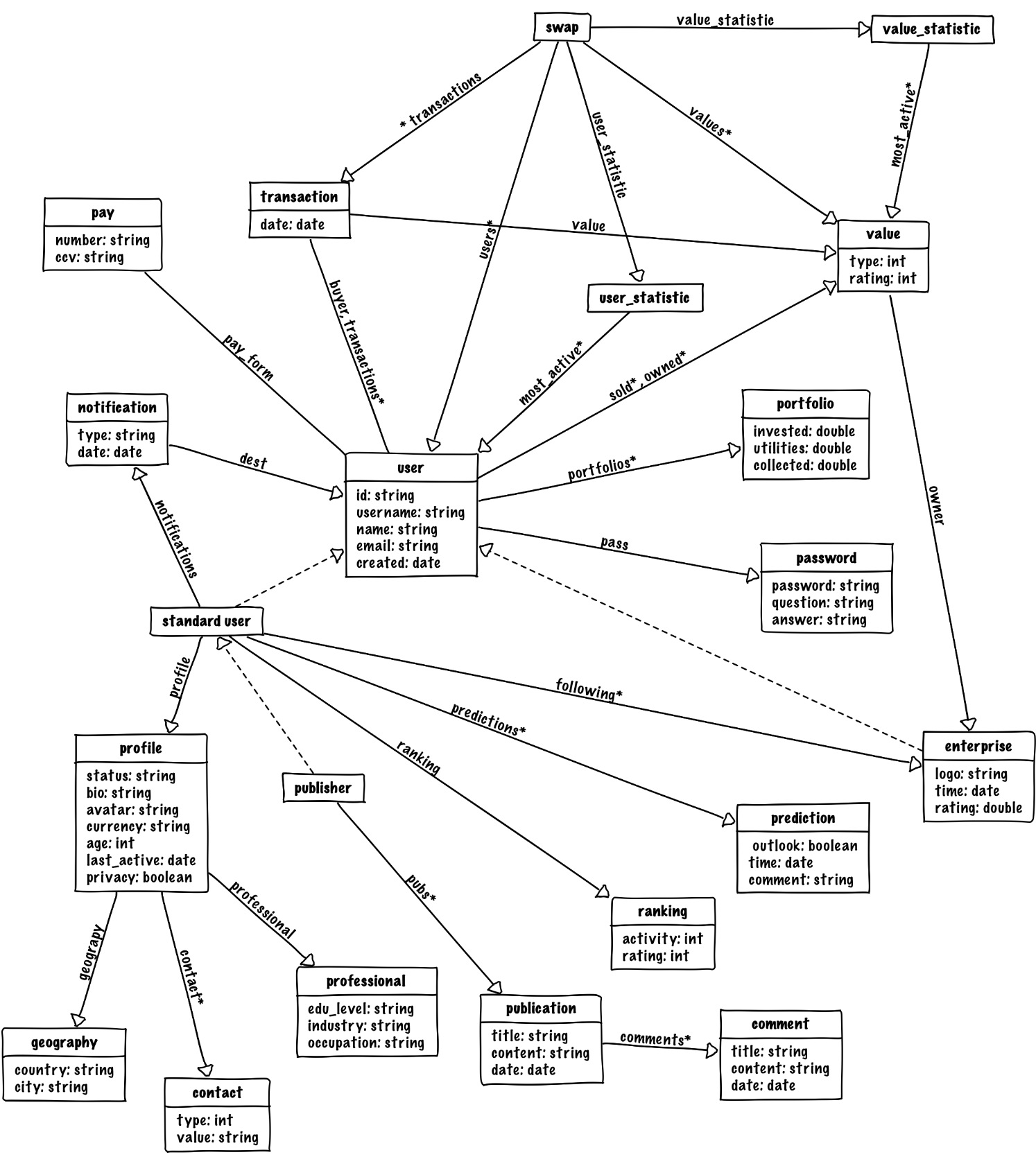
Sebastián Valencia Calderón

Juan Camilo Bages Prada

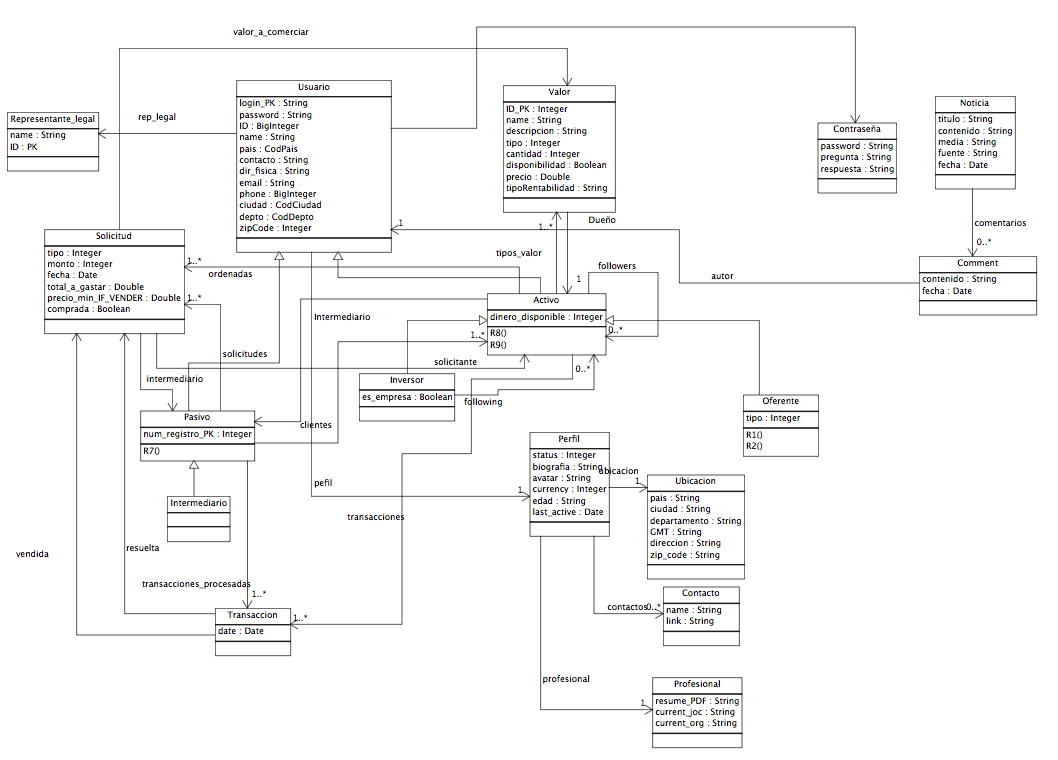
**Análisis.**

Para el desarrollo de esta iteración, fue necesario agregar ciertas características al modelo de la aplicación, es decir, la disposición de nuevos requerimientos hizo necesaria la inclusión o modificación de nuevas clases, asociaciones o atributos en el modelo propuesto originalmente. A continuación, se muestran los dos modelos, el anterior y el nuevo, es pertinente decir que el nuevo incluye los modelos auxiliares de desarrollo, así como los contemplados para cubrir el enunciado actual de la aplicación.

1. Modelo anterior: (UML-former.png)



1. Modelo actual: (UML-current.png)



Como puede observarse, algunas clases fueron agregadas, otras fueron modificadas y pocas fueron eliminadas, la comparación, se encuentra a continuación:

|  |  |  |
| --- | --- | --- |
| Comparación de las clases y modelos propuestos en ambas iteraciones. | | |
| Clase | Modelo anterior | Modelo actual |
| Swap | En el modelo anterior, se traía el modelo conceptual de diseño adoptado por los cursos anteriores, donde era necesaria la presencia de una clase maestra que manejara los recursos y servicios principales de una aplicación. | Gracias a las nuevas metodologías de diseño adoptadas por la nueva ola de desarrollo de software, así como los patrones de diseño, el modelo enseñado por los cursos anteriores carece de sentido ante los requerimientos de una aplicación robusta en la modernidad. |
| Usuario | La clase de usuario fue fortalecida a la luz de los nuevos requerimientos. Algunas asociaciones únicas de esta clase, fueron encapsuladas en una sola clase de usuario. Esto favorece el diseño, sin embargo, la planeación posee problemas para la implementación o normalización de las tablas. | Se agregó la información adecuada para el manejo de monedas, básicamente, los modelos de perfil y contraseña, colapsaron en la clase usuario. Asimismo, se enfatizó la presencia de el usuario como un todo, del cual heredan los objetos reactivos de la aplicación. Esto facilitó el diseño normal de las tablas. |
| Representante legal |  | La clase fue agregada para favorecer el proceso de normalización. |
| Valor | La clase y el diseño de las asociaciones, no contemplaba la pluralidad de opciones que ofrecía o debía ofrecer este modelo (modelo como en MVC). | La tabla de valor, contempla las opciones de compra y otros requerimientos asociados a esta. Asimismo, se tiene en cuenta el proceso de adquisición por parte de un intermediario. |
| Solicitud | El concepto no estaba del todo integrado en la implementación del prototipo, anteriormente, todo el procesos se realizaba a través de transacciones. Asimismo, se fortaleció el concepto de transacción. | Las solicitudes poseen facilidades para ofertar, y manejar transacciones (en el sentid bancario) fácilmente. Pensando en lo anterior, poseen conocimiento del intermediario, oferente y fecha de realización. |
| Transacción |  | El concepto, así como la implementación, son resultado de des solicitudes. |
| Clases misceláneas |  | Aún no implementadas |

**Diseño.**

1. A partir del diseño existente, analice el impacto que representa la introducción de los nuevos requerimientos y restricciones a nivel del modelo conceptual. Realice los cambios necesarios en su modelo relacional para respetar las reglas de negocio y asegurar la calidad del mismo. Tenga en cuenta los comentarios recibidos en la sustentación del taller 2. Documente el diseño y las decisiones tomadas para crear los elementos de la base de datos que da el respaldo de persistencia a la aplicación, a partir del modelo conceptual. Incluya un listado con las tablas generadas en la base de datos, utilizando los estándares establecidos, disponibles en la wiki del curso (sección tutoriales). Este listado debe incluir el nombre de la tabla, el nombre y el tipo de dato de sus campos así como los nombres de restricciones de llaves primarias, llaves foráneas y de chequeo. (Sea claro en mencionar explícitamente los cambios relevantes entre su diseño entregado con el taller anterior y este.)

Dada la similitud estructural de ambos modelos, el cambio o migración manual de los modelos existentes, no representa un cambio significativo en costos de tiempo o instrucción de alguna tecnología o metodología. Esto se debe, a que el modelo inicial incluyó exhaustivamente los requerimientos propuestos en el primer enunciado. A continuación, se incluye el esquema de tabla de cada modelo propuesto, posteriormente, los cambios relevantes en el modelo con respecto a la anterior iteración.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **USERS** | | | | | | |
| **LOGIN** | **USER\_ID** | **FIRST\_NAME** | **LAST\_NAME** | **EMAIL** | **PHONE** | **USER\_PASS** |
| varchar(20) | varchar(20) | varchar(20) | varchar(20) | varchar(20) | varchar(20) | varchar(20) |
| PK | NN | NN | NN | NN | NN | NN |

La tabla USERS contiene información relevante sobre los datos de acceso del usuario, asimismo, como se explicó, es la entidad fundamental de todo usuario de Swap, es decir, cualquier agente en Swap es un usuario, entiéndase agente como entidad manejada por un humano o un sistema de inteligencia artificial capaz de simular las acciones relativas a un humano. Se decidió incluir información de contacto dada la relación única de esta información con cada usuario. Sin embargo, se maneja flexibilidad para la normalización, nótese que ni EMAIL ni PHONE son PK.

|  |  |
| --- | --- |
| **ACTIVES** | |
| **USER\_LOGIN** | **AVAILABLE\_MONEY** |
| varchar(20) | float |
| PK, FK(USERS.LOGIN) | NN, |

La tabla ACTIVES contiene información relevante sobre los usuarios que son capaces de comprar acciones en el sistema, es decir, los usuarios activos. Pudo haberse incluido información sobre el tipo en la tabla USERS pero esto generaba redundancia no conveniente para la normalización del modelo. Por lo tanto, se maneja la tabla ACTIVES con información del login del usuario, es decir el PK en USERS, y el PK de ésta tabla. AVAILABLE\_MONEY, representa la cantidad de dinero del usuario para comprar valores, o los que ha ganado en el proceso cotidiano de Swap.

|  |  |
| --- | --- |
| **PASSIVES** | |
| **PASSIVE\_LOGIN** | **PASSIVE\_REGISTER** |
| varchar(20) | varchar(20) |
| PK, FK(USERS.LOGIN) | NN, UNIQUE |

De manera análoga con la tabla ACTIVES, la tabla PASSIVES contiene información asociada a los usuarios pasivos del sistema. Éstos usuarios son los intermediarios, es decir, los usuarios capaces de interactuar por un usuario activo ante el mercado. Actúa como empleado de Swap, como el enunciado lo pedía, este tiene un número de registro único, que no es PK para favorecer la organización de índices dentro de la base de datos.

|  |  |
| --- | --- |
| **ACTIVESPASSIVES** | |
| **ACTIVE\_LOGIN** | **PASSIVE\_LOGIN** |
| varchar(20) | varchar(20) |
| PK, FK(ACTIVES.USER\_LOGIN) | PK, FK(PASSIVES.PASSIVE\_LOGIN), UNIQUE |

Esta relación, evidencia la conexión entre los intermediarios y los inversionistas, es decir, la asignación entre los intermediarios y sus clientes, además, tiene en cuenta que puede haber un intermediarios atendiendo solicitudes de varios clientes, sin embargo, no existe un inversionista con varios intermediarios asociados.

|  |  |
| --- | --- |
| **INVESTORS** | |
| **USER\_LOGIN** | **IS\_ENTERPRISE** |
| varchar(20) | varchar(1) |
| PK, FK(ACTIVES.USER\_LOGIN) | NN |

Evidencia explícitamente la relación entre usuarios activos e inversionistas, donde cada uno puede ser empresa o no. Esta información, puede manejarse desde la relación anterior, sin embargo, dados los privilegios que ser empresa corresponde, existe la pertinencia de manejar estos datos de esta manera.

|  |  |  |
| --- | --- | --- |
| **LEGALS** | | |
| **LEGAL\_ID** | **LEGAL\_NAME** | **USER\_LOGIN** |
| varchar(20) | varchar(20) | varchar(20) |
| PK | NN, UNIQUE | NN, FK(USERS.LOGIN) |

La relación LEGALS, representa los representantes legales dado el usuario, se sabe que cada usuario debe tener un representante legal, y que un representante legal trabaja únicamente para un usuario activo.

|  |  |
| --- | --- |
| **OFFERANTS** | |
| **USER\_LOGIN** | **OFFERANT\_TYPE** |
| varchar(20) | varchar(1) |
| PK, FK(ACTIVES.USER\_LOGIN) | NN |

La relación OFFERENATS, ayuda a entender la pertinencia de la tabla INVESTORS, ya que existen varios tipos de usuario no mutuamente excluyentes, la información sobre el tipo de usuario puede determinarse a través del login, pero es más claro separar a los usuarios en tablas según su tipo, es claro que esta decisión de diseño o afecta la forma normal de la tabla dada.

|  |  |  |
| --- | --- | --- |
| **PORTFOLIOS** | | |
| **PK\_ID** | **USER\_LOGIN** | **RISK** |
| number(38, 0) | varchar(20) | varchar(1) |
| PK | FK(USER.LOGIN) |  |

La relación contiene información sobre el portafolio de cada usuario, es decir, la colección de valores que éste posee, para ello, posee información del riesgo, y el usuario correspondiente.

|  |  |  |
| --- | --- | --- |
| **PORTFOLIOS\_VALS** | | |
| **PK\_ID** | **PK\_PORTFOLIO** | **PK\_VAL** |
| number(38, 0) | number(38, 0) | number(38, 0) |
| PK | FK(PORTFOLIO.PK\_ID) | FK(VALS.PK\_ID) |

La relación PORTFOLIOS\_VALS evidencia la colección de un portafolio con unos valores, es decir, los valores contenidos en un portafolio, lo cual corresponde al portafolio en su totalidad. Esta tabla es pertinente dada la multiplicidad de la cantidad de valores de un portafolio.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **RENTS** | | | | | | |
| **PK\_ID** | **RENT\_NAME** | **DESCRIPTION** | **RENT\_FN** | **RENT\_LEN** | **RENT\_TYPE** | **OFFERANT\_LOGIN** |
| number(38) | varchar(20) | varchar(140) | varchar(1) | varchar(1) | varchar(1) | varchar(20) |
| PK | NN | NN | NN | NN | NN | FK(OFFERNATS) IN USER\_LOGIN |

La relación RENTS, representa un tipo de renta de un valor en Swap, este posee información sobre el nombre, descripción, función, tipo, y login del oferente.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **VALS** | | | | | | |
| **PK\_ID** | **VAL\_NAME** | **DESCRIPTION** | **VAL\_TYPE** | **AMOUNT** | **PRICE** | **RENT\_ID** |
| number(38) | varchar(20) | varchar(140) | varchar(1) | number(38) | float | number(38) |
| PK | NN | NN | NN | NN | NN | NN |

La relación VALS, representa un valor, posee información básica así como el tipo de renta (es decir un apuntador al registro real que representa la renta de este valor).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SOLICITUDES** | | | | | |
| **PK\_ID** | **REQUEST\_TYPE** | **AMOUNT** | **AMOUNT\_UNIT** | **TIMESTAMP** | **ACTIVE\_LOGIN** |
| number(38) | varchar(20) | varchar(140) | varchar(1) | number(38) | float |
| PK | NN | NN | NN | NN | FK(ACTIVES.LOGIN) |

La relación de solicitudes, contiene únicamente información acerca de

Con respecto al impacto de introducción de los nuevos requerimientos, a nivel de modelo y restricciones, ….

Si se compara el modelo actual con el inmediatamente anterior….

1. Valide que su modelo se encuentra en BCNF y que no presenta anomalías de inserción, borrado o actualización con respecto a las reglas de negocio.
2. Documente la lógica de los nuevos requerimientos a desarrollar, descritos en la sección de caso de estudio de este documento. En este punto se requiere definir los mecanismos que utiliza para garantizar las propiedades ACID del requerimiento.

**Implementación.**

1. Ajuste las tablas creadas en Oracle de acuerdo a las decisiones del punto anterior.
2. Pueble las tablas con información suficiente para poder realizar pruebas.

* Diseñe los datos que le permitan verificar adecuadamente las reglas de negocio. Note que es más importante generar adecuadamente los datos, que obtener un número muy grande de ellos.
* Puede escribir un programa de generación automática de datos acorde al diseño establecido para los mismos.
* Para la población de las tablas utilice herramientas de carga masiva como SQLLoader o las disponibles en SQLDeveloper. Consulte el tutorial disponible en la wiki del curso sobre SQLLoader.

1. Ver código.
2. Verifique el comportamiento transaccional de los requerimientos que implican actualización, inserción o borrado de información.
3. Diseñe un esquema de respaldo de datos. En máximo una página describa los procesos y la infraestructura que usted propone para que el negocio al cual corresponde la aplicación desarrollada pueda tener un esquema confiable de disponibilidad y recuperación de datos.
4. Implemente los escenarios de prueba para que le permiten asegurar el correcto funcionamiento de la aplicación y la corrección y calidad de los datos en la base de datos.

* En un archivo Excel documente claramente, para cada caso de uso, cuáles son los datos que le permiten realizar las pruebas, tanto para los casos de terminación exitosa como los fallidos. Indique cuáles son las respuestas esperadas que corresponden a los datos de prueba.

**Bono.**

1. ¿Qué diferencias hay en el manejo transaccional por parte de un contenedor de aplicaciones con respecto al manejo dado por parte del programador de la aplicación?. Explique claramente las ventajas y desventajas de cada uno de ellos.

SW DE BAJO NIVEL QUE JUEGA CON EL PROTOCOLO DE RED….

OPEN SOURCE

CONCURRENCIA HEREDADA DE LAS CAPACIDADES DEL SERVIDOR

AISLA POR COMPLETO CADA SESIÓN……