**Gestión de Datos**

08-2030

**UberFRBA**

**Estrategia**



|  |  |  |
| --- | --- | --- |
| **Profesor** | REINOSA, Enrique |  |
| **Alumnos** | RODRIGUEZ, Luis | 149567-7 |
|  | BULBULIAN, Juan Pablo | 116134-9 |
|  | COHEN, Damián | 143668-5 |
|  | BROWARNIK, Gabriel | 151893-8 |
| **Curso** | K-3012 |  |
| **Año** | 2017 |  |
| **Cuatrimestre** | 1º |  |

**Índice**

[Modelo de Datos 4](#_Toc484975961)

[DER 4](#_Toc484975962)

[Entidades, claves y relaciones 4](#_Toc484975963)

[Campos comunes: 4](#_Toc484975964)

[Usuario: 5](#_Toc484975965)

[Rol: 5](#_Toc484975966)

[Funcionalidad: 5](#_Toc484975967)

[Chofer/Cliente: 5](#_Toc484975968)

[Auto: 5](#_Toc484975969)

[Turno: 5](#_Toc484975970)

[Viaje: 5](#_Toc484975971)

[Rendición: 6](#_Toc484975972)

[Factura: 6](#_Toc484975973)

[Aplicación Desktop 7](#_Toc484975974)

[Conexión con la Base de Datos 7](#_Toc484975975)

[BDHandler.cs 7](#_Toc484975976)

[BDParametro.cs 7](#_Toc484975977)

[Usuario 8](#_Toc484975978)

[Usuario.cs 8](#_Toc484975979)

[Registro de usuario 8](#_Toc484975980)

[AltaUsuario.cs 8](#_Toc484975981)

[Login 9](#_Toc484975982)

[Login.cs 9](#_Toc484975983)

[SeleccionRol.cs 9](#_Toc484975984)

[Búsqueda de Chofer / Cliente 10](#_Toc484975985)

[BuscarIndividuo.cs. 10](#_Toc484975986)

[ABM Cliente 11](#_Toc484975987)

[AltaModiCliente.cs 11](#_Toc484975988)

[ABM Chofer 12](#_Toc484975989)

[AltaModiChofer.cs 12](#_Toc484975990)

[ABM Automóvil 13](#_Toc484975991)

[BuscarAuto.cs 13](#_Toc484975992)

[AltaModiAuto.cs 13](#_Toc484975993)

[Registro de Viajes 14](#_Toc484975994)

[RegistroViaje.cs 14](#_Toc484975995)

[ABM Rol 15](#_Toc484975996)

[AltaRol.cs 15](#_Toc484975997)

[BajaRol.cs 15](#_Toc484975998)

[EditarRol.cs 15](#_Toc484975999)

[ABM Turno 16](#_Toc484976000)

[AltaTurno.cs 16](#_Toc484976001)

[BajaTurno.cs 16](#_Toc484976002)

[EditarTurno.cs 16](#_Toc484976003)

[Listado Estadístico 17](#_Toc484976004)

[Estadistica.cs 17](#_Toc484976005)

[Facturación 18](#_Toc484976006)

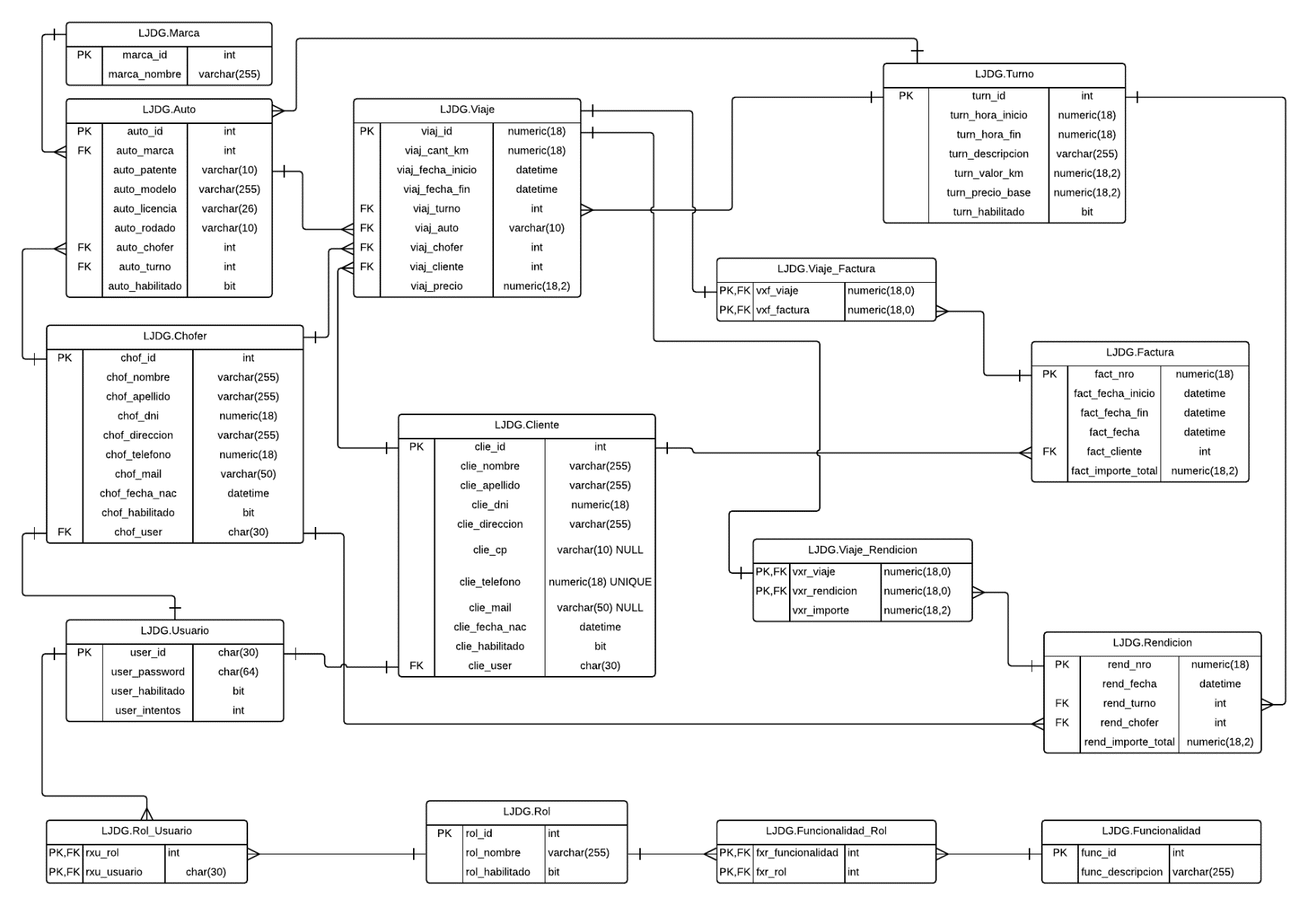
[Facturacion.cs 18](#_Toc484976007)

[Rendición de cuenta del chofer 19](#_Toc484976008)

[Rendicion.cs 19](#_Toc484976009)

# Modelo de Datos

## DER

Diagrama Entidad Relación del modelo utilizado

## Entidades, claves y relaciones

### Campos comunes:

**habilitado:** campo de tipo bit que se encuentra en múltiples entidades, determina el estado activo/no activo de la fila en cuestión. Todas las entidades ingresan nuevas filas por defecto habilitadas salvo la entidad Automóvil que puede insertarse deshabilitada.

**id:** campo de tipo int o numeric (18,0) que actúa como PK de múltiples entidades, se utiliza IDENTITY (1,1) como autoincremental.

**nro:** similar a id, son de tipo numeric (18,0) y se utilizan en Rendición y Factura. A diferencia de id son datos migrados por lo tanto el autoincremento comienza después de la migración, para futuras filas.

**nombre/descripción:** es una cadena de tipo varchar(255) que se encuentra en múltiples entidades para identificar en forma de texto una fila.

### Usuario:

**id:** cadena de 30 caracteres que actúa como *username* y es PK de la tabla, por lo que identifica unívocamente a un usuario del sistema.   
**password:** es una cadena de 64 caracteres proveniente de la encriptación en SHA256.

**intentos:** entero cuyo valor inicial es 3 y disminuye con cada fallo del usuario al intentar loguearse.

La tabla Usuario se relaciona con la tabla Rol mediante la tabla Rol\_Usuario (relación muchos a muchos) y a las tablas Cliente y Chofer mediante FK (**clie\_user** y **chof\_user** - en relación uno a uno)

### Rol:

Utiliza los campos **id,** **nombre** y **habilitado** ya explicados. Al deshabilitarse se eliminan las relaciones del rol en cuestión con los usuarios.

### Funcionalidad:

Utiliza los **id** y **descripción** ya explicados. No pueden deshabilitarse.

### Chofer/Cliente:

Contiene el campo **user** que actúa como FK con Usuario ya explicada, campos **id** y **habilitado** y diversos campos correspondientes a los datos personales del chofer/cliente.

Entre estas entidades se pueden notar las siguientes diferencias:

El campo **cp** (código postal), dato inexistente en los tabla de origen está definido como varchar (10), inicializado en NULL y solo se encuentra en la tabla Cliente, que debe ingresarlo obligatoriamente en futuras altas/modificaciones.

El campo **teléfono** es clave única en la tabla Cliente (no en Chofer), y el campo **mail** puede ser NULL en Cliente ya que no es obligatorio en dicha entidad.

### Auto:

Contiene el campo **chofer** que actúa como FK con Chofer, relacionando uno o más autos con un chofer. Además contiene campos **id** y **habilitado**, campos correspondientes a los datos del vehículo, incluyendo el campo **patente** que es clave única.

Por último contiene **turno**, que actúa como FK con Turno, relacionando uno o más autos con un turno.

### Turno:

Utiliza los **id, habilitado** y **descripción** ya explicados. Además contiene campos que contienen los datos específicos del turno inicialmente migrados, y que luego pueden modificarse.

### Viaje:

Modela las operaciones realizadas en la empresa. Utiliza el **id** ya explicado como PK. Contiene múltiples FK con las tablas Turno, Chofer, Auto y Cliente, en relaciones uno a muchos (un turno/chofer/auto/cliente, muchos viajes) y se relaciona con Rendicion y Factura mediante las tablas Viaje\_Rendicion y Viaje\_Factura, en relaciones uno a muchos (una Factura/Rendicion, muchos viajes).

Ademas contiene campos con los datos específicos del viaje. El campo **fecha\_fin**, que no existe en los datos previos se inicializa en NULL pero debe registrarse para nuevos viajes.

El campo **precio** es calculado utilizando campos del turno **precio\_base** y **valor\_km**, y el campo del viaje **cant\_km** y se agrega a la entidad a fin de fijar el costo del viaje al momento en el cual se efectuó.

### Rendición:

Utiliza el **nro** ya explicado como PK, además se relaciona con Turno y Chofer mediante FK y con Viaje mediante tabla Viaje\_Rendicion. Esta tabla contiene además el importe rendido por cada viaje, dato que es inicialmente migrado y luego calculado para nuevos viajes *(porcentaje por defecto utilizado: 30%)*

Contiene el campo **fecha** y el campo **importe\_total,** calculado como la sumatoria del campo **importe** de Viaje\_Rendicion.

### Factura:

Utiliza el **nro** ya explicado como PK, además se relaciona con Cliente mediante FK y con Viaje mediante tabla Viaje\_Factura. Además contiene la **fecha** en la que se generó la factura y los campos **fecha\_inicio** y **fecha\_fin** que determinan los viajes abarcados por la misma.

Contiene el campo **importe\_total,** calculado como la sumatoria del campo **precio** de los viajes abarcados.

# Aplicación Desktop

# Conexión con la Base de Datos

En la carpeta ConexiónBD se encuentran dos clases.

## BDHandler.cs

Aquí se levanta la conexión desde el archivo App.config, el cual tiene una entrada llamada BDConfig con los datos para la conexión que exige el enunciado.

Posee tres métodos…

publicDataTableexecSelectSP(String nameStoredProcedure, List<BDParametro>listParametros = null)

Permite ejecutar un SP que devuelva un Select con su lista de parámetros y nos devuelve un DataTable para mostrar en la aplicación.

public void execSP(String nameStoredProcedure, ref List<BDParametro>listParametros)

Permite ejecutar un SP, con su lista de parámetros pasada por referencia, en la cual devuelve los parámetros de salida devueltos por el motor.

public List<String>execListSP(String nameStoredProcedure)

Permite ejecutar un SP, y devuelve una lista con la primera columna del Select que devuelve el motor.

## BDParametro.cs

Sirve para el modelado de los parámetros de entrada y salida usados por los métodos de la clase **BDHandler.cs.**

# Usuario

Se modeló al usuario de la aplicación en la siguiente clase.

## Usuario.cs

La cual entiende los siguientes métodos.

public String iniciarSesion()

Utiliza el SP LJDG.iniciar\_sesion para validar los datos de inicio de sesión. La contraseña ya viaja encriptada con el algoritmo SHA256.

public List<String> obtenerRolesUsuario()

Utiliza el SPLJDG.obtener\_roles\_usuario para tener al usuario como variable global del sistema luego del login. Y devuelve una lista que será usada en el combobox en la segunda pantalla del login.

# Registro de usuario

Se utiliza el siguiente formulario para dar de alta un usuario con un rol en particular.

## AltaUsuario.cs

En este formulario se deben ingresar los siguientes datos:

- Nombre de usuario.

- Contraseña y su repetición.

- El selector de rol

La selección del rol se obtiene de una lista completada con los roles habilitados cargados en la Base de Datos por medio de una consulta dentro de la aplicación.

La aplicación se encarga de realizar la verificación de la existencia del nombre de usuario por medio de una consulta a la Base de Datos.

Completada la verificación la aplicación realiza el alta del usuario por medio del stored procedure **LJDG.alta\_usuario.**

# Login

Se cuenta con dos formularios dentro de la carpeta Logueo.

## Login.cs

Aquí se solicita el nombre de usuario y contraseña la cual es encriptada al momento y a través de la clase Usuario.cs se validan los datos. Si el usuario no tiene roles asignados no deja ingresar a la aplicación.

## [SeleccionRol.cs](https://github.com/pycongroo/GDD2017-uber/blob/master/App/Logueo/SeleccionRol.cs)

Aquí se muestra un combo desplegable con todos los roles disponibles para el usuario.

Los cuales fueron obtenidos en el paso anterior a través de los métodos de la clase Usuario.cs. Una vez realizado un login exitoso se accede al menú principal de la aplicación.

# Búsqueda de Chofer / Cliente

Se trata de un formulario auxiliar que asiste a diversas funcionalidades, permitiendo buscar un chofer o un cliente (entre habilitados o entre todos) filtrando por nombre, apellido y/o DNI.

## BuscarIndividuo.cs.

publicBuscarIndividuo(Form \_formPadre, String \_tipoIndividuo, char \_modo)

En \_formPadre recibe el formulario anterior al cual le serán devueltos los parámetros del individuo a buscar. Estos son ID, nombre y apellido.

En \_tipoIndividuo se especifica el tipo de individuo a buscar “Cliente” o “Chofer”**.**

En \_modo se especifica la función que va a desempeñar…

* **M -** para modificación, buscara tanto individuos habilitados como deshabilitados.
* **B -** busca solo individuos habilitados y gestionara la baja.
* **Otros -** devuelve los datos del individuo seleccionado al formulario que lo invoque, se buscará entre habilitados o entre todos según corresponda

Se utilizan los siguientes StoredProcedures.

LJDG.buscar\_chofer

LJDG.buscar\_chofer\_habilitado

LJDG.buscar\_cliente

LJDG.buscar\_cliente\_habilitado

# ABM Cliente

Se utiliza el siguiente formulario

## AltaModiCliente.cs

public altaModiCliente (char \_modo, int \_idCliente)

Recibe el modo de trabajo…

* **A** – Para alta, ofrece todas las opciones de carga exigidas por el enunciado.
* **M** – Pre carga el cliente pasado por \_idCliente haciendo uso del formulario **BuscarIndividuo.cs.**

En el modo de Alta, se debe seleccionar si se desea dar de alta un cliente con un usuario existente (es decir que tenga otro rol asignado) en cuyo caso se completan los datos del chofer; o con un usuario nuevo, que no exista en el sistema, en cuyo caso debe ingresarse una contraseña, además de los datos del cliente. En ambos casos se asigna el rol cliente al usuario y se da de alta una fila nueva a la tabla Cliente.

Para dar de baja, se utiliza el formulario **BuscarIndividuo.cs** para seleccionar el cliente a deshabilitar.

Para este caso de uso. Se utilizan los siguientes StoredProcedures.

LJDG.alta\_cliente\_usuario\_nuevo

LJDG.alta\_cliente\_usuario\_existente

LJDG.modi\_cliente

LJDG.baja\_cliente

LJDG.obtener\_cliente

# ABM Chofer

Se utiliza el siguiente formulario.

## AltaModiChofer.cs

public altaModiChofer(char \_modo, int \_idChofer)

Recibe el modo de trabajo…

* **A** – Para alta, ofrece todas las opciones de carga exigidas por el enunciado.
* **M** – Pre carga el chofer pasado por \_idChofer haciendo uso del formulario **BuscarIndividuo.cs.**

En el modo de Alta, se debe seleccionar si se desea dar de alta un chofer con un usuario existente (es decir que tenga otro rol asignado) en cuyo caso se completan los datos del chofer; o con un usuario nuevo, que no exista en el sistema, en cuyo caso debe ingresarse una contraseña, además de los datos del chofer. En ambos casos se asigna el rol chofer al usuario y se da de alta una fila nueva a la tabla Chofer.

Para dar de baja, se utiliza el formulario **BuscarIndividuo.cs** para seleccionar el chofer a deshabilitar.

Para este caso de uso. Se utilizan los siguientes StoredProcedures.

LJDG.alta\_chofer\_usuario\_nuevo

LJDG.alta\_chofer\_usuario\_existente

LJDG.modi\_chofer

LJDG.baja\_chofer

LJDG.obtener\_chofer

# ABM Automóvil

Se utilizan dos formularios.

## BuscarAuto.cs

public BuscarAuto(char \_modo)

Recibe el modo de trabajo…

* **M** – Para modificar, muestra todos los automóviles.
* **B** – Para Baja, solo muestra los automóviles habilitados.

## AltaModiAuto.cs

public altaModiAuto(char \_modo, int \_idAuto)

Recibe el modo de trabajo…

* **A** – Para alta, ofrece todas las opciones de carga exigidas por el enunciado.
* **M** – Pre carga el automóvil pasado por \_idAuto.

Para este caso de uso. Se utilizan los siguientes StoredProcedures.

LJDG.buscar\_auto

LJDG.buscar\_auto\_habilitado

LJDG.baja\_auto

LJDG.obtener\_marcas

LJDG.obtener\_descripcion\_turnos

LJDG.alta\_auto

LJDG.modi\_auto

LJDG.obtener\_auto

# Registro de Viajes

Se utiliza el formulario.

## RegistroViaje.cs

Se recolectan todos los datos necesarios, haciendo uso del formulario **BuscarIndividuo.cs.**

Se realizan todas las validaciones solicitadas por el enunciado, y al motor se le deja la responsabilidad de chequear los viajes duplicados del cliente con el SP LJDG.registrar\_viaje

Para este caso de uso. Se utilizan los siguientes StoredProcedures.

LJDG.obtener\_auto\_habilitado\_chofer

LJDG.validar\_horarios\_turno

LJDG.obtener\_precio\_viaje

LJDG.registrar\_viaje

# ABM Rol

Se utilizan tres formularios y las clasesRol.cs y Funcionalidad.cs.

## AltaRol.cs

Se recolectan todos los datos necesarios del rol y las funcionalidades existentes para poder asociarlas al mismo, las mismas fueron obtenidas por el método publicstaticList<Funcionalidad>obtenerFuncionalidades()de la clase Funcionalidad.cs.Se realizan todas las validaciones solicitadas por el enunciado, y se procede a realizar el guardado del mismo mediante el método publicstaticintinsertarRol(stringnombreRol)de la clase Rol.cs y luego al método publicstaticvoidinsertarFuncxRol(intidRol,intidFuncionalidad)de la clase Funcionalidad.cs.

## BajaRol.cs

Aquí se muestra un combo desplegable con todos los roles existentes que se encuentren habilitados.

Los mismos fueron obtenidos a través del método publicstaticList<Rol>obtenerRoles()de la clase Rol.cs. Una vez seleccionado un rol y dado de baja a través del formulario, se llamará al métodopublicstaticvoideliminarRol(introlId)perteneciente a la misma clase.

## EditarRol.cs

Aquí se muestra un combo desplegable con todos los roles existentes (con el mismo método utilizado en el formulario BajaRol.cs). Al ser seleccionado uno, se habilitan los campos para ser editado y se permite eliminar o agregar nuevas funcionalidades asociadas que fueron obtenidas por el método publicstaticList<Funcionalidad>obtenerFuncxRol(intidRol)de la clase Funcionalidad.cs, así como habilitar un rol que se encontraba deshabilitado. Para salvar la información del rol editado, se llamará al método publicstaticvoideditarRol(introlId, stringrolNombre, introlHabilitado)de la clase Rol.cs.

Para este caso de uso. Se utilizan los siguientes StoredProcedures.

LJDG.obtener\_roles

LJDG.crear\_rol

LJDG.editar\_rol

LJDG.eliminar\_rol

LJDG.obtener\_funcionalidades

LJDG.obtener\_funcionalidadesxrol

LJDG.crear\_funcxrol

# ABM Turno

Se utilizan tres formularios y la clase Turno.cs.

## AltaTurno.cs

Se recolectan todos los datos necesarios del turno.Se realizan todas las validaciones solicitadas por el enunciado, y se procede a realizar el guardado del mismo mediante el método publicstaticintinsertarTurno(stringdescripcion, decimal horaInicio, decimal horaFin, decimal valorKm, decimal precioBase)de la clase Turno.

## BajaTurno.cs

Aquí se muestra un combo desplegable con todos los turnos existentes que se encuentren habilitados.

Los mismos fueron obtenidos a través del método publicstaticList<Turno>obtenerTurnos()de la clase Turno.cs. Una vez seleccionado un rol y dado de baja a través del formulario, se llamará al métodopublicstaticvoideliminarTurno(intturnoId)perteneciente a la misma clase.

## EditarTurno.cs

Aquí se muestra un combo desplegable con todos los turnos existentes (con el mismo método utilizado en el formulario BajaTurno.cs). Al ser seleccionado uno, se habilitan los campos para ser editado y se permite editar o habilitar un turno que se encontraba deshabilitado. Para salvar la información del turno editado, se llamará al método publicstaticvoideditarTurno(intturnoId, stringdescripcion, decimal horaInicio, decimal horaFin, decimal valorKm, decimal precioBase, intturnoHabilitado)de la clase Turno.cs.

Para este caso de uso. Se utilizan los siguientes StoredProcedures.

LJDG.obtener\_turnos

LJDG.crear\_turno

LJDG.editar\_turno

LJDG.eliminar\_turno

# Listado Estadístico

## Estadistica.cs

A través del menú principal se accede a dicho formulario, en el tenemos un campo para ingresar el año de la consulta y un combo para seleccionar el trimestre. Ambos datos serán pasados a los StoredProcedures que nos devuelven el listado correspondiente a un datagrid de acuerdo lo que devuelva el SELECT de la query.

Con una estructura Switch se determina cual será ejecutado.

LJDG.choferes\_mayor\_recaudacion.sql

LJDG.choferes\_viaje\_mas\_largo.sql

LJDG.cliente\_auto.sql

LJDG.clientes\_mayor\_consumo.sql

# Facturación

## Facturacion.cs

En este formulario se deben ingresar los datos exigidos en el enunciado.

Fecha de inicio de y fin de fin, para determinar el rango de los viajes facturas. El buscador de individuos nos permite seleccionar al Cliente.

El sistema automáticamente mostrara en un datagrid todos los viajes realizados por ese cliente en ese periodo y adicionalmente el apellido del chofer, la patente del automóvil utilizado y el precio de dicho viaje. Al final se calcula el monto a facturar.

Para este objetivo se utiliza el StoredProcedure.

LJDG.**viajes\_cliente.sql**

Al registrar la factura, se realizan las validaciones y si son correctas, se llama al método public static string insertarFactura (int fact\_cliente, decimal fact\_total, DateTime fact\_fecha, DateTime fact\_fecha\_inicio, DateTime fact\_fecha\_fin) de la clase Factura.cs que procederá a registrar la factura.

El StoredProcedure encargado del registro de la misma es

LJDG.crear\_factura.sql

Al finalizar se nos mostrará un mensaje informando el éxito (si se registró la factura) o error de la operación (si no se puedo registrar, ya sea porque existía una factura o porque no hay viajes en el período elegido para el cliente seleccionado).

# Rendición de cuenta del chofer

## Rendicion.cs

En este formulario se deben ingresar los siguientes datos:

- Fecha de rendición, para determinar el rango de los viajes facturas a considerar en la rendición.

- El selector del turno que se quiere considerar en la rendición.

- El buscador de individuos nos permite seleccionar al Chofer.

El sistema muestra mediante un datagrid todos los viajes realizados por ese chofer en ese día y turno, apellido del cliente, la patente del automóvil utilizado, el precio de dicho viaje y el importe de rendición correspondiente, calculado mediante el porcentaje establecido en la configuración de la aplicación. Esto se realiza mediante el siguiente stored procedure:

LJDG.**viajes\_chofer.sql**

El importe total se calcula dentro de la aplicación y se muestra al usuario.

Por último la aplicación permite registrar la rendición en la BD mediante el llamado al stored procedure:

LJDG.**crear\_rendicion.sql**

Al finalizar se nos mostrará un mensaje informando el éxito (si se registró la rendición) o error de la operación (si no se puedo registrar, ya sea porque existía una rendición o porque no hay viajes en el día y turno elegido para el chofer seleccionado).