**Formación SQL**

**Significado Terminaciones AI y CI:**

Son especificaciones de collation.

**AI** (Accent Insensitive): se ignoran los acentos al hacer comparaciones entre cadenas de texto (útil para búsquedas y comparaciones)

**CI** (Case Insensitive): se ignoran las diferencias entre mayúsculas y minúsculas (útil para búsquedas y comparaciones)

**Que otras teminaciones existen:**

**AS** (Accent Sensitive)

Se tienen en cuenta los acentos (útil cuando hay que tener en cuenta los caracteres acentuados de los no acentuados)

**CS** (Case Sensitive)

Se tienen en cuenta la diferencia entre mayúsculas y minúsculas

**KS** (Kanatype Sensitive)

Se tienen en cuenta los caracteres Kana en el idioma japonés, las diferencias de caracteres Hiragana y Katakana

**WS** (Widht Sensitive)

Se tienen en cuenta las diferencias de ancho entre caracteres. Esto se debe a que algunos caracteres tienen un ancho de visualización diferente (algunos caracteres pueden ocupar dos espacios de visualizacion)

**VS** (Variation Selector Sensivity)

Se tienen en cuenta la presencia de diferentes selectores de variación (caracteres Unicode para cambiar la visualización de otros caracteres Unicode)

**BIN** (Binary)

Permite la compatibilidad con el ordenamiento binario.

**BIN2** (Binary-code point)

Permite el ordenamiento y la comparacion para datos Unicode.

**UTF8**

Especifica que puede usarse UTF-8 para los tipos compatibles.

**Tipos de datos**

|  |  |  |
| --- | --- | --- |
| Tipo de dato | Tamaño | Uso |
| Char | El espacio especificado | Almacena cadenas de caracteres de tamaño fijo  (8000 caracteres) |
| Varchar | Espacio especificado + 2 byte | Almacena cadenas de caracteres de tamaño variable  (8000 caracteres) |
| Text | Espacio especificado + 4 bytes | Almacena cadenas de caracteres de tamaño variable.  Cuando supera el límite de char y varchar  (2,147,483,647 caracteres) |
| Nchar | 2 veces el espacio especificado | Almacena cadenas de caracteres de tamaño fijo basados en Unicode (4000 caracteres) |
| Nvarchar | 2 veces el espacio especificado | Almacena cadenas de caracteres de tamaño variable basados en Unicode (4000 caracteres) |
| Ntext | 2 veces del string | Almacena cadenas de caracteres de tamaño variable basados en Unicode.  Cuando supera el límite de nchar y nvarchar (1,073,741,823 caracteres) |
| Int | 4 bytes | Almacena cualquier entero dentro de su capacidad  (-2,147,483,647 / 2,147,483,647) |
| Smallint | 2 bytes | Almacena cualquier entero dentro de su capacidad  (-32,768 / 32,768) |
| Decimal | Entre 5 y 17 bytes | Almacena cualquier valor decimal dentro de su capacidad  (-10^38+1 / 10^38+1) |
| Float | Entre 4 y 8 bytes | Números aproximativos dentro de su capacidad  (-1.79E +308 / 1.79E +308 ) |
| Bigint | 8 bytes | Almacena cualquier entero dentro de su capacidad |
| Numeric | Entre 5 y 17 bytes | Almacena cualquier valor decimal dentro de su capacidad. Idéntico al decimal (-10^38+1 / 10^38+1) |
| Bit | 1 byte | Campo binario |
| Money | 8 bytes | Valores de moneda. Más capacidad que el smallmoney  (−922,337,203,685,477.5808 /922,337,203,685,477.5807) |
| Smallmoney | 4 bytes | Valores de moneda  (214,478.3647 /-214,478.3648) |
| Real | 4 bytes | Números aproximativos dentro de su capacidad  (-214,478.3648 / 214,478.3648) |
| Tinyint | 1 byte | Almacena números enteros (255) |
| Date | 3 bytes | Almacena solo fechas  (01/01/0001 – 31/12/9999) |
| Time | 5 bytes | Almacena solo horas, 100ns de precisión  (00:00:00.0000000 – 23:59:59.9999999) |
| Datetime | 8 bytes | Almacena fechas con horas con precisión en segundos de .000, .003, .007  (01/01/1900 00:00:00 – 31/12/999 23:59:59) |
| Datetime2 | Entre 6 y 8 bytes | Almacena fechas con horas con precisión de 100ns ente 0 y7 dígitos. Mejor precisión que datetime  (01/01/0001 00:00:00 - 31/12/999 23:59:59.9999999) |
| Datetimeoffset | 10 bytes | Similar a datetime2 pero con el offset de la zona horaria  (-14 / 14 horas) |
| Smalldatetime | 4 bytes | Almacena fechas con horas con precisión de 1 minuto  (01/01/1900 00:00:00 - 06/06/2079 23:59:59) |
| Varbinary | Espacio del string +2 bytes | Almacena un string binario de tamaño fijo o variable  (800 bytes) |
| Binary | Espacio disponible | Alamcena un string binario de tamaño fijo |
| Uniqueidentifier | 16 bytes | Genera un GUID |
| Xml |  | Alamcena XML en una columna |
| Image | Espacio disponible en su rango | Almacena una imagen en formato binario |
| Geography |  | Almacena datos de tipo espacial como GPS |
| Geometry |  | Almacena datos en sistema de coordenadas |
| Hierachyid |  | Representa la posición dentro de una jerarquía |
| Rowversion |  | Almacena un valor binario único autogenerado a nivel de base de datos |

**Int autonumérico vs unique identifer**

Son dos formas diferentes de generar identificadores.

1. **Espacio de almacenamiento:**

INT: Ocupa 4 bytes de almacenamiento.

UNIQUEIDENTIFIER: Ocupa 16 bytes de almacenamiento.

1. **Generación de valores:**

INT: Sin atributos adicionales, no genera valores automáticamente. Debes proporcionar explícitamente los valores al insertar filas en la tabla.

UNIQUEIDENTIFIER: Los valores se generan automáticamente utilizando algoritmos que garantizan la unicidad global.

1. **Eficiencia y velocidad:**

INT: Es más eficiente en términos de espacio de almacenamiento y velocidad de búsqueda debido a su tamaño más pequeño y su naturaleza secuencial.

UNIQUEIDENTIFIER: Tiene un tamaño de 16 bytes y requiere más recursos de almacenamiento y procesamiento en comparación con INT.

1. **Unicidad:**

INT: Los valores de una columna INT no necesariamente son únicos, a menos que se aplique una restricción de unicidad o sea utilizada como clave primaria.

UNIQUEIDENTIFIER: Los valores son globalmente únicos, lo que significa que son únicos no solo dentro de una tabla, sino también en todas las bases de datos y servidores.

1. **Usos:**

INT: Adecuado para identificadores simples dentro de una base de datos donde la unicidad global no es un requisito.

UNIQUEIDENTIFIER: Adecuado para identificadores únicos globales en entornos distribuidos donde la unicidad global es crucial.

**Cuando es adecuado usar default values en el diseño de una tabla**

1. **Valores predeterminados comunes:** Cuando la mayoría de las filas en una tabla tienen un valor común para una columna, establecer ese valor como predeterminado puede simplificar las inserciones de datos y reducir la cantidad de código necesario en las aplicaciones para gestionar esos casos.
2. **Seguridad de datos**: Al establecer un valor predeterminado para una columna, puedes garantizar que siempre haya un valor válido presente, incluso si la aplicación no proporciona uno explícitamente. Esto puede ayudar a mantener la integridad de los datos y evitar errores de inserción.
3. **Mejora de la consistencia:** Al proporcionar un valor predeterminado para una columna, puedes asegurarte de que todas las filas nuevas tengan ciertos campos inicializados de la misma manera, lo que puede mejorar la consistencia de los datos en la tabla.
4. **Compatibilidad con versiones anteriores**: Al agregar una nueva columna a una tabla existente, establecer un valor predeterminado para esa columna puede garantizar que las aplicaciones existentes que insertan datos en esa tabla no se rompan debido a la falta de un valor para la nueva columna.
5. **Optimización del rendimiento:** En algunas situaciones, establecer un valor predeterminado puede ayudar a mejorar el rendimiento de las consultas, especialmente cuando se trata de búsquedas y filtros en los que el valor predeterminado se utiliza con frecuencia.

**Ejemplo:**

CREATE TABLE Usuarios (

ID INT PRIMARY KEY,

FechaRegistro DATE DEFAULT GETDATE(),

EstadoActivo BIT DEFAULT 1);

La columna FechaRegistro tiene un valor predeterminado GETDATE(), que establece automáticamente la fecha de registro del usuario en la fecha actual cuando no se proporciona explícitamente un valor durante la inserción.

La columna EstadoActivo tiene un valor predeterminado de 1, lo que indica que el usuario está activo por defecto.