# Buenas prácticas en SQL

DigitalHouse >
Coding School

Certified Tech Developer
The Ultimate Degree

# Introducción

Las siguientes recomendaciones tienen la finalidad de **optimizar los tiempos de ejecución de las consultas y escribir consultas SQL sólidas**.

Cuando no implementamos estas recomendaciones, SQL realiza el análisis sobre toda la/s tabla/s que se están consultando. Si la tabla tiene numerosos registros, el tiempo en devolver el resultado puede perjudicar a la aplicación.

**¡Arranquemos!**

# Índice

# 1 CREATE

- **Intentemos utilizar VARCHAR en vez de TEXT.**

  Si requiere almacenar volúmenes de texto muy grandes, pero son menores a 8000 caracteres, usemos el tipo de dato VARCHAR en lugar de TEXT.

- **Evalúemos cuidadosamente el uso de CHAR y VARCHAR.**

  El uso de CHAR y VARCHAR depende de si el campo en el que se va a usar varía mucho o no de tamaño. Esto para sopesar rendimiento de velocidad sobre rendimiento de almacenamiento. El motor de SQL procesa más rápido las columnas de longitud fija. Usemos CHAR para columnas de poca variación en longitud y VARCHAR para aquellas que no tienen una longitud estable o promedio.

- **No usemos columnas con tipos de datos FLOAT, REAL o DATETIME como FOREIGN KEY.**

- **Usemos CONSTRAINT para mantener la integridad de los datos.**

- **Evitemos claves primarias COMPUESTAS.**

  Tengamos en cuenta que si esperamos que nuestra tabla con una clave primaria compuesta tenga millones de filas, el rendimiento de la operación CRUD está muy degradado.
  En ese caso, es mucho mejor usar una clave primaria ID simple que tenga un índice lo suficientemente compacto y establezca las restricciones de motor de bases de datos necesarias para mantener la singularidad.

# 2 | SELECT

- **Evitemos usar SELECT * FROM tabla.**

  Aunque resulte fácil y cómodo usar el comodín (*) para traer todos los campos, este debe omitirse y en su lugar especificarse los campos que sean necesario traerse. El uso del comodín impide, además, un uso efectivo de forma eficiente de los índices.

- **Anteponer el ALIAS de la tabla a cada columna.**

  Especificar el alias de la tabla delante de cada campo definido en el SELECT ahorra tiempo al motor de tener que buscar a qué tabla pertenece el campo especificado.

- **Evitemos en la medida de lo posible el uso de GROUP BY, DISTINCT y ORDER BY.**

  Evadamos, siempre que sea posible, el uso de GROUP BY, DISTINCT y ORDER BY, dado que consume una elevada cantidad de recursos.
  Consideremos si es realmente necesario usarlo o si, por otro lado, se puede dejar el ordenamiento de los resultados a la aplicación que recibirá los datos.

# 3 | WHERE

- **Evitemos usar de Wilcards en LIKE como “%valor%”**

  En el caso de que se use la instrucción LIKE, no usemos el comodín “%” al inicio de la cadena a buscar. Esto debido a que si se aplica, la búsqueda tendría que leer todos los datos de la tabla o tablas involucradas para responder a la consulta. Se recomienda que existan al menos tres caracteres antes del comodín.

- **Evitar usar IN en subconsultas, es mejor EXISTS**

  Promover el uso de EXISTS y NOT EXISTS, en lugar de IN y NOT IN.

- **Intente no utilizar funciones dentro de las condiciones del WHERE.**

  SQL no puede buscar eficientemente los registros cuando utiliza funciones, por ejemplo, de conversión, dentro de una columna. En las condiciones intente utilizar el formato de la columna original.

# 4 | UNION

- ## Utilice UNION ALL para evitar un distinct implícito

  En caso de usar la instrucción UNION y existiera la seguridad de que en los SELECT involucrados no se obtendrán registros duplicados, entonces, lo recomendable en este escenario es sustituir UNION por UNION ALL para evitar que se haga uso implícito de la instrucción DISTINCT ya que esta aumenta el consumo de recursos.

# 5 | CRUD

- **Usar SET NOCOUNT ON con operaciones CRUD.**

  Usar SET NOCOUNT ON con operaciones CRUD para no contar el número de filas afectadas y ganar rendimiento sobre todo en tablas con muchos registros.


DigitalHouse>
Coding School