**De la Salle Bajío**

**2014**

Maestría en Tecnologías web y dispositivos móviles STD- 0121.1 Programación de bases de datos relacionales

Versión 1.3



Jorge Fragoso Mora

Pedro Iracheta

Eduardo Medina

Arturo Najar Madrigal

**Descripción del documento y control de cambios**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Descripción** | | El presente documento presenta las convenciones y estándares de programación para la creación de componentes de bases de datos. El documento define las prácticas a utilizar dentro de este y futuros proyectos. | | |
|  | | | | |
| **Control de cambios** | | | | |
| **Versión** | **Fecha** | | **Adiciones / Modificaciones** | **Preparado / Revisado** |
| 1.3 | 13/08/2014 | | Se anexo encabezado pie de página, formato de letras. | Jfragoso |
| 1.2 | 19/07/2014 | | Revisión y validación de las reglas del documento. | Jfragoso, Piracheta, Emedina, Anajar |
| 1.1 | 12/07/2014 | | Se agregó modelo físico y lógico. | Anajar |
| 1.0 | 05/07/2014 | | Se agregó convención de nombres para objetos de base de datos | Jfragoso |
| 1.0 | 01/07/2014 | | Creación del documento | Emedina |

# CONTENIDO

[CONTENIDO i](#_Toc290290668)

[1 Introducción 1](#_Toc290290669)

[2 Estándar aprobado 1](#_Toc290290670)

[2.1 Lineamientos de base de datos 1](#_Toc290290671)

[2.1.1 Lineamientos que se deben de seguir al momento de realizar la contrucción de una base de datos se enlistan a continuación 1](#_Toc290290671)

[2.2 Reglas para crear nombre de objetos de base de datos 3](#_Toc290290672)

[2.2.1 Caracteres: 4](#_Toc290290673)

[2.2.2 Longitud: 4](#_Toc290290674)

[2.2.3 Palabras reservadas: 4](#_Toc290290675)

[2.2.4 Separadores 4](#_Toc290290676)

[2.2.5 Identificadores numéricos (##): 4](#_Toc290290677)

[2.3 Convención de nombres para objetos de base de datos 4](#_Toc290290678)

[2.3.1 Nombres de objetos de base de datos. 5](#_Toc290290679)

[2.3.2 Verbos 5](#_Toc290290680)

[2.3.3 Palabras simples 5](#_Toc290290681)

[2.3.4 Expresiones cotidianas 5](#_Toc290290682)

[2.3.5 Uso de formas singulares y plurales 5](#_Toc290290683)

[2.3.6 Uso de abreviaciones 5](#_Toc290290684)

[2.3.7 Las abreviaciones que se pueden usar son las siguientes 6](#_Toc290290686)

[2.4 Estándares para creación de un modelo lógico 7](#_Toc290290685)

[2.4.1 Atributos 7](#_Toc290290686)

[2.4.2 Entidades 7](#_Toc290290687)

[2.4.3 Relaciones 8](#_Toc290290688)

[2.4.4 Definiciones 8](#_Toc290290688)

[2.4.5 Propiedades definidas por el usuario 8](#_Toc290290688)

[2.5 Estándares para la creación de un modelo físico 8](#_Toc290290685)

[2.5.1 Nombre de tablas 8](#_Toc290290686)

[2.5.2 Nombres de elementos de tablas 10](#_Toc290290686)

[2.5.3 Esquemas 11](#_Toc290290686)

[2.5.4 Triggers 12](#_Toc290290686)

[2.5.5 Procedimientos almacenados 13](#_Toc290290686)

[2.5.6 Indices 14](#_Toc290290686)

[2.5.7 Variables 15](#_Toc290290686)

[2.5.8 Integridad 15](#_Toc290290686)

[2.5 Estándares de programación de base de datos 15](#_Toc290290685)

[2.5.1 Estándares y mejores prácticas 16](#_Toc290290686)

[3 Ciclo de revisión 26](#_Toc290290689)

1. Introducción

La convención de nombres tiene la ventaja de mejorar la lectura y comprensión de la aplicación tanto para el desarrollo como para el mantenimiento, así mismo tiene soporte para ciertos aspectos de la operación debido a que permite una mejor interpretación de la información contenida en cada elemento de la base de datos. Varios aspectos son posibles o más fáciles si tenemos definidos los estándares y convenciones de nombres

* Actualizaciones (Cambio de versiones) a la base de datos o de software
* Cambio al modelo de datos físico o lógico
* Mover una aplicación de un servidor a otro
* Mejor administración de la base de datos (Uso de utilerías y scripts)

Los aspectos que serán definidos en el documento son los siguientes:

* Uso de nombres y convenciones
* Reglas para crear nombres de los objetos de base de datos
* Convención de nombres de objetos de base de datos.
* Estándares de base de datos.

1. Estándar aprobado
   1. Lineamientos de base de datos
      1. Lineamientos que se deben de seguir al momento de realizar la construcción de una base de datos se enlistan a continuación.

2.1.1.1 En cada construcción de consultas, verificar siempre los planes de ejecución con el fin de reducir al máximo el uso de CPU, I/O.

2.1.1.2 Evitar que las consultas arrojen **INDEX SCAN** o **TABLE SCAN**. Siempre es aconsejable obtener **INDEX SEEK**. En algunos casos, cuando las tablas son muy pequeñas respecto a la cantidad de registros el manejador puede elegir **INDEX SCAN** o **TABLE SCAN**.

2.1.1.3 **NO** usar **WHERE….IN**, sustituirlo por **WHERE EXIST**...

2.1.1.4 **NO** usar el operador **OR**, sustituirlo por otro tipo de consulta o por el **CASE**.

2.1.1.5 **NO** usar operadores como **NOT** o **<>**.

2.1.1.6 En la ejecución de consultas, después del **WHERE**, hacer siempre referencia a las columnas indexadas. (Mismo caso con los **JOINS**).

2.1.1.7 En sentencias **INSERT** siempre colocar el nombre de las columnas. Lo anterior ayuda a evitar problemas de carga / inserción derivado de cambios en la estructura de la tabla.

2.1.1.8 **NO** usar comodines. Escribir siempre el nombre de las columnas requeridas. (Reducción en tráfico de la red y problemas de I/O

2.1.1.9 Mantener las transacciones SQL lo más pequeñas posibles, es decir, generar varias consultas SQL o varios procedimientos almacenados

2.1.1.10 Para el proceso de cargas masivas, es necesario borrar los índices antes de iniciar la carga, reconstruyéndolos al final de la misma

2.1.1.11 Todos los procedimientos almacenados deben analizarse y probarse para de esta forma, obtener un alto desempeño en las consultas incluidas en el cuerpo de los procedimientos almacenados

2.1.1.12 Siempre comentar el código generado dentro de los objetos de base de datos creados como procedimientos almacenados, triggers, índices, etc. Ejemplo

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Descripción : Descripción del objeto de base de datos.

Creado/Modificado : Juan Pérez.

Fecha : 01/01/2014.

Parámetros Entrada : Listado y descripción de parámetros de entrada.

Parámetros Salida : Listado y descripción de parámetros de salida.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

2.1.1.13 A efectos de reducir el tráfico de la red configurar a nivel del cuerpo (inicio) de triggers, procedimientos almacenados o procesos SQL batch el comando **SET NOCOUNT ON**

2.1.1.14 Usar limitadores en la obtención de registros de cualquier tabla. (paginación)

2.1.1.15 **NO** usar tablas temporales en el procesamiento de datos. Solo en algunos casos, el uso de tablas temporales es más eficiente que consultas complejas.

2.1.1.16 **NO** usar cursores. En su lugar utilizar la sentencia **WHILE**

2.1.1.17 Después de la cláusula **ORDER BY**, siempre poner el nombre de la(s) columna(s) en lugar de valores numéricos que hagan referencia al orden de las columnas

2.1.1.18 Cada base de datos será creada como propiedad de un login único para la fase de desarrollo y un login por usuario con los permisos que les corresponda según indicación de los responsables del proyecto

2.1.1.19 Solamente el administrador de la base de datos (**DBA)** podrá utilizar el login **sa**

2.1.1.20 Los desarrolladores tendrán el manejo y la responsabilidad de la generación de la estructura interna de la base de datos durante la etapa de desarrollo, durante la etapa de producción los cambios deberán ser realizados únicamente por el administrador de la base de datos (**DBA)** a petición escrita y debidamente justificada bajo un formato prediseñado

2.1.1.21 Por ningún motivo se le dará acceso a los servidores de base de datos a los desarrolladores, siempre el acceso será mediante el cliente de SQL en sus respectivos equipos

2.1.1.22 La cadena de conexión a la base de datos deberá contener como valor del parámetro “**Server**”, el **DNS** del servidor de base de datos, en lugar de la **dirección IP**. Además deberá llevar el parámetro “**Application Name**”, para identificar el aplicativo que está realizando la conexión. Ejemplo:

Database=xx; Server=Servidor.cpm.mx; User Id=xx; Pwd=xx;

Application Name=SCACS

2.1.1.23 Cumplir con los estándares definidos para la base de datos.

* 1. Reglas para crear nombre de objetos de base de datos

A continuación se definen los nombres estándar para los elementos de una base de datos. Las siguientes reglas son aplicadas cuando se requiere crear objetos dentro de una base de datos.

* + 1. Caracteres:
* Los nombres de los elementos deben consistir solamente de letras A-Z, números 0-9 y guion bajo (\_).
* Los nombres No deben tener espacios en blanco.
* Los nombres NO deben tener caracteres especiales.
* Los nombres No deben iniciar con caracteres numéricos.
  + 1. Longitud:

La longitud de los nombres es propia de cada software por lo que habrá que revisar la documentación propia de la versión utilizada para determinar la longitud máxima del nombre.

* + 1. Palabras reservadas:

El uso de palabras reservadas no está permitido.

* + 1. Separadores

Usar el carácter (\_) para nombres de objetos de bases de datos que están compuestos por varias palabras. No usar (\_) en el nombre de la bases de datos.

* + 1. Identificadores numéricos (##):

En la convención de números el símbolo ## es usado muchas veces para indicar que varios objetos de la base de datos deben de diferenciarse unos de otros; esto se realiza mediante la asignación de identificadores únicos consistente de uno o dos dígitos.

* 1. Convención de nombres para objetos de base de datos

Una de las metas de utilizar convenciones de nombres es hacer que los nombres utilizados sean lo más simple posible ya que el nombre debe ser fácil de recordar y debe de seguir ciertas reglas que sean fáciles de explicar. En este capítulo veremos recomendaciones para:

* Nombres de objetos de base de datos
* Uso de formas singulares y plurales
* Uso de abreviaciones
  + 1. Nombres de objetos de base de datos.

En general siempre es bueno realizar definiciones cortas que involucren solamente una palabra y que a su vez esta sea auto descriptiva; de esta forma se necesita menos tiempo para entender el nombre de un objeto dentro de un documento o diccionario de datos.

Para la definición de nombres se deben de cumplir dos metas principales: el nombre del objeto debe ser tan corto como sea posible y debe de ser auto descriptivo. Una regla que hay que seguir es que cuando el nombre no pueda ser corto se debe elegir siempre que el nombre sea auto descriptivo pensando en que probablemente no sea la misma persona la que realice una implementación y el mantenimiento.

* + 1. Verbos

Si los nombres de los objetos contienen un verbo este deberá ser en infinitivo y siempre se tiene que iniciar el nombre con él. Por ejemplo es mejor escribir Buscar\_Nombre\_Cliente que Nombre\_Cliente\_Buscar.

* + 1. Palabras simples

Utilizar palabras simples para describir un objeto. Por ejemplo usar Buscar\_Complemento o solo Complemento en lugar de Buscar\_Y\_Complementar

* + 1. Expresiones cotidianas

Las expresiones cotidianas no deben ser usadas ya que pueden causar confusión al momento de leer o implementar.

* + 1. Uso de formas singulares y plurales

Las formas singulares son más comunes que las plurales en el uso diario, son más cortas y simples de entender, por lo que es recomendable usarlas en lugar de las plurales.

Como una regla de consistencia de la información cuando se defina el nombre de una base de datos se pueden usar formas singulares o plurales pero no ambas.

* + 1. Uso de abreviaciones

Si es necesario tener que usar abreviaciones, las siguientes recomendaciones nos pueden ayudar a definir una buena abreviación (tomaremos como ejemplo la palabra ERROR).

* Obtener todas las consonantes o caracteres especiales de la palabra original (Por ejemplo ERROR sería RRR)
* Si la palabra original comienza con vocal mantener la vocal en su lugar (ERROR quedaría ERRR)
* Si la regla de arriba resulta en una abreviación que contiene más de dos consonantes idénticas en un renglón solamente dos de estas consonantes deben de permanecer. (ERROR quedaría ERR)
* Si el resultado de la abreviación es difícil de entender, usa una abreviación que sea más comprensible. (Por ejemplo la primera silaba de una palabra o la primera y última consonante de la palabra: PROD en lugar de PRDCT).
  + 1. Las abreviaciones que se pueden usar son las siguientes:
* Id : identificación o identificador
* cant : cantidad
* imp : importe
* min : mínimo
* max : máximo
* fec : fecha ( Dt : Date )
* fecini : fecha Inicial
* fecfin : fecha final
* num : numero
* grp : grupo
* desc : descripción
* loc : localización
* cve : clave
* edo : estado
* mpio : municipio
* cp : código postal
* tel : teléfono
* dom : domicilio
* cod : código
* vta : venta
  1. Estándares para creación de un modelo lógico
     1. Atributos:

1. Todos los nombres lógicos deben ser expresados en términos de negocio con las siguientes excepciones: acrónimos de negocio similares pueden ser usados de manera limitada, el Id es siempre usado para identificadores.
2. Todos los nombres deben estar en formato Title case (inicial en mayúscula)
3. Todos los nombres son singulares
4. Las entidades deben de ser únicas dentro del modelo
5. No se deben de usar abreviaciones al menos que sea requerido por consideraciones de espacio y se deberán de seguir las recomendaciones de la sección anterior.
6. Los nombres deben de consistir e iniciar con caracteres alfabéticos. Los números no son recomendables pero pueden ser usados en caso de ser requeridos. Caracteres especiales no deben de ser usados.
7. Múltiples términos en el nombre deben estar separados por espacio, las diagonales no pueden ser usadas
   * 1. Entidades
8. Un nombre de entidad debe ser un nombre o una frase en forma singular
9. Todos los nombres de entidades, son nombres de negocio y pueden ser agrupados por un término que las clasifique, tal como, Socio para todas las tablas de Socio.
10. Entidades asociadas deben ser creadas usando un nombre con un significado de negocios en lugar de la concatenación de dos entidades. La entidad asociada debe ser sufijo con una abreviación para la referencia cruzada.
    * 1. Relaciones
11. Todas las relaciones tienen significado de una frase verbal y están en minúsculas.
    * 1. Definiciones
12. Las definiciones deben ser una frase detallada de los atributos que típicamente contiene referencias al sujeto componente del atributo. La definición debe consistir de varios enunciados que describan concisamente los datos. Ejemplos de datos deben de proporcionarse siempre que sea posible.
    * 1. Propiedades definidas por el usuario
13. Para todos los objetos, los comentarios deben ser proporcionados en las propiedades definidas por el usuario. Las funciones definidas por el usuario (UDF) son muy similares a los procedimientos almacenados con la diferencia de que las funciones definidas por el usuario pueden ser usadas con la sentencia SELECT; si no fuera así tanto los procedimientos almacenados como las funciones definidas por el usuario serían iguales.

Por tal motivo las reglas aplicadas a los procedimientos almacenados son aplicables a las funciones definidas por el usuario. Las funciones definidas por el usuario deberán de tener el prefijo udf\_. Por ejemplo udf\_Convertir\_Cadena, udf\_Calcular\_RFC.

* 1. Estándares para la creación de un modelo físico
     1. Nombre de tablas:

La regla principal para crear los nombres de las tablas físicas debe estar basada en los nombres de tablas descritas en los estándares del modelo lógico visto en la sección anterior con las excepciones siguientes:

* El guión bajo **(\_)** remplazará a los espacios que dividen los términos en el nombre.
* Uso de términos estándar apropiados. Los nombres deben ser amigables al usuario y consistentes con el mismo término utilizado para el modelo lógico.
* Los nombres **NO** deben contener palabras reservadas propias de la base de datos como **type, name, column, row**, etc.
* **NO** se deben de usar acentos ni la letra ñ.
* Las tablas de trabajo (Tablas cuyo contenido es borrado constantemente para completar un **Script** o **Job**) deben tener el sufijo **\_Wrk** y contener el nombre de la tabla fuente asociada. Por ejemplo: **Detalle\_Cliente\_Wrk**, **Detalle\_Transacciones\_Wrk.**
* Las tablas de reportes deben tener el sufijo **\_Rpt** y contener el nombre de la tabla principal asociada. Por ejemplo: **Historico\_Ventas\_Rpt**, **Cuentas\_Atrasadas\_Rpt**, **Cuentas\_Morosas\_Rpt.**
* Las tablas históricas (Tablas que se usan para almacenar datos históricos de tablas activas) deben tener el sufijo **\_Hst** y contener el nombre de la tabla principal asociada. Por ejemplo: **Transacciones\_Hst**, **Saldos\_Hst.**
* Las tablas temporales deberán de tener el sufijo **\_Temp** y un nombre descriptivo del contenido de la tabla temporal. Por ejemplo **Cuentas\_Socio\_Temp**, **Direcciones\_Socio\_Temp.**
* Las vistas siguen la convención estándar para nombres y deben tener el sufijo **\_View**. Si las vistas son una combinación de varias tablas se debe colocar una palabra representativa de cada tabla para el nombre de la vista. Por ejemplo: **Detalle\_Cuentas\_View**, **Suma\_Saldos\_View**
* Si se usan vistas materializadas estas deben seguir el estándar para la convención de nombres de tablas físicas y deben tener el sufijo **\_Mview**. Por ejemplo **Detalle\_Cuentas\_Mview**, **Suma\_Saldos\_Mview**.
* Los índices deben ser nombrados de acuerdo al siguiente esquema **“NombreTabla\_NombreColumna\_[u][c]\_idx\_n”** donde:
* **NombreTabla** es el nombre de la tabla para la cual se está definiendo el índice.
* **NombreColumna** es el nombre de la columna que será el índice.
* **[u]** es una constante que indica que es índice único. En otro caso es omitido.
* **[c]** es una constante que indica un índice agrupado. En otro caso es omitido.
* **idx** es una constante que indica que el objeto es un índice.
* **n** es un identificador secuencial donde 1 indica que el índice está en la llave primaria y 2 o mayor es usado para todos los otros índices.
* Ejemplo para crear un índice en la columna **Id\_Socio** de la tabla Socios deberá de ser como se muestra: **Socios\_Id\_Socio\_uc\_idx\_1.**
* Ejemplo para crear un índice en la columna **Id\_Cuenta** de la tabla Cuentas que sea un índice no único: **Cuentas\_Id\_Cuenta\_c\_idx\_2.** 
  + 1. Nombre de elementos de las tablas.

Los fundamentos para crear los nombres de los elementos físicos de la base de datos deben de estar basados en los nombres de los atributos de negocio definidos en los estándares para el modelo lógico con las siguientes excepciones:

2.5.2.1 El carácter guion bajo **(\_)** remplazará a los espacios en blanco que separan los términos en el nombre.

2.5.2.2 Uso de términos estándar apropiados. Los nombres deben de ser amigables al usuario y consistentes con el mismo término usado en la definición del modelo lógico.

2.5.2.3 Las siguientes reglas establecen un formato estándar para todos los nombres de los elementos de las tablas. Los elementos de datos físicos consisten de tres partes: **Nombre\_Primario**, **Calificador** y **Palabra\_Clase**.

2.5.2.3.1 El **Nombre\_Primario** es generalmente un nombre que representa un concepto de negocio y a veces contiene la pieza de información más significativa dentro del nombre del atributo. Cada palabra del **Nombre\_Primario** debe tener una definición clara y no ambigua. El **Nombre**\_**Primario** debe describir lo que el objeto es.

Ejemplos: Cliente, Producto, Ciudad, Cuenta.

2.5.2.3.2 El **Calificador** es una descripción opcional que puede también definir o describir al **Nombre\_Primario** y a la **Palabra\_Clase**.

2.5.2.3.3 La **Palabra\_Clase** describe la clasificación de datos asociada con el elemento de la tabla. La **Palabra\_Clase** debe responder a la pregunta básica ¿Qué tipo de objeto es este?

Ejemplos: Promedio, Cuenta, Fecha, Id, Flag.

Generalmente la **Palabra\_Clase** es usada para asegurar que el nombre es único entre términos de negocio similares.

2.5.2.4 Las llaves técnicas o subrogadas deben tener el sufijo **\_Tk**. Las llaves técnicas son sistemas generadores de valores numéricos las cuales no contienen información del negocio. Por ejemplo: **Socios\_Tk**, **Cuentas\_Tk**.

2.5.2.5 Los nombres de los constrains deben ser llamados de la siguiente forma “**NombreTabla\_Tipo\_Descripción”** donde:

o **NombreTabla** es el nombre de la tabla para el cual el constrain aplica.

o **Tipo** es el tipo de constrain y solamente puede ser una de las siguientes opciones:

**pk** llave primaria.

**fk** llave foránea.

**c** validar constrain.

o **Descripción** es información adicional correspondiente al nombre de las columnas usadas en el constrain o a su uso. Por ejemplo:

**Clientes\_pk\_Id\_Cliente**, **Cuentas\_fk\_Id\_Socio**

2.5.2.6 Restricciones **DEFAULT** y **CHECK**. Se debe de usar el nombre de la columna a la cual la restricción está unida y colocar un sufijo **\_def** o **\_chk** respectivamente para los valores **DEFAULT** y **CHECK**.

Por ejemplo: definir la restricción por **DEFAULT** de la columna **Fecha\_Transaccion** como **Fecha\_Transaccion\_def** y la restricción **CHECK** para la columna **Fecha\_Alta** como **Fecha\_Alta\_chk.**

* + 1. Esquemas.

Los esquemas permiten organizar los objetos de la base de datos como si fueran carpetas y facilitan la configuración de seguridad (**GRANT**, **DENY**, **REVOKE**) al poder conceder permisos sobre todos los objetos contenidos dentro de un esquema mediante una única instrucción, ejemplo: **GRANT SELECT ON SCHEMA**.

En cualquier caso, los esquemas, como objetos de base de datos que son, tienen a su vez un propietario o creador. Es decir, un usuario con suficientes permisos puede crear un esquema, siendo él el propietario. Después podrá crear múltiples objetos y asignarlos a dicho esquema, con el fin de utilizar el espacio de nombres que desee.

El uso de esquemas será responsabilidad del administrador de base de datos (**DBA**), en conjunto con el área de la Subdirección de Arquitectura de TI, de acuerdo a la especificación de la Arquitectura de Solución en cuestión.

Se utilizará la siguiente convención de nombres para los esquemas “**Esquema.Objeto**” donde:

* **Esquema** es el nombre del esquema, que sirve para clasificar los objetos de la base de datos. (ejemplo: clasificación por unidad de negocio, **Contabilidad.Socio, Finanzas.Socio**).
* **Objeto** es el objeto de base de datos (tabla, vista, procedimiento almacenado, etc.) que será clasificado dentro del esquema, ejemplos:

o **Contabilidad.Cuenta**

o **Contabilidad.Detalle\_Cuentas\_View**

* + 1. Triggers.

Se utilizará la siguiente convención de nombres para triggers “trg\_xx\_yyyy” dónde:

* trg prefijo que denota que el objeto es de tipo trigger
* xx uno o dos caracteres que denotan el tipo de trigger que se utiliza.
  + d borrar trigger
  + i insertar trigger
  + u actualizar trigger
  + iu insertar y actualizar trigger
* yyyy es la descripción del título del trigger, por ejemplo:
  + trg\_d\_Direcciones
  + trg\_i\_Direcciones
  + trg\_iu\_Direcciones

En caso de que el Sistema Manejador de Base de Datos Relacionales (RDBMS) permita más de un trigger por tabla, se deberá de dar una descripción más detallada para diferenciar entre uno y otro. Por ejemplo:

* trg\_d\_Direcciones\_ValidarDirecciones
* trg\_i\_Direcciones\_ActualizarDirecciones.
  + 1. Procedimientos almacenados

Los procedimientos almacenados realizan la tarea por el usuario por lo que es bueno seguir una serie de reglas para la definición de los nombres.. Las reglas que se deberán de seguir son las siguientes:

2.5.5.1 Usar la siguiente nomenclatura para nombrar a los procedimientos almacenados

“pa\_x\_yyyy”, dónde:

* **pa** es el prefijo que denota que se trata de un procedimiento almacenado.
* **x** es el prefijo que denota el tipo de procedimiento almacenado a ejecutar:
  + **d** procedimiento de borrado.
  + **i** procedimiento de inserción.
  + **s** procedimiento de selección.
  + **u** procedimiento de actualización.
  + **x** procedimiento que combina inserción, selección y actualización.
* **yyyy** es la descripción del procedimiento almacenado.

2.5.5.2 Usar un verbo para describir la acción del procedimiento almacenado. Por ejemplo: **pa\_u\_Actualizar\_Socios**, **pa\_d\_Borrar\_Socios**.

2.5.5.3 Opcionalmente se puede usar un prefijo que agrupe los procedimientos almacenados de una misma tabla. Por ejemplo agrupar todos los procedimientos almacenados pertenecientes a cuentas.

* **pa\_d\_cta\_Borrar\_Cuentas**
* **pa\_s\_cta\_Buscar\_Cuentas**
* **pa\_s\_cta\_Buscar\_Cuentas**

2.5.5.4 Parámetros y una breve descripción: Define los parámetros que maneja el procedimiento almacenado.

2.5.5.5 Resultados y una breve descripción: Define los resultados de las variables que se generaran por la ejecución del procedimiento almacenado.

2.5.5.5 El nombre de la variable deberá identificar plenamente el dato que contiene.

2.5.5.7 El tamaño de un procedimiento almacenado **NO** debe rebasar más de 200 líneas de código.

2.5.5.8 Los procedimientos almacenados **NO** deberán ser utilizados para procesos de extracción, transformación y carga de datos (**ETL**).

2.5.5.9 Los procedimientos almacenados deberán ser utilizados solamente para realizar las siguientes tareas:

* Consultar y persistir datos de aplicaciones sin contener lógica de negocio.
* Procesos batch por ejemplo: mantenimiento y limpieza de tablas, volcado de datos, etc...

**2.5.6 Índices.**

Dado que éste es el punto crítico dentro de todo desarrollo, se deberán seguir las siguientes reglas para los índices.

2.5.6.1 Indexar las columnas que sean solamente estrictamente necesarias.

2.5.6.2 Una tabla deberá tener como máximo 5 índices.

2.5.6.3 El desarrollador tendrá que garantizar que los índices definidos estén siendo utilizados por el **SQL Server**, de lo contrario tendrán que ser borrados.

2.5.6.4 En las tablas de mayor incidencia en actualizaciones, los índices **CLUSTERED** deberán contar con un **FILLFACTOR** adecuado para evitar la fragmentación de la tabla.

**2.5.7 Variables.**

Para variables que almacenen el contenido de las columnas, se puede utilizar la misma regla que se utiliza para nombrar columnas. A continuación se muestran algunas reglas generales:

2.5.7.1 **NO** usar espacios en los nombres de los objetos de la base de datos, ya que los espacios hacen confuso el acceso a las herramientas y aplicaciones de la base de datos. En el caso obligatorio de tener que usar espacios, asegúrate de colocar el nombre dentro de corchetes como se muestra: **[Orden Detalles].**

2.5.7.2  **NO** usar palabras reservadas para nombrar objetos de base de datos, porque eso puede resultar en situaciones impredecibles.

**2.5.8 Integridad.**

2.5.8.1 La integridad referencial será manejada a través de **Constraints Primary Keys** y **Foreign Keys**.

2.5.8.2 Los **Triggers** deberán ser utilizados solo en caso que se quiera hacer alguna acción en cascada sobre la integridad referencial o cuando se quiera evitar que una clave primaria sea modificada o si la funcionalidad del aplicativo así lo requiere, tomando en cuenta que el uso de los mismos degrada según sea el caso la velocidad de actualización de las tablas.

2.5.8.3 En caso de que se requieran hacer validaciones sobre columnas se deberá utilizar el **constraint** del tipo **CHECK** en vez de crear **RULES**, salvo que se quiera hacer una validación de un **USER DEFINED DATATYPE**.

2.6 Estándares de programación de base de datos

Las bases de datos son la fuente principal de muchas de las aplicaciones actuales, razón por la cual es muy importante poner énfasis en su programación. Para obtener mejores desempeños es necesario contar con personal experimentado en programación y en administración de base de datos; esté requerimiento es importante para obtener el mejor desempeño y que la base de datos no se convierta en un cuello de botella para el sistema.

2.6.1 Estándares y mejores prácticas

2.6.1.1 Establecer una regla general para nombrar las bases de datos, estandarizarla por toda la organización y ser consistente al cumplir con los términos, esto ayudará a hacer el código más fácil de entender.

2.6.1.2 **NO** usar las tablas del sistema de base de datos directamente. Las estructuras de las tablas pueden cambiar con futuras versiones. De ser posible, usar el procedimiento almacenado **sp\_help\*** o las vistas **INFORMATION\_SCHEMA**.

2.6.1.3 Normalizar la información de la base de datos, por lo menos en la **3ra Forma Normal**. Hay que tener en cuenta el desempeño de las consultas y en caso de ser requerido desnormalizar los datos para crear consultas más rápidas.

2.6.1.4 **SIEMPRE** escribir comentarios para todos los objetos de base de datos creados, esto ayudará a otros programadores a entender el código claramente. **NO** limitarse en la longitud de los comentarios, ya que no impactará en el desempeño, a diferencia de lenguajes de programación como ASP 2.0.

2.6.1.5 **NO** utilizar **SELECT \*** en las consultas, escribir siempre el nombre de la columna; por ejemplo **SELECT Id\_Cuenta**, **Id\_Socio**, **Numero\_Cuenta**, **Nombre\_Socio**. Esta técnica permite menor utilización de disco y menor tráfico, con lo que se mejora el rendimiento.

2.6.1.6 **NO** usar cursores. **SIEMPRE** tomar en cuenta el **set based approach** en lugar de **procedural approach** para el acceso y la manipulación de datos. Los cursores pueden ser fácilmente evitados con las sentencias **SELECT** en muchos de los casos. Si es inevitable usar cursores, utiliza un ciclo **WHILE** para ir haciendo repeticiones en la tabla. Un ciclo **WHILE** es más rápido que un cursor.

2.6.1.7 **NO** usar tablas temporales mientras se procesa información. Ya que esto genera mayor procesamiento en disco. Considerar vistas avanzadas o tablas de variables en **SQL Server**, en lugar de tablas temporales. Tomar en cuenta que en algunos casos, tiene un mejor desempeño una tabla temporal que una consulta complicada.

2.6.1.8 **NO** usar caracteres **wildcard** al principio de una palabra mientras se realice una búsqueda con **LIKE**, ya que eso se transforma en un escaneo índice, el cual va en contra del propósito de tener índices. La primera sentencia muestra un escaneo índice, mientras que la segunda una búsqueda índice:

* SELECT IdSucursal FROM Sucursales WHERE Nombre\_Sucursal LIKE '%Matamoros'
* SELECT IdSucursal FROM Sucursales WHERE Nombre\_Sucursal LIKE 'M%os'

2.6.1.9 **NO** usar búsquedas con operadores de desigualdad (**<>** y **NOT**), ya que resultan en escaneos de índices y tablas.

2.6.1.10 Utilizar tablas derivadas ya que poseen un mejor desempeño. Observa la siguiente consulta para encontrar el segundo salario más alto en la tabla Empleados:

SELECT MIN(Saldo)

FROM Cuentas

WHERE IDCuenta IN

(

SELECT TOP 2 IDCuenta

FROM Cuentas

ORDER BY Saldo Desc

)

La misma consulta puede ser codificada usando una tabla derivada como se muestra, logrando un rendimiento dos veces más rápido que la consulta anterior:

SELECT MIN(Saldo)

FROM

(

SELECT TOP 2 Saldo

FROM Cuentas

ORDER BY Saldo Desc

) AS A

Este es solo un ejemplo, los resultados pueden cambiar según sea el escenario y dependiendo también del diseño de la base de datos, sus índices, cantidad de datos, etc.

2.6.1.11 Durante el diseño de la base de datos, tomar en cuenta su desempeño. Se puede ajustar el desempeño después, cuando la base de datos esté en producción, ya que conlleva la restructuración de tablas, índices y consultas.

2.6.1.12 **SIEMPRE** utilizar el plan gráfico de ejecución en los comandos del **Query Analyzer** o utilizar el **SHOWPLAN\_TEXT** o **SHOWPLAN\_ALL** para analizar las consultas. Hay que asegurar que las consultas realicen búsquedas de índices en lugar de escaneos de índices o escaneo de tablas.

2.6.1.13 Usar **SET NOCOUNT ON** al comienzo de los procesos batch en SQL, procedimientos almacenados y triggers en los ambientes de producción, ya que se omiten mensajes como este **1 row(s) affected** después de ejecutar las sentencias **INSERT, UPDATE, DELETE y SELECT**. Esto mejora el rendimiento de los procedimientos almacenados reduciendo el tráfico de red.

2.6.1.14 **SIEMPRE** utilizar cláusulas de unión estándares de **ANSI** en vez de las uniones más obsoletas. Con las uniones de **ANSI**, la cláusula **WHERE** es usada para filtrar datos. A diferencia de las obsoletas, donde esta cláusula es usada tanto como una condición de unión como para filtrar datos. A continuación mostramos un ejemplo de su uso: La primera consulta muestra el viejo estilo, mientras que la segunda muestra la unión **ANSI**:

SELECT a.IDAutor, t.Titulo

FROM Autores a

INNER JOIN Titulo\_Autor ta ON a.IDAutor = ta.IDAutor

INNER JOIN Titulos t ON ta.IDTitulo = t.IDTitulo

WHERE t.Titulo LIKE '%Computer%'

En lugar de

SELECT a.IDAutor, IDTitulo

FROM Titulos t, Autores a, Titulo\_Autor ta

WHERE a.IDAUTOR = ta.IDAutor AND

ta.IDTitulo = t.IDTitulo AND

t.Titulo LIKE ‘'%Computer%'

Hay que tomar en cuenta que los operadores \*= y =\* pueden no estar soportados por las nuevas versiones de algunas bases de datos, por lo que es mejor adoptar en estándar ANSI para realizar las consultas.

2.6.1.15 Las vistas son generalmente usadas para mostrar datos específicos a determinados usuarios. También son usadas para restringir el acceso a las tablas base a través de la concesión de permisos en las vistas. Otro uso significativo de las vistas es que te permiten simplificar las consultas. Incorporar las uniones y cálculos recurrentes en una vista, para que no tengas que repetirlas después en las consultas.

2.6.1.16 Usar tipos de datos definidos por el usuario, si una columna en particular se repite en muchas de las tablas, esto permitirá que la columna sea consistente con todas las tablas.

2.6.1.17 Las aplicaciones **Front-End NO** deberán manipular o realizar consultas directamente con las sentencias **SELECT**, **INSERT**, **UPDATE** o **DELETE**. Es mejor crear procedimientos almacenados y dejar que las aplicaciones accedan a ellos. Esta funcionalidad mantiene el acceso a la base de datos limpio y consistente a través de los módulos de las aplicaciones y al mismo tiempo centralizan la lógica del negocio con la base de datos.

2.6.1.18 No usar el tipo de datos texto para almacenar cantidades grandes de texto. Este tipo de dato tiene algunos problemas asociados ya que no se pueden escribir o actualizar datos directamente usando las sentencias **INSERT** y **UPDATE,** en su lugar se deben usar sentencias especiales como **READTEXT**, **WRITETEXT** y **UPDATETEXT**). Por lo que, si no existe la necesidad de almacenar un texto mayor a **8 KB**, se deberán usar los tipos de datos **char(8000)** o **varchar(8000)**.

2.6.1.19 **NO** almacenar archivos binarios de imágenes dentro de la base de datos; es mejor almacenar la ruta del archivo binario o la imagen usándola como puntero al archivo original

2.6.1.20 Usar el tipo de dato **char** para una columna solamente cuando esta sea **no-nula**. Si una columna es **no-nula**, se considera una columna de longitud fija. Por ejemplo, un **char(100)**, cuando es nulo (**NULL**), tendrá 100 bytes, provocando perdida de espacio. Usar **varchar(100)** en esta situación. Hay que tener cuidado al momento de seleccionar entre el uso de **char** y **varchar**, dependiendo de la longitud del dato que será almacenado.

**NO** usar sentencias dinámicas en SQL. **SQL dinámico** tiende a ser más lento que **SQL estático**, ya que se debe generar un plan de ejecución cada vez que se corre algún proceso. Las sentencias **IF** y **CASE** son útiles para evitar **SQL dinámico**. Otra desventaja al usar **SQL dinámico** es que requiere que los usuarios tengan permisos de acceso directo en todos los objetos, tablas y vistas. Generalmente, al usuario se le da acceso a los procedimientos almacenados que hacen referencia a una tabla, pero no directamente a las tablas. En este caso, el **SQL dinámico** no funcionará. Consideremos el siguiente escenario, donde el nombre de usuario **usuario\_Ejemplo** es añadido a la base de datos, y tiene acceso al procedimiento almacenado

**pa\_s\_Ejemplo**, pero no tiene acceso a ningún otro objeto de la base de datos. El procedimiento **pa\_s\_Ejemplo** ejecuta directamente un **SELECT** en los títulos de la tabla y funciona. La segunda sentencia ejecuta el mismo **SELECT** en los títulos de la tabla, usando **SQL dinámico** y nos muestra el siguiente error:

*Server: Msg 229, Level 14, State 5, Line 1*

*SELECT permission denied on object 'titles', database 'pubs', owner 'dbo'.*

Para reproducir el problema anterior, utiliza los siguientes comandos:

sp\_addlogin 'usuario\_Ejemplo'

GO

sp\_defaultdb 'usuario\_Ejemplo', 'pubs'

USE pubs

GO

sp\_adduser 'usuario\_Ejemplo', 'usuario\_Ejemplo'

GO

CREATE PROC pa\_s\_Ejemplo

AS

BEGIN

SELECT \* FROM titles WHERE title\_id = 'BU1032' --This works

DECLARE @str CHAR(100)

SET @str = 'SELECT \* FROM titles WHERE title\_id = ''BU1032'''

EXEC (@str) --This fails

END

GO

Ahora, hay que entrar a la base de datos pubs usando el login **usuario\_Ejemplo** y ejecutar el procedimiento almacenado **pa\_s\_Ejemplo** para visualizar el problema.

2.6.1.21 Hay que considerar las desventajas de usar la propiedad **IDENTITY** para generar llaves primarias. Las columnas **IDENTITY** tienen otros problemas inherentes. Las columnas **IDENTITY** pueden agotar sus números en cualquier momento. Los números pueden ser utilizados automáticamente otra vez, después de borrar filas. Las columnas **Replication** e **IDENTITY** no siempre trabajan bien juntas. Así que, generar un algoritmo para generar una llave primaria, de principio a fin o desde la inserción de un procedimiento almacenado, pueden traer problemas con la generación de llaves primarias, como concurrencias carentes de valores mientras se genera la llave.

2.6.1.22 Minimizar el uso de los valores **NULL**, ya que éstos siempre confunden el desempeño de las aplicaciones, a menos que las aplicaciones hayan sido codificadas para eliminar los **NULL** o convertirlos en otra forma. Cualquier expresión que opere con **NULL** da como resultado un valor **NULL**. Funciones como **ISNULL** y **COALESCE** son de gran utilidad al manejar valores **NULL**. Se presenta un ejemplo que explica el problema: Contamos con la tabla Clientes, la cual almacena los nombres de los clientes y el segundo nombre puede ser **NULL**.

CREATE TABLE Clientes (

Nombre varchar(20),

Segundo\_Nombre varchar(20),

Apellido\_Paterno varchar(20)

)

Ahora, inserta un cliente en la tabla cuyo nombre sea Tony Blair, sin un segundo nombre:

INSERT INTO Clientes

(Nombre, Segundo\_Nombre, Apellido\_Paterno)

VALUES ('Tony',NULL,'Blair')

La siguiente sentencia SELECT regresa un valor NULL, en vez del nombre del cliente:

SELECT Nombre + ' ' + Segundo\_Nombre + ' ' + Apellido\_Paterno FROM Clientes

Para evitar este problema, utiliza ISNULL como se muestra a continuación:

SELECT Nombre + ' ' +ISNULL( Segundo\_Nombre +’ ‘, ‘’ ) + Apellido\_Paterno

FROM Clientes

2.6.1.23 Utilizar tipos de datos como nchar, nvarchar, ntext, si la base de datos va a almacenar otros datos que no sean solo caracteres en inglés. Usa estos tipos de datos cuando sea absolutamente necesario ya que necesitan el doble de espacio que los tipos de datos no-unicode.

2.6.1.24 Siempre utilizar una lista de columnas en las sentencias INSERT. Esto nos ayuda a evitar problemas cuando la estructura de la tabla cambia (como al añadir una columna). Aquí se muestra un ejemplo que explica el problema.

CREATE TABLE EuropeanCountries(

IDCountry int PRIMARY KEY,

Country\_Name varchar(25)

)

Se observa una sentencia INSERT sin una lista de columnas, la cual funciona perfectamente:

INSERT INTO EuropeanCountries

VALUES (1, 'Ireland')

Ahora, vamos añadir una columna a la tabla:

ALTER TABLE EuropeanCountries

ADD EuroSupport bit

Al ejecutar de nuevo la sentencia INSERT. Aparecerá el siguiente error en SQL Server:

*Server: Msg 213, Level 16, State 4, Line 1*

*Insert Error: Column name or number of supplied values does not match table definition.*

Este problema puede ser evitado escribiendo la sentencia INSERT con una lista de columna como se muestra a continuación:

INSERT INTO EuropeanCountries

(IDCountry, Country\_Name)

VALUES (1, 'England')

2.6.1.25 Realizar todas las revisiones de integridad y validación de datos usando restricciones (llave foránea y restricciones **CHECK**). Estas restricciones son más rápidas que los triggers. Debido a ello, utilizar los triggers solo para ejecutar tareas y validaciones que no puedan ser realizadas usando restricciones. Estas restricciones ahorran tiempo también, porque no se tiene que escribir código para estas validaciones y los sistemas manejadores de base de datos relacionales (**RDBMS)** harán todo el trabajo.

2.6.1.26 Siempre entrar a las tablas en el mismo orden en que se entra a los procedimientos almacenados o triggers. Esto ayuda a evitar accesos bloqueados. Para evitar accesos bloqueados tomar en cuenta:

* Mantener las transacciones lo más cortas posibles.
* Manipular la menor cantidad de datos durante una transacción.
* Nunca esperar por una entrada de usuario en la mitad de una transacción.
* **No** usar expresiones complicadas o restrictivas a menos que sea absolutamente necesario.
* Realizar los bloqueos de aplicaciones de forma inteligente; es decir, las aplicaciones deben ser capaces de volver a ejecutar la transacción en caso de que la transacción anterior haya fallado con el error 1205.

2.6.1.27 **NO** ejecutar tareas como: manipulación de cadenas, concatenaciones, enumeración de filas, conversiones **CASE**, conversiones de datos, etc. en las aplicaciones, si estas operaciones van a consumir más ciclos de CPU en el servidor de la base de datos (es mejor hacer manipulaciones simples al final de la base de datos). Tratar de realizar validaciones básicas en el mismo desarrollo durante la inserción de datos.

2.6.1.28 Si la portabilidad del desarrollo final es una prioridad, evitar las manipulaciones con **T-SQL**, ya que esto es algo muy específico del sistema manejador de la base de datos relacional (**RDBMS)**. Más que eso, usar bitmaps para representar diferentes estados de una entidad y no entrar en conflicto con las reglas de normalización.

2.6.1.29 Considerar añadir a los procedimientos almacenados el parámetro **@Debug**. Aplica para un tipo de dato de bit. Cuando el valor procesado es 1, se imprimen todos los resultados intermedios y contenidos de variables usando las sentencias **SELECT** o **PRINT** y cuando el valor procesado es 0 no imprime la información del **Debug**. Esto ayuda a una rápida depuración de los procedimientos almacenados, ya que no es necesario añadir o remover estas sentencias **PRINT** y **SELECT** antes o después de la solución de problemas.

2.6.1.30 **NO** llamar a funciones que se repitan con los procedimientos almacenados, 0triggers, funciones y procedimientos batch. Por ejemplo, se podría necesitar la longitud de una variable de tipo **string** en muchas partes de un procedimiento, pero no utilizar la función **LEN** siempre que sea requerida, es mejor ejecutar la función una vez, y almacenar el resultado en una variable para su uso posterior.

2.6.1.31 Asegurar que los procedimientos almacenados siempre devuelvan un valor indicando el estatus. Así mismo estandarizar la obtención de aciertos y fallas de los procedimientos almacenados. La sentencia **RETURN** tiene como único propósito regresar el estado de la ejecución. Si es necesario regresar datos, usar el parámetro **OUTPUT**.

2.6.1.32 Si los procedimientos almacenados siempre regresan como resultado una sola fila, se debe de considerar regresar el valor resultante usando los parámetros **OUTPUT** en vez de una sentencia **SELECT**, ya que los manejadores **ADO** regresan los parámetros más rápido que los valores resultantes por la sentencia **SELECT**.

2.6.1.33 Siempre verificar la variable global **@@ERROR** inmediatamente después de ejecutar una sentencia de manipulación (como **INSERT, UPDATE o DELETE**), para que así se pueda revertir la transacción en caso de un error (el **@@ERROR** será mejor que el 0 en caso de error).

2.6.1.34 Para hacer las sentencias de SQL más entendibles, empezar cada cláusula en una nueva línea e indentar cuando sea necesario. A continuación se muestra un ejemplo:

SELECT IDTitulo, Titulo

FROM Titulos

WHERE Titulo LIKE 'Computing%' AND

Titulo LIKE 'Gardening%'

2.6.1.35 **SIEMPRE** usar 4 dígitos en el año dentro de las fechas (especialmente, cuando se utilizan columnas con tipos de datos **char** o **int**), en lugar de dos dígitos para evitar confusiones y problemas. Esto no es un problema cuando se utilizan columnas **datetime**, porque el siglo está almacenado aún y cuando el año sea de dos dígitos.

2.6.1.36 En las consultas y otras sentencias SQL, siempre representar la fecha con el formato **yyyy/mm/dd**. Este formato siempre será interpretado correctamente, sin importar cuál sea el formato de fecha asignado por default en SQL Server. Esto también previene errores cuando se trabaja con fechas.

2.6.1.37 **NO** usar la sentencia **GOTO**. El excesivo uso del **GOTO** puede hacer que el código sea difícil de entender.

2.6.1.38 Implementar restricciones únicas en las llaves alternas.

2.6.1.39 **SIEMPRE** ser consistente con el uso del **Case Sensitive** en el código. En un servidor que no sea sensitivo al case, el código puede funcionar bien, pero fallará en uno que si lo sea si el código no es consistente. Por ejemplo, si se crea una tabla o una base de datos en SQL Server que tenga **Case-Sensitive** u orden binario aleatorio, todas las referencias a la tabla deberán usar el mismo **Case Sensitive** especificado en la sentencia **CREATE TABLE**. Si el nombre de la tabla es **MiTabla** con la sentencia **CREATE TABLE** y se usa **mitabla** en la sentencia **SELECT**, el resultado será un error como: **objeto no encontrado** o **nombre de objeto inválido**.

2.6.1.40 Aunque **T-SQL** no cuenta con el concepto de constantes (como las de lenguaje C), las variables se usarán con el mismo propósito. Usando variables en vez de valores constantes en las sentencias de SQL, mejora el entendimiento y mantenimiento del código. Considerar el siguiente ejemplo:

UPDATE dbo.Orders

SET Order\_Status = 5

WHERE OrdDate < '2001/10/25'

La misma sentencia de actualización puede ser re-escrita de una manera más entendible como se muestra a continuación:

DECLARE @ORDER\_PENDING int

SET @ORDER\_PENDING = 5

UPDATE dbo.Orders

SET Order\_Status = @ORDER\_PENDING

WHERE OrdDate < '2001/10/25'

2.6.1.41 **NO** usar los números de la columna en la cláusula **ORDER BY** porque dificulta el entendimiento de la sentencia SQL. Además, cambiar el orden de las columnas en la lista **SELECT** no tiene ningún impacto en el **ORDER BY** cuando las columnas son referidas por nombres en vez de números. Considerar el siguiente ejemplo, en el cual la segunda consulta es más entendible que la primera:

SELECT IDOrden, Fecha\_Orden

FROM Ordenes

ORDER BY 2

SELECT IDOrden, Fecha\_Orden

FROM Ordenes

ORDER BY Fecha\_Orden

1. Ciclo de revisión

Semestral.