Gobierno Bolivariano de Venezuela | Ministerio del Poder Popular para Transporte Terrestre

Manual de Normas y Procedimientos para el Desarrollo en PHP, JavaScript, HTML y SQL

Oficina de Tecnología de Información Y Comunicación

COORDINACIÓN DE DESARROLLO DE APLICACIONES

Nombre del documento: Manual de Procedimientos para el Desarrollo de Aplicaciones PHP, JavaScript, HTML, SQL		**FECHA: 10-07-2013**
Elaborado por:	**Revisado por:**	**Aprobado por:**
Ing. Juan L. Carrasco L.	Ing. Juan L. Carrasco L.	Ing. Ramón A. Santander P.

¡Independencia y Patria Socialista... Viviremos y Venceremos!"

HISTÓRICO DE REVISIONES

Fecha	Versión	Descripción	Autor	Revisado por
04/07/2013	RC1	Primera Versión	Ing. Juan Carrasco	Ing. Ramón Santander
07/07/2013	RC1.1	Correcciones de forma. Aplicación de Licencia Creative Commons.	Ing. Juan Carrasco	Ing. Ramón Santander
08/07/2013	RC1.2	Cambios de acuerdo a reunión día lunes 08/07/2013	Ing. Juan Carrasco	Ing. Ramón Santander
10/07/2013	1.0	Versión definitiva a la fecha	Ing. Juan Carrasco	Ing. Ramón Santander

PROPÓSITO DE ESTE DOCUMENTO

El documento siguiente norma y establece los estándares sobre los cuales deben codificarse los programas generados por la Coordinación de Desarrollo de Aplicaciones del Instituto de Ferrocarriles del Estado. Toma como base los Lenguajes PHP y JavaScript. La extensibilidad de su aplicación a otros lenguajes o entornos de desarrollo se estudiará en cuanto sea necesario.

Cualquier segmento de código a partir de la implementación de este manual debe cumplir estos estándares de manera fiel y exacta. Cualquier excepción no contemplada en el documento deberá normarse tan pronto como sea identificada.

Queda entendido que el código de toda aplicación a desarrollar estará sometido a auditoría, con el fin de garantizar su estandarización. Cualquier incumplimiento a las reglas aquí establecidas, sin la debida justificación, podría comprometer seriamente la calidad del trabajo del desarrollador y conllevar a los correctivos pertinentes.

I

PROGRAMACIÓN EN GENERAL

Esta sección brindará una serie de normas y recomendaciones que permitirán una programación más ordenada y eficiente. Todos los archivos a crear tanto en PHP, HTML, y otros formatos, deben estar con codificación de caracteres UTF-8 (encoding UTF-8). Igualmente, la base de datos a usar debe poseer dicha codificación. Así mismo, los editores a usar para programar deben configurarse con la codificación de caracteres UTF-8.

1.1.- Consideraciones Generales

a. Para conservar recursos sea muy selectivo en la elección del tipo de dato, asegúrese que el tamaño de una variable no sea excesivamente grande, ni del tipo incorrecto. Esta observación aplica sobre todo a aquellos lenguajes con asignación dinámica de tipos.

b. Utilice las variables con un solo propósito; en el caso contrario, estos propósitos deben estar relacionados directamente.

c. Use siempre rutinas de manejo de excepciones cuando sea posible.

d. En la programación PHP, utilice print en lugar de echo al momento de mostrar información en pantalla. Esto aplica en los archivos HTML.

e. Evite utilizar la etiqueta abreviada de PHP (<? ?>). Use en su lugar la etiqueta completa (<?php ?>).

f. Siempre que pueda, use sentencias switch en lugar de utilizar sentencias if repetititvas. Ejemplo:

Sustituya:

```
if ($valor1 == 1) {
   //bloque de instrucciones 1
} else if ($valor1 == 2) {
   //bloque de instrucciones 2
} else if ($valor1 == 3) {
   //bloque de instrucciones 3
```

```
}
```

O incluso:

```
if ($valor1 == 1) {
   //bloque de instrucciones 1
}
//Entre estas dos líneas no hay ninguna línea de código
if ($valor1 == 2) {
   //bloque de instrucciones 2
}
//Entre estas dos líneas no hay ninguna línea de código
if ($valor1 == 3) {
   //bloque de instrucciones 3
}
```

Por:

```
switch ($valor1) {
   case 1:
      //bloque de instrucciones 1
      break;
   case 2:
      //bloque de instrucciones 2
      break;
   case 3:
   case 4:
      //bloque de instrucciones 3 que aplica para ambos casos
      break;
   default:
      /*
      bloque de instrucciones si no se cumple ninguna de las
      condiciones de arriba. Omitir si no es necesario.
      No utilizar break aquí
      */
}
```

Esto le permitirá obtener un código más ordenado y eficiente. Puede omitirse la instrucción `break` cuando sea necesario evitar la repetición de código entre dos

casos consecutivos.

g. Queda prohibido utilizar `$_REQUEST` al momento de capturar datos del cliente desde PHP. En su lugar utilice debidamente `$_GET` o `$_POST`, según sea el caso.

h. Evite utilizar nombres de archivo o de páginas diferentes al propósito para el cual se han creado.

i. Establezca un valor por defecto siempre que declare una variable. Procure utilizar `NULL` en lugar de cadenas vacías. Ejemplo:

Sustituya:

```
$valor = ‘’;
```

Por:

```
$valor = NULL;
```

j. Cada sentencia `if` debe contener su o sus sentencias `else` o `else if`, sólo si lo requiere.

1.2.- Sobre la Identación del Código

a. Toda instrucción que se encuentre dentro de alguna estructura que posea un inicio y fin (class, function, for, while, sentencias if, switch-case, entre otros) debe identarse a un tabulador equivalente a 3 espacios. Ejemplo:

```
for ($i = 1; $i <= 10; $i++) {
   if ($i > 5) {
      //bloque de instrucciones
                                                    }
   print $i;
}
```

b. Cualquier estructura o etiqueta HTML debe identarse a un tabulador equivalente a 3 espacios. Es necesario configurar el editor que se use para programar para que reconozca este tabulador. Ejemplo:

```
<html>
   <head>
      <title>T&aacute;tulo de ejemplo</title>
   </head>
   <body>
      <div>
      </div>
   </body>
</html>
```

c. Los símbolos que representen el inicio y fin de una estructura deben colocarse siempre en líneas aparte, y aunque se ejecute una sola sentencia deben colocarse ambos símbolos. Ejemplo:

```
function ejemplo ($arg1, $arg2)
{
   //bloque de instrucciones
}
```

Nota: Será aceptable que el símbolo de apertura esté en la inicial; sin embargo, deberá aplicarse **obligatoriamente** la misma convención en todo el código de la solución que se esté desarrollando. Ejemplo:

```
function ejemplo ($arg1, $arg2) {
   //bloque de instrucciones
}
```

1.3.- Sobre los Comentarios

a. En el caso de la programación web, debe evitarse cualquier comentario que un usuario pueda ver desde su equipo cliente. Estará **prohibido** utilizar comentarios en los archivos de JavaScript o dentro de cualquier bloque de instrucciones contenidas dentro de una etiqueta `<script>` en los archivos HTML.

b. En el caso de los archivos HTML, sólo se aceptarán los comentarios que estén contenidos dentro de una etiqueta **<?php ?>**. Ejemplo:

```
<?php //Comentario ?>
```

c. Los títulos (si los lleva) de los comentarios deben escribirse en letras mayúsculas; el resto del comentario se colocará mediante las reglas de gramática del castellano. Ejemplo:

```
/*
TÍTULO DEL COMENTARIO
Detalle del comentario.
*/
```

d. Todo nuevo archivo de extensión php debe iniciarse con el siguiente comentario:

```
/* ------------------------------------------------
ARCHIVO: nombrearchivo.php
DESCRIPCION: Descripción del propósito principal del archivo.
FECHA DE CREACIÓN: dd/mm/aaaa
OTIC
------------------------------------------------ */
```

e. Todo Comentario debe ir escrito en una línea aparte. Es decir, no deben mezclarse en una misma línea comandos y comentarios. Cualquier instrucción o línea de código que pueda generar algún tipo de confusión debe ser comentada en la línea precedente. Ejemplo:

```
$valor = $valor == 0 ? “a” : “b”;  //Este comentario no va aquí

//Este comentario debería ir aquí
$valor = $valor == 0 ? “a” : “b”;
```

1.4.- Convenciones para la Definición y Asignación de Nombres

1.4.1.- General

a. Estas consideraciones aplican a variables, funciones, rutinas, procedimientos, estructuras y a cualquier otro elemento de desarrollo. Aplican algunas excepciones en

la Programación Orientada a Objetos y en la maquetación en HTML, que se explicarán más adelante en el documento.

b. Los nombres deberán estar compuestos únicamente de letras (sólo minúsculas en el caso de la programación web), números y el guión bajo (_). Debe evitarse a toda costa el uso de caracteres especiales como acentos, puntos y otros.

c. Los nombres estarán compuestos por un máximo de 4 palabras.

1.4.2.- Variables

a. Cada variable debe poseer un nombre estrictamente acorde al propósito de su contenido.

b. Todo nombre de variable debe poseer entre cuatro (4) y treinta (30) caracteres. Sólo podrá omitirse esta regla en las variables usadas para ciclos o incluso en variables de control, como contadores. Ejemplo: `for ($i = 1; $i <= 10; $i++).`

c. Para asignar nombre a una variable, deberá tomarse en cuenta las **cuatro (4)** primeras letras del mismo. Utilizar, por ejemplo: `$cant` en lugar de `$cantidad`. Si el nombre es compuesto, aplica la misma regla, ejemplo: `$suma_deud` en lugar de `$suma_deuda.`

d. No debe emplearse ningún pronombre como parte de las variables (por ejemplo: yo, tu, el, la, los, las, mi, mis). Tampoco deben denotar acciones, por lo cual no deben emplearse verbos en infinitivo (como por ejemplo: `$suma_artic`).

e. Es recomendable que las variables de tipo booleano contengan una palabra que contenga su estado. Por ejemplo: $es_titular, $tiene_hijos. Como puede observarse, es aceptable que se utilice el nombre completo en este tipo de variables. Por otra parte, siempre debe referirse a su estado verdadero. Por ejemplo: `$tiene_hijos` en lugar de `$no_tiene_hijos`.

1.4.3.- Procedimientos y Funciones

a. Definimos como procedimiento a cualquier rutina que realice alguna tarea en particular sin retornar resultados. En los lenguajes en los cuales sólo existe como método de estructuración la palabra función (como por ejemplo function en PHP o Java),

entonces, debe estudiarse el objetivo que persigue la función y determinar si será tratada como función o como procedimiento.

b. Definimos como función, aquellas rutinas que realizan alguna operación y devuelven siempre un resultado.

c. Todo nombre de procedimiento o función debe poseer como longitud mínima cuatro (4) caracteres, y como longitud máxima 35 caracteres.

d. Todo procedimiento debe denotar la ejecución de un trabajo, por lo que su nombre debe comenzar siempre con un verbo en infinitivo.

e. Toda función denota la devolución de un valor, por lo que siempre deben comenzar con un sustantivo.

Ejemplos de nombres de funciones correctos: `promedio_total(), longitud(), resultado_total(), sueldo_total().`

Ejemplos de nombres incorrectos `calcular_sueldo(), o calcula_sueldo().`

Excepciones a esta regla serán:

- Funciones que retornan valores booleanos, las cuales pueden nombrarse iniciando un con verbo pero no en infinitivo. Ejemplo: `es_valido(), tiene_hijos().`

- Funciones que son utilizadas como métodos de Modelos bajo el patrón MVC, lo cual será especificado más adelante en este documento.

II

PROGRAMACIÓN ORIENTADA A OBJETOS

Esta sección tiene como objetivo servir de directriz para la codificación clara y precisa en la Programación Orientada a Objetos. En tal sentido, esta guía se enfocará principalmente en las definiciones de Clases, así como en la forma de representar variables, métodos y objetos.

2.1.- Consideraciones Generales

a. Para nombres de clases y objetos aplica el mismo reglaje que para los nombres de procedimiento en cuanto a longitud, ver regla 1.4.2, literal (a) de la sección I: Programación en General.

b. La misma regla anterior aplica para la definición de atributos, propiedades y parámetros.

c. Para los atributos y métodos privados o protegidos, los nombres deberán iniciar con el prefijo "_" (guión bajo). Por ejemplo:

```
protected $_variable = 'valor';

private function _ejemplo ($arg1) {
   //bloque de instrucciones
}
```

d. Un objeto no debe denotar en ningún caso acciones, por lo que no debe poseer verbos en infinitivo.

e. Para los comentarios de código en sentencias simples, se utilizan las reglas establecidas en el apartado 1.2 de la sección I: Programación en General.

f. Debe optimizarse el código de manera que un bloque de instrucciones no se asemeje en más de un método. Es recomendable valerse de sentencias de control para que esta optimización sea posible.

III

PROGRAMACIÓN DEL LADO DEL CLIENTE

Esta sección tiene como objetivo la estandarización del código cliente, tomando como base los lenguajes HTML y JavaScript y prestando especial atención al framework JQuery.

3.1.- JavaScript

a. Se reiteran las consideraciones indicadas en los apartados 1.1, 1.2, 1.3 literal (a), y 1.4 de la sección I de este documento.

b. Todo el código JavaScript de la aplicación deberá ser guardada en un archivo llamado **common.js**, dentro de una carpeta llamada **public** en la estructura de archivos de la aplicación. Queda **prohibida** la utilización de la etiqueta `<script>` en el cuerpo del documento HTML. La única excepción a esta regla será para las páginas de inicio y cierre de sesión, así como cualquier otra que sea externa a la solución desarrollada.

c. Toda librería JavaScript proveniente de terceros será almacenada dentro de su propia carpeta en **public**.

d. Toda función dentro del archivo **common.js** deberá ser ordenada alfabéticamente.

e. Es aceptable el uso de variables globales, siempre y cuando estén presentes al inicio del archivo **common.js.**

f. Evitar en lo posible el uso de ventanas emergentes (pop-ups) fuera de la página web. Esta regla sólo tendrá su excepción en la presentación de reportes o el cuadro de diálogo de abrir / guardar archivo.

3.2.- Convenciones para la Definición y Asignación de Nombres en JavaScript

a. Para este apartado aplican las mismas consideraciones que se establecen en el apartado 1.4 de la Sección I de este documento.

b. Está prohibido utilizar nombres de variables similares a los que se utilicen en la Base de Datos de la aplicación. Tenga en cuenta que estos nombres podrían ser observados por el usuario final, si este último poseyese las herramientas de

depuración adecuadas (como por ejemplo Firebug, Web Developer o los depuradores nativos de cada navegador).

3.3.- Sobre el uso de JQuery.

a. JQuery es un framework de desarrollo en JavaScript, de software libre y código abierto, con características que facilitan la compatibilidad de aplicaciones en diversos navegadores. En tal sentido, se recomienda su utilización en lugar de la librería Prototype siempre que sea posible**.**

b. Todo evento o función JQuery deberá estar contenido dentro de la función llamada `$(document).ready()`. La única excepción a esta regla serán las funciones siguientes:

```
$(window).resize()
$(window).load()
```

c. La función `$(document).ready()` no deberá utilizarse más de una vez en todo el archivo **common.js** y se recomienda utilizarla posterior a la declaración de variables globales (si existen), pero antes de cualquier otra función desarrollada bajo JavaScript nativo.

d. Toda función `$` o evento contenidos dentro de `$(document).ready()`deberán estar ordenados alfabéticamente.

e. Si ha implementado esta librería en su desarrollo, evite en lo posible utilizar eventos directamente en las etiquetas HTML, como por ejemplo: onLoad, onChange, onClick, onKeypress, entre otros. Utilice en su lugar eventos de JQuery. Para mayor información, visite la página: **http://www.jquery.com**. Ejemplo:

Implementación de un evento en HTML:

```
<body onLoad="ajustar();">
```

Implementación recomendada utilizando JQuery (dentro del archivo **common.js**):

```
$(window).load(ajustar());
```

3.3.- Sobre AJAX y JSON.

a. Utilizando JQuery, se recomienda aprovechar al máximo las siguientes funciones AJAX:
`$.ajax()`
`$.load()`
`$.getJSON()`

b. Cuando se envíen parámetros GET o POST a través de AJAX, recordar que los nombres de los mismos **no** deberán ser iguales a los que se utilicen en Bases de Datos o variables del lado del servidor. Esto aplica de igual manera cuando se obtengan arreglos JSON con AJAX.

c. Al capturar estos valores desde PHP, igualmente tomar en cuenta lo indicado en el apartado 1.1, literal (h) de la Sección I: Programación en General.

d. Utilice JSON siempre que sea posible para mostrar datos para su edición en formularios. No utilice la función $.load para volver a pintar el HTML con los datos en estos casos.

3.4.- HTML

a. La composición de los nombres de objetos HTML se regirá bajo lo indicado en el apartado 1.4.1, literales (b) y (c), 1.4.2, literales (a) y (b) de la sección I de este documento.

b. Se indican los prefijos a utilizar para estos objetos:

Operación	Prefijo
Caja de texto	txt
Combo	cmb
Lista	lst
Radio Button	rbt
Checkbox	chk
Imagen	img
Campo oculto	hdd
Button	btn
Submit	smt

c. Para asignar nombre a un elemento HTML, deberá tomarse en cuenta las **tres (3)** primeras letras del mismo. Ejemplo: `txt_ced` en lugar de `txt_cedula`. Por seguridad, deberá evitarse que estos elementos hereden los mismos nombres de columnas correspondientes en la Base de Datos.

IV

PROGRAMACIÓN BAJO EL PATRÓN MODELO-VISTA-CONTROLADOR (MVC)

4.1.- Consideraciones Generales

a. La programación bajo este patrón se regirá según el framework que se esté utilizando al momento de desarrollar la aplicación. Esto aplicará para nombres de archivos, clases, constructores, destructores, métodos, variables, atributos y cualquier otro elemento inherente.

4.2.- Modelos

a. Los modelos deberán seguir las convenciones establecidas para cada framework en cuanto a los nombres de archivo y de clases. Sólo una clase por archivo es permitida.

b. Para cada tabla de la Base de Datos que se vaya a manipular durante la ejecución del programa, debe crearse su correspondiente modelo en el framework.

c. Cuando se realice una consulta en más de una tabla en una instrucción SQL generada mediante código, esta deberá ser alojada en el modelo correspondiente a la tabla de mayor importancia.

d. Siempre que no colida con la regla 4.1, se indican los prefijos a utilizar para los métodos dentro de los modelos:

Operación	Prefijo
Consultar una sola fila	get
Consultar cantidad de filas	cnt
Consultar varias filas (listar)	lst
Insertar una fila	ins
Modificar un solo valor de la fila	set
Modificar una fila entera	upd
Eliminar una fila	del

e. Aplicando los prefijos anteriores, el resto del nombre de cada método del modelo, deberá escribirse tomando en cuenta las cuatro primeras letras del nombre de la tabla

y, si se trata sólo de un valor de la fila, indicar también el campo. Por ejemplo, si se desea insertar un estudiante, el nombre del método sería `ins_estu()`, pero si sólo se desea utilizar un método para cambiar el campo status de un estudiante, éste deberá llamarse `set_estu_stat()`.

4.3.- Vistas

a. Siempre que no colida con la regla 4.1, las vistas pueden tener nombres acordes al propósito para el cual han sido creadas. El diseño será igualmente adaptado a la funcionalidad de la aplicación.

b. Exceptuando las páginas de inicio y cierre de sesión, se recomienda el uso de plantillas para la aplicación, compuestas de la siguiente manera (de acuerdo a las reglas establecidas por cada framework):

1) Un archivo header1, con la etiqueta `<head>` y su contenido.
2) Un archivo header2, con el encabezado de la página que el usuario visualizará.
3) Un archivo footer, con el pie de página.
4) Un archivo layout, que hará referencia a los tres anteriores, pero que permitirá además definir un contenido principal dinámico.

4.4.- Controladores

a. Siempre que no colida con la regla 4.1, los métodos públicos de cada controlador pueden tener nombres acordes al propósito para el cual han sido creados, ya que para el usuario representan direcciones URL que, dependiendo del uso que le dará a la solución desarrollada, podrían serle de utilidad.

V

DISEÑO Y CREACIÓN DE BASES DE DATOS

5.1.- *Sobre la Representación Lógica*

a. Las entidades deben tener un nombre acorde con la información que se almacena dentro de ellas. Su nombre no debe exceder los 30 caracteres y puede estar compuesto de hasta 3 palabras, sin espacios en blanco ni caracteres especiales (incluyendo acentos, la letra "ñ" o el guión bajo "_").

b. Si el nombre de alguna entidad contiene originalmente la letra "ñ", esta deberá ser sustituida por doble "n". Ejemplo: "`anno`" en lugar de "`año`".

c. Los nombres de las entidades deben ser escritos todos en letra minúscula.

d. Los nombres de atributos poseerán un tamaño de 3 caracteres por palabra. Es decir, si por ejemplo el atributo originalmente se llama "`fecha_ingreso`" se escribirá "`fec_ing`". Por último, se colocará como sufijo las tres primeras letras del nombre de la entidad. Ejemplo: siguiendo con el caso anterior, si el atributo "`fec_ing`" está contenido dentro de la tabla "`empleado`", deberá agregársele el sufijo "`_emp`", quedando el nombre en definitiva como "`fec_ing_emp`". Además, no deberá exceder los 30 caracteres, poseer espacios en blanco o caracteres especiales además de guiones bajos.

e. Si la suma de los caracteres que conforman el nombre de la entidad o el atributo excede de 30, deben cortarse las palabras hasta con un máximo de 6 caracteres y un mínimo de 2.

f. Están permitidos como parte del nombre de entidades y atributos cualquier palabra o abreviatura representativa, pero siempre tomando en cuenta que deberán indicarse en singular.

g. Toda representación de la Base de datos deberá ser documentada a través del diccionario de datos. En él se indicará tanto el nombre completo de cada entidad o atributo, su tipo y propósito.

5.2.- *Sobre la Representación Física*

a. Se implementará la penúltima versión estable de **PostgreSQL** como Sistema Manejador de Base de Datos para el desarrollo de aplicaciones, a menos que las disposiciones legales vigentes o restricciones de tipo técnico obliguen a utilizar otro.

b. Las tablas deben poseer un nombre acorde con el brindado en la entidad correspondiente dentro del modelo lógico (si esta tuviere representación en él), y estará conformado por la misma cantidad de palabras que allí posee.

c. No se permite el uso de espacios en blanco, acentos, letra "ñ", caracteres matemáticos (+-*/) ni ningún tipo de carácter especial.

d. Los nombres de tablas no deben exceder de 30 caracteres de longitud.

e. Los nombres de las tablas deben ser escritos en su totalidad en letras minúsculas.

f. Los nombres de los campos siguen los mismos estándares previos de las tablas.

g. Los nombres de campos que sean clave primaria y del tipo serial deben comenzar con el prefijo "`id_ `". En caso contrario, comenzará con el prefijo "`cod_`". Excepciones a esto último serán campos que hagan referencia a datos personales, números de expediente o similares, como por ejemplo, la cédula de identidad de una persona. En estos casos, el campo iniciará con el prefijo "ced_" o "num_" según sea el caso.

h. Toda tabla poseerá como clave primaria un identificador único de tipo serial. Se crearán índices de tipo UNIQUE para aquellos campos que no puedan poseer valores duplicados.

i. Toda tabla, vista, columna, secuencia, procedimiento, función, trigger, regla, deberán ser documentados utilizando la sentencia SQL `COMMENT`. Algunos ejemplos:

```
COMMENT ON COLUMN tabla.columna IS 'descripción de ejemplo';
COMMENT ON FUNCTION funcion_ej (parametros) IS 'ejemplo';
COMMENT ON RULE reglaejemplo IS 'descripción de ejemplo';
COMMENT ON SEQUENCE secuencia IS 'descripción de ejemplo';
COMMENT ON TABLE tablaejemplo IS 'descripción de ejemplo';
```

```
COMMENT ON TRIGGER triggerejemplo IS ‘descripción de ejemplo’;
COMMENT ON VIEW vistaejemplo IS ‘descripción de ejemplo’;
```

Los procedimientos almacenados y las funciones deberán contener el siguiente comentario al inicio:

```
/*
NOMBRE: fn_ejemplo
DESCRIPCION: Descripción del propósito principal de la función.
FECHA DE CREACIÓN: dd/mm/aaaa
AUTOR: Juan Pérez
MODIFICACIONES EN ORDEN CRONOLÓGICO:
dd/mm/aaaa Por Juan Pérez
dd/mm/aaaa Por José Martínez
*/
```

Nótese que las últimas líneas contendrán un histórico de modificaciones que deberá actualizarse conforme la función o procedimiento sea cambiada. Estos comentarios podrán realizarse a través de la sentencia `COMMENT`, previamente citada.

j. Los nombres de campo que no sean clave primaria de tipo serial, deberán cumplir el mismo estándar de la regla 5.2 literal (d) de esta misma sección.

k. No están permitidos que se usen como nombre o parte del nombre de tablas y campos los artículos gramaticales (el, la, los, las, lo, un, una, unos, una), pronombres, ni preposiciones.

l. Los nombres de los campos deben estar en singular.

m. Cada tabla debe poseer los índices necesarios para agilizar el proceso de búsquedas entre ellas.

n. Por cada tabla se generarán tantos índices como criterios de búsqueda existan para acceder a la misma.

o. El nombre de los índices se regirá por la siguiente estructura:

```
ix_[nombre de tabla]_[nombre de campo].
```

Ejemplos: `ix_empleado_tip_emp`, `ix_empleado_nac_emp_ced_emp`. Como puede observarse, corresponderá a la combinación del nombre de la tabla desde donde se generará y el nombre del o los campos involucrados en la indexación, siempre unidos por guiones bajos y precedidos por el prefijo "`ix`".

p. El nombre de las claves principales se regirá por la siguiente estructura:

`pk_[nombre de tabla]_[nombre de campo].`

Ejemplo: `pk_empleado_id_emp`.

q. El nombre de las claves foráneas se regirá por la siguiente estructura:

`fk_[nombre de tabla que referencia]_[nombre de la tabla referenciada]_[nombre de campo].`

Ejemplo: `fk_hijo_empleado_id_emp.`

r. El nombre de los procedimientos almacenados, funciones, disparadores y reglas se regirá por la siguiente estructura:

`[prefijo]_[nombre de tabla]_[nombre de campo si aplica]_[sufijo]`

El prefijo dependerá del tipo de código a desarrollar en la Base de Datos, a saber:

Tipo	Prefijo
Procedimiento (No retorna valor)	pr
Función (Retorna valor)	fn
Disparador o Trigger	tg
Regla	ru

En cuanto al nombre de la tabla, se tomarán en cuenta las primeras cuatro (4) letras de la tabla cuyos campos predominen en el código a desarrollar. El nombre del campo sólo aplicará en aquellos casos que sólo afecten a una columna del o los registros.

El sufijo dependerá del tipo de operación:

Operación	Prefijo
Obtener uno o un conjunto de resultados	get
Consultar cantidades	cnt
Consultar varias filas (listar)	lst
Insertar una fila	ins
Modificar un solo valor de la fila	set
Modificar una fila entera	upd
Eliminar una fila	del
Devolver el resultado de un cálculo	cal

Ejemplos referentes a procedimientos ligados a una tabla llamada “empleado”:

`fn_empl_ins`	(Insertar registros y devolver si fue exitoso)
`fn_empl_fec_nac_get`	(Devolver el valor de “fec_nac” de un registro)
`fn_empl_edad_cal`	(Calcular la edad a través de una función)
`fn_empl_lst`	(Devolver un conjunto de registros)
`pr_empl_sta_emp_set`	(Cambiar el valor de “status” de un registro)
`fn_empl_upd`	(Modificar registros y devolver si fue exitoso)
`fn_empl_del`	(Eliminar registros y devolver si fue exitoso)
`tg_empl_del`	(Disparador al eliminar registros)
`ru_empl_upd`	(Regla al modificar registros)

s. Operaciones de importancia como cálculos, cambios en uno o varios registros, deben ser creadas en sus propios procedimientos o funciones. Esto permitirá que puedan ser útiles sin necesidad de repetir líneas de código dentro de la base de datos.

t. Con el fin de garantizar mayor seguridad en la información que la base de datos almacenará, el uso de funciones y procedimientos deberá ser implementado en lugar de consultas SQL durante la programación.

VI

SENTENCIAS SQL

6.1.- *Consideraciones Generales*

a. El contenido de esta sección proveerá un estándar para la escritura de consultas SQL. Esto aplica tanto en su uso en el lenguaje de programación como al momento de escribir procedimientos almacenados, funciones, disparadores o reglas dentro de la base de datos.

b. Toda consulta debe tener exactamente la cantidad de campos que se requiera. En caso de que se deba utilizar una misma consulta para mostrar distintos resultados, deberán adaptarse los métodos para permitir que sólo se obtenga la información requerida.

c. Toda palabra reservada o instrucción debe ir completamente en mayúsculas, a menos que el lenguaje restrinja tal práctica.

d. Cuando se utilice directamente una consulta SQL mediante código, se dividirá la sentencia en tantas líneas como sea necesario, y se seguirá la siguiente estructura:

```
SELECT campo1, campo2, campo3, (…) campoN
FROM tabla
WHERE campo1 = valor
ORDER BY campo2 ASC;
```

e. Cada modificador o segmento de la sentencia debe colocarse en una línea aparte. Por ejemplo:

```
INSERT INTO empleados (codigo,nombre)
VALUES (‘01’,’Pedro Perez’);
```

f. Se deben identar los subquerys. Esto aplica tanto en el lenguaje de programación como al momento de escribir funciones o procedimientos almacenados en la BD. Ejemplo:

```
SELECT t1.campo1, t2.campo2, t2.campo3
FROM tabla1 AS t1
     LEFT JOIN tabla2 AS t2 ON t1.campo1 = t2.campo1
WHERE t1.campo1='89'
AND t1.campo1 NOT IN
     (SELECT t3.campo1
     FROM tabla3 AS t3)
ORDER BY t2.campo2, t2.campo3 ASC;
```

g. Colocar el alias de las tablas en consultas donde se involucren más de una tabla con el fin de mejorar la legibilidad. Ejemplo:

```
SELECT t1.campo1, t2.campo2, t2.campo3
FROM tabla1 AS t1
     LEFT JOIN tabla2 AS t2 ON t1.campo1 = t2.campo1
WHERE t1.campo1='89'
ORDER BY t2.campo2, t2.campo3 ASC;
```

h. Toda sentencia SQL debe terminar con ";" (punto y coma).