TRABAJO PRACTICO

GESTION DE DATOS
PRIMER CUATRIMESTRE
2017

Curso: K3114

Nombre del grupo: CRAZYDRIVER

Integrantes:

Nombre: Ruben Maier Enzler
Legajo: 1506420
Email responsable: ruben.maier@hotmail.com

Nombre: Norman Perrin
Legajo: 1472150

Nombre: Santiago Nicolás Maresco
Legajo: 1522790

Nombre: Nicolas Matías Pfeger
Legajo: 1402547

Criterios tomados en la migración:

Usuarios:
Consideramos que un usuario será todo aquel que tenga acceso al sistema, en este caso hasta ahora son los choferes, clientes y administradores. Como en la base maestra existen choferes y clientes, a la hora de migrarlos les asignamos un usuario a cada uno con el username igual a su DNI (único) y la password la evaluamos por default igual a la contraseña definida para el administrador por requisito del enunciado del trabajo practico (w23e). Consideramos que la misma podría ser modificada a futuro si se implementara un sistema de configuración para el usuario donde podría modificar su perfil propiamente.

Doble Facturación:
Consideramos que hay facturas impagas y que anteriormente a que el sistema sea relevado, se pretendía manipular la información impaga y paga duplicando los datos.
Query para validar que hay duplicados:

```
SELECT Auto_Patente, Chofer_Dni, Cliente_Dni, Viaje_Fecha, Viaje_Cant_Kilometros,
Turno_Descripcion, Turno_Precio_Base, Turno_Valor_Kilometro, Factura_Nro,
Rendicion_Nro, Rendicion_Importe, count(*)
FROM gd_esquema.Maestra
WHERE Factura_Nro IS NOT NULL AND Rendicion_Nro IS NULL
GROUP BY Auto_Patente, Chofer_Dni, Cliente_Dni, Viaje_Fecha,
Viaje_Cant_Kilometros, Turno_Descripcion, Turno_Precio_Base,
Turno_Valor_Kilometro, Factura_Nro, Rendicion_Nro, Rendicion_Importe
HAVING count(*) > 1
ORDER BY Chofer_Dni, Auto_Patente, Cliente_Dni, Viaje_Fecha, Turno_Descripcion,
Factura_Nro, Rendicion_Nro
```

Doble Rendición:
Consideramos que las rendiciones son únicas, pero sin embargo hay rendiciones duplicadas:
Query para validar que hay duplicados:

```
SELECT Auto_Patente, Chofer_Dni, Cliente_Dni, Viaje_Fecha, Viaje_Cant_Kilometros,
Turno_Descripcion, Turno_Precio_Base, Turno_Valor_Kilometro, Factura_Nro,
Rendicion_Nro, Rendicion_Importe, count(*)
FROM gd_esquema.Maestra
WHERE Factura_Nro IS NULL AND Rendicion_Nro IS NOT NULL
GROUP BY Auto_Patente, Chofer_Dni, Cliente_Dni, Viaje_Fecha,
Viaje_Cant_Kilometros, Turno_Descripcion, Turno_Precio_Base,
Turno_Valor_Kilometro, Factura_Nro, Rendicion_Nro, Rendicion_Importe
HAVING count(*) > 1
ORDER BY Chofer_Dni, Auto_Patente, Cliente_Dni, Viaje_Fecha, Turno_Descripcion,
Factura_Nro, Rendicion_Nro
```

Ejemplo:

```
select * from gd_esquema.Maestra where Chofer_Dni = 48584942 and Cliente_Dni =
18911370 and Rendicion_Nro = 22937
```

Pero nótese que para cada doble rendición existen 4 facturaciones y a su vez 2 viajes idénticos sin facturación ni rendición, por lo que consideramos que son dos viajes distintos.

```
select Auto_Patente, Chofer_Dni, Cliente_Dni, Turno_Descripcion,
Viaje_Cant_Kilometros, Viaje_Fecha, Rendicion_Fecha, Rendicion_Importe,
Rendicion_Nro, Factura_Fecha, Factura_Fecha_Fin, Factura_Fecha_Inicio,
Factura_Nro
from gd_esquema.Maestra m
where exists
(
```

```
	select
Cliente_Dni,Chofer_Dni,Auto_Patente,Turno_Descripcion,Rendicion_Nro,Rendicion_Fec
ha,Rendicion_Importe,Viaje_Fecha,Viaje_Cant_Kilometros, count(Rendicion_Nro)
cantidad
	from gd_esquema.Maestra
	where Rendicion_Nro is not null
	and Cliente_Dni = m.Cliente_Dni and Chofer_Dni = m.Chofer_Dni and
Auto_Patente = m.Auto_Patente and Turno_Descripcion = m.Turno_Descripcion
	and Viaje_Fecha = m.Viaje_Fecha and Viaje_Cant_Kilometros =
m.Viaje_Cant_Kilometros
	group by
Cliente_Dni,Chofer_Dni,Auto_Patente,Turno_Descripcion,Rendicion_Nro,Rendicion_Fec
ha,Rendicion_Importe,Viaje_Fecha,Viaje_Cant_Kilometros
	having count(rendicion_nro) > 1
)
order by Auto_Patente, Chofer_Dni, Cliente_Dni, Turno_Descripcion,
Viaje_Cant_Kilometros, Viaje_Fecha, Rendicion_Fecha, Rendicion_Importe,
Rendicion_Nro, Factura_Fecha,Factura_Fecha_Fin, Factura_Fecha_Inicio,
Factura_Nro;
```

Nótese que también hay rendiciones duplicadas, pero tienen solo 2 facturaciones, por lo que consideramos que estos casos son facturaciones impagas

Fecha en los viajes:
Basados en los requisitos que el enunciado solicita a la hora de registrar un viaje pidiéndonos ingresar una fecha de inicio y una fecha de fin del viaje, tomamos como condición de migración utilizar la “fecha” que figuraba en la tabla maestra como “fecha_inicio” y frente a la eventualidad de no dejar valores nulos en la hora fin también duplicamos este dato ingresando el valor de la columna “fecha” de la tabla maestra en nuestra columna “fecha_fin”.

Facturaciones:
Consideramos que las facturaciones se hacen 1 vez por mes para los clientes, pero permitimos que si el administrador decide se pueda facturar a un cliente antes de que termine el mes, aunque si esto ocurre este cliente no podrá realizar viajes hasta no haber finalizado el mes.

Rendiciones:
Consideramos que las rendiciones se efectúan una sola vez por día y al cierre del día, es decir se permite realizar la rendición a partir de las 0 horas, que es el comienzo del día siguiente. Por este motivo, implica a su vez que el chofer solo pueda trabajar en un único turno, aunque el sistema en sí, está modelado para poder llegar a soportar la posibilidad de poder tener choferes que hagan más de un turno de ser necesario.

Usuarios del sistema:
Para los administradores definimos que tengan permiso/acceso a todas las funcionalidades, pero para los clientes y choferes solo definimos algunas funcionalidades randoms como para validar que funcionan, pero al no tener un requerimiento de esta condición que nos especifique que funcionalidades puede realizar cada tipo de usuarios, tomamos un criterio propio a fin de poder realizar un testeo de su funcionamiento.

Criterio de programación:

- Se decidió trabajar con una arquitectura de capas para facilitar la detección de errores separando así el programa en:

    - **Capa datos:** encargada de prestar servicio a la capa negocio y ser la única capaz de interactuar con la base de datos de manera directa.
    - **Capa negocio:** cumple la función de intermediario entre las necesidades de la interfaz y la solicitud de peticiones a la base de datos.
    - **Capa interfaz:** es la encargada de brindarle servicios solicitados a las distintas ventanas que el sistema implementa. La capa interfaz cumple la función de separar código de lógica de interfaz con respecto a código propiamente de ventanas que sí interactúan estrechamente con el usuario.
    - **Entidades:** es una capa que cumple la función puramente de evitar repetir código innecesario facilitar así la programación, en especial la intercomunicación entre las distintas capas mencionadas anteriormente que solicitan datos que cumplen con cierta "lógica común" o "patrón". Abstrae las condiciones de las entidades de la base de datos permitiéndonos así poder aplicar criterios de programación orientada a objetos.
    - **UBER FRBA:** Contiene las distintas ventanas con las que el usuario va a interactuar con el programa.
- Tomamos el valor -1 como consideración de "invalidez", es decir, dato o valor erróneo.