El editor de DataTables (VI). Tablas con relaciones m-n (editar)

Cuando trabajamos con bases de datos con relaciones m-n entre tablas, la edición de registros, así como, en general, cualquier proceso que se haga sobre los mismos, debe llevarse a cabo teniendo en cuenta el modelado de datos. Una relación m-n implica el uso de una tabla intermedia donde se almacenan las relaciones. En el anterior artículo aprendimos sobre este tipo de escenarios, viendo como recuperar los datos y renderizarlos en la página.
En este artículo vamos a completar el proceso, viendo cómo usar el editor de DataTables de un modo que podamos gestionar este modelado de datos lo más eficientemente posible.

EL SCRIPT PRIMARIO
Este script es el primero en el que vamos a conocer algunas novedades. Realmente, la diferencia entre el script primario de este artículo y el del anterior es que, en este, hemos incorporado el editor, que ya conocemos. Sin embargo, esta vez el editor nos va a enseñar algunos conceptos o detalles, nuevos, que nos resultarán muy útiles. El script se llama articulo_editor_05_2.php, y su listado es el siguiente:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <meta http-equiv="X-UA-Compatible" content="IE=edge">
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <title>Uso de DataTables</title>
 <!-- El CSS de jQuery UI -->
 <link rel="stylesheet" type="text/css" href="http://code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">
 <!-- El CSS de DataTables -->
 <link rel="stylesheet" type="text/css"
href="https://cdn.datatables.net/v/dt/jszip-2.5.0/pdfmake-0.1.18/dt-1.10.13/af-2.1.3/b-1.2.4/b-colvis-1.2.4/b-flash-1.2.4/b-html5-1.2.
4/b-print-1.2.4/cr-1.3.2/fc-3.2.2/fh-3.1.2/kt-2.2.0/r-2.1.0/rr-1.2.0/sc-1.4.2/se-1.2.0/datatables.min.css"/>
 <!-- El CSS de Bootstrap -->
 <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
 <!-- El CSS de font awesome -->
 <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">
 <!-- El CSS del editor -->
 <link rel="stylesheet" type="text/css" href="editor/css/editor.dataTables.css">
</head>
<body>
 <br>
 <div class="container">
  <div class="row">
   <div class="col-sm-12">
    <table id="tabla_de_personal" class="table display table-striped table-bordered">
     <thead>
      <tr>
```

```
      <th> </th>
      <th>NOMBRE</th>
      <th>APELLIDOS</th>
      <th>F. INC.</th>
      <th id="cabeceraDeSalario">SALARIO</th>
     </tr>
    </thead>
    <tbody>
    </tbody>
   </table>
  </div>
 </div>
</div>

<!-- jQuery -->
<script language="javascript" src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
<!-- jQuery UI -->
<script language="javascript" src="http://code.jquery.com/ui/1.12.1/jquery-ui.min.js"></script>
<!-- El JavaScript de DataTables -->
<script type="text/javascript"
src="https://cdn.datatables.net/v/dt/jszip-2.5.0/pdfmake-0.1.18/dt-1.10.13/af-2.1.3/b-1.2.4/b-colvis-1.2.4/b-flash-1.2.4/b-html5-1.2.4/
b-print-1.2.4/cr-1.3.2/fc-3.2.2/fh-3.1.2/kt-2.2.0/r-2.1.0/rr-1.2.0/sc-1.4.2/se-1.2.0/datatables.min.js"></script>
<!-- El JavaScript de BootStrap -->
<script language="javascript" src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
<!-- El JavaScript del editor -->
<script type="text/javascript" src="editor/js/dataTables.editor.js"></script>
<!-- La librería moment.js para fechas -->
<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.13.0/moment.min.js"></script>

<!-- El código JavaScript -->
<script language="javascript">
 /* Definimos una variable en la que se almacenará la lista de especialidades,
 para construir el select del formulario de nuevo o edición. Esta variable se recreará
 por Ajax cada vez que se invoque una operación de creación o edición de un registro.
 De este modo, siempre tendrá la lista actualizada, si, por ejemplo, otro usuario
 modificase la tabla de dichas especialidades. */
 var listaDeEspecialidades = '';

 /* CREAMOS EL OBJETO EDITOR, PARA LAS FUNCIONES DE NUEVO, EDICIÓN Y ELIMINACIÓN DE REGISTROS. */
 var objetoEditor = new $.fn.dataTable.Editor({
  ajax: 'crud_editor_05_2.php', // El script que crea, actualiza o borra registros.
  table: '#tabla_de_personal', // La tabla HTML sobre la que se actúa.
  idSrc: 'id', // El nombre del campo clave primaria de la tabla de base de datos (desarrolladores) sobre la que se actúa.
  /* Las traducciones para los botones, los valores múltiples,
  el campo de fecha y la confirmación de eliminación. */
  i18n: {
   create: {
    button: "Nuevo",
    title:  "Crear nuevo registro",
    submit: "Grabar"
```

```
    },
    edit: {
     button: "Editar",
     title:  "Editar registro",
     submit: "Actualizar"
    },
    remove: {
     button: "Borrar",
     title:  "Borrar registro",
     submit: "Borrar",
     confirm: {
      _: "¿Estás seguro de eliminar estos %d registros?",
      1: "¿Estás seguro de eliminar este registro?"
     }
    },
    multi: {
     title: "Múltiples valores",
     info: "Los registros seleccionados contienen diversos valores para este campo. Para editar este campo con el mismo valor en los registros seleccionados, pulsa aquí. En caso conterario, los registros mantendrán sus valores individuales en este campo.",
     restore: "Restaurar los valores múltiples",
     noMulti: "Esta entrada puede ser modificada individualmente, pero no como parte de un grupo."
    },
    datetime: {
     previous: 'Anterior',
     next:  'Siguiente',
     months:  ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio', 'Julio', 'Agosto', 'Septiembre', 'Octubre', 'Noviembre', 'Diciembre'],
     weekdays: ['Dom', 'Lun', 'Mar', 'Mié', 'Jue', 'Vie', 'Sáb']
    }
   },
   /* La difinición de los campos del formulario. */
   fields: [
    /* ATENCIÓN.
    /* Los campos de tipo text se pueden definir sin especificar el tipo.
    El valor text es el tipo de campo por defecto. */
    {label: 'Nombre:', name: 'nombre', attr: {class:'form-control'}},
    {label: 'Apellidos:', name: 'apellidos', attr: {class:'form-control'}},
    /* Para el campo de tipo select (el combo), es necesario establecer el
    atributo type, con el valor 'select'. Además, en general, se debería definir
    aquí el atributo options, con las opciones. Sin embargo, en este ejemplo, no se
    definen estas, porque se definirán de forma dinámica, ya que otro usuario podría
    modificar las listas de ciudades o cargos.
    Observa también que, como un desarrollador puede tener varias especialidades,
    este campo lo hemos definido con el atributo multiple, con el valor true.
    Además, dentro de attr, hemos especificado el tamaño con la propiedad size.
    Es importante recordar que, cuando se define un campo select con el atriibuto
    multiple activado, el valor que se pasa es una matriz con los elementos seleccionados.
    Esta matriz se genera inherentemente a la naturaleza de estos campos en el editor,
    por lo que no es necesario crearla a partir de los valores que se hayan seleccionado,
    como sí sería necesario si estuviéramos usando HTML "a pelo".*/
    {
```

```
    label: 'Especialidades:',
    name: 'id_especialidades',
    attr: {class:'form-control', size:'10'},
    type: 'select',
    fieldInfo: 'Selecciona las especialidades',
    multiple: true
   },
   /* El campo de fecha con el datepicker jQueryUI, por funcionalidad. */
   {
    label: 'F. Ingreso:',
    name: 'fecha_incorporacion',
    type: 'date',
    def: function(){return new Date();},
    dateFormat: 'dd-mm-yy',
    attr: {readonly:true, class:'form-control', style:'display:inline'},
    opts:{
     buttonImage:'editor/images/calendar.png',
     buttonImageOnly: true,
     buttonText: 'Elegir fecha',
     dayNames: ['Domingo', 'Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes', 'Sábado'],
     dayNamesMin: ['Do', 'Lu', 'Ma', 'Mi', 'Ju', 'Vi', 'Sá'],
     dayNamesShort: ['Dom', 'Lun', 'Mar', 'Mié', 'Jue', 'Vie', 'Sáb'],
     firstDay: 1,
     monthNames: ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio', 'Julio', 'Agosto', 'Septiembre', 'Octubre', 'Noviembre',
'Diciembre'],
     monthNamesShort: ['Ene', 'Feb', 'Mar', 'Abr', 'May', 'Jun', 'Jul', 'Ago', 'Sep', 'Oct', 'Nov', 'Dic'],
     changeMonth: true,
     changeYear: true,
     prevText: "Anterior",
     nextText: "Siguiente"
    }
   },
   {
    label: 'Salario anual:',
    name: 'salario_bruto_anual',
    attr: {class:'form-control'},
    fieldInfo: 'El salario debe ir como valor numérico, con dos decimales (incluso si son 00), separados por un punto.'
   },
  ]
 });
 /* FINAL DE LA DEFINICIÓN DE UN OBJETO EDITOR. */

 /* Cuando se pulsa un botón de creación, edición o eliminación, y antes de que
 el editor abra el correspondiente formulario, se dispara el evento preOpen.
 Aquí lo usamos para recuperar, mediante un ajax síncrono, la lista de especialidades
 actualizada, incorporándolas al correspondiente campo de tipo select.
 Observa que esto sólo se hace si la operación es una edición, o una creación.
 Si se trata de una eliminación, no es necesario, ya que no se muestran campos
 en el formulario. */
 objetoEditor.on('preOpen', function(e, mode, action){
```

```
 if (action == "remove") return;
 $.ajax({
  url:"leer_especialidades_editor_05_2.php",
  async:false,
  dataType: "JSON",
  complete:function(datosRecibidos) {
   listaDeEspecialidades = datosRecibidos.responseText;
  }
 });
 objetoEditor.field('id_especialidades').update(JSON.parse(listaDeEspecialidades));
});


/* Cuando se pulsa el botón de enviar un formulario, y antes de que este sea, efectivamente,
enviado (al script definido en el atributo ajax del objeto editor), se dispara el evento
preSubmit. Nosotros lo utilizamos para comprobar los campos de texto del formulario, donde,
a priori, el usuario puede teclear cualquier cosa, o ninguna. Si algún campo de texto
está vacío, le mostramos un mensaje en rojo debajo de dicho campo, mediante el uso de la
propiedad error (nativa de los campos de formulario) y retornamos un valor false, lo que
aborta el envío.
Como ves, si la acción pedida es la eliminación de un registro, se abandona la prevalidación,
ya que no es necesaria. */
objetoEditor.on('preSubmit', function(e, data, action){
 if (action == 'remove') return true;

 // Se comprueba que el nombre tenga contenido, si no es multiedición.
 var dato_nombre = objetoEditor.field('nombre').val();
 if (!objetoEditor.field('nombre').isMultiValue()) dato_nombre = dato_nombre.trim();
 if (dato_nombre == "" && !objetoEditor.field('nombre').isMultiValue()){
  objetoEditor.field('nombre').error('Debes teclear el nombre.');
  objetoEditor.field('nombre').focus();
  return false;
 } else {
  objetoEditor.field('nombre').error('');
 }

 // Se comprueba que los apellidos tengan contenido, si no es multiedición.
 var dato_apellidos = objetoEditor.field('apellidos').val();
 if (!objetoEditor.field('apellidos').isMultiValue()) dato_apellidos = dato_apellidos.trim();
 if (dato_apellidos == "" && !objetoEditor.field('apellidos').isMultiValue()){
  objetoEditor.field('apellidos').error('Debes teclear los apellidos.');
  objetoEditor.field('apellidos').focus();
  return false;
 } else {
  objetoEditor.field('apellidos').error('');
 }

 // Se comprueba que el salario tenga contenido, si no es multiedición. Además, se comprueba que se ajuste a formato.
 var dato_salario_bruto_anual = objetoEditor.field('salario_bruto_anual').val();
 if (!objetoEditor.field('salario_bruto_anual').isMultiValue()) dato_salario_bruto_anual = dato_salario_bruto_anual.trim();
```

```
  if (dato_salario_bruto_anual == "" && !objetoEditor.field('salario_bruto_anual').isMultiValue()){
   objetoEditor.field('salario_bruto_anual').error('Debes teclear el salario bruto anual.');
   objetoEditor.field('salario_bruto_anual').focus();
   return false;
  } else {
   objetoEditor.field('salario_bruto_anual').error('');
  }
  if (dato_salario_bruto_anual != parseFloat(dato_salario_bruto_anual).toFixed(2) &&
!objetoEditor.field('salario_bruto_anual').isMultiValue()){
   objetoEditor.field('salario_bruto_anual').error('El salario bruto anual no tiene el formato correcto.');
   objetoEditor.field('salario_bruto_anual').focus();
   return false;
  } else {
   objetoEditor.field('salario_bruto_anual').error('');
  }

  return true; // Si todo ha ido bien, se retorna true para que se complete el envío.
 });

 /* SE DEFINE EL OBJETO DATATABLES. SU USO ES, PRÁCTICAMENTE,
 EL MISMO QUE SIEMPRE, PARA RECUPERAR UN DATASET Y RENDERIZARLO EN LA PÁGINA. */
 var objetoDataTables_personal = $('#tabla_de_personal').DataTable({
  "language": {
   select: {
    rows: {
     _: "%d registros seleccionados",
     0: "No se han seleccionado registros",
     1: "1 registro seleccionado"
    }
   },
   "emptyTable":   "No hay datos disponibles en la tabla.",
   "info":        "Del _START_ al _END_ de _TOTAL_ ",
   "infoEmpty":   "Mostrando 0 registros de un total de 0.",
   "infoFiltered":   "(filtrados de un total de _MAX_ registros)",
   "infoPostFix":   "(actualizados)",
   "lengthMenu":   "Mostrar _MENU_ registros",
   "loadingRecords":  "Cargando...",
   "processing":   "Procesando...",
   "search":    "Buscar:",
   "searchPlaceholder": "Dato para buscar",
   "zeroRecords":   "No se han encontrado coincidencias.",
   "paginate": {
    "first":   "Primera",
    "last":    "Última",
    "next":    "Siguiente",
    "previous":   "Anterior"
   },
   "aria": {
    "sortAscending": "Ordenación ascendente",
    "sortDescending": "Ordenación descendente"
```

```
 }
},
"lengthMenu":    [[5, 10, 20, 25, 50, -1], [5, 10, 20, 25, 50, "Todos"]],
"iDisplayLength":   10,
"bServerSide":    true,
"sAjaxSource":    "datos_externos_editor_05_2.php", // OJO. Ver el código para comprender.

/* Observa que los datos para las columnas se definen precediéndolos del nombre de la
tabla de la que proceden, ya que estamos trabajando con tres tablas, y el JSON retornado
tiene datos de las tres.
Observa que el campo ciudad se toma de la tabla de ciudades, ya que, en la tabla de personal,
lo que hay es el campo id_ciudad, que contiene una clave foránea.
Sin embargo, en el objeto editor se referencia el campo como personal.ciudad, porque se toma
el literal de ciudad, pero se está trabajando con la tabla de personal.
Con los cargos se aplica la misma operativa. */
"columns" : [
 {
  "className":  'celda_de_descripcion',
  "orderable":  false,
  "data":    null,
  "defaultContent": '<div class="text-center fa fa-plus-circle" style="width:100%; color: #3dc728;"></div>'
 },
 {"data": 'nombre'},
 {"data": 'apellidos'},
 {"data": 'fecha_incorporacion'},
 {"data": 'salario_bruto_anual', className: "text-right"}
],
"order": [[1, "asc"]],
"columnDefs": [
 {
  "targets": 4,
  "render": function (data) {
   return salaryFormat(data);
  }
 }
],
dom:      'Bfrtipl',
select:      true,
select: {
 // Con la siguiente línea conseguimos que no se puedan seleccionar múltiples registros.
 style: 'single',
 // Impedimos que la celda izquierda (la de desplegar más datos), sirva para seleccionar o deseleccionar filas.
 selector: 'td:not(:first-child)'
},
buttons: [
 {extend: 'create', editor: objetoEditor},
 {extend: 'edit', editor: objetoEditor},
 {extend: 'remove', editor: objetoEditor}
]
});
```

```
  $('label').addClass('form-inline');
  $('select, input[type="search"]').addClass('form-control input-sm');
  $('#cabeceraDeSalario').removeClass("text-right");

  $('#tabla_de_personal tbody').on('click', 'td.celda_de_descripcion', function () {
   var filaDeLaTabla = $(this).closest('tr');
   var filaComplementaria = objetoDataTables_personal.row(filaDeLaTabla);
   var celdaDeIcono = $(this).closest('td.celda_de_descripcion');

   if (filaComplementaria.child.isShown() ) { // La fila complementaria está abierta y se cierra.
    filaComplementaria.child.hide();
    celdaDeIcono.html('<div class="text-center fa fa-plus-circle" style="width:100%; color: #3dc728;"></div>');
   } else { // La fila complementaria está cerrada y se abre.
    filaComplementaria.child(formatearSalidaDeDatosComplementarios(filaComplementaria.data(), "lista_especialidades")).show();
    celdaDeIcono.html('<div class="text-center fa fa-minus-circle" style="width:100%; color: #e80909;"></div>');
   }
  });

  function formatearSalidaDeDatosComplementarios (filaDelDataSet, columna) {
   var cadenaDeRetorno = '';
   cadenaDeRetorno += '<div>';
   cadenaDeRetorno += 'ESPECIALIDADES: ' + filaDelDataSet[columna];
   cadenaDeRetorno += '</div>';
   return cadenaDeRetorno;
  }

  function salaryFormat(valorRecibido) {
   var separado = valorRecibido.split(".");
   var parteEntera = separado[0];
   var unidades = new Array();
   var fragmento;
   while (parteEntera > ""){
    fragmento = parteEntera.substr(-3, 3);
    parteEntera = parteEntera.substr(0, parteEntera.length - 3);
    unidades.unshift(fragmento);
   }
   var valorFinal = unidades.join(".");
   if (separado.length > 1) valorFinal += ("," + separado[1]);
   valorFinal += " ?";
   return valorFinal;
  }
 </script>
</body>
</html>
```

Vamos a ver las novedades, aunque, salvo que fuera necesario, no insitiremos en temas que ya se hayan tratado en anteriores artículos. En primer lugar fíjate en el primer grupo de líneas seleccionadas, entre la 123 y la 130. En ellas ves cómo definimos un campo select de múltiples opciones. Por lo demás, las opciones del combo se cargan dinámicamente por ajax, como respuesta al evento preOpen, como ya hemos hecho en otro artículo anterior.

Ten en cuenta que cuando se crea un campo combo múltiple, el editor lo gestiona de una forma muy eficiente. Si estas familiarizado con lo que eran estos combos tradicionalmente en HTML, el value sólo contenía la última opción marcada, y había que recuperar

todas las opciones marcadas con JS. El editor de DataTables hace esto de forma transparente, almacenando todos los valores de las opciones marcadas en una matriz directamente. Lo verás claro cuando hablemos del script que procesa el formulario.
En la línea 312 vemos una novedad. Hemos hecho que el usuario sólo pueda seleccionar un registro para editar o borrar. Con esta línea evitamos las ediciones y borrados múltiples, y reducimos los errores causados por el usuario.
En la línea 314 vemos como hacer que la columna de la izquierda no sirva para seleccionar o desseleccionar una fila. Como la columna de la izquierda ya se usa para desplegar los datos adicionales, no deberíamos permitir que, además, dispare el selector. De este modo lo evitamos.

EL SCRIPT DE RECUPERACIÓN DEL DATASET

Este no presenta ya nada nuevo. Es idéntico al que vimos en el artículo anterior, y funciona del mismo modo. Lo hemos llamado datos_externos_editor_05_2.php.

EL SCRIPT DE RECUPERACIÓN DE ESPECIALIDADES

Este script, que hemos llamado leer_especialidades_editor_05_2.php, es similar a los que usamos en un artículo anterior para obtener dinámicamente las opciones de dos combos, de cargos y ciudades. Por lo tanto, tampoco nos demoraremos en él.

EL SCRIPT DE RESPUESTA AL EDITOR

Este script si presenta algunos puntos interesantes que quiero comentar contigo. El listado es crud_editor_05_2.php:

```
<?php
 // Establecemos la codificacion para las llamadas y respuestas HTTP
 mb_internal_encoding ('UTF-8');

 /* CREAMOS LA CONEXION A LA BASE DE DATOS, O BIEN LA IMPORTAMOS
 DESDE UN ARCHIVO EXTERNO DE CONFIGURACION. */
 include ('conexion_bd_editor.php');

 /* RECUPERAMOS TODOS LOS PARAMETROS DE $_POST. LOS QUE NO APAREZCAN EN LA CONSULTA
 RECIBIRAN UN VALOR PREDETERMINADO. ESTOS DATOS SON LOS QUE SE RECIBEN CADA VEZ QUE EL
 EDITOR DE DATATABLES LLAMA A ESTE SCRIPT. */
 $datosDeLlamada = $_POST;

 /* SE INDICA(N) LA(S) TABLA(S) QUE SE VA(N) A USAR EN LA CONSULTA. */
 $tablasDeBBDD = array(
  'desarrolladores',
  'relaciones_desarrolladores_especialidades'
 );

 /* SE DEFINE LA LISTA DE COLUMNAS QUE SE DEVOLVERON PARA SER MOSTRADAS EN
 LA TABLA HTML.
 LOS NOMBRES DE ESTAS COLUMNAS DEBEN COINCIDIR CON LOS DE LAS COLUMNAS DE
 LA(S) TABLA(S) AFECTADA(S) POR LA CONSULTA. */
 $columnasDeDatos = array(
  $tablasDeBBDD[0].'.nombre',
  $tablasDeBBDD[0].'.apellidos',
  $tablasDeBBDD[0].'.fecha_incorporacion',
  $tablasDeBBDD[0].'.salario_bruto_anual',
  $tablasDeBBDD[0].'.id',
  $tablasDeBBDD[1].'.id_desarrollador',
  $tablasDeBBDD[1].'.id_especialidad'
 );

 /* Se determina qué es lo que se va a hacer. Esto viene por POST en la variable "action".
 Puede ser remove, edit o create.
```

```
En cada caso se llevara a cabo la acción elegida, sobre los registros que estuvieran seleccionados.
Además, se prepara la respuesta adecuada para devolver al script primario. */
$data = array();

switch ($datosDeLlamada["action"]){
 case "remove":
  /* Si es una eliminación debemos borrar el desarrollador en la tabla de
  desarrolladores, y también todas las relaciones que tuviera con especialidades,
  ya que carece de sentido mantener esta información. */
  foreach ($datosDeLlamada["data"] as $keyReg=>$registro){
   $consulta = "DELETE FROM ".$tablasDeBBDD[0]." ";
   $consulta .= "WHERE ".$columnasDeDatos[4]."='".$keyReg."';";
   $conexion->query($consulta);
   $consulta = "DELETE FROM ".$tablasDeBBDD[1]." ";
   $consulta .= "WHERE ".$columnasDeDatos[5]."='".$keyReg."';";
   $conexion->query($consulta);
  }
  break;
 case "edit":
  /* Cuando se procesa una edición de modifican directamente los campos
  de la tabla de desarrolladores. Para las relaciones con especialidades,
  se eliminan todas las que hay actualmente, y se graban las que se hayan
  seleccionado, como si fueran nuevas. De esta forma, hayan sido modificadas
  o no las especialidades, siempre quedan actualizadas. */
  foreach ($datosDeLlamada["data"] as $keyReg=>$registro){
   $consulta = "UPDATE ".$tablasDeBBDD[0]." SET ";
   $consulta .= $columnasDeDatos[0]."='".$registro["nombre"]."', ";
   $consulta .= $columnasDeDatos[1]."='".$registro["apellidos"]."', ";
   $consulta .= $columnasDeDatos[2]."='".date("Y-m-d", strtotime($registro["fecha_incorporacion"]))."', ";
   $consulta .= $columnasDeDatos[3]."='".$registro["salario_bruto_anual"]."' ";
   $consulta .= "WHERE ".$columnasDeDatos[4]."='".$keyReg."';";
   $conexion->query($consulta);
   $consulta = "DELETE FROM ".$tablasDeBBDD[1]." ";
   $consulta .= "WHERE ".$columnasDeDatos[5]."='".$keyReg."';";
   $conexion->query($consulta);
   if (isset($registro["id_especialidades"])){
    foreach ($registro["id_especialidades"] as $esp){
     $consulta = "INSERT INTO ".$tablasDeBBDD[1]." (";
     $consulta .= $columnasDeDatos[5].", ";
     $consulta .= $columnasDeDatos[6];
     $consulta .= ") VALUES (";
     $consulta .= "'".$keyReg."', ";
     $consulta .= "'".$esp."'";
     $consulta .= ");";
     $conexion->query($consulta);
    }
   }

   $elemento = array("id"=>$keyReg);
   foreach ($registro as $keyItem=>$item) $elemento[$keyItem] = $item;
```

```
    $data[] = $elemento;
   }
   break;
  case "create":
   /* Para la creación de un nuevo desarrollador, primero creamos el registro
   del mismo en la tabla de desarrolladores. Después, tras obtener el id con
   el que ha sido creado, le creamos las relaciones con las especialidades
   que se le hayan asignado. */
   $consulta = "INSERT INTO ".$tablasDeBBDD[0]." (";
   $consulta .= $columnasDeDatos[0].", "; // nombre
   $consulta .= $columnasDeDatos[1].", "; // apellido
   $consulta .= $columnasDeDatos[2].", "; // fecha_incorporacion
   $consulta .= $columnasDeDatos[3]."'"; // salario_bruto_anual
   $consulta .= ") VALUES (";
   $consulta .= "'".$datosDeLlamada["data"][0]["nombre"]."', "; // nombre
   $consulta .= "'".$datosDeLlamada["data"][0]["apellidos"]."', "; // apellidos
   $consulta .= "'".date("Y-m-d", strtotime($datosDeLlamada["data"][0]["fecha_incorporacion"]))."', "; // fecha_incorporacion
   $consulta .= "'".$datosDeLlamada["data"][0]["salario_bruto_anual"]."'"; // salario_bruto_anual
   $consulta .= ");";
   $conexion->query($consulta);
   $elemento = array("id"=>$conexion->lastInsertId());
   if (isset($datosDeLlamada["data"][0]["id_especialidades"])){
    foreach ($datosDeLlamada["data"][0]["id_especialidades"] as $esp){
     $consulta = "INSERT INTO ".$tablasDeBBDD[1]." (";
     $consulta .= $columnasDeDatos[5].", ";
     $consulta .= $columnasDeDatos[6];
     $consulta .= ") VALUES (";
     $consulta .= "'".$elemento["id"]."', ";
     $consulta .= "'".$esp."'";
     $consulta .= ");";
     $conexion->query($consulta);
    }
   }
   foreach ($datosDeLlamada["data"][0] as $keyItem=>$item) $elemento[$keyItem] = $item;
   $data[] = $elemento;
   break;
 }
 $datosParaDevolver = array("data"=>$data);
 $resultado = json_encode($datosParaDevolver);
 echo $resultado;
?>
```

Si llamamos al editor para modificar los datos de un registro, cuando pulsamos el botón de Actualizar, la matriz de datos del formulario que recibe este script, aparte de indicar la acción, tiene los datos en un formato similar al siguiente:

```
array(1){
 [13]=> array(5){
  ["nombre"]=>string(9) "Alejandro"
  ["apellidos"]=>string(17) "Fernández Pérez"
  ["id_especialidades"]=>array(3){
   [0]=>string(1) "1"
   [1]=>string(1) "7"
```

[2]=>string(1) "4"
}
["fecha_incorporacion"]=>string(10) "17-01-2017"
["salario_bruto_anual"]=>string(8) "87500.00"
}
}

Como ves, el contenido del combo múltiple de especialidades ha llegado como una matriz (id_especialidades) que, a su vez, es un elemento pertenciente a la matriz general de datos. Teniendo esto en cuenta, mira los comentarios que detallan como se procesan los datos en caso de que sea un nuevo registro, una edición, o un borrado.

Cómo ves, para gestionar tablas con relaciones m-n debemos contar con combos múltiples, y el editor de DataTables, nos facilita mucho el uso de estos componentes. En este enlace tienes todos los scripts para que puedas copiarlos en tu localhost y comprobar su funcionamiento, mientras lees el código, que es la mejor forma de aprender. Aparte, por supuesto, de experimentar, cambiando aquí y allí el código, y viendo lo que pasa, y por qué pasa.