio de pago o hasta la realización del pago.

\* **[FC\_08] Baja Pago Beneficios:** La eliminación de información sobre pagos de beneficios sólo podrá ser realizada por el regulador del mercado de valores a petición de la empresa que lo ha realizado (en la base de datos no se almacenará ninguna información sobre dichas peticiones de baja).

\* **[FC\_09] Pago Beneficios:** Las empresas podrán realizar pagos de beneficios asociados a participaciones en cualquier momento, con anuncio previo o sin anuncio previo. En el caso de anuncio previo, en el momento de realizar el pago se eliminará el anuncio asociado y se realizará el pago con los fondos (saldo y/o participaciones) previamente bloqueados. En el caso de que no exista anuncio previo, el pago se realizará directamente, si la empresa dispone de fondos suficientes en su cuenta.

## 1.4 Funcionalidades extra

\* **[FC\_10] Stacking:** Un usuario puede depositar parte de sus acciones durante un periodo mínimo de tiempo generando intereses en base a su valor. El porcentaje de interés es estipulado por el regulador. Cada mes el usuario tiene la posibilidad durante un día de retirar esas acciones, en caso de que no lo haga seguirán generando durante otro mes. Puede activarse una opción para que se retiren automáticamente pasando ese tiempo.

\* **[FC\_11] Contratos de Futuros:** La posibilidad de la utilización de contratos de futuros. Un contrato de futuros es un pacto según el cual en un momento dado se acuerda el precio de una compra o venta que se va a realizar dentro de un periodo de tiempo X. Este tipo de contratos se utilizan para favorecer la especulación de manera que se aprovecha la diferencia entre el precio pactado y el real en el momento X para generar beneficios o pérdidas.



**Regulador** (usuario, comision, interes)

PK: usuario

FK: usuario referencia Usuario.id

**Prerregistro** (marcaTemporal, id, clave, tipoUsuario)

PK: marcaTemporal

**Datos** (marcaTemporalPrerregistro, dato)

PK: (marcaTemporalPrerregistro, dato)

FK: marcaTemporalPrerregistro referencia Prerregistro.marcaTemporal

**Baja** (usuario, marcaTemporal)

PK: (usuario, marcaTemporal)

FK: usuario referencia a Usuario.id

**Mercado** (usuario, empresa, marcaTemporal, comisionVenta, cantidadParticipaciones, precioParticipacion)

PK: (usuario, marcaTemporal)

FK: usuario referencia a Usuario.id

FK: empresa referencia a Empresa.id

**TenerParticipaciones** (usuario, cantidad, empresa)

PK: (usuario, empresa)

FK: usuario referencia a Usuario.id

FK: empresa referencia a Empresa.id

**InformacionBeneficios** (empresa, fechaPago, importePorParticipacion, cantidadParticipaciones, participacionesPorUsuario, usuariosConParticipaciones)

PK: (empresa, fechaPago)

FK: empresa referencia a Empresa.id

**Stacking** (marcaTemporal, usuario, cantidad, empresa, interesStacking)

PK: (marcaTemporal, usuario, empresa)

FK: usuario referencia a TenerParticipaciones.usuario

FK: empresa referencia a TenerParticipaciones.empresa

**OfertaPacto** (marcaTemporal, comprador, empresa, cantidad, precio, fechaVenta, comisionPacto)

PK: (marcaTemporal, comprador, empresa)

FK: comprador referencia a Usuario.id

FK: empresa referencia a Empresa.id

**AcuerdoPacto**(marcaTemporal, comprador, empresa, vendedor)

PK: (marcaTemporal, comprador, empresa)

FK: marcaTemporal referencia a OfertaPacto.marcaTemporal

FK: comprador referencia a OfertaPacto.comprador

FK: empresa referencia a OfertaPacto.empresa

FK: vendedor referencia a TenerParticipaciones.usuario

Se optó por mantener la superclase *Usuario* ya que se necesita asegurar que la ID de las empresas, inversores y del regulador son únicas en el sistema. A mayores, se introdujo en dicha clase un atributo llamado *tipoUsuario* para especificar el tipo al que pertenece, facilitando la identificación.

## 3 Script de Creación de la Base de Datos

### 3.1 Tabla usuario

```
create table usuario
(
    id varchar(10) not null primary key,
    clave varchar(30) not null,
    saldoDisponible float default 0.0 check (saldoDisponible >= 0.0),
    tipoUsuario varchar(15) check (tipoUsuario in
('Inversor','Empresa','Regulador'))
);
```

### 3.2 Tabla empresa

```
create table empresa
(
    usuario varchar(10) not null,
    cif varchar(9) not null,
    nombre varchar(20) not null,
    direccion varchar(40),
    telefono varchar(10),
    saldoBloqueado float default 0.0 check(saldoBloqueado >= 0.0),
    partiBloqueadas integer default 0,

    primary key(usuario),
    foreign key(usuario) references usuario(id)
        on delete cascade on update cascade
);
```

### 3.3 Tabla inversor

```
create table inversor
(
    usuario varchar(10) not null,
    dni varchar(9) not null,
    nombre varchar(20) not null,
    apellido1 varchar(15) not null,
    apellido2 varchar(15),
    direccion varchar(40),
    telefono varchar(10),

    primary key(usuario),
    foreign key(usuario) references usuario(id)
        on delete cascade on update cascade
);
```

### 3.4 Tabla regulador

```
create table regulador
(
    usuario varchar(10) not null,
    comision float default 1.0 check(comision > 0.0),
    interes float default 1.0 check(interres > 0.0),

    primary key(usuario),
    foreign key(usuario) references usuario(id)
        on delete cascade on update cascade
);
```

### 3.5 Tabla prerregistro

```
create table prerregistro
(
    marcaTemporal timestamp without time zone not null primary key,
    id varchar(10) not null,
    clave varchar(30) not null,
    tipoUsuario varchar(15) check (tipoUsuario IN ('Inversor', 'Empresa'))
);
```

### 3.6 Tabla datos

```
create table datos
(
    marcaTemporalPrerregistro timestamp without time zone not null,
    dato varchar(150) not null,

    primary key(marcaTemporalPrerregistro,dato),
    foreign      key(marcaTemporalPrerregistro)      references
prerregistro(marcaTemporal)
    on delete cascade on update cascade
);
```

### 3.7 Tabla baja

```
create table baja
(
    usuario varchar(10) not null,
    marcaTemporal timestamp without time zone not null,

    primary key(usuario),
    foreign key(usuario) references usuario(id)
    on delete cascade on update cascade
);
```

### 3.8 Tabla mercado

```
create table mercado
(
    usuario varchar(10) not null,
    empresa varchar(10) not null,
    marcaTemporal timestamp without time zone not null,
    comisionVenta float default 1.0 check(comisionVenta > 0.0),
    cantidadParticipaciones integer not null,
    precioParticipacion float not null,

    primary key(usuario,marcaTemporal),
    foreign key(usuario) references usuario(id)
    on delete cascade on update cascade,
    foreign key(empresa) references empresa(usuario)
    on delete cascade on update cascade
);
```

### 3.9 Tabla tenerParticipaciones

```
create table tenerParticipaciones
(
    usuario varchar(10) not null,
    empresa varchar(10) not null,
    cantidad integer default 0,
```

```

    primary key(usuario,empresa),
    foreign key(usuario) references usuario(id)
        on delete cascade on update cascade,
    foreign key(empresa) references empresa(usuario)
        on delete cascade on update cascade
);

```

### 3.10 Tabla informacionBeneficios

```

create table informacionBeneficios(
    empresa varchar(10) not null,
    fechaPago timestamp without time zone not null,
    importePorParticipacion float,
    cantidadParticipaciones integer,
    participacionesPorUsuario integer,
    usuariosConParticipaciones integer,

    foreign key(empresa) references empresa(usuario)
        on delete cascade on update cascade
);

```

### 3.11 Tabla stacking

```

create table stacking
(
    marcaTemporal timestamp without time zone not null,
    usuario varchar(10) not null,
    empresa varchar(10) not null,
    cantidad integer default 0,
    interesStacking float default 1.0 check(interresStacking > 0.0),

    primary key(marcaTemporal,usuario,empresa),
    foreign key(usuario,empresa) references
tenerParticipaciones(usuario,empresa)
        on delete cascade on update cascade
);

```

### 3.12 Tabla ofertaPacto

```

create table ofertaPacto
(
    marcaTemporal timestamp without time zone not null,
    comprador varchar(10) not null,
    empresa varchar(10) not null,
    cantidad integer not null,
    precio float not null,
    fechaVenta timestamp without time zone not null,
    comisionPacto float not null,

    primary key(marcaTemporal,comprador,empresa),
    foreign key(comprador) references usuario(id)
        on delete cascade on update cascade,
    foreign key(empresa) references empresa(usuario)
        on delete cascade on update cascade
);

```

### 3.13 Tabla acuerdoPacto

```
create table acuerdoPacto
(
    marcaTemporal timestamp without time zone not null,
    comprador varchar(10) not null,
    empresa varchar(10) not null,
    vendedor varchar(10) not null,

    primary key(marcaTemporal,comprador,empresa),
    foreign      key(marcaTemporal,comprador,empresa)      references
ofertaPacto(marcaTemporal,comprador,empresa)
    on delete cascade on update cascade,
    foreign      key(vendedor,empresa)                    references
tenerParticipaciones(usuario,empresa)
    on delete cascade on update cascade
);
```

### 3.14 Comentarios

Se introdujeron restricciones en cuanto a la inserción de datos de tipo *not null* y cláusulas *check* para forzar la inserción de valores para determinados atributos y comprobar que las cantidades introducidas fueran positivas, ya que valores negativos carecerían de sentido.

Se ha seleccionado el tipo de dato *timestamp* (marca temporal) para evitar cualquier tipo de problema en tablas como mercado, en donde un usuario puede poner a la venta varias participaciones en un mismo día.

Como no tendría sentido mantener en el sistema información sobre usuarios que se han dado de baja en el mismo, las claves foráneas usadas en cuanto a éstos tienen borrado en cascada. La actualización es en cascada ya que si cambia el ID del usuario este debería poder seguir referenciándose en las demás tablas en las que está involucrado.



## 4 Interfaz

### 4.1 Ventana de autenticación de usuarios

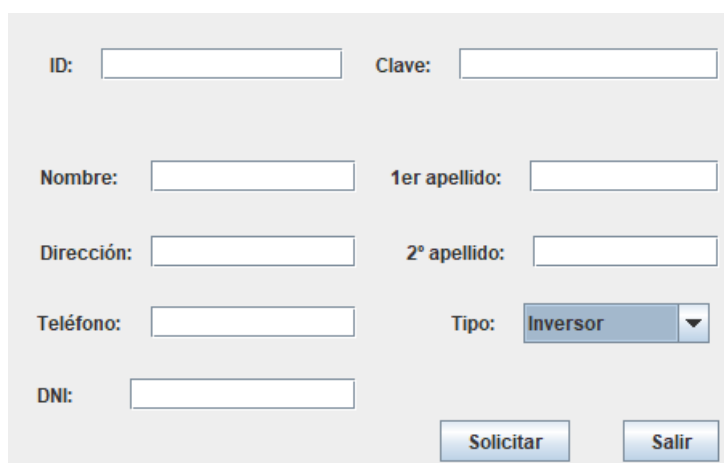
Al iniciar la aplicación se le pedirá al usuario su ID y su contraseña para su autenticación. En caso de no estar registrado, el usuario debe seleccionar el botón “Solicitar registro” que lo redigirá a la ventana de solicitud de registro (4.2). En caso de introducir datos incorrectos y pulsar el botón “Aceptar”, se mostrará una etiqueta con el aviso correspondiente. Si el usuario existe y se han introducidos los datos correctamente, el siguiente paso será la apertura de las correspondientes ventanas según su tipo de usuario.



Esta imagen muestra la interfaz de autenticación de usuarios. Incluye dos campos de entrada: "Usuario:" y "Contraseña:". Debajo de los campos, hay un mensaje de error en rojo que dice "Autenticación incorrecta". En la parte inferior, hay tres botones: "Solicitar registro", "Aceptar" y "Cancelar".

### 4.2 Ventana de solicitud de registro

Si el usuario pulsa el botón “Solicitar registro” de la ventana de autenticación de usuarios (4.1), ésta abrirá una nueva ventana que permitirá a una empresa o inversor introducir los diferentes datos necesarios para tener un perfil en el sistema. El tipo de usuario por defecto será un inversor. Los datos “ID”, “Clave”, “Nombre” y “DNI” son campos requeridos obligatoriamente por ambos tipos de usuarios, mientras que el “1er apellido” sólo lo será en caso de ser un inversor el interesado. Al finalizar, si el botón “Solicitar” es pulsado se registrará la solicitud con los datos correspondientes.



Esta imagen muestra la interfaz de solicitud de registro. Incluye varios campos de entrada: "ID:", "Clave:", "Nombre:", "1er apellido:", "Dirección:", "2º apellido:", "Teléfono:", "DNI:". Hay un menú desplegable para "Tipo:" con la opción "Inversor" seleccionada. En la parte inferior, hay dos botones: "Solicitar" y "Salir".

### 4.3 Ventana del regulador

El botón “Cerrar sesión” permite salir de la aplicación en cualquier momento y está presente en todas las pestañas del regulador. La ventana consta de 5 pestañas. La primera permite al regulador modificar sus datos: la ID, la clave, su saldo, la comisión del mercado activa y el interés de stacking activo en ese momento. Estos datos se actualizan en el momento que se pulsa el botón “Guardar cambios”.

Cuenta
Saldos
Prerregistros
Bajas
Pagos

ID: APerez
Clave: •

Saldo disponible: 0.12 €

Comisión de compraventa: 1.5 %

Interés de stacking: 1.0 %

Guardar cambios

Usuario conectado: APerez
Cerrar sesión

La segunda pestaña concede al regulador la oportunidad de modificar los saldos de los diferentes usuarios de la aplicación. La aplicación permite el filtrado de un usuario por su ID. Para cambiar los saldos se hace directamente sobre la columna de la derecha. Estos cambios no serán efectuados hasta que el regulador presione el botón “Guardar cambios”

Cuenta
Saldos
Prerregistros
Bajas
Pagos

Buscar por ID de usuario:
Buscar

ID	Tipo	Saldo
APerez	Regulador	0.12
carloslv	Inversor	175.0
ImpeTelf	Empresa	999.88
JaiVazq	Inversor	0.0
joseF	Inversor	1000.0
Nlnfor	Empresa	1000.0
pepeS	Inversor	50.0
QLCiberSec	Empresa	975.5

Guardar cambios

Usuario conectado: APerez
Cerrar sesión

La pestaña “Prerregistros” da acceso al regulador a las peticiones de entrada a la aplicación, permite ver los datos de estos y la fecha de solicitud. El regulador puede confirmar o rechazar los registros. En caso de pulsar “Confirmar registro”, el usuario será creado e introducido en el sistema. Por otro lado, si el regulador decide pulsar “Rechazar registro” el registro este se eliminará.

Marca temporal	ID	Tipo
2021-03-18 00:00:00.0	JaiVazq	Inversor
2021-04-01 00:00:00.0	KWPc	Empresa

DNI	<input type="text" value="35662751H"/>	Nombre:	<input type="text" value="Jaime Vázquez"/>
Dirección:	<input type="text" value="Rúa Compostela"/>	Teléfono:	<input type="text" value="629955212"/>

Usuario conectado: APerez

La cuarta pestaña se corresponde con las solicitudes de baja por parte de usuarios de la aplicación. En ella se aprecia una tabla con dichas solicitudes. Se podrá seleccionar el usuario del cuál se quiere gestionar la baja. En caso de que el usuario no tenga participaciones ni saldo, se permite presionar el botón “Confirmar baja”; en caso contrario, la aplicación no permitirá presionarlo. Si el usuario tiene saldo distinto de 0, antes deberá transferirse el mismo (yendo a la pestaña de saldos y poniendo a 0 el saldo del usuario correspondiente), y después ya se podrá confirmar la baja a través del botón. También se pueden rechazar bajas seleccionándolas en la tabla y pulsando “Rechazar baja”.

Marca temporal	ID	Saldo disponible	Num. participaciones
2021-03-02 00:00:00.0	pepeS	0.0	0
2021-05-05 20:14:07.152...	carloslv	175.0	55

Usuario conectado: APerez

La última pestaña de la que dispondrá un regulador será la de “Pagos”. En ella puede ver los distintos anuncios sobre pagos de beneficios realizados por las empresas de la aplicación. Si el regulador pulsa el botón “Dar de baja anuncio”, la oferta de pago será eliminada y no se llevará a cabo.

Fecha de pago	ID de la empresa	Importe/part.	Part/usuario
2021-05-06 20:14:07.15...	QLCiberSec	20.0	1

Usuario conectado: APerez

#### 4.4 Ventana principal de empresa e inversor

Esta ventana es común para ambos tipos de usuario, con la excepción de ciertos botones y campos de texto que se explicarán más adelante. Tiene un botón “Cerrar sesión” que al ser presionado envía al usuario a la pestaña de autenticación (4.1) y está presente en ambas pestañas de la ventana.

En primera instancia se encuentra la pestaña “Operaciones”, que consta de una tabla que muestra las participaciones de las que el usuario es propietario, distinguiendo cuales de ellas están a la venta en el mercado. Las participaciones se consiguen de las tablas “tenerParticipaciones” (3.9) y “mercado”

(3.8) de la base de datos. Las participaciones de una empresa que el usuario haya puesto a la venta aparecerán en una fila distinta de aquellas que no están en el mercado, aunque sean de la misma empresa. Los botones “Mercado”, “Futuros” y “Stacking” permite al usuario acceder al mercado de participaciones, gestor de contratos de futuros y gestor de stacking, respectivamente.

Si el usuario que se ha registrado es una empresa, dispondrá de dos botones, en la parte inferior izquierda, que le permitirán dar de alta o baja una cantidad de participaciones que podrán determinar en el campo superior a dichos botones.

Cuando el usuario selecciona una fila de la tabla, los campos “Empresa”, “Precio” y “Cantidad” tomarán los valores que se correspondan a la tabla. Estos dos últimos podrán ser modificados si las participaciones no están en venta, habilitándose el botón de “Vender” para que el usuario pueda introducirlas en el mercado con el precio establecido. En caso de que ya estén a la venta, solo se podrá utilizar el botón “Retirar del mercado” para que ya no estén en venta. Estas dos últimas operaciones se realizan sobre la tabla “mercado” (3.8) de la base de datos. Además, se dispondrá en cada momento de la comisión actual marcada por el regulador y que afectará a cada una de las ventas que se realicen en ese periodo. Si alguna participación se pone a la venta, la tabla se actualizará con una nueva entrada mostrando las participaciones en venta con la etiqueta “Sí” en la columna “En el mercado”.

Empresa	Cantidad	Precio/part.	En el mercado
ImpeTelf	2	0.0	No

*Operaciones de inversor*

Empresa	Cantidad	Precio/part.	En el mercado
Ninfor	15	25.0	Sí
Ninfor	5	20.0	Sí

*Operaciones de empresa*

La segunda pestaña que poseen ambos tipos de usuarios es la de “Cuenta”, la cual proporciona una vista de los datos de perfil del usuario. Estos datos son extraídos de la tabla “usuario” (3.1) y de la tabla correspondiente al tipo de usuario. En esta pestaña podrá editar determinados datos de su perfil y solicitar una baja. Los cambios en el perfil no serán actualizados en las diferentes tablas hasta que el botón “Guardar” sea pulsado. La baja se solicitará a través de “Solicitar baja”. Una empresa podrá visualizar los datos asociados a su saldo y participaciones bloqueadas.

*Cuenta de un inversor*

*Cuenta de una empresa*

## 4.5 Ventana de mercado

Esta ventana es común para ambos tipos de usuario. En ella se puede ver una tabla que contiene todas las participaciones que se encuentran en un estado de venta por parte de algún usuario. Esta tabla se rellena con los datos de la tabla “mercado” (3.8). A través de un desplegable, el usuario podrá seleccionar una de las empresas que poseen participaciones en venta, podrá seleccionar un número de participaciones menor o igual a las disponibles, no permitiéndose a seleccionar un número mayor, para indicar el precio máximo se debe tener en cuenta que un usuario puede tener diferentes ventas de participaciones de una empresa a distintos precios, por lo tanto se deberá indicar como mínimo el mayor precio, siempre que se vayan seleccionar de ofertas distintas. En la parte inferior de la sección de compra, aparece otra tabla que informará al usuario de los diferentes beneficios que pagarán las empresas por la posesión de sus participaciones. En la parte superior se podrá visualizar la comisión actual del regulador en caso de querer realizar una venta de participaciones. Si una compra no se puede realizar, se le informará al usuario con distintas ventanas de excepciones y si una compra se realiza con éxito se mostrará una etiqueta debajo del botón comprar.

*Vista del mercado de un inversor*

*Vista del mercado de una empresa*

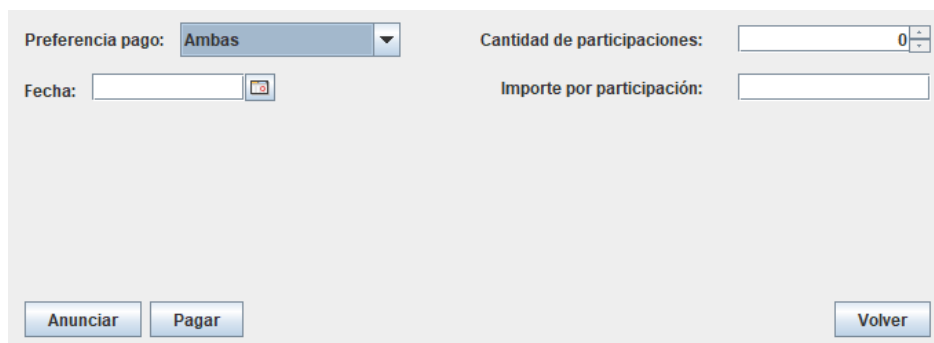
La ventana de mercado de empresas posee un botón a mayores, “Gestionar pagos”. La interacción con este botón activará la ventana de gestión de pagos (4.6).

#### 4.6 Ventana de gestión de pagos

La ventana de gestión de pagos permite a una empresa anunciar un tipo de beneficio a aquellos usuarios que posean participaciones de su empresa. Este beneficio puede ser dinero en efectivo, participaciones o ambas.

La empresa debe seleccionar en el desplegable la preferencia de pago: saldo, participaciones o ambos. Según la opción seleccionada se podrá seleccionar una cantidad de participaciones o un importe por cada participación poseída, pudiendo ser ambas. Además, podrá elegir un día a través de un calendario que se desplegará pulsando en el botón que aparece en el lateral derecho del campo “Fecha”. Por último, la empresa puede seleccionar si desea que este pago cuente con un anuncio previo o se realice el pago inmediatamente, pulsando los botones “Anunciar” o “Pagar” respectivamente.

Cuando se anuncia un pago esta se introducirá en la tabla “informacionBeneficios” (3.10).



Formulario de gestión de pagos:

- Preferencia pago: **Ambas** (desplegable)
- Cantidad de participaciones:  (campo con botones de incremento y decremento)
- Fecha:  (campo con botón de calendario)
- Importe por participación:
- Botones: **Anunciar**, **Pagar**, **Volver**

#### 4.7 Ventana de gestión de contratos de futuros

Si en la ventana principal (4.4) se pulsa el botón “Futuros” se abrirá una ventana que muestra una tabla con las diferentes ofertas realizadas por otros usuarios para comprar diferentes participaciones a través de un precio marcado en una fecha indicada. Esta tabla se rellena con una consulta realizada a la tabla “ofertaPacto” (3.12). El inversor o empresa podrán filtrar ofertas a través del buscador proporcionado y que también utiliza la misma tabla. Se pueden realizar ofertas de contratos de futuros, para esto se necesita complementar los campos “Fecha de venta” que proporciona un calendario para seleccionar la fecha de compra de dichas participaciones, se debe introducir la cantidad de participaciones que se desean comprar, el precio a pagar, con la ayuda del desplegable se selecciona la empresa de la cual se quiere comprar participaciones y finalmente se debe pulsar el botón “Ofrecer”.

Por otro lado, un usuario podrá aceptar ofertas de compra que aparecen en la tabla, para ello, debe seleccionar la fila deseada en la tabla y pulsar el botón “Aceptar”, esto producirá un almacenamiento en la base de datos, concretamente en la tabla “acuerdoPacto” (3.13). El usuario será conocedor de la comisión del mercado por el regulador en todo momento.

Operaciones **Mis pactos**

Buscar por nombre:

Fecha de venta	Empresa	Cantidad	Precio/part.
2021-05-25 00:00:...	Nirvana Informática	5	10.0

Fecha de venta: 23-jun-2021   
 Cantidad:   
 Precio:   
 Empresa:

Comisión actual: 1.5 %

La segunda pestaña visible en la ventana será “Mis pactos”, que permite al usuario visualizar las ofertas de contratos de futuros que ha realizado y los contratos ya pactados. Para esto se muestran dos tablas, la primera de ellas (contratos ofertados) obtiene la información de la tabla “ofertaPacto” (3.12), mientras que la segunda de las tablas (contratos ya pactados) se obtiene de la tabla “acuerdoPacto” (3.13).

Operaciones **Mis pactos**

Ofertas:

Fecha de venta	Empresa	Cantidad	Precio/part.
2021-06-23 13:06:36...	Imperio teléfono	12	12.0
2021-05-10 13:06:36...	Imperio teléfono	7	20.0

Ya pactados:

Fecha de venta	Empresa	Cantidad	Precio/part.	Comprador	Vendedor
2021-05-20 0...	QuantumLea...	5	10.0	joseF	carloslv

Comisión actual: 1.5 %

#### 4.8 Ventana de gestión de stacking

Si un inversor o empresa que se encuentre en la ventana principal (4.4) pulsa el botón “Stacking”, se abrirá una ventana que le permitirá a dicho usuario introducir diferentes participaciones en stacking. En el lateral derecho se observa la cartera de participaciones, en ella se visualizan todas las participaciones de las cuales el usuario es propietario. Es importante destacar que el usuario solo podrá depositar participaciones si estas no se encuentran en el mercado. En el lateral derecho se



obtiene la tabla de las participaciones que se encuentran en un estado de stacking y el porcentaje de interés que tiene dicho stacking. El interés está estipulado por el regulador del sistema.

Para depositar participaciones en stacking se debe seleccionar la fila de la tabla deseada y con ayuda del campo “cantidad” de la parte inferior de la tabla se seleccionará la cantidad de participaciones a introducir, no se dejará introducir una mayor cantidad de participaciones que las totales. Estos cambios se guardarán cuando el usuario pulse el botón “Depositar” y se introducirá en la tabla “stacking” (3.11).

Si el usuario decide retirar esas participaciones de stacking y que estas vuelvan a su carta de participaciones, debe pulsar el botón “Retirar”.

**Carta de participaciones**

Empresa	Cantidad	Precio/part.	En el merc...
ImpeTelf	1	0.0	No

**Stacking**

Fecha	Empresa	Cantidad	Interés
2021-05-05...	Imperio telé...	1	1.0

Cantidad:

Interés actual: 1.0 %

## 4.9 Ventana de avisos/excepciones

Además de las ventanas descritas anteriormente, existe una ventana para mostrar avisos o excepciones en caso de ser necesarios.

AVISO

Esta empresa no tiene tantas participaciones a la venta.  
Selecciona una cantidad menor

Cerrar



## 5 Desarrollo de Funcionalidades

### 5.1 [FU\_01] Acceso Sistema

La ventana utilizada para esta funcionalidad será la ventana de autenticación de usuarios (4.1). En primera instancia se realiza una búsqueda en la tabla “usuarios” (3.1) de la base de datos con las credenciales introducidas.

La consistencia de la base de datos se mantiene a través del uso del commit. Para ello, se desactiva antes de la preparación del string que realizará la consulta para ser ejecutado manualmente después de la ejecución de dicha consulta. Se asegura que en caso de excepción, se restauren los valores al estado anterior de la ejecución. Finalmente, se vuelve a activar el auto-commit.

Para saber si las credenciales introducidas son correctas, se realiza la siguiente consulta:

```
public Usuario validarUsuario(String idUsuario, String clave) {

    //Variables locales
    Usuario resultado = null;
    Connection con;
    PreparedStatement stmUsuario = null;
    ResultSet rsUsuario;
    String id;

    //Acceso a la conexion
    con = this.getConexion();
    try {
        con.setAutoCommit(false);

        //Preparacion string
        stmUsuario = con.prepareStatement("select id, clave "
            + "from usuario "
            + "where id = ? and clave = ?");
        stmUsuario.setString(1, idUsuario);
        stmUsuario.setString(2, clave);

        //Ejecucion query
        rsUsuario = stmUsuario.executeQuery();

        //Lectura de resultados
        if (rsUsuario.next()) {
            id = rsUsuario.getString("id");
            resultado = obtenerUsuario(id);
        }

        con.commit();
    } catch (SQLException e) {
        try {
            con.rollback();
            System.out.println(e.getMessage());

            this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
        }
    }

    this.getFachadaAplicacion().muestraExcepcion(ex.getMessage());
} finally {
    try {
        con.setAutoCommit(true);
    }
```

```

        stmUsuario.close();
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
    this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
}
return resultado;
}

```

Para obtener el usuario se comprueba su tipo y se traslada la petición al método correspondiente para obtener la información del mismo, bien sea en la tabla “usuario”.

```

public Usuario obtenerUsuario(String id) {
    Usuario resultado = null;
    Connection con;
    PreparedStatement stmUsuario = null;
    ResultSet rsUsuario;

    con = this.getConexion();

    try {
        con.setAutoCommit(false);

        //CREACION DE USUARIO
        stmUsuario = con.prepareStatement("select
id,clave,saldodisponible,tipousuario from usuario where id = ?");
        stmUsuario.setString(1, id);
        rsUsuario = stmUsuario.executeQuery();

        if (rsUsuario.next()) {

            switch (rsUsuario.getString("tipousuario")) {
                case "Empresa":
                    resultado = obtenerEmpresa(id,
rsUsuario.getString("clave"), rsUsuario.getFloat("saldodisponible"));
                    break;
                case "Inversor":
                    resultado = obtenerInversor(id,
rsUsuario.getString("clave"), rsUsuario.getFloat("saldodisponible"));
                    break;
                case "Regulador":
                    resultado = obtenerRegulador(id,
rsUsuario.getString("clave"), rsUsuario.getFloat("saldodisponible"));
                    break;
            }

        }

        con.commit();
    } catch (SQLException e) {
        try {
            con.rollback();
            System.out.println(e.getMessage());
        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
        }
    }
    this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
}

```

```

this.getFachadaAplicacion().muestraExcepcion(ex.getMessage());
    }
} finally {
    try {
        stmUsuario.close();
    } catch (SQLException e) {
        System.out.println("Imposible cerrar cursores");
    }
}
return resultado;
}
}

```

Los métodos llamados en el switch son los siguientes:

```

private Empresa obtenerEmpresa(String id, String clave, float
saldoDisponible) {

    //Variables locales
    Empresa resultado = null;
    Connection con;
    PreparedStatement stmUsuario = null;
    ResultSet rsUsuario;

    //Conexion
    con = this.getConexion();

    try {

        //EMPRESA
        stmUsuario = con.prepareStatement("select * from empresa where
empresa.usuario = ?");
        stmUsuario.setString(1, id);

        //Query
        rsUsuario = stmUsuario.executeQuery();

        //Obtencion de datos
        if (rsUsuario.next()) {
            resultado = new Empresa(id,
                clave,
                saldoDisponible,
                rsUsuario.getString("cif"),
                rsUsuario.getString("nombre"),
                rsUsuario.getString("direccion"),
                rsUsuario.getString("telefono"),
                rsUsuario.getFloat("saldoBloqueado"),
                rsUsuario.getInt("partiBloqueadas"));
        }

    } catch (SQLException e) {
        System.out.println(e.getMessage());
        this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
    } finally {
        try {
            stmUsuario.close();
        } catch (SQLException e) {
            System.out.println("Imposible cerrar cursores");
        }
    }
}

```

```

        return resultado;
    }

    private Inversor obtenerInversor(String id, String clave, float
saldoDisponible) {

        //Variables locales
        Inversor resultado = null;
        Connection con;
        PreparedStatement stmUsuario = null;
        ResultSet rsUsuario;

        //Conexion
        con = this.getConexion();

        try {
            //EMPRESA
            stmUsuario = con.prepareStatement("select * from inversor where
inversor.usuario = ?");
            stmUsuario.setString(1, id);

            //Query
            rsUsuario = stmUsuario.executeQuery();

            //Obtencion de datos
            if (rsUsuario.next()) {
                resultado = new Inversor(id,
                    clave,
                    saldoDisponible,
                    rsUsuario.getString("dni"),
                    rsUsuario.getString("nombre"),
                    rsUsuario.getString("apellido1"),
                    rsUsuario.getString("apellido2"),
                    rsUsuario.getString("direccion"),
                    rsUsuario.getString("telefono"));
            }

        } catch (SQLException e) {
            System.out.println(e.getMessage());
            this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
        } finally {
            try {
                stmUsuario.close();
            } catch (SQLException e) {
                System.out.println("Imposible cerrar cursores");
            }
        }

        return resultado;
    }

    private Regulador obtenerRegulador(String id, String clave, float
saldoDisponible) {

        //Variables locales
        Regulador resultado = null;
        Connection con;
        PreparedStatement stmUsuario = null;
        ResultSet rsUsuario;

```

```

//Conexion
con = this.getConexion();

try {

    //EMPRESA
    stmUsuario = con.prepareStatement("select * from regulador
where regulador.usuario = ?");
    stmUsuario.setString(1, id);

    //Query
    rsUsuario = stmUsuario.executeQuery();

    //Obtencion de datos
    if (rsUsuario.next()) {
        resultado = new Regulador(id,
            clave,
            saldoDisponible,
            rsUsuario.getFloat("comision"),
            rsUsuario.getFloat("interes"));
    }

} catch (SQLException e) {
    System.out.println(e.getMessage());
    this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
} finally {
    try {
        stmUsuario.close();
    } catch (SQLException e) {
        System.out.println("Imposible cerrar cursors");
    }
}

return resultado;
}

```

## 5.2 [FU-02] Solicitud Registro

La solicitud de un registro se realiza en la ventana “Solicitud de registro” (4.2). Cuando el botón “Solicitar” sea presionado y los campos necesarios para cada tipo de usuario estén rellenos, se enviará la solicitud a la tabla “prerregistro” (3.5) y “datos” (3.6) de la base de datos.

La consistencia de la base de datos se mantiene a través del uso del commit, ya que en él se realizarán inserciones en diferentes tablas. Cuando la conexión con la base de datos se ha realizado, se realiza una búsqueda del ID introducido por si este ya existe. Posteriormente se desactiva el auto-commit y se prepara la primera inserción a realizar. Esta es la inserción de los datos *id*, *clave* y *tipoUsuario* en la tabla “prerregistro” (3.5). A continuación, se preparan las diferentes inserciones en la tabla “datos” (3.6), en la cual se insertará cada dato por separado. Las consultas se comprometen cuando estén preparadas, para, finalmente reactivar el auto-commit. En caso de que se produzca cualquier tipo de excepción se realizará un rollback.

```

public boolean insertarPrerregistro(Prerregistro p) {
    Connection con;
    PreparedStatement stmPre = null;
    boolean resultado = true;

    con = this.getConexion();

```

```

        try {
            if
(this.getFachadaAplicacion().obtenerUsuario(p.getUsuario().getId()) ==
null) {
                con.setAutoCommit(false);

                stmPre = con.prepareStatement("insert into
prerregistro(marcaTemporal, id, clave, tipoUsuario) "
                    + "values(?, ?, ?, ?)");
                stmPre.setTimestamp(1, p.getMarcaTemporal());
                stmPre.setString(2, p.getUsuario().getId());
                stmPre.setString(3, p.getUsuario().getClave());
                if (p.getUsuario() instanceof Empresa) {
                    stmPre.setString(4, "Empresa");
                } else if (p.getUsuario() instanceof Inversor) {
                    stmPre.setString(4, "Inversor");
                }
                stmPre.executeUpdate();

                stmPre = con.prepareStatement("insert into
datos(marcaTemporalPrerregistro, dato) values(?,?)");
                stmPre.setTimestamp(1, p.getMarcaTemporal());
                stmPre.setString(2, "id:" + p.getUsuario().getId());
                stmPre.executeUpdate();

                if (p.getUsuario() instanceof Empresa) {
                    Empresa e = (Empresa) p.getUsuario();

                    stmPre = con.prepareStatement("insert into
datos(marcaTemporalPrerregistro, dato) values(?,?)");
                    stmPre.setTimestamp(1, p.getMarcaTemporal());
                    stmPre.setString(2, "cif:" + e.getCif());
                    stmPre.executeUpdate();

                    stmPre = con.prepareStatement("insert into
datos(marcaTemporalPrerregistro, dato) values(?,?)");
                    stmPre.setTimestamp(1, p.getMarcaTemporal());
                    stmPre.setString(2, "nombre:" + e.getNombre());
                    stmPre.executeUpdate();

                    stmPre = con.prepareStatement("insert into
datos(marcaTemporalPrerregistro, dato) values(?,?)");
                    stmPre.setTimestamp(1, p.getMarcaTemporal());
                    stmPre.setString(2, "direccion:" + e.getDireccion());
                    stmPre.executeUpdate();

                    stmPre = con.prepareStatement("insert into
datos(marcaTemporalPrerregistro, dato) values(?,?)");
                    stmPre.setTimestamp(1, p.getMarcaTemporal());
                    stmPre.setString(2, "tlf:" + e.getTelefono());
                    stmPre.executeUpdate();
                } else if (p.getUsuario() instanceof Inversor) {
                    Inversor i = (Inversor) p.getUsuario();

                    stmPre = con.prepareStatement("insert into
datos(marcaTemporalPrerregistro, dato) values(?,?)");
                    stmPre.setTimestamp(1, p.getMarcaTemporal());
                    stmPre.setString(2, "dni:" + i.getDni());
                    stmPre.executeUpdate();

                    stmPre = con.prepareStatement("insert into

```



```

datos(marcaTemporalPrerregistro, dato) values(?,?)");
        stmPre.setTimestamp(1, p.getMarcaTemporal());
        stmPre.setString(2, "nombre:" + i.getNombre());
        stmPre.executeUpdate();

        stmPre = con.prepareStatement("insert into
datos(marcaTemporalPrerregistro, dato) values(?,?)");
        stmPre.setTimestamp(1, p.getMarcaTemporal());
        stmPre.setString(2, "ap1:" + i.getApellido1());
        stmPre.executeUpdate();

        stmPre = con.prepareStatement("insert into
datos(marcaTemporalPrerregistro, dato) values(?,?)");
        stmPre.setTimestamp(1, p.getMarcaTemporal());
        stmPre.setString(2, "ap2:" + i.getApellido2());
        stmPre.executeUpdate();

        stmPre = con.prepareStatement("insert into
datos(marcaTemporalPrerregistro, dato) values(?,?)");
        stmPre.setTimestamp(1, p.getMarcaTemporal());
        stmPre.setString(2, "direccion:" + i.getDireccion());
        stmPre.executeUpdate();

        stmPre = con.prepareStatement("insert into
datos(marcaTemporalPrerregistro, dato) values(?,?)");
        stmPre.setTimestamp(1, p.getMarcaTemporal());
        stmPre.setString(2, "tlf:" + i.getTelefono());
        stmPre.executeUpdate();

    }

    con.commit();
} else {
    this.getFachadaAplicacion().muestraExcepcion("El ID
introducido ya está en uso");
    resultado = false;
}
} catch (SQLException e) {
    try {
        con.rollback();
        System.out.println(e.getMessage());

this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
    } catch (SQLException ex) {
        System.out.println(ex.getMessage());

this.getFachadaAplicacion().muestraExcepcion(ex.getMessage());
    }
} finally {
    try {
        con.setAutoCommit(true);
        if (stmPre != null) {
            stmPre.close();
        }
    } catch (SQLException e) {
        System.out.println("Imposible cerrar cursores");
    }
}

return resultado;
}

```

### 5.3 [FU\_03] Confirmación Registro

La ventana disponible para la confirmación de un registro en el sistema se encuentra en la pestaña “Prerregistros” de la ventana del regulador (4.3). El regulador puede seleccionar cualquiera de las solicitudes que haya en la tabla.

Se puede observar como la consistencia en la base de datos se mantiene con el uso del commit. En él se realizan varias modificaciones pues intervienen varias tablas de la base de datos. Una vez establecida la conexión con la base de datos se desactiva el auto-commit y se preparan las diferentes consultas a realizar. La primera de ellas será la responsable de borrar los datos del usuario seleccionado en la tabla anteriormente, para ello se usará de la tabla “datos” (3.6), posteriormente el borrado se realizará en la tabla de “prerregistros” (3.5). Finalmente, se insertará al usuario de dicha solicitud en el sistema, para ello, se introducirá en la tabla “usuario” (3.1) y en la correspondiente a su tipo (“empresa” (3.2) o “inversor” (3.3)). Una vez están preparadas las consultas se realiza el commit para que se comprometan todas juntas. Para finalizar se activa de nuevo el auto-commit.

```
public void confirmarPrerregistro(Prerregistro p) {
    Connection con;
    PreparedStatement stmPre = null;

    con = this.getConexion();
    try {
        con.setAutoCommit(false);

        stmPre = con.prepareStatement("delete from datos where
marcaTemporalPrerregistro = ?");
        stmPre.setTimestamp(1, p.getMarcaTemporal());
        stmPre.executeUpdate();

        stmPre = con.prepareStatement("delete from prerregistro where
marcaTemporal = ?");
        stmPre.setTimestamp(1, p.getMarcaTemporal());
        stmPre.executeUpdate();

        this.getFachadaAplicacion().insertarUsuario(p.getUsuario());

        con.commit();
    } catch (SQLException e) {
        try {
            con.rollback();
            System.out.println(e.getMessage());

            this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
        } catch (SQLException ex) {
            System.out.println(ex.getMessage());

            this.getFachadaAplicacion().muestraExcepcion(ex.getMessage());
        }
    } finally {
        try {
            con.setAutoCommit(true);
            stmPre.close();
        } catch (SQLException e) {
            System.out.println("Imposible cerrar cursores");
        }
    }
}
```

```
}
```

El método “insertarUsuario” tiene lo siguiente:

```
public void insertarUsuario(Usuario u) {
    Connection con = this.getConexion();
    PreparedStatement stmUsuario = null;

    try {
        stmUsuario = con.prepareStatement("insert into usuario
values(?,?,?,?,?)");
        stmUsuario.setString(1, u.getId());
        stmUsuario.setString(2, u.getClave());
        stmUsuario.setFloat(3, u.getSaldoDisponible());
        stmUsuario.setString(4, u.getTipo().name());
        stmUsuario.executeUpdate();

        //Se tiene que insertar tambien en la tabla de Inversor o
Empresa
        if (u instanceof Inversor) {
            Inversor i = (Inversor) u;
            stmUsuario = con.prepareStatement("insert into inversor
values(?,?,?,?,?,?,?,?)");

            stmUsuario.setString(1, i.getId());
            stmUsuario.setString(2, i.getDni());
            stmUsuario.setString(3, i.getNombre());
            stmUsuario.setString(4, i.getApellido1());
            stmUsuario.setString(5, i.getApellido2());
            stmUsuario.setString(6, i.getDireccion());
            stmUsuario.setString(7, i.getTelefono());
            stmUsuario.executeUpdate();
        } else {
            Empresa e = (Empresa) u;
            stmUsuario = con.prepareStatement("insert into empresa
values(?,?,?,?,?,?)");

            stmUsuario.setString(1, e.getId());
            stmUsuario.setString(2, e.getCif());
            stmUsuario.setString(3, e.getNombre());
            stmUsuario.setString(4, e.getDireccion());
            stmUsuario.setString(5, e.getTelefono());
            stmUsuario.executeUpdate();
        }
    } catch (SQLException e) {
        System.out.println(e.getMessage());
        this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
    } finally {
        try {
            stmUsuario.close();
        } catch (SQLException e) {
            System.out.println("Imposible cerrar cursores");
        }
    }
}
```

## 5.4 [FU\_04] Modificación Datos

Las modificaciones de los datos de un usuario tienen lugar en la pestaña cuenta de la ventana principal de cada tipo de usuario (4.3 y 4.4). En esta pestaña, el usuario podrá modificar diferentes datos de su perfil. Estos cambios se producirán cuando se pulse el botón “Guardar” y se ejecuta la funcionalidad.

Estos cambios afectan a distintas tablas por lo que se desactiva el auto-commit luego de establecer la conexión con la base de datos, dejando su activación al final de la ejecución de la funcionalidad. Cuando las actualizaciones están preparadas se realiza el commit, produciendo que se comprometan todas juntas. En caso de que las sentencias producen una excepción de SQL se realizará un rollback.

En primer lugar, se debe comprobar qué tipo de usuario es el que pide la modificación de datos. Si el usuario es de tipo “Regulador” se hará una actualización en la tabla “regulador” (3.4) y con los datos necesarios que interesa saber de un regulador.

```
if (u instanceof Regulador) {
    Regulador r = (Regulador) u;
    stmActualizacion = con.prepareStatement("update regulador "
                                           + "set comision = ?, "
                                           + "    interes = ? "
                                           + "where usuario = ?");

    stmActualizacion.setFloat(1, r.getComisionCompraventa());
    stmActualizacion.setFloat(2, r.getInteresStacking());
    stmActualizacion.setString(3, idAntigua);
    stmActualizacion.executeUpdate();
}
```

Por otro lado, si el usuario que modifica sus datos es del tipo “Empresa” los cambios se realizarán en la tabla “empresa” (3.8). La actualización resultante es:

```
else if (u instanceof Empresa) {
    Empresa e = (Empresa) u;
    stmActualizacion = con.prepareStatement("update empresa "
                                           + "set cif = ?, "
                                           + "    nombre = ?, "
                                           + "    direccion = ?, "
                                           + "    telefono = ?, "
                                           + "    saldoBloqueado = ? "
                                           + "where usuario = ?");

    stmActualizacion.setString(1, e.getCif());
    stmActualizacion.setString(2, e.getNombre());
    stmActualizacion.setString(3, e.getDireccion());
    stmActualizacion.setString(4, e.getTelefono());
    stmActualizacion.setFloat(5, e.getSaldoBloqueado());
    stmActualizacion.setString(6, idAntigua);
    stmActualizacion.executeUpdate();
}
```

El último tipo de usuario que puede realizar cambios en los datos es el inversor. Por lo tanto, los cambios se realizarán en la tabla “inversor” (3.4), resultando así la sentencia a ejecutar:

```
else if (u instanceof Inversor) {
    Inversor i = (Inversor) u;
    stmActualizacion = con.prepareStatement("update inversor "
                                           + "set dni = ?, "
                                           + "    nombre = ?, "
```

```

+ "    apellido1 = ?, "
+ "    apellido2 = ?, "
+ "    direccion = ?, "
+ "    telefono = ? "
+ "where usuario = ?");

stmActualizacion.setString(1, i.getDni());
stmActualizacion.setString(2, i.getNombre());
stmActualizacion.setString(3, i.getApellido1());
stmActualizacion.setString(4, i.getApellido2());
stmActualizacion.setString(5, i.getDireccion());
stmActualizacion.setString(6, i.getTelefono());
stmActualizacion.setString(7, idAntigua);
stmActualizacion.executeUpdate();
}

```

Una vez que una de estas tablas ha sido actualizada, se necesita realizar una actualización de los datos en la tabla de “usuarios” (3.1). Esta actualización se realizará siempre, sea cual sea el tipo de usuario anterior.

```

stmActualizacion = con.prepareStatement("update usuario "
+ "set id = ?, "
+ "    clave = ?, "
+ "    saldoDisponible = ? "
+ "where id = ?");
stmActualizacion.setString(1, u.getId());
stmActualizacion.setString(2, u.getClave());
stmActualizacion.setFloat(3, u.getSaldoDisponible());
stmActualizacion.setString(4, idAntigua);
stmActualizacion.executeUpdate();

```

## 5.5 [FU\_05] Modificación Saldo Cuenta

Esta modificación se realiza a través de la pestaña de “Saldos” que aparece en la ventana del regulador (4.3). El regulador modifica el saldo directamente de la tabla de la pestaña y pulsará el botón “Guardar cambios” para confirmar la operación. La tabla afectada por esta transacción será la tabla “usuario” (3.1).

La función que se encargará de llevar a cabo la modificación del saldo constará de una única consulta, la cual ejecuta un update al usuario cuyo ID coincida con el del usuario que recibe la función como parámetro. En vista de esto, podemos mantener el auto-commit. Para terminar, comprobamos si ha habido alguna excepción e imprimimos su mensaje asociado.

```

public void actualizarSaldo(Usuario u, float saldoNuevo) {
    Connection con;
    PreparedStatement stmActualizacion = null;

    con = this.getConexion();

    try {

        stmActualizacion = con.prepareStatement("update usuario set
saldoDisponible = ? where id = ?");
        stmActualizacion.setFloat(1, saldoNuevo);
        stmActualizacion.setString(2, u.getId());
        stmActualizacion.executeUpdate();
    }
}

```

```

    } catch (SQLException e) {
        System.out.println(e.getMessage());
        this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
    } finally {
        try {
            stmActualizacion.close();
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }
    this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
}
}
}

```

## 5.6 [FU\_06] Solicitud Baja

Para que un usuario solicite su baja del sistema tendrá que presionar el botón “Solicitar baja” que se encuentra en la pestaña “Cuenta” de la ventana principal del usuario (4.4). En caso de que la solicitud se tramite satisfactoriamente, aparecerá en la pestaña un mensaje de “Baja solicitada correctamente”.

La transacción que se ejecuta tras presionar el botón es la que se muestra a continuación, la cual implica a la tabla “baja” (3.7) de nuestra base de datos.

Comenzamos deshabilitando el auto-commit tras realizar la conexión con la base de datos para, de esta manera, garantizar la consistencia de nuestra base. A continuación, comprobamos a través de una breve consulta si existe otra solicitud de baja por parte del usuario que la acaba de solicitar. Si esto no es cierto, insertamos una nueva solicitud en nuestra tabla “baja” con el ID del usuario que la acaba de solicitar.

```

public boolean solicitarBaja(String usuario) {
    Connection con;
    PreparedStatement stmBaja = null;
    ResultSet rsCheck;
    boolean resultado = false;

    con = this.getConexion();
    try {
        con.setAutoCommit(false);

        stmBaja = con.prepareStatement("select * from baja where
usuario = ?");
        stmBaja.setString(1, usuario);
        rsCheck = stmBaja.executeQuery();

        if (rsCheck.next()) {
            this.getFachadaAplicacion().muestraExcepcion("Ya has
solicitado la baja anteriormente.\nEspera a que el regulador la procese");
        } else {
            stmBaja = con.prepareStatement("insert into
baja(usuario,marcatemporal) values(?,now())");
            stmBaja.setString(1, usuario);
            stmBaja.executeUpdate();
            resultado = true;
        }
    }
}

```

Una vez tenemos las consultas hechas, hacemos que se comprometan ejecutando un “con.commit()” y, en caso de haberse producido alguna excepción, ejecutamos un “con.rollback()”. Finalmente, volvemos a habilitar el auto-commit que habíamos desactivado al principio.

```

        con.commit();
    } catch (SQLException e) {
        try {
            con.rollback();
            System.out.println(e.getMessage());
        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
        }
    }

    this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
    } catch (SQLException ex) {
        System.out.println(ex.getMessage());
    }

    this.getFachadaAplicacion().muestraExcepcion(ex.getMessage());
    }
} finally {
    try {
        con.setAutoCommit(true);
        stmBaja.close();
    } catch (SQLException e) {
        System.out.println("Imposible cerrar cursores");
    }
}

return resultado;
}

```

## 5.7 [FU\_07] Confirmación Baja

La confirmación de una baja tiene lugar en la pestaña “Bajas” de la ventana del regulador (4.3). En esta pestaña el regulador selecciona la fila correspondiente a la baja que desea aceptar y pulsa el botón “Confirmar baja”.

Al pulsar el botón se ejecuta la siguiente transacción en la que podemos ver que se hace un borrado en cascada de los registros de cada tabla que hacen referencia a ese usuario comenzando por la tabla “baja” (3.7) y terminando en la tabla “usuario” (3.1), que debe ser de la última de la que se elimina.

```

public void validarBaja(String usuario) {
    Connection con;
    PreparedStatement stmBaja = null;

    con = this.getConexion();
    try {
        con.setAutoCommit(false);

        stmBaja = con.prepareStatement("delete from baja where usuario
= ?;"
        + "delete from inversor where usuario = ?;"
        + "delete from empresa where usuario = ?;"
        + "delete from usuario where id = ?");
        stmBaja.setString(1, usuario);
        stmBaja.setString(2, usuario);
        stmBaja.setString(3, usuario);
        stmBaja.setString(4, usuario);

        stmBaja.executeUpdate();

        con.commit();
    }
}

```

```

    } catch (SQLException e) {
        try {
            con.rollback();
            System.out.println(e.getMessage());
        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
        }
    }

    this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
    } catch (SQLException ex) {
        System.out.println(ex.getMessage());
    }

    this.getFachadaAplicacion().muestraExcepcion(ex.getMessage());
    }
} finally {
    try {
        stmBaja.close();
    } catch (SQLException e) {
        System.out.println("Imposible cerrar cursores");
    }
}
}
}

```

## 5.8 [FC\_01] Alta Participaciones

Podremos dar de alta las nuevas participaciones en la pestaña de “Operaciones” dentro de la ventana principal de empresa (4.4) seleccionando el número de ellas que queremos añadir y después, al pulsar el botón “Dar de Alta” llamaremos a la siguiente función “darAlta”. Las diferentes consultas se llevan a cabo en la tabla “tenerParticipaciones” (3.9) de la base de datos.

La consistencia en la base de datos se determina a través del uso del commit. Cuando la conexión con la base ha sido establecida se desactiva el auto-commit y se prepara la primera de las consultas. En ella se comprueba si la empresa posee participaciones de sí misma.

```

stmParticipacion = con.prepareStatement("select cantidad "
    + "from tenerparticipaciones where usuario = "
    + "empresa and empresa = ?");
stmParticipacion.setString(1, e);
rsParti = stmParticipacion.executeQuery();

```

En caso de que el número obtenido sea distinto de cero significa que la empresa posee participaciones de sí misma de las cuales es dueña, produciendo así la preparación de una actualización de la base de datos para sumar las nuevas participaciones a estas.

```

if (numParticipaciones > 0) {
    stmParticipacion = con.prepareStatement("update tenerparticipaciones "
        + "set cantidad = cantidad + ? "
        + "where usuario = empresa and empresa = ? ");
    stmParticipacion.setInt(1, num);
    stmParticipacion.setString(2, e);
    stmParticipacion.executeUpdate();
}

```

Si la cantidad obtenida en la primera consulta es cero implica una nueva inserción en la tabla “tenerParticipaciones” pues se crearán nuevas participaciones en las que la propia empresa será la propietaria.

```

else {
    stmParticipacion = con.prepareStatement("insert into tenerParticipaciones

```



```

(usuario, empresa, cantidad) values(?, ?, ?)");
stmParticipacion.setString(1, e);
stmParticipacion.setString(2, e);
stmParticipacion.setInt(3, num);
stmParticipacion.executeUpdate();
}

```

Una vez establecidas las consultas a realizar se comprometen usando el commit. Se precisa de un rollback en caso de que se produzca algún tipo de excepción relacionada con SQL. Finalmente, el auto-commit se reactiva cuando se ha finalizado la tarea en la base de datos.

En este caso accederemos a la tabla “tenerParticipaciones” (3.9) y, dependiendo de si tiene o no participaciones propias crearemos o modificaremos la tupla en la tabla.

## 5.9 [FC\_02] Baja Participaciones

De manera similar a la funcionalidad “Alta Participaciones” (5.8), accediendo a la ventana principal de empresa (4.4) y en este caso, seleccionando el número de participaciones de igual forma, pero pulsando el botón “Dar de baja” una empresa puede eliminar participaciones de su cartera. Al igual que en la sección anterior se modificará la tabla “tenerParticipaciones” (3.9).

Se asegura la consistencia en la base de datos al igual que en el caso de dar de alta diferentes participaciones, con el uso del commit. Se desactivará el auto-commit una vez se haya establecido la conexión con la base de datos. A continuación, se comprueba si la empresa es poseedora de participaciones de sí misma. Debemos tener en cuenta las participaciones que están en venta (3.8) y en stacking (3.11), para no contabilizarlas como participaciones disponibles para dar de baja.

```

stmParticipacion = con.prepareStatement("select cantidad -
(coalesce((select sum(cantidadparticipaciones) from mercado where usuario =
empresa and usuario = ?), 0) \n" +
        "+ coalesce((select sum(stacking.cantidad)
from stacking where usuario = empresa and usuario = ?),0))\n" +
        "as cantidad \n" +
        "from tenerparticipaciones \n" +
        "where usuario = empresa \n" +
        "and empresa = ?");
stmParticipacion.setString(1, e);
stmParticipacion.setString(2, e);
stmParticipacion.setString(3, e);
rsParti = stmParticipacion.executeQuery();

```

Si la cantidad obtenida es mayor o igual que las solicitadas para dar de baja se actualiza el número de participaciones disponibles de dicha empresa.

```

if (numParticipaciones >= num) {
    stmParticipacion = con.prepareStatement("update
tenerparticipaciones "
        + "set cantidad = cantidad - ? "
        + "where usuario = empresa and empresa = ?");
    stmParticipacion.setInt(1, num);
    stmParticipacion.setString(2, e);
    stmParticipacion.executeUpdate();
}

```

A continuación, se recalcula de nuevo las participaciones que la empresa posee de sí misma, ya que se puede dar el caso de que este ahora sea igual a cero y se pueda eliminar la tupla de la tabla “tenerParticipaciones”.

```

        stmParticipacion = con.prepareStatement("select cantidad "
            + "from tenerparticipaciones where usuario = "
empresa and empresa = ?");
        stmParticipacion.setString(1, e);
        rsParti = stmParticipacion.executeQuery();

        if (rsParti.next()) {
            numParticipaciones = rsParti.getInt("cantidad");
        }
        if (numParticipaciones <= 0) {
            stmParticipacion = con.prepareStatement("delete from
tenerparticipaciones "
                + "where usuario=empresa and empresa= ?");
            stmParticipacion.setString(1, e);
            stmParticipacion.executeUpdate();
        }

        con.commit();

```

Finalmente, se realizará un commit, produciendo que todas las consultas e actualizaciones anteriores pasen a un estado comprometido. En caso de que se produzca alguna excepción en la base de datos o de que la empresa no tenga la cantidad de participaciones indicadas para la baja, se realizará un rollback, pasando al estado anterior a la ejecución de las sentencias. Se reactivará el auto-commit al acabar con la funcionalidad.

## 5.10 [FC\_03] Comisión Compra/Venta

En la ventana del regulador (4.3), este podrá reajustar el valor de la comisión, y a continuación, al pulsar el botón “Guardar cambios”, llamaremos a la función “actualizarUsuario”, que en el caso de ser una instancia de regulador, como sucede aquí, accederemos a la tabla “regulador” (3.4), y cambiaremos los datos que hayan introducido en la ventana.

```

if (u instanceof Regulador) {
    Regulador r = (Regulador) u;
    stmActualizacion = con.prepareStatement("update regulador "
        + "set comision = ?, "
        + "    interes = ? "
        + "where usuario = ?");

    stmActualizacion.setFloat(1, r.getComisionCompraventa());
    stmActualizacion.setFloat(2, r.getInteresStacking());
    stmActualizacion.setString(3, idAntigua);
    stmActualizacion.executeUpdate();
}

```

Para obtener esta comisión a posteriori se accede a la misma tabla.

```

public float obtenerComisionActual() {
    float resultado = 0;
    Connection con;
    PreparedStatement stmComision = null;
    ResultSet rsComision;

```

```

        con = this.getConexion();
        try {
            stmComision = con.prepareStatement("select comision from
regulador");
            rsComision = stmComision.executeQuery();
            if (rsComision.next()) {
                resultado = rsComision.getFloat("comision");
            }
        } catch (SQLException e) {
            System.out.println(e.getMessage());
            this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
        } finally {
            try {
                stmComision.close();
            } catch (SQLException e) {
                System.out.println("Imposible cerrar cursores");
            }
        }

        return resultado;
    }

```

### 5.11 [FC\_04] Venta Participaciones

Para llevar a cabo una venta de Participaciones tendremos que situarnos en la pestaña de operaciones dentro de la ventana principal (4.4). Para ello es necesario seleccionar una empresa de la cartera de participaciones, a continuación, fijar un precio y la cantidad de participaciones que se quieren sacar a la venta. Una vez seleccionados estos parámetros haciendo clic sobre el botón “Vender” se llamará a la función correspondiente de hacer la consulta.

En dicha función lo primero que haremos será obtener la comisión establecida por el regulador. Para ello accederemos a la tabla “regulador” (3.4) y seleccionaremos la columna con el nombre de “comisión”.

```

con.setAutoCommit(false);

stm = con.prepareStatement("select comision from regulador");
rs = stm.executeQuery();
if (rs.next()) {
    comision = rs.getFloat("comision");
}

```

Cabe destacar el hecho de que no esté activado el auto-commit puesto que se realizaran varias consultas necesarias para completar esta funcionalidad.

Una vez se ha obtenido el valor para la comisión actual realizaremos una inserción en la tabla de la base de datos “mercado” (3.8) con los valores correspondientes al usuario actual, a la empresa a la cual pertenecen las participaciones, una marca temporal en el momento en el que se pone dichas participaciones en venta, la comisión que se lleva el regulador por la transacción, el número de participaciones a vender y para finalizar si precio. Como se puede ver en el siguiente fragmento de código:

```

stm = con.prepareStatement("insert into mercado values(?, ?,
now(), ?, ?, ?)");
stm.setString(1, usuario.getId());
stm.setString(2, empresa);
stm.setFloat(3, comision);

```

```
stm.setInt(4, num);
stm.setFloat(5, precio);
stm.executeUpdate();
```

Como fue mencionado con anterioridad la funcionalidad de auto-commit ha sido deshabilitada para esta función por lo tanto es necesario realizar un commit explícito para la base de datos. Al finalizar la funcionalidad se vuelve a activar el auto-commit.

## 5.12 [FC\_05] Baja Venta Participaciones

Esta funcionalidad se podrá llevar a cabo desde la pestaña “Operaciones” de la ventana principal de usuario (4.4). Para poder eliminar la oferta de venta de una participación, será necesario seleccionar dicha participación, la cual tiene que estar en el mercado, y hacer clic sobre el botón “Retirar del mercado”.

La implementación de esta funcionalidad es relativamente sencilla, puesto que solo es necesario eliminar una fila de una tabla, en concreto de la tabla “mercado” (3.8), que es donde se almacena la información sobre las participaciones disponibles a la venta. Se eliminará la entrada correspondiente al usuario que desea dar de baja la venta de la participación seleccionada en la cartera de participaciones. Para ello utilizaremos el siguiente fragmento de código:

```
stm = con.prepareStatement("delete from mercado where usuario = ? and
marcatemporal = ?");
stm.setString(1, idUsuario);
stm.setTimestamp(2, marcaTemporal);
stm.executeUpdate();
```

## 5.13 [FC\_06] Compra Participaciones

Como su nombre indica, esta funcionalidad permite la compra de las participaciones que previamente hayan sido puestas a la venta por diferentes usuarios. Para poder llevar a cabo la compra tendremos que acceder a la ventana de mercado (4.5) haciendo clic en el botón mercado desde la pestaña “Operaciones” de la ventana principal del usuario (4.4). En esta ventana se seleccionará la empresa de la cual son las participaciones que se quieren comprar, la cantidad que se desea comprar y el importe máximo por participación dispuesto a pagar.

Es importante destacar que a lo largo de esta funcionalidad se ejecutaran varias consultas a la base de datos, por ello el auto-commit estará deshabilitado y se ejecutara un commit manual al final de la funcionalidad.

Lo primero que debemos comprobar una vez introducidos los datos es comprobar que exista un número suficiente de participaciones a la venta por debajo del precio establecido, para ello utilizaremos la siguiente consulta:

```
// Se obtiene la cantidad de participaciones disponibles por debajo del
precio indicado
stm = con.prepareStatement("select sum(cantidadparticipaciones)
as numVenta from mercado "
+ "where usuario != ? and empresa = ? and
precioparticipacion <= ?");
stm.setString(1, usuario);
stm.setString(2, empresa);
stm.setFloat(3, precio);
```

```
rs = stm.executeQuery();

if (rs.next()) {
    numVenta = rs.getInt("numVenta");
```

Una vez comprobada que es posible la compra, vamos a ordenar las participaciones que están disponibles para su compra según su precio para poder empezar comprando las más baratas y acabar con las más caras. Esto será posible gracias a esta consulta sobre la tabla “mercado” (3.8):

```
// Ordena de menor a mayor precio las participaciones a la venta de dicha
// empresa por debajo del precio indicado
stm = con.prepareStatement("select * from mercado where
usuario != ? and empresa = ? "
+ "and precioparticipacion <= ? order by
precioParticipacion, marcaTemporal");
stm.setString(1, usuario);
stm.setString(2, empresa);
stm.setFloat(3, precio);
rs = stm.executeQuery()
```

A continuación, mediante la utilización de un bucle while recorreremos el ResultSet hasta que se hayan comprado el número total de participaciones especificadas. Dentro del bucle, nada más comienza su ejecución nos encontramos con un if, el cual nos permite realizar dos consultas diferentes en función de cuantas participaciones se vayan a retirar de la tabla “mercado” (3.8) sobre la fila seleccionada. En caso de que dicha fila se quede con las participaciones a 0, se elimina; y se permanece con más, simplemente se actualiza al número correspondiente. Esto se puede ver en el siguiente fragmento de código:

```
while (rs.next() && count < num) {
    if ((num - count) <
rs.getInt("cantidadparticipaciones")) {
        stm = con.prepareStatement("update mercado set
cantidadparticipaciones = ? where usuario = ? and empresa = ? and
marcatemporal = ?");
        stm.setInt(1, (rs.getInt("cantidadparticipaciones")
- (num - count)));
        stm.setString(2, rs.getString("usuario"));
        stm.setString(3, rs.getString("empresa"));
        stm.setTimestamp(4,
rs.getTimestamp("marcatemporal"));
        stm.executeUpdate();
        cantidadActual = (num - count);
    } else {
        stm = con.prepareStatement("delete from mercado
where usuario = ? and empresa = ? and marcatemporal = ?");
        stm.setString(1, rs.getString("usuario"));
        stm.setString(2, rs.getString("empresa"));
        stm.setTimestamp(3,
rs.getTimestamp("marcatemporal"));
        stm.executeUpdate();
        cantidadActual =
rs.getInt("cantidadparticipaciones");
    }
}
```

A continuación es necesario eliminar las participaciones del usuario que decide ponerlas en venta, para ello accederemos a la tabla “tenerParticipaciones” (3.9) en la cual se almacena la información relativa a que participaciones tiene cada usuario. Para ello primero modificaremos el número de participaciones del usuario que las vende:

```

        // Modificamos las participaciones de la empresa
        stm = con.prepareStatement("update tenerparticipaciones
set cantidad = cantidad - ? where usuario = ? and empresa = ? ");
        stm.setInt(1, cantidadActual);
        stm.setString(2, rs.getString("usuario"));
        stm.setString(3, rs.getString("empresa"));

```

El siguiente paso que debemos llevar a cabo es modificar los saldos de ambos usuarios que se ven implicados en esta transacción y el regulador que se llevará una comisión por la misma. Lo primero que hacemos es calcular los diferentes valores que utilizaremos para actualizar la tabla “usuario” (3.1), donde se encuentra el atributo saldoDisponible. A continuación se ejecutarán dos consultas para actualizar el saldo del vendedor y del regulador. Esto se puede ver a continuación:

```

// Se modifican los saldos del vendedor y del regulador
float dineroRegulador = (float) (cantidadActual *
rs.getFloat("precioparticipacion")
* ((rs.getFloat("comisionventa") / 100)));
dineroComprador += (float) (cantidadActual *
rs.getFloat("precioparticipacion"));
float dineroVendedor = (float) ((cantidadActual *
rs.getFloat("precioparticipacion")) - dineroRegulador);

stm = con.prepareStatement("update usuario set
saldodisponible = saldodisponible + ? where id = ?");
stm.setFloat(1, dineroVendedor);
stm.setString(2, rs.getString("usuario"));
stm.executeUpdate();

stm = con.prepareStatement("update usuario set
saldodisponible = saldodisponible + ? where tipousuario = 'Regulador'");
stm.setFloat(1, dineroRegulador);
stm.executeUpdate();
}

```

Una vez han sido eliminados y actualizadas las filas de la tabla “mercado” (3.8) correspondientes a las participaciones compradas, se hará una comprobación de que el usuario tenga participaciones previas de la empresa seleccionada, y según esto se ejecutarán dos consultas diferentes, una en la que se crea una nueva fila en la tabla “tenerParticipaciones” (3.9), u otra en la cual se actualiza una fila ya existente. El código sería:

```

// Comprobación de participaciones de la empresa por parte del usuario.
stm = con.prepareStatement("select exists(select * from
tenerparticipaciones where usuario = ? and empresa = ?) as result");
stm.setString(1, usuario);
stm.setString(2, empresa);

rs = stm.executeQuery();
if (rs.next()) {
    if (rs.getBoolean("result")) {
        //El usuario posee participaciones de la empresa,
se actualiza su número
        stm = con.prepareStatement("update
tenerparticipaciones set cantidad = cantidad + ? where usuario = ? and
empresa = ?");

        stm.setInt(1, num);
        stm.setString(2, usuario);
        stm.setString(3, empresa);
        stm.executeUpdate();
    } else {
        //Si no tiene participaciones de la empresa se

```

insertan en la tabla

```
stm = con.prepareStatement("insert into
tenerparticipaciones values(?, ?, ?)");
stm.setString(1, usuario);
stm.setString(2, empresa);
stm.setInt(3, num);
stm.executeUpdate();
}
}
```

Para finalizar es necesario comprobar que el usuario dispone del saldo suficiente para efectuar la compra deseada. Para ello se realiza una consulta simple para obtener el saldo disponible del comprador y se compara con el valor calculado acumulado de la compra. Esta comprobación dará lugar a una bifurcación: en caso de que el saldo no sea suficiente, se mostrará un mensaje de error y se realizará un rollback, por lo cual ninguna de las sentencias anteriores habrá tenido lugar; en caso de que este saldo sí sea suficiente se actualizará al nuevo mediante una simple consulta sobre la tabla “usuario” (3.1) y se realizará el commit de finalización. El código es el siguiente:

```
// Por último comprobamos si la compra es posible por el saldo del usuario
stm = con.prepareStatement("select saldodisponible from
usuario where id = ?");
stm.setString(1, usuario);
rs = stm.executeQuery();
if (rs.next()) {
    if (rs.getFloat("saldodisponible") >= dineroComprador)
    {
        // Se actualiza el mismo
        stm = con.prepareStatement("update usuario set
saldodisponible = saldodisponible - ? where id = ?");
        stm.setFloat(1, dineroComprador);
        stm.setString(2, usuario);
        stm.executeUpdate();

        con.commit();
    } else {
        this.getFachadaAplicacion().muestraExcepcion("No
tienes saldo suficiente para realizar la compra");
        resultado = false;
        con.rollback();
    }
}
```

## 5.14 [FC-07] Alta Pago Beneficios

El alta de pago de beneficios tiene lugar en la ventana de gestión de pagos (4.6). En esta pestaña la empresa seleccionará la fecha en la cual se realizará el pago y la preferencia de este, rellenando los campos correspondientes a dicha preferencia (o saldo o participaciones o ambas). Al pulsar el botón “Anunciar”, se iniciará la ejecución de la funcionalidad.

Como la funcionalidad requiere de más de una sentencia SQL, se desactiva el auto-commit al inicio de esta y no se volverá a activar hasta que finalice, bien sea correctamente o no. Después de que la última sentencia se ejecute correctamente, se realizará el commit. En caso de que salte alguna excepción SQL o de que la empresa no tenga saldo y/o participaciones suficientes, se realizará un rollback.

Esta funcionalidad tiene que comprobar que la empresa que realiza el anuncio tiene saldo y/o participaciones suficientes para hacer frente al bloque. Para ello se obtiene el número de

participaciones totales que otros usuarios tienen de la empresa y cuántos usuarios tienen participaciones de la empresa mediante la información de la tabla “tenerParticipaciones” (3.9) usando el siguiente fragmento de código:

```
// Se obtiene el número de participaciones que otros usuarios
tienen de la empresa y cuántos usuarios tienen participaciones de la misma
stmAnuncio = con.prepareStatement("select sum(cantidad) as
cantidadTotal, count(*) as usuariosConParticipaciones "
+ "from tenerParticipaciones "
+ "where usuario != empresa and empresa = ?");
stmAnuncio.setString(1, pb.getIdEmpresa());
rsAnuncio = stmAnuncio.executeQuery();
if (rsAnuncio.next()) {
    cantidadTotalPart = rsAnuncio.getInt("cantidadTotal");
    usuariosConParticipaciones =
rsAnuncio.getInt("usuariosConParticipaciones");
}
```

Una vez obtenida esta información, se usará en la inserción del anuncio en la tabla “informacionBeneficios” (3.10) de la base de datos:

```
stmAnuncio = con.prepareStatement("insert into
informacionBeneficios values(?,?,?,?,?,?)");
stmAnuncio.setString(1, pb.getIdEmpresa());
stmAnuncio.setTimestamp(2, pb.getFecha());
stmAnuncio.setFloat(3, pb.getImporteParticipacion());
stmAnuncio.setInt(4, cantidadTotalPart);
stmAnuncio.setInt(5, pb.getParticipacionesPorUsuario());
stmAnuncio.setInt(6, usuariosConParticipaciones);
stmAnuncio.executeUpdate();
```

Tras la inserción se obtienen el saldo disponible y las participaciones propias de la empresa, para saber si es posible publicar el anuncio, ya que estos dos valores no pueden sobrepasar los consecuentes de lo especificado en el anuncio. Esta información está disponible en las tablas “usuario” (3.1) y “tenerParticipaciones” (3.9). Se obtiene de la siguiente forma:

```
// Se obtiene el saldo disponible de la empresa
stmAnuncio = con.prepareStatement("select saldoDisponible from
usuario where id = ?");
stmAnuncio.setString(1, pb.getIdEmpresa());
rsAnuncio = stmAnuncio.executeQuery();
if (rsAnuncio.next()) {
    saldoDisponible = rsAnuncio.getFloat("saldoDisponible");
}

// Se obtienen las participaciones propias de la empresa
stmAnuncio = con.prepareStatement("select cantidad from
tenerParticipaciones where usuario = empresa and empresa = ?");
stmAnuncio.setString(1, pb.getIdEmpresa());
rsAnuncio = stmAnuncio.executeQuery();
if (rsAnuncio.next()) {
    participacionesDisponibles = rsAnuncio.getInt("cantidad");
}
```

Ahora se ejecutaría lo que realmente implicaría la publicación o no del anuncio: la comprobación de si el saldo disponible es menor o igual que el importe por participación por el número de participaciones que otros usuarios tienen de la empresa, y de si las participaciones disponibles son menores o iguales que el número de participaciones por usuario por el número de usuarios que tienen



participaciones de la empresa. Si esta comprobación se cumple, se pasa a bloquear a la empresa el saldo y las participaciones correspondientes cambiando el valor de los atributos correspondientes en la tabla “empresa” (3.2), así como disminuir el saldo disponible cambiando el valor del atributo correspondiente en la tabla “usuario” (3.1), ya que ahora hay más saldo bloqueado. Al finalizar esta labor correctamente, se realiza el commit de la transacción. Si la comprobación anterior no se cumple, se realiza un rollback. Se inserta aquí el código correspondiente:

```

        // Si la empresa tiene saldo disponible suficiente y
        // participaciones suficientes...
        if (((pb.getImporteParticipacion() * cantidadTotalPart) <=
saldoDisponible)
            && ((pb.getParticipacionesPorUsuario() *
usuariosConParticipaciones) <= participacionesDisponibles)) {
            // Se actualizan el saldo bloqueado y las participaciones
            // bloqueadas
            stmAnuncio = con.prepareStatement("update empresa set
saldoBloqueado = saldoBloqueado + ?, "
            + "partiBloqueadas = partiBloqueadas + ? "
            + "where usuario = ?");
            stmAnuncio.setFloat(1, pb.getImporteParticipacion() *
cantidadTotalPart);
            stmAnuncio.setInt(2, pb.getParticipacionesPorUsuario() *
usuariosConParticipaciones);
            stmAnuncio.setString(3, pb.getIdEmpresa());
            stmAnuncio.executeUpdate();

            // Se actualizan las participaciones disponibles
            stmAnuncio = con.prepareStatement("update
tenerparticipaciones set cantidad = cantidad - ? "
            + "where usuario = empresa and empresa = ?");
            stmAnuncio.setInt(1, pb.getParticipacionesPorUsuario() *
usuariosConParticipaciones);
            stmAnuncio.setString(2, pb.getIdEmpresa());

            // Se actualiza el saldo disponible
            stmAnuncio = con.prepareStatement("update usuario set
saldoDisponible = saldoDisponible - ? "
            + "where id = ?");
            stmAnuncio.setFloat(1, pb.getImporteParticipacion() *
cantidadTotalPart);
            stmAnuncio.setString(2, pb.getIdEmpresa());
            stmAnuncio.executeUpdate();
            resultado = true;
            con.commit();
        } else {
            this.getFachadaAplicacion().muestraExcepcion("La empresa no
tiene saldo disponible suficiente o participaciones suficientes\n"
            + "para publicar el anuncio");
            con.rollback();
        }
    }

```

Nota: la variable “pb” es una instancia de la clase PagoBeneficios, que actúa como contenedor de datos con todos los datos correspondientes a la entrada de una tupla de la tabla “informacionBeneficios” (3.10) de la base de datos.

### 5.15 [FC\_08] Baja Pago Beneficios

La baja de pago de beneficios tiene lugar en la pestaña “Pagos” de la ventana del regulador (4.3). En esta pestaña el regulador del mercado tiene acceso a los diferentes anuncios de pagos de beneficios que están activos de las diferentes empresas en una tabla. El regulador selecciona un anuncio concreto y presiona el botón “Dar de baja anuncio”, momento en el cual se inicia esta funcionalidad.

En primer lugar, se desbloquean el saldo y las participaciones correspondientes de la empresa en la tabla “empresa” (3.2) mediante el siguiente trozo de código:

```
// Se desbloquean el saldo y las participaciones
stmBorrado = con.prepareStatement("update empresa set
saldoBloqueado = saldoBloqueado - ?, "
+ "partiBloqueadas = partiBloqueadas - ? "
+ "where usuario = ?");
stmBorrado.setFloat(1, pb.getImporteParticipacion() *
pb.getCantidadParticipaciones());
stmBorrado.setInt(2, pb.getParticipacionesPorUsuario() *
pb.getUsuariosConParticipaciones());
stmBorrado.setString(3, pb.getIdEmpresa());
stmBorrado.executeUpdate();
```

Nota: la variable “pb” es una instancia de la clase PagoBeneficios, que actúa como contenedor de datos con todos los datos correspondientes a la entrada de una tupla de la tabla “informacionBeneficios” (3.10) de la base de datos.

A continuación, hay que añadir el saldo desbloqueado con anterioridad al saldo disponible de la empresa en la tabla “usuario” (3.1):

```
// Se aumenta el saldo disponible
stmBorrado = con.prepareStatement("update usuario set
saldoDisponible = saldoDisponible + ? "
+ "where id = ?");
stmBorrado.setFloat(1, pb.getImporteParticipacion() *
pb.getCantidadParticipaciones());
stmBorrado.setString(2, pb.getIdEmpresa());
stmBorrado.executeUpdate();
```

Para finalizar, se borra el anuncio de la tabla “informacionBeneficios” (3.10):

```
// Se borra el anuncio
stmBorrado = con.prepareStatement("delete from
informacionbeneficios "
+ "where empresa = ? and fechapago = ?");
stmBorrado.setString(1, pb.getIdEmpresa());
stmBorrado.setTimestamp(2, pb.getFecha());
stmBorrado.executeUpdate();
```

Como la funcionalidad requiere de más de una sentencia SQL, se desactiva el auto-commit al inicio de esta y no se volverá a activar hasta que finalice, bien sea correctamente o no. Después de que la última sentencia se ejecute correctamente, se realizará el commit. En caso de que salte alguna excepción SQL, se realizará un rollback.

### 5.16 [FC\_09] Pago Beneficios

El pago de beneficios tiene una ventana propia para su gestión (4.6). Esta funcionalidad tiene dos casos de ejecución bien diferenciados: por un lado, si la empresa tiene un anuncio previo, se darán

como beneficio el importe y/o participaciones bloqueadas en el momento del anuncio; por el otro, si la empresa no tiene anuncio previo, se darán como beneficio los indicados el importe y participaciones indicados a todos los usuarios con participaciones de la empresa (el importe dependerá número de participaciones de estos posean).

Primero se describirá el desarrollo de la funcionalidad para el primer caso: la empresa tiene un anuncio previo. Para saber si la ejecución tiene que tomar este camino se consulta en la tabla “informacionBeneficios” (3.10) si la empresa tiene algún anuncio.

```
stmPago = con.prepareStatement("select * from
informacionBeneficios where empresa = ?");
stmPago.setString(1, pb.getIdEmpresa());
rsPago = stmPago.executeQuery();

// Si existe un anuncio de la empresa, se repartirán los
beneficios bloqueados
if (rsPago.next()) {
```

Si el ResultSet tiene algún resultado, se meterá en el if, que es el que engloba todo este primer caso. A continuación, se crea una instancia de PagoBeneficios “pbAnuncio” con la información de la tupla encontrada. Como es posible que entre la publicación del anuncio y el pago de este haya aumentado el número de participaciones que otros usuarios tienen de la empresa, este se obtiene de nuevo de la tabla “tenerParticipaciones” (3.9). A partir de ese dato, se divide el total del saldo bloqueado del anuncio (en caso de haber aumentado el número de participaciones, el importe por usuario disminuirá). Después se procede a la repartición de este importe entre todos los usuarios multiplicado por el número de participaciones de la empresa que poseen. Para la repartición se usan las tablas “usuario” (3.1) y “tenerParticipaciones” (3.9).

```
// Se obtiene el número de participaciones que otros
usuarios tienen de la empresa
stmPago = con.prepareStatement("select sum(cantidad) as
cantidadTotal "
+ "from tenerParticipaciones "
+ "where usuario != empresa and empresa = ?");
stmPago.setString(1, pb.getIdEmpresa());
rsPago = stmPago.executeQuery();
if (rsPago.next()) {
    cantidadTotalPart = rsPago.getInt("cantidadTotal");
}

// Se divide la cantidad total entre el saldo bloqueado
para obtener el importe por participación real
nuevoImportePorPart = ((pbAnuncio.getImporteParticipacion())
* pbAnuncio.getCantidadParticipaciones()) / cantidadTotalPart);

// Se reparte eso entre los usuarios con participaciones
stmPago = con.prepareStatement("select * "
+ "from tenerParticipaciones as tp, usuario as u "
+ "where tp.usuario != tp.empresa and tp.usuario =
u.id "
+ "and tp.empresa = ?");
stmPago.setString(1, pb.getIdEmpresa());
rsPago = stmPago.executeQuery();
while (rsPago.next()) {
    stmPago = con.prepareStatement("update usuario set
saldoDisponible = saldoDisponible + ? "
```

```

        + "where id = ?");
        stmPago.setFloat(1, nuevoImportePorPart *
rsPago.getInt("cantidad"));
        stmPago.setString(2, rsPago.getString("id"));
        stmPago.executeUpdate();
    }

```

Para la repartición de participaciones, como también puede haber más usuarios con participaciones de la empresa que en el momento del anuncio, éstas se repartirán de mayor a menor cantidad de participaciones que los usuarios posean de dicha empresa. De esta manera, en caso de que no lleguen las participaciones bloqueadas totales para completar el reparto, serán beneficiados los usuarios que tienen más participaciones de la empresa. Esta ordenación de mayor a menor cantidad de participaciones, al igual que la repartición, tienen lugar en la tabla “tenerParticipaciones” (3.9).

```

        // Las participaciones se darán de mayor a menor cantidad
        de participaciones de la empresa
        stmPago = con.prepareStatement("select * from
tenerparticipaciones "
            + "where usuario != empresa and empresa = ? "
            + "order by cantidad desc");
        stmPago.setString(1, pbAnuncio.getIdEmpresa());
        rsPago = stmPago.executeQuery();
        participacionesPendientes =
pbAnuncio.getParticipacionesPorUsuario() *
pbAnuncio.getUsuariosConParticipaciones();
        while (rsPago.next() && (participacionesPendientes > 0)) {
            stmPago = con.prepareStatement("update
tenerparticipaciones set cantidad = cantidad + ? "
                + "where usuario = ? and empresa = ?");
            stmPago.setInt(1,
pbAnuncio.getParticipacionesPorUsuario());
            stmPago.setString(2, rsPago.getString("usuario"));
            stmPago.setString(3, pbAnuncio.getIdEmpresa());
            stmPago.executeUpdate();
            participacionesPendientes -=
pbAnuncio.getParticipacionesPorUsuario();
        }

```

Para finalizar, se actualizan el saldo y las participaciones bloqueados de la empresa en la tabla “empresa” (3.2). Como el saldo fue bloqueado anteriormente (y por tanto restado del saldo disponible), no hay que actualizar el saldo disponible.

```

// Se actualiza el saldo bloqueado y las participaciones bloqueadas de la
empresa
        stmPago = con.prepareStatement("update empresa set
saldoBloqueado = saldoBloqueado - ?, "
            + "partiBloqueadas = partiBloqueadas - ? "
            + "where usuario = ?");
        stmPago.setFloat(1, pbAnuncio.getImporteParticipacion() *
pbAnuncio.getCantidadParticipaciones());
        stmPago.setInt(2, pbAnuncio.getParticipacionesPorUsuario()
* pbAnuncio.getUsuariosConParticipaciones());
        stmPago.setString(3, pb.getIdEmpresa());
        stmPago.executeUpdate();

        resultado = true;
        con.commit();

```

El segundo caso de desarrollo de la funcionalidad es aquel en el cual la empresa no tiene anuncio previo. Este caso tiene similitudes con el anterior. Para empezar, se empieza obteniendo la cantidad de usuarios con participaciones de la empresa y la cantidad total de participaciones que otros usuarios tienen de ésta. También se obtienen las participaciones propias de la empresa y el saldo disponible para saber si puede hacer frente al pago. Esta información está en las tablas “usuario” (3.1) y “tenerParticipaciones” (3.9).

```
// Se obtiene el número de participaciones que otros usuarios tienen de la
// empresa y cuántos usuarios tienen participaciones de la misma
stmPago = con.prepareStatement("select sum(cantidad) as
cantidadTotal, count(*) as usuariosConParticipaciones "
+ "from tenerParticipaciones "
+ "where usuario != empresa and empresa = ?");
stmPago.setString(1, pb.getIdEmpresa());
rsPago = stmPago.executeQuery();
if (rsPago.next()) {
    cantidadTotalPart = rsPago.getInt("cantidadTotal");
    usuariosConParticipaciones =
rsPago.getInt("usuariosConParticipaciones");
}

// Se obtiene el saldo disponible de la empresa
stmPago = con.prepareStatement("select saldoDisponible from
usuario where id = ?");
stmPago.setString(1, pb.getIdEmpresa());
rsPago = stmPago.executeQuery();
if (rsPago.next()) {
    saldoDisponible = rsPago.getFloat("saldoDisponible");
}

// Se obtienen las participaciones propias de la empresa
stmPago = con.prepareStatement("select cantidad from
tenerParticipaciones where usuario = empresa and empresa = ?");
stmPago.setString(1, pb.getIdEmpresa());
rsPago = stmPago.executeQuery();
if (rsPago.next()) {
    participacionesDisponibles = rsPago.getInt("cantidad");
}
```

A continuación tiene lugar la comprobación de si la empresa tiene saldo disponible y/o participaciones suficientes para hacer frente al pago. De ser así, se reparte el importe y/o las participaciones entre los diferentes usuarios que tengan participaciones de la empresa. El importe dependerá de la cantidad de participaciones que cada usuario tiene de ésta. Esta repartición tiene lugar en usando las tablas “usuario” (3.1) y “tenerParticipaciones” (3.9).

```
// Si la empresa tiene saldo disponible suficiente y participaciones
// suficientes...
if (((cantidadTotalPart * pb.getImporteParticipacion()) <=
saldoDisponible)
    && ((pb.getParticipacionesPorUsuario() *
usuariosConParticipaciones) <= participacionesDisponibles)) {

    // Repartición de importe
    stmPago = con.prepareStatement("select * "
+ "from tenerParticipaciones as tp, usuario as
```

```

u "
                                + "where tp.usuario != tp.empresa and
tp.usuario = u.id "
                                + "and tp.empresa = ?");
stmPago.setString(1, pb.getIdEmpresa());
rsPago = stmPago.executeQuery();
while (rsPago.next()) {
    stmPago = con.prepareStatement("update usuario set
saldoDisponible = saldoDisponible + ? "
                                + "where id = ?");
    stmPago.setFloat(1, rsPago.getInt("cantidad") *
pb.getImporteParticipacion());
    stmPago.setString(2, rsPago.getString("id"));
    stmPago.executeUpdate();
}

// Repartición de participaciones
stmPago = con.prepareStatement("update
tenerParticipaciones set cantidad = cantidad + ? "
                                + "where usuario != empresa and empresa = ?");
stmPago.setInt(1, pb.getParticipacionesPorUsuario());
stmPago.setString(2, pb.getIdEmpresa());
stmPago.executeUpdate();

```

Para finalizar, se restan el saldo y/o participaciones empleados a la empresa en las tablas “usuario” (3.1) y “tenerParticipaciones” (3.9).

```

// Se actualiza el saldo de la empresa
stmPago = con.prepareStatement("update usuario set
saldoDisponible = saldoDisponible - ? "
                                + "where id = ?");
stmPago.setFloat(1, pb.getImporteParticipacion() *
cantidadTotalPart);
stmPago.setString(2, pb.getIdEmpresa());
stmPago.executeUpdate();

// Se actualizan las participaciones de la empresa
stmPago = con.prepareStatement("update
tenerParticipaciones set cantidad = cantidad - ? "
                                + "where usuario = empresa and empresa = ?");
stmPago.setInt(1, pb.getParticipacionesPorUsuario() *
usuariosConParticipaciones);
stmPago.setString(2, pb.getIdEmpresa());
stmPago.executeUpdate();

resultado = true;
con.commit();

```

En caso de que la empresa no pueda hacer frente al pago de beneficios, ninguna de las sentencias SQL habrá tenido lugar ya que se ejecutará un rollback.

```

else {
    this.getFachadaAplicacion().muestraExcepcion("La empresa no
tiene saldo disponible suficiente o participaciones suficientes\n"
                                + "para publicar el anuncio");
    con.rollback();
}

```

Como la funcionalidad requiere de más de una sentencia SQL, se desactiva el auto-commit al inicio de esta y no se volverá a activar hasta que finalice, bien sea correctamente o no. Después de que la última sentencia se ejecute correctamente, se realizará el commit. En caso de que salte alguna excepción SQL o de que la empresa no sea capaz de hacer frente al pago de beneficios por falta de saldo disponible o participaciones, se realizará un rollback.

## 5.17 [FC\_10] Stacking

Esta funcionalidad se desarrolla en su totalidad en una ventana propia (4.8). En ella, se ofrece la posibilidad de realizar un “apilamiento” o de retirarlo.

Comencemos tratando la creación/depósito. Para ello el usuario debe seleccionar que participaciones quiere “apilar” y la cantidad, una vez se comprueba que la cifra es coherente se realiza la siguiente transacción:

```
public void insertarStacking(Stacked s) {
    Connection con;
    PreparedStatement stmStac = null;
    ResultSet rsStac;
    float interes = 0;
    int cantidadAntigua = 0;

    con = this.getConexion();
    try {
        con.setAutoCommit(false);

        //Primero obtenemos el interes vigente en ese momento
        stmStac = con.prepareStatement("select interes from
regulador");
        rsStac = stmStac.executeQuery();
        if (rsStac.next()) {
            interes = rsStac.getFloat("interes");
        }

        //Introducimos en la tabla stacking
        stmStac = con.prepareStatement("insert into stacking
values(?,?,?, ?,?)");
        stmStac.setTimestamp(1, s.getMarcaTemporal());
        stmStac.setString(2, s.getUsuario());
        stmStac.setString(3, s.getIdEmpresa());
        stmStac.setInt(4, s.getCantidad());
        stmStac.setFloat(5, interes);

        stmStac.executeUpdate();

        //Primero obtenemos las participaciones que tenía antes
        stmStac = con.prepareStatement("select cantidad from
tenerParticipaciones where usuario = ? and empresa = ?");
        stmStac.setString(1, s.getUsuario());
        stmStac.setString(2, s.getIdEmpresa());
        rsStac = stmStac.executeQuery();
        if (rsStac.next()) {
            cantidadAntigua = rsStac.getInt("cantidad");
        }

        //Retiramos las participaciones correspondientes
        stmStac = con.prepareStatement("update tenerParticipaciones set
```

```

cantidad = ? where usuario = ? and empresa = ?");
stmStac.setInt(1, cantidadAntigua - s.getCantidad());
stmStac.setString(2, s.getUsuario());
stmStac.setString(3, s.getIdEmpresa());
stmStac.executeUpdate();

con.commit();
} catch (SQLException e) {
    try {
        con.rollback();
        System.out.println(e.getMessage());

this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
    } catch (SQLException ex) {
        System.out.println(ex.getMessage());

this.getFachadaAplicacion().muestraExcepcion(ex.getMessage());
    }
} finally {
    try {
        con.setAutoCommit(true);
        stmStac.close();
    } catch (SQLException e) {
        System.out.println(e.getMessage());

this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
    }
}
}
}

```

El primer paso es obtener el interés vigente en ese momento. Una vez hecho esto, se introduce en la tabla “stacking” (3.11) la información con el “apilamiento” que se está realizando. Para terminar se procede a retirar las participaciones correspondientes de la cartera de participaciones del usuario, disponibles en la tabla “tenerParticipaciones” (3.9).

A continuación, se muestra la transacción correspondiente al retiro de un “apilamiento”:

```

public float retirarStacking(Stacked s) {
    Connection con;
    PreparedStatement stmStac = null;
    ResultSet rsStac;
    float precio = 1;
    int dias = 0;
    int cantidadAntigua = 0;
    float res = 0;

    con = this.getConexion();
    try {
        con.setAutoCommit(false);

        //Primero calculamos los dias que han pasado
        stmStac = con.prepareStatement("select (cast(current_timestamp
as date) - cast(? as date)) as resultado");
        stmStac.setTimestamp(1, s.getMarcaTemporal());

        rsStac = stmStac.executeQuery();
        if (rsStac.next()) {
            dias = rsStac.getInt("resultado");
        }
    }
}

```



```

        //Luego borramos de la tabla de stacking
        stmStac = con.prepareStatement("delete from stacking where
marcaTemporal = ? and usuario = ? and empresa = ?");
        stmStac.setTimestamp(1, s.getMarcaTemporal());
        stmStac.setString(2, s.getUsuario());
        stmStac.setString(3, s.getIdEmpresa());

        stmStac.executeUpdate();

        //Introducimos las participaciones correspondientes
        //Primero comprobamos si ya tiene mas participaciones como esas
        stmStac = con.prepareStatement("select cantidad from
tenerParticipaciones where usuario = ? and empresa = ?");
        stmStac.setString(1, s.getUsuario());
        stmStac.setString(2, s.getIdEmpresa());

        rsStac = stmStac.executeQuery();
        if (rsStac.next()) {
            if (!rsStac.isNull()) {
                cantidadAntigua = rsStac.getInt("cantidad");
                //Una vez sabemos la cantidad antigua actualizamos
                stmStac = con.prepareStatement("update
tenerParticipaciones set cantidad = ? where usuario = ? and empresa = ?");
                stmStac.setInt(1, s.getCantidad() + cantidadAntigua);
                stmStac.setString(2, s.getUsuario());
                stmStac.setString(3, s.getIdEmpresa());
            } else { //Si no tenia participaciones se inserta un nuevo
registro
                stmStac = con.prepareStatement("insert into
tenerParticipaciones values(?,?,?)");
                stmStac.setString(1, s.getUsuario());
                stmStac.setString(2, s.getIdEmpresa());
                stmStac.setInt(3, s.getCantidad());
            }
        }
        stmStac.executeUpdate();
        //Queda calcular el precio sobre el que se aplicará el interés
        (Se coge el mas barato)
        stmStac = con.prepareStatement("select precioparticipacion "
+ "from mercado, tenerParticipaciones "
+ "where mercado.usuario = tenerParticipaciones.usuario
"
+ "and mercado.empresa = ? "
+ "order by precioparticipacion asc");
        stmStac.setString(1, s.getIdEmpresa());

        rsStac = stmStac.executeQuery();
        if (rsStac.next()) {
            precio = rsStac.getFloat("precioparticipacion");
        }

        //Por ultimo calculamos la ganancia
        res = precio * s.getInteres() * dias;

        con.commit();
    } catch (SQLException e) {
        try {
            con.rollback();
            System.out.println(e.getMessage());
        }

        this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
    }
}

```

```

    } catch (SQLException ex) {
        System.out.println(ex.getMessage());
    }
    this.getFachadaAplicacion().muestraExcepcion(ex.getMessage());
}
} finally {
    try {
        con.setAutoCommit(true);
        stmStac.close();
    } catch (SQLException e) {
        System.out.println(e.getMessage());
    }
    this.getFachadaAplicacion().muestraExcepcion(e.getMessage());
}
}
return res;
}

```

Primeramente, se comprueban los días que han pasado desde que se realizó mediante una resta de fechas. Luego se procede a eliminar el registro correspondiente en la tabla “stacking” (3.11). A continuación, se devuelven las participaciones a la cartera del usuario en la tabla “tenerParticipaciones” (3.9). En caso de que no tuviera más como esas se hace mediante un *insert* y, en caso de que sí que las tuviera, mediante un *update*. Por último, queda calcular el beneficio obtenido por el usuario mediante el “apilamiento” y para ello se obtiene el menor precio del mercado para las participaciones de esa empresa, y se multiplica por el interés. Para este cálculo se usan las tablas “mercado” (3.8) y “tenerParticipaciones” (3.9).

Hay que destacar que debido a que las transacciones requieren más de una sentencia SQL se desactiva el auto-commit hasta la finalización de la transacción para preservar la atomicidad. Si ocurre alguna excepción SQL, se realizará un rollback; si todo va bien, un commit al final.

## 5.18 [FC\_11] Contratos de Futuros

Esta funcionalidad cuenta con su propia ventana de gestión (4.7), en ella se pueden visualizar las diferentes ofertas que hay disponibles y aceptarlas. En caso de querer hacerlo, y también se puede realizar tu propia oferta. La transacción de creación de una oferta de pacto requiere de una única sentencia *insert* en la tabla “ofertaPacto” (3.12) que sería la siguiente:

```

stmPacto = con.prepareStatement("insert into ofertaPacto
values(?,?,?,?,?,?,?)");
stmPacto.setTimestamp(1, p.getMarcaTemporal());
stmPacto.setString(2, p.getComprador());
stmPacto.setString(3, p.getIdEmpresa());
stmPacto.setInt(4, p.getCantidad());
stmPacto.setFloat(5, p.getPrecio());
stmPacto.setTimestamp(6, p.getFechaVenta());
stmPacto.setFloat(7, comision);

```

Y también de una consulta previa al *insert* que comprueba la comisión vigente en el momento, comentada en la funcionalidad FC\_03 (5.10). Al requerir de más de una sentencia SQL se desactiva el auto-commit al iniciar la transacción.

Por último, se muestra la transacción de aceptación de un pacto. Esta sólo requiere de la inserción del vendedor, y los datos ya presentes al hacer la oferta, en la tabla “acuerdoPacto” (3.13):

```
stmPacto = con.prepareStatement("insert into acuerdoPacto  
values(?,?,?,?)");  
stmPacto.setTimestamp(1, p.getMarcaTemporal());  
stmPacto.setString(2, p.getComprador());  
stmPacto.setString(3, p.getIdEmpresa());  
stmPacto.setString(4, p.getVendedor());
```

Al final de la funcionalidad se realiza un commit y se activa de nuevo el auto-commit. En caso de alguna excepción SQL, se realiza un rollback.



## Apéndice A. Manual de Funcionamiento de la Aplicación

En primer lugar, se debe crear una base de datos vacía con el nombre “MercadoValores” en el gestor de PostgreSQL. Tras la creación, se inserta el código SQL referente a la creación de la base de datos (el script de creación del apartado 3). Una vez insertado, hay que crear un nuevo usuario en la base de datos que tenga el nombre “alumnogreibd” y la contraseña “greibd2015”. Estas restricciones en el nombre de la base de datos y en el nombre y contraseña del usuario de la misma son debidas a que el programa Java se conecta a la base de datos referenciada en el fichero baseDatos.properties que se puede encontrar en la carpeta raíz del proyecto.

Para probar la aplicación se insertarán los datos del script de inserción.

Al iniciar el programa saltará la ventana de autenticación de usuarios (4.1). En el script de inserción hay un usuario de cada tipo con una contraseña fácil para acceder de forma rápida a la aplicación:

- Regulador:
  - Usuario: APerez
  - Contraseña: a
- Inversor:
  - Usuario: joseF
  - Contraseña: b
- Empresa:
  - Usuario: ImpeTelf
  - Contraseña: c