

# GeSoc

# Sistema de Gestión de Proyectos Sociales

Documento de Justificaciones - Entrega 4.2

# Grupo 5

| Apellidos    | Nombres              | Legajo    | Mail                            |
|--------------|----------------------|-----------|---------------------------------|
| COVELLO      | JORGE JULIO FERNANDO | 163.661-3 | jcovello@est.frba.utn.edu.ar    |
| CUELI        | JUAN FRANCISCO       | 163.279-6 | jcueli@est.frba.utn.edu.ar      |
| LEGUIZAMÓN   | ROCÍO                | 163.609-1 | rleguizamon@est.frba.utn.edu.ar |
| LESNIAK      | MATIAS               | 162.992-0 | mlesniak@est.frba.utn.edu.ar    |
| URTEAGA NAYA | MARTIN IGNACIO       | 164.412-9 | martin.un@est.frba.utn.edu.ar   |

#### Primera Sección

Para comenzar identificamos las siguientes entidades fundamentales:

- <u>Usuario</u> (pers\_usuarios): es quien usará nuestro sistema, por ello es fundamental persistir sus datos como su nombre, su contraseña (esta será mapeada mediante el uso la biblioteca SHA256 para garantizar la seguridad de la misma) y su rol. Asimismo, el <u>Usuario</u> posee una Primary Key (PK) autogenerada con el objetivo de crear un identificador inequívoco y además, nos es más sencillo para la utilización del ORM.
- <u>Egreso</u> (dom\_egresos): representa el dinero que la empresa gasta. Esta entidad es fundamental en nuestro dominio debido al hecho de que el sistema es de gestión contable. De forma similar al Usuario, el Egreso tiene una PK autogenerada para poder identificarlos inequívocamente y para facilitar la utilización del ORM. También cuenta con siete FK siendo ellas:
  - Proveedor (pers\_proveedores): es a quién le damos el dinero para que nos brinde un determinado bien o servicio.
  - o DocumentoComercial (dom documentos comerciales): respalda legalmente al Egreso.
  - Ingreso: representa el dinero que ingresa a la empresa y con dicho monto se puede afrontar los egresos de la misma.
  - o Criterio (dom criterios): tiene como objetivo clasificar los egresos de la organización.
  - Entidad (dom\_entidades): es la empresa que usará nuestra sistema y por ende tendrá muchos egresos asociados.
  - Valor (mon valor): representa el monto del egreso.

También posee un atributo para la fecha en el que se realizó el egreso, un boolean para saber qué egresos hay que validar, el tipo de medio de pago y su descripción.

- Ingreso (dom\_ingresos): por el contrario al Egreso, el Ingreso representa el dinero que entra a la organización y es fundamental persistir esta entidad ya que es vital en nuestro sistema de gestión contable. Aquí también se utiliza una PK autogenerada inequívoca para poder identificar fácilmente cada Ingreso y para utilizar sin problemas el ORM. Además, posee dos FK, una para relacionarse con la tabla Entidad ya que cada una de ellas puede tener muchos ingresos y otra que se relaciona con la tabla Valor que representa el monto del Ingreso en cuestión. Finalmente, posee una fecha para saber cuándo sucedió el Ingreso y una descripción para datos extra del mismo.
- Proveedor (pers\_proveedores): es quien le brinda un bien o servicio a la organización. Tiene una PK autogenerada e inequívoca para tener bien identificado a cada Proveedor. Inicialmente pensamos en modelar la dirección del proveedor con cuatro tablas (Dirección, DireccionPostal, Ciudad, Provincia y País) pero no era performante ya que para obtener la dirección completa era necesario realizar cinco JOINs. Por esta razón, decidimos desnormalizar la dirección del Proveedor y agregarle a dicha tabla la calle, la altura de dicha calle, el piso y el departamento (en caso de necesitarlos). Siguiendo con lo anterior, la tabla Proveedor posee un FK para relacionarse con la tabla Ciudad en caso de que se requiera más información de la dirección (es decir, la

ciudad, la provincia y el país). De esta manera, se reducen los posibles JOINs ya que el Proveedor posee todos los datos de su dirección en su propia tabla y sólo se realizaría un JOIN si surge la necesidad de obtener datos más específicos (de la tabla Ciudad) como los mencionados anteriormente. Asimismo, cuenta con atributos importantes como razón social, nombre y DNI/CUIT.

- <u>Presupuesto</u> (dom\_presupuestos): representa el precio de un bien o servicio que la empresa deberá pagar en caso de adquirirlo. En esta tabla hay una PK autogenerada con el fin de poder identificar cada Presupuesto. Además, posee dos FK, una para el <u>DocumentoComercial</u> que acompaña al <u>Presupuesto</u> y otra para establecer una relación con los proveedores ya que un <u>Proveedor</u> puede presentar varios presupuestos en la organización.
- <u>Entidad</u> (dom\_entidades): nos permite diferenciar entre entidades jurídicas y entidades base mediante el campo *tipoEntidad* (**B** si es Base y **J** si es Jurídica). Su PK es autogenerada y numérica para hacer buen uso del ORM.
- <u>TipoEntidadJuridica</u> (dom\_entidades\_juridicas): representa los datos de las entidades jurídicas dados por el dominio, deben ser almacenados para su futura consulta o modificación en caso de que se quiera agregar un nuevo egreso/ingreso o se quiera re-categorizar dicha entidad. Está siendo identificada por una PK autogenerada y numérica para facilitar las consultas a través del ORM. Asimismo, tiene una dirección que está formada por la calle, la altura de dicha calle, el piso y el departamento (en caso de necesitarlos). Análogamente con lo modelado en la tabla Proveedor, TipoEntidadJuridica también tiene un FK a Ciudad en caso de que se necesiten más datos sobre su dirección, evitando así, JOINs innecesarios. Una FK a la Categoría a la cual pertenece dicha Entidad, y una FK al Sector de la entidad.
- Categoría (ent\_categorias): representa los valores asociados (promedio de ventas anuales y personal máximo) de cada tipo de categoría. Contiene una PK autogenerada y numérica junto con los atributos previamente mencionados. Además, posee una FK a la tabla Sector debido a que un sector contiene muchas categorías. Es necesario aclarar que decidimos no modelar una relación muchos a muchos entre Categoría y Sector porque no existe más de una categoría que pertenezca a distintos sectores. En otras palabras, habría un muchos a muchos entre el tipoDeCategoria y el Sector pero como dicho atributo está desnormalizado dentro de la tabla Categoría, no es necesario una relación de muchos a muchos.
- <u>Sector</u> (ent\_sector): describe la actividad de una empresa y permite conocer en qué categoría está dicha empresa. Contiene una PK autogenerada y numérica junto con el nombre y la descripción del <u>Sector</u> en cuestión.
- CategoríaCustom (dom\_categorias): representa la categoría que crea la propia organización para clasificar u ordenar sus egresos de manera que le sea más fácil operar con ellos. No tiene nada que ver con la Categoría, ya que dicha tabla tiene una cantidad limitada de filas dadas por las regulaciones que se imponen en nuestro sistemas mientras que esta tabla tiene tantas filas como los usuarios lo deseen (por ahora). Esta tabla posee columnas de nombre, descripción y de id, siendo este último para almacenar el identificador de la fila. También tiene una columna con el Criterio al cual está asociado.

<u>Criterio</u> (dom\_criterios): es el criterio que crea dicha entidad para que se aplique a los diferentes egresos con el fin de que estos sean más fáciles de reconocer. Posee una PK numérica y autogenerada. También tiene una FK a la <u>Entidad</u> que creó dicho criterio, y otra al criterio que es padre. Así permitimos anidar criterios a un egreso y de esta manera se pueden crear jerarquías de <u>criterios</u>. Además, incluimos nombre y descripción para que el usuario pueda reconocerlos más fácilmente.

Las tablas País, Provincia y Ciudad se mantienen porque buscamos persistir lo que se obtiene de la API de ML, y luego poder consultar a la base de datos directamente, ya que es mucho más eficiente. Tampoco perdemos la consistencia de estos datos, al dejarlos como tablas separadas. A su vez, podemos aprovechar las tablas para poder listar sus respectivos registros en un menú desplegable.

- <u>Ciudad:</u> posee un nombre (varchar) y está relacionada con la <u>Provincia</u> a la cual pertenece geográficamente. Su PK es numérica y autogenerada.
- <u>Provincia:</u> sus campos son parecidos a los de <u>Ciudad</u>, pero en cuestiones de concepto son cosas totalmente distintas. Posee un nombre (varchar), un valor numérico autogenerado que la representa con respecto a otras tablas (PK) y su relación con el <u>País</u> al cual pertenece geográficamente (FK).
- <u>País:</u> tal como dijimos antes, se parece a las tablas previamente mencionadas pero su significado es totalmente diferente. Representa los países que reconocemos y a los cuales prestamos nuestro servicio de software. Está compuesto por un nombre (varchar) y un valor numérico que es su identificador.

#### Relaciones de Muchos a Muchos

- <u>CriteriosXEgreso</u>: como los criterios son personalizados, la empresa puede utilizarlos en más de un Egreso, y a su vez, por requerimiento del enunciado, un Egreso puede tener más de un Criterio asociado.
- <u>RevisorXEgreso</u>: un mismo Egreso puede tener más de un revisor y los revisores se repiten en distintos egresos. Además, el <u>Revisor</u> puede tener muchos egresos asociados con el objetivo de revisar cada uno de ellos.

#### Mapeo de Herencia

Entidad: en este caso mapeamos con la estrategia Single Table porque la diferencia entre "entidades" está dada por la relación entre entidades base y jurídicas que se refleja con la relación autorreferencial. Este campo será el único que podrá estar con valor nulo utilizando esta estrategia, por lo que consideramos que es un precio justo para el performance que nos brinda al evitar el uso de reiterados joins en las consultas a la base de datos.

• <u>TipoEntidadJuridica</u>: volvemos a usar la estrategia de mapeo *Single Table* ya que el único campo que podría estar con valores nulos es la relación con la <u>Categoría</u> correspondiente y puesto a que evitamos hacer joins, tal como aclaramos antes, nos parece correcto el uso de esta estrategia.

Queremos destacar que utilizamos composición entre las clases TipoEntidadJuridica y Entidad por lo que cuando mapeamos podría considerarse que utilizamos la estrategia de mapeo *Joined* pero como aclaramos antes surgió de la composición utilizada en el dominio de objetos y no en el mapeo de traer datos de mi dominio de objetos al mundo relacional. Por lo que nosotros **no** consideramos que fuese uso de dicha estrategia.

#### Ejemplos de las Tablas

Probamos la persistencia con algunos Tests y también con INSERTs directos a la base de datos.

#### Proveedor



#### País



#### • Provincia



#### Ciudad



#### • Documento Comercial



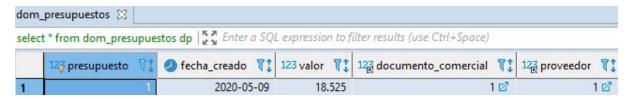
#### Valor



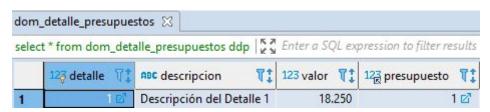
#### Usuario



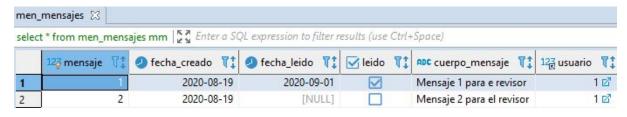
#### Presupuesto



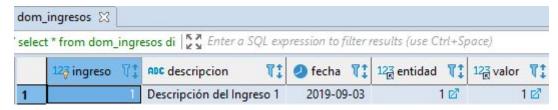
#### Detalle



#### Mensaje



#### Ingreso



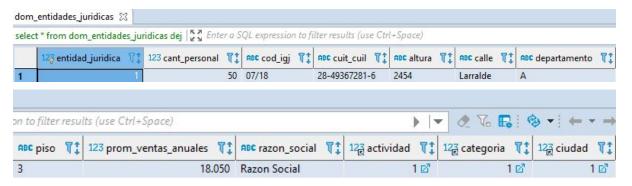
#### Egreso



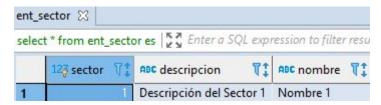
## Entidad Base



#### • Entidad Jurídica



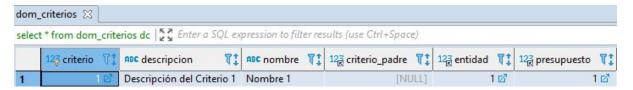
#### Sector



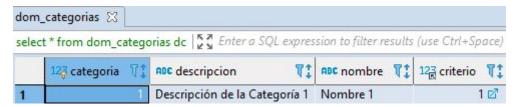
#### Categoría



#### Criterios



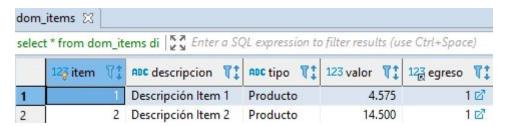
#### Categoría



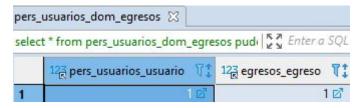
#### CriteriosXEgreso



#### Items



## RevisorXEgreso



# Segunda Sección

Para comenzar, detectamos las siguientes entidades fundamentales:

- SolicitarVinculacion: es la entidad encargada de generar la vinculación de egresos e ingresos.
- Importe: representa lo que nos interesa de un ingreso o egreso a vincular.
- Criterio: determina cómo se realizará la vinculación.
- <u>Condicion</u>: establece ciertos estándares para poder realizar la vinculación.
- <u>Vinculacion</u>: genera el JSON con el resultado de la vinculación.

Inicialmente, habíamos modelado una clase Egreso y otra Ingreso pero nos dimos cuenta que era innecesario porque ambas compartían la mayoría de sus atributos. Por esta razón, modelamos la clase Importe debido al hecho de que sólo nos interesa la fecha, el importe y la fecha de un ingreso o egreso. En consecuencia, ganamos mantenibilidad ya que reutilizamos código y es muy sencillo localizar la clase Importe en caso de tener que cambiar algo.

Por otro lado, la clase SolicitarVinculacion posee una lista de condiciones con el objetivo de poder agregar más de una Condicion a la hora de querer vincular egresos con ingresos. En consecuencia, nuestra solución es más extensible porque damos la posibilidad de incorporar más de una condición en el futuro. Cabe destacar, que elegimos modelar una interfaz Condicion con el fin de añadir nuevas clases que implementen el método *cumple()*. De esta manera, cumplimos con el requerimiento del enunciado que solicitaba poder agregar nuevas condiciones en un futuro, logrando así, una solución más extensible.

Por último, como se nos solicitaba la posibilidad de setear un criterio *Mix*, es decir, una combinación de otros criterios, decidimos agregarle una lista de criterios a la clase SolicitarVinculacion. De esta manera, si se escoge un solo Criterio la lista solamente tendrá un elemento mientras que si se desea un *Mix* la lista tendrá más de uno. En consecuencia, nos ahorramos el modelar una clase *Mix* logrando así una solución más consistente ya que se evitan posibles condiciones de recursividad entre criterios. Asimismo, modelamos una interfaz Criterio ya que ganabamos mayor mantenibilidad porque es fácil de localizar a cada Criterio y extensibilidad debido al hecho de que se pueden agregar más criterios simplemente haciendo que éstos implementen dicha interfaz.

En conclusión, con todo lo mencionado anteriormente nuestra clase SolicitarVinculación posee mayor cohesión ya que se delega en dos interfaces (Condicion y Criterio) la responsabilidad de establecer ciertos estándares a la hora de realizar una vinculación y de determinar cómo se llevará a cabo dicha vinculación. Análogamente, se delega en la clase Vinculación la responsabilidad de generar el JSON con la vinculación concretada que será enviada al módulo GeSoc, logrando así mayor cohesión en la clase SolicitarVinculación debido al hecho de que ésta última no debe preocuparse por la confección de dicho JSON.