

Enunciado:

Tema: Bases de datos y ADO.

Desarrollo de una aplicación biblioteca con préstamo de libros a sus lectores.

La **Figura 1**, muestra el DER (**D**iagrama de **I**ntegridad **R**referencial) de una base de datos del tipo **Microsoft Access** de la aplicación, este será como el cimiento de la misma y sobre él se construirá el resto del edificio, la aplicación, por este motivo es importante comprender el modelo o plano descripto en la **Figura 1**.

Notemos que al igual que en la ingeniería civil, un cambio en los cimientos tendrá un impacto muy grande sobre el edificio, análogamente un cambio en el DER de la **Figura 1**, tendrá un gran impacto en nuestra aplicación, solo imaginemos que pasaría si desapareciera una tabla o un campo o cambiase el tipo de relación entre las tablas o la clave natural de las mismas.

En el DER podemos ver dos tablas, “**Libro**” y “**Lector**” que conformaran la aplicación de **biblioteca** que desarrollaremos, notemos que un lector puede poseer cero o más libros en calidad de préstamo, y que un libro puede estar prestado o no, pero en caso de estarlo solo lo estará a un solo lector, esto se ve en la relación de integridad que relaciona ambas tablas.

Otra distinción interesante a destacar es la diferencia entre una **clave natural** y **clave sustituta**, los campos “**Libro.id_libro**” y “**Lector.id_lector**” conforman la clave sustituta y por tanto están despegados del negocio, son una especie de punteros a registro e identifican inequívocamente a uno, en otras palabras dado un valor de “**Libro.id_libro**” solo existirá uno que lo posea y acompañara al mismo en forma constante durante todo el tiempo de vida del libro o registro de la tabla.

La **clave natural**, a diferencia de la sustituta, puede cambiar a lo largo del tiempo y está relacionada al negocio y también identifica en forma inequívoca a un registro, en el DER podemos ver el campo “**Lector.dni**” y “**Libro.codigo_identificacion_unico**”, los valores de las claves naturales no tienen por qué ser constantes y pueden cambiar en el tiempo, recordemos por ejemplo un caso real y conocido, los cambios que han sufrido las matriculas de los coches, pasaron de ser 6 números a ser tres letras y tres números para finalmente pasar a ser dos letras, tres números y dos letras, como podemos ver en la **Figura 0**.

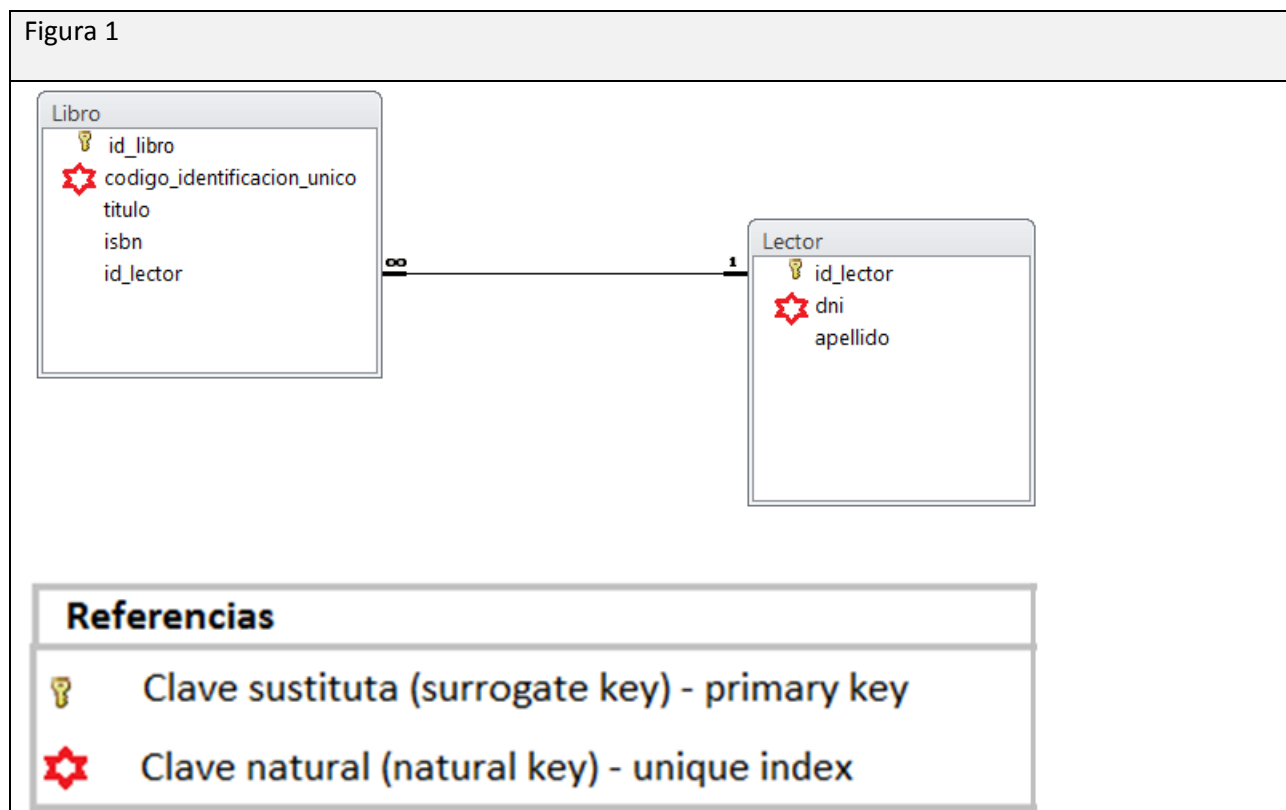
Entonces cabe una pregunta que deberá desvelar al intrépido lector:

¿Por qué no usamos claves naturales y eliminamos las claves sustitutas, simplificando de este modo el DER y ahorrando espacio en disco?

Figura 0



Figura 1

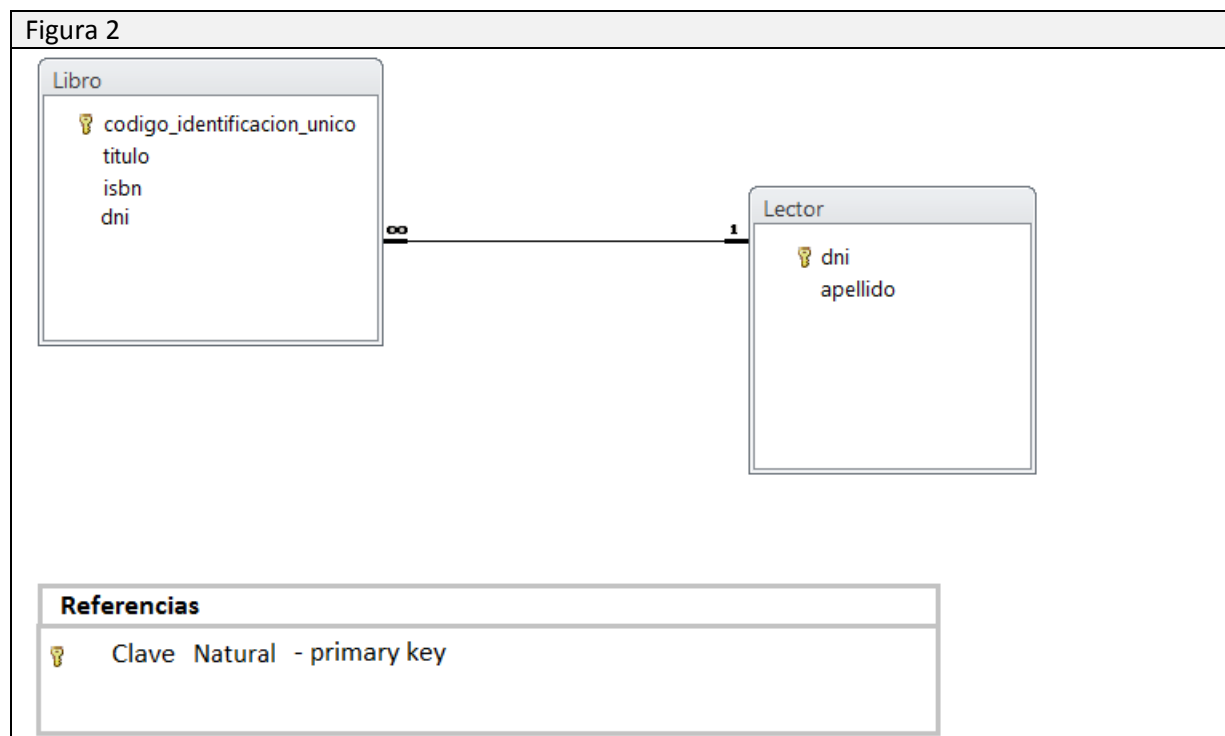


Una gran ventaja de las claves sustitutas es que no tienen relación con el negocio, es más, el usuario ni siquiera las conoce y ni siquiera las ha pedido, razón por la cual cambios en el negocio tendrán cero impacto sobre dichas claves y por ende el DER (cimiento de la aplicación) cambiara solo en el resto de la estructura dejando las relaciones entre tablas intactas al igual que las modificaciones de los valores de las claves sustitutas.

Comparando el DER de la **Figura 1** versus el de la **Figura 2**,

¿Cuál sufriría menos impacto ante un cambio en el valor o en el formato del campo “**Lector.dni**”?

Supongamos un DER imaginario en el cual la tabla “**Lector**” este relacionada con otras 100 tablas, una estructura como la de la **Figura 1**, sufriría menos impacto ante un cambio en el valor o en el formato del campo “**Lector.dni**” en comparación con la **Figura 2**.



Le archivo “**BibliotecaDB.mdb**” contiene la base de datos mencionada y se encuentra en el directorio “**Database**”.

La **Figura 3**, muestra los módulos que conforman la aplicación, una descripción de cada uno podemos encontrarla en la **Figura 4**.

Figura 3

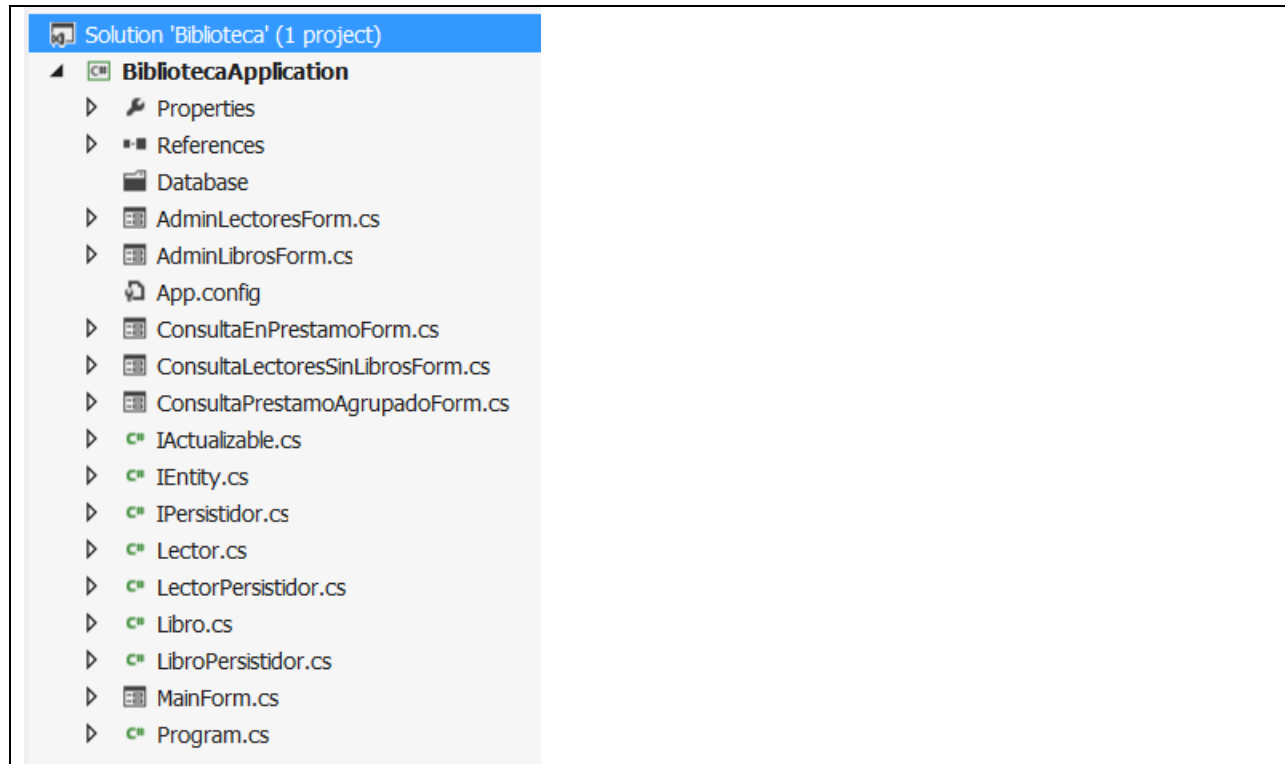


Figura 4	
Database	Directorio en el que se aloja la base de datos "BibliotecaDB.mdb"
AdminLectoresForm.cs	Formulario de administración de lectores.
AdminLibrosForm.cs	Formulario de administración de libros.
App.config	Contiene información de configuración, como el "connectionString" o cadena de conexión a la base de datos.
ConsultaEnPrestamoForm.cs	Formulario que muestra los libros que están en préstamo.
ConsultaLectoresSinLibrosForm.cs	Formulario que muestra a aquellos lectores que no poseen libros.
ConsultaPrestamoAgrupadoForm.cs	Formulario que muestra la cantidad de libros que posee cada uno de los lectores.
Lector.cs	Entidad lector, representación en objetos de un lector de la biblioteca.
Libro.cs	Entidad libro, representación en objetos de un libro de la biblioteca.
LectorPersistidor.cs	Su responsabilidad es persistir un lector, más conocida como Aata Access Layer (DAL).
LibroPersistidor.cs	Su responsabilidad es persistir un libro, más conocida como Aata Access Layer (DAL).
MainForm.cs	Formulario principal o contenedor padre del resto de los formularios, es un formulario MDI

	(Multiple Document Interface).
IEntity.cs	Obliga a todas la entidades a implementar una propiedad Id, que la identifica inequívocamente.
IActualizable.cs	Obliga a las clases de consulta a implementarla para conseguir el refresco automático de todas ellas.
IPersistidor.cs	Es implementada por todos los persistidores, para lograr una funcionalidad común, básicamente el CRUD (C reate, R ead, U psert and D elete).

Las Figuras 5 y 6, muestran los formularios “AdminLectoresForm.cs” y “AdminLibrosForm.cs” respectivamente, su responsabilidad es administrar los lectores y los libros que pueden ser prestados.

Figura 5

DNI	Apellido
222	Pepe
555555	Juan
555555	Adolfo

Lector:

DNI: 222

Apellido: Pepe

Save Delete List Clean

Figura 6

CodigoIdentific	Titulo	ISBN	Prestado
22	El túnel	978843762...	<input checked="" type="checkbox"/>
444	Martin Fierro	978848916...	<input type="checkbox"/>
43	Rayuela	978843762...	<input checked="" type="checkbox"/>
55555	Rayuela	978843762...	<input checked="" type="checkbox"/>
22222222	El túnel	978843762...	<input checked="" type="checkbox"/>

Libro:

CodigoIdUn: 55555

Titulo: Rayuela

ISBN: 9788437624747

Lector: Adolfo

Save Delete List Clean

Restricción ante la eliminación de un lector.

Es interesante destacar que si se intenta eliminar un lector que posee un libro prestado la aplicación lanzara una excepción, impidiendo dicha operación, en otras palabras solo se podrá eliminar al lector que haya devuelto todos los libros que la biblioteca le ha prestado. Es bueno destacar que la validación de la restricción antes mencionada no está definida en el código de la aplicación, está definida en la base de datos, mas precisamente en la relación de integridad referencial existente entre las tablas, tal como se ve en la **Figura 1**.

Restricción ante la eliminación de un libro.

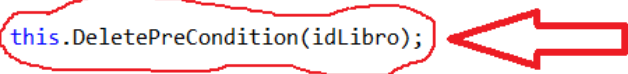
Si se intenta eliminar un libro que esta prestado a un lector, la aplicación lanzara una excepción impidiendo la eliminación del libro, ya que si se lograra eliminar nadie de la biblioteca le reclamaría al lector que lo regrese. Esta restricción esta validada en el código fuente de la aplicación como una precondition, ver línea 198 en la **Figura 7**. En el caso de que la precondition falle, se lanza un excepción personalizada (custom) llamada “*EliminarLibroPrestadoException*” ver línea 222.

Figura 7

```

196 public void Delete(int idLibro)
197 {
198     this.DeletePreCondition(idLibro);
199
200     string connectionString = ConfigurationManager.ConnectionStrings["cnnString"].ToString();
201     using (OleDbConnection cnx = new OleDbConnection(connectionString))
202     {
203         cnx.Open();
204         const string sqlQuery = "DELETE FROM Libro WHERE id_libro = @id_libro";
205         using (OleDbCommand cmd = new OleDbCommand(sqlQuery, cnx))
206         {
207             cmd.Parameters.AddWithValue("@id_libro", idLibro);
208
209             cmd.ExecuteNonQuery();
210         }
211     }
212 }
213
214 /// <summary>
215 /// Evalúa la restricción que impide eliminar un libro que este prestado.
216 /// </summary>
217 /// <param name="idLibro"></param>
218 private void DeletePreCondition(int idLibro)
219 {
220     Libro libro = this.GetByid(idLibro);
221     if (libro.Prestado)
222         throw new EliminarLibroPrestadoException(libro);
223 }

```



La **Figura 8**, muestra cómo se intenta eliminar un libro (70) y como en caso de no poder lograrse se atrapa la excepción *EliminarLibroPrestadoException*, y se transforma la misma en un mensaje amigable para el usuario.

Figura 8

```

66 private void buttonDelete_Click(object sender, EventArgs e)
67 {
68     try
69     {
70         this.Delete();
71         this.ReLoadGrid();
72         this.CleanUI();
73     }
74     catch (LibroPersistidor.EliminarLibroPrestadoException exception)
75     {
76         this.MostrarMensaje(exception);
77     }
78 }

```

La **Figura 9**, muestra un mensaje al usuario indicándole que el libro no se puede eliminar y quien es el lector que lo tiene en préstamo, observemos que el valor del **IdLector** (87) es una propiedad de la propia excepción.

Figura 9

```

84 private void MostrarMensaje(LibroPersistidor.EliminarLibroPrestadoException exception)
85 {
86     LectorPersistidor lectorPersistidor = new LectorPersistidor();
87     Lector lector = lectorPersistidor.GetByid(exception.Libro.IdLector.Value);
88     string mensaje = "No puede eliminar el libro ya que ha sido prestado al lector: {0} DNI: {1}.";
89     mensaje = string.Format(mensaje, lector.Apellido, lector.DNI);
90     MessageBox.Show(mensaje);
91 }

```