

Inicio rápido a Firebird

Xavier Martínez Garsaball
cadetill

Versión 1.1
08 de noviembre de 2004

Índice

1. Antes de empezar....4
2. Instalación del motor....5
2.1 ¿Dónde está?....5
2.2. Instalando el motor....5
3. Transacciones....7
3.1. Nivel de aislamiento (Isolation Level)....7
3.2. Modo de bloqueo....8
4. Mi primera base de datos....9
4.1. Creando la primera base de datos....9
4.2. Terminando de configurar la base de datos....10
5. Creación de tablas y campos....11
5.1. Consideraciones previas....11
5.2. Crear tablas....11
6. Nuestra agenda....13
6.1. El TIBDataBase....13
6.2. El TIBTransaction....15
6.2.1. Propiedades....15
6.2.2. Métodos....16
6.2.3. Configurando el componente....17
6.3. El TIBDataset....17
6.3.1. ¿Por qué escogemos este componente?....17
6.3.2. Funcionamiento del componente....17
6.3.3. Sus métodos....19
6.3.4. Configurando el componente....19
6.4. Facilitando el trabajo al usuario....21
6.5. Conectando hasta los componentes visuales....21
6.6. Cuatro lineas de código....22
7. Para terminar....25

1. Antes de empezar

En este artículo vamos a describir los pasos principales para la instalación del motor de base de datos Firebird y la configuración de los componentes InterBase eXpress (IBX) para una agenda muy sencilla realizada en Delphi. En el momento de la realización de este artículo, la versión más reciente de Firebird es la *1.5.1* y de los IBX la *x.08* (siendo *x* la versión de Delphi). Podéis descargaros estas versiones en Firebird - SourceForge y en CodeCentral respectivamente.

Si queréis ver las novedades incorporadas en la versión 1.5 de este motor, descargaros las **Release Notes** traducidas al español. Desde aquí agradecer a las personas que trabajaron en esta estupenda traducción (entre ellas un buen amigo, Marc Guillot). Hay que tener presente que, la documentación de Firebird se compone de la documentación de Interbase 6 + las Release Notes de Firebird 1 + las Release Notes de Firebird 1.5.

No vamos a entrar en muchos detalles, es sólo una simple iniciación a este motor y a estos componentes.

El motivo para escoger estos componentes de acceso es debido a que nos proporcionan acceso nativo al motor (al menos a Interbase, no tanto a Firebird) y viene “de serie” con Delphi Enterprise. No obstante, los IBX no nos aseguran un acoplamiento al motor al 100% ya que Borland no quiere dar soporte a Firebird. Si queremos componentes que sí se acoplen al 100% podemos escoger entre varios que hay en el mercado como los InterBase Objects (IBO), los FreeIBPlus, Unified Interbase (UIB), Zeos Lib,..... De éstos, los hay de pago y free, con y sin código fuente, pero de los que yo he podido probar, por su similitud de manejo a los IBX y su rápido aprendizaje, me quedo con los segundos mencionados, los FreeIBPlus (FIB).

2. Instalación del motor

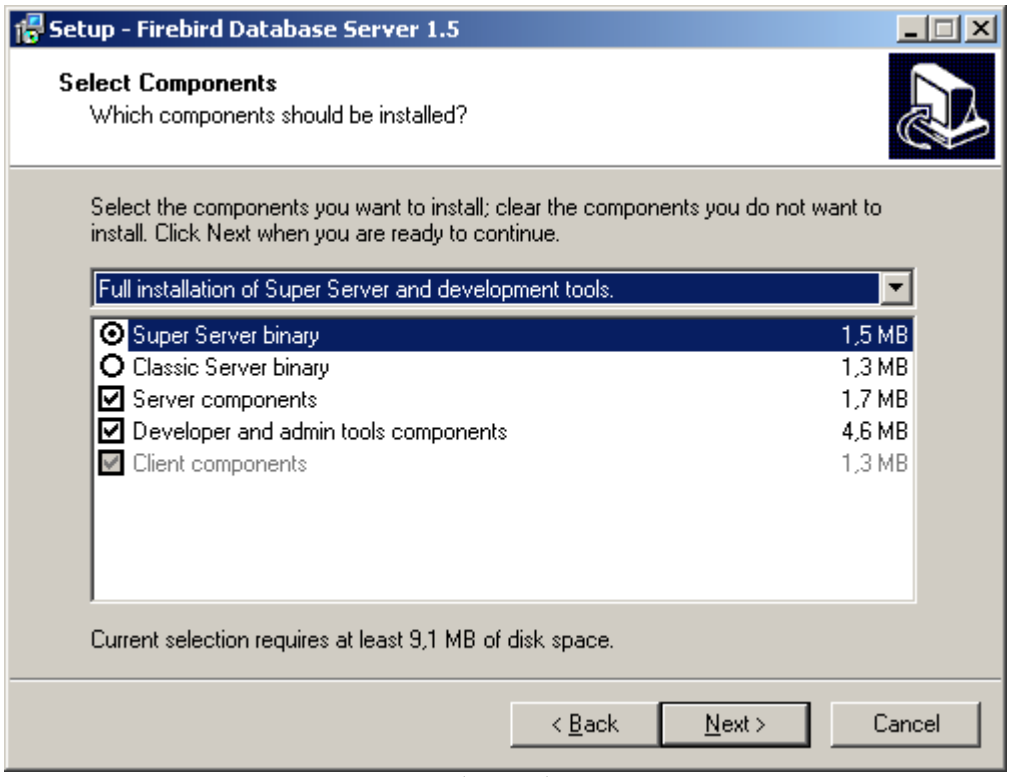
2.1 ¿Dónde está?

Antes de nada, hay que conseguir el motor. ¿De dónde podemos descargarlo? Pues una buena web donde conseguirlo es IBPhoenix o bien directamente desde la web oficial del proyecto en SourceForge.

Este motor lo tenemos para varias plataformas, pero nosotros nos vamos a centrar en Windows, así que nos descargaremos la versión para Windows (a la realización de este documento es la versión 1.5.1.4481 para Win32).

2.2. Instalando el motor

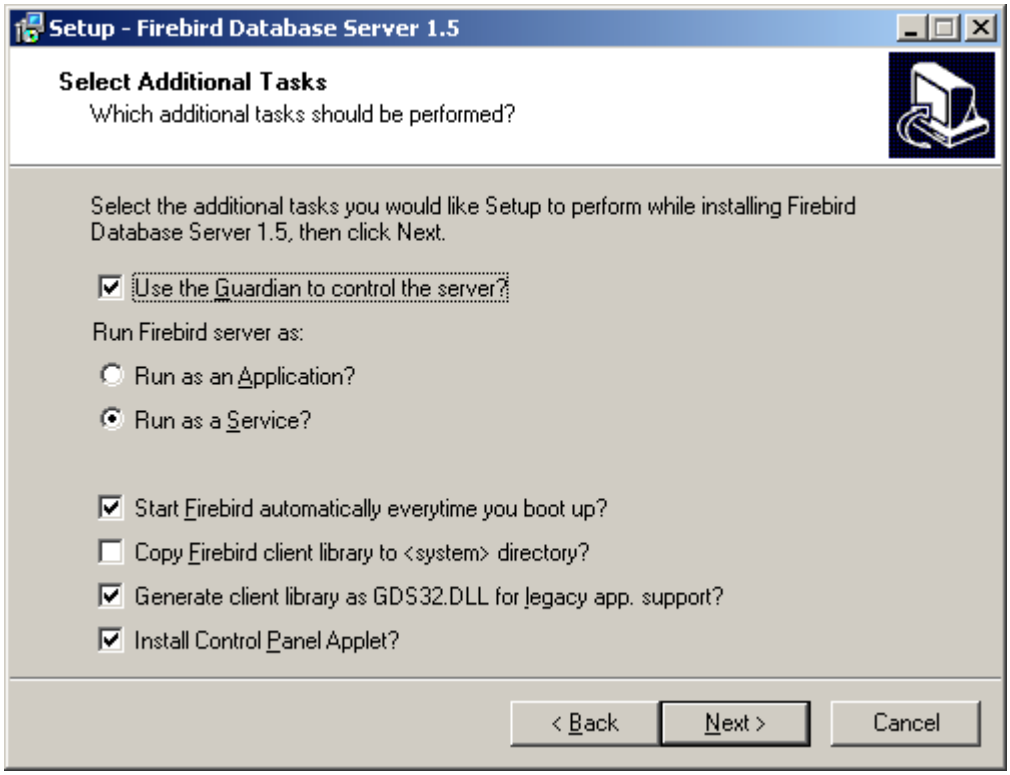
La instalación es simple (es un instalador clásico de Windows). Veamos algunas de las pantallas de la instalación:



(Figura 1)

En la *Figura 1* vemos los tipos de servidores posibles a escoger en la instalación, o el *Super Server* o el *Classic Server*. La diferencia entre uno y el otro es que el primero, el Super Server, crea un único proceso para todas las conexiones mientras que el segundo, el Classic Server, crea tantos procesos como conexiones haya.

Según sea las necesidades, escogeremos una u otra opción. Nosotros escogemos el Super Server.



(Figura 2)

La *Figura 2* nos muestra las distintas opciones de instalación. Lo más destacable de la pantalla es la elección de la forma de ejecución del motor. Podremos escoger entre "*ejecutarlo como una aplicación*" o "*ejecutarlo como un servicio*" ¿Cual escoger? Sencillo, dependiendo del sistema operativo que tengamos. Si estamos en Windows 9x o Me, escogeremos "*Run as an Application*", pero si estamos en uno con tecnología NT (léase NT, 2000, 2003 o XP) escogeremos "*Run as a Service*".

Las demás opciones las dejaremos como muestra la imagen.

Y con esto terminamos la instalación del sistema.

Nota: con esta configuración de instalación estoy dando por supuesto que no existe ninguna versión de Interbase o Firebird instalada en el sistema. Si se tuviera alguna, recomiendo hacer una copia de seguridad de las bases de datos que ya podamos tener y la desinstalación de la versión anterior antes de proceder a la instalación de la nueva versión (recordad a reiniciar el PC antes de la nueva instalación).

3. Transacciones

Este es un tema a tener en cuenta cuando tratamos con motores transaccionales ya que, gracias a ellas(o por culpa de ellas) mantendremos la *coherencia* y *consistencia* de los datos en nuestra base de datos, a la vez que crearemos bloqueos (a nivel de registro) en las tablas que modifiquemos (y durante el tiempo que duren la transacción).

Una transacción no es más que un conjunto de acciones sobre la base de datos que se ejecutan de *forma atómica*. Esto quiere decir que todas las acciones (o operaciones) involucradas en la transacción se aceptarán (*commit*) o se rechazarán (*rollback*) en bloque de y de forma indivisible.

Dicho de otra forma, los motores de bases de datos han de cumplir 4 propiedades básicas (conocidas como *ACID*):

- **Atomicidad**: las acciones realizadas en la base de datos serán ejecutadas o rechazadas en bloque, sin posibilidad a división.
- **Consistencia**: la transacción ha de garantizar que la base de datos sale de un estado consistente para llegar a otro también consistente.
- **Isolation** (*Aislamiento*): cada transacción se ejecuta independientemente de las demás, como si fuera la única.
- **Durabilidad**: la finalización de la transacción garantiza la durabilidad de los datos.

Por regla general, las transacciones pueden gestionarse de forma automática o manual, aunque el dejarla de forma automática es bastante costoso debido a que se arranca una transacción por cada instrucción y se termina (commit o rollback) al finalizarla.

3.1. Nivel de aislamiento (*Isolation Level*)

Cuando estamos en situaciones de multiusuario, hay que tener especial atención a las transacciones. Con el nivel de aislamiento definimos la visión que tendrá la transacción a los cambios producidos por otras transacciones que se estén ejecutando en ese momento.

Así pues, tenemos los siguientes niveles de aislamiento:

- **Read Commited**: con este nivel, cualquier cambio realizado des de otra transacción y que haya sido confirmado (*commit*) podremos verlo, es decir, tenemos acceso a la última versión confirmada de los datos. Es el nivel más bajo de aislamiento.
- **Read Commited no record_version**: la transacción tiene acceso a los registros modificados por otras y, si los cambios no han sido confirmados, dependiendo del modo de bloqueo (los veremos posteriormente) actuará de una forma y otra. Si el bloqueo está configurado como "*wait*", la transacción esperará a la confirmación de la otra transacción que tenga bloqueado el registro, en cambio, si es "*no wait*", la transacción devolverá error de forma inmediata.
- **SnapShot**: con este nivel de aislamiento se asegura que el usuario siempre vea los mismos datos aunque otros usuarios hagan cambios en las tablas afectadas y los hayan aceptado.
- **SnapShot table stability**: es lo mismo que el nivel de aislamiento anterior, pero con la característica que bloquea la tabla en cuestión a las demás transacciones impidiéndoles sólo la escritura.

Hay que tener presente que en Firebird no existe el nivel de aislamiento "*Read Uncommited*" (o "*Dirty Read*"), es decir, el poder ver los cambios no confirmados por otras transacciones.

3.2. Modo de bloqueo

Este es otro de los parámetros que se pueden configurar al inicio de una transacción. Tiene dos posibles valores:

- **Modo en espera** (**Wait**): si se intenta acceder a un registro bloqueado por otra transacción, la segunda *en llegar* quedará bloqueada a la espera de que la primera termine.
- **Modo sin espera** (**No Wait**): si se intenta acceder a un registro bloqueado por otra transacción, la segunda devolverá el control a la aplicación cliente informándola de la existencia de dicho bloqueo.

Nota: Para todo aquel que quiera ampliar la explicaciones sobre las transacciones, les recomiendo un artículo realizado por el amigo Juan José Rodríguez (kinobi) que es excepcional.

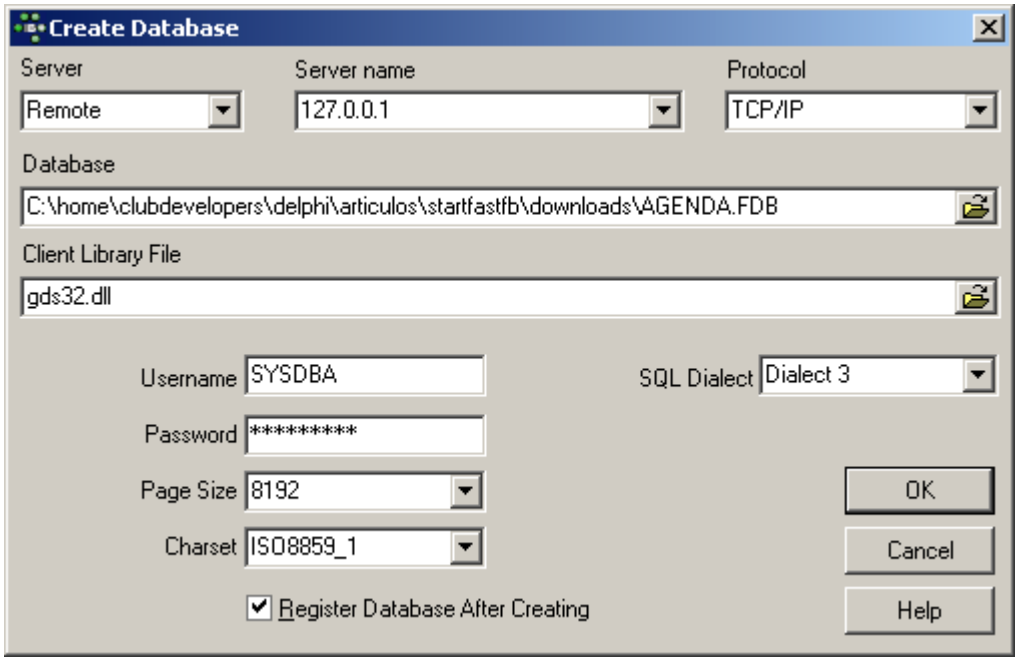
4. Mi primera base de datos

Para facilitarnos las cosas, vamos a usar un programa (de los varios que podemos encontrar por Internet) que nos ayudará en la creación de nuestras bases de datos y la gestión de las mismas. Éste es el IB-Expert que, en su versión personal, es gratuita. Con esta versión (aunque "algo capada") podremos realizar todas las operaciones imprescindibles, así que ya nos sirve para nuestras necesidades. Además, podemos configurarlo en español ;o)

La instalación de esta aplicación es muy sencilla y no creo que valga la pena comentarla. Es un simple instalador y sólo hay que ir siguiendo los pasos.

4.1. Creando la primera base de datos

Para crear una base de datos vamos al menú *Database – Create Database* y nos aparecerá la pantalla de la *Figura 3*.



(Figura 3)

Vamos a comentarla un poco.

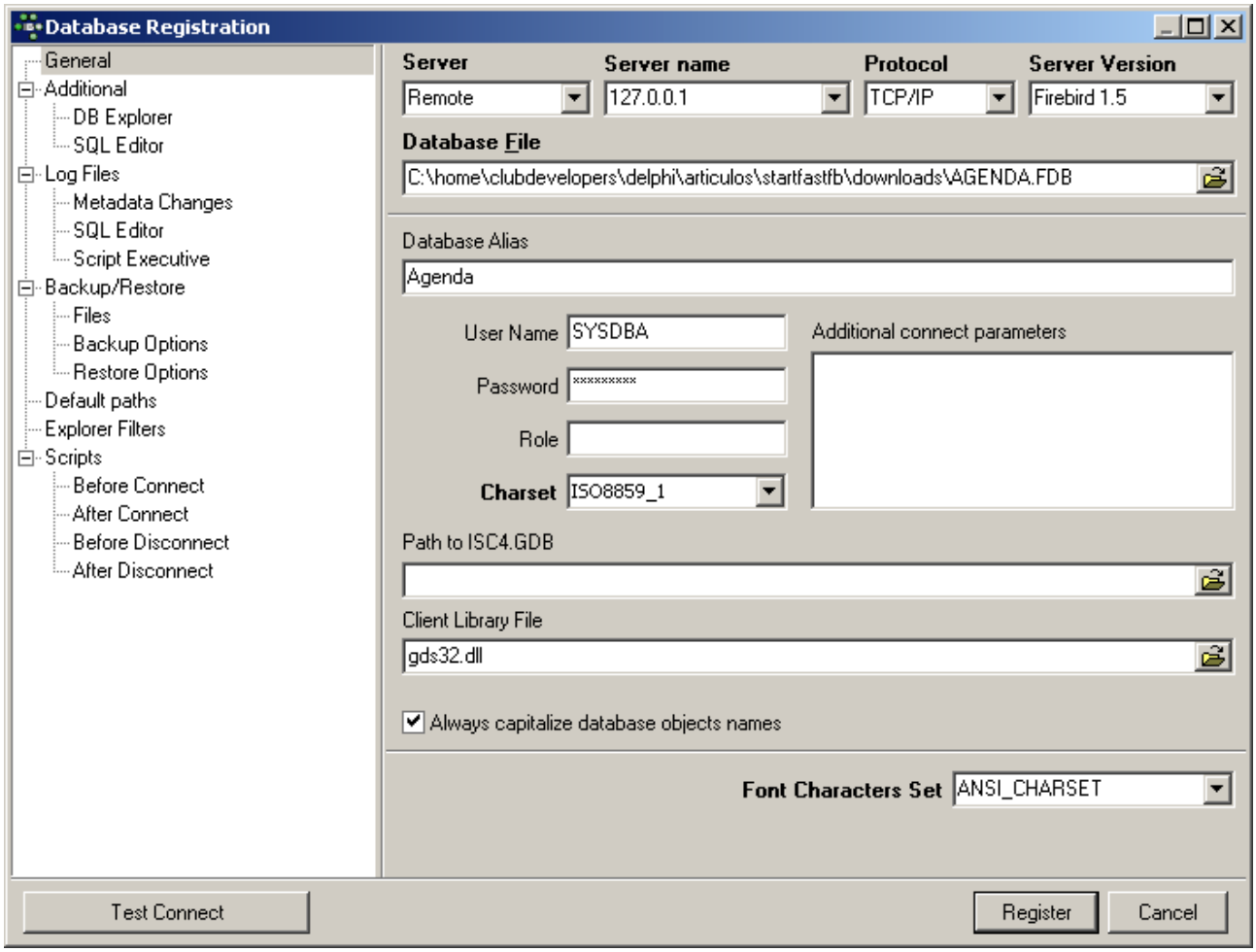
- **Server**: remoto o local. Si escogemos "Remote" tendremos que configurar "Server Name" y "Protocol".
- **Server Name**: nombre del PC servidor de bases de datos. Podemos especificar IP o nombre NetBIOS.
- **Protocol**: protocolo a seguir para la comunicación con el servidor. Normalmente TCP/IP
- **Database**: path completo y nombre de la base de datos. Notad la extensión, es ***.fdb**. ¿Motivo? Primero que es la extensión de las bases de datos de Firebird (aunque podemos poner la que más nos guste) y segundo que la extensión ***.gdb** la utilizan algunos Windows como archivos temporales y la base de datos se vuelve lenta por ese motivo.

- **Client Library Name**: dejarlo tal cual, es la librería donde está el API del motor.
- **Username** y **Password**: usuario y contraseña de acceso al motor. Si no se ha modificado es *sysdba* y *masterkey*.
- **Page Size**: tamaño de la página.
- **Charset**: el ISO8859_1 es el español
- **SQL Dialect**: escogeremos el dialecto 3

Aceptando la pantalla, crearemos nuestra primera base de datos.

4.2. Terminando de configurar la base de datos

Una vez aceptada la pantalla anterior, nos aparecerá la de la *Figura 4* donde podremos terminar de configurar los parámetros que tendrá nuestra base de datos por defecto. Os recomiendo que os la miréis con tranquilidad para conocer todas las posibilidades.



(Figura 4)

Ahora sólo nos queda darle al botón “Register” y ya podemos empezar a trabajar con nuestra nueva base de datos.

5. Creación de tablas y campos

5.1. Consideraciones previas

Antes de realizar cualquier acción sobre la base de datos, tenemos que conectarnos a ella. ¿Cómo lo hacemos? Hay varias maneras, pero la más sencilla es hacer un doble clic encima de la base de datos (también podemos optar por los botones, por el menú del programa, por el menú contextual o por la combinación de teclas Shift+Ctrl+C). Ahora ya estamos en disposición de trabajar con ella.

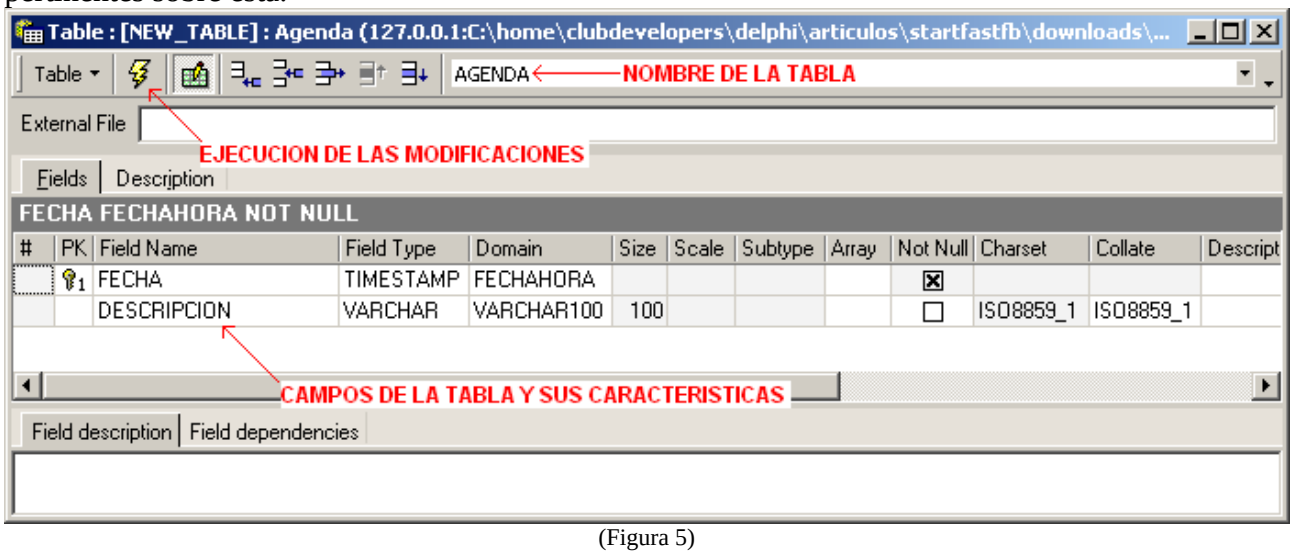
También me gustaría recomendaros el uso de "Domains" para agrupar tipos de datos. La creación de éstos es sencilla, botón derecho encima de "Domains" y escoger "New Domain". En la pantalla que nos aparecerá especificar el nombre, tipo de campo y demás opciones.

Para terminar los "consejos", decir que recomiendo escribir en mayúsculas los nombres de tablas, campos y demás para posteriormente olvidarnos de cómo han sido escritos (si usamos minúsculas, deberemos también usarlas en el momento de escribir las sentencias SQL, si usamos mayúsculas, no importará cómo las escribamos).

5.2. Crear tablas

Aunque podamos realizar todas las operaciones mediante sentencias SQL, ya que estamos en un entorno que nos permite facilitarnos el trabajo, vamos a aprovecharlo. Un detalle bueno que tiene el programa es que puede ayudarnos a conocer un poco el lenguaje SQL y las tablas de sistema de Firebird ya que, al realizar cualquier acción sobre la base de datos, se nos mostrará una pantalla con la sentencia SQL lanzada, así pues, si estamos atentos de lo que nos dice, podemos ir aprendiendo poco a poco.

Dicho esto, vamos a crear nuestras tablas. Para ello haremos clic con el botón secundario encima de "Tables" en el árbol que aparece a la derecha de la pantalla (en entorno MDI) y escogeremos "New Table". Esto nos mostrará la pantalla de la *Figura 5* desde donde vamos a realizar todas las acciones pertinentes sobre ésta.



(Figura 5)

Aquí podemos definir los campos que tendrá nuestra tabla, así como el tipo y otras características. Una vez introducidos los campos y sus características (y creado el índice primario – PK – si se diera

el caso), procederíamos a aceptar los cambios realizados mediante el "*rayo amarillo*". Esto nos mostrará la pantalla anteriormente descrita con la sentencia SQL lanzada y el estado de la misma.

Una vez realizado este primer paso, veremos que nos aparecerán más pestañas en la pantalla. A través de ellas podremos definir los índices secundarios, las relaciones de integridad y demás. Aconsejo revisarlas para ver las posibilidades.

Así crearemos tantas tablas como necesitemos. Para el ejemplo (recordemos, una agenda muy sencilla), el script de las tablas creadas es el siguiente:

```
/******************************************************************************/
/****                                Domains                                 ****/
/******************************************************************************/
CREATE DOMAIN FECHAHORA AS
TIMESTAMP;

CREATE DOMAIN VARCHAR100 AS
VARCHAR(100);

/******************************************************************************/
/****                                 Tables                                 ****/
/******************************************************************************/
CREATE TABLE AGENDA (
    FECHA        FECHAHORA NOT NULL,
    DESCRIPCION  VARCHAR100
);

/******************************************************************************/
/****                              Primary Keys                              ****/
/******************************************************************************/
ALTER TABLE AGENDA ADD CONSTRAINT PK_AGENDA PRIMARY KEY (FECHA);
```

6. Nuestra agenda

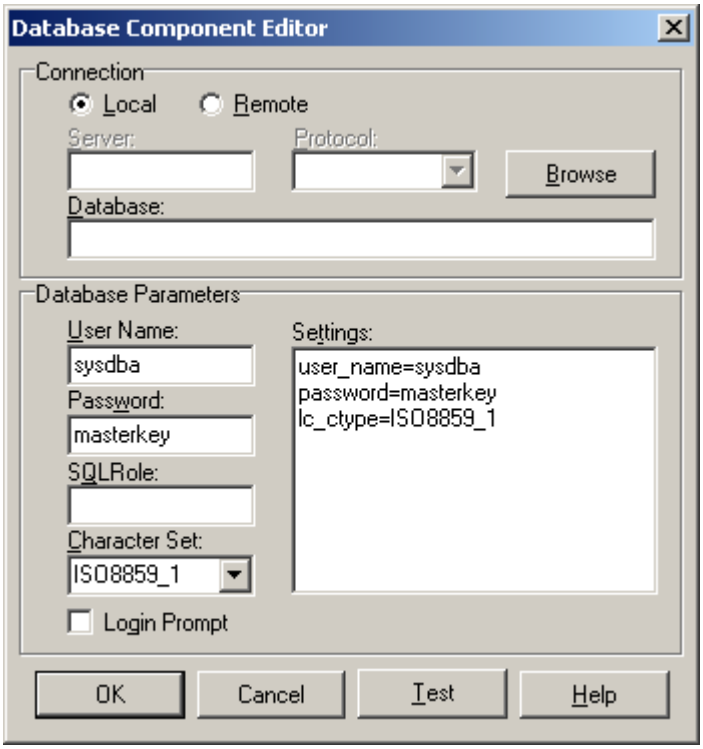
Será una aplicación sencilla, pero que nos va a permitir explicar algunos de los componentes de los IBX, entre ellos, el TIBDatabase, el TIBTransaction y el TIBDatabase.

Como los buenos modos de programación no están reñidos con la sencillez, vamos a tener un módulo de datos (DataModule) donde vamos a colocar los componentes de acceso a la base de datos (y todo el código relacionado) y un formulario donde colocaremos los componentes visuales (y su código relacionado).

Veamos los diferentes componentes necesarios para acceder a la base de datos.

6.1. El TIBDataBase 

Este componente es el que se encargará de establecer la conexión con nuestra base de datos. Para configurarlo, lo mejor es hacer un doble clic encima de él para que se nos muestre la pantalla de la *Figura 6*.



(Figura 6)

Personalmente me gusta mucho especificar la base de datos desde código, así puedo tener mi aplicación en la carpeta que quiera sin depender del código de la aplicación.

Hay varias maneras para controlar la ubicación de la base de datos:

- Poner una clave en el registro de Windows (personalmente, cuanto menos lo toque mejor)
- Tener un archivo Ini de configuración (lo he usado en muchas ocasiones)
- Ubicar la base de datos en una carpeta fija dentro de la carpeta donde esté el ejecutable (con la que nos quedamos en el ejemplo y será la misma carpeta donde resida el ejecutable).
- Cualquier otra opción que se le pueda ocurrir al lector ;o)

Como ya hemos dicho, nos quedamos con la tercera opción. No obstante, haremos el código de tal manera que sea "sencillo" el cambio de una forma a otra. Para ello crearemos un nuevo método en el módulo de datos que se llame "*Connect*". Miremos como quedaría el código de la unit hasta ahora.

```
Interface

uses
  SysUtils, Classes, DB, IBDatabase, Dialogs; // se ha añadido la uses Dialogs

const
  NameBD = 'AGENDA.FDB';

type
  TMainDM = class(TDataModule)
    IBDatabase1: TIBDatabase;
  private
  public
    function Connect: boolean; virtual;
  end;

resourcestring
  errConnectBD = 'Error conectando con la base de datos.'#13#13'Mesaje de error: '#13'%
s';

implementation

function TMainDM.Connect: boolean;
begin
  Result := true;

  // Si estamos conectados, nos desconectamos
  if IBDatabase1.Connected then
    IBDatabase1.Connected := false;

  // asignamos nombre de la base de datos
  IBDatabase1.DatabaseName :=
        IncludeTrailingPathDelimiter(ExtractFilePath(ParamStr(0))) + NameBD;

  // intentamos conectar
  try
    IBDatabase1.Connected := true;
  except
    on E: Exception do
    begin
      ShowMessage(Format(errConnectBD, [E.Message]));
      Result := false;
    end;
  end;
end;
```

Nota: no se ha puesto todo el código de la unit, sólo aquello que tiene relevancia con lo que estamos tratando.

El método "*Connect*" se ha definido "*public*" (para poder ser llamado desde donde lo necesitemos) y "*virtual*" (para poder ser reescrito por un descendiente suyo – herencia, tema en el que no vamos a entrar ya que no es lo que nos ocupa).

El código de este método (función) es muy sencillo. Comprobamos que no estemos conectados, asignamos la ruta de la base de datos e intentamos conectar. Si no sabéis que hace alguna de las funciones usadas, recomiendo mirar la explicación en la ayuda que están muy bien explicadas.

La conexión (la llamada a "*Connect*") la haremos desde el DPR del proyecto, para así, si hay un error, no iniciar ni la aplicación, y siempre después de la creación del formulario.

```
Begin
  // recordar de añadir la unit SysUtils en el uses (requerida para el FreeAndNil)
  Application.CreateForm(TMainDM, MainDM);
  if not MainDM.Connect then
  begin
    FreeAndNil(MainDM);
    Exit;
  end;

  IBUtils.CopyMasterFieldToDetail := true;
  Application.Initialize;
  Application.CreateForm(TMainFrm, MainFrm);

  Application.Run;
end.
```

Antes de terminar la explicación de este componente, no podemos dejar de mencionar una de sus propiedades más importantes, la propiedad "*DefaultTransaction*", donde vamos a enganchar el componente TIBTransacion que explicamos a continuación.

6.2. El TIBTransaction 

Es el componente encargado de gestionar las transacciones de nuestra base de datos y, por lo tanto, uno de los más importantes y del que vamos a detallar todas sus propiedades y formas de uso.

6.2.1. Propiedades

- **Active**: determina si la transacción está activada.
- **AllowAutoStart**: determina si está permitido el comienzo automático de las transacciones.
- **AutoStopAction**: determina qué acción se realizará al cerrarse el último dataset asociado a la transacción. Hay que tener en cuenta que esto sólo afectará a los datasets descendientes de TdataSet, por lo que si sólo se usan TIBSQL es recomendable ponerlo en saNone (como viene por defecto). Los posibles valores son:
  - saNone: valor por defecto. La transacción no se parará implícitamente.
  - saRollback: al cierre del último dataset se realizará un "*Rollback*" de la transacción y esta

se cerrará.
    - saCommit: al cierre del último dataset se realizará un "*Commit*" de la transacción y esta se cerrará.
    - saRollbackRetaining: al cierre del último dataset, se realizará un "*Rollback*" de la transacción conservando el contexto de ésta. Esta opción es sólo aplicable a partir de IB 6.0 o Firebird (en cualquiera de sus versiones).
    - saCommitRetaining: al cierre del último dataset se realizará un "*Commit*" de la transacción conservando el contexto de la misma.
- **Databases**: lista de los TIBDatabases asociados a la transacción.
- **DefaultAction**: determina la acción a realizar al terminar o expirar el temporizador *IdleTimer*. Las posibles acciones son:
    - taRollback: realiza un "*Rollback*" de la transacción.
    - taCommit: realiza un "*Commit*" de la transacción.
    - taRollbackRetaining: realiza un "*Rollback*" de la transacción conservando el contexto de la misma.
    - taCommitRetaining: realiza un "*Commit*" de la transacción conservando el contexto de la misma.
- **DefaultDatabase**: determina cuál es el TIBDatabase asociado a la transacción por defecto.
- **IdleTimer**: determina el tiempo (en milisegundos) de espera de la transacción inactiva antes de realizar la acción determinada en *DefaultAction*.
- **InTransaction**: determina si la transacción está o no en progreso.
- **Params**: determina los parámetros asociados a la transacción (para ver los posibles parámetros mirar la documentación de Interbase o Firebird). Estos parámetros puedes especificarse también desde el cuadro de diálogo (*Transaction Editor*) que aparece al realizar doble clic encima del componente.
- **SQLObjects**: contiene una lista de todos los datasets y TIBSQL asociados a la transacción.
- **TPB** (**T**ransaction **P**arameter **B**uffer): propiedad de sólo lectura que facilita un puntero a un buffer con los parámetros de la transacción. Para establecer parámetros a la transacción ver la propiedad *Params*.

6.2.2. Métodos

Sólo vamos a explicar los más importantes o relevantes.

- **Commit**: hace permanentes los cambios realizados a los datasets asociados a la transacción y la cierra encapsulando una llamada a la función del API *isc_commit_transaction*.
- **CommitRetaining**: hace permanentes los cambios realizados a los datasets asociados a la transacción conservando el contexto de la misma encapsulando una llamada a la función del API *isc_commit_retaining*.
- **Rollback**: descarta cualquier cambio realizado a los datasets asociados a la transacción y la cierra encapsulando una llamada a la función del API *isc_rollback_transaction*.
- **RollbackRetaining**: descarta cualquier cambio realizado a los datasets asociados a la transacción conservando el contexto de la misma encapsulando una llamada a la función del API *isc_rollback_retaining*.
- **StartTransaction**: inicia una nueva transacción en el servidor generando un nuevo TPB para la transacción encapsulando una llamada a la función del API *isc_start_multiple*.

6.2.3. Configurando el componente

Una vez explicado el funcionamiento del mismo, vamos a ver cómo lo configuramos para nuestra aplicación.

Empezaremos configurando la propiedad *DefaultDatabase* asignándole nuestro TIBDatabase. Otra propiedad a configurar será el *AutoStopAction* que lo pondremos en *saCommit*. Ahora el *DefaultAction* al que pondremos en *TARollbackRetaining*. Y la última propiedad a modificar desde el Inspector de Objetos será *IdleTimer* al que pondremos 300000 milisegundos (5 min).

Configuradas las propiedades, sólo nos queda especificarle el nivel de aislamiento de la transacción. Para ello tenemos dos opciones; o bien lo hacemos mediante la propiedad *Params* o bien realizando un doble clic encima del componente. Vamos hacerlo de la segunda manera.

Al realizar el doble clic, nos aparecerá la pantalla de la *Figura 7* y lo configuraremos en *ReadCommitted*.

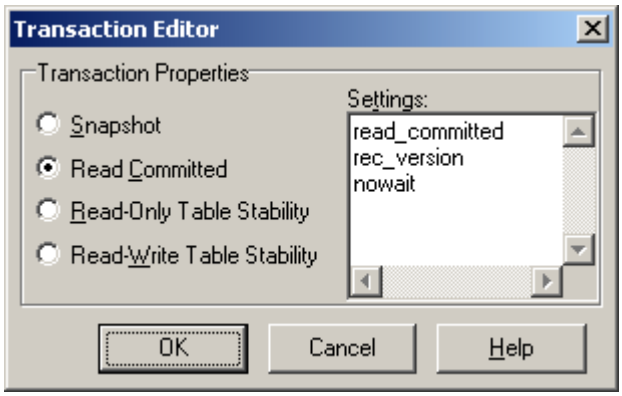


Figura 7

Con estos sencillos pasos ya tenemos configurada nuestra conexión a la base de datos

6.3. El TIBDataset

6.3.1. ¿Por qué escogemos este componente?

Componentes de acceso a datos, InterBase eXpress tiene tres, el TIBTable, el TIBQuery y el TIBDataset. Los dos primeros, a parte de que cualquier persona que haya tratado algo con bases de datos en Delphi le resultará fácil u uso, tienen cada uno alguna desventaja respecto al TIBDataset que hace que prefiera explicar éste.

Desventajas: por un lado, tenemos al TIBTable, el cual nos proporciona acceso a una única tabla de nuestra base de datos. En cambio, el TIBDataset, al funcionar con sentencias SQL, podemos acceder a tantas tablas como quedamos usando *joins*. Esto también puede hacerse con el TIBQuery, pero la desventaja de éste es que, para poder hacer una consulta editable, necesitamos usar un segundo componente, el TIBUpdateSQL, en cambio, con el TIBDataset, con un sólo componente nos basta.

6.3.2. Funcionamiento del componente

Este componente, al igual que el TIBTable y el TIBQuery, deriva de TDataSet, por lo que podremos enlazarlo sin problemas a un TDataSource y, por consiguiente, a componentes DataAware.

La configuración no es muy complicada, basta con definir alguna de sus propiedades principales para que lo tengamos en marcha. No obstante, vamos a ver las más importantes:

- **Active**: derivada de TDataset, abre o cierra el dataset.
- **BufferChunks**: especifica el número de registros que se almacenaran en el buffer en cada una de las lecturas a la base de datos. Cuando la propiedad *Unidirectional* está activada (a *true*), *BufferChunks* no se tiene en cuenta.
- **CachedUpdates**: habilita o deshabilita el uso en caché de las actualizaciones. El estar habilitado implica que los cambios no se grabarán de forma directa en las tablas de la base de datos, sino que se guardarán en una caché en el PC cliente. Cuando los cambios se han terminado, la aplicación ha de forzar la escritura de éstos en la base de datos (haciendo una llamada a *ApplyUpdates*), proceso el cual se realizará en el contexto de una sola transacción. Es recomendable, para un mayor control y para más ventajas, que si se quiere usar actualizaciones en caché se usen el ClientDataSet en lugar de activar esta propiedad.
- **Database**: TIBDatabase al que queremos asociar el componente.
- **DataSource**: para relaciones maestro/detalla, especifica el TDataSource de donde se cogerán los datos.
- **DeleteSQL**: propiedad de tipo TStrings que contiene la sentencia SQL a lanzar en el momento de realizar el borrado de un registro en la tabla (entraremos en más detalle posteriormente).
- **ForcedRefresh**: habilita o deshabilita el refresco automático del dataset cada vez que se realice un *post* en la tabla.
- **GeneratorField**: especifica un generador que proporcione valores automáticamente a uno de los campos del dataset. Han de usarse las propiedades del *TIBGeneratorField* para indicar qué valor de los campos ha de ser asignado y cómo ha de ser calculado.
- **InsertSQL**: propiedad de tipo TStrings que contiene la sentencia SQL a lanzar en el momento de realizar una inserción en la tabla (entraremos en más detalle posteriormente).
- **ModifySQL**: propiedad de tipo TStrings que contiene la sentencia SQL a lanzar en el momento de realizar una modificación en la tabla (entraremos en más detalle posteriormente).
- **Plan**: indica el plan usado por el motor para realizar la última sentencia SQL lanzada desde el componente.
- **RefreshSQL**: propiedad de tipo TStrings que contiene la sentencia SQL a lanzar en el momento de realizar el refresco de la(s) tabla(s) (entraremos en más detalle posteriormente).
- **RowsAffected**: indica el número de registros afectados en la última sentencia SQL lanzada desde el componente.
- **SelectSQL**: propiedad de tipo TStrings que contiene la sentencia SQL a lanzar en el momento de realizar una selección de la(s) tabla(s) (entraremos en más detalle posteriormente).
- **Transaction**: componente *TIBTransaction* asociado a database asignado en *Database*.
- **UniDirectional**: especifica si se activará la bidireccionalidad del cursor. Si se activa la unidireccionalidad, sólo se podrá hacer lectura secuencial, no habrá acceso aleatorio ni se podrá retroceder. Se aconseja el uso cuando usamos un TDatasetProvider y un TClientDataSet asociados al componente.

Algunas de estas propiedades pertenecen a TIBCustomDataset, del cual derivan tanto el TIBDataset como el TIBTable y el TIBQuery, por lo que nos sirve de explicación ya de las propiedades *no normales* de estos componentes (entiéndase por "*no normales*" aquellas propiedades que aparecen en el TIBCustomDataset y que no han sido heredades de TDataset).

6.3.3. Sus métodos

Repasemos, pues, los más importantes del componente y que no dependan de TDataset.

- **ApplyUpdates**: escribe los cambios pendientes de actualización que se encuentran en la caché.
- **ExecSQL**: procedimiento al que hay que llamar si queremos ejecutar la sentencia SQL asignada a la propiedad SelectSQL siempre y cuando esta no sea de selección (sentencia *SELECT*).
- **FetchAll**: recoge todos los registros que devuelve una consulta SQL para almacenarlos localmente. No es muy aconsejable su uso para consultas "*pesadas*" (entiéndase por "*pesadas*" aquellas que devuelvan muchos registros).

6.3.4. Configurando el componente

Una vez tenemos claras las propiedades y los métodos de éste componente, podemos empezar a configurarlo a nuestro antojo.

Empezaremos por definirle la propiedad *Database* a la que le asignaremos nuestro TIBDatabase. Vemos que, si ya teníamos asociado el TIBDataset con el TIBTransaction, la propiedad *Transaction* se ha rellenado de forma automática. Si no fuera así, la rellenamos con nuestro TIBTransaction y comprobamos la unión entre el TIBDatabase y el TIBTransaction.

Otra propiedad que vamos a retocar es la *BufferChunks* a la que le asignaremos una cantidad de registros más pequeña que esos 1000 que vienen por defecto. Por ejemplo le asignaremos 50.

Nos fijaremos que *CachedUpdates* esté en *false* (por defecto) y que *UniDirectional* también esté desactivada (también por defecto).

Y ahora vamos a por las propiedades que realmente nos van a dar acceso a los datos. Empezaremos por la que nos traerá los datos del servidor, la *SelectSQL*. Para ello activaremos nuestro TIBDatabase (propiedad *Active* a *true*) para que nos facilite el trabajo y, dando clic en el botón de los puntos suspensivos de la propiedad, entraremos en el *CommandText Editor* (ver Figura 8).

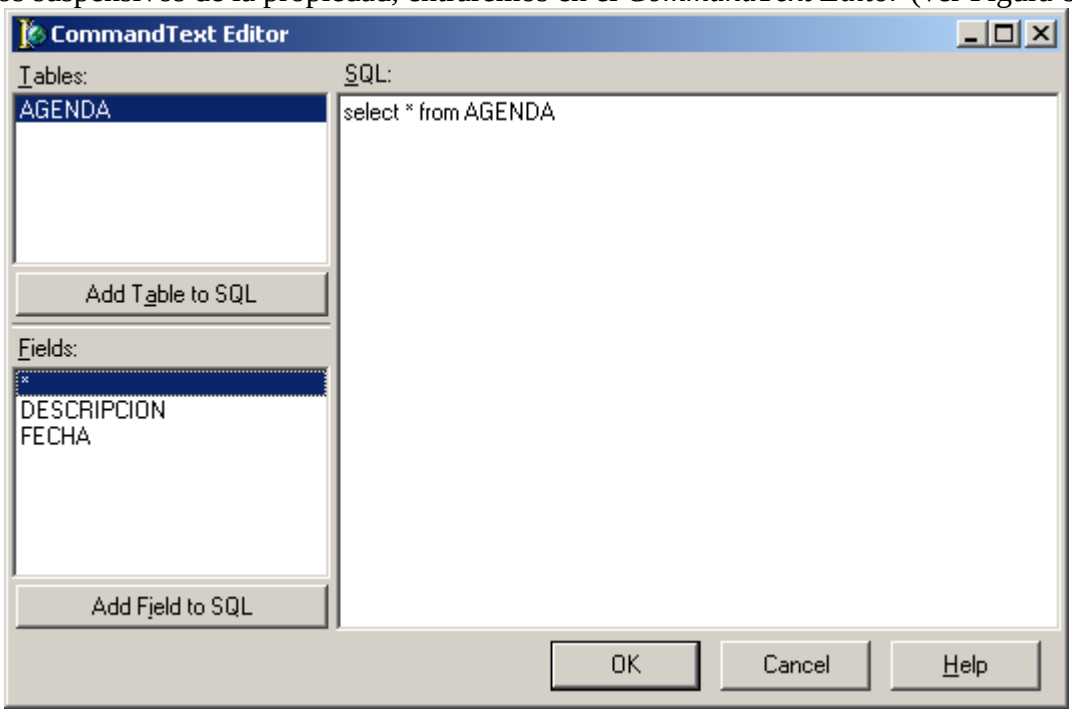


Figura 8

Esta pantalla nos ayudara a realizar la sentencia SQL deseada. Puede ser lo compleja que nosotros queramos, sin problemas, pero en este caso nos conformaremos con algo sencillo como:

```
SelectSQL := 'select * from AGENDA'
```

Para la sentencia de refresco, o lo que es lo mismo, la propiedad *RefreshSQL*, vamos a usar la misma sentencia SQL.

```
RefreshSQL := 'select * from AGENDA'
```

Para terminar, tenemos que informar las sentencias que nos proporcionarán acceso a la gestión de los los datos, es decir, las sentencias para poder añadir (*InsertSQL*), modificar (*ModifySQL*) y borrar (*DeleteSQL*). Para estas propiedades tenemos que construir sentencias SQL válidas según corresponda, es decir, tendremos las siguientes:

```
InsertSQL := 'insert into AGENDA
               (FECHA , DESCRIPCION)
             values
               (:FECHA, :DESCRIPCION)'
ModifySQL := 'update AGENDA
             set
               FECHA = :FECHA,
               DESCRIPCION = :DESCRIPCION
             where
               FECHA = :OLD_FECHA'
DeleteSQL := 'delete from AGENDA
             where
               FECHA = :OLD_FECHA'
```

Fijaros en las sentencias *ModifySQL* y *DeleteSQL* que en la parte del *where* sólo tenemos los campos que forman parte de la clave primaria de la tabla.

Truco

Campo clave en la inserción: a veces se asigna éste mediante un *trigger* en la base de datos. En ese caso quitaremos el(los) campo(s) que formen parte de la clave de nuestra sentencia SQL de inserción. Para que no tengamos errores en el lado cliente, deberemos así mismo establecer la propiedad *Required* de los TFields que conforman la clave a *false*.

Nota: en relaciones maestro/detalle, el TIBDatset detalle no coge por defecto el código de la tabla maestra, sino que nosotros tenemos que ponerlo manualmente en el momento de hacer un Insert en la tabla detalle. Para poder subsanar este pequeño inconveniente, en cualquier parte del programa y, antes de hacer las relaciones maestro-detalle, debemos de poner a *true* la variable *CopyMasterFieldToDetail* que se encuentra en la unit *IBUtils* (por lo que deberá añadirse a la cláusula uses del formulario/módulo de datos donde vayamos a utilizarla).

6.4. Facilitando el trabajo al usuario

Para facilitarle el trabajo al usuario, vamos a añadir los campos persistentes y modificar alguna de sus propiedades. Para ello abriremos el *Fields Editor* haciendo un doble clic encima del TIBDataset. En este, haremos clic con el botón secundario del ratón y escogeremos la opción *Add all fields* (ver Figura 9).

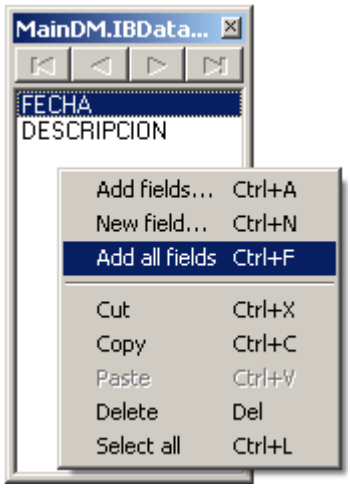


Figura 9

Con esta acción añadimos todos los campos (*TFields*) de nuestra tabla al editor y, seleccionándolos, podremos ir modificando sus propiedades.

Vamos a seleccionar el campo FECHA. En este vamos a cambiar la propiedad *DisplayFormat*, *DisplayLabel* y *EditMask* para ponerle los valores siguientes:

```
DisplayFormat := 'dd/mm/yyyy hh:nn'
DisplayLabel := 'Fecha'
EditMask := '!99/99/0000  !90:00;1;_'
```

Si alguien tiene problemas con estas propiedades, con consulte la ayuda ;o)
Y ahora seleccionamos el otro campo, DESCRIPCION, donde sólo vamos a modificar la propiedad *DisplayLabel*.

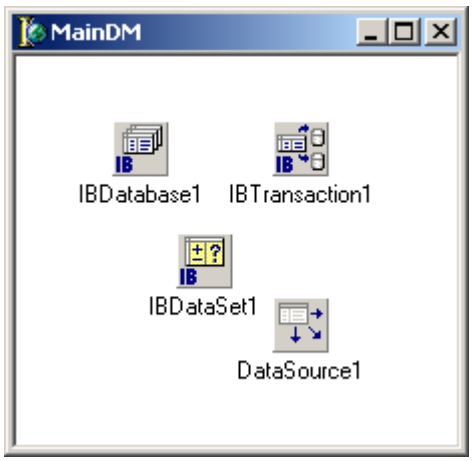
```
DisplayLabel := 'Descripción'
```

6.5. Conectando hasta los componentes visuales

Ya tenemos casi todo listo, ahora sólo nos queda dibujar un formulario para poder hacer la entrada de datos y realizar las conexiones pertinentes desde el TIBDataset hasta los componentes visuales.

En nuestro módulo de datos vamos a añadir un TDataSource y lo vamos a unir al TIBDataset (propiedad *Dataset*). Así, el módulo de datos puede quedar una cosa parecida a esta:

Figura 10



Y en el formulario necesitaremos un TDBGrid, un TDBNavigator y un TDateTimePicker (entre otros y asignandoles las propiedades pertinentes para enlazar con el TDataSource y para hacerlo "bonito") quedando con un aspecto parecido a esto:

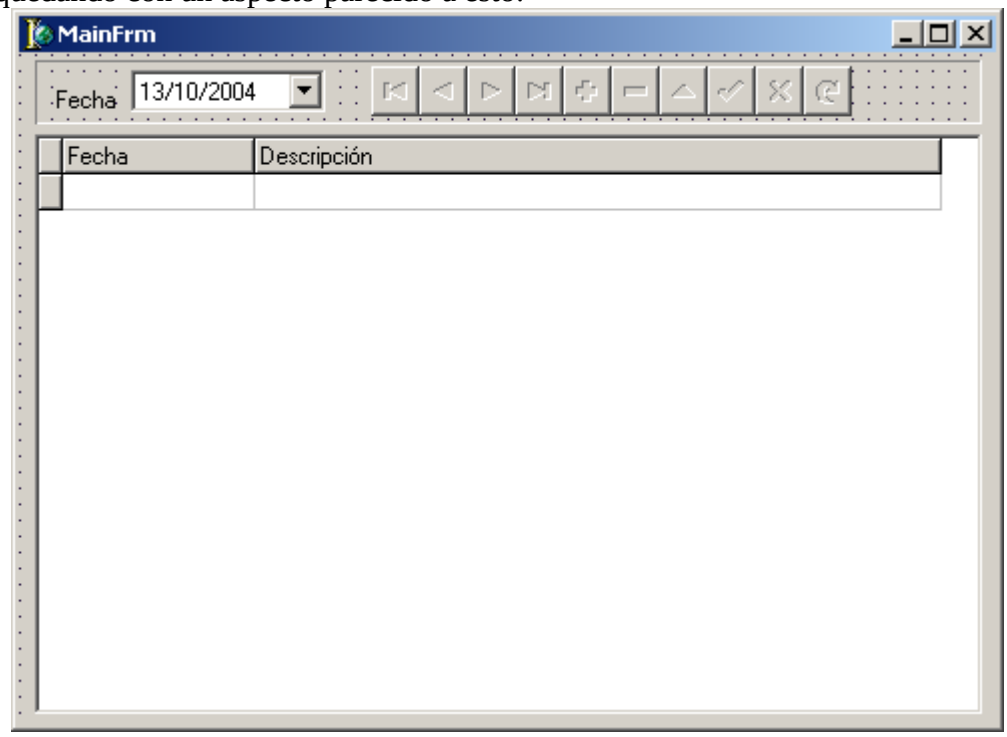


Figura 11

6.6. Cuatro lineas de código

Todo esto está muy bien, pero..... ¿cuándo realizamos el *commit* de la transacción?

Cierto, tenemos configurado nuestro componente TIBTransaction para que empiece una transacción nueva cada vez que realicemos una acción de escritura sobre la tabla, pero aun no hemos definido cuándo realizar el C*ommit* de la transacción. Opciones hay varias:

- Poner un botón de *commit* y que sea el propio usuario el que se encargue de realizarlo.
- Poner un temporizador (*TTimer*) y que cada cierto tiempo realice el C*ommit* si hay una transacción activa.
- etc..... (en este etcétera puede incluir el uso de TClientDataSet, o el uso de CacheUpdates,.....)

Todo depende de la situación y de la forma de programar cada uno. Personalmente me gusta el uso del TClientDataSet, pero en este ejemplo, y al tratarse de una única tabla, vamos a realizar el *Commit* (de hecho será un *CommitRetaining*) en el evento *AfterPost* del TIBDataset.

Será algo tan sencillo como el siguiente trozo de código:

```
procedure TMainDM.IBDataSet1AfterPost(DataSet: TDataSet);
begin
  IBTransaction1.CommitRetaining;
end;
```

Y por supuesto devemos acordarnos de activar el TIBDataset para poder añadir información a la tabla. Para ello vamos a usar el evento *OnShow* del formulario donde también podemos aprovechar e inicializar el TDateTimePicker:

```
procedure TMainFrm.FormShow(Sender: TObject);
begin
  MainDM.IBDataSet1.Open;
  eFecha.DateTime := Now;

  // llamamos al evento OnChange del TDateTimePicker
  if Assigned(eFecha.OnChange) then
    eFechaChange(Sender);
end;
```

Esto..... ¿y para qué queremos el TDateTimePicker? Bueno, la idea es que este componente nos ayude a filtrar la tabla según el día escogido. Perfecto, ¿pero cómo lo hacemos? Formas hay muchas, pero a mi me gusta usar la programación orientada a objetos (ya que el lenguaje nos lo permite). Para ello vamos a añadir una propiedad al módulo de datos que sea de tipo *Date*:

```
  .....
  private
    FFecha: TDate;
    procedure SetFecha(const Value: TDate);
  public
    .....
    property Fecha: TDate read FFecha write SetFecha;
  end;
```

Aprovecharemos el *Set* de la propiedad para realizar el filtrado de la tabla:

```
procedure TMainDM.SetFecha(const Value: TDate);
begin
  if FFecha <> Value then
  begin
    FFecha := Value;
    IBDataSet1.Close;
      IBDataSet1.SelectSQL.Text := 'select * from AGENDA where cast(fecha as
date) = ' +
        QuotedStr(FormatDateTime('mm/dd/yyyy', Value));
    IBDataSet1.Open;
  end;
end;
```

Y ahora sólo nos quedaría por programar el OnChange del TDateTimePicker para que actualice el valor de la propiedad:

```
procedure TMainFrm.eFechaChange(Sender: TObject);
begin
  MainDM.Fecha := eFecha.Date;
end;
```

Ahora que sabemos la fecha, podemos facilitar aun más el trabajo al usuario poniendo en el *OnNewRecord* del TIBDataset la inicialización de esta.

```
procedure TMainDM.IBDataSet1NewRecord(DataSet: TDataSet);
begin
  IBDataSet1FECHA.AsDateTime := Trunc(FFecha) + Frac(Time);
end;
```

Pues ya tenemos nuestra agenda en marcha y el resultado.....

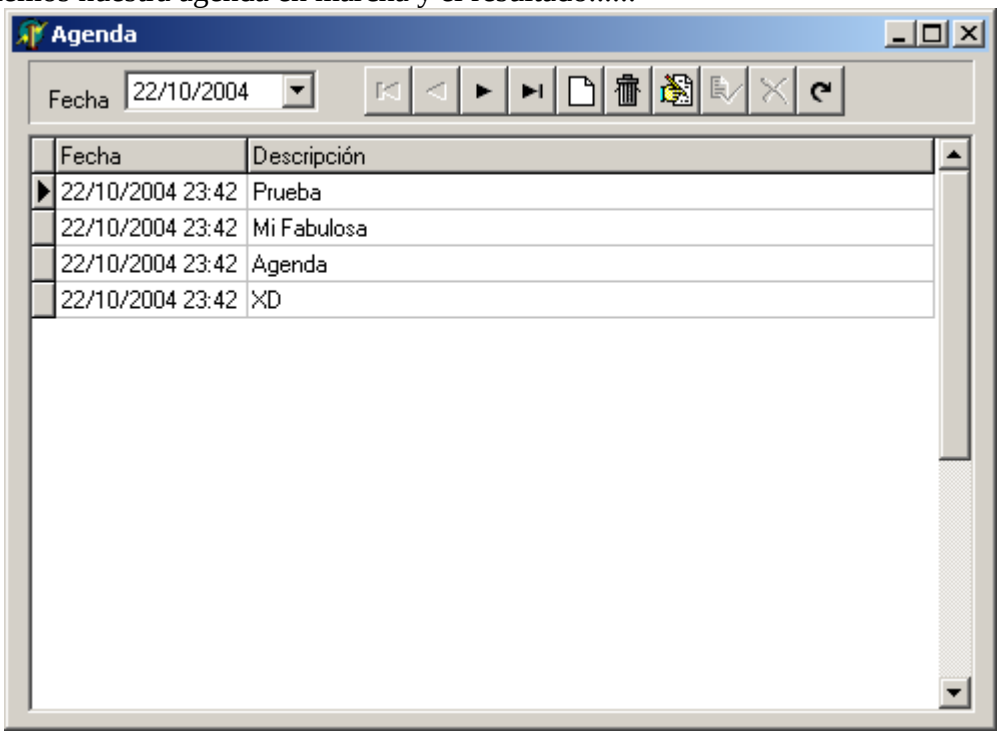


Figura 12

7. Para terminar

Para terminar quería agrupar aquí todos los enlaces que han ido apareciendo a lo largo de este artículo y añadir algún otro que creo puede ser interesante.

Webs sobre el motor:

- Firebird en SourceForge: http://firebird.sourceforge.net/
- IBPhoenix: http://www.ibphoenix.com/
- Release Notes en Español: http://www.ibphoenix.com/downloads/Firebird_v15.108_ReleaseNotesSpanish.pdf
- Firebird en Español: http://www.firebird.com.mx/modules/news/

Webs de los componentes de acceso:

- InterBase eXpress (IBX): http://codecentral.borland.com/codecentral/ccweb.exe/author?authorid=102
- InterBase Objects (IBO): http://www.ibobjects.com/
- FreeIBPlus: http://www.devrace.com/en/fibplus/?r1=fibplus&r2=Direct_link
- Unified Interbase (UIB): http://www.progdigy.com/
- Zeos Lib: http://www.zeoslib.net/

Libros o artículos sobre el motor:

- De momento (y que yo sepa) sólo existe un libro que hable sobre este motor y es "*The Firebird Book: a Reference for Database Developers*" de **Helen Borrie**.
  Podéis conseguirlo en Amazon o bien en IBPhoenix (comparar precios antes de comprar).
  http://www.amazon.fr/exec/obidos/ASIN/1590592794/qid=1096739340/171-0019294-0983439
  http://www.ibphoenix.com/main.nfs?a=ibphoenix&page=ibp_firebird_book
- Artículo sobre transacciones de Juan José Rodríguez.
  http://www.terra.es/personal2/jrodriguezpe/kinobi/descargas/transacciones-0.2.6.pdf
- Documentación de Interbase 6: http://www.ibphoenix.com/downloads/60All.zip

Foros:

- ClubDevelopers, por supuesto: http://www.clubdevelopers.com/foros/index.php?board=29.0
- ClubDelphi: http://www.clubdelphi.com/foros/forumdisplay.php?f=19

GUI

- IB-Expert: http://www.ibexpert.com/
- IB Manager: http://www.ems-hitech.com/ibmanager/
- Database WorkBench: http://www.upscene.com/index.htm?./products/dbw/index.htm
- IBO-Console (clon del IBConsole realizado con las IBO): http://www.mengoni.it/downloads.html
- FlameRobin o FBManager: http://fbmanager.sourceforge.net/