

Universidad Tecnologica Nacional

Facultad Regional Buenos Aires

**Gestion de Datos**

Trabajo Práctico

2do Cuatrimestre 2016

Clínica FRBA

Grupo: Stranger Strings

*Documento de Estrategia*

Indice:

***DER:***

**El der se encuentra adjunto en JPEG como DER TP GDD.JPEG**

***Entidades:***

**Tabla Plan\_Medico:**

*Tabla que representa a los planes medicos. Posee una PK “Codigo\_Plan” no subrogada, dichos codigos se migran de la tabla maestra*

CREATE TABLE STRANGER\_STRINGS.Plan\_Medico(

Codigo\_Plan INT PRIMARY KEY,

Descripcion VARCHAR(255),

Precio\_Bono\_Consulta NUMERIC(18,0),

Precio\_Bono\_Farmacia NUMERIC(18,0)

)

-----------------------------------------------------------

**Tabla Usuario:**

*Tabla que representa a los usuarios del sistema. Utiliza una PK “Id\_Usuario” subrogada y autoincremental. El campo pasword esta definido VARBINARY para que almacene correctamente los hash de la constraseña encriptada*

CREATE TABLE STRANGER\_STRINGS.Usuario(

Id\_Usuario INT IDENTITY(1,1) PRIMARY KEY,

Usuario VARCHAR(255),

Pasword VARBINARY(255),

Cantidad\_Intentos SMALLINT,

)

-----------------------------------------------------------

**Tabla Paciente:**

*Tabla que representa a los afiliados en el sistema. Utiliza una PK “Id\_Paciente” subrogada y autoincremental. Posee un check en el Sexo del afiliado para que sea Femenino (F) o Masculino (M). A su vez posee otro check para el estado de dicho afiliado, el cual será utilizado para la baja del mismo. A su vez posee una FK Id\_Usuario a la PK del mismo nombre en la tabla Usuario.*

CREATE TABLE STRANGER\_STRINGS.Paciente(

Id\_Paciente INT IDENTITY(1,1) PRIMARY KEY,

Num\_Afiliado\_Raiz NUMERIC(20,0),

Num\_Afiliado\_Resto NUMERIC(2,0),

Nombre VARCHAR(255),

Apellido VARCHAR(255),

Tipo\_Doc VARCHAR(10),

Num\_Doc NUMERIC(18,0),

Direccion VARCHAR(255),

Telefono NUMERIC(18,0),

Mail VARCHAR(255),

Fecha\_Nac DATETIME,

Sexo CHAR(1) CHECK(Sexo = 'F' OR Sexo = 'M' OR Sexo IS NULL),

Estado\_Civil VARCHAR(15),

Familiares\_A\_Cargo NUMERIC(10,0),

Codigo\_Plan INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Plan\_Medico(Codigo\_Plan),

Cantidad\_Consulta INT,

Id\_Usuario INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Usuario(Id\_Usuario),

Estado\_Afiliado CHAR(1) CHECK(Estado\_Afiliado = 'A' OR Estado\_Afiliado = 'D' OR Estado\_Afiliado IS NULL),

)

-----------------------------------------------------------

**Tabla Compra:**

*Tabla que representa la compra de bonos en el sistema. Posee una PK “Id\_Compra” subrogada y autoincremental. Posee una FK Id\_Paciente a la PK del mismo nombre en la tabla Paciente para poder referenciar al paciente con al compra que realizó.*

CREATE TABLE STRANGER\_STRINGS.Compra(

Id\_Compra INT IDENTITY(1,1) PRIMARY KEY,

Fecha\_Compra datetime,

Cantidad\_Bonos INT,

Importe\_Total NUMERIC(7,2),

Id\_Paciente INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Paciente(Id\_Paciente))

-----------------------------------------------------------

**Tabla Baja\_Paciente**

*Tabla que representa a las bajas logicas realizadas sobre los afiliados. Dicha tabla posee una PK “Id\_Baja” subrogada y autoincremental. A su vez posee una FK Id\_Paciente a la PK del mismo nombre en la tabla Paciente a fin de referenciar la baja logica con un paciente.*

CREATE TABLE STRANGER\_STRINGS.Baja\_Paciente(

Id\_Baja INT IDENTITY(1,1) CONSTRAINT PK\_Id\_Baja PRIMARY KEY,

Id\_Paciente INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Paciente(Id\_Paciente),

Fecha\_Baja DATETIME)

-----------------------------------------------------------

**Tabla Cambio\_Plan**

*Tabla que representa los cambios de plan de un Paciente. Posee una PK “Id\_Cambio” subrogada y autoincremental. Posee una FK Id\_Paciente a la PK del mismo nombre en la tabla Paciente a fin de registrar que paciente realizó el cambio. Tambien posee 2 FKs (“Codigo\_Plan\_Viejo” y “Codigo\_Plan\_Nuevo”) a los fines de registrar entre que planes se realizó el cambio.*

CREATE TABLE STRANGER\_STRINGS.Cambio\_Plan(

Id\_Cambio INT IDENTITY(1,1)PRIMARY KEY,

Id\_Paciente INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Paciente(Id\_Paciente),

Motivo VARCHAR(255),

Codigo\_Plan\_Viejo INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Plan\_Medico(Codigo\_Plan),

Codigo\_Plan\_Nuevo INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Plan\_Medico(Codigo\_Plan))

-----------------------------------------------------------

**Tabla Funcionalidad**

*Tabla que representa las funcionalidades disponibles en el sistema. Posee una PK “Id\_Funcionalidad” subrogada y autoincremental.*

CREATE TABLE STRANGER\_STRINGS.Funcionalidad(

Id\_Funcionalidad INT IDENTITY(1,1) PRIMARY KEY,

Descripcion VARCHAR(225))

-----------------------------------------------------------

**Tabla Rol**

*Tabla que representa los roles disponibles en el sistema. Posee una PK “Id\_Rol” subrogada y autoincremental. A su vez posee un campo Estado que sirve para identificar un rol dado de baja.*

CREATE TABLE STRANGER\_STRINGS.Rol(

Id\_Rol INT IDENTITY(1,1)PRIMARY KEY,

Descripcion VARCHAR(225),

Estado CHAR(1)CHECK(Estado = 'E' OR Estado = 'D' OR Estado IS NULL))

-----------------------------------------------------------

**Tabla Funcionalidad\_x\_Rol**

*Tabla intermedia entre las tablas Rol y Funcionalidad. Su funcion es relacionar cada rol con sus n funcionalidades que puede tener. Posee una clave compuesta “Id\_Rol,Id\_Funcionalidad” que a su vez son FK a las PKs de las tablas Rol y Funcionalidad.*

CREATE TABLE STRANGER\_STRINGS.Funcionalidad\_X\_Rol(

Id\_Rol INT ,

Id\_Funcionalidad INT ,

PRIMARY KEY (Id\_Rol, Id\_Funcionalidad),

FOREIGN KEY (Id\_Rol) REFERENCES STRANGER\_STRINGS.Rol (Id\_Rol),

FOREIGN KEY (Id\_Funcionalidad) REFERENCES STRANGER\_STRINGS.Funcionalidad (Id\_Funcionalidad))

-----------------------------------------------------------

**Tabla Rol\_X\_Usuario**

*Tabla intermedia entre las tablas Rol y Usuario. Su funcion es relacionar cada usuario con sus n roles que puede tener. Posee una clave compuesta “Id\_Rol,Id\_Usuario” que a su vez son FK a las PKs de las tablas Rol y Usuario.*

CREATE TABLE STRANGER\_STRINGS.Rol\_X\_Usuario(

Id\_Rol INT ,

Id\_Usuario INT ,

PRIMARY KEY (Id\_Rol, Id\_Usuario),

FOREIGN KEY (Id\_Rol) REFERENCES STRANGER\_STRINGS.Rol (Id\_Rol),

FOREIGN KEY (Id\_Usuario) REFERENCES STRANGER\_STRINGS.Usuario (Id\_Usuario))

-----------------------------------------------------------

**Tabla Medico**

*Tabla que representa a los profesionales en el sistema. Utiliza una PK “Id\_Medico” subrogada y autoincremental. Posee un check en el Sexo del afiliado para que sea Femenino (F) o Masculino (M). A su vez posee una FK Id\_Usuario a la PK del mismo nombre en la tabla Usuario.*

CREATE TABLE STRANGER\_STRINGS.Medico(

Id\_Medico INT IDENTITY(1,1) PRIMARY KEY,

Nombre VARCHAR(225),

Apellido VARCHAR(225),

Tipo\_Doc VARCHAR(225),

Num\_Doc NUMERIC(18,0),

Direccion VARCHAR(255),

Telefono NUMERIC(18,0),

Mail VARCHAR(225),

Fecha\_Nac DATETIME,

Sexo CHAR(1) CHECK(Sexo = 'F' OR Sexo = 'M' OR Sexo IS NULL),

Id\_Usuario INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Usuario(Id\_Usuario))

-----------------------------------------------------------

**Tabla Especialidad**

*Tabla que representa las especialidades disponibles que pueden adoptar los medicos en el sistema. Posee una PK “Especialidad Codigo” subrogada y autoincremental.*

CREATE TABLE STRANGER\_STRINGS.Especialidad(

Especialidad\_Codigo NUMERIC(18,0) PRIMARY KEY,

Especialidad\_Descripcion VARCHAR(225),

Tipo\_Especialidad\_Codigo NUMERIC(18,0),

Tipo\_Especialidad\_Descripcion VARCHAR(225))

-----------------------------------------------------------

**Tabla Especialidad\_X\_Medico**

*Tabla intermedia entre las tablas Especialidad y Medico. Su funcion es relacionar cada medico con sus n especialidades que puede tener. Posee una PK “Id” subrogada y autoincremental. A su vez posee 2 FKs(“Id\_Medico” y “Especialidad\_Codigo”) las cuales referencian a las PKs del mismo nombre en sus respectivas tablas.*

CREATE TABLE STRANGER\_STRINGS.Especialidad\_X\_Medico(

Id INT IDENTITY(1,1),

Id\_Medico INT ,

Especialidad\_Codigo NUMERIC(18,0) ,

PRIMARY KEY (Id),

FOREIGN KEY (Id\_Medico) REFERENCES STRANGER\_STRINGS.Medico (Id\_Medico),

FOREIGN KEY (Especialidad\_Codigo) REFERENCES STRANGER\_STRINGS.Especialidad (Especialidad\_Codigo))

-----------------------------------------------------------

**Tabla Horarios\_Agenda**

*Tabla que representa la agenda de cada medico. Dicha tabla posee una PK “Id\_Horario” subrogada y autoincremental. A su vez posee una FK “Id\_Especialidad\_Medico” que referencia a la PK “Id” en la tabla Especialidad\_X\_Medico, dado que los horarios son para cada medico, por especialidad.*

CREATE TABLE STRANGER\_STRINGS.Horarios\_Agenda(

Id\_Horario INT IDENTITY(1,1) PRIMARY KEY,

Dia SMALLINT,

Hora\_Desde TIME,

Hora\_Hasta TIME,

Id\_Especialidad\_Medico INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Especialidad\_X\_Medico(Id))

-----------------------------------------------------------

**Tabla Bono**

*Tabla que representa los bonos que puede adquirir un afiliado para poder atenderse, es decir realizar una consulta con un medico. Posee una PK “Id\_Bono” subrogada y autoincremental. A su vez posee 2 FKs (“Id\_Paciente Compro” y “Id\_Paciente\_Uso”), las cuales ambas referencian a la PK “Id\_Paciente” de la tabla Paciente. Tambien posee una FK “Codigo\_Plan” referenciando a la PK del mismo nombre en la tabla Plan\_Medico, indicando el plan del paciente que compró el bono. Por ultimo posee una FK “Id\_Compra” referenciando a la PK del mismo nombre en la tabla Compra, a fin de relacionar el bono con la compra de la cual se adquirió el mismo.*

CREATE TABLE STRANGER\_STRINGS.Bono(

Id\_Bono INT IDENTITY(1,1) PRIMARY KEY,

Fecha\_Compra DATETIME,

Fecha\_Impresion DATETIME,

Id\_Paciente\_Compro INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Paciente(Id\_Paciente),

Id\_Paciente\_Uso INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Paciente(Id\_Paciente),

Codigo\_Plan INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Plan\_Medico(Codigo\_Plan),

Numero\_Consulta INT,

Id\_Compra INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Compra(Id\_Compra))

-----------------------------------------------------------

**Tabla Consulta**

*Tabla encargada de representar las consultas realizadas por los pacientes con un medico en especial, previa reserva de turno. Posee una PK “Id\_Consulta” subrogada y autoincremental. A su vez posee 2 PKs (“Bono\_Consulta\_Id y “Id\_Paciente”) las cuales referancian a las PKs “Id\_Bono” y “Id\_Paciente” de las tablas Bono y Paciente respectivamente.*

CREATE TABLE STRANGER\_STRINGS.Consulta(

Id\_Consulta INT IDENTITY(1,1) PRIMARY KEY,

Fecha\_Y\_Hora\_Llegada DATETIME,

Fecha\_Y\_Hora\_Atencion DATETIME,

Sintomas VARCHAR(225),

Enfermedades VARCHAR(225),

Bono\_Consulta\_Id INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Bono(Id\_Bono),

Id\_Paciente INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Paciente(Id\_Paciente))

-----------------------------------------------------------

**Tabla Cancelación Turno**

*Tabla en la que se representan las cancelaciones hechas tanto por los afiliados como por los profesionales. Posee una PK “Id\_Cancelacion” subrogada y autoincremental. A su vez posee un check que limita a indicar quien fue el autor de la cancelacion (A=afiliado, M=medico)*

CREATE TABLE STRANGER\_STRINGS.Cancelacion\_Turno(

Id\_Cancelacion INT IDENTITY(1,1) PRIMARY KEY,

Tipo\_Cancelacion CHAR(1) CHECK(Tipo\_Cancelacion = 'A' OR Tipo\_Cancelacion = 'M'),

Motivo VARCHAR(225))

-----------------------------------------------------------

**Tabla Turno**

*Tabla que representa los turnos solicitados en la clinica por parte de los afiliados. Posee una PK “Turno\_Numero” subrogada y autoincremental. Posee una FK “Id\_Paciente” referenciando la PK del mismo nombre en la tabla paciente a fin de relacionar el turno con el paciente que lo solicitó. Del mismo modo posee una FK “Id\_Medico\_x\_Esp” la cual referencia a la PK “Id” de la tabla Especialidad\_X\_Medico a fin de relacionar al medico y la especialidad del mismo sobre el cual se solicito un turno. Por ultimo posee 3 FKs (“Id\_Consulta”,”Id\_Cancelacion” y “Id\_Horario) las cuales referencian a las PKs del mismo nombre en sus respectivas tablas*

CREATE TABLE STRANGER\_STRINGS.Turno(

Turno\_Numero INT IDENTITY(1,1) PRIMARY KEY,

Turno\_Fecha DATETIME,

Id\_Paciente INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Paciente(Id\_Paciente),

Id\_Medico\_x\_Esp INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Especialidad\_X\_Medico(Id),

Id\_Consulta INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Consulta(Id\_Consulta),

Id\_Cancelacion INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Cancelacion\_Turno(Id\_Cancelacion),

Id\_Horario INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Horarios\_Agenda(Id\_Horario))

-----------------------------------------------------------

**Tabla Registro\_Cancelacion\_Medico**

*Tabla adicional que sirve al sistema a fin de registrar los intervalos en los que el medico canceló sus horarios, ya sea por dia o por rango. Posee una PK “Id\_Registro” subrogada y autoincremental.*

CREATE TABLE STRANGER\_STRINGS.Registro\_Cancelacion\_Medico(

Id\_Registro INT IDENTITY(1,1) PRIMARY KEY,

Id\_Med INT FOREIGN KEY REFERENCES STRANGER\_STRINGS.Medico(Id\_Medico),

Dia\_Desde DATETIME,

Dia\_Hasta DATETIME

)

***Stored Procedures:***

**SP\_LOGIN:**

Este procedimiento recibe como parametros @Usuario y @Pass como varchar(255), evualua la existencia de dicho usuario en la base de datos, y de ser asi devuelve por OUTPUT el valor 1, en caso contrario devuelve 0.

**SP\_ACTUALIZAR\_INTENTOS:**

Este procedimiento disminuye en uno la cantidad de intentos del usuario, dados usuario y contraseña, en caso de que el logueo sea incorrecto.

**SP\_REINCIAR\_INTENTOS:**

Este procedimiento reinicia la cantidad de intentos del usuario correctamente logueado

**SP\_GET\_ROLES:**

Este procedimiento devuelve los roles que tiene asociado un usuario, dado su nombre de usuario(@Usuario) y contraseña (@Pass)

**SP\_GET\_ESPECIALIDADES:**

Este procedimiento lista la totalidad de especialidades que posse un medico dado su numero de documento (@Num\_Doc). Devolviento el codigo y descripcion de la especialidad, junto con el nombre y apellido del medico.

**SP\_PEDIR\_TURNOS\_AFILIADO:**

Este procedimiento lista los turnos que tiene reservados un afiliado, dado su numero de documento(@Num\_Doc). Devuelve los campos Turno\_Numero, Turno\_Fecha, Apellido del medico, Descripcion y codigo de la especialidad.

**SP\_PEDIR\_TURNOS\_MEDICO:**

Este procedimiento lista todos los turnos que tiene asignado un medico, dado su numero de documento (@Num\_Doc). Devuelve el campo Turno\_Fecha

**SP\_CANCELAR\_TURNO\_AFILIADO:**

Este procedimiento, dados los parametros @Turno\_Fecha, @Num\_Doc,@Apellido\_Profesional,@Especialidad\_Codigo,@Tipo\_Cancelacion y @Motivo, cancela un turno, agregando una nueva entrada en la tabla Cancelacion\_Turno, y asignandole al turno dicho Id\_Cancelacion de esa entrada.

**SP\_CANCELAR\_TURNOS\_DIA\_PROFESIONAL:**

Este procedimiento se encarga de, mediante los parametros @Turno\_Fecha,@Num\_Doc,@Tipo\_Cancelacion y @Motivo, cancelar todos los turnos de un medico en un dia, realizando los inserts en la tabla Cancelacion\_Turno y cargando el Id\_Cancelacion en los turnos cancelados.

**SP\_CANCELAR TURNOS RANGO PROFESIONAL:**

Este procedimiento se encarga de, mediante los parametros @Turno\_Fecha,@Num\_Doc,@Tipo\_Cancelacion y @Motivo, cancelar todos los turnos de un medico en un intervalo de dias, realizando los inserts en la tabla Cancelacion\_Turno y cargando el Id\_Cancelacion en los turnos cancelados

**SP\_BUSCAR\_AFILIADO:**

Este procedimiento, dado el numero de documento(@Num\_Doc) de un afiliado, devuelve todos los datos del mismo afiliado.

**SP\_BAJA\_AFILIADO:**

Este procedimiento, dado el numero de documento(@Num\_Doc) de un afiliado, realiza la baja logica de dicho afiliado. Si el afiliado no existiere en la base de datos, esta devolveria un parametro por output con el valor -1. En caso de que el paciente estuviera dado de baja anteriormente, dicho valor de retorno sera -2. Si se realiza todo acorde, el retorno sera 0.

**SP\_MODIFICAR\_AFILIADO:**

Este procedimiento realiza un update en la tabla de afiliados, de un afiliado en particular, sobre los campos Direccion, Telefono, Mail,Fecha\_Nac y Estado\_Civil.

**SP\_REGISTRAR\_RESULTADO\_CONSULTA:**

Este procedimiento, dados los parametros @Id\_Consulta, @Fecha\_Y\_Hora\_Atencion,@Sintomas y @Diagnostico, registra el resultado de una consulta efectuada por el medico. En caso de que la consulta no se rellenare, se completan en NULL los ultimos 3 campos. Ret

**SP\_PEDIR\_TURNO\_MEDICO\_FECHA:**

Este procedimiento, dados los parametros @Num\_Doc(Medico), @Especialidad\_Codigo y @Fecha, devuelve todos los turnos que tiene asignados un determinado medico con su especialidad, para una determinada fecha. Devuelve los campos Turno\_Fecha, Nombre y Apellido del paciente y el Id\_Consulta.

**SP\_CAMBIO\_PLAN:**

Este procedimiento, se encarga de realizar el cambio de plan de un afiliado, en base a los parametros @Num\_Doc,@Motivo,@Descripcion\_Plan\_Viejo,@Descripcion\_Plan\_Nuevo. Realiza el insert pertinente en la tabla Cambio\_Plan, la cual registra los cambios de plan de los afiliados, y actualiza la FK Codigo\_Plan de la tabla paciente.

**SP\_ALTA\_AGENDA:**

Este procedimiento realiza la insercion de un horario de un medico en su agenda, para una determinada especialidad, dados los parametros @Num\_Doc, @Especialidad\_Codigo, @Dia\_Semana, @Hora\_Desde, @Hora\_Hasta.

**SP\_SOLICITAR\_TURNO:**

Este procedimiento, dados los parametros @Fecha\_Turno,@Num\_Doc\_Paciente,@Num\_Doc\_Profesional,@Especialidad\_Codigo, reserva un turno para el afiliado, en la fecha determinada y con un medico, ya determinada su especialidad.

**SP\_OBTENER\_ESPECIALIDADES:**

Este procedimiento lista todas las especialidades existentes. Cuyos campos a listar serán: Especialidad\_Descripcion, Especialidad\_Codigo

**SP\_OBTENER\_MEDICOS:**

Este procedimiento lista todos los médicos existentes en la tabla Médicos. Los campos a listar serán: Nombre, Apellido, Num\_Doc

**SP\_OBTENER\_HORARIOS: REVISAR**

Este procedimiento se encarga de listar todos los Horarios que posee los Turnos de un determinado Profesional en base a su número de documento (@Num\_Doc), su código de especialidad (@Especialidad\_Codigo), y la fecha del turno (@Fecha).

**SP\_TOP5\_CANCELACIONES:**

Este procedimiento se encarga de listar las 5 primeras cancelaciones , agrupando por especialidad y ordenando por cantidad de cancelaciones para esa especialidad de mayor a menor. Se selecciona el semestre desde la app y se envian los parametros @Fecha\_Inicio\_Semestre y @Fecha\_Fin\_Semestre.

**SP\_TOP5\_BONOS\_ESPECIALIDAD:**

Este procedimiento se encarga de listar las 5 primeras especialidades, agrupando por especialidad y ordenando por cantidad de bonos utilizados en consultas de esa especialidad de mayor a menor. Se selecciona el semestre desde la app y se envian los parametros @Fecha\_Inicio\_Semestre y @Fecha\_Fin\_Semestre.

**SP\_TOP5\_PROFESIONALES\_CONSULTADOS:**

Este procedimiento se encarga de listar a los 5 primeros profesionales, agrupando por profesional y especialidad, y ordenando por cantidad de consultas que tuvieron de mayor a menos . Se selecciona el semestre desde la app y se envian los parametros @Fecha\_Inicio\_Semestre y @Fecha\_Fin\_Semestre.

**SP\_TOP5\_PROFESIONALES\_POCAS\_HORAS:**

Este procedimiento se encarga de listar a los 5 primeros profesionales, agrupando por profesional y especialidad, y ordenando por cantidad de horas trabajadas para esa especialidad, de menor a mayor. Se selecciona el semestre desde la app y se envian los parametros @Fecha\_Inicio\_Semestre y @Fecha\_Fin\_Semestre

**SP\_TOP5\_AFILIADOS\_BONOS:**Este procedimiento se encarga de listar a los 5 primeros afiliados , ordenando por cantidad de bonos comprados mayor a menor. Se selecciona el semestre desde la app y se envian los parametros @Fecha\_Inicio\_Semestre y @Fecha\_Fin\_Semestre

**SP\_GET\_PRECIO\_BONO:**

Este procedimiento es utilizado para obtener el Precio (Precio\_Bono\_Consulta) de un determinado Bono (@Descripcion).

**SP\_COMPRA\_BONOS:**Este procedimiento es utilizado para realizar la compra de bonos. Por medio de los parametros @Num\_Doc, @Fecha\_Compra y @Cantidad\_de\_Bonos, se realizan los inserts en la tabla bonos segun la cantidad adquiria y se realiza el insert en la tabla compra, registrando asi la compra del paciente.

**SP\_ALTA\_AFILIADO:**

Este procedimiento es el encargado de realizar el alta de un nuevo afiliado, recibiendo previamente los parametros necesarios para ello. El mismo posee un OUTPUT que devuelve -1 si el paciente ya existe en la base de datos y de lo contrario, devuelve en numero de afiliado raiz.

**SP\_BUSCAR\_PROFESIONAL\_SEGUN\_CRITERIOS:**

Este procedimiento arma una query dinamica en base a los parametros que le lleguen no nulos desde la aplicacion. En base a eso evalua si devolver la totalidad de medicos, o solo los que posean determinada especialidad(@Especialidad\_Codigo), si es que dicho campo es no nulo.

**SP\_LISTAR\_TURNOS\_MEDICO:**

Este procedimiento en base al número de documento (@Num\_Doc), el código de la especialidad (@Especialidad\_Codigo) de un profesional, y una fecha (@Fecha) de un determinado dia, lista los turnos de ese día de dicho médico, cuyos campos serán: Turno\_Numero, Nombre, Apellido, Num\_Doc. Turno\_Fecha. Donde el Nombre y apellido serán los de los pacientes relacionados con dichos turnos.

**SP\_CREAR\_CONSULTA:**Este procedimiento es el último paso del registro de llegada de un paciente. El mismo recibe los parametros @Fecha,@Num\_Doc,@Nro\_Turno,@Id\_Bono con los cuales realiza el insert en la tabla consulta dejándola lista para que el medico termine de completarla. A su vez le asigna el id de esa consulta al turno correspondiente. Posee un OUTPUT que devuelve 1 en caso de que la consulta haya sido registrada y 0 en caso contrario.

**SP\_OBTENER\_FECHAS\_FUTURAS:**

Este procedimiento lista las fechas futuras disponibles de un medico, filtrado por especialidad, partiendo desde el dia de la fecha hasta un mes despues. Recibe los parametros @Num\_Doc, @Especialidad\_Codigo y @Fecha\_Actual

**SP\_OBTENER\_HORARIO\_DISPONIBLE\_PARA\_FECHA:**

Se correlaciona con SP\_OBTENER\_FECHAS\_FUTURAS, ya que la app manda la @Fecha seleccionada posteriormente a la ejecucion de dicho SP. Este procedimiento lista los horarios disponibles para ese medico(@Num\_Doc) junto con su @Especialidad\_Descripcion.

**SP\_GET\_ESPECIALIDADES\_ABM\_ROL:**

Este procedimiento se utiliza para obtener las Descripciones y el Estado (habilitado, E, o deshabilitado, D) de todos los Roles existentes.

**SP\_MODIFICAR\_NOMBRE\_ROL:**

Este procedimiento permite modificar el nombre (@Nombre) de un rol (@Rol\_Descripcionl) suplantando al viejo nombre.

**SP\_ELIMINAR\_FUNCIONALIDAD\_ROL:**

Este procedimiento es utilizado para eliminar una funcionalidad (@Funcionalidad\_Descripcion) a un determinado rol (@Rol).

**SP\_AGREGAR\_FUNCIONALIDAD\_ROL:**Este procedimiento es utilizado para agregar una funcionalidad (@Funcionalidad\_Descripcion) a un determinado rol (@Rol) en caso que dicha funcionalidad no la tuviera. Posee una variable @Retorno output para retornar 0 en el caso que la funcionalidad ya existe en dicho rol, o 1 si pudo agregarse exitosamente.

**SP\_HABILITAR\_ROL:**Este procedimiento permite habilitar un Rol (@Rol) modificando su campo Estado a ‘E’ (Enable) para indicar que está habilitado.

**SP\_ELIMINAR\_ROL:**

Este procedimiento se encarga de eliminar un Rol (@Rol) lógicamente, esto significa que ingresa en el campo Estado de dicho rol ‘D’ para ubicar que esa deshabilitado. Contiene un output @Retorno el cual retorna 0 si el rol a eliminar ya esté deshabilitado o 1 si se logró deshabilitar.

**SP\_AGREGAR\_ROL:**Este procedimiento se encarga de agregar un Rol (@Rol) en caso que ya no exista, contiene un output @Retorno el cual retorna 0 si el rol a agregar ya existe o 1 si se logró agregar.

**SP\_MOSTRAR\_BONOS\_PACIENTE:**

Este procedimiento es el encargado de listar todos los bonos disponibles que tiene un paciente determinado, previa identificación del paciente con su @Num\_Doc. Posee un OUTPUT que devuelve 1 en caso de tener bonos disponibles y 0 en caso contrario.

**SP\_VALIDAR\_AFILIADO:**

Este procedimiento es utilizado para conocer si un afiliado existe en base a su número de documento (@Num\_Doc) y luego a traves de un output (@Retorno) devuelve 1 en el caso que exista o 0 en caso contrario.

**SP\_OBTENER\_FUNCIONALIDADES\_DEL\_ROL:**

En base a un rol que recibe (@Rol) obtiene todas las descripciones de sus funconalidades.

**SP\_OBTENER\_AFILIADO:**

Este procedimiento logra obtener un afiliado buscando el Tipo de documento (@Tipo) y Número de documento (@Num\_Doc) que le llegan. Logrando devolver los campos Apellido, Num\_Doc, Cantidad\_Intentos

**SP\_OBTENER\_PROFESIONAL:**

Este procedimiento en base a @Nombre y @Apellido como varchar(225) devuelve el profesional de con ese nombre y apellido, con sus campos Usuario, Num\_Doc, Cantidad\_Intentos.

***Triggers:***

**TR\_ACTUALIZAR\_CONSULTAS:**

Este trigger after insert se encarga de actualizar el numero de consultas de un afiliado en base a la consulta que fue recientemente insertada en su tabla.

***Funciones:***

**FX\_OBTENER\_RESTO:**

Esta funcion devuelve, en base a un @Num\_Afiliado\_Raiz que recibe, el numero de afiliado resto que deberá asignarse al nuevo afiliado que se esta dando de alta.

**FX\_OBTENER\_ID\_PACIENTE:**

Esta funcion nos abstrae de tener que usar siempre el mismo select para obtener el Id\_Paciente en base a un @Num\_Doc que recibe. Devuelve un Id\_Paciente

**FX\_OBTENER\_CANTIDAD\_DE\_BONOS:**

Esta funcion devuelve, en base a un @Num\_Afiliado\_Raiz y un @Codigo\_Plan que recibe, el la cantidad de bonos que posee ese numero de afiliado para ese codigo de plan.

**FX\_FECHA\_DISPONIBLE:**

Esta funcion evalua, en base a una @Fecha, un @Id\_Medico y un @Id\_Medico\_Esp si en dicha fecha el medico se encuentra disponible para una especialidad determinada. Devuelve 0 en caso de disponibiliad y 1 en caso contrario

***Indices:***

**CREATE NONCLUSTERED INDEX IX\_DNI\_PACIENTE ON STRANGER\_STRINGS.Paciente(Num\_Doc)**

Utilizamos este indice para acelerar todas las operaciones en la que interviniera una busqueda de pacientes, dado que generalmente todas las busquedas se hacian por numero de documento, que es lo que envia la aplicación.

***Vistas:***

No vimos necesaria la utilizacion de vistas en el desarrollo de este trabajo practico.

*Consideraciones en la Migración:*

* En cuanto al número de afiliado raíz de cada paciente, tomamos como decisión realizar un REVERSE del número de documento dado que en principio dicho número es único. Pero en caso de ingresar un número de LE o de LC, dentro del motor se cambia el valor por un múltiplo para evitar repeticiones con el número de DNI
* Respecto a los afiliados y médicos cargados en la tabla Maestra, los campos sexo y estado civil fueron seteados en NULL, permitiendo modificar a futuro los campos de los afiliados.
* Dado que no se especifica la fecha de compra de los bonos en la tabla maestra, optamos por replicar la fecha de impresión de los bonos, de mismo modo con el id\_paciente\_compra y id\_paciente uso
* Conservamos la FK Id\_compra junto con la fecha\_compra del bono, para evitar un join innecesario con la tabla Compra para obtener solo un campo
* Para las consultas existentes en la tabla maestra, la fecha de llegada y la fecha de atención son las mismas y corresponden con la fecha del turno que tenía asociada dicha consulta.
* Asumimos que no existen cancelaciones ni baja de pacientes en la tabla maestra.
* Para setear el número de consulta de los bonos, optamos por ordenar por fecha de llegada las consultas de un paciente y le asignamos al bono correspondiente a esa consulta, el número de fila que ocupaba dicha consulta en el select con order by.
* Para los usuarios, tomamos como premisa utilizar como nombre de usuario el apellido del usuario que se quiere loguear(sea afiliado o médico) y como contraseña el número de documento de dicho usuario.
* Para las funcionalidades, realizamos inserts de las funcionales enumeradas en el enunciado.
* Hicimos un insert del administrador general, con los datos dados por la cátedra, es decir el usuario “admin” y la contraseña “w23e” y otro usuario administrativo de la clínica, cuyo usuario es “administrativo” y la contraseña es “admin1234”.
* Tanto para la tabla Bono como para la tabla Turno, al utilizar una PK autoincremental, activamos el IDENTITY INSERT para realizar la migración y poder hacer un insert sobre la PK tomando los Numero de bonos y turnos existentes en la tabla maestra. Al finalizar la migración de estas dos tablas procedimos a desactivar nuevamente dicho IDENTITY INSERT

*Consideraciones en la Aplicación Desktop:*

* Para el desarrollo de la funcionalidad del rol “Administrador General”, habilitamos todas las funcionalidades disponibles en la aplicación. La diferencia con los afiliados y médicos radica en una selección previa del afiliado, ingresando su dni, al querer realizar alguna operación inherente a este. En cuanto a los médicos, listamos todos los médicos, dando la posibilidad al administrador de que seleccione uno, para poder realizar luego las operaciones inherentes con ese médico seleccionado.
* Para una mejor organización dentro del proyecto decidimos agrupar a las clases asociadas a entidades de la base de datos utilizadas en el aplicativo en una carpeta llamada “BD”.
* En relación con la persistencia, decidimos crear una clase de tipo singleton que contiene todos los métodos asociados a la interacción entre la *App* y la *Base de datos*. La clase se encuentra ubicada dentro de la subcarpeta “BD”. Esta misma posee 3 métodos esenciales que son: GetDataReader(), ExecStoredProcedure() y ExecStoredProcedureAlta(). En la generalidad, usamos el primer método para ejecutar todos aquellos *stored procedures* que no tengan por parámetro una variable de tipo *Output*, mientras que los dos restantes se emplean para ejecutar aquellos que emplean ese tipo de variables con la salvedad de que cada uno retorna un tipo diferente de dato.   
  Los métodos restantes no son más que auxiliares a los 3 descritos anteriormente, ya que solo son llamados por estos métodos y sirven para impedir repetición de lógica a la hora de crear los sqlcommand necesarios para ejecutar los *stored procedures*.
* Para poder resolver el requerimiento del archivo de configuración, decidimos crear un archivo de tipo xml llamando app.config, en donde establecimos los parámetros necesarios para la conexión con la *Base de Datos*. En el mismo se debe definir el Data Source que hace referencia a donde se encuentra alojada la base de datos (Usuario\SQLSERVER2012); el Initial Catalog que se relaciona con el nombre de la base de datos a la cual nos vamos a conectar; el usuario y password con los cuales nos conectaremos a la base de datos; y por último, la fecha que el sistema utiliza.  
  Este archivo de configuración puede configurarse desde Visual Studio o desde el archivo mismo, abriéndolo con un bloc de notas.
* En la pantalla que muestra los listados *“TOP5”*, se debe comprender que a la hora de seleccionar el semestre la opción Ene-Jun corresponde al primer semestre del año y Jul-Dic, al segundo. Por otra parte, para seleccionar el año creamos un *ComboBox* y le establecimos un rango de años acorde a lo observado en la tabla maestra dado que al no ser un sistema productivo real, nos tomamos el atrevimiento de definir el mismo aún siendo conscientes de que estamos generando una deuda técnica.
* Para el ABM de Rol, luego de una serie de discusiones, decidimos resolver los requerimientos asociados a agregar o quitar funcionalidades permitiendo al usuario realizar las acciones de una por vez, es decir, si se quieren agregar más de una funcionalidad, las mismas se deben agregar de a una a la vez.
* Dado que en el listado de requerimientos expuestos en el trabajo práctico se exponían ciertas funcionalidades que no debían ser implementadas, tomamos la decisión de crear los botones de las mismas. Cabe aclarar que al seleccionar los mismos, se visualizará una pantalla emergente que mostrará el siguiente mensaje: *“Funcionalidad no disponible”*.