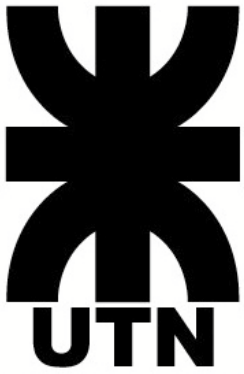


Universidad Tecnológica Nacional

Facultad Regional Buenos Aires

Gestión De Datos

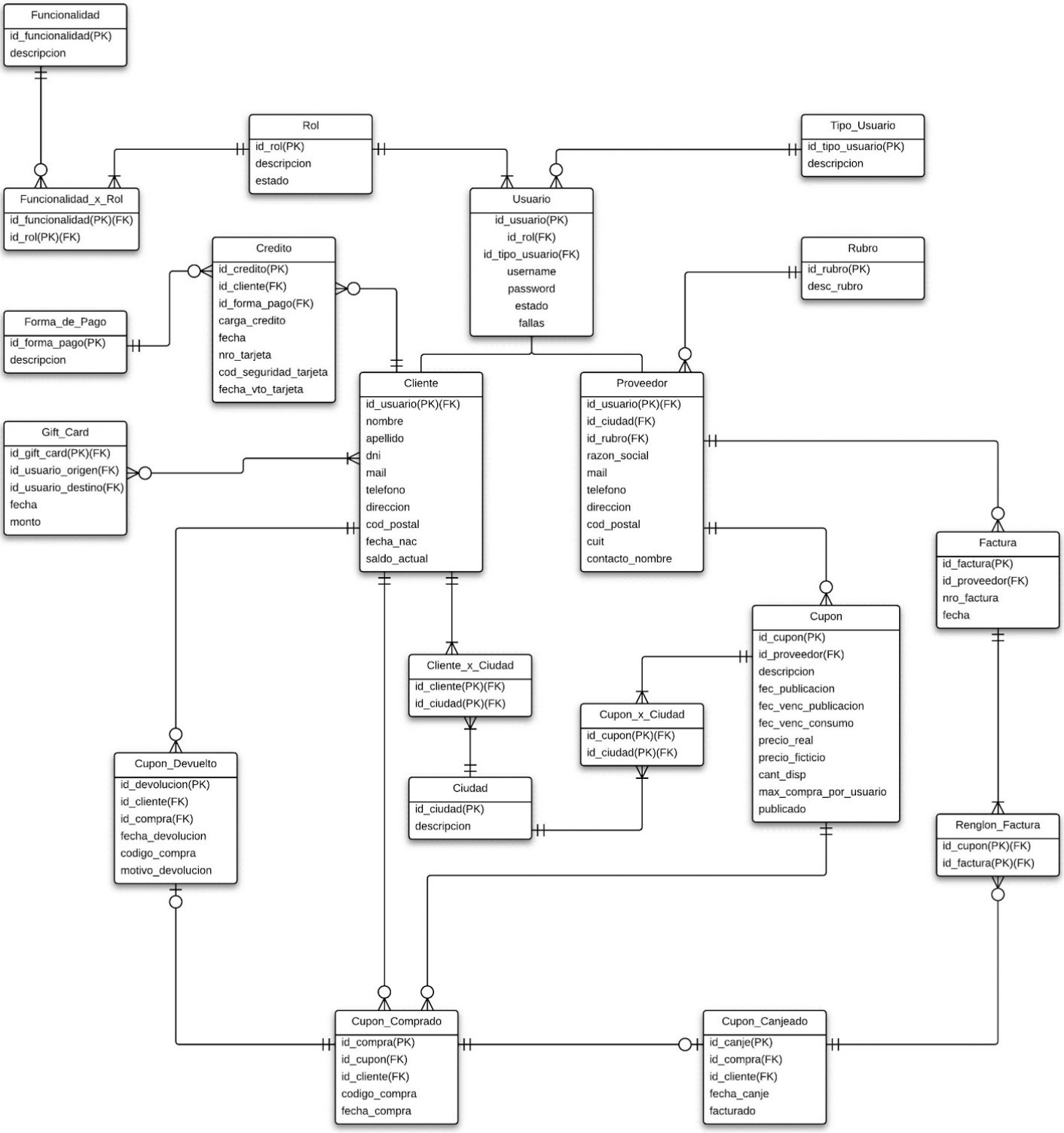
Trabajo Práctico: Cuponete - 2C2012

Random

Zavatto, Martín	140.229-8
Aloi, Federico	140.347-3
Perez Stolfa, Tiziano	141.575-0
Fontenla, Tomás	140.469-6

Índice

pag


Funcionalidad
id_funcionalidad(PK)
descripcion
Rol
id_rol(PK)
descripcion
estado
Tipo_Usuario
id_tipo_usuario(PK)
descripcion
Funcionalidad_x_Rol
id_funcionalidad(PK)(FK)
id_rol(PK)(FK)
Usuario
id_usuario(PK)
id_rol(FK)
id_tipo_usuario(FK)
username
password
estado
fallas
Credito
id_credito(PK)
id_cliente(FK)
id_forma_pago(FK)
carga_credito
fecha
nro_tarjeta
cod_seguridad_tarjeta
fecha_vto_tarjeta
Rubro
id_rubro(PK)
desc_rubro
Forma_de_Pago
id_forma_pago(PK)
descripcion
Cliente
id_usuario(PK)(FK)
nombre
apellido
dni
mail
telefono
direccion
cod_postal
fecha_nac
saldo_actual
Proveedor
id_usuario(PK)(FK)
id_ciudad(FK)
id_rubro(FK)
razon_social
mail
telefono
direccion
cod_postal
cuit
contacto_nombre
Gift_Card
id_gift_card(PK)(FK)
id_usuario_origen(FK)
id_usuario_destino(FK)
fecha
monto
Factura
id_factura(PK)
id_proveedor(FK)
nro_factura
fecha
Cupon
id_cupon(PK)
id_proveedor(FK)
descripcion
fec_publicacion
fec_venc_publicacion
fec_venc_consumo
precio_real
precio_ficticio
cant_disp
max_compra_por_usuario
publicado
Cliente_x_Ciudad
id_cliente(PK)(FK)
id_ciudad(PK)(FK)
Cupon_x_Ciudad
id_cupon(PK)(FK)
id_ciudad(PK)(FK)
Cupon_Devuelto
id_devolucion(PK)
id_cliente(FK)
id_compra(FK)
fecha_devolucion
codigo_compra
motivo_devolucion
Ciudad
id_ciudad(PK)
descripcion
Renglon_Factura
id_cupon(PK)(FK)
id_factura(PK)(FK)
Cupon_Comprado
id_compra(PK)
id_cupon(FK)
id_cliente(FK)
codigo_compra
fecha_compra
Cupon_Canjeado
id_canje(PK)
id_compra(FK)
id_cliente(FK)
fecha_canje
facturado

Migración (Modelo de Datos)

SQL

Todos los INSERTs y UPDATEs decidimos manejarlos con Store Procedures, ya que de esa forma abstraemos a la aplicación del esquema de la base de datos y evitamos tener que realizar cambios a medida que fue avanzando el proyecto. Esto también nos permitió aprovechar que estamos del lado del SQL y poner ahí las validaciones más “interesantes” y que involucraban a datos previos: dni duplicado en cliente, proveedor habilitado a la hora de armar cupones, etc. De esta forma, del lado del cliente C# sólo validamos que los tipos de datos fueran los correctos y que no se pueda submitear un formulario sin haber completado todos los campos obligatorios.

En general las queries más simples las hardcodeamos en C#, mientras que preferimos armar vistas para aquellas que implicaban joinear algunas tablas y tenían condiciones estáticas (a excepción de las utilizadas por los listados estadísticos, dado que estas vistas nos tardaban demasiado en armar). De esa forma dejamos la parte que no cambia especificada en dichas vistas y desde la aplicación sólo nos ocupamos de los filtros ingresados por el usuario.

Por cambiar un rol entendimos que el usuario cambiaba a un rol similar al que registró, por ejemplo, si se quisiera crear un administrador con menos poderes, se podría, pero encontramos cambiar el rol de un administrador a un cliente/proveedor innecesario, lo mismo sucede con los clientes y proveedores. Para ello creamos una tabla con los tres tipos posibles de usuario, a la cual hace referencia la tabla usuario.

Tablas Utilizadas (PK = Primary Key, PK Comp = Primary Key Compuesta, FK = Foreign Key):

Usuario:	**Cliente:**
id_usuario (PK) - bigint **username** - nvarchar(255) **password** - nvarchar(255) **id_rol (FK)** - bigint **id_tipo_usuario(FK)** - bigint **estado** - numeric(1,0) **fallas** - numeric(1,0)	**id_usuario (PK) (FK)** - bigint **nombre** - nvarchar(255) **apellido** - nvarchar(255) **dni** - numeric(18,0) **mail** - nvarchar(100) **telefono** - numeric(18,0) **direccion** - nvarchar(255) **cod_postal** - numeric(18,0) **fecha_nac** - datetime **saldo_actual** - bigint
Rol:	**Proveedor:**
id_rol (PK) - bigint	**id_usuario (PK) (FK)** - bigint

descripcion - nvarchar(50) **estado** - numeric(1,0)	**razon_social** - nvarchar(100) **mail -** nvarchar(100) **telefono** - numeric(18,0) **direccion** - nvarchar(100) **cod_postal** - numeric(10,0) **id_ciudad (FK)** - bigint **cuit** - nvarchar(20) **id_rubro (FK)** - bigint **contacto_nombre** - nvarchar(100)
Tipo_usuario: **id_tipo_usuario (PK)** - bigint **descripcion** - nvarchar(50)	**Gift_Card:** **id_gift_card (PK)** - bigint **fecha** - datetime **monto** - numeric(18,2) **id_usuario_origen (FK)** - bigint **id_usuario_destino (FK)** - bigint
Cupon: **id_cupon (PK)** - bigint **descripcion** - nvarchar(255) **fec_publicacion** - datetime **fec_venc_publicacion** - datetime **fec_venc_consumo** - datetime **precio_real** - numeric(18,2) **precio_ficticio** - numeric(18,2) **id_proveedor (FK)** - bigint **cant_disp** - numeric(18,0) **max_compra_por_usuario** - numeric(18,0) **publicado** - numeric(1,0)	**Cupon_Devuelto:** **id_devolucion (PK)** - bigint **fecha_devolucion** - datetime **id_cliente (FK)** - bigint **id_compra (FK)** - bigint **codigo_compra** - nvarchar(50) **motivo_devolucion** - nvarchar(255)
Cupon_Comprado: **id_compra (PK)** - bigint **codigo_compra** - nvarchar(50) **fecha_compra** - datetime **id_cupon (FK)** - bigint **id_cliente (FK)** - bigint	**Cupon_Canjeado:** **id_canje (PK)** - bigint **fecha_canje** - datetime **id_compra (FK)** - bigint **id_cliente (FK)** - bigint **facturado** - int
Funcionalidad: **id_funcionalidad (PK)** - bigint	**Rubro:** **id_rubro (PK)** - bigint

descripcion - nvarchar(255)	**desc_rubro** - nvarchar(100)
Credito: **id_credito (PK)** - bigint **id_cliente (FK)** - bigint **carga_credito** - bigint **fecha** - datetime **id_forma_pago** (FK) - bigint **nro_tarjeta** - numeric(15,0) **cod_seguridad_tarjeta** - numeric(3,0) **fecha_vto_tarjeta** - nvarchar(5)	**Ciudad**: **id_ciudad (PK)** - bigint **descripcion** - nvarchar(255)
Factura: **id_factura (PK)** - bigint **nro_factura** - numeric(18,0) **fecha** - datetime **id_proveedor (FK)** - bigint	**Funcionalidad_x_rol:** **id_funcionalidad (PK Comp) (FK)** - bigint **id_rol (PK Comp) (FK)** - bigint
Cliente_x_Ciudad: **id_cliente (PK Comp) (FK)** - bigint **id_ciudad (PK Comp) (FK)** - bigint	**Forma_de_Pago:** **id_forma_pago (PK)** - bigint **descripcion** - nvarchar(255)
Renglon_Factura: **id_cupon_canjeado (PK Comp) (FK)** - bigint **id_factura (PK Comp) (FK)** - bigint	**Cupon_x_Ciudad:** **id_cupon (PK Comp) (FK)** - bigint **id_ciudad (PK Comp) (FK)** - bigint

Usuario: Contiene los datos principales sobre los usuarios.

Cliente: Contiene los datos sobre los clientes.

Rol: Contiene datos sobre los distintos roles. La **descripción** posee el nombre del rol (ej: Cliente, Administrador, etc.) y el **estado** es un entero que nos sirve de booleano (0 = deshabilitado, 1 = habilitado). Leer apartado Rol / Tipo de Usuario en decisiones estratégicas.

Proveedor: Contiene los datos sobre los proveedores.

Tipo_Usuario: Contiene el número asignado de tipo de usuario y la descripción del tipo de usuario. Leer apartado Rol / Tipo de Usuario en decisiones estratégicas.

Cupón: Contiene datos pertinentes sobre los cupones como pueden ser fechas, cantidades disponibles, cantidades máximas de compra, entre otras indicadas en la tabla. También contiene la propiedad **publicado**, la

cual es un entero que sirve de booleano (0 = Sin publicar, 1 = Publicado).

Cupón_Devuelto: Contiene información sobre la devolución de cada cupón, la tabla se vincula con otras tablas sobre cupón.

Cupón_Comprado: Contiene información sobre la compra de cada cupón, esta tabla también se vincula con otras tablas sobre cupón.

Cupón_Canjeado: Contiene información sobre el canje de cada cupón, la tabla se vincula con otras tablas sobre cupón.

Gift_Card: Contiene información sobre las giftcards, incluido en ellos los id de los clientes origen y destino involucrados en cada una.

Funcionalidad: Contiene el id de cada funcionalidad y su descripción, de esta manera se puede relacionar que funcionalidades están permitidas para cada rol. Leer apartado Rol / Tipo de Usuario en decisiones estratégicas.

Rubro: Contiene el id de cada rubro y su descripción (Ej: Electrónica).

Credito: Contiene información sobre cada carga de crédito realizada por los clientes.

Ciudad: Tabla que vincula la ciudad con su id.

Factura: Tabla que contiene información sobre cada factura.

Funcionalidad_x_rol: Tabla vinculante entre las tablas rol y funcionalidad.

Cliente_x_ciudad: Tabla vinculante entre las tablas cliente y ciudad.

Forma_de_pago: Contiene el id de cada forma de pago y su descripción (crédito, efectivo).

Renglon_Factura: Tabla contenedora de la información de los renglones de cada factura.

Cupon_x_ciudad: Tabla vinculante entre las tablas cupon y ciudad.

Stored procedures, funciones, triggers y vistas

Procedures

RANDOM.AgregarRol: Procedure utilizado para agregar un rol. Contiene validaciones.

RANDOM.AgregarFuncionalidadPorRol: Procedure utilizado para agregar una funcionalidad por rol.

RANDOM.QuitarFuncionalidadPorRol: Procedure utilizado para quitar una funcionalidad por rol.

RANDOM.CambiarNombreRol: Procedure utilizado para cambiar el nombre de un rol. Contiene validaciones.

RANDOM.HabilitarRol: Procedure utilizado para habilitar un rol.

RANDOM.DeshabilitarRol: Procedure utilizado para deshabilitar un rol.

RANDOM.ModificarUsuario: Procedure utilizado para la modificación de los usuarios. Contiene validaciones.

RANDOM.HabilitarUsuario: Procedure utilizado para la habilitación de los usuarios.

RANDOM.DeshabilitarUsuario: Procedure utilizado para deshabilitar usuarios.

RANDOM.IncrementarFallas: Procedure utilizado para incrementar las fallas de un usuario cuando no ingresa la contraseña correcta.

RANDOM.ReiniciarFallas: Procedure utilizado para reiniciar las fallas de un usuario a 0.

RANDOM.RegistrarProveedor: Procedure utilizado para la registración de un proveedor. Este contiene validaciones que serán chequeadas al momento de registrar o se procederá a avisar en el programa.

RANDOM.ModificarProveedor: Procedure utilizado para la modificación de un proveedor. También contiene validaciones.

RANDOM.RegistrarCliente: Procedure utilizado para la registración de un cliente. Contiene validaciones.

RANDOM.ModificarCliente: Procedure utilizado para la modificación de un cliente. Contiene validaciones.

RANDOM.AgregarClientePorCiudad: Procedure utilizado para agregar un cliente por ciudad.

RANDOM.QuitarClientePorCiudad: Procedure utilizado para quitar un cliente por ciudad.

RANDOM.PublicarCupon: Procedure utilizado para la publicación de cupones.

RANDOM.FacturarCupones: Procedure utilizado para la facturación de cupones.

RANDOM.ArmarCupon: Procedure utilizado para el armado de cupones. Contiene validaciones.

RANDOM.AgregarCuponPorCiudad: Procedure utilizado para agregar un cupón por ciudad.

RANDOM.QuitarCuponPorCiudad: Procedure utilizado para quitar un cupón por ciudad.

RANDOM.RegistrarConsumo: Procedure utilizado para registrar que un cupón fue consumido. Contiene validaciones.

RANDOM.CargarCredito: Procedure utilizado para el registro de la carga de crédito. Contiene validaciones.

RANDOM.ComprarGiftCard: Procedure utilizado para el registro de una compra de GiftCard. Contiene

validaciones.

RANDOM.ComprarCupon: Procedure utilizado para el comprado de un cupón. Contiene validaciones.

RANDOM.PedirDevolucion: Procedure utilizado para el pedido de una devolución de un cupón. Contiene validaciones.

RANDOM.DevolverCupon: Procedure utilizado para la devolución de un cupón.

Funciones

RANDOM.EsCliente: Función que devuelve 1 si el usuario es cliente o 0 si no lo es.

RANDOM.EsProveedor Función que devuelve 1 si el usuario es proveedor o 0 si no lo es.

Vistas

RANDOM.Cupones_Para_Cliente: Vista que trae los datos necesarios para seleccionar qué cupón puede comprar cada cliente.

RANDOM.Historial_Compra_Cupones: Vista que trae los cupones con los que se realizó alguna operación relacionado con su cliente y en qué estado está el cupón.

RANDOM.Facturacion_Proveedor: Vista que trae los consumos registrados por proveedor.

Triggers

RANDOM.Inhabilitar_Rol: Cambia el rol de los clientes a NULL cuando se deshabilita dicho rol en la tabla rol.

Decisiones estratégicas (incluidas decisiones sobre la migración)

- **Decisiones pertinentes a los cupones:**
    - Se decidió tomar el código de cupón de la tabla Maestra (ej: NJWYXOELWQ12) como el código único que representa la compra de un cupón. Dado que se encontraron casos en el que dicho código estaba repetido (distintos clientes poseían el mismo código de compra), se decidió diferenciarlos agregando una numeración consecutiva luego de una barra (/). Por ejemplo:

NJWYXOELWQ12/1 ,NJWYXOELWQ12/2. De esta manera se distingue el código.
- Decidimos que una vez migrada la tabla, trabajando con el nuevo sistema, las nuevas compras registradas tendrán como código de compra el id autoincremental de la tabla cupon_comprado.
- Debido a la falta de ciudad de disponibilidad de cada cupón en la tabla maestra, se optó que en los cupones migrados, las ciudades en los que el cupón estará disponible son las ciudades de los clientes que ya lo han comprado.
- Por cuestiones de migración, se decidió que, al no estar presente en la tabla maestra, la fecha de vencimiento de la publicación sea la misma que la de vencimiento del consumo. Pudiendo realizar canjes y devoluciones hasta el día del vencimiento del consumo inclusive.
- Debido a que la tabla maestra no especificaba una fecha de vencimiento para el consumo y otra para la publicación, se tomó la única fecha de vencimiento para ambos datos.
- Se decidió que todos los cupones que ya estaban en la base de datos, en el nuevo modelo pasarán a estar publicados.
- Debido a la falta de especificación en la tabla maestra, se optó que la máxima compra por usuario para los cupones migrados sea de un solo cupón.
- Sobre “Devolución Cupón” : Debido a que en la tabla maestra no figuraban los motivos de los cupones devueltos, se migraron sin motivo de devolución.

- **Decisiones pertinentes a los clientes:**
  - Se decidió poner en blanco los datos que no estaban en la tabla maestra (ej: código postal) y obligar al usuario a que los complete cuando se loguea al sistema.
  - Se resolvió, debido a falta de información en la tabla maestra, que la ciudad de preferencia de los clientes migrados por defecto sea la ciudad de residencia de los mismos.

- **Decisiones pertinentes a los proveedores:**
  - Se decidió migrar en blanco los datos faltantes de la tabla maestra (ej. email, código postal, nombre de contacto) y obligar al usuario a completarlos cuando se loguea al sistema.

- **Decisiones pertinentes a los usuarios:**
  - Se decidió armar una relación tipo-subtipo para las tablas usuario, cliente y proveedor dado que los clientes, administradores y proveedores tienen datos en común.
  - Decidimos que los usuarios tengan un atributo tipo_usuario que los define como entidad. Se decidió que el tipo_usuario no cambie pero el rol si puede ser modificado. Leer apartado Rol / Tipo de Usuario en decisiones estratégicas.
  - El nombre de usuario es armado siguiendo el siguiente patrón:

    Clientes : u+[DNI del cliente]
    Proveedores: u+[CUIT del proveedor]

- Utilizamos un password por defecto para todos los usuarios migrados, dándoles la posibilidad de cambiarlo cuando los mismos deseen:

Password: pass123 Password admin: w23e

- **Decisiones pertinentes a la carga de crédito:**
  Se decidió, para la carga de crédito, que los clientes migrados que cargaron con tarjeta de crédito tienen por defecto los datos de su tarjeta seteados en null.

Apartado: Rol / Tipo de Usuario

La decisión tomada por el grupo, de manera de cumplir con la funcionalidad y los requerimientos solicitados, y a su vez lograr consistencia y el buen funcionamiento del sistema fue la siguiente:

Se cuenta por un lado con el “rol”, y por el otro el llamado “tipo de usuario”. Dentro de la tabla usuario se tiene una columna tipo de usuario, que puede ser administrador, proveedor o cliente.

En el momento de la creación del usuario se define como cliente o proveedor, teniendo que llenar la información correspondiente en cada caso y agregando el tipo de usuario correspondiente en la tabla “Usuario”. Para cada tipo, el rol por defecto es el mismo que el nombre del tipo de usuario, para ejemplificar en un caso concreto: si alguien crea un cliente, va a tener por defecto el rol de cliente y el tipo de usuario cliente. Si el administrador le cambia el rol a proveedor le van a aparecer los botones (en el menú) de las funcionalidades de proveedor pero va a seguir siendo un usuario de tipo cliente.

De esta manera, para cada funcionalidad se chequea el tipo de usuario (no el rol), para ver si se tiene acceso.

Con esto nos garantizamos que un cliente no pueda armar cupones ni tampoco que un proveedor compre cupones.

Las demás funcionalidades como Ej. Listado Estadístico, ABM cliente, ABM Proveedor, etc. Pueden ser asignadas a un nuevo rol y si este es asignado a un cliente o a un proveedor , podrán ser utilizadas sin problemas.

Para ejemplificar: si se crea un cliente y se le asigna el rol proveedor, cuando ingrese al sistema le va a aparecer el botón “Armar Cupón”, porque es una funcionalidad adecuada al rol de proveedor, pero al ejecutar esta funcionalidad se procede a la validación contra el tipo de usuario. El cliente tiene el rol para ver el botón de esa funcionalidad, pero el tipo de usuario cliente, no permite utilizar la funcionalidad de armar cupón.

Aplicación desktop

Aclaraciones previas

- El archivo de configuración es app.config, su ruta es “src/GrouponDesktop/App.config”:
- La fecha del sistema (editable en el app.config) se guarda con una clave en el formato

  ```
  <add key="fecha" value="02/01/2009" />
  ```

  Para editar la fecha, cambiar el valor indicado en rojo por el valor deseado en formato **dd/MM/yyyy.**

- Debido al diseño de la aplicación utilizamos 2 claves más llamadas “valoresGiftCard” y “valoresCredito”, estas son internas de la aplicación, valoresGiftCards contiene los valores de la ventana “Comprar Gift Card” y valoresCredito contiene los valores enteros de la ventana “Cargar Crédito”.

Decidimos aprovechar las características de la tecnología que estamos utilizando, por lo tanto encaramos un diseño orientado a objetos. El sistema puede resumirse en clases de negocio, clases de mapeo SQL-objetos y clases de interacción directa con el SQL.

Para la parte del negocio decidimos respetar los nombres de los atributos del SQL, para así poder mapear uno a uno propiedades de C# con campos de SQL. Prácticamente no hay lógica en las clases, actúan esencialmente como DTOs.

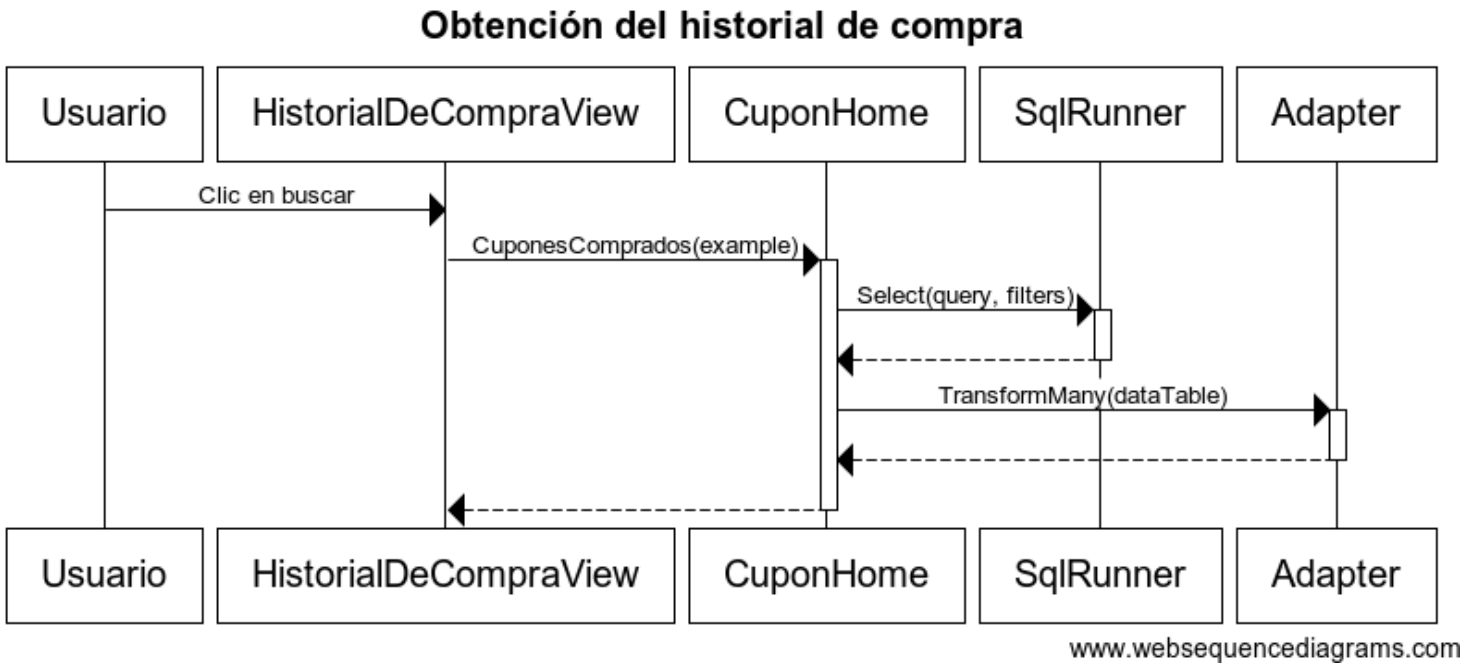
Cada vista tiene entonces un modelo asociado, contra el cual se bindean los distintos fields. En el caso de los listados, además del objeto que representa al ejemplo (una especie de query by example) también hay una IList de esos objetos, que representan los resultados de la última búsqueda.

En cuanto a la transformación de relacional a objetos, optamos por implementar un mini-ORM utilizando las capacidades reflexivas que nos provee el lenguaje, genérico para poder utilizarlo a lo largo de toda la aplicación sin tener que configurar nada. Básicamente busca propiedades que se llamen igual que los campos y las rellena; en el otro sentido, crea un SqlParameter por cada propiedad del objeto.

A la hora de persistir o recuperar datos persistidos nos manejamos con distintas Homes, donde cada una expone los mensajes necesarios según las operaciones que puede realizar. La función de estos objetos es realizar el mapping, ejecutar los comandos en la BBDD y luego mapear la respuesta cuando corresponda.

El listado de porcentaje de devolución lo ordenamos de manera decreciente por porcentaje de devolución.
El listado de giftcards asignadas lo ordenamos decrecientemente por cantidad de giftcards recibidas.

A modo de ejemplo integrador de lo explicado se incluye el siguiente diagrama de secuencia, que corresponde a la implementación de la funcionalidad “Historial de compra de cupones”



Al registrar un consumo de cupón, decidimos que la fecha a guardar es la fecha en la que se está registrando el cupón (fecha de archivo de configuración)