Universidad Tecnológica Nacional

Facultad Regional Buenos Aires

Gestión de Datos

Trabajo Práctico - UBER

(1C2017)

Entrega N°1

Fecha: 10/06/2017

Grupo: MAIDEN (41)

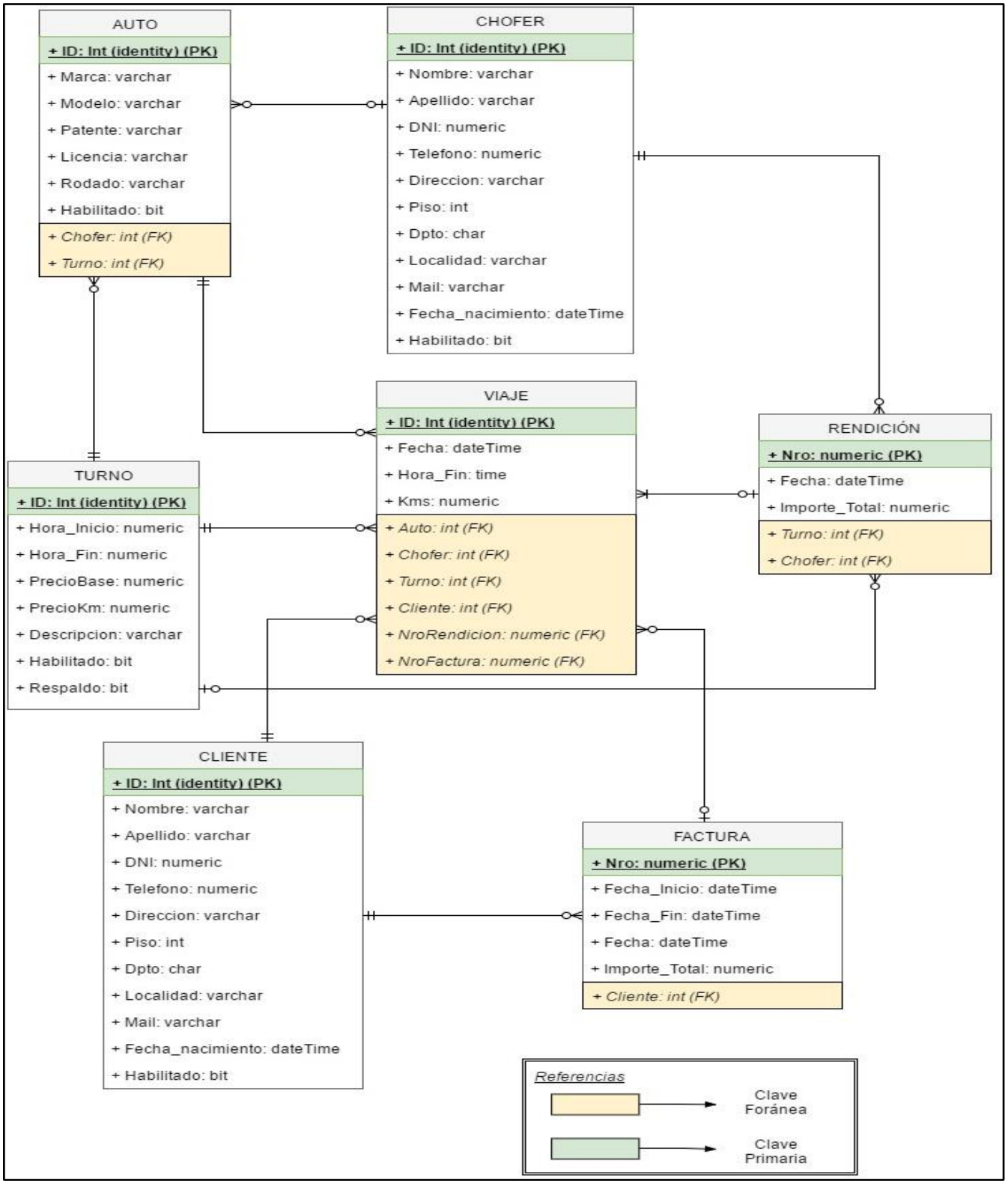
Integrantes

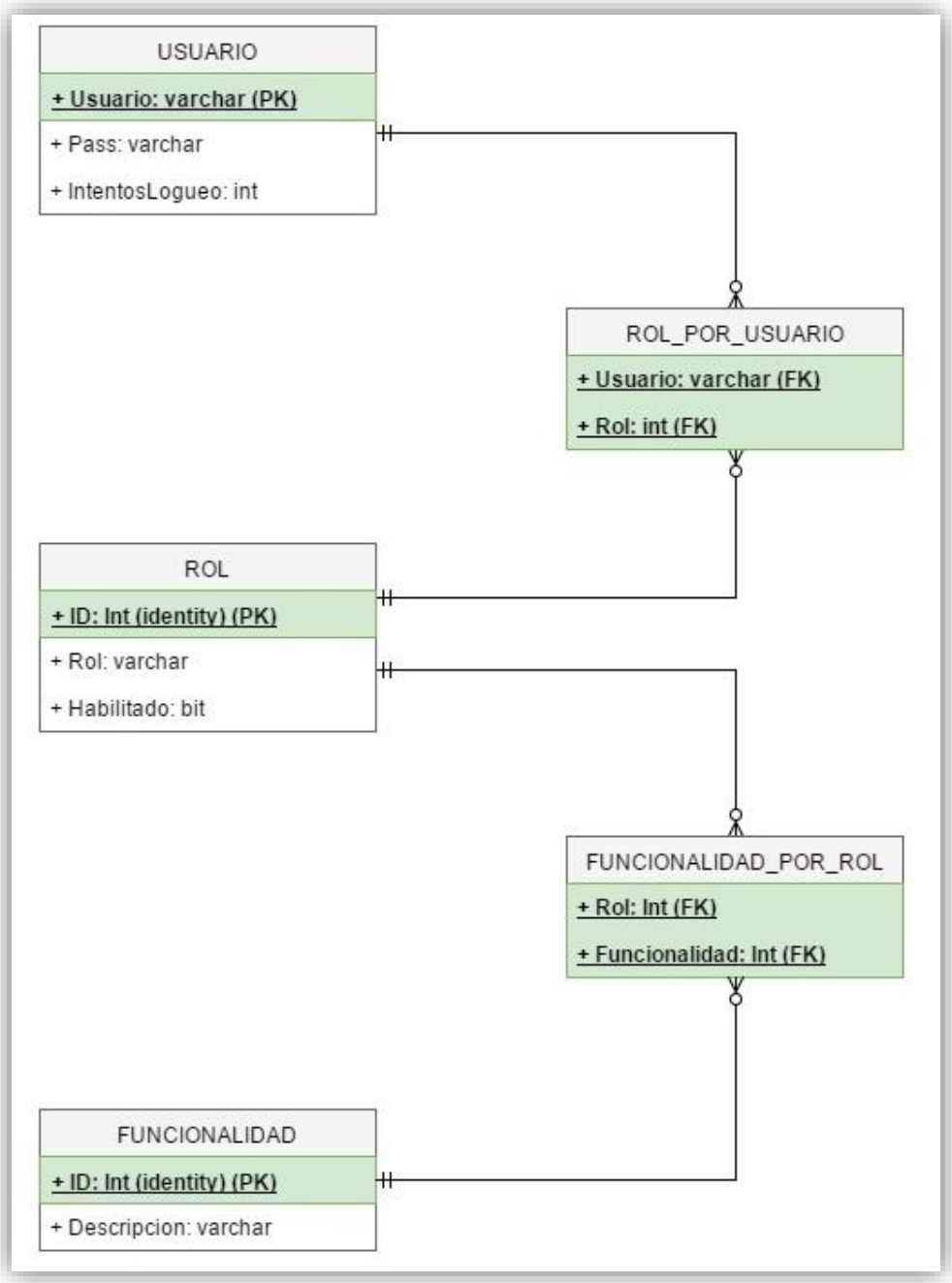
Alumno	Legajo	Curso
Aguerrido Joaquín	152.105-6	K3014
Pirotte Daniela	152.318-1	K3014
Pulleiro Gastón	143.970-4	K3052
Rombolá Julián	152.309-0	K3014

Contenido

Modelo de Datos

DER




USUARIO
+ Usuario: varchar (PK)
+ Pass: varchar
+ IntentosLogueo: int
ROL_POR_USUARIO
+ Usuario: varchar (FK)
+ Rol: int (FK)
ROL
+ ID: Int (identity) (PK)
+ Rol: varchar
+ Habilitado: bit
FUNCIONALIDAD_POR_ROL
+ Rol: Int (FK)
+ Funcionalidad: Int (FK)
FUNCIONALIDAD
+ ID: Int (identity) (PK)
+ Descripcion: varchar

Entidades

- **USUARIO:** Esta entidad representa los distintos usuarios que podrán acceder a las funcionalidades del Sistema.

Cuenta con los siguientes campos:
- **Usuario** (varchar) **(PK)*
- Pass (varchar)
- Intentos de Logueo (Int)

El campo **Usuario** se refiere al nombre del usuario con el que los mismos se loguearán al sistema. No podrán existir distintos usuarios con un mismo nombre, por lo que utilizamos este campo como *Primary Key*.
El campo **Pass** almacena la contraseña del usuario, la cual es encriptada desde la base de datos mediante el algoritmo SHA256.
Dado que los usuarios deberán bloquearse luego de 3 intentos fallidos, utilizamos el campo **Intentos de Logueo** para controlar dicha condición. Si el valor de dicho campo fuese 3, entonces no podrá ingresar al sistema.
Cada usuario podrá tener distintos roles los cuales le brindarán acceso a distintas funcionalidades.

- **ROL_POR_USUARIO**: Dado que un usuario puede tener muchos roles, y un rol puede corresponder a varios usuarios (relación muchos-muchos) es necesaria esta tabla intermedia.

Cuenta con los siguientes campos:
- **Usuario** **(FK)*
- **Rol** **(FK)*

El campo **Usuario** es una *Foreign Key* que hace referencia a un usuario, y el campo **Rol** es otra *Foreign Key* que hace referencia a un rol.
A su vez, ambos campos en conjunto forman la *Primary Key* de esta tabla ya que ningún usuario puede tener repetidos sus roles.

- **ROL:** Los registros de esta tabla hacen referencia a los distintos roles que podrán tener los usuarios del sistema.

Cuenta con los siguientes campos:
- **ID** (Identity - Int) **(PK)*
- Nombre (varchar)
- Habilitado (bit)

Dado que el **Nombre** del rol es modificable decidimos utilizar un **ID** de tipo Identity como *Primary Key* ya que, en caso de utilizarse este campo (Nombre) como clave, si el mismo fuese modificado, podrían generarse inconsistencias en los datos.

Los roles podrán ser asignados y utilizados según se encuentren activos o no en el sistema. Por ello, utilizamos el campo **Habilitado** que representa dicho estado. (1 = Habilitado, 0 = Deshabilitado).

- **FUNCIONALIDAD_POR_ROL**: Dado que un rol puede tener muchas funcionalidades y una funcionalidad puede corresponder a varios roles (relación muchos-muchos) es necesaria esta tabla intermedia que permita relacionar esas dos tablas.

Cuenta con los siguientes campos:

- **Rol** (Int) *(FK)
- **Funcionalidad** (Int) *(FK)

El campo **Rol** es una *Foreign Key* que hace referencia a la tabla Rol, y el campo **Funcionalidad** es una *Foreign Key* que hace referencia a la tabla Funcionalidad.
A su vez ambos campos forman la *Primary Key* de esta tabla, dado que un rol no puede tener repetidas sus funcionalidades.

- **FUNCIONALIDAD**: Los registros de esta tabla hacen referencia a las distintas funcionalidades del sistema.

Cuenta con los siguientes campos:

- **ID** (Identity - Int) **(PK)*
- Descripción (varchar)

Cada **ID** representa una funcionalidad distinta:
1 = ABM Clientes
2 = ABM Choferes
3 = ABM Autos
4 = ABM Roles
5 = ABM Turnos
6 = Registro Viajes
7 = Facturación a clientes
8 = Rendición a choferes
9 = Estadísticas

La aplicación Desktop dará acceso a las distintas Funcionalidades del sistema según cuales corresponden a cada Rol.
Por ejemplo, el rol Administrador contará con todas estas Funcionalidades. Por lo que el sistema le dará acceso a todas esas opciones.

- **CLIENTE**: Cada registro de la tabla representa a un Cliente del Sistema.

Cuenta con los siguientes campos:

- **ID** (identity - Int) **(PK)*
- Nombre (varchar)

- Apellido (varchar)
- DNI (numeric)
- Teléfono (numeric) *(Unique)
- Dirección (varchar)
- Piso (int)
- Dpto (char)
- Localidad (varchar)
- Mail (varchar) *(Null)
- Fecha de Nacimiento (dateTime)
- Habilitado (bit)

Si bien podría haberse indicado el **DNI** como *Primary Key* dado que el mismo es único para cada persona, el enunciado indica que el mismo es modificable. Esto podría generar inconsistencias en la información (por ejemplo, en el caso de que otra tabla hiciera referencia al cliente al que se le modificó el DNI), para ello habría que, además de modificar el DNI del cliente, actualizar todas las referencias existentes. Esto requeriría una gran cantidad de procesamiento y consultas, afectando negativamente la performance del sistema.
Por ello, con el objetivo de evitar esos inconvenientes, utilizamos un **ID** de tipo Identity como *Primary Key* que no podrá ser modificado por los usuarios y que será único para cada Cliente.
De acuerdo al dominio, el **Teléfono** deberá ser único para cada Cliente, por lo que no podrán existir distintos clientes con un mismo número telefónico.
En esta tabla todos los campos son obligatorios a excepción del **Mail**, por lo que, a diferencia del resto, está columna aceptará valores nulos.
Dado que la baja de Clientes es lógica, utilizamos el campo **Habilitado** (bit), que representará el estado del Cliente.
El valor '1' indica que el cliente está habilitado, pudiendo realizar viajes. Mientras que el valor '0' indicará que el cliente se encuentra deshabilitado, sin posibilidad de realizar viajes.

- **CHOFER**: Cada registro de la tabla representa a un Chofer del Sistema.

Cuenta con los siguientes campos:
- **ID** (identity - Int) *(PK)
- Nombre (varchar)
- Apellido (varchar)
- DNI (numeric) *(Unique)
- Teléfono (numeric)
- Direccion (varchar)
- Piso (int)
- Dpto (char)
- Localidad (varchar)
- Mail (varchar)
- Fecha de Nacimiento (dateTime)
- Habilitado (bit)

En cuanto a la *Primary Key*, se utilizó el mismo criterio que para los Clientes. Es decir, un campo Identity para evitar problemas de consistencia de datos o de performance.

En este caso, se indica que el **DNI** debe ser *único* para cada chofer. Por lo que no podrán existir DNI duplicados en esta tabla.
En esta tabla ningún campo aceptará valores nulos dado que todos ellos son obligatorios.
Dado que la baja de Choferes es lógica, utilizamos el campo **Habilitado** (bit), que representará el estado del Chofer.
El valor '1' indica que el chofer está habilitado, pudiendo registrarse viajes a su nombre y realizarse la rendición de los mismos.
Mientras que el valor '0' indicará que el chofer se encuentra deshabilitado, sin dichas posibilidades.

- **AUTO:** Esta tabla almacena la información referida a los distintos vehículos del Sistema que podrán ser utilizados por los choferes.

Cuenta con los siguientes campos:
- **ID** (Identity - Int) **(PK)*
- Marca (varchar)
- Modelo (varchar)
- Patente (varchar) **(Unique)*
- Licencia (varchar)
- Rodado (varchar)
- Habilitado (bit)
- Chofer (Int) **(Unique, FK)*
- Turno (Int) **(FK)*

Todos los campos de la entidad Auto son modificables, por lo tanto utilizamos nuevamente un **ID** de tipo Identity como *Primary Key.*
En el sistema no pueden existir autos mellizos, es decir, con patentes duplicadas. Por lo tanto indicamos dicha columna como *Unique*.
Lo mismo ocurre con el **Chofer**, el cual, además de ser una *Foreign Key* a la tabla Chofer, únicamente puede estar asignado a un auto.
El campo **Turno** también es una *Foreign Key* que hace referencia al turno en el que el vehículo podrá realizar viajes.

- **TURNO:** Esta tabla representa los distintos turnos en los que los choferes y clientes podrán realizar viajes.

Cuenta con los siguientes campos:
- **ID** (Identity - Int) **(PK)*
- Hora Inicio (numeric)
- Hora Fin (numeric)
- Precio Base (numeric)
- Precio Km (numeric)
- Descripcion (varchar)
- Habilitado (bit)
- Respaldo (bit)

Todos los campos de esta estructura deben ser modificables. Por lo tanto utilizamos el campo **ID** de tipo Identity como *Primary Key* que permita hacer referencia a cada uno de los turnos (registros) existentes en el sistema.
Los choferes podrán realizar viajes siempre que el horario del mismo se encuentre dentro del intervalo de tiempo del turno al que se encuentra asignado. Es decir, entre la **Hora Inicio** y la **Hora Fin**.
Además, de acuerdo a los precios del turno, es decir, **Precio base** y **Precio KM** se realizarán las rendiciones a choferes y la facturación a clientes.
El campo **Respaldo** se utiliza para indicar que el turno existe únicamente para asegurar la consistencia de los datos y, por lo tanto, el usuario no tendrá acceso al mismo.
Esto es necesario en el caso de que los datos del turno se modifiquen y existieran referencias al mismo desde algún viaje registrado.
Por ejemplo, si el viaje fue realizado con un precio base de $10, y antes de facturarse se le cambiara el precio a $100, el monto de la factura no sería correcta ya que el precio real fue de $10. Por ello guardamos una copia del turno anterior, para que los viajes se mantengan consistentes.

- **VIAJE:** en esta tabla se almacena la información que representa los viajes realizados por los clientes y choferes.

Cuenta con los siguientes campos:

- **ID** (Identity - Int) **(PK)*
- Fecha (datetime)
- Hora Fin (time)
- Kms (numeric)
- Auto **(FK)*
- Chofer **(FK)*
- Turno **(FK)*
- Cliente **(FK)*
- Rendición **(FK)*
- Factura **(FK)*

Utilizamos el campo **ID** como *Primary Key* (identity) que represente cada uno de los viajes realizados.
En cuanto a cada viaje es necesario guardar la **Fecha** (y hora) en que se realizó y la **Hora Fin** del mismo.
También se necesita conocer la cantidad de **Kms** (kilómetros) del recorrido para que puedan realizarse las rendiciones y facturaciones correspondientes.
El campo **Auto** es una *Foreign Key* que hace referencia al auto con el que se realizó el viaje.
Es necesaria una referencia al **Cliente** para luego poder hacerse la Facturación, así como también una referencia al **Chofer** para poder realizarse la rendición del mismo.
Dado que una Factura hará referencia a varios Viajes, y un Viaje únicamente estará en una Factura (relación uno-muchos), agregamos una *Foreign Key* a **Facturas** en esta tabla para conocer a cuál factura corresponde cada viaje.
Lo mismo ocurre con las Rendiciones, por lo que se utilizó la misma idea. Se agregó una *Foreign Key* a **Rendición**.

*** En ambos casos (Rendición - Factura) podría haberse creado una tabla intermedia que genere una mejor abstracción (dado que tener dichas referencias en el viaje puede ser un poco confuso).*
Sin embargo, dichas tablas contendría mayor cantidad de registros que los viajes existentes (ya que tendría que existir uno en cada tabla para cada viaje) con la desventaja de agregar un grado más de indirección.
Por lo que priorizamos la performance, la simpleza por sobre la abstracción.

- **RENDICIÓN**: En esta tabla se guarda la información correspondiente a cada una de las rendiciones realizadas a cada chofer de forma diaria.

Cuenta con los siguientes campos:
- **Número** (Identity - numeric) *(PK)
- Fecha (datetime)
- Importe Total (numeric)
- Turno *(FK)
- Chofer *(FK)

Cada rendición tendrá un **Número** único que lo identifique, por lo que podrá ser utilizado como *Primary Key* de la tabla.
Únicamente podrá realizarse una rendición por día para cada chofer, por lo que será necesario guardar la **Fecha** de la misma y así evitar que la misma se realice varias veces.
Para esto, además, es necesario contar con una *Foreign Key* al **Chofer**.
También se requiere conocer el **Turno** en el que se encontraba trabajando el chofer al momento de realizarse.

- **FACTURA:** En esta tabla se guarda la información correspondiente a cada una de las facturas realizadas a cada cliente de forma mensual.

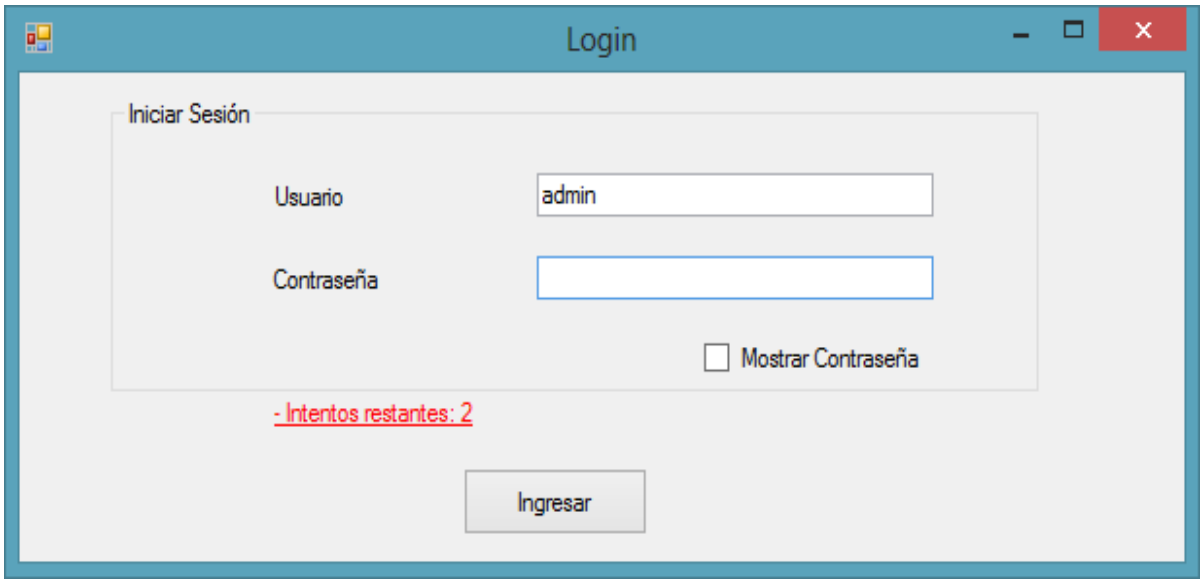
Cuenta con los siguientes campos:
- **Número** (Identity - numeric) **(PK)*
- Fecha (datetime)
- Fecha Inicio (datetime)
- Fecha Fin (datetime)
- Total (numeric)
- Cliente **(FK)*

Al igual que las rendiciones, cada factura tendrá un **Número** único que la identifique, el cual será utilizado como *Primary Key* de la tabla.
Se guardará la **Fecha** en que se realiza la factura, así como también la **Fecha Inicio** y **Fecha Fin** que contempla la misma. Es decir, se facturarán únicamente los viajes que fueron realizados en el intervalo de fechas indicado.
Dado que cada factura corresponde a un sólo **Cliente**, es necesario contar con una *Foreign Key* que haga referencia a dicha tabla y permita conocer el cliente en cuestión.

Programación

Aplicación Desktop (C#)

La aplicación brindará a los usuarios todas las funcionalidades solicitadas, utilizando para ello la base de datos creada.
Inicialmente el usuario deberá **loguearse** en el sistema. La aplicación consultará la base de datos, donde se controlará que el usuario y contraseña coincidan.
En caso de que eso no ocurra, se le descontarán intentos de logueo hasta que no le queden oportunidades y el usuario se bloqueará.
Luego, si los datos son correctos, la persona tendrá acceso a las distintas funcionalidades del sistema de acuerdo al rol del usuario. Si posee más de un rol, se deberá seleccionar uno de ellos para iniciar.

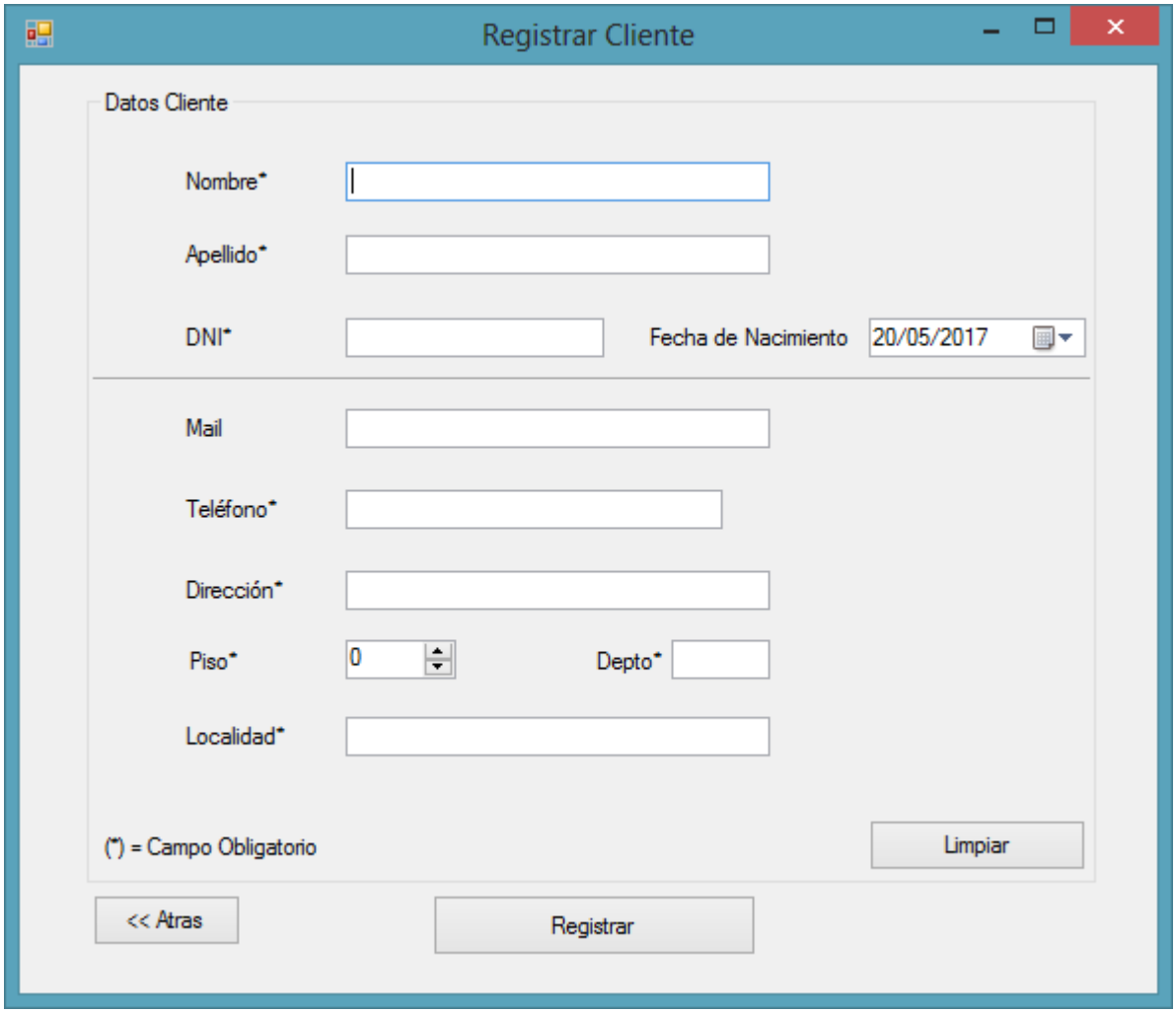


A continuación se mostrará un **menú inicial** con las opciones correspondientes.
Este menú inicial consulta a la base de datos las funcionalidades que posee el rol seleccionado previamente.
Dado que cada funcionalidad posee un número único que lo identifica (pk), el menú recibirá los ID's de cada una de ellas y eliminará los botones que no correspondan para ese rol.
Por ejemplo: si existen 7 funcionalidades, y el rol "Chofer" posee las [1, 2, 3, 4, 5,6], el menú eliminará el botón correspondientes a la funcionalidad 7, de forma que el usuario no podrá acceder a dicha función.

Los botones de Clientes, Choferes, Autos, Roles y Turnos permiten utilizar las funciones de Alta, Baja y Modificación de los datos correspondientes.

ABM

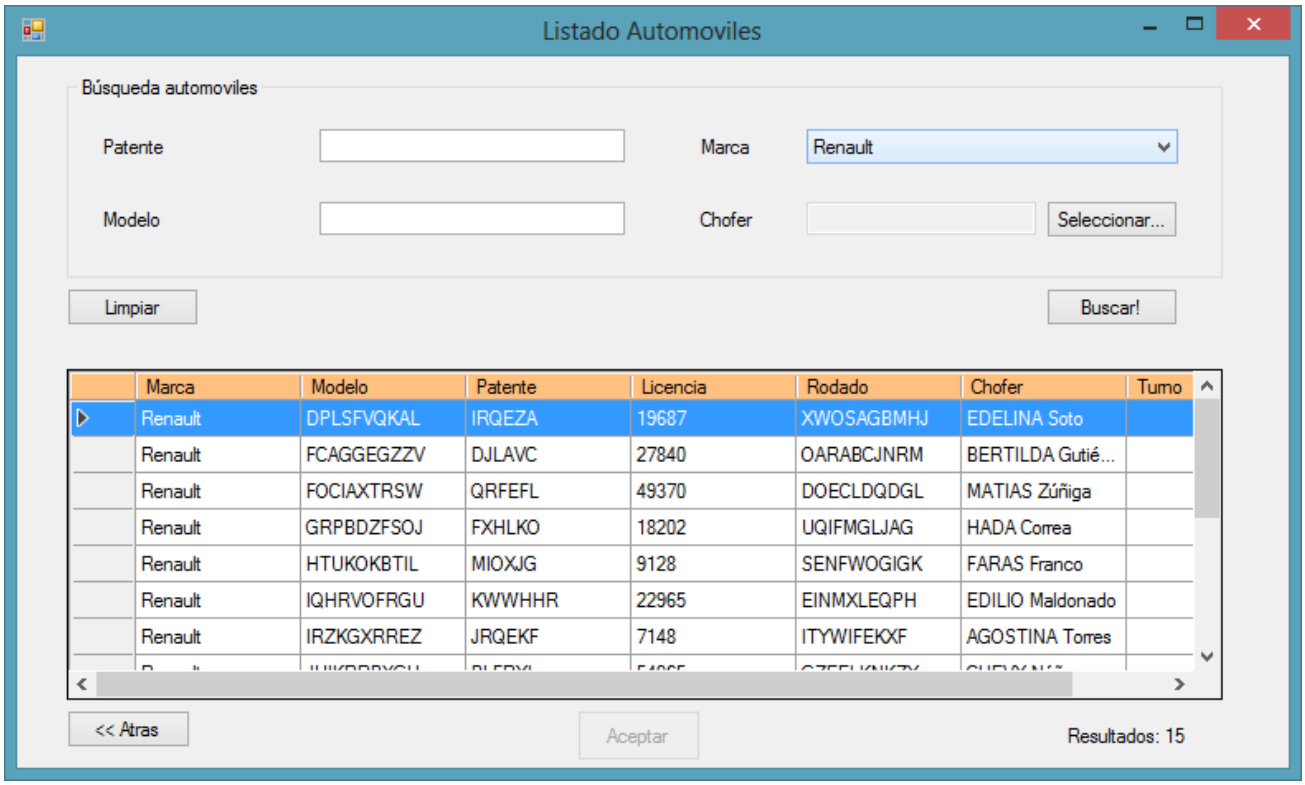
- **Alta**: estos formularios mostrarán todos los campos que el usuario deberá completar para poder registrar la entidad seleccionada.



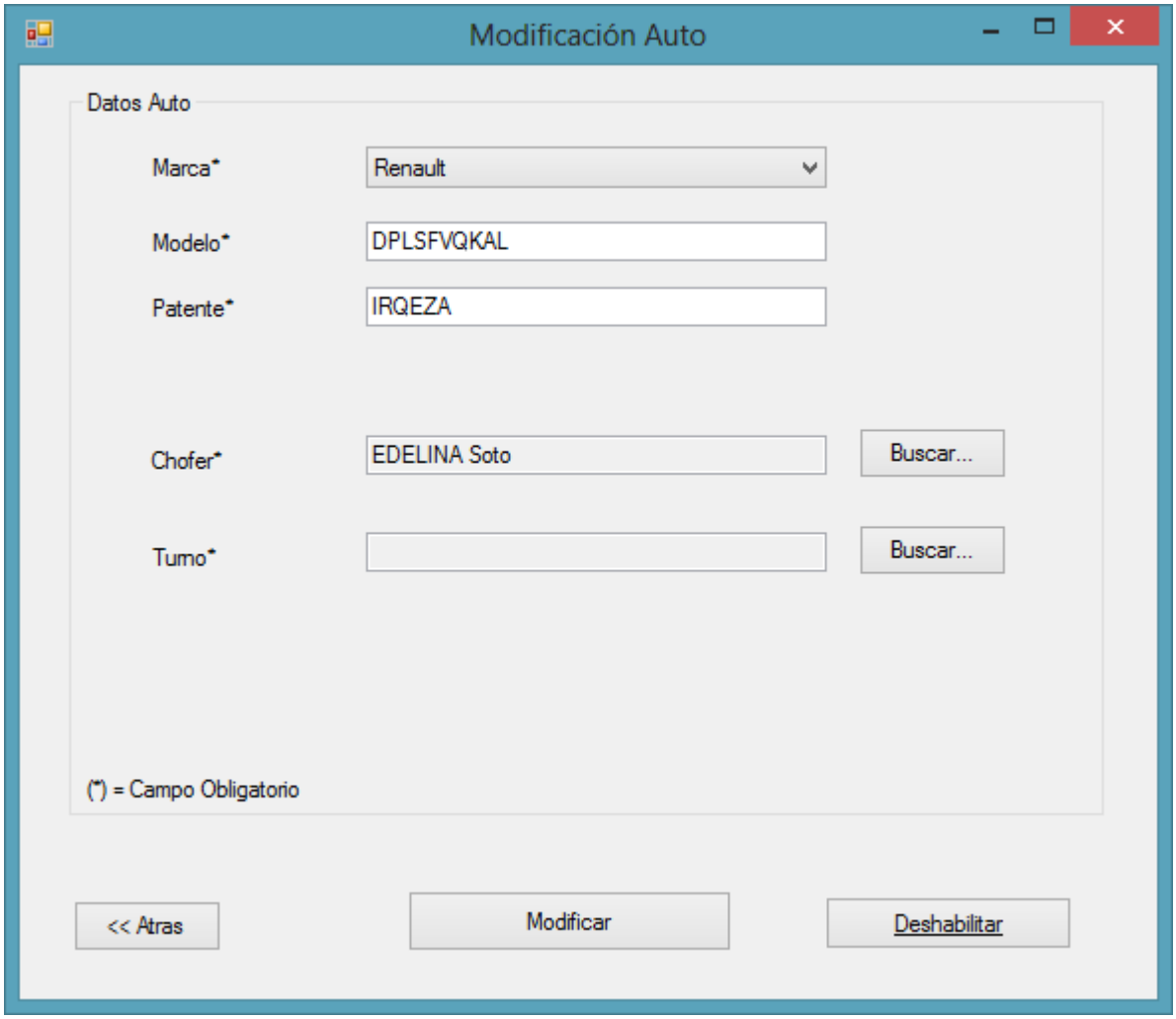
* *Ejemplo de alta de cliente*

- **Modificación / Baja**: el usuario podrá realizar búsquedas filtrando por determinados criterios.

Estos filtros no son excluyentes. Es decir, se obtendrán como resultados aquellos registros que cumplan alguna de las condiciones, no es necesario que cumpla con todas ellas.
Los resultados se muestran en un listado, en el cual se deberá seleccionar el registro a modificar o eliminar.

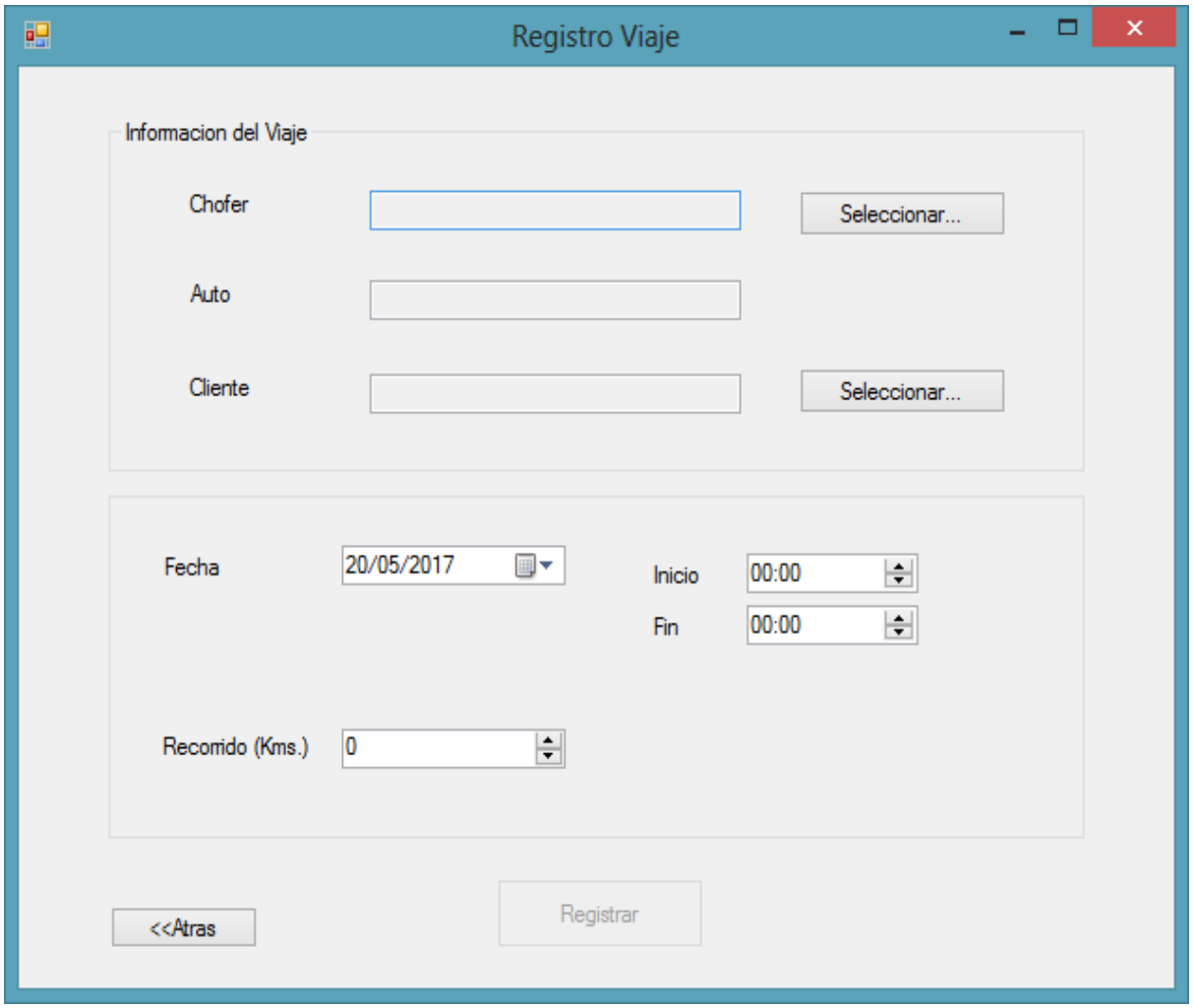


Una vez seleccionado, se mostrará un formulario similar al de alta pero con todos los campos completados con la información registrada actualmente para que el usuario pueda modificar. Aquí mismo se podrá deshabilitar el registro.


Modificación Auto
Datos Auto
Marca*
Renault
Modelo*
DPLSFVQKAL
Patente*
IRQEZA
Chofer*
EDELINA Soto
Buscar...
Turno*
Buscar...
(*) = Campo Obligatorio
<< Atras
Modificar
Deshabilitar

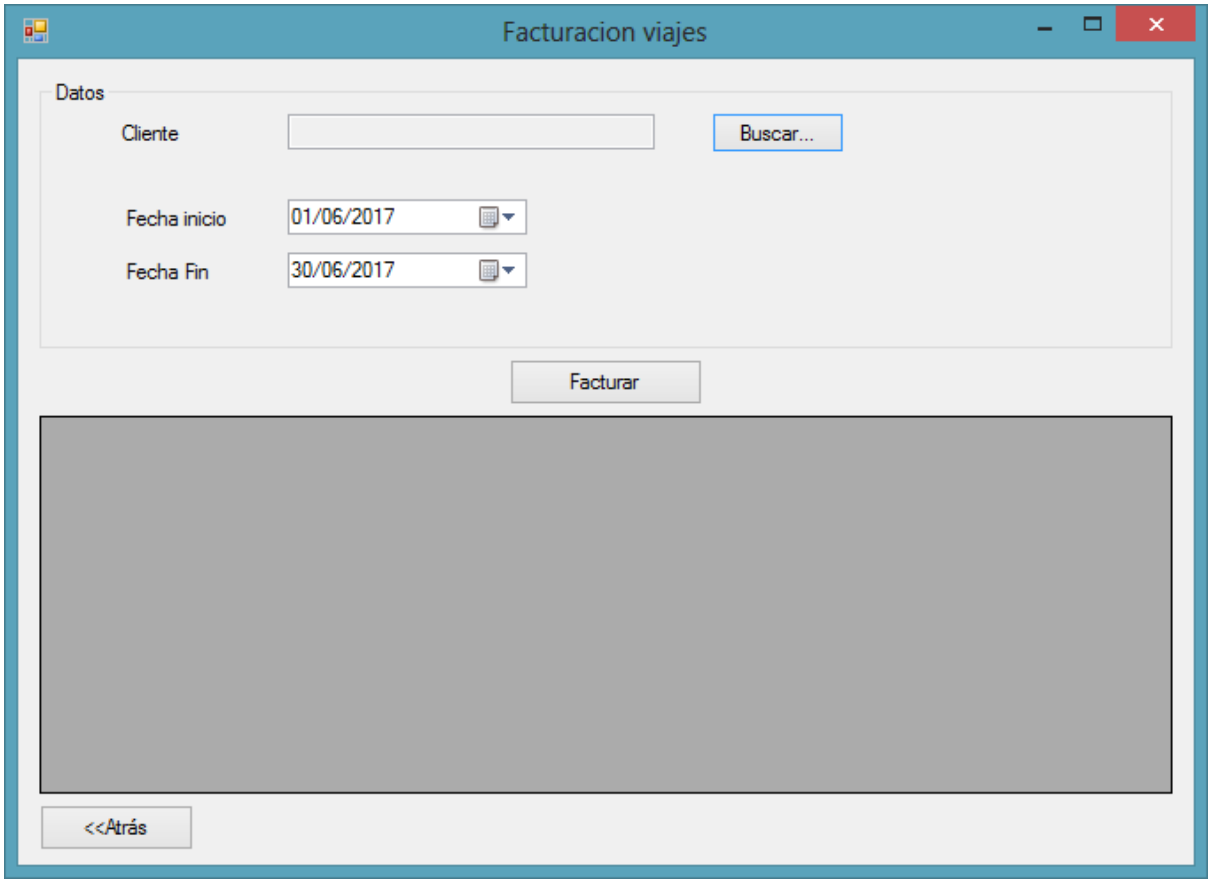
Viajes

Al registrar un viaje, el usuario deberá seleccionar los datos necesarios.
Luego de seleccionar el Chofer, el Auto se completará automáticamente con el que él posee actualmente.



Luego de controlar que los campos sean correctos, desde la base de datos se hará las validaciones correspondientes.
Es decir, que el viaje no se superponga con otros tanto del chofer como del cliente.

Facturación

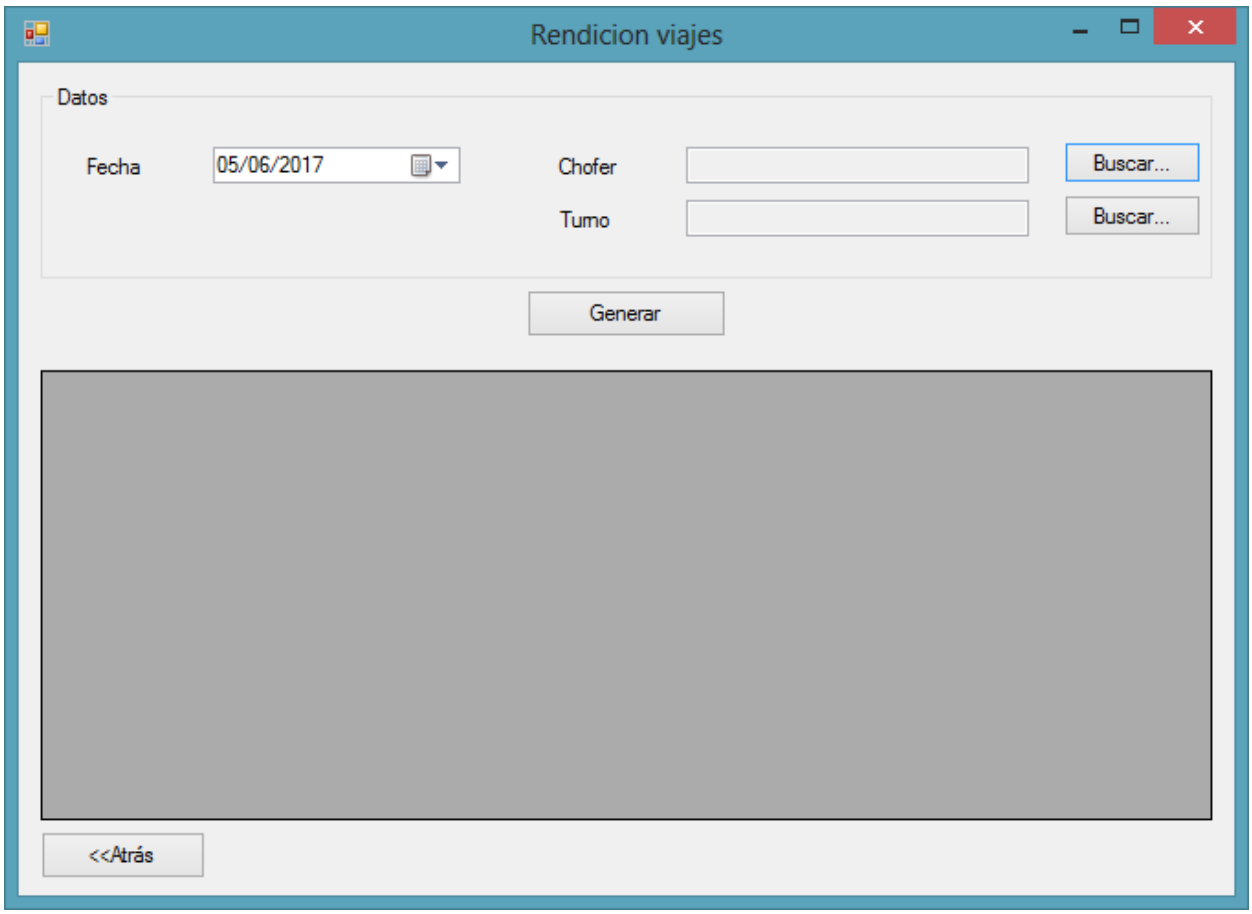


El usuario deberá ingresar el cliente al cual se realizará la facturación.
Dado que la facturación se realiza de forma mensual, al seleccionar alguna de las dos fechas posibles (inicio o fin), la otra se modificará automáticamente para cumplir con el requisito.

De esta forma, se facturarán todos los viajes del cliente comprendidos entre dichas fechas.
Para calcular el precio de cada viaje se accederá a los datos del mismo y, en base a la cantidad de kilómetros y a los precios del turno, se calcula el precio de cada uno de ellos. Luego, sumando estos precios se obtiene el importe total de la factura.

Además, si ya se había realizado la facturación en ese mes, el sistema lo informará al usuario e impedirá que la misma vuelva a realizarse. Sin embargo, el usuario podrá visualizar la información de dicha factura existente.

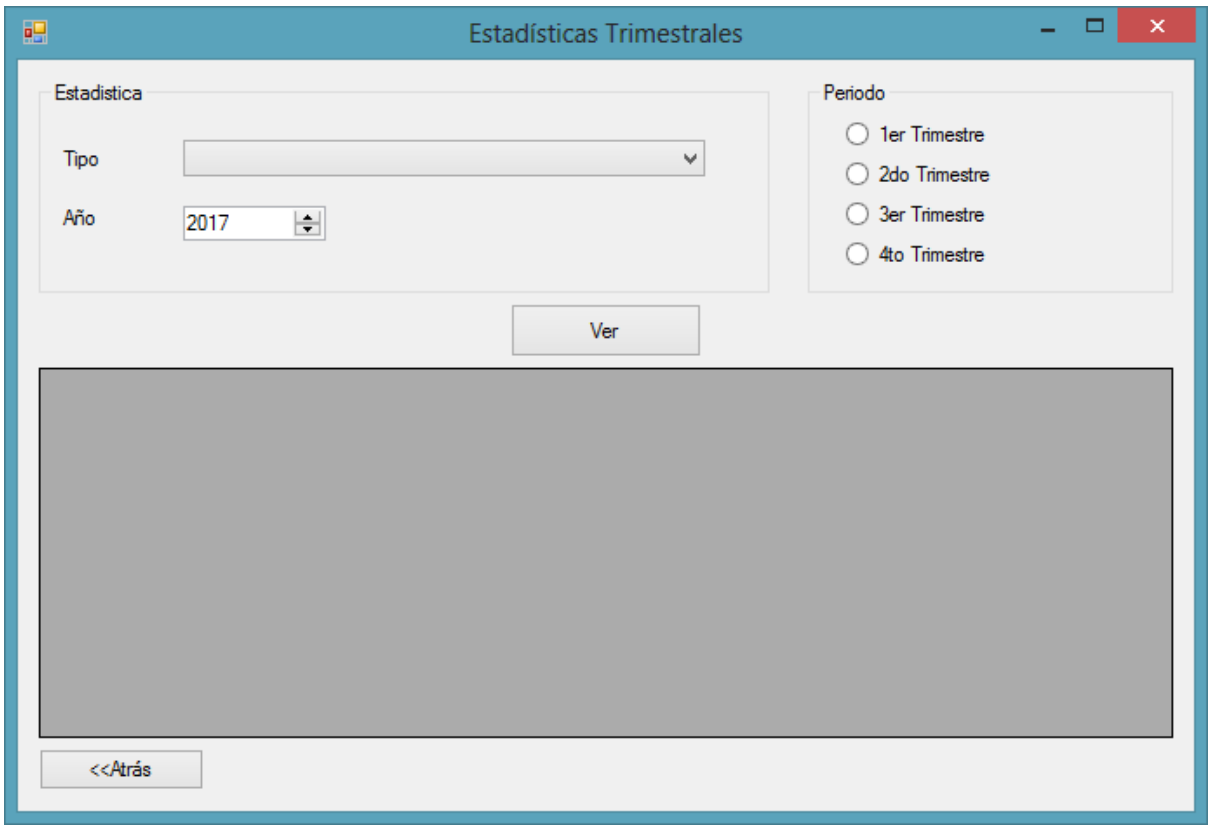
Rendición



El usuario deberá seleccionar el chofer, la fecha y el turno que se desea realizar la rendición.
El sistema verificará que la rendición para ese chofer en ese día aún no haya sido realizada. Caso contrario lo informará al usuario e impedirá que la misma se lleve a cabo, brindando la posibilidad de visualizar los datos existentes.

Para realizar la rendición se consultarán todos los viajes realizados por el chofer en esa fecha y, en base a la cantidad de kilómetros y precios del turno se calculará el precio del viaje (similar a la facturación de los clientes). Luego se le aplicará el porcentaje que corresponde a las rendiciones (30%) y así se obtendrá el importe total del chofer.

Estadísticas



En este formulario el usuario deberá seleccionar el tipo de estadística que desea obtener junto con el año y el trimestre a consultar.
Luego, el formulario ejecuta una función almacenada en la base de datos y carga el listado con los resultados obtenidos.

Consideraciones Importantes (C#)

1. Ya que tanto para realizar altas como modificaciones (de una misma entidad) se requieren los mismos campos, decidimos utilizar un sólo formulario para ambos casos.

Inicialmente dicho formulario está preparado para realizar altas, con los campos y botones definidos.
Cuando se desea realizar una modificación, los listados (luego de seleccionar un registro a modificar), utilizan métodos de este formulario para configurarlo y cargar los datos correspondientes.
Por lo tanto, si este método no se utiliza entonces esto indica que se realizará un alta.

2. Todos los formularios de la aplicación comparten ciertas características.

Por ejemplo, el hecho de realizar consultas a la base de datos y, con los resultados obtenidos, completar listados. También contarán con formularios siguientes o anteriores a los que deberán referirse luego de realizar determinadas acciones.
Es por esto que creamos una clase **FormsAdapter**, de la que todos los formularios heredarán comportamiento.

3. Para realizar el envío de información entre formularios creamos clases específicas para cada tipo de entidad necesarias: **Auto**, **Turnos**, **Persona** (utilizable tanto para clientes como choferes ya que comparten los mismos datos).

**Por ejemplo: si el usuario desea modificar un Cliente, deberá seleccionarlo de un listado. Este listado ya posee la información de los clientes, entonces completa un objeto de tipo 'Cliente' con dicha información (data object) y lo envía al formulario de altas/modificaciones para que este complete los campos con esos datos. De esta forma evitamos tener que realizar otra consulta a la base de datos para obtener nuevamente información que la aplicación ya había consultado (o sea, los datos del cliente).*

4. Para garantizar el polimorfismo en determinados métodos tuvimos que crear una interfaz **IDominio** implementada por las clases (data objects) mencionadas anteriormente ya que, por ejemplo, varios formularios implementan el método "Configurar", pero cada uno de ellos recibe distintas clases.

La modificación de autos recibirá un objeto de tipo Auto y la modificación de cliente recibirá uno de tipo Persona. Por lo tanto, para que dicho método pueda ser utilizado por ambos formularios, recibirá como parámetro un "IDominio" y luego adaptado por cada uno.

5. La conexión con la base de datos se establece mediante el objeto **Buscador** (Singleton).

Además, este objeto se utiliza para la configuración general de los objetos necesarios para realizar consultas (*SqlCommands*).

En caso de existir alguna falla en la configuración de los SqlCommands, la misma estará en esta única clase en lugar de dispersa por todo el sistema, por lo que será más fácil detectarla y corregirla. Otorgando así mayor robustez a la aplicación ya que existe un único punto de falla.

6. En el menú de Roles, también existe la funcionalidad para crear **Usuarios**.

7. Las marcas de los autos disponibles (selección acotada) se obtienen de la tabla maestra.

8. La aplicación configurará automáticamente todos los campos donde se deban ingresar fechas con la fecha indicada en el **archivo de configuración** (*App.config*). Esta fecha se tomará como la actual del sistema, por lo que no se podrán ingresar fechas posteriores a la misma.

Base de Datos

Todas las operaciones sobre la base de datos se encuentran programadas en la misma.
La aplicación desktop solamente se relaciona con la base mediante procedimientos almacenados y funciones.

Procedimientos Almacenados

ABM

Todas las operaciones de **Altas**, **Bajas** (deshabilitar/habilitar) y **Modificaciones** de datos se realizan mediante procedimientos almacenados.
Sus nombres fueron especificados como:

- SP_alta[:*Entidad*]
- SP_modif[:*Entidad*]
- SP_deshabilitar[:*Entidad*]
- SP_habilitar[:*Entidad*]

Pudiendo ser *Entidad*: Chofer, Cliente, Turno, Viaje, Auto, Rol, Usuario.
** Ejemplo*: SP_altaChofer, SP_modifChofer, SP_deshabilitarAuto, SP_habilitarCliente.

Login

Cuando el usuario intenta loguearse, se utiliza el procedimiento SP_login.
En el mismo se verifica que la contraseña sea correcta y, además, que la cantidad de logueos restantes del usuario sea menor a 3.
Si la cantidad de intentos registrados es 3, entonces el usuario se considera bloqueado y no podrá ingresar. Caso contrario, si la contraseña es correcta, la cantidad de intentos volverá a ser 0. Si la contraseña es errónea entonces se le sumará un intento fallido.

Usuarios - Roles

Dado que un usuario puede estar relacionado con distintos roles, al asignar un rol al usuario (SP_asignarRol), el procedimiento se encarga de crear el registro correspondiente en la tabla intermedia Rol_por_Usuario Quedando así relacionado cada usuario con sus diferentes roles.

Viajes

Al registrar un viaje, el procedimiento (SP_altaViaje) validará:

- Que el chofer no tenga otro viaje registrado en dicho horario,
- Que el cliente no tenga otro viaje registrado en dicho horario,
- Que exista turno en dicho horario y que el mismo coincida con el que posee el auto indicado.

Si los datos son correctos, se almacena el viaje. En caso de que no coincidan, el procedimiento lanza la excepción correspondiente, que luego será atrapada por la aplicación.

Rendición

Al realizar la rendición de los viajes, el procedimiento (SP_Rendicion) controla si ya fue realizada al chofer en el turno y día indicados.
Si no existe entonces se registra la rendición, se actualizan los viajes correspondientes insertando el nro. de rendición y luego se devuelven los datos.
Si la rendición ya se encuentra registrada, lanza una excepción con el número de rendición de acuerdo a los datos indicados (chofer, turno, día) para que, si el usuario lo desea, puedan visualizarse los mismos sin volver a realizar la rendición.
Para esto, el sistema consulta los viajes registrados correspondientes con los parámetros ingresados, calculando el precio de cada uno de ellos en base a los precios del turno.
Luego aplica el porcentaje (30%) para calcular el importe destinado al chofer.

Facturación

Al realizar la facturación de los viajes, el procedimiento (SP_Facturacion) controla si la factura al cliente en el mes indicado ya había sido realizada.
Si no existía entonces se registra la factura, actualizando el número de factura en los viajes correspondientes y luego se devuelven los datos.
Para eso el sistema consulta los viajes registrados de acuerdo a los parámetros ingresados (cliente, fecha inicio y fecha fin) y, con los datos del turno, calcula el precio de cada viaje.
Si la factura ya se encontraba registrada, lanza una excepción con el número de factura para que, si el usuario lo desea, puedan visualizarse los mismos sin volver a registrarse la misma.

Funciones

Todas las consultas desde la aplicación desktop se hacen por medio de funciones almacenadas en la base de datos.

Filtros

Todas las búsquedas de los listados se realizan por medio de funciones (fx_filtrar[:entidad]).
Por ejemplo: fx_filtrarChoferes.
Estas funciones devuelven tablas con los resultados de la consulta realizada.
Se devolverán aquellos registros que cumplan con alguna de las condiciones, sin ser necesario que cumplan con todas ellas.

Rendición

En cuanto a las rendiciones existen 2 funciones que se utilizan con distintos propósitos.
> *fx_getDatosRendicion*: Esta función se utiliza al momento de realizar la rendición (invocada desde el stored procedure *SP_Rendicion*).
La misma devuelve los datos de los viajes correspondientes con el nro. de rendición indicado, pero, dado que el total de la rendición ya se había calculado en el procedimiento, esta función evita tener que realizar una nueva consulta para obtenerlo.

> *fx_getRendicionExistente*: Esta función se utiliza para obtener los datos de una rendición ya existente.
Al no estar registrándose la rendición, no se ha calculado su importe. Por lo tanto, en este caso se invoca a la función anterior para conocer los viajes, pero además es necesario consultar las rendiciones existentes para obtener el total de la misma.

Facturación

Este caso es similar al anterior. Existen 2 funciones utilizadas con distintos propósitos.
> *fx_getDatosFactura*: Esta función se utiliza al momento de realizar la facturación (invocada desde el stored procedure *SP_Facturación*).
La misma devuelve los datos de los viajes correspondientes con el nro. de factura indicado, pero, dado que el total de la misma ya se había calculado en el procedimiento, esta función evita tener que realizar una nueva consulta para obtenerlo.

> *fx_getFacturaExistente*: Esta función se utiliza para obtener los datos de una factura ya existente.
Al no estar registrándose la factura, no se ha calculado su importe. Por lo tanto, en este caso se invoca a la función anterior para conocer los viajes, pero además es necesario consultar las facturas existentes para obtener el total de la misma.

Estadísticas

Utilizamos funciones para cada una de las estadísticas dado que estas solamente realizan consultas sobre la base sin modificar información existente.

- Chóferes con mayor recaudación: Se consultan todas las rendiciones de cada chofer dentro del intervalo de tiempo indicado (año y trimestre) y se suman sus importes. Luego se los ordena de forma descendente, obteniendo así los 5 de mayor recaudación.

- Choferes con el viaje más largo realizado: Para cada chofer se obtiene su viaje más largo dentro del intervalo de tiempo indicado. Es decir, de mayor recorrido. Luego, estos resultados son ordenados en forma decreciente y se muestran los 5 primeros.

- Clientes con mayor consumo: Para cada cliente se consultan y suman los importes de todas sus facturas dentro del intervalo de tiempo indicado y luego se seleccionan los 5 con mayor importe.

- Cliente que utilizó más veces el mismo automóvil en los viajes que ha realizado: Se consultan todos los viajes realizados por cada cliente dentro del intervalo controlando la cantidad de veces que utilizó el mismo vehículo.
  Para cada cliente se ordenarán los registros en forma descendente según la cantidad de veces utilizado el mismo vehículo y, luego, se ordenarán todos ellos (independientemente del cliente) en forma descendente.

*** Si en ese top se obtiene un mismo cliente varias veces (utilizó distintos autos y las cantidad fueron mayores a las del resto de clientes), el mismo será mostrado una única vez.*

*Ya que repetir clientes llevaría a que la estadística fuese de **autos** más utilizados en lugar de **clientes**.*

Triggers

Existen 2 triggers utilizados para el tratamiento de los turnos.

- tg_controlTurnosSuperpuestos: Este trigger se utiliza para controlar que cuando se realiza un alta o una modificación de algún **turno**, sus turnos no se superpongan con los horarios de otro turno existente y activo.

Esta validación podíamos haberla realizado en los métodos de alta y modificación de turno, pero implicaría repetición de lógica ya que en ambos casos las condiciones son iguales.

- tg_controlActualizacionTurnos: Este trigger se utiliza cuando el usuario modifica los datos de algún **Turno**.

En caso de que algún turno fuera referenciado por un viaje y, por ejemplo, se modificara su precio, esto generaría errores en los cálculos.
Por ejemplo: si el viaje fue realizado con un precio base de $10, y antes de facturarse se le cambiara el precio a $100, el monto de la factura no sería correcta ya que el precio real fue de $10.
Este trigger se encarga de mantener los datos del turno existente para que no queden datos inconsistentes en los viajes. Para esto los datos “actualizados” se guardan como un turno nuevo y se mantiene así el turno anterior marcándolo como copia de respaldo para que el mismo no pueda ser visualizado por los usuarios. Únicamente serán utilizados por el sistema.

Migración

Todas las migraciones de datos se realizan por medio de procedimientos que reciben como nombre migrar[:entidad]. (migrarAutos, migrarClientes, etc.)

Clientes

Analizando la tabla maestra, notamos que existen campos que se utilizan para identificar a los clientes:
[Cliente_Nombre, Cliente_Apellido, Cliente_DNI, Cliente_Telefono, Cliente_Direccion, Cliente_Mail, Cliente_Fecha_Nac]
Por lo tanto, cada diferente combinación encontrada de estos campos representaría a un cliente distinto.
Entonces, si por ejemplo existieran 5 registros donde estos campos fueran totalmente iguales entre sí, estos 5 registros harían referencia a un mismo cliente.
Por lo tanto, para normalizar los datos, este cliente se registra una sola vez en la tabla **Cliente**. Y luego se harán las referencias correspondientes.

Choferes

Notamos que existen campos que se utilizan para identificar a los choferes:
[Chofer_Nombre, Chofer_Apellido, Chofer_DNI, Chofer_Telefono, Chofer_Direccion, Chofer_Mail, Chofer_Fecha_Nac]

Por lo tanto, cada diferente combinación encontrada en estos campos representaría a un chofer distinto.
Por lo tanto, para normalizar estos datos, el chofer se registra una sola vez en la tabla **Chofer** y luego se hará referencia a este chofer desde otras tablas.

Autos

Notamos que existen campos que se utilizan para identificar los autos: *[Auto_Marca, Auto_Modelo, Auto_Patente, Auto_Licencia, Auto_Rodado]*
Por lo tanto, cada diferente combinación presente en estos campos representaría un auto distinto.
Además, dado que en todos los registros a un auto le corresponde un único chofer, a partir del DNI del mismo podemos consultar la tabla *Chofer* para obtener su ID y de esta forma mantener en la tabla **Autos** la referencia al chofer.
No pudimos hacer lo mismo con los *Turnos* dado que existen múltiples turnos registrados para un mismo auto y no tenemos ningún criterio para seleccionar uno sólo de ellos. Por lo tanto, la referencia quedará en null hasta que el usuario lo actualice desde la aplicación.

Turnos

Existen campos en la base original que se utilizan para identificar los diferentes Turnos:
[Turno_Hora_Inicio, Turno_Hora_Fin, Turno_Precio_Base, Turno_Valor_Kilometro, Turno_Descripcion].
Cada distinta combinación de valores presentes en estos campos representaría un turno diferente. Por lo tanto, se inserta un registro en la tabla **Turnos** con cada combinación existente.

Viajes

Pudimos identificar registros donde los campos *Factura_Nro* y *Rendicion_Nro* se encuentran vacíos (nulos), por lo que interpretamos que estos son los que representan a los viajes registrados.
Algunos de estos se encuentran repetidos (todos sus campos con idénticos valores).
Sin embargo, dado que estamos realizando una migración de un sistema existente y que no poseemos suficiente información al respecto, no nos parece correcto descartarlos.
Por lo tanto decidimos migrarlos así repetidos.
Además notamos que para cada uno de estos viajes existe un número de factura y rendición asociados, por lo que mantenemos dichas referencias.
De esta forma, todos los registros de nuestra tabla **Viaje** tendrán las referencias a sus facturas y rendiciones correspondientes, pudiendo haber algunos repetidos.
En cuanto a la hora de fin de los viajes, la misma no figura en la tabla maestra.
Pensamos ponerle a todos los viajes como hora de fin la hora de fin del turno en el que fue realizado. Pero no conocemos el impacto e importancia que esto podría tener en el sistema, por lo que optamos finalmente por dejar el campo vacío (null).

Rendiciones

Encontramos los siguientes campos referidos a las rendiciones:
[Rendicion_Nro, Rendicion_Fecha, Rendicion_Importe]
Por cada número de rendición encontrado existirá un único registro en nuestra tabla **Rendición** con los datos correspondientes (Fecha e importe total de la misma).
El ID del chofer lo obtenemos consultando nuestra tabla de choferes.
- Notamos que existen viajes que fueron rendidos más de una vez. Si bien esto no nos parece correcto, no poseemos suficiente información al respecto como para ignorar dichos registros. **Por lo tanto, cada uno de estos repetidos fueron tenidos en cuenta a la hora de calcular el importe total de la rendición (incluidos los que no se repetían), con el objetivo de no descartar ninguno de ellos.**

Facturas

Encontramos ciertos campos referidos específicamente a las facturas:
[Factura_Nro, Factura_Fecha, Factura_Fecha_Inicio, Factura_Fecha_Fin]
Aquellos registros con número de factura similares fueron agrupados ya que consideramos que hacen referencia a una misma factura.
Luego, a partir del DNI del cliente podemos consultar la tabla **clientes** y así obtener su ID para mantener la referencia.
Finalmente, para cada registro (correspondiente a esta factura) sumamos *precio base del turno* + *cantidad de kilómetros del viaje* * *precio del kilómetro*. A continuación se suman todos estos importes y así se obtiene el **total de la factura**.
- Notamos que existen viajes que fueron facturados más de una vez, ya que todos los campos poseen valores idénticos.
Sin embargo, no los descartamos ya que se encuentran en la base y no podemos tomar tal decisión sin tener suficiente información al respecto.
Por lo tanto, cada uno de estos registros repetidos fueron tenidos en cuenta para calcular el importe total de cada factura.
Es decir, si un viaje fue facturado dos veces, en el importe total se sumarán ambos importes aunque estuvieran repetidos (además del resto de los viajes de dicha factura).

Consideraciones Importantes (BD)

1. Las funcionalidades son creadas por default en el stored procedure *SP_crearFuncionalidadesDefault* de acuerdo a las indicadas en el enunciado.
2. También se generan 3 roles (admin, clientes, choferes) con funcionalidades default en el procedimiento *SP_crearRolesDefault*. Sin embargo, el usuario **admin** tiene acceso a la administración de los mismos. Por lo que podrían ser modificadas sus funcionalidades en caso de ser requerido, o crearse nuevos roles.
3. Dado que no hay demasiados detalles acerca de la creación de usuarios, decidimos que se crearán por default 3 de ellos con distintos roles en el stored procedure *SP_crearUsuariosDefault*. El resto de los usuarios deberán ser dados de alta por medio de la aplicación. (No se crean automáticamente para cada cliente ni chofer migrados).

Los usuarios existentes son:

Usuario	Contraseña	Rol	Funcionalidades
admin	w23e	admin	Todas
cliente	cliente	cliente	Facturación
chofer	chofer	chofer	Rendición

4. Resumen con respecto a la interpretación de los registros de la tabla maestra relacionado con los *viajes*, *facturas* y *rendiciones*:

- Si los campos Rendicion_Nro y Factura_Nro son null => ese registro representa 1 **viaje**.
- Si el campo Rendicion_Nro posee valor => ese registro permite relacionar 1 *viaje* con su correspondiente **rendición**.
- Si el campo Factura_Nro posee valor => ese registro permite relacionar 1 viaje con su correspondiente **factura**.