UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

SISTEMA DE GESTION DE FLOTA DE TRANSPORTE

Oleksandr Volenko

Tutor: José Manuel Mossi García

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2017-18

Valencia, 31 de agosto de 2018

Resumen

El presente trabajo fin de grado está dedicado a realizar un estudio de instalación de sistema de gestión de flotas y automatizar procesamiento de datos obtenidos de dicho sistema.

El proyecto está compuesto por dos partes básicas donde se explica la implementación de la tecnología GPS en una empresa de transporte además de la creación de un programa relacionado con el procesamiento de datos. La primera parte se dedica a definición de los aspectos clave de la tecnología, consideración de diferentes modelos de GPS con sus características, la instalación de dispositivos y tipos de plataformas según sus criterios básicos. La segunda parte se divide en la creación de una base de datos local y una aplicación que permite guardar los datos de GPS en dicha base de datos y trabajar con la información almacenada. Los instrumentos fundamentales aplicados en este trabajo son: la base de datos Microsoft SQL Server 2014 y el entorno de desarrollo de software Embarcadero Delphi XE3.

Índice

Introducción

El éxito de cualquier empresa depende en gran medida de la planificación e implementación adecuadas del proceso de trabajo. Concretamente las empresas logísticas, el cálculo preliminar de los costos financieros, la planificación racional de las rutas y el control del transporte son las tareas prioritarias para la cooperación beneficiosa de la empresa con el cliente. No es, por tanto suficiente, tener solo una gran flota de vehículos. En primer lugar, es necesario planificar correctamente la gestión de su trabajo, calcular todos los costos y optimizar los gastos.

Conseguir la entrega cualitativa y oportuna, independientemente de la distancia, el volumen y el peso de la carga, se puede hacer por medio del monitoreo. El monitoreo permite conseguir la entrega de las cargas a tiempo, es esencial reducir los gastos para el combustible que influirá positivamente sobre la rentabilidad de toda la empresa de transporte, planear las rutas ulteriores tomando en cuenta los datos estadísticos recibidos. Con la instalación del sistema del *monitoring* y *management* del transporte, la compañía recibe la posibilidad de la observación de flota en el tiempo real, el control del nivel del combustible, el trabajo en la itinerancia, el control del estado de las cargas, el bloqueo del motor, las comunicaciones de voz bilaterales con el conductor, etc.

Toda esta información se conserva en la base de datos y es accesible para el cliente a través del software especial o la página web proveedora del GPS. En general todos los datos son concedidos en la interfaz conveniente con la posibilidad del análisis de las rutas en un cierto intervalo del tiempo. Pero hasta el sistema más desarrollado del monitoreo puede encontrarse limitado en las ciertas situaciones y no definir con claridad las tareas necesarias para la planificación eficaz de las rutas. En este caso la empresa proveedora del sistema de monitoreo puede entregar al cliente los datos iniciales en un formato necesario. Y con la creación de una base de datos y plataforma conveniente para la decisión de las tareas necesarias, se puede conseguir un trabajo más cómodo y eficaz de toda la empresa.

Tomando en consideración los grandes volúmenes de datos que se manejan y el mínimo tiempo que se desea invertir en su tratamiento, se llega a la conclusión de que la automatización del proceso de trabajo por medio del software específico es el medio más adecuado para lograr la máxima eficacia. De este modo se agiliza el trabajo considerablemente y se aumenta la productividad. Algo muy positivo es que cualquier persona con la mínima formación puede manejar el software y conseguir una optimización

notable del proceso de trabajo. Se trata de una actividad con muchas perspectivas de futuro y grandes posibilidades de crecimiento.

Tecnología GPS

El GPS (Global Positioning System: sistema de posicionamiento global) o NAVSTAR-GPS1 es un sistema global de navegación por satélite (GNSS) que permite determinar en todo el mundo la posición de un objeto, una persona, un vehículo o una nave, con una precisión hasta de centímetros (si se utiliza GPS diferencial), aunque lo habitual son unos pocos metros de precisión.

Ilustración 1: Esquema básico de GPS

Fuente: http://sgrastreo.com.mx/tecnologia-GPS-GPRS-SMS.php (Consultado el 31 de agosto de 2018)

El GPS funciona mediante una red de 32 satélites (28 operativos y 4 de respaldo) en órbita sobre el globo, a 20.200 km, con trayectorias sincronizadas para cubrir toda la superficie de la Tierra. Cuando se desea determinar la posición, el receptor que se utiliza para ello localiza automáticamente como mínimo tres satélites de la red, de los que recibe unas señales indicando la identificación y la hora del reloj de cada uno de ellos. Con base en estas señales, el aparato sincroniza el reloj del GPS y calcula el tiempo que tardan en llegar las señales al equipo, y de tal modo mide la distancia al satélite mediante "triangulación" (método de trilateración inversa), la cual se basa en determinar la distancia de cada satélite respecto al punto de medición. Conocidas las distancias, se determina fácilmente la propia posición relativa respecto a los tres satélites. Conociendo, además, las coordenadas o posición de cada uno de ellos por la señal que emiten, se obtiene la posición absoluta o coordenadas

reales del punto de medición. También se consigue una exactitud extrema en el reloj del GPS, similar a la de los relojes atómicos que llevan a bordo cada uno de los satélites.

AVL (Localización Automática de Vehículos). Es un sistema que sirve para determinar la posición de un vehículo con coordenadas de Latitud y Longitud. Para ello utiliza la tecnología de GPS (Sistema Global de Posicionamiento); por lo general, estas coordenadas son transmitidas a la base en tiempo real o en post proceso.

Dispositivos

Para conocer la posición de la flota debemos poseer un receptor GPS en cada vehículo.
Hay dos principales tipos de dispositivos receptor: internos (son los que se conectan en la cabeza tractora) y externos (especiales para plataformas, contenedores, frigos, mercancías).

En [2], el catálogo de GlobalAVL, podemos consultar numerosos modelos con diferentes características. Como por ejemplo **Albatros GPRS 6**:

Ilustración 2: Localizador Albatros GPRS 6

Fuente: https://www.globalavl.es/localizadores-gps/gps-vehiculos/315-albatross-gprs-6-gps-vehiculos.html (Consultado el 31 de agosto de 2018)

Marca: Albatross System
Modelo: GPRS 6
Mercado Objetivo: Rastreo y Seguimiento, Recuperación de Vehículo, Gestión de Flotas, Mensajes al conductor
Banda GSM: Cuatribanda
Voz: No
Back up de la batería: Si
Memoria Interna: Si
Método de comunicación: GPRS, TCP, UDP
Posición por: Tiempo, Distancia, Cambio de ángulo
Ahorro de Energía: Si
Entradas Digitales: 2
Entradas predefinidas: Ignición, Puertas
Entradas Analógicas: 2
Eventos Internos: Si
Antenas: GPS, GSM
Cubierta: Plástica
Conexiones Extras: Antenas externas, Sensor de combustible, Sensor de Temperatura, Módulos inalámbricos, 1-wire
Can Bus: J1708, Extended
Fabricado en: Europa

Ilustración 3: Características técnicas de Localizador Albatros GPRS 6

Fuente: https://www.globalavl.es/localizadores-gps/gps-vehiculos/315-albatross-gprs-6-gps-vehiculos.html (Consultado el 31 de agosto de 2018)

Estos tipos de GPS necesitan del protocolo *Can Bus* del vehículo de donde tomamos los datos del vehículo.

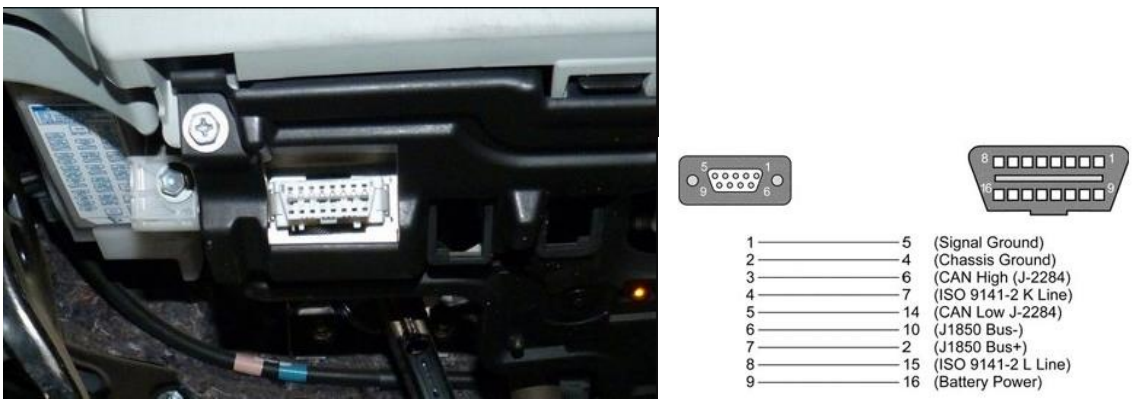


Ilustración 4: OBD conexión

CAN (acrónimo del inglés *Controller Area Network*) es un protocolo de comunicaciones, basado en una topología bus para la transmisión de mensajes en entornos distribuidos. Esto quiere decir que hay un solo cable que recorre el vehículo al que se van conectando los diferentes aparatos electrónicos que necesiten comunicarse. Además, ofrece una solución a la gestión de la comunicación entre múltiples CPUs. Cualquier dispositivo electrónico

conectado al bus puede mandar mensajes y el resto le escuchan. Cada tipo de mensaje lleva un identificador. Los oyentes deciden qué mensajes les interesan y cuáles no. Para que la cosa funcione, los dispositivos eléctricos se van turnando para "hablar" de uno en uno. El protocolo está muy bien descrito en [4].

Para los dispositivos externos conectados en las plataformas lo más importante es tener suficientes días de autonomía, es decir la batería debe ser de larga duración y consumo mínimo, y también disponer de la memoria interna. Para remolques y contenedores frigoríficos los dispositivos GPS tienen las mismas características comentadas anteriormente añadiendo uno o varios sensores de temperatura.

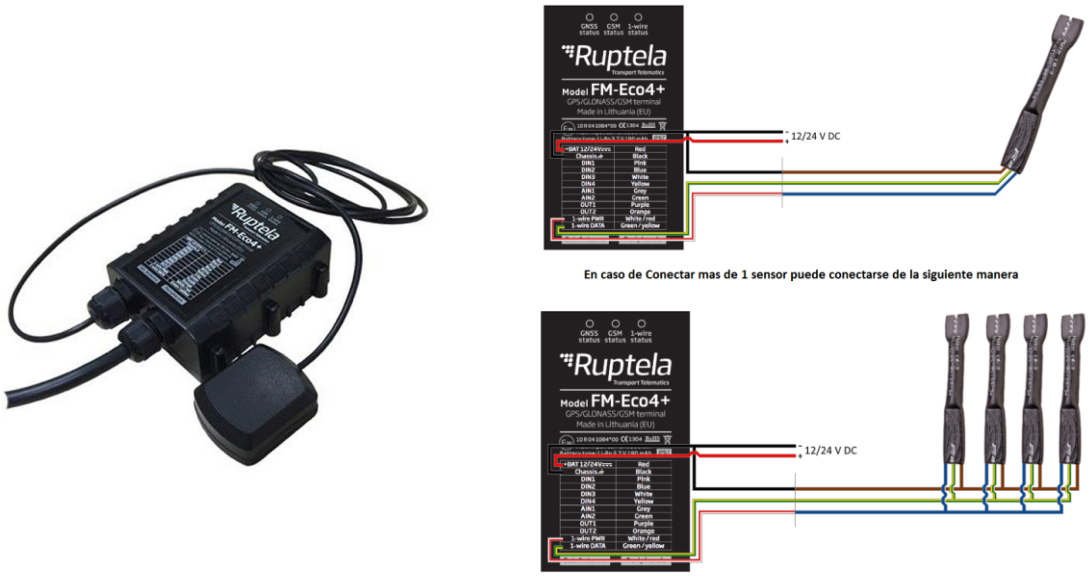


Ilustración 5: Localizador GPS ECO4+ y sensor de temperatura

Fuente: https://www.globalavl.es/localizadores-gps/gps-vehiculos.html (Consultado el 31 de agosto de 2018)

Plataformas

Para obtener la información de las unidades GPS, almacenar, procesar y visualizar de forma intuitiva con gráficos y estadísticas tenemos que disponer de una plataforma.

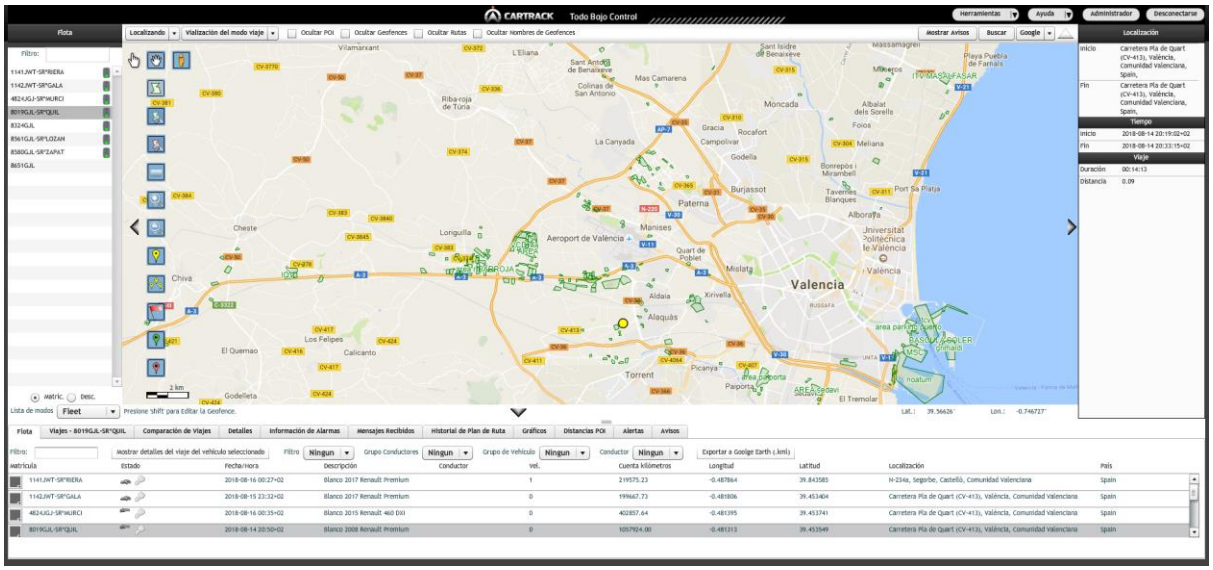


Ilustración 6: Plataforma Cartrack

Los proveedores ofrecen dos tipos de plataformas: plataforma GPS hosting (alquiler) y plataforma GPS personal (local). Hay tres criterios básicos de comparación entre solución plataforma GPS y solución local:

CRITERIO	**PLATAFORMA GPS**	**LOCAL**
Actividad	Centro de Datos de proveedor	Servidor de cliente
Política de pago	Anual / Mensual	Un único pago
Máx. de unidades soportadas	Sin límite	Aprox. 10.000

Las estructuras de ambas soluciones son muy parecidas, la diferencia principal depende de la ubicación de la base de datos y gestión de esta.

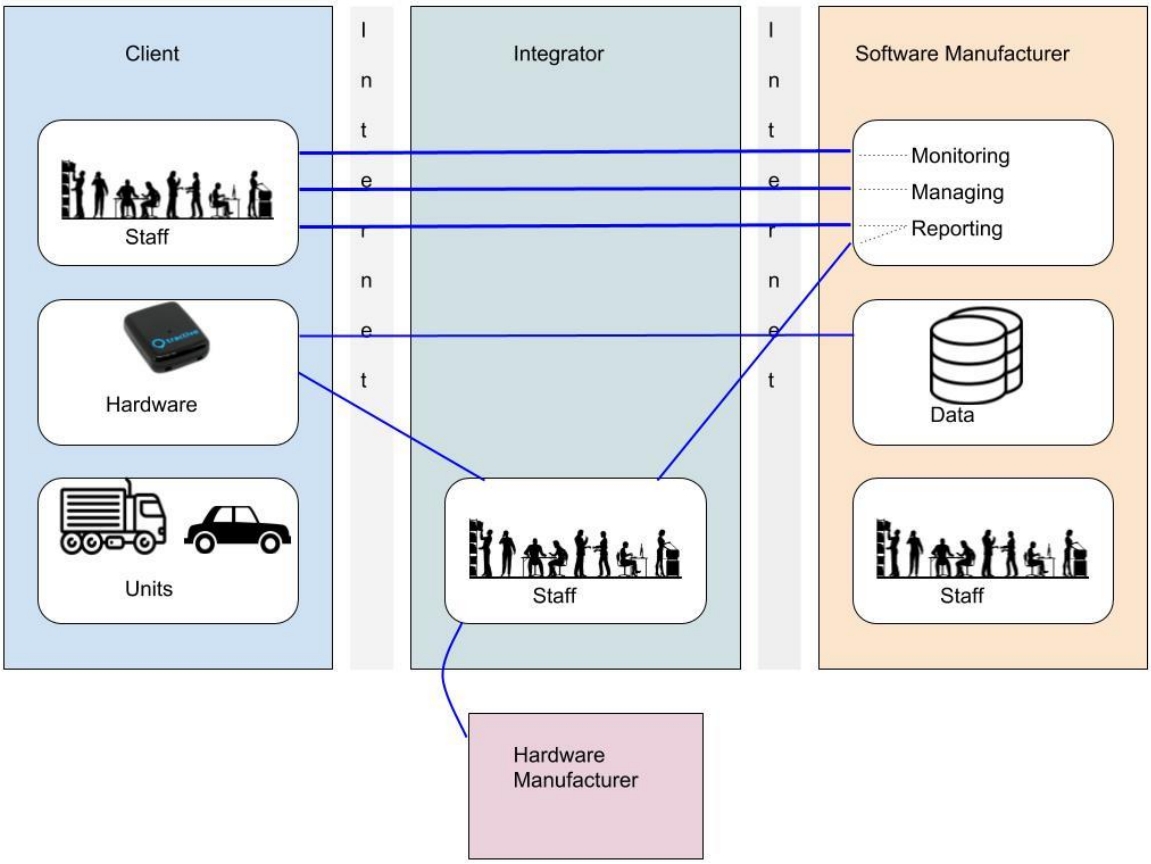


Ilustración 7: Estructura de plataforma GPS

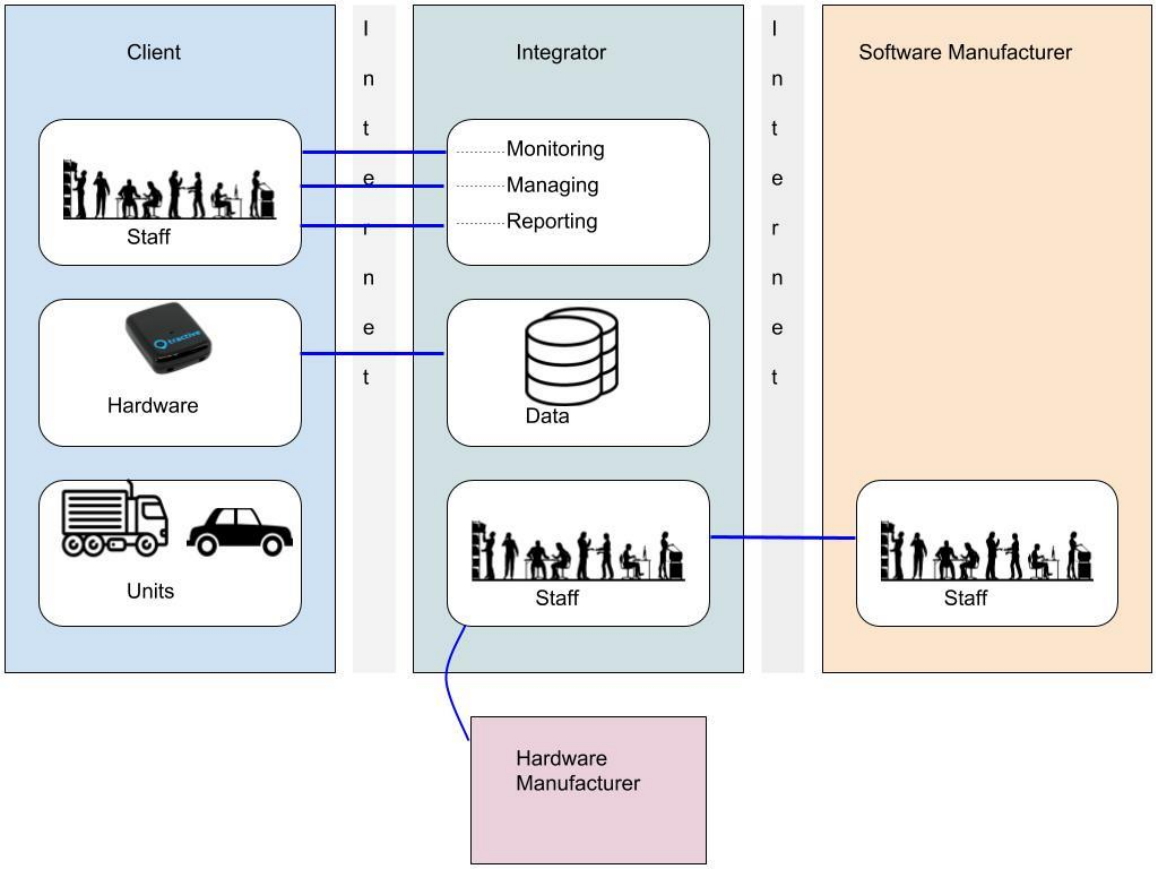


Ilustración 8: Estructura de plataforma Local

Las plataformas disponen también de varios componentes adicionales.

- **Informes.** El módulo permite crear diferentes informes y gráficos sobre los parámetros, que se detectaron en el sistema de rastreo GPS. Los informes se generan en base a las plantillas que posee el software y parámetros establecidos totalmente personalizables. Se puede incluir cualquier número de tablas o gráficos y mostrar la información sobre el periodo de tiempo establecido, generar informes sobre los grupos de unidades, programar envío de las plantillas a un correo electrónico con periodicidad establecida. Hay opciones para imprimir los informes y también para exportarlos a formatos HTML, PDF, Excel, XML, CSV etc.

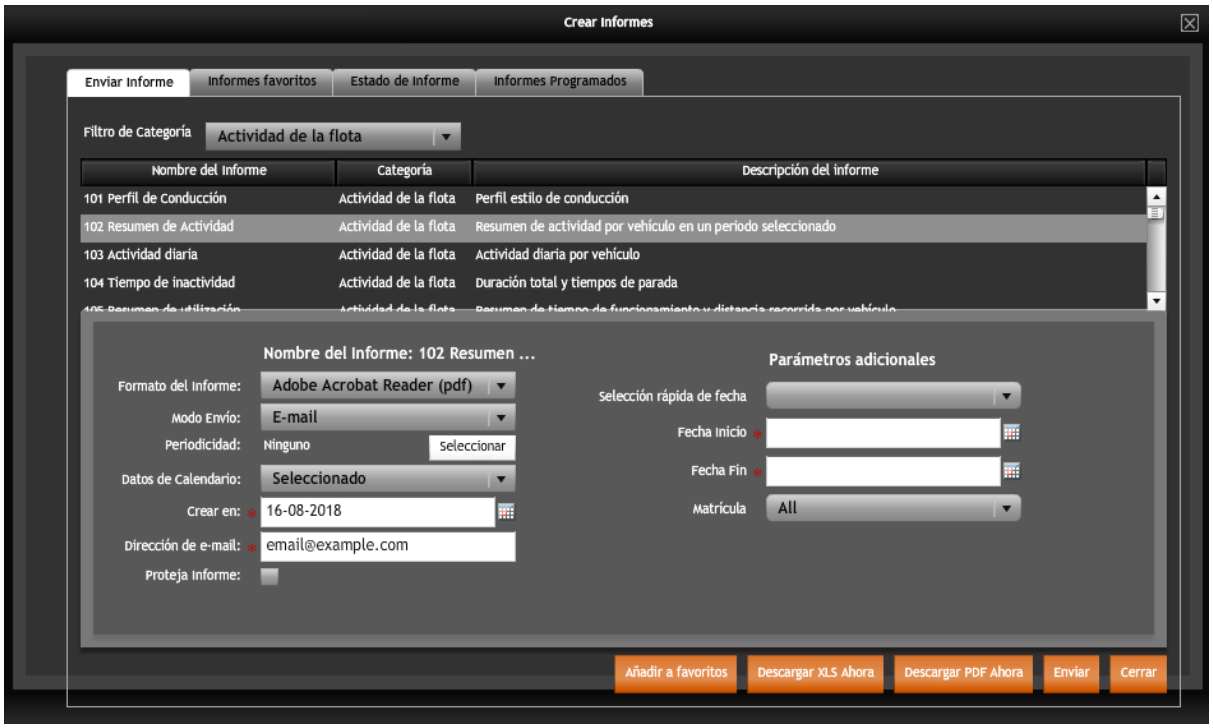


Ilustración 9: Informes de la plataforma Cartrack

- **Control de combustible.** Existe la posibilidad de utilizar los sensores de combustible en el sistema de seguimiento de GPS y calcular el nivel y consumo de combustible.

Ilustración 10: Control de combustible

- **Geofences.** Dispone de funciones de GEOCERCAS (geofences), zonas de aviso o puntos de paso para saber cuándo un dispositivo entra o sale de las mismas, así como localización en el mapa sus propios puntos de interés.

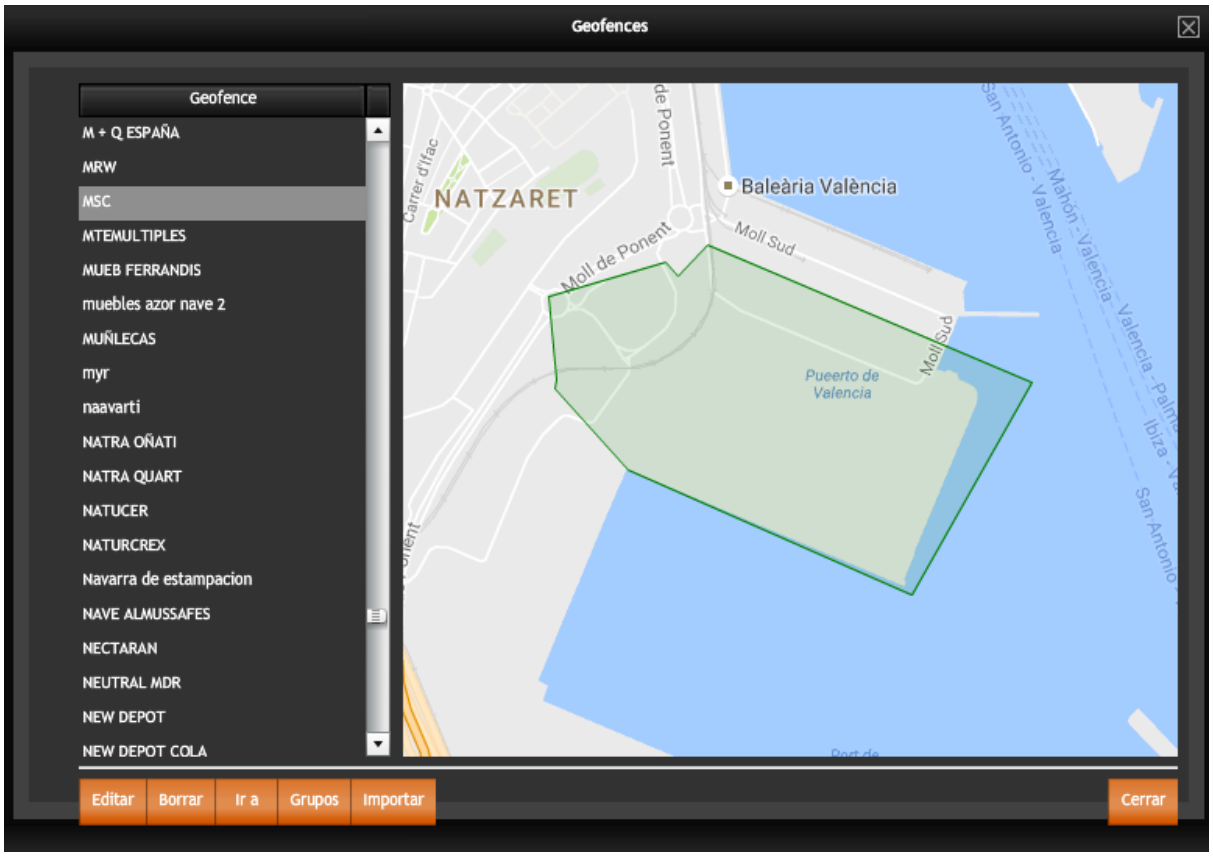


Ilustración 11: Pantalla de gestión de geofences

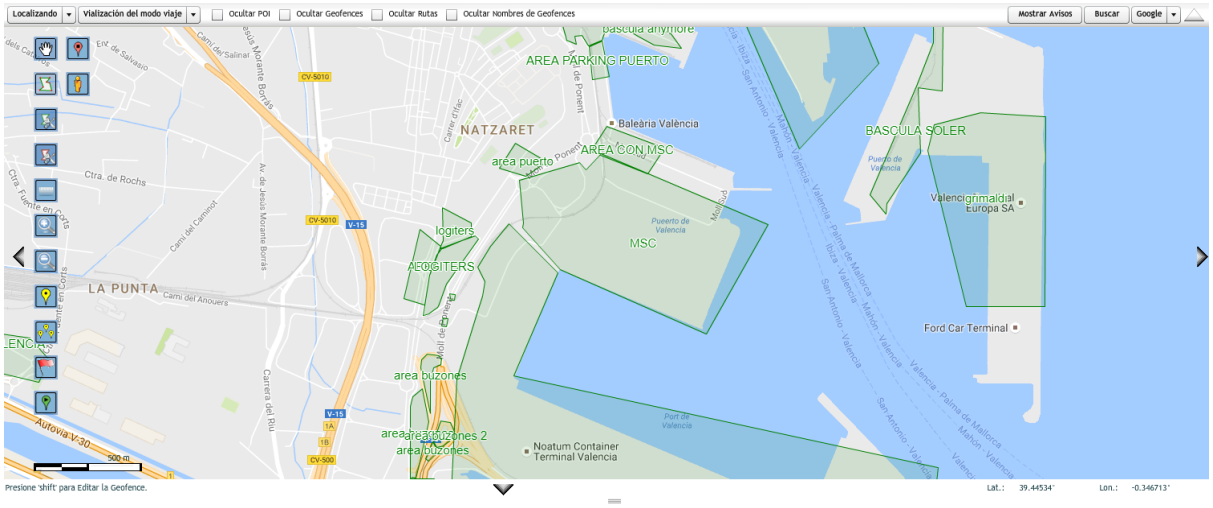


Ilustración 12: Geofences marcados en el mapa

- **Planificador de ruta.** Este módulo permite crear rutas y realizar un seguimiento de una unidad que realiza una ruta con puntos de control definidos en un orden predefinido o arbitrario, en tiempo definido o sin horario estricto.

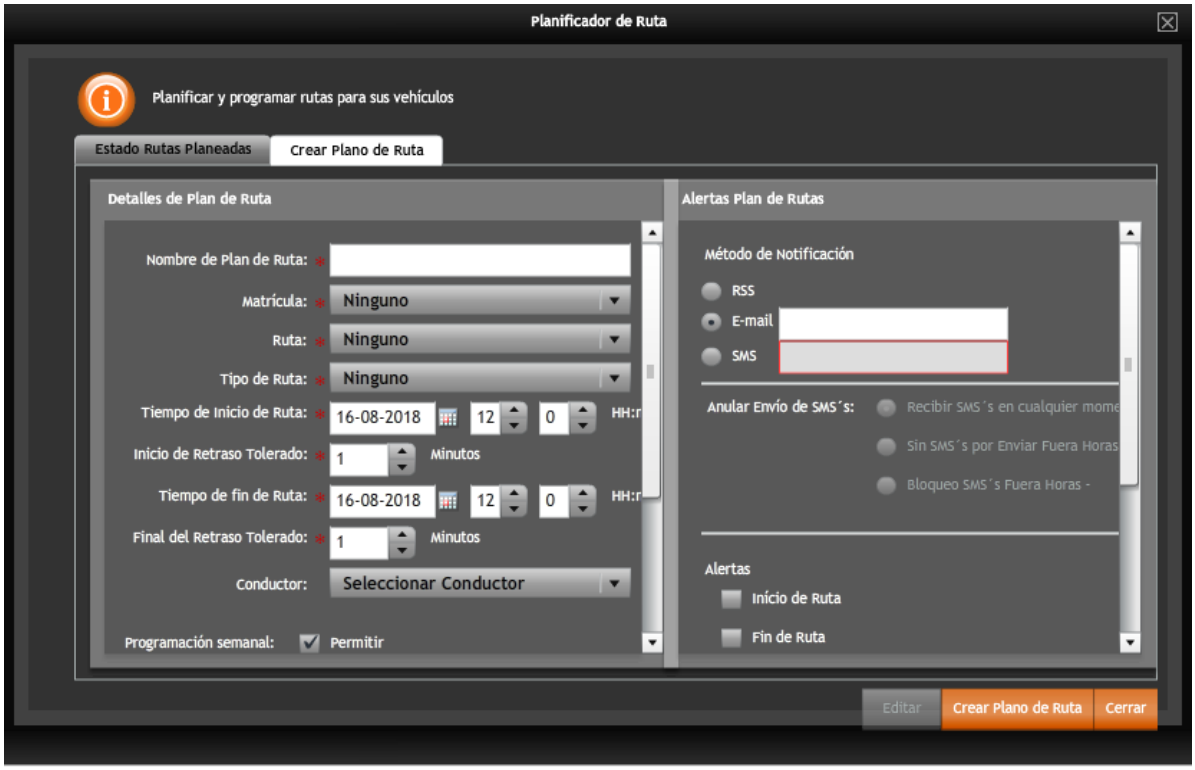


Ilustración 13: Planificador de ruta

- **Alarmas.** Las alarmas permiten envío de mensajes de información sobre cualquier actividad de una unidad (exceso de límites de velocidad, salidas de geofences, etc). Una notificación puede ser enviada por correo electrónico, SMS, o aparece en Histórico de Alarmas.

Ilustración 14: Alarma de gestión de flota

Casi todas las plataformas ponen a disposición de clientes aplicaciones complementarias para obtener información de sus unidades GPS de forma diferente, disfrutando de sencillos gráficos y estadísticas, pero no siempre son suficientes para las necesidades de la empresa. En este caso utilizando los datos de la plataforma realizamos nuestra aplicación sobre la base de los programas corporativos.

Programación

Objetivo

El objetivo de esta parte es crear una aplicación basada en los programas de la empresa que permite guardar, reordenar, hacer cálculos y visualizar los datos obtenidos de la plataforma GPS y según las necesidades del cliente. En concreto, tenemos que hacer una base de datos para toda la información útil de Cartrack, a partir de estos datos poder sacar por pantalla una tabla detallada sobre los viajes de todos los choferes para cualquier periodo de tiempo, y por último hacer los cálculos para una hoja de dietas de los choferes. Para la realización de esta tarea disponemos de dos componentes principales: un entorno de desarrollo de software y una base de datos.

Base de datos

En primer lugar, sería conveniente familiarizarse con los conceptos que son necesarios para trabajar con Microsoft SQL Server 2014.
Al abrir el programa lo primero que tenemos que hacer es conectar con el servidor

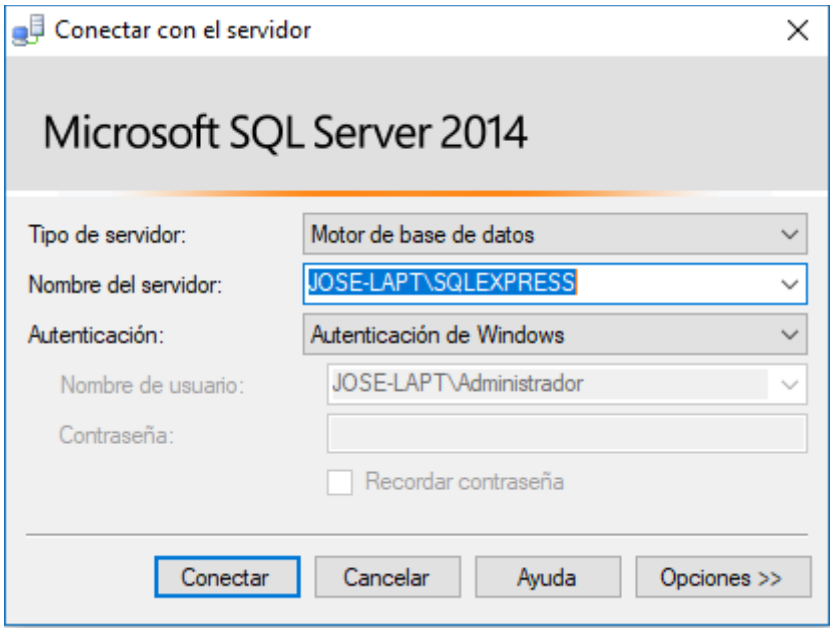


Ilustración 15: Conexión con el servidor

El tipo de servidor que nos interesa es Motor de base de datos. Nuestro servidor local va a ser el propio ordenador con el servicio de Windows *SQLexpress* y autenticación del sistema. Como en el sistema Windows no todos los servicios están activos por defecto, es posible obtener un mensaje de error en la primera ejecución:

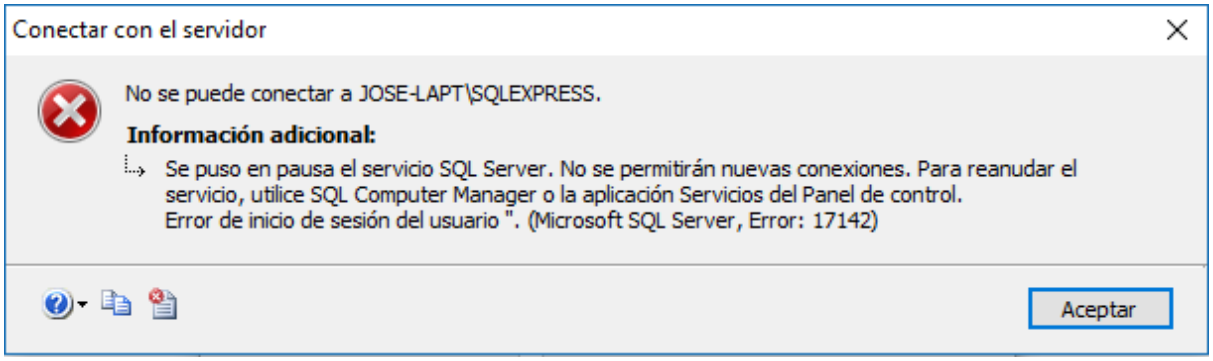


Ilustración 16: Mensaje de error de conexión con SQL Server

Para solucionar este tipo de error procedemos a la ventana de Servicios de Windows y activamos el servicio SQLEXPRESS

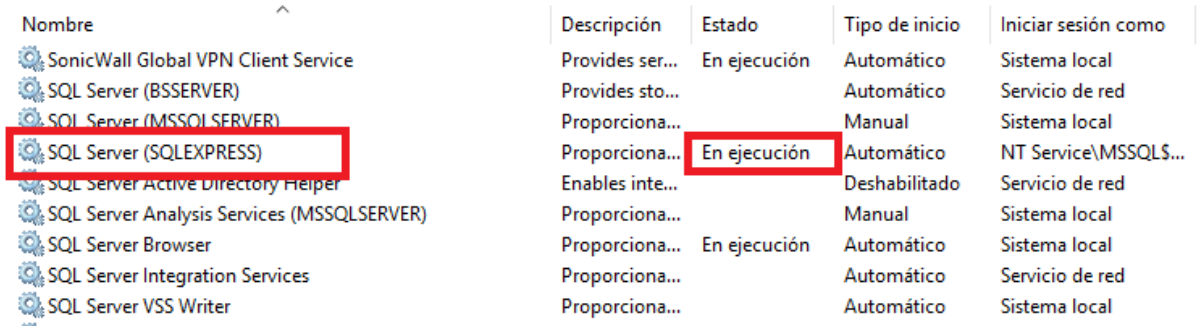


Ilustración 17: Ventana de Servicios de Windows

Al final procedemos a la pantalla principal de Microsoft SQL Server donde encontraremos Explorador de objetos, que contiene las bases de datos del sistema con sus tablas correspondientes, la ventana de diseño y consultas.

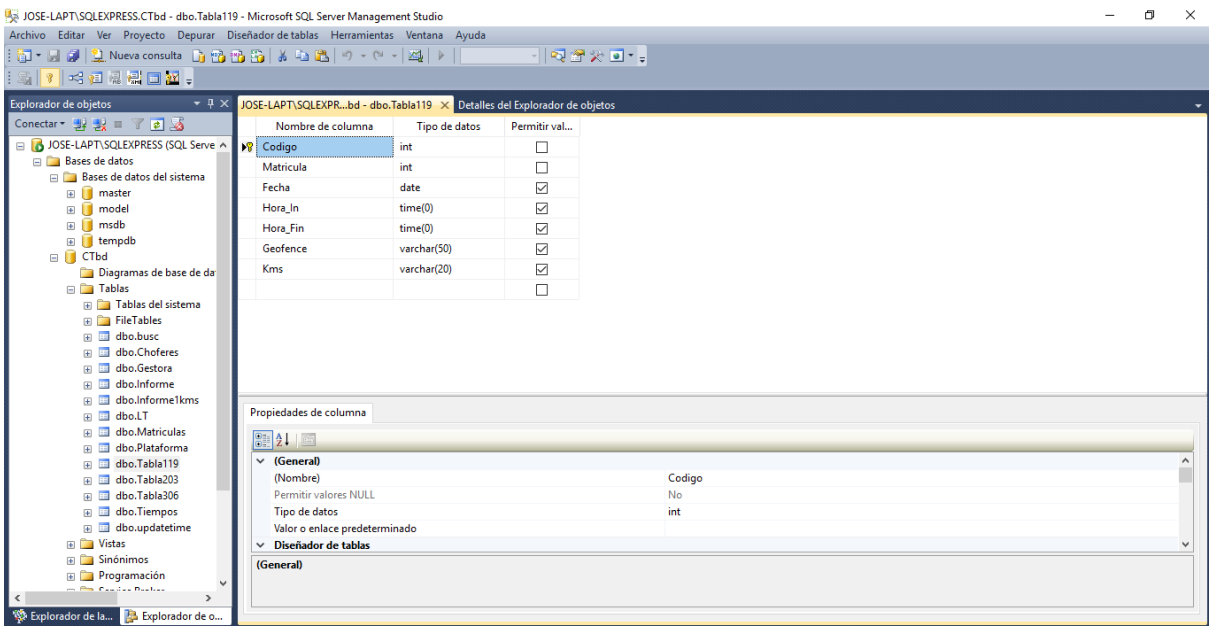


Ilustración 18: Interfaz de la base de datos SQL server

S.Q.L. significa lenguaje estructurado de consulta (Structured Query Language). Es un lenguaje estándar de cuarta generación que se utiliza para definir, gestionar y manipular la información contenida en una Base de Datos Relacional. En este lenguaje no se deben especificar todos los pasos que hay que dar para conseguir el resultado, tan solo deberemos indicar qué es lo que queremos obtener, y el sistema decidirá cómo obtenerlo.

Es un lenguaje sencillo y potente que se emplea para la gestión de la base de datos a distintos niveles de utilización: usuarios, programadores o administradores de la base de datos.

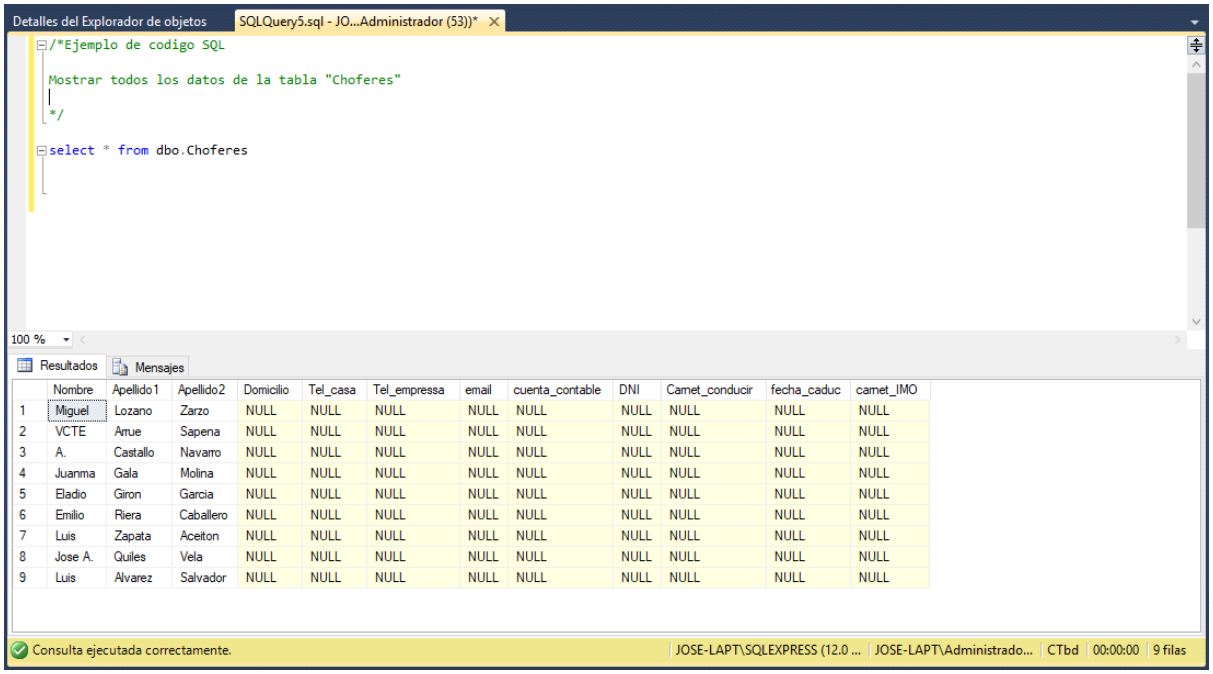
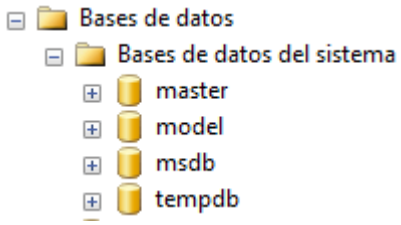


Ilustración 19: Código SQL

En cada instalación de SQL Server hay cuatro bases de datos de sistema, y la capacidad de crear nuevas bases. Los datos quedan almacenados en forma de tablas en dichas bases. Las cuatro bases instaladas por defecto son las siguientes:



- **master** – Se define como el conjunto de procedimientos, funciones y tablas del sistema que están utilizadas por parte de todas las bases de datos y que están instaladas automáticamente, incluyendo también las que han sido creado por parte de los administradores del sistema. Además, todas las definiciones en respecto a la seguridad a nivel del servidor, están almacenadas en esta base de datos.
- **model** – Se refiere al molde de las bases de datos. Cada nueva base de datos se crea como una copia de ésta, a menos que algo más estuviera definido explícitamente.

- **msdb** - Almacenamiento de las tareas del agente, los códigos de CLR combinados en el sistema, los paquetes de SSIS, y otros más.
- **tempdb** - Base de datos temporal que se crea de nuevo cada vez que el servicio se reinicia. Se utiliza para almacenar tablas temporales creadas por parte de los usuarios o el sistema (por ejemplo, en ordenaciones complejos).

El siguiente paso es crear una base de datos nueva. Para ello con el botón derecho sobre la carpeta *Bases de datos* elegimos la opción *Nueva base de datos*. En esta ventana ponemos el nombre para nuestra base de datos y el resto de las opciones las dejamos sin cambiar.

En *Ruta de acceso* podemos consultar la carpeta donde está almacenada la base de datos creada para poder copiarla y/o llevarla al otro sitio.

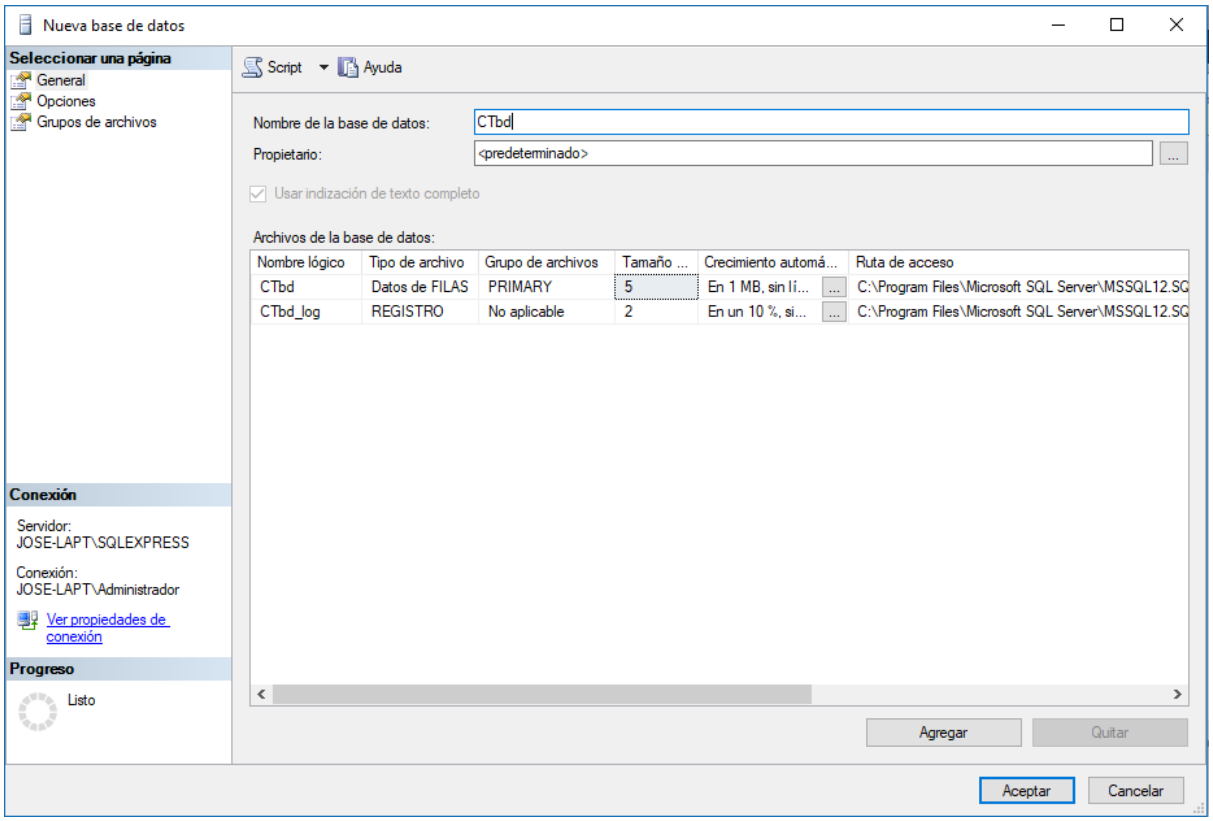
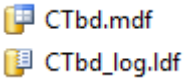


Ilustración 20: Ventana de creación de nueva base de datos nueva



Esta base de datos creada incluye básicamente un archivo de datos (con el sufijo mdf) con las tablas y los distintos objetos a nivel de la base de datos; y un archivo de registro (con el sufijo ldf) con las transacciones abiertas, y transacciones cerradas. Se puede crear un conjunto de archivos de datos además del principal (con el sufijo ndf) por consideraciones de eficiencia, partición de carga de trabajo entre los discos rígidos, etc.

Dentro de la base de datos creada agregamos todas las tablas necesarias para nuestro trabajo, tanto fijas como temporales. Dado que de momento no sabemos la cantidad de tablas necesarias crearemos una con el formato del informe final solicitado por el cliente y resto de tablas las crearemos en el proceso de trabajo.

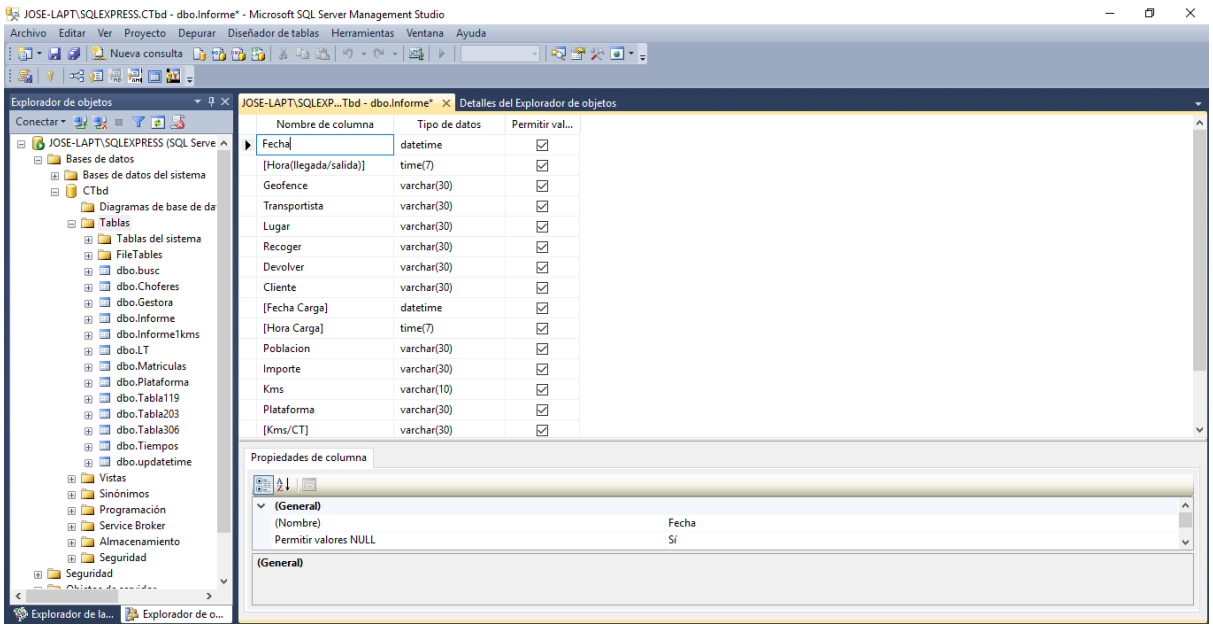


Ilustración 21: Ventana de diseño de tablas

Una tabla es una estructura de datos que organiza los datos en columnas y filas; cada columna es un campo (o atributo) y cada fila, un registro. La intersección de una columna con una fila contiene un dato específico, un solo valor. Cada registro contiene un dato por cada columna de la tabla. Cada campo (columna) debe tener un nombre. El nombre del campo hace referencia a la información que almacenará. Cada campo (columna) también debe definir el tipo de dato que almacenará.

A las tablas se pueden asociar índices. Los índices se almacenan junto a la tabla (Non Clustered Index) o son la tabla en sí (Clustered Index). Los índices asisten en la búsqueda de datos en las tablas (como los ficheros en las librerías) y ayudan a ordenarlas y a definir las claves primarias. También, entre las tablas se puede crear una relación de uno a muchos. En nuestro caso para una sola tabla final no hace falta crear índices ni relaciones.

Después de rellenar los datos de la tabla vamos a nuestro entorno de desarrollo de software, creamos la conexión con nuestra base de datos con la herramienta *ADOConnection* y todas las modificaciones sobre las tablas las realizaremos implementando el código de programa.

Delphi

Dado que la mayoría de los programas corporativos se basan en Embarcadero Delphi escogemos el mismo como el entorno de desarrollo de software para nuestra aplicación.

Delphi es un entorno de desarrollo que se creó con el propósito de agilizar la creación de software basándolo en una programación visual. En Delphi se utiliza una versión más actual del Pascal conocida como Object pascal como lenguaje de programación. Este lenguaje es muy versátil y se usa para casi cualquier proyecto como por ejemplo aplicaciones de consola, conectividad con bases de datos, servicios del sistema operativo, establecer comunicación entre servidor web y un programa, para realizar aplicaciones visuales, etc. Es un lenguaje que produce aplicaciones en código máquina, por lo que la computadora las interpreta inmediatamente y no precisa de un lenguaje intérprete como es necesario en otros lenguajes de programación. Además, una de las principales ventajas de Delphi es su capacidad para desarrollar aplicaciones con conectividad a bases de datos de diferentes fabricantes. El motor de conexión a bases de datos Borland Database Engine (BDE) permite manejar bases de datos de escritorio como dBase, Foxpro, Paradox y también conectarse a servidores SQL locales y remotos.

Como entorno visual, la programación en Delphi consiste en diseñar los formularios que componen al programa colocando todos sus controles (botones, etiquetas, campos de texto, etc.) en las posiciones deseadas, normalmente usando un ratón. Posteriormente se asocia un código a los eventos de dichos controles.

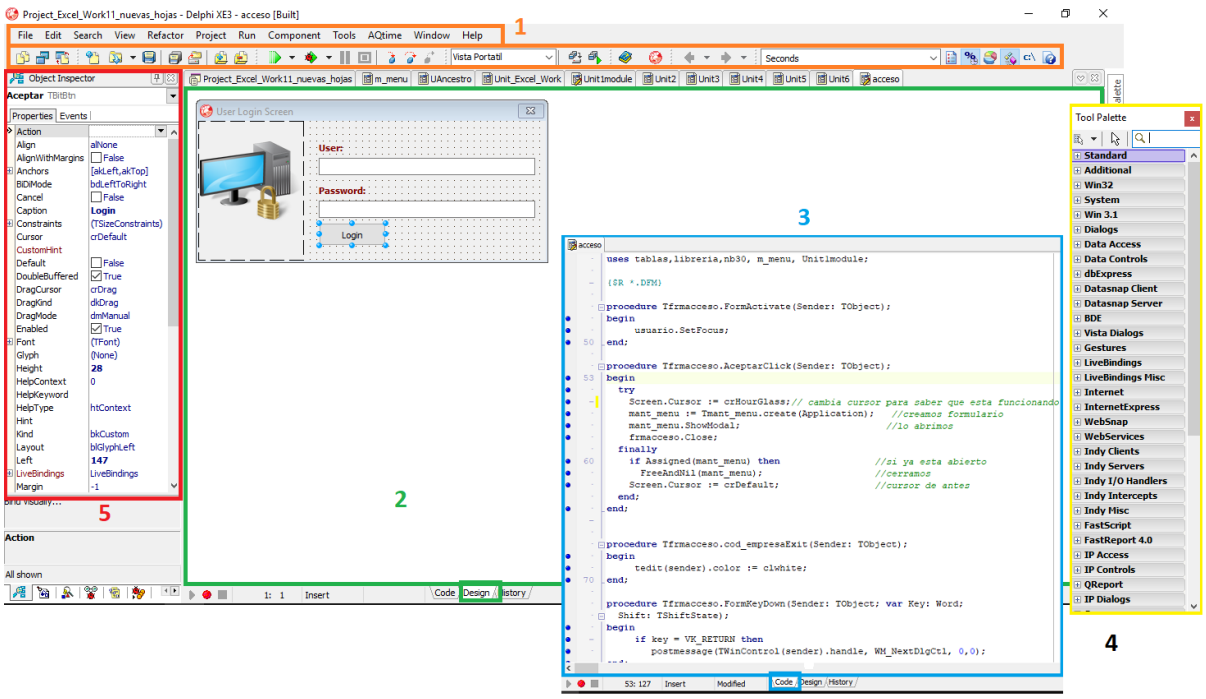


Ilustración 22: Entorno de Delphi

El entorno de Delphi está dividido en cinco partes principales:

1) Menú de herramientas.
2) Formulario, donde colocamos nuestros componentes
3) Editor de codigo
4) Componentes
5) Inspector de objetos. Contiene una lista de propiedades y eventos para un componente

En primer lugar, tenemos que crear un formulario nuevo para poder añadir componentes necesarios. En menú de herramientas *File* → *New* → *Other* obtenemos la ventana con todo tipo de *Items* disponibles en un proyecto en Delphi.

Conexión a la base de datos

Antes de empezar a construir nuestro programa tenemos que hacer la conexión entre el programa y la base de datos creada anteriormente. Para ello tenemos que seleccionar un módulo de datos *Data Module* ubicado en la carpeta *Delphi Files*.

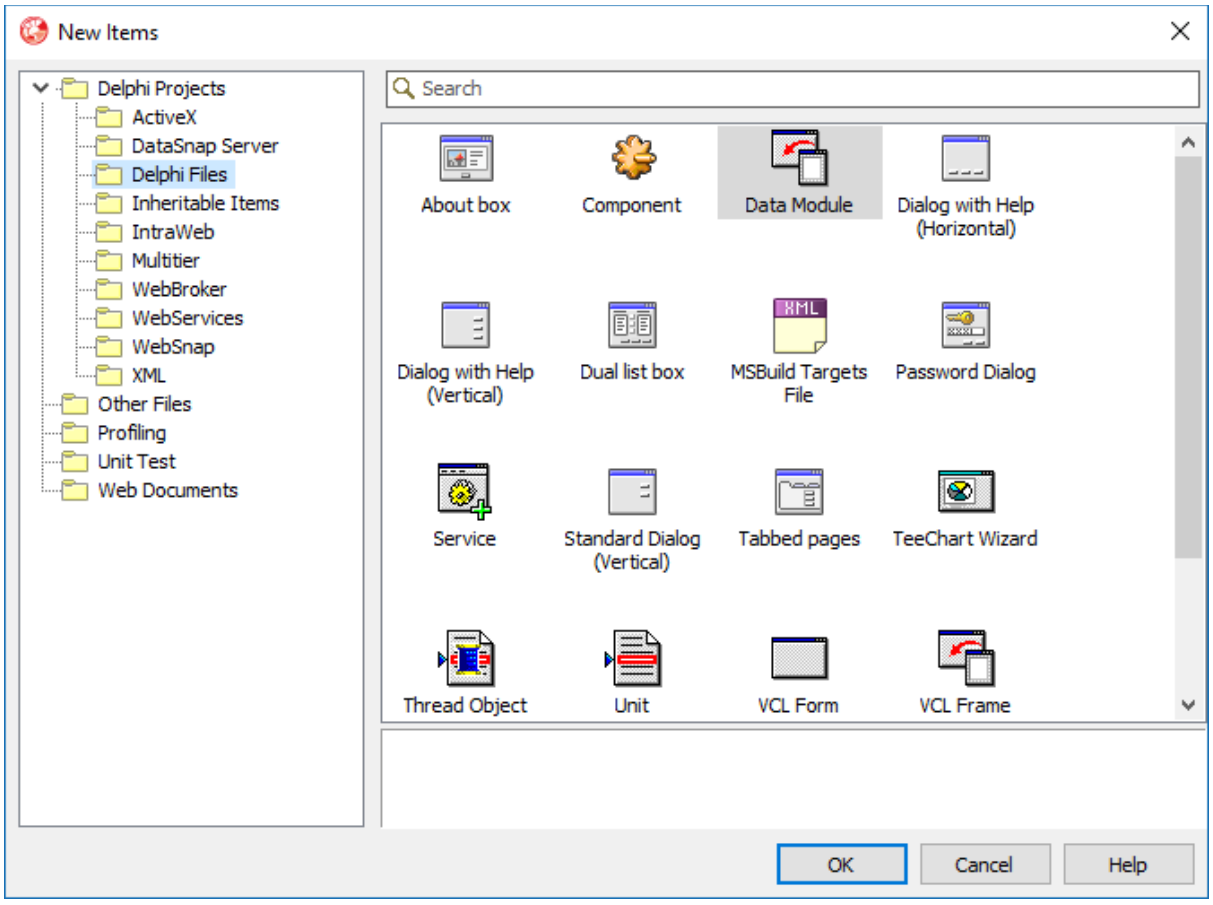


Ilustración 23: Ventana de formularios del proyecto

Un módulo de datos es un formulario no visual (contenedor) que se usa para la gestión centralizada de componentes no visuales de una aplicación. Generalmente incluye componentes para el acceso a datos (TADOConnection, TADOQuery, DataSource, etc), aunque se puede agregar cualquier componente, como diálogos y componentes Indy.

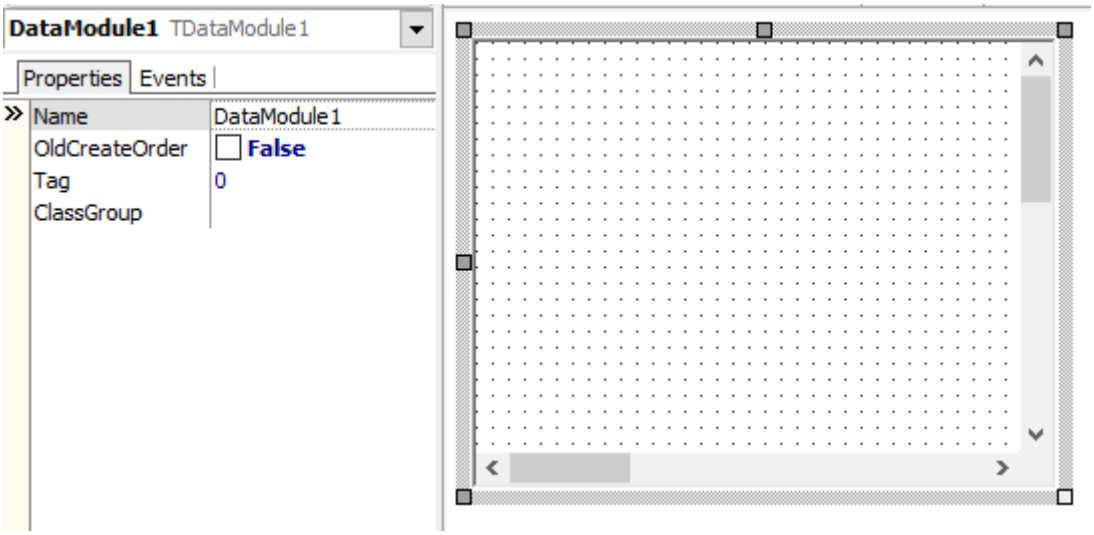


Ilustración 24: Módulo de datos

Un módulo de datos nos permite:

- Mantener todos los componentes de acceso a datos en un solo contenedor visual en modo de diseño, en lugar de duplicarlos para cada form de la aplicación
- Diseñar tablas y consultas una sola vez y usarlas en varios forms, en lugar de crearlas por separado para cada form.
- Guardarlo en el Object Repository para poder reutilizarlo.

La característica más interesante para nuestro proyecto es la centralización de recursos. El método de trabajo que usaremos será el de crear los componentes no visuales en un módulo de datos, y usarlo en todas las units que lo requieran. Así lo tendremos disponible para toda la aplicación sin tener que repetirlos.

Para nuestro proyecto necesitaremos añadir en el módulo de datos cuatro tipos de componentes no visuales: *ADOConnection, ADOQuery, DataSource* y *DataSet*.

ADO es la tecnología de referencia estándar para las estructuras de datos relacionales de Microsoft. El primer componente de esta tecnología que vamos a usar es *ADOConnection*.

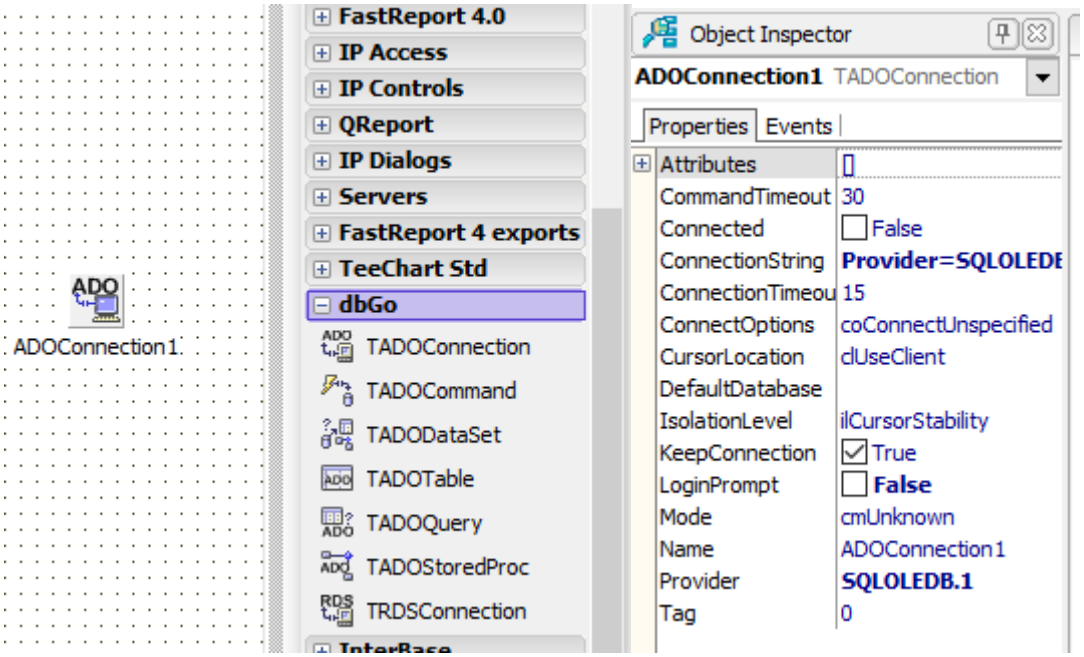


Ilustración 25: Componente ADOConnection

El componente *ADOConnection* permite especificar la ubicación de la base de datos y trabajar con las transacciones. Consideramos las principales propiedades del componente *ADOConnection* reflejadas en la ventana Inspector de objetos:

CommandTimeout	Integer	Define el tiempo en segundos para la ejecución de un comando. A la expiración de este tiempo el comando se considerará incumplido. Por defecto el tiempo para la ejecución de un comando es igual a 30 segundos.
Connected	Boolean	Se utiliza para establecer una conexión con la base de datos. Por defecto está en *false*, que significa que la conexión con la base de datos no está establecida.
ConnectionString	WideString	Contiene la información de la ubicación de la base de datos (del servidor).
ConnectionTimeout	Integer	Define el tiempo en segundos necesario para establecer la conexión con la base de datos. A la expiración de este tiempo salta el

		mensaje de imposibilidad de conexión. Por defecto este tiempo es igual a 15 segundos.
ConnectOptions	TConnectOption	Define el tipo de la conexión (síncrono o asíncrono). Por defecto es *coConnectUnspecified*, la conexión síncrona.
DefaultDatabase	WideString	Asigna una base de datos a la que se conecta cuando el servidor indicado en ConnectionString no está disponible.
KeepConnection	Boolean	Una vez que se cierra el último componente vinculado de ADO, puede mantener la conexión abierta o se cierra.

Para conectar nuestro componente *ADOConnection* con la base de datos vamos a la propiedad *ConnectionString*.

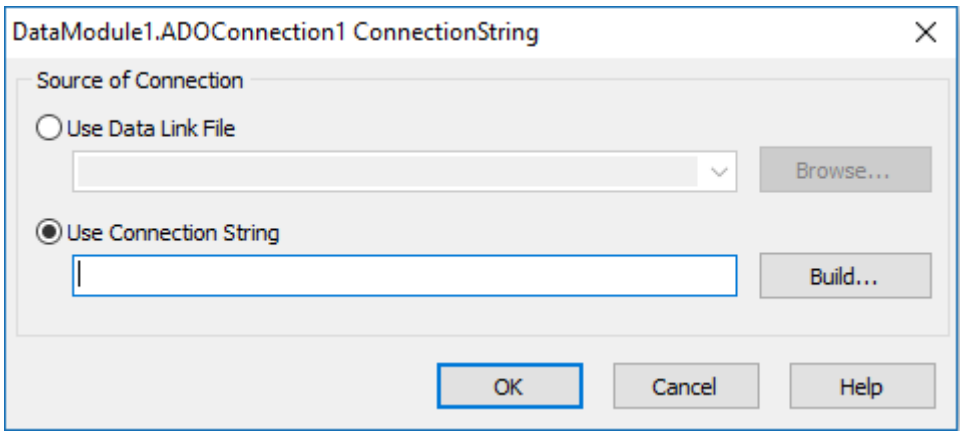


Ilustración 26: ConnectionString

Nuestro componente todavía no tiene ninguna información sobre la ubicación del servidor. Por tanto, tenemos que vincular todos los datos necesarios. Vamos a *Build…*

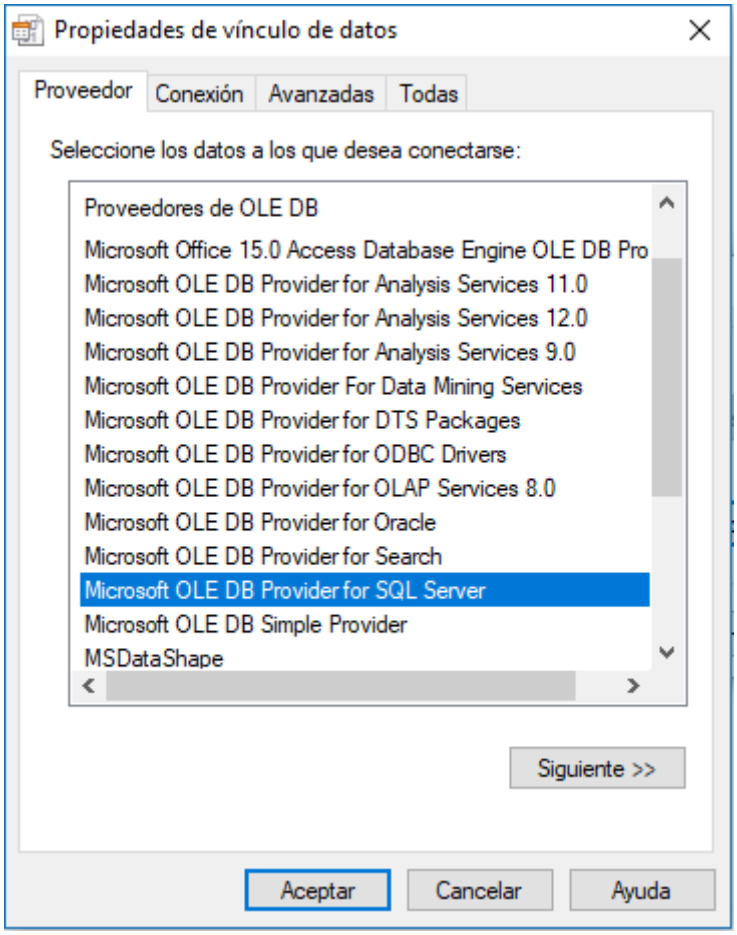


Ilustración 27: Propiedades de vínculo de datos

En la primera ventana tenemos que seleccionar al proveedor de la base de datos, en nuestro caso es *Microsoft OLE DB Provider for SQL Server*. Después vamos a la siguiente

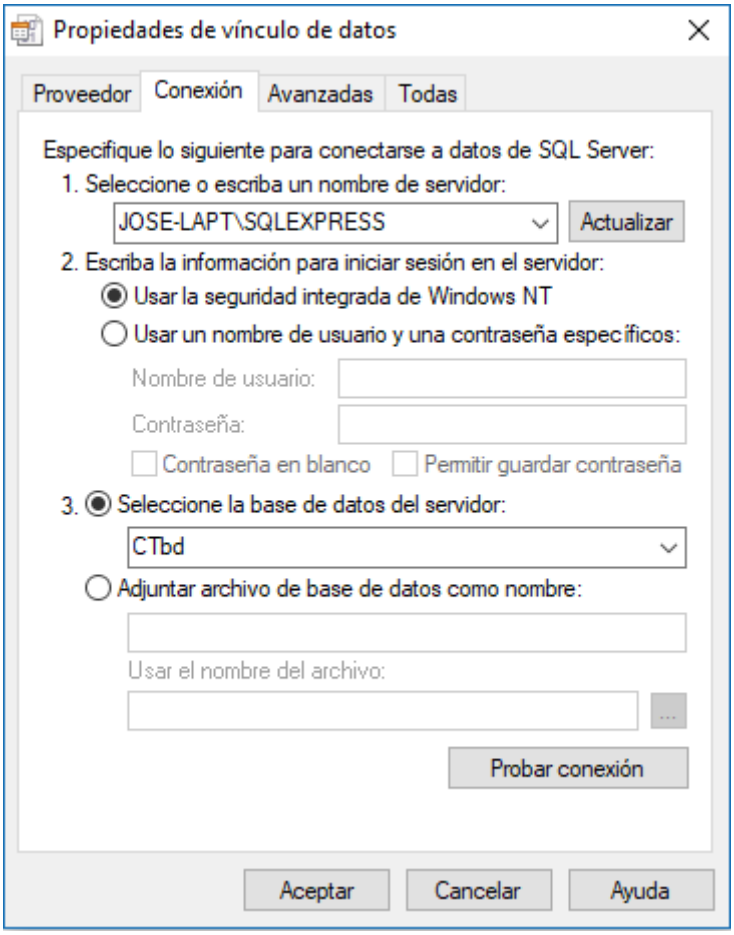


Ilustración 28: Propiedades de vínculo de datos

En la ventana de Conexión seleccionamos el servidor. Teniendo en cuenta que nuestra base de datos está ubicada en el mismo ordenador, el servidor es local *SQLEXPRESS*. Se iniciará la sesión el sistema de la seguridad integrada de Windows, y la base de datos creada será CTbd. Apretamos el botón *Probar conexión* para comprobar que hemos hecho todo correctamente. Y si todo está bien en la ventana ConnectionString aparecerá toda la información necesaria para la conexión correcta en formato *string*.

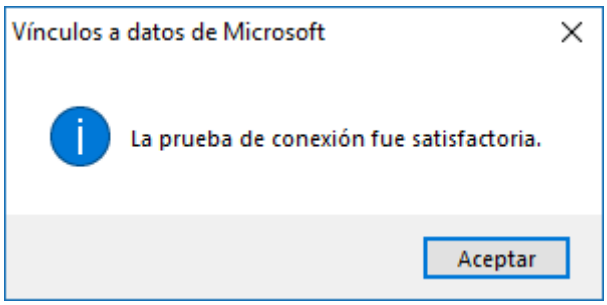


Ilustración 29: Mensaje de prueba de conexión

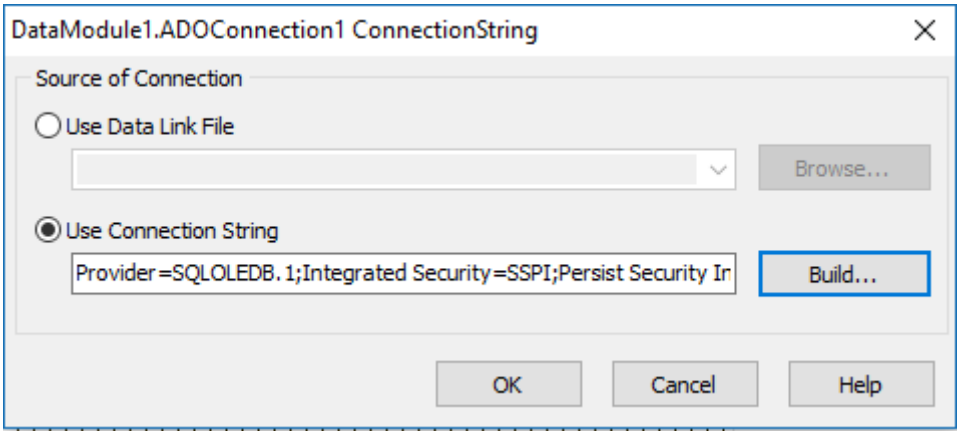


Ilustración 30: ConnectionString

Siguiente componente es *ADOQuery*. Este componente es una consulta de base de datos. Puede tratarse de una consulta que devuelve datos de la base de datos, por ejemplo, SELECT, o una consulta que no forma el conjunto de datos de resultados, por ejemplo, INSERT.

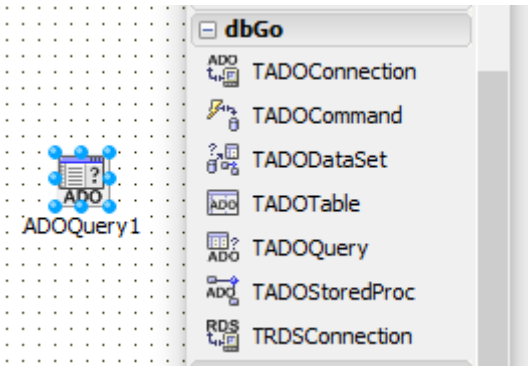


Ilustración 31: Componente ADOQuery

Los parámetros principales de este componente son:

- *Connection*, donde seleccionamos nuestro componente de conexión con la base de datos *ADOConnection1*
- *SQL*, la consulta en formato TString

Todas las consultas SQL las implementaremos en nuestro código de programa para cada caso necesario, por tanto, no tenemos que rellenar este campo dentro del inspector de objetos.

El componente no visual *DataSource* en Delphi es una fuente de datos que proporciona un enlace entre el conjunto de datos y los componentes de visualización y edición de datos.

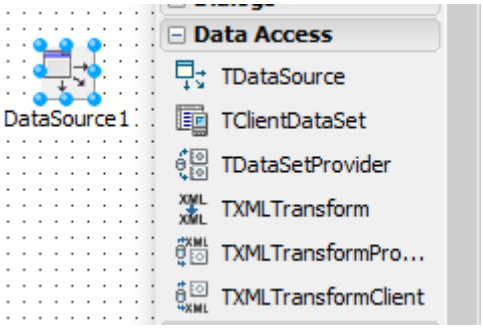


Ilustración 32: Componente DataSource

Todos los conjuntos de datos deben estar asociados con el componente fuente de datos si se desean editar dichos datos. La propiedad principal de la fuente de datos es *DataSet*, donde indicaremos nuestro *ADOQuery*.

En nuestra aplicación necesitaremos trabajar con varias consultas a diferentes tablas de nuestra base de datos en paralelo. Para facilitar este trabajo, crearemos tantos componentes *ADOQuery* y *DataSource* como tablas queramos.

El último componente no visual en nuestro módulo es *ClientDataSet*. Sirve para proporcionar el acceso a los datos. Este componente nos va a servir para combinar los datos de diferentes tablas de nuestra base de datos y representar el conjunto de estos datos en una sola tabla dentro del programa. Toda la configuración de este componente la realizaremos con el código de programa para las consultas necesarias.

Diseño de formulario para volcado de datos

En este apartado crearemos una pantalla (formulario visible) que nos permite comunicar con el usuario con el objetivo de guardar las tablas obtenidas de Cartrack en nuestra base de datos. Veremos cómo usar los componentes visibles, hacer la referencia a los objetos de otros formularios y trabajar con tablas Excel mediante el lenguaje de programación de Delphi.

Para la realización de nuestra tarea necesitaremos tres tablas Excel principales:

- Informe de Cartrack Nº **119** ***Informe general de viajes con ralenti*** de la categoría *Actividad de flota*. Este informe contiene la información de inicio y fin de viaje, duración, distancia y alertas generadas por vehículo.

- Informe de Cartrack Nº **306 *Detalle Horario de Visitas por Vehículo > 5 minutos*** de la categoría *Geofences*. Este informe detalla horario de entrada y salida de geofences por vehículo.
- Informe del programa corporativo **Gestora** que indica la matrícula real para cada chofer por día.

Aunque para resolver nuestro problema hace falta solamente estas tres tablas en el programa (Ilustración 36) y la base de datos (Ilustración 21) se puede observar los campos para dos tablas más: *Gasoil (LT)* y *203* (Odometer). No vamos a prestar atención a estos campos ya que están pensados para otros proyectos.

Empezamos a crear un formulario visible que será nuestra ventana para seleccionar las tablas Excel. Para ello vamos a *File → New → VCL Form - Delphi*

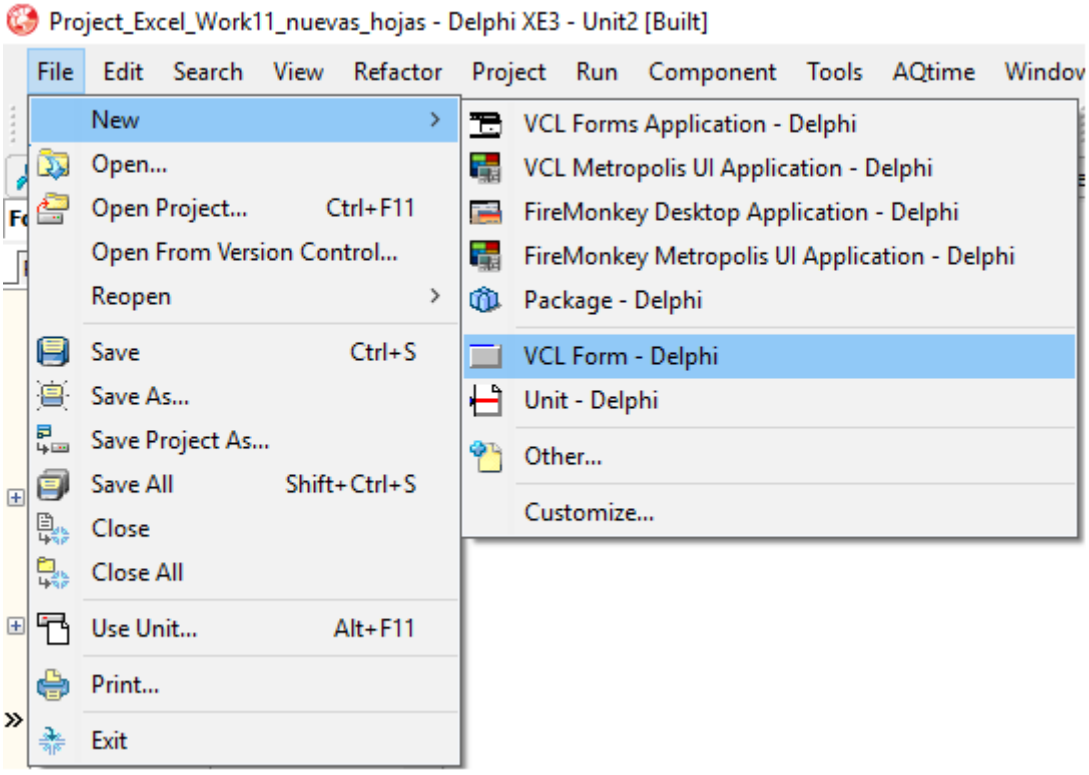


Ilustración 33: Menú para crear la forma VCL

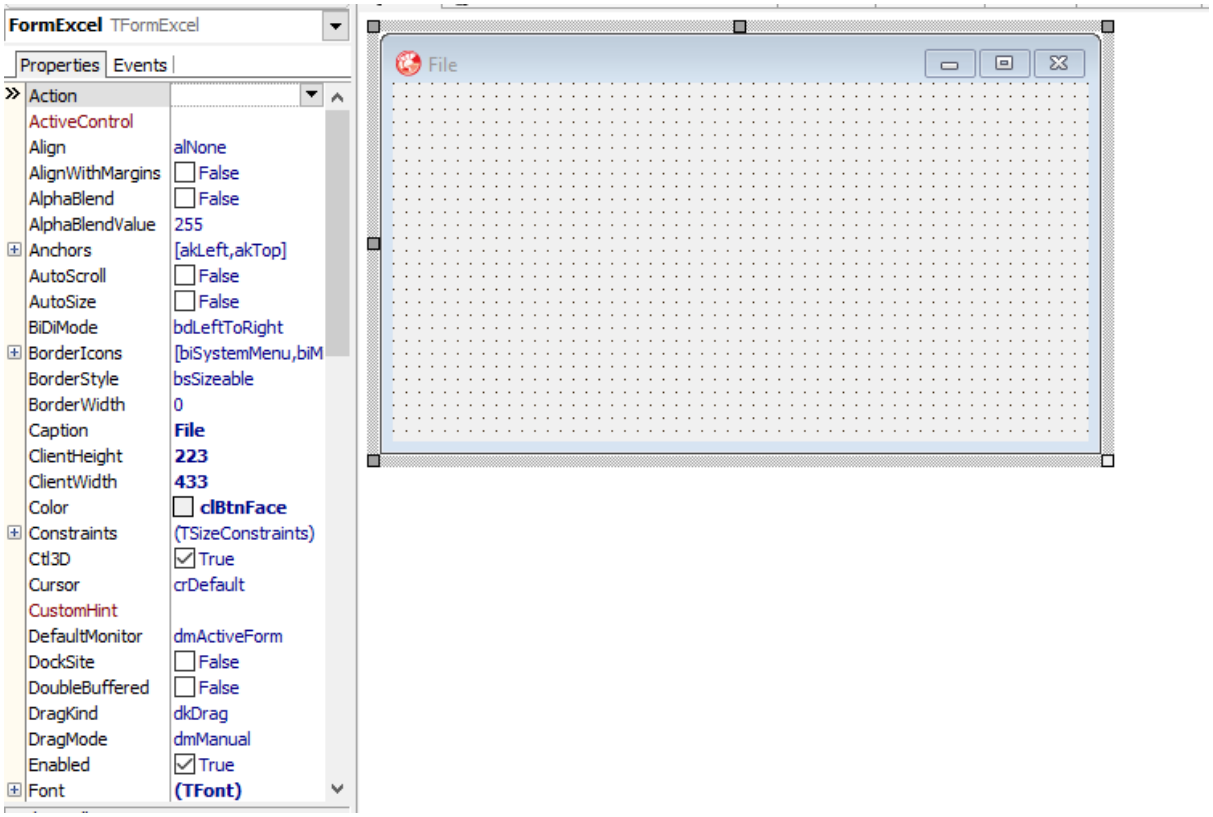


Ilustración 34: Interfaz de Formulario 1

Dentro del Inspector de Objetos tenemos que cambiar dos propiedades:

- **Caption**. Es el nombre de la ventana que va a aparecer en la esquina superior izquierda, ponemos por ejemplo “File”.
- **Name**. Es un nombre real del objeto o forma que debemos usar en nuestro código. Para tener claro que es un formulario para Excel lo llamaremos “FormExcel”.

Otros parámetros (como alineación, fuentes, modos, colores, etc) se puede poner al gusto.

Ahora añadimos unos elementos visibles de tipo *botones*, *etiquetas* y *edit*; y elementos no visibles *OpenDialog,* que nos permiten abrir en nuestro programa un diálogo estándar de Windows para poder seleccionar los archivos.

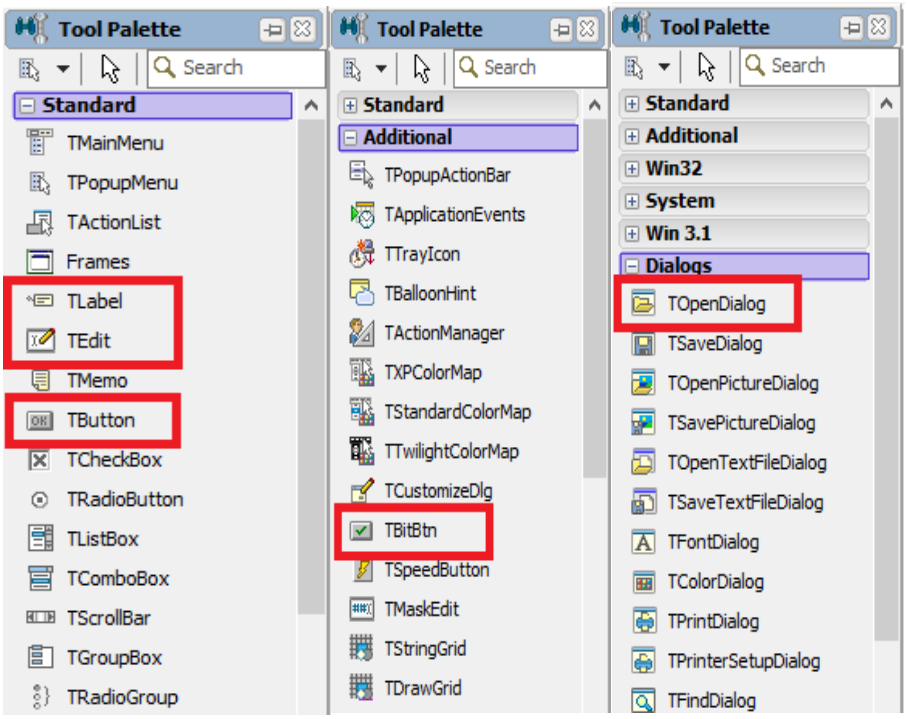


Ilustración 35: Paleta de componentes

En algunos casos en lugar de utilizar componente *TButton* cogemos *TBitBtn*. El componente Delphi *BitBtn* es un botón de icono, una de las versiones del botón *Button* estándar. A diferencia del último, el botón *BitBtn* puede mostrar en su superficie no solo una inscripción, sino también una imagen configurada por la propiedad *Glyph*. Delphi incluye una gran cantidad de imágenes prediseñadas que se pueden colocar en los botones *BitBtn*. En el caso estándar, están ubicados en *C:\Program Files\Common Files\Borland Shared\Images\Buttons.* Colocaremos todos los elementos necesarios de manera adecuada y visualmente agradable.

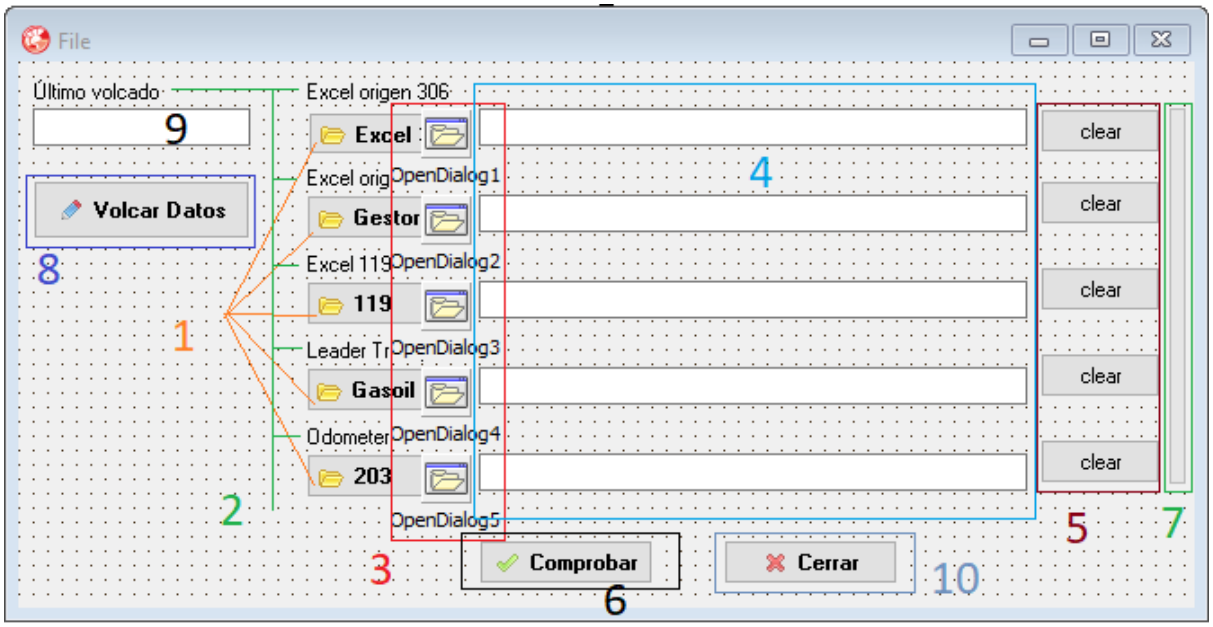
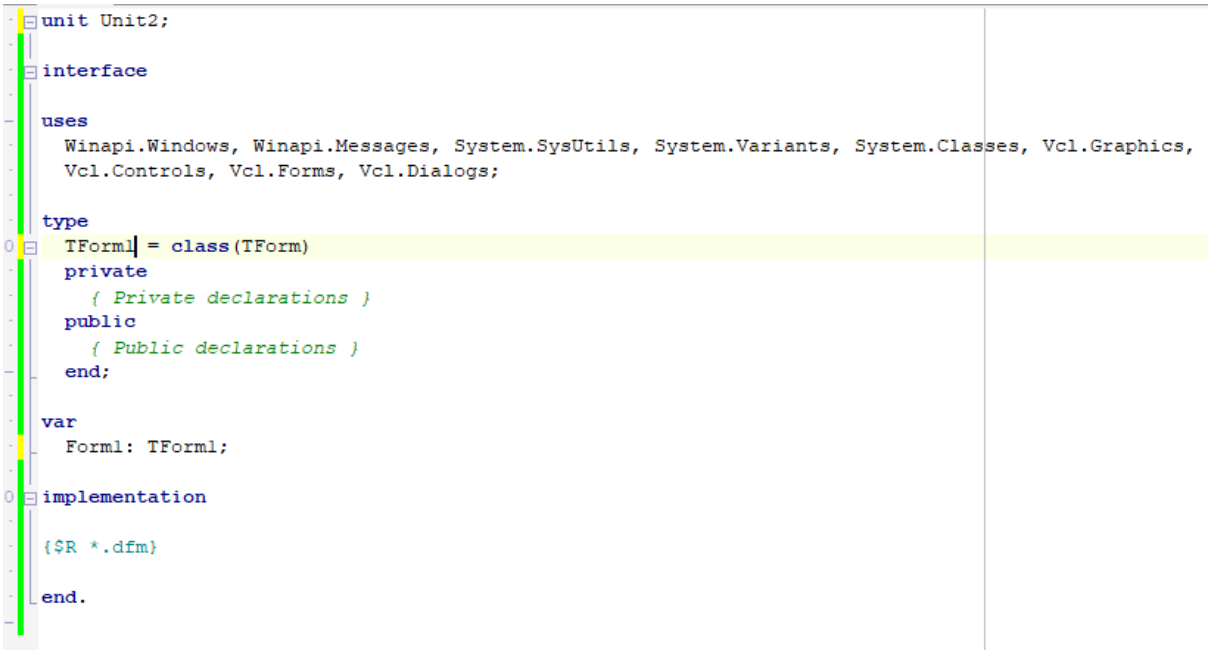


Ilustración 36: Diseño de formulario File

1. Colocamos los botones *TBitBtn*. Estos botones nos servirán para abrir las tablas Excel. Cada objeto *TBitBtn* llamará al objeto *OpenDialog* correspondiente. Añadimos las imágenes con la propiedad *Glyph*.
2. Etiquetamos los botones con *TLabel*.
3. Como componentes *OpenDialog* no son visibles los ponemos al lado del botón *TBitBtn* correspondiente. En la propiedad *Filter* aplicamos un filtro para los formatos de Excel (*.xls; *.xlsx).
4. Utilizamos componente *TEdit* como el campo que nos muestra la ubicación del archivo en formato de texto. Vaciamos la propiedad *Text* para tener las líneas en blanco antes de seleccionar archivos.
5. Añadimos los botones *TButton* para limpiar el contenido de *TEdit* más rápido.
6. Un botón *TBitBtn* para comprobar el formato y contenido de los Excel que intentamos volcar.
7. Un componente *ProgressBar* que nos permite ver el estado del proceso de volcado de datos.
8. Un botón *TBitBtn* para hacer las manipulaciones necesarias sobre las tablas Excel y guardar la información en la base de datos.
9. Un campo de tipo *TEdit* para ver la fecha del último volcado de datos. La propiedad *ReadOnly* ponemos en *True*, el contenido de este *TEdit* no se debe poder cambiar a mano.

Programación de formulario para volcado de datos

Pasamos ahora a la ventana de código *Unit* que nos permite la programación de formulario correspondiente. Para ello consideremos en qué partes está compuesto cualquier código en Delphi.



```
unit Unit2;

interface

uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,
  Vcl.Controls, Vcl.Forms, Vcl.Dialogs;

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

end.
```

Ilustración 37: Ventana de texto de código

El módulo comienza con un encabezado: instrucción *Unit*, que especifica el nombre del módulo. Cuando se guarda el módulo, este nombre se cambiará automáticamente por el nombre especificado por el programador.

En la sección *Interface*, el programador debe colocar las declaraciones de los procedimientos y funciones del módulo que pueden llamarse desde otros módulos utilizando el módulo dado.

La palabra clave *Uses* especifica una lista de módulos, programas o bibliotecas, que son utilizados por este módulo.

En la parte *Type* está definido el formulario, los elementos añadidos con sus eventos seleccionados. Las definiciones en este apartado se rellenan automáticamente cuando añadimos un elemento al formulario o asociamos algún evento para elemento añadido.

Con la palabra clave *Var* declaramos las variables.

La palabra clave *Implementation* inicia la sección activa de código del módulo. En este apartado tenemos que describir completamente todas las funciones y procedimientos.

Empezamos nuestro código añadiendo las librerías necesarias dentro del campo de la palabra clave *Uses*. Por defecto el módulo ya tiene las librerías que nos permiten trabajar con mayoría de las funciones, pero en nuestro caso tenemos que añadir también tres librerías

más (*ComObj, ComCtrls y ActiveX)* para poder implementar algunas funciones concretas del Excel.

```
uses
  Windows, Messages, SysUtils, Variants, ActiveX, ComObj, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, UnitVolcado, Buttons, ComCtrls, variables, Vcl.Grids,
  Vcl.DBGrids, Vcl.ExtCtrls;
```

Ilustración 38: Librerías y formas declaradas

En el campo de la palabra clave *Type* automáticamente aparecen definidos todos los elementos de nuestro formulario.

```
type
  TFormExcel = class(TForm)
    Label3: TLabel;
    Label6: TLabel;
    OpenDialog1: TOpenDialog;
    OpenDialog2: TOpenDialog;
    Edit1: TEdit;
    Edit2: TEdit;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    BitBtn6: TBitBtn;
    BitBtn7: TBitBtn;
    Label1: TLabel;
    OpenDialog3: TOpenDialog;
    Edit3: TEdit;
    ProgressBar1: TProgressBar;
    Edit4: TEdit;
    OpenDialog4: TOpenDialog;
    Label2: TLabel;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Edit5: TEdit;
    Button5: TButton;
    Label4: TLabel;
    OpenDialog5: TOpenDialog;
    BtnInforme: TBitBtn;
    LabeledEdit1: TLabeledEdit;
    Label5: TLabel;
```

Ilustración 39: Definición de los elementos de formulario

Para definir los procedimientos para cada elemento de nuestro formulario tenemos dos opciones:

1. En la parte del Inspector de objetos seleccionamos el campo de evento necesario. Por ejemplo, si queremos añadir el evento de un simple click sobre botón seleccionamos el campo *OnClick*

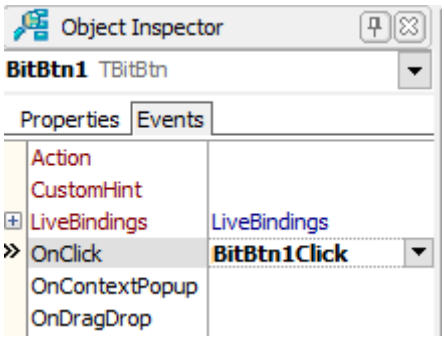


Ilustración 40: Evento OnClick

2. Hagamos la acción directamente sobre el elemento en nuestro formulario. Por ejemplo, el simple clic sobre el botón

De esa manera definimos los eventos de los elementos creados.

```
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure Edit2Change(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure BitBtn7Click(Sender: TObject);
    procedure Edit3Change(Sender: TObject);
    procedure Edit4Change(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Edit5Change(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure BtnInformeClick(Sender: TObject);
    procedure LabeledEdit1Click(Sender: TObject);
```

Ilustración 41: Definición de los procedimientos

En cuanto tengamos todos los elementos definidos pasaremos a programarlos. En primer lugar, programamos los botones 1 de la ilustración 36 que nos permiten seleccionar archivos

Excel. Haciendo doble click sobre el botón o seleccionando el campo correspondiente en los eventos nos aparece una plantilla del procedimiento elegido. Esta plantilla contiene el nombre del objeto con el evento a realizar, por ejemplo, *BitBtn**Click***, y las palabras claves *begin* y *end* entre las cuales programaremos todas las acciones que queremos que realice nuestro programa al apretar el botón.

```
procedure TFormExcel.BitBtn1Click(Sender: TObject);
begin
   if OpenDialog1.Execute then
  begin
    od1 :=1;
    ruta1 := OpenDialog1.FileName;        //guardamos la ruta
    Edit1.Text := ruta1;                  //la ponemos en edit para ver
  end;
end;

procedure TFormExcel.BitBtn2Click(Sender: TObject);
begin
  if OpenDialog2.Execute then
  begin
    od2 := 1;
    ruta2:=OpenDialog2.FileName;        //guardamos la ruta
    Edit2.Text:=ruta2;
  end
end;
```

Ilustración 42: Código para los botones uno y dos

Con una condición *if / then* conectamos cada botón con su *OpenDialog* correspondiente. También tenemos que añadir dos tipos de variables: llamamos **od[n]** de tipo integer las variables de control y **ruta[n]** de tipo string para guardar la ubicación del fichero, donde [n] representa el número de botón. Para poder llamar a estas variables desde cualquier otro formulario creamos nuevo *unit* sin formulario y definimos dichas variables dentro.

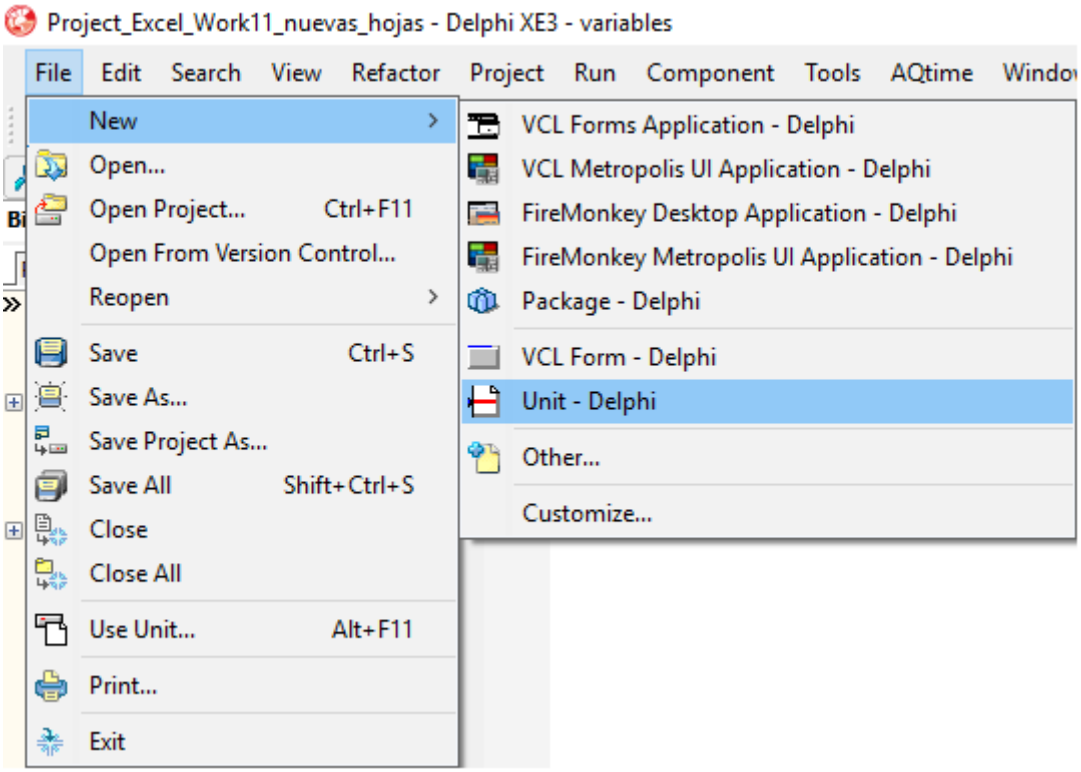


Ilustración 43: Menú File

```
unit variables;

interface
var
 ruta1, ruta2, ruta3, ruta4, ruta5 : string;
 od1,od2,od3,od4,od5 : integer;
implementation

end.
```

Ilustración 44: Definición de las variables

La variable *od[n]* obtiene el valor uno cuando se ejecuta *OpenDialog[n]* y la variable *ruta[n]* copia la dirección del archivo de la propiedad *FileName* del dicho *OpenDialog*.

La línea *Edit[n].Text := ruta[n]* introduce la dirección copiada en el campo de texto de objeto *Edit*, (4) de la ilustración 36.

Este procedimiento se repite para todos los botones (1) de la ilustración 36.

Como ya sabemos los informes que vamos a necesitar y las tablas que nos hacen falta para realización de nuestro proyecto podemos crear las tablas correspondientes en nuestra base de datos.

Nombre de columna	Tipo de datos	Permitir val...
Codigo	int	☐
Matricula	varchar(10)	☐
Activo	varchar(5)	☐
Marca	varchar(50)	☑
Modelo	varchar(50)	☑
Año	date	☑
CV	varchar(10)	☑
Ejes	varchar(30)	☑
Cuenta_contable	varchar(50)	☑
		☐

Ilustración 45: Tabla dbo.Matriculas

Nombre de columna	Tipo de datos	Permitir val...
Numero	varchar(20)	☑
Marca	varchar(50)	☑
Modelo	varchar(50)	☑
Año	date	☑
Ejes	varchar(30)	☑
Cuenta_contable	varchar(50)	☑
		☐

Ilustración 46: Tabla dbo.Plataforma

Nombre de columna	Tipo de datos	Permitir val...
Nombre	varchar(20)	☑
Apellido1	varchar(20)	☑
Apellido2	varchar(20)	☑
Domicilio	varchar(50)	☑
Tel_casa	varchar(20)	☑
Tel_empressa	varchar(20)	☑
email	varchar(50)	☑
cuenta_contable	varchar(50)	☑
DNI	varchar(50)	☑
Carnet_conducir	varchar(50)	☑
fecha_caduc	datetime	☑
carnet_IMO	varchar(50)	☑

Ilustración 47: Tabla dbo.Choferes

Nombre de columna	Tipo de datos	Permitir val...
Codigo	int	☐
Matricula	int	☐
Fecha	date	☑
Hora_In	time(0)	☑
Hora_Fin	time(0)	☑
Geofence	varchar(50)	☑
Kms	varchar(20)	☑
		☐

Ilustración 48: Tabla dbo.Tabla119

Nombre de columna	Tipo de datos	Permitir val...
Codigo	int	☐
Matricula	int	☐
Fecha	date	☑
Kms	varchar(30)	☑
Distancia	varchar(20)	☑
		☐

Ilustración 49: Tabla dbo.Tabla203

Nombre de columna	Tipo de datos	Permitir val...
Codigo	int	☐
Matricula	int	☐
Fecha	date	☑
Hora	time(0)	☑
Geofence	varchar(50)	☑
Conduccion	time(7)	☑
Descanso	time(7)	☑
Presencia	time(7)	☑
		☐

Ilustración 50: Tabla dbo.Tabla306

Nombre de columna	Tipo de datos	Permitir val...
Codigo	int	☐
Matricula	int	☐
Transportista	varchar(50)	☑
Lugar	varchar(50)	☑
Recoger	varchar(50)	☑
Devolver	varchar(50)	☑
Cliente	varchar(50)	☑
Fecha	varchar(50)	☑
Hora	time(0)	☑
Poblacion	varchar(50)	☑
Importe	varchar(20)	☑
Kms	varchar(20)	☑
Plataforma	varchar(20)	☑
fecha2	datetime	☑

Ilustración 51: Tabla dbo.Gestora

Nombre de columna	Tipo de datos	Permitir val...
Codigo	int	☐
Matricula	int	☐
Fecha	date	☑
[E.S.]	varchar(30)	☑
Ref	varchar(30)	☑
Producto	int	☑
Tipo	varchar(30)	☑
Cantidad	varchar(20)	☑
Precio	varchar(20)	☑
Total	varchar(20)	☑
Dt	varchar(20)	☑
Resta	varchar(20)	☑
dato	varchar(20)	☑

Ilustración 52: Tabla dbo.LT

Nombre de columna	Tipo de datos	Permitir val...
ini	datetime	☑
fin	datetime	☑
chofer	varchar(50)	☑
Matricula	int	☑
Fecha	date	☑
Hora	time(7)	☑
Geofence	varchar(50)	☑
		☐

Ilustración 53: Tabla dbo.busc

Nombre de columna	Tipo de datos	Permitir val...
Conduccion	time(7)	☑
Descanso	time(7)	☑
Presencia	time(7)	☑

Ilustración 54: Tabla dbo.Tiempos

Las dos últimas tablas utilizaremos para representar los datos finales y construir la hoja de dietas.

Para cada tabla de la base de datos crearemos correspondientes *ADOQuery* y *DataSource* dentro de nuestro contenedor *DataModule1* conectando estos objetos como antes a nuestro *ADOConnection*. También, se añaden dos *ClientDataSet* para visualizar histórico de los datos y tabla de dietas.

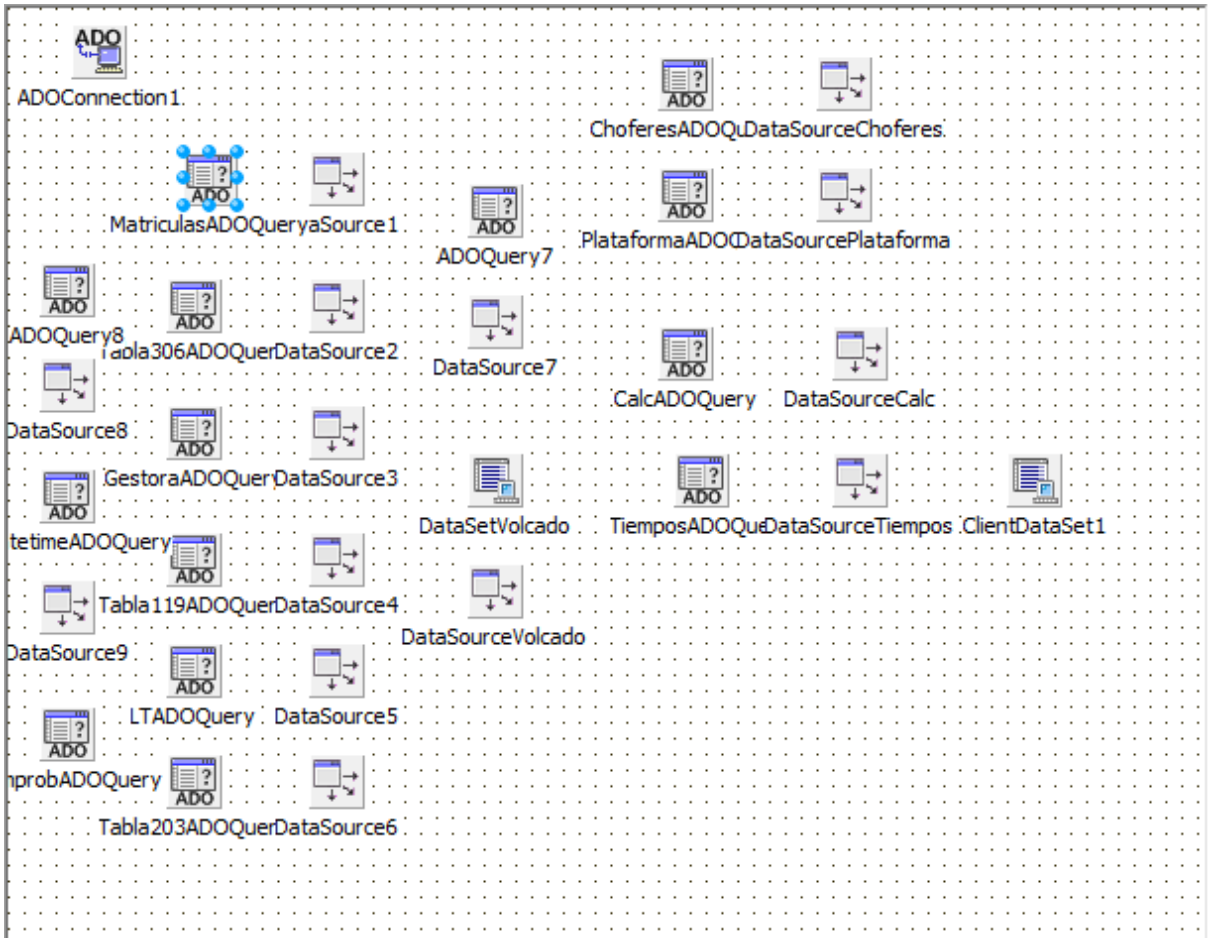


Ilustración 55: DataModule1

Volviendo a nuestro entorno de desarrollo de software, el siguiente paso es comprobar si hemos seleccionado los ficheros Excel correctamente. El fichero principal para realización de nuestra tarea es informe 119, por tanto, tenemos que comprobar que lo hemos seleccionado, que está correcto y que contiene todas Geofences.

Empezamos creando *procedure* para nuestro botón *Comprobar* haciendo doble click o seleccionando el evento correspondiente en inspector de objetos. Entre las palabras claves *begin* y *end* describimos las instrucciones. Antes de empezar a comparar cosas tenemos que hacer algunos pasos previos. En primer lugar, al apretar el botón cambiamos la forma de cursor con la instrucción *Screen.Cursor* para visualmente entender que el programa está en ejecución. Quitamos disponibilidad de los botones. Para poder trabajar con los ficheros Excel tenemos que abrir la aplicación creando un objeto *Excel.Application*, y como no la queremos ver en la pantalla dejamos *EnableEvents* en *false*. Al final añado una variable "***a***" que cambiará su valor de 0 a 1 si ocurre algún error en los ficheros seleccionados.

```
begin
    Screen.Cursor := crHourGlass; //para saber que esta funcionando
    BitBtn1.Enabled := false;
    BitBtn2.Enabled := false;
    BitBtn3.Enabled := false;
    BitBtn4.Enabled := false;
    BitBtn5.Enabled := false;
    BitBtn6.Enabled := false;
    BitBtn7.Enabled := false;
    BtnInforme.Enabled := false;
    Excel := CreateOleObject('Excel.Application');      //crear objeto
    Excel.Application.EnableEvents := false;
    Excel.Application.ReferenceStyle := 0;
    a:=0;
```

Ilustración 56: Código para cambio de cursor, botones y objeto Excel

Ahora comprobamos que hemos seleccionado al menos un fichero comparando la variable *od[n]* con un uno. Y en el caso afirmativo tenemos que mirar que no hemos introducido dos ficheros iguales comparando las variables *ruta[n]* entre las mismas.

```
if ((variables.od1=1) or (variables.od2=1) or (variables.od3=1) or
    (variables.od4=1) or (variables.od5=1)) then //si hemos elegido
begin                                            //por lo menos 1 excel
  if (((rutal='') or ((rutal<>ruta2)and(rutal<>ruta3)and(rutal<>ruta4)and
       (rutal<>ruta5))) and ((ruta2='') or ((ruta2<>rutal)and
       (ruta2<>ruta3)and(ruta2<>ruta4)and(ruta2<>ruta5))) and
       ((ruta3='') or ((ruta3<>ruta2)and(ruta3<>rutal)and
       (ruta3<>ruta4)and(ruta3<>ruta5))) and
       ((ruta4='') or ((ruta4<>ruta2)and(ruta4<>ruta3)and
       (ruta4<>rutal)and(ruta4<>ruta5))) and
       ((ruta5='') or ((ruta5<>ruta2)and(ruta5<>ruta3)and
       (ruta5<>ruta4)and(ruta5<>rutal)))) then
  begin
```

Ilustración 57: Código para comparar variables y rutas

Si alguna de las condiciones no se cumple sacaremos un mensaje de aviso con la instrucción *ShowMessage*. Luego devolvemos la forma del cursor original, cerramos objeto Excel con la instrucción *Quit* y desbloqueamos todos los botones.

```
    end else
    begin
      ShowMessage('Los ficheros no pueden ser iguales!');
      Screen.Cursor := crDefault;                //cursor de antes
      Excel.Quit;         //cerramos los excels
      Excel := Unassigned;
    end
  end else
  ShowMessage('Elige archivos');
  BitBtn1.Enabled := true;
  BitBtn2.Enabled := true;
  BitBtn3.Enabled := true;
  BitBtn4.Enabled := true;
  BitBtn5.Enabled := true;
  BitBtn6.Enabled := true;
  BitBtn7.Enabled := true;
  BtnInforme.Enabled := true;
  Screen.Cursor := crDefault;                //cursor de antes
```

Ilustración 58: Mensajes de aviso

Ahora procedemos a revisión de cada fichero excel por separado. Con la variable *od[n]* y función *if* seleccionamos el fichero concreto. Creamos variables nuevas *BookOrigen[nombre]* y *cell[n]* para no escribir cada vez instrucciones largas para abrir libro o seleccionar una celda concreta. También, en cada fichero excel tenemos que descombinar las celdas con función *MergeCells* para todo el campo con datos *UsedRange* poniendo el valor *false*. Para cada excel comparamos la celda *A1* con el nombre completo del fichero, si esta condición se cumple el fichero seleccionado es correcto. Cuando se cumple la condición anterior para informe 306 en la última fila de primera columna colocaremos copia del nombre completa para tener una referencia de fin de línea en futuro. Y como siempre en el caso de que fichero sea incorrecto sacamos un mensaje de aviso y ponemos un uno a nuestra variable de error.

306 Detalle Horario de Visitas por Vehículo > 5 Minutos

Descripción:	**Detalle de horas de entrada y salida de geofences por vehículo**
Fecha inicio:	**2016-08-22 00:00:00**
Fecha fin:	**2016-08-26 23:59:59**
Fecha Informe:	**2016/09/12**
Marca y Modelo:	DAF FT XF 95480
Matrícula:	2652FJJ
Descripción Vehículo:	

Claves

- **> 5 min**
- **> 10 min**
- **> 30 min**
- **> 60 min**
- **>120 min**

Geofence	Conductor	Fecha Llegada	Fecha Salida	Duración	Distancia
area parking		2016/08/22 01:15:02	2016/08/22 06:22:20	05:07:18	0,18
area tymsa		2016/08/22 06:23:57	2016/08/22 06:34:48	00:10:51	0,45
gama decor		2016/08/22 07:33:03	2016/08/22 08:50:51	01:17:48	1,18
porcelanosa		2016/08/22 09:01:59	2016/08/22 10:19:08	01:17:09	1,03
AREA		2016/08/22 10:39:46	2016/08/22 11:03:14	00:23:28	0,31
tcv		2016/08/22 11:45:48	2016/08/22 12:04:31	00:18:43	5,91
area buzones 2		2016/08/22 12:08:46	2016/08/22 12:18:35	00:09:49	0,22
AREA sedavi		2016/08/22 12:24:44	2016/08/22 16:28:21	04:03:37	1,18

Ilustración 59: Excel del informe 306

```
begin
  if variables.od1=1 then    //si tenemos primer excel (306)
  begin
    BookOrigen306 := Excel.Workbooks.Open(OpenDialog1.FileName);    //abrimos el
    BookOrigen306.Sheets[1].UsedRange.MergeCells:=False;           //descombinamo
    cell1:=BookOrigen306.Sheets[1].Range['A1'];
    if (cell1 = '306 Detalle Horario de Visitas por Vehículo > 5 Minutos') then
    begin
      Row2 := BookOrigen306.Sheets[1].UsedRange.Row +
      BookOrigen306.Sheets[1].UsedRange.Rows.Count-1;
      BookOrigen306.Sheets[1].Cells[Row2,1]:=
      '306 Detalle Horario de Visitas por Vehículo > 5 Minutos';    //en la ulti
      BookOrigen306.Save;     //guardamos el libro
    end
    else
    if cell1 <> '306 Detalle Horario de Visitas por Vehículo > 5 Minutos' then
    begin
      ShowMessage('Excel 306 no esta correcto');   //si excel no esta correcto s
      a:=1;
    end
  end;
```

Ilustración 60: Código para comprobar informe 306

El mismo procedimiento aplicaremos para Excel de Gestora e informe 203

```
if variables.od2=1 then          //lo mismo para segundo excel (gestora)
begin
  BookOrigenGest := Excel.Workbooks.Open(OpenDialog2.FileName);
  SheetGest := BookOrigenGest.Sheets[1];
  cell2:=BookOrigenGest.Sheets[1].Range['A1'];
  BookOrigenGest.Save;
  if (cell2 <> ' Consultas ') then
  begin
    ShowMessage('Esxcel de gestora no esta correcto');
    a:=1;
  end
end;
```

Ilustración 61: Código para comprobar excel de Gestora

```
if variables.od5=1 then            // lo mismo para quinto excel (203)
begin
  Book203 := Excel.Workbooks.Open(OpenDialog5.FileName);
  cell5:=Book203.Sheets[1].Range['A1'];
  if (cell5 <> '203 Kilómetros Diarios por Vehículo') then
  begin
    ShowMessage('Excel 203 no esta correcto');
    a:=1;
  end
end;
```

Ilustración 62: Código para comprobar informe 203

La parte del informe 119 es más compleja ya que además de comprobar solamente el nombre del fichero tenemos que asegurarnos de que no se quede ninguna casilla de Geofences sin rellenar. Pero empezamos del mismo modo que antes, abrimos el libro y comparamos la celda *A1* con el nombre completo del fichero.

```
if variables.od3=1 then          //lo mismo para terser excel (119)
begin
  BookOrigen106119 := Excel.Workbooks.Open(OpenDialog3.FileName);
  BookOrigen106119.Sheets[1].UsedRange.MergeCells:=False;
  cell3:=BookOrigen106119.Sheets[1].Range['A1'];
  if (cell3 <> '119 Informe General de Viajes con Ralenti') then
  begin
    ShowMessage('Excel 119 no esta correcto');
    a:=1;
  end else
```

Ilustración 63: Código para comprobar el nombre del fichero

Si cumplimos la condición tenemos que crear una consulta SQL a través de objeto *ADOQuery* de la tabla Matrículas. Para que esta consulta funcione tenemos que cerrar la conexión primero con la función *Close*, limpiamos el campo SQL con *Clear*, seleccionamos todo (*) de la tabla Matrículas, volvemos a abrir la conexión y colocamos nuestro cursor en el primer dato de la tabla. Al final generamos un bucle para recorrer los datos de la tabla Matrículas utilizando un simple *for / do*.

```
begin
  DataModule1.MatriculasADOQuery.Close;
  DataModule1.MatriculasADOQuery.SQL.Clear;
  DataModule1.MatriculasADOQuery.SQL.ADD('select * from Matriculas');
  DataModule1.MatriculasADOQuery.Open;
  DataModule1.MatriculasADOQuery.First;
  for  pk := 0 to DataModule1.MatriculasADOQuery.RecordCount - 1 do
```

Ilustración 64: Conexión a la base de datos a través de ADO

Al recorrer la tabla Matrículas copiamos la matrícula de la posición actual en una variable *busc*. Gracias a la función *Cells.Find* podemos encontrar rápidamente todos los datos correspondientes a una variable de búsqueda. Aplicamos esta función para Excel 119 con la variable de búsqueda *busc*, comparando el valor de la celda, el orden de línea y buscando la siguiente. Así vamos a buscar en qué línea empiezan los datos de la matrícula correspondiente. Cuando se encuentra el primer dato no vacío para la variable no vacía, guardamos la posición de este dato y desplazamos cinco filas hacia abajo y doce columnas hacia derecha, el primer dato no vacío de la columna Geofences. En esta columna buscamos una celda vacía hasta llegar a la palabra "Viag: ", que es la última línea para la matrícula seleccionada. En el caso de encontrar al menos una celda vacía sacamos mensaje de aviso.

Por ejemplo, la primera matrícula en nuestra base de datos es 2652FJJ, se guarda este texto en la variable *busc*. En el Excel 119 encontramos una celda que contiene el mismo texto, se ubica en C10. Desplazamos cinco líneas abajo y doce columnas a la derecha y justamente encontramos el primer Geofence. Luego recorremos esta columna hasta encontrar la palabra "Viag: " y si por lo menos una celda tiene valor nulo debe saltar el mensaje de aviso.

Ilustración 65: Excel del informe 119

```
begin                                                                    //
  busc:=DataModule1.MatriculasADOQuery.FieldByName('Matricula').AsString;     //
  cds := BookOrigen106119.Sheets[1].Cells.find(What := busc,
    LookIn := xlValues, SearchOrder := xlByRows, SearchDirection := xlNext);
  if (not VarIsClear(cds)) and (busc <> '') then
  begin
    FirstAddress := cds.Address;
    str:= BookOrigen106119.Sheets[1].Range[FirstAddress].Offset[5,12].Address;
    Rangel:=BookOrigen106119.Sheets[1].Range[str].End[xlDown];  //
    col:='Viag:';                                                //  en estas ir
    if Rangel.Text <> col then                                   //
    begin
    showmessage('Falta Geofence! Matriculas: ' + busc);
    a:=1;
    end
  end;
  DataModule1.MatriculasADOQuery.Next;
end
```

Ilustración 66: Código para buscar geofences vacíos

Cuando hemos comprobado todas las tablas podemos proceder a transformar los datos y volcarlos a nuestra base de datos. De la misma manera que para el botón *Comparar* creamos *procedure* para el botón *Volcar Datos* (BtnInforme). En los pasos previos, en analogía con el caso anterior, cambiamos el tipo de cursor, deshabilitamos los botones, creamos una consulta SQL a través de *ADOQuery* para sacar todos los datos de la tabla Matrículas. También, añadimos *ProgressBar*, abrimos la aplicación Excel con la función *try* por si estaba abierta antes, y por último ponemos las variables *vv[n]* a cero. Estas variables implementamos para gestionar los datos en caso de repeticiones, contendrán la información sobre los datos que se están repitiendo y que hay que hacer con ellos (reemplazar, guardar o cancelar).

```
begin
  ProgressBar1.Position:=0;
  Label5.Caption:='00:00:00';
  Screen.Cursor := crHourGlass;  //cursor de espera
  BtnInforme.Enabled := false;   //desactivar los botones
  BitBtn3.Enabled := false;
  BitBtn4.Enabled := false;
  timerprogramal:=Now;  //tiempo de inicio del programa
  Application.ProcessMessages;
  DataModule1.MatriculasADOQuery.Close;
  DataModule1.MatriculasADOQuery.SQL.Clear;
  DataModule1.MatriculasADOQuery.SQL.ADD('select * from Matriculas');
  DataModule1.MatriculasADOQuery.Open;
  maxa:= DataModule1.MatriculasADOQuery.RecordCount - 1;    //numero de
  cs:=0;
  a:=0;
  vv1:=0;
  vv2:=0;
  vv3:=0;
  vv4:=0;
  vv5:=0;
  vv6:=0;
  try
  FormExcel := TFormExcel.create(Application);       //
  Excel := CreateOleObject('Excel.Application');     //crear objeto ex
  Excel.Application.EnableEvents := false;           //
  Excel.Application.ReferenceStyle := 0;             //
```

Ilustración 67: Pasos previos para el código de volcado

Hay que comprobar si existe por lo menos un fichero seleccionado para el caso si damos a volcar datos sin comprobar. Posteriormente, para los Excel de los informes de Cartrack creamos una segunda hoja para poder realizar las operaciones necesarias. Asignamos nombres a cada hoja para no escribir cada vez las instrucciones largas en el momento en que necesitamos acceso a los datos de las hojas. Los ficheros de Gestora y LT no necesitan segunda hoja.

```
begin
  if ((variables.od1=1) or (variables.od2=1) or
      (variables.od3=1) or (variables.od4=1) or
      (variables.od5=1)) then   //si tenemos por lo menos 1 exce
  begin
    if variables.od1=1 then
    begin
      BookOrigen306 := Excel.Workbooks.Open(variables.rutal);  /
      Sheet := BookOrigen306.Sheets[1];              //nombrar pr
      if BookOrigen306.Worksheets.Count > 1 then   //combrar si
      Sheet2 := BookOrigen306.Sheets[2]           //si la tiene,
      else
      Sheet2 := BookOrigen306.Sheets.Add(After:=Sheet); //si no,
      Sheet.UsedRange.MergeCells:=False;         //descambinar la
    end;

    if variables.od2=1 then
    begin
      BookOrigenGest := Excel.Workbooks.Open(variables.ruta2);
      SheetGest := BookOrigenGest.Sheets[1];   //asigno el nombr
    end;
```

Ilustración 68: Código para crear hoja nueva y nombrarla

Repetimos el código para otros ficheros Excel.

Igual que en el apartado del botón *Comprobar* realizaremos un bucle en la tabla con las matrículas que van a ser nuestro punto de referencia para guardar y sacar los datos. Esta tabla puede tener tanto las matrículas activas como no activas. Las diferenciamos por el código **s** (si está activa) o **n** (si no está activa) en el campo *Activo*. De esta manera consideramos solo las matrículas activas.

```
for a:=0 to maxa do     //contador para todas las matriculas de la tabla "Matricul
begin
  clrbx:=DataModule1.MatriculasADOQuery.FieldByName('Activo').AsString; //sacamos
  if clrbx = 's' then     //si la matricula es activa ("s" en la columna Activo)
  begin
      ProgressBar1.Max:=maxa+1;
      busc := DataModule1.MatriculasADOQuery.FieldByName('Matricula').AsString;
      cien:=1;
      ProgressBar1.Position:=a+1;    //contador para progressBar
```

Ilustración 69: Bucle para la tabla Matrículas

A partir de aquí bajo el mismo bucle vamos a tener cinco trozos muy parecidos correspondientes a cada Excel. Estos trozos van a tener la misma estructura que la del ejemplo de la tabla 119. En primer lugar, creamos una consulta SQL compleja. A través del *ADOQuery* de la tabla 119 sacamos los campos de la *Tabla119* junto con los de *Matrículas*

de la base de datos. Estos campos van a depender de la tabla *Matrículas* para que se muestre solo los datos de las matrículas activas.

```
if variables.od3=1 then  //excel 119
begin
      DataModule1.Tabla119ADOQuery.Close;
      DataModule1.Tabla119ADOQuery.SQL.Clear;
      DataModule1.Tabla119ADOQuery.SQL.ADD('select Tabla119.Codigo,Tabla119.Matricula'
      +' as MatriculaCod,Matriculas.Matricula,Fecha,Hora_In,Hora_Fin,Geofence,Kms'
      +' from Tabla119 INNER JOIN Matriculas ON Matriculas.Codigo=Tabla119.Matricula'
      +' where Matriculas.Activo=:s');
      DataModule1.Tabla119ADOQuery.Parameters.ParamByName('s').Value:='s';
      DataModule1.Tabla119ADOQuery.Open;                    //sacamos la tabla 119 de
      DataModule1.DataSource4.Enabled:=False;
```

Ilustración 70: Consulta SQL

Añadimos una condición para el caso de cancelación de la operación si se repiten los datos. Si decidimos cancelar el proceso no procedemos a realizar ninguna modificación de los datos. Para la primera "vuelta" del bucle la variable *vv1* = 0, no tenemos cancelaciones, por lo tanto, seguimos con el código. Buscamos la matrícula actual en la primera hoja con la instrucción *Cells.Find* y copiamos su posición. Preparamos la segunda hoja borrando todo el contenido por si tenía grabado algo anteriormente.

```
if vv1<>2 then
begin
  cds := Sheetkm.Cells.find(What := busc, LookIn := xlValues, SearchOrder := xlByRows,
    SearchDirection := xlNext);   //buscar la celda con la matricula correspondiente
  if (not VarIsClear(cds)) and (busc <> '') then     //si celda no esta vacia
  begin
    FirstAddress := cds.Address;       //guardamos la posicion
    FAdd119 := cds.Address;
    Sheet2km.Name := cds;

    Range1:=Sheet2km.UsedRange.Delete;  //limpio la hoja
```

Ilustración 71: Código para buscar la matrícula

Con la instrucción *Offset* vamos a la primera posición de los datos que queremos copiar, seleccionamos el campo donde queremos dejar los datos en la segunda hoja, y con la instrucción *Copy* se copian todos los datos desde la posición inicial seleccionada hasta *End[xlDown]* (la última posición hacia abajo hasta encontrar una celda vacía). El mismo procedimiento utilizamos para copiar las fechas y horas de entrada y salida, geofences y la distancia de recorrido. Al final, guardaremos el *Código* de la última posición de los datos de la *Tabla119* de la base de datos que nos permite comprobar si se repiten los datos.

```
str119:= Sheetkm.Range[FAdd119].Offset[5,6].Address; //a partir de la celda encontrada busco
Range2:=Sheet2km.Range['A1']; //Eligo la posicion donde se copian los datos (columna A)
Range1:=Sheetkm.Range[str119, Sheetkm.Range[str119].End[xlDown]].Copy(Range2);

Range1:=Sheet2km.Range['A1', Sheet2km.Range['A1'].End[xlDown]];
Range2:=Sheet2km.Range['A1:B1', Sheet2km.Range['A1:B1'].End[xlDown]];
Range1.TextToColumns(Range2, ConsecutiveDelimiter := True, Space:=True); //Para delimitacion
Range1.NumberFormat := 'dd/mm/aaaa';  //poner la columna en formato de fecha
Range1:=Sheet2km.Range['B1', Sheet2km.Range['B1'].End[xlDown]];

str119:= Sheetkm.Range[FAdd119].Offset[5,10].Address; //a partir de la celda encontrada busc
Range2:=Sheet2km.Range['C1']; //Eligo la posicion donde se copian los datos (columna A)
Range1:=Sheetkm.Range[str119, Sheetkm.Range[str119].End[xlDown]].Copy(Range2);

Range1:=Sheet2km.Range['C1', Sheet2km.Range['C1'].End[xlDown]];
Range2:=Sheet2km.Range['C1:D1', Sheet2km.Range['C1:D1'].End[xlDown]];
Range1.TextToColumns(Range2, ConsecutiveDelimiter := True, Space:=True); //Para delimitacion
Range1.NumberFormat := 'dd/mm/aaaa';  //poner la columna en formato de fecha
Range1:=Sheet2km.Range['D1', Sheet2km.Range['D1'].End[xlDown]];

str119:= Sheetkm.Range[FAdd119].Offset[5,12].Address; //a partir de la celda encontrada busc
Range2:=Sheet2km.Range['E1']; //Eligo la posicion donde se copian los datos (columna A)
Range1:=Sheetkm.Range[str119, Sheetkm.Range[str119].End[xlDown].Offset[-1,0]].Copy(Range2);

str119:= Sheetkm.Range[FAdd119].Offset[5,17].Address; //a partir de la celda encontrada busc
Range2:=Sheet2km.Range['F1']; //Eligo la posicion donde se copian los datos (columna A)
Range1:=Sheetkm.Range[str119, Sheetkm.Range[str119].End[xlDown].Offset[-1,0]].Copy(Range2);

BookOrigen106119.Save;
Row3:=Sheet2km.Range['A1',Sheet2km.Range['A1'].End[xlDown]].Row +
Sheet2km.Range['A1',Sheet2km.Range['A1'].End[xlDown]].Rows.Count-1;  //la ultima fila de tra

DataModule1.Tabla119ADOQuery.Last; //ir a la ultima posicion de la tabla119
bdcount:=DataModule1.Tabla119ADOQuery.FieldByName('Codigo').AsInteger;  //sacar "codigo" de
```

Ilustración 72: Código para copiar los datos a otra hoja Excel

Para poder comprobar la existencia de datos que queremos volcar tenemos que convertir las fechas y horas copiadas en un formato *FormatDateTime.* Mediante una condicional simple *if / then* comparamos los datos de fechas y matrículas. En el caso de encontrar alguna coincidencia a la variable *vv5* asignamos un tres y sacamos un mensaje para preguntar qué queremos hacer: guardar los datos, reemplazar o cancelar la ejecución.

```
for i:=1 to Row3+1 do
begin
  if vv1<>2 then  //numero 2 para vv1 significa que la opcion elegida es "Cancelar"
  begin
    bdcount:=bdcount+1;
    if vv5<>3 then   //numero 3 para vv5 significa que los datos se repiten
    begin
      comprobstr:=FormatDateTime('yyyy-mm-dd', Sheet2km.Cells[i,1]);
      comprobkms:=FormatDateTime('hh:mm:ss', Sheet2km.Cells[i,2]);
      comprobkms2:=FormatDateTime('hh:mm:ss', Sheet2km.Cells[i,4]);
      if ((DataModule1.Tabla119ADOQuery.Locate('Fecha',comprobstr,[])=true)   //si se encuentre l
        and (DataModule1.Tabla119ADOQuery.FieldByName('MatriculaCod').AsInteger=codigo)) then
      begin
        vv5:=3;                                                          //anotamos que los d
        if vv1<>3 then        //numero 3 para vv1 significa que se ha elegido la opcion "reemplaz
        begin                   //si no se lo ha hecho
          buttonSelected := MessageDlg('Se repiten los datos. ¿Quieres reemplazar todos (Yes)?
          +'  La opción (No) guarda los datos como nuevos!                '
          +'(Cancel) para cancelar todo.',mtConfirmation,mbYesNoCancel, 0);
          if buttonSelected = mrYes then     //si confirmamos
          vv1:=3;
          if buttonSelected = mrNo then
          vv1:=1;
          if buttonSelected = mrCancel then
          vv1:=2;
          Screen.Cursor := crHourGlass;
        end
      end
    end;
```

Ilustración 73: Código para comparar los datos

En el caso de no tener los datos repetidos o a la pregunta anterior hemos contestado “guardar”, aplicamos la instrucción *Insert* para volcar directamente los datos preparados.

```
if ((vv5=0) or (vv1=1)) then       //si los datos son nuevos o se ha elegido la opcion
begin
  with DataModule1.Tabla119ADOQuery do
  begin
    Insert;
    FieldByName('Codigo').AsInteger := bdcount;
    FieldByName('MatriculaCod').AsInteger :=codigo;
    FieldByName('Fecha').AsDateTime :=Sheet2km.Cells[i,1];
    FieldByName('Hora_In').AsString :=FormatDateTime('hh:mm:ss',Sheet2km.Cells[i,2]);
    FieldByName('Hora_Fin').AsString :=FormatDateTime('hh:mm:ss',Sheet2km.Cells[i,4]);
    FieldByName('Geofence').AsString :=Sheet2km.Cells[i,5];
    FieldByName('Kms').AsString :=Sheet2km.Cells[i,6];
  end;
end;
```

Ilustración 74: Consulta SQL Insert

En el caso de querer reemplazar los datos, primero tenemos que sacar los mediante la consulta SQL con los parámetros correspondientes de matrícula, fecha de entrada y salida. Luego borramos los datos con la instrucción *Delete* y volvemos a volcar los datos preparados con la instrucción *Insert*.

```
if vv5=3 then        //si se ha elegido reemplazar
begin
  comprobstr:=FormatDateTime('yyyy-mm-dd', Sheet2km.Cells[i,1]);
  comprobkms:=FormatDateTime('hh:mm:ss', Sheet2km.Cells[i,2]);
  comprobkms2:=FormatDateTime('hh:mm:ss', Sheet2km.Cells[i,4]);
  repeat
    DataModule1.ComprobADOQuery.Close;
    DataModule1.ComprobADOQuery.SQL.Clear;
    DataModule1.ComprobADOQuery.SQL.ADD('select * from Tabla119 '
    +' where Fecha = :comprobstr and Matricula = :codigo and '
    +' [Hora_In] = :comprobkms and [Hora_Fin] = :comprobkms2');
    DataModule1.ComprobADOQuery.Parameters.ParamByName('comprobstr').Value:=comprobstr;
    DataModule1.ComprobADOQuery.Parameters.ParamByName('codigo').Value:=codigo;
    DataModule1.ComprobADOQuery.Parameters.ParamByName('comprobkms').Value:=comprobkms;
    DataModule1.ComprobADOQuery.Parameters.ParamByName('comprobkms2').Value:=comprobkms2;
    DataModule1.ComprobADOQuery.Open;       //sacamos los datos de la tabla
    DataModule1.ComprobADOQuery.Locate('Fecha',comprobstr,[]);
    if ((DataModule1.ComprobADOQuery.FieldByName('Matricula').AsInteger=codigo)           //
      and (DataModule1.ComprobADOQuery.FieldByName('Hora_In').AsString=comprobkms)
      and (DataModule1.ComprobADOQuery.FieldByName('Hora_Fin').AsString=comprobkms2)) then
    DataModule1.ComprobADOQuery.Delete;                                                   //
  until DataModule1.ComprobADOQuery.Locate('Fecha',comprobstr,[])=false;
  with DataModule1.Tabla119ADOQuery do
  begin
    Insert;                                                  //y volcamos los datos a
    FieldByName('Codigo').AsInteger := bdcount;
    FieldByName('MatriculaCod').AsInteger :=codigo;
    FieldByName('Fecha').AsDateTime :=Sheet2km.Cells[i,1];
    FieldByName('Hora_In').AsString :=FormatDateTime('hh:mm:ss',Sheet2km.Cells[i,2]);
    FieldByName('Hora_Fin').AsString :=FormatDateTime('hh:mm:ss',Sheet2km.Cells[i,4]);
    FieldByName('Geofence').AsString :=Sheet2km.Cells[i,5];
    FieldByName('Kms').AsString :=Sheet2km.Cells[i,6];
  end;
```

Ilustración 75: Código para reemplazar los datos

Los pasos anteriores se repiten para cada fichero excel con las posiciones de datos, variables, objetos *ADO* correspondientes.

Al final añadimos instrucciones para parar el *ProgressBar*, guardar los ficheros modificados, cerrar la aplicación Excel, habilitar los botones, devolver el cursor original, etc. Es conveniente crear *procedure* para apertura y cierre de la ventana del programa para evitar diferentes “bugs”.

```
finally
if vvl=4 then
ProgressBar1.Position:=0
else
ProgressBar1.Position:=maxa+1;
if variables.od1=1 then
BookOrigen306.Save; //guardamos los cambios e.
if variables.od2 = 1 then
BookOrigenGest.Save;  //guardamos los cambios
if variables.od3=1 then
BookOrigen106119.Save;
if variables.od5=1 then
Book203.Save;
Excel.Quit;          //cerramos los excels
Excel := Unassigned;
tmp := Unassigned;
cien:= Unassigned;
a:=0;
BtnInforme.Enabled := true;    //volvemos boto.
BitBtn3.Enabled := true;
BitBtn4.Enabled := true;
DataModule1.MatriculasADOQuery.Close;
DataModule1.Tabla306ADOQuery.Close;
DataModule1.GestoraADOQuery.Close;
DataModule1.Tabla119ADOQuery.Close;
DataModule1.LTADOQuery.Close;
DataModule1.Tabla203ADOQuery.Close;
end;
Screen.Cursor := crDefault;
```

Ilustración 76: Instrucciones finales del código

Para ver como se queda el resultado final de esta parte compilamos el código y obtenemos la siguiente ventana:

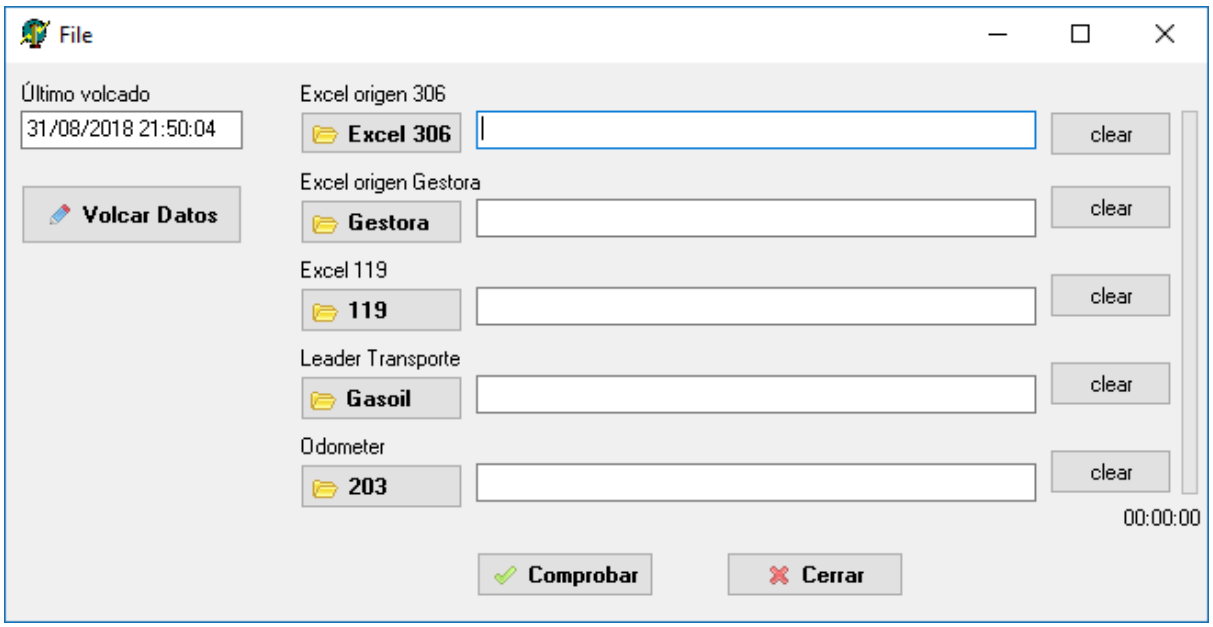


Ilustración 77: Ventana File

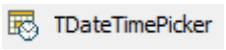
Diseño de formulario para visualización de la información de la base de datos

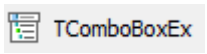
Para visualizar la información de nuestra base de datos y sacar la hoja de dietas creamos un formulario nuevo donde colocamos dos objetos de tipo *TPanel* 

. Uno de los paneles añadidos pegamos a la parte superior de la ventana, aplicando la propiedad *Align - alTop*, y la otra a la parte inferior con *Align - alBottom*. Estos paneles nos van a servir para separar los botones y campos de edición de la tabla *DBGrid*.

El componente *TDBGrid* 

está destinado únicamente para mostrar y editar información asociada a la base de datos, es un reflejo directo de los contenidos actuales de la tabla de la base de datos asociada. Este componente lo colocamos entre los paneles anteriores y para ocupar todo el espacio restante de la ventana aplicamos la propiedad *Align - alClient*.

En el panel superior añadimos dos componentes de tipo *DateTimePicker* 

para poder filtrar nuestros datos por fechas de inicio y fin de viaje, un *ComboBox* 

para filtrar por el nombre de chofer y dos botones para visualizar los datos y crear un fichero Excel con los datos mostrados.

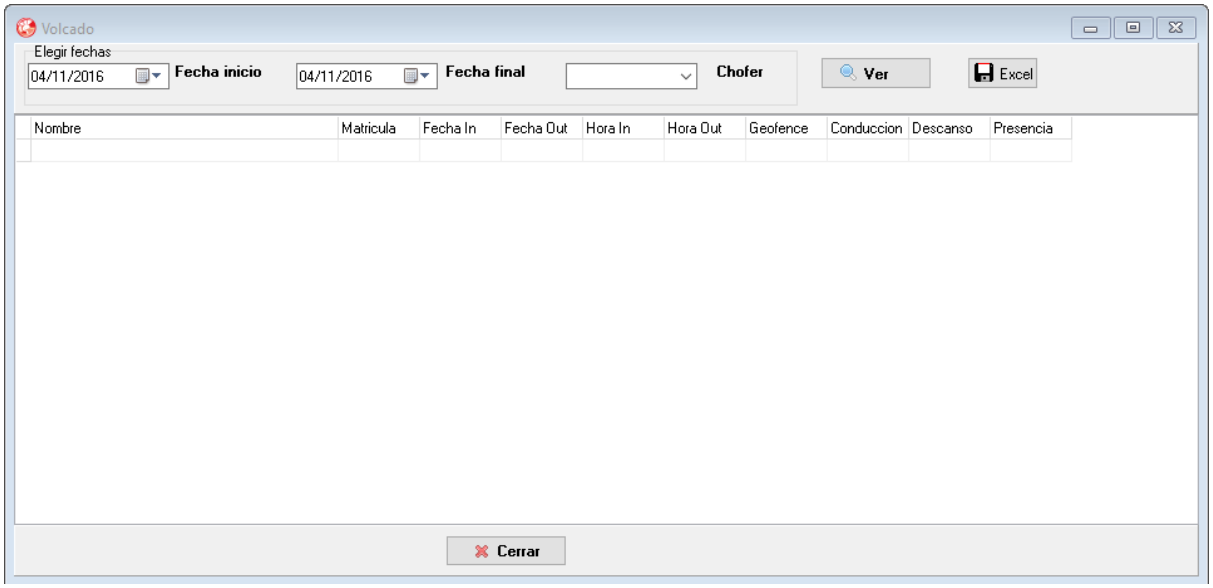


Ilustración 78: Diseño de formulario Volcado

Programación del formulario

En el código del formulario tenemos que definir con la palabra clave *uses* nuestro módulo de datos *Unit1module* para poder conectar la tabla *DBGrid* con *DataSource*. Dicha conexión realizamos con la propiedad *DataSource* del objeto *DBGrid*. En nuestro caso nos sirve *DataSourceTiempos*.

Dentro del procedimiento *FormCreate* describimos los pasos previos. En la ventana de este formulario solamente tenemos que poner la fecha actual de nuestros *DateTimePicker* con la instrucción *DateTime:=Now* para no tenerlos vacíos, aplicar una consulta SQL a través de nuestro componente *ADO* correspondiente a la tabla *Choferes* y copiar los datos de esta tabla de la columna "Apellido" al componente *ComboBox* para poder filtrar los datos según el chofer seleccionado.

```
procedure TFormVolcadoDatos.FormCreate(Sender: TObject);
var
  pk:integer;
  clrbx:string;
begin
  controlexcel:=0;
  DateTimePicker1.DateTime:=Now;
  DateTimePicker2.DateTime:=Now;
  DataModule1.ChoferesADOQuery.Close;
  DataModule1.ChoferesADOQuery.SQL.Clear;
  DataModule1.ChoferesADOQuery.SQL.ADD('select * from Choferes');
  DataModule1.ChoferesADOQuery.Open;       //sacamos los datos de la tabla con los chofer
  DataModule1.ChoferesADOQuery.First;
  for  pk := 0 to DataModule1.ChoferesADOQuery.RecordCount - 1 do     //contador hasta la
  begin
    ComboBox2.Items.Add(DataModule1.ChoferesADOQuery.FieldByName('Apellido1').AsString);
    DataModule1.ChoferesADOQuery.Next;
  end;
  ComboBox2.ItemIndex:=0;
end;
```

Ilustración 79: Código para procedimiento de creación del formulario

En el procedimiento *Change* del elemento *ComboBox* tenemos que añadir los pasos para aplicar el filtro cada vez que cambiamos de chofer. Realmente este trozo consiste en cambiar la línea de filtro de *ADOQuery* correspondiente y volver a aplicar este filtro.

```
procedure TFormVolcadoDatos.ComboBox2Change(Sender: TObject);
begin
    if ComboBox2.Text='Todos' then
    DataModule1.CalcADOQuery.Filtered:=False
    else
    begin
      DataModule1.CalcADOQuery.Filter:=
      'Transportista LIKE ' + #39 + '%' + ComboBox2.Text + '%' + #39;
      DataModule1.CalcADOQuery.Filtered:=True;
    end;
end;
```

Ilustración 80: Código para aplicar el filtro por apellido de chofer

Otros elementos para filtrar los datos son fechas de inicio y fin. Al cambiar la fecha de *DateTimePicker* esta información se guarda en las variables *dateforbd[n]* en formato *TDateTime*.

```
procedure TFormVolcadoDatos.DateTimePicker1Change(Sender: TObject);
var
Numero:string;
begin
  dateforbd1:=DateTimePicker1.Date;           //fecha ini
end;

procedure TFormVolcadoDatos.DateTimePicker2Change(Sender: TObject);
var
numero:string;
begin
  dateforbd2:=DateTimePicker2.Date;          //fecha fin
end;
```

Ilustración 81: Código para DateTimePicker

Pasamos a programar el botón *Ver* con el objetivo de visualizar datos de diferentes tablas en nuestro elemento *DBGrid* según los filtros aplicados. En esta parte del código principalmente trabajaremos con consultas SQL donde indicaremos los parámetros de búsqueda y agrupación de datos. En primer lugar, comprobamos que hemos seleccionado las fechas de inicio y fin y luego creamos las consultas SQL para las tablas *Matrícula, Tabla306 y Gestora*. La consulta a la *Tabla306* parametrizamos según las matrículas activas y a la *Gestora* según matrículas activas y fechas de inicio y final, ordenando también los datos por la matrícula y fecha.

```
begin
  Screen.Cursor := crHourGlass;  //cursor de espera
  if((dateforbd1 <> 0) and (dateforbd2 <> 0)) then          //si se ha elegido as fechas de inic
  begin
    if DataModule1.ClientDataSet1.Active=true then
    DataModule1.ClientDataSet1.Active:=false;
    DataModule1.MatriculasADOQuery.Close;
    DataModule1.MatriculasADOQuery.SQL.Clear;
    DataModule1.MatriculasADOQuery.SQL.ADD('select * from Matriculas');
    DataModule1.MatriculasADOQuery.Open;
    DataModule1.Tabla306ADOQuery.Close;
    DataModule1.Tabla306ADOQuery.SQL.Clear;
    DataModule1.Tabla306ADOQuery.SQL.ADD('select Tabla306.Codigo,Tabla306.Matricula as
    +' MatriculaCod,Matriculas.Matricula,Fecha,Hora,Geofence  from Tabla306 INNER JOIN
    +' Matriculas ON Matriculas.Codigo=Tabla306.Matricula where Matriculas.Activo=:s');
    DataModule1.Tabla306ADOQuery.Parameters.ParamByName('s').Value:='s';
    DataModule1.Tabla306ADOQuery.Open;
    DataModule1.GestoraADOQuery.Close;
    DataModule1.GestoraADOQuery.SQL.Clear;
    DataModule1.GestoraADOQuery.SQL.ADD('select Gestora.Codigo,Gestora.Matricula as
    +' MatriculaCod,Matriculas.Matricula,Gestora.Transportista,Gestora.Lugar,
    +' Gestora.Recoger,Gestora.Devolver,Gestora.Cliente,'
    +' Gestora.Fecha,Gestora.Hora,Gestora.Poblacion,Gestora.Importe,Gestora.Kms,
    +' Gestora.Plataforma,Gestora.fecha2'
    +' from Gestora inner join Matriculas on Matriculas.Codigo=Gestora.Matricula'
    +' where Matriculas.Activo=:s and Gestora.Fecha>=:dateforbd1 and Gestora.Fecha<=:dateforbd2 '
    +' order by Gestora.Matricula,Gestora.Fecha');
    DataModule1.GestoraADOQuery.Parameters.ParamByName('s').Value:='s';
    DataModule1.GestoraADOQuery.Parameters.ParamByName('dateforbd1').Value:=dateforbd1;
    DataModule1.GestoraADOQuery.Parameters.ParamByName('dateforbd2').Value:=dateforbd2;
    DataModule1.GestoraADOQuery.Open;

    DataModule1.GestoraADOQuery.First;
```

Ilustración 82: Código de consultas SQL

Cuando tenemos todos los datos seleccionados pasamos a agruparlos en una tabla nueva, llamamos este componente *CalcADOQuery*.

```
DataModule1.CalcADOQuery.Close;
DataModule1.CalcADOQuery.SQL.Clear;
DataModule1.CalcADOQuery.SQL.ADD('select distinct Gestora.Transportista,
+ ' Matriculas.Matricula,Tabla306.Fecha,Tabla306.Hora,Tabla306.Geofence,
+ ' Tabla306.Conduccion,Tabla306.Descanso,Tabla306.Presencia '
+ ' from Matriculas '
+ ' inner join Tabla306 on Tabla306.Matricula = Matriculas.Codigo '
+ ' inner join Gestora on Gestora.Matricula = Matriculas.Codigo, '
+ ' (select b.Geofence,MIN(b.Codigo) minid,MAX(b.Codigo) maxid '
+ ' from Tabla306 b group by b.Geofence,b.Fecha,b.Hora) c '
+ ' where Tabla306.Fecha >= :dateforbd1 and Tabla306.Fecha <= :dateforbd2 '
+ ' and Gestora.Importe <> :Numero and Matriculas.Activo = :s'    //al quitar "Ge
+ ' and (Tabla306.Codigo=c.minid or Tabla306.Codigo=c.maxid)'      //AÑADIDO:  o
+ ' order by Matriculas.Matricula,Tabla306.Fecha,Tabla306.Hora');

DataModule1.CalcADOQuery.Parameters.ParamByName('dateforbd1').Value:=dateforbd1;
DataModule1.CalcADOQuery.Parameters.ParamByName('dateforbd2').Value:=dateforbd2;
DataModule1.CalcADOQuery.Parameters.ParamByName('s').Value:='s';
DataModule1.CalcADOQuery.Parameters.ParamByName('Numero').Value:='0,01';
DataModule1.CalcADOQuery.Open;
DataModule1.CalcADOQuery.First;
```

Ilustración 83: Consulta SQL para agrupar diferentes tablas en una

Hasta aquí hemos conseguido tener casi todos los datos. Nos falta calcular las horas de conducción, presencia en la fábrica y descanso. Lo que tenemos hasta este momento podemos copiar a nuestra tabla *DBGrid* mediante un elemento *ClienDataSet*.

```
DataModule1.ClientDataSet1.CreateDataSet;

DataModule1.ClientDataSet1.Append;                          //añadimmos la primera linea
DataModule1.ClientDataSet1Conduccion.Value:=0;
DataModule1.ClientDataSet1Nombre.Value:=DataModule1.CalcADOQuery.FieldByName('Transportista').AsString;
DataModule1.ClientDataSet1Matricula.Value:=DataModule1.CalcADOQuery.FieldByName('Matricula').AsString;
DataModule1.ClientDataSet1FechaIn.Value:=DataModule1.CalcADOQuery.FieldByName('Fecha').Value;
DataModule1.ClientDataSet1HoraIn.Value:=StrToTime(DataModule1.CalcADOQuery.FieldByName('Hora').Value);
DataModule1.ClientDataSet1Geofence.Value:=DataModule1.CalcADOQuery.FieldByName('Geofence').Value;
DataModule1.ClientDataSet1.Post;
```

Ilustración 84: Código para guardar los datos en la tabla

Para el cálculo de horas correspondientes a conducción, presencia y descanso recorremos los datos de *CalcADOQuery* y para cada nueva vuelta comprobamos las horas, minutos y segundos con la anterior. Los resultados obtenidos salen como *integer*, por tanto, tenemos que añadir un cero delante de los números menores que diez para poder convertirlos en el formato de hora.

```
for  pk := 0 to DataModule1.CalcADOQuery.RecordCount - 1 do      //recorri
begin
      hbt1:=hbt1+hbt;       //sumador de la horas          //
      mbt1:=mbt1+mbt;       //sumador de minutos             // Horas,minut
      if mbt1>=60 then      //si supera 60 minutos           //
      mbt1:=mbt1-60;        //restamos 60                    //
      sbt1:=sbt1+sbt;       //sumador de segundos            //
      if sbt1>=60 then      //si supera 60 seg               //
      sbt1:=sbt1-60;        //restamos 60                  //
      if ((mbt1<10) and (sbt1<10)) then    //siguientes 4 IF's son para añ
      strt:= IntToStr(hbt1)+':'+'0'+IntToStr(mbt1)+':'+'0'+IntToStr(sbt1);
      if ((mbt1<10) and (sbt1>9)) then
      strt:= IntToStr(hbt1)+':'+'0'+IntToStr(mbt1)+':'+IntToStr(sbt1);
      if ((mbt1>9) and (sbt1<10)) then
      strt:= IntToStr(hbt1)+':'+IntToStr(mbt1)+':'+'0'+IntToStr(sbt1);
      if ((mbt1>9) and (sbt1>9)) then
      strt:= IntToStr(hbt1)+':'+IntToStr(mbt1)+':'+IntToStr(sbt1);
```

Ilustración 85: Código para sacar horas

Sacamos los datos de la posición actual para compararlos con anteriores en la siguiente vuelta.

```
calcgeof:=DataModule1.CalcADOQuery.FieldByName('Geofence').Value;
calchora:=StrToTime(DataModule1.CalcADOQuery.FieldByName('Hora').Value);
calchoraout:=StrToTime(DataModule1.CalcADOQuery.FieldByName('Hora').Value);
stime:=DataModule1.CalcADOQuery.FieldByName('Hora').Value;
calchoradate:=DataModule1.CalcADOQuery.FieldByName('Fecha').Value;
sdatetime:=calchoradate[9]+calchoradate[10]+'/'+calchoradate[6]+calchoradate[7]
+'/'+calchoradate[1]+calchoradate[2]+calchoradate[3]+calchoradate[4]+' '+stime; //ca
calchoranombre:=DataModule1.CalcADOQuery.FieldByName('Transportista').AsString;
cdate:=StrToDateTime(sdatetime);
DataModule1.CalcADOQuery.Next;    //siguiente posicion de la tabla
stime:=DataModule1.CalcADOQuery.FieldByName('Hora').Value;
calchoradate1:=DataModule1.CalcADOQuery.FieldByName('Fecha').Value;
sdatetime:=calchoradate1[9]+calchoradate1[10]+'/'+calchoradate1[6]+calchoradate1[7]
+'/'+calchoradate1[1]+calchoradate1[2]+calchoradate1[3]+calchoradate1[4]+' '+stime;
  cdate1:=StrToDateTime(sdatetime);
calcgeof1:=DataModule1.CalcADOQuery.FieldByName('Geofence').Value;
calchora1:=DataModule1.CalcADOQuery.FieldByName('Hora').Value-calchora;
dbt:=DaysBetween(cdate1,cdate);                                    //
hbt:=HoursBetween(cdate1,cdate);                                   // sacamos la c
mbt:=MinutesBetween(cdate1,cdate)-HoursBetween(cdate1,cdate)*60;   //
sbt:=SecondsBetween(cdate1,cdate)-MinutesBetween(cdate1,cdate)*60; //
```

Ilustración 86: Código para obtener los datos actuales

Y, por último, si geofence actual se ha cambiado respecto al anterior, guardamos los datos en *ClientDataSet*.

```
if calcgeof1 <> calcgeof then     //si geofences de la posicion
begin
  DataModule1.ClientDataSet1.Last;
  DataModule1.ClientDataSet1.Prior;
  dateprior:=DataModule1.ClientDataSet1FechaOut.Value;  //la fe
  DataModule1.ClientDataSet1.Last;
  DataModule1.ClientDataSet1.Edit;
  DataModule1.ClientDataSet1Presencia.Value:= strt;
  DataModule1.ClientDataSet1HoraOut.Value:=calchoraout;
  DataModule1.ClientDataSet1FechaOut.Value:=calchoradate;
```

Ilustración 87: Código para guardar los datos en ClientDataSet

En el caso contrario comprobamos si el chofer estaba presente en la fábrica o descansando. Para ello sacamos las primeras cuatro letras del nombre de geofence y si la palabra obtenida es “area” entonces las horas calculadas anteriormente como *presencia* pasamos a *descanso* y guardamos todos los datos.

```
plb:=calcgeof;        //saco geofence
str:=plb[1]+plb[2]+plb[3]+plb[4];   //guardo las primeras 4 letras
str:=AnsiUpperCase(str);            //paso a mayusculas
strclr:=str;
if str='AREA' then                  //comparo estas 4 letras con la palabra "AREA"
begin
  DataModule1.ClientDataSet1Descanso.Value:=strt;   //si geofence corresponde al area el tiempo de "pre
  DataModule1.ClientDataSet1Presencia.Clear;        //y limpio el campo de "presencia"
end;
DataModule1.ClientDataSet1.Post;
DataModule1.ClientDataSet1.Append;                              //guardo los datos nuevos del dia actual
DataModule1.ClientDataSet1Conduccion.Value:= calchoral;
DataModule1.ClientDataSet1Nombre.Value:=DataModule1.CalcADOQuery.FieldByName('Transportista').AsString;
DataModule1.ClientDataSet1Matricula.Value:=DataModule1.CalcADOQuery.FieldByName('Matricula').AsString;
DataModule1.ClientDataSet1FechaIn.Value:=DataModule1.CalcADOQuery.FieldByName('Fecha').Value;
DataModule1.ClientDataSet1HoraIn.Value:=StrToTime(DataModule1.CalcADOQuery.FieldByName('Hora').Value);
DataModule1.ClientDataSet1Geofence.Value:=DataModule1.CalcADOQuery.FieldByName('Geofence').Value;
DataModule1.ClientDataSet1Presencia.Value:= strt;
DataModule1.ClientDataSet1.Post;
```

Ilustración 88: Código para comprobar la actividad y guardar los datos

Como resultado final obtenemos la siguiente pantalla con la información preparada para crear la hoja de dietas en el formato Excel. Con los filtros en la parte superior sacamos las horas de conducción, presencia en la fábrica y descanso en un intervalo de fechas para un chofer concreto.

Volcado

Elegir fechas | 03/10/2016 | Fecha inicio | 17/10/2016 | Fecha final | Lozano | Chofer | Ver | Excel

Nombre	Matricula	Fecha In	Fecha Out	Hora In	Hora Out	Geofence	Conduccion	Descanso	Presencia
LOZANO	2673FJJ	2016-10-03	2016-10-03	8:23:07	8:35:25	area parking	0:00:00	0:12:18	
LOZANO	2673FJJ	2016-10-03	2016-10-03	8:39:00	13:42:08	TALLER	0:03:35		5:03:08
LOZANO	2673FJJ	2016-10-03	2016-10-03	13:44:27	14:41:31	AREA NIII	0:02:19	0:57:04	
LOZANO	2673FJJ	2016-10-03	2016-10-03	14:42:35	14:55:23	area parking	0:01:04	0:12:48	
LOZANO	2673FJJ	2016-10-03	2016-10-03	15:03:26	16:06:34	taller	0:08:03		1:03:08
LOZANO	2673FJJ	2016-10-03	2016-10-03	16:11:02	16:18:15	LUCOTRANS	0:04:28		0:07:13
LOZANO	2673FJJ	2016-10-03	2016-10-03	16:34:47	16:40:47	area buzones 2	0:16:32	0:06:00	
LOZANO	2673FJJ	2016-10-03	2016-10-03	16:41:09	17:53:45	noatum	0:00:22		1:12:36
LOZANO	2673FJJ	2016-10-03	2016-10-03	18:16:46	18:41:48	TALLER	0:23:01		0:25:02
LOZANO	2673FJJ	2016-10-03	2016-10-04	18:45:07	7:55:33	area parking	0:03:19	13:10:26	
LOZANO	2673FJJ	2016-10-04	2016-10-04	8:25:54	8:38:41	LUCOTRANS	0:30:21		0:12:47
LOZANO	2673FJJ	2016-10-04	2016-10-04	8:56:28	9:25:55	area buzones 2	0:17:47	0:29:27	
LOZANO	2673FJJ	2016-10-04	2016-10-04	9:26:16	9:55:46	noatum	0:00:21		0:29:30
LOZANO	2673FJJ	2016-10-04	2016-10-04	10:11:45	11:14:33	LUCOTRANS	0:15:59		0:01:48
LOZANO	2673FJJ	2016-10-04	2016-10-04	11:30:50	11:35:54	area buzones 2	0:16:17	0:05:04	
LOZANO	2673FJJ	2016-10-04	2016-10-04	11:36:30	12:02:49	noatum	0:00:36		0:26:19
LOZANO	2673FJJ	2016-10-04	2016-10-04	12:10:10	12:31:15	AREA sedavi	0:07:21	0:21:05	
LOZANO	2673FJJ	2016-10-04	2016-10-04	12:48:52	13:32:21	LUCOTRANS	0:17:37		0:42:29
LOZANO	2673FJJ	2016-10-04	2016-10-04	13:53:44	14:35:33	MSC	0:21:23		0:41:49
LOZANO	2673FJJ	2016-10-04	2016-10-04	15:00:02	15:45:30	area parking	0:24:29	0:45:28	
LOZANO	2673FJJ	2016-10-04	2016-10-04	15:53:01	18:01:37	COMERCIAL SPAINW	0:07:31		2:08:36
LOZANO	2673FJJ	2016-10-04	2016-10-04	18:19:52	18:33:35	noatum	0:18:15		0:13:43
LOZANO	2673FJJ	2016-10-04	2016-10-05	18:56:56	7:57:53	area parking	0:23:21	13:00:57	
LOZANO	2673FJJ	2016-10-05	2016-10-05	8:29:09	8:34:37	LUCOTRANS	0:31:16		0:05:28

Cerrar

Ilustración 89: Tabla con los datos finales

	A	B	C	D	E	F	G	H	I
1									
2		**CONDUCTOR:**		**LOZANO**					
3									
4		DIAS TRABAJO:		9					
5		HORAS TRABAJO:		72					
6		HORAS CONDUCCION:		41:32:11					
7		HORAS PRESENCIA:		34:54:55					
8		HORAS TRABAJADAS:		76:27:06					
9									
10									
11	**Matricula**	**Fecha In**	**Fecha Out**	**Hora In**	**Hora Out**	**Geofence**	**Conduccion**	**Descanso**	**Presencia**
12	2673FJJ	03/10/2016	03/10/2016	8:23:07	8:35:25	area parking	0:00:00	0:12:18	
13	2673FJJ	03/10/2016	03/10/2016	8:39:00	13:42:08	TALLER	0:03:35		5:03:08
14	2673FJJ	03/10/2016	03/10/2016	13:44:27	14:41:31	AREA NIII	0:02:19	0:57:04	
15	2673FJJ	03/10/2016	03/10/2016	14:42:35	14:55:23	area parking	0:01:04	0:12:48	
16	2673FJJ	03/10/2016	03/10/2016	15:03:26	16:06:34	taller	0:08:03		1:03:08
17	2673FJJ	03/10/2016	03/10/2016	16:11:02	16:18:15	LUCOTRANS	0:04:28		0:07:13
18	2673FJJ	03/10/2016	03/10/2016	16:34:47	16:40:47	area buzones 2	0:16:32	0:06:00	
19	2673FJJ	03/10/2016	03/10/2016	16:41:09	17:53:45	noatum	0:00:22		1:12:36
20	2673FJJ	03/10/2016	03/10/2016	18:16:46	18:41:48	TALLER	0:23:01		0:25:02
21	2673FJJ	03/10/2016	04/10/2016	18:45:07	7:55:33	area parking	0:03:19	13:10:26	
22	2673FJJ	04/10/2016	04/10/2016	8:25:54	8:38:41	LUCOTRANS	0:30:21		0:12:47
23	2673FJJ	04/10/2016	04/10/2016	8:56:28	9:25:55	area buzones 2	0:17:47	0:29:27	

Lozano

Ilustración 90: La hoja de dietas en formato Excel

Conclusiones

Para realizar este trabajo, fue necesario estudiar en profundidad las áreas más importantes de las actividades de la empresa. Comenzando con la selección de GPS para vehículos, es necesario elegir la opción óptima para la introducción de esta tecnología en el sistema de la compañía. Hay que tener claro con qué tipo de mercancía se trabaja, duración de viaje máxima y la cantidad de dispositivos necesaria considerando el posible crecimiento de la flota de la empresa. También estudiar en detalle todas las herramientas incluidas en la plataforma y buscar soluciones no ofrecidas por el proveedor de sistema de monitoreo.

Para cumplir el objetivo de nuestra tarea que no es más que optimizar el trabajo con datos, se debe comprender el componente técnico tanto de la tecnología GPS como de los programas corporativos disponibles. Aplicando los conocimientos de sistemas de redes hemos construido una base de datos estructurada y adaptada a nuestras necesidades. Al mismo tiempo avanzando más en programación visual hemos creado una aplicación potente para alcanzar el objetivo final.

Desarrollando cada vez más el código llegamos a la conclusión que todo se puede mejorar y conseguir aplicaciones cada vez más rápidas y eficientes. Un ejemplo tan simple como un pequeño cambio de diseño en el agrupado del botón de comprobación de los ficheros con el botón de volcado para la ventana *File* le confiere al sistema más eficiencia y la aplicación de la programación paralela para aumentar la velocidad de procesamiento de los datos le aporta más velocidad.

Con todo esto podemos sacar la conclusión que aplicando los conocimientos técnicos y estudiando el sector de transporte hemos cumplido todas las tareas establecidas y aún quedan muchas posibilidades para nuestro crecimiento y desarrollo profesional.

Valoración personal

El desarrollo y confección del presente proyecto ha sido una experiencia positiva en muchos aspectos. En la parte técnica he de decir que han supuesto un reto la resolución de los siguientes aspectos:

- Selección del sistema GPS más adecuado de los existentes en el mercado adaptado al futuro uso.
- Montaje en vehículos e implementación.
- Gestión de los Geofences de la plataforma.
- Desarrollo de la aplicación para el control de la flota de camiones mediante GPS.

En la parte personal he de decir que disfruto de un buen entorno laboral con compañeros que valoran mi trabajo desde que he llegado a la empresa.

Bibliografía

[1] F. Charte, “Programación avanzada con Delphi 2.0”. Ediciones Anaya Multimedia, 1996.

[2] Global AVL, <https://www.globalavl.es/localizadores-gps/gps-vehiculos.html> (11 agosto 2018)

[3] Cartrack, <https://fleet.cartrack.com.es/> (31 agosto 2018)

[4] History of CAN technology, <https://www.can-cia.org/can-knowledge/can/can-history/> (11 agosto 2018)

[5] J.Gabillaud, “SQL Server 2014”, 2013

[6] Mapon, <https://mapon.com.ua/> (11 agosto 2018)

[7] La tienda del GPS, <https://www.latiendadelgps.com> (18 agosto 2018)

[8] Catálogo cartrack GPS, <https://www.cartrackgps.com/vehicle-tracking-itrackproavl.html> (11 agosto 2018)

[9] SNK Software, <http://www.snkey.net> (23 agosto 2018)

[10] DelphiComponent, <https://delphicomponent.ru/> (31 agosto 2018)