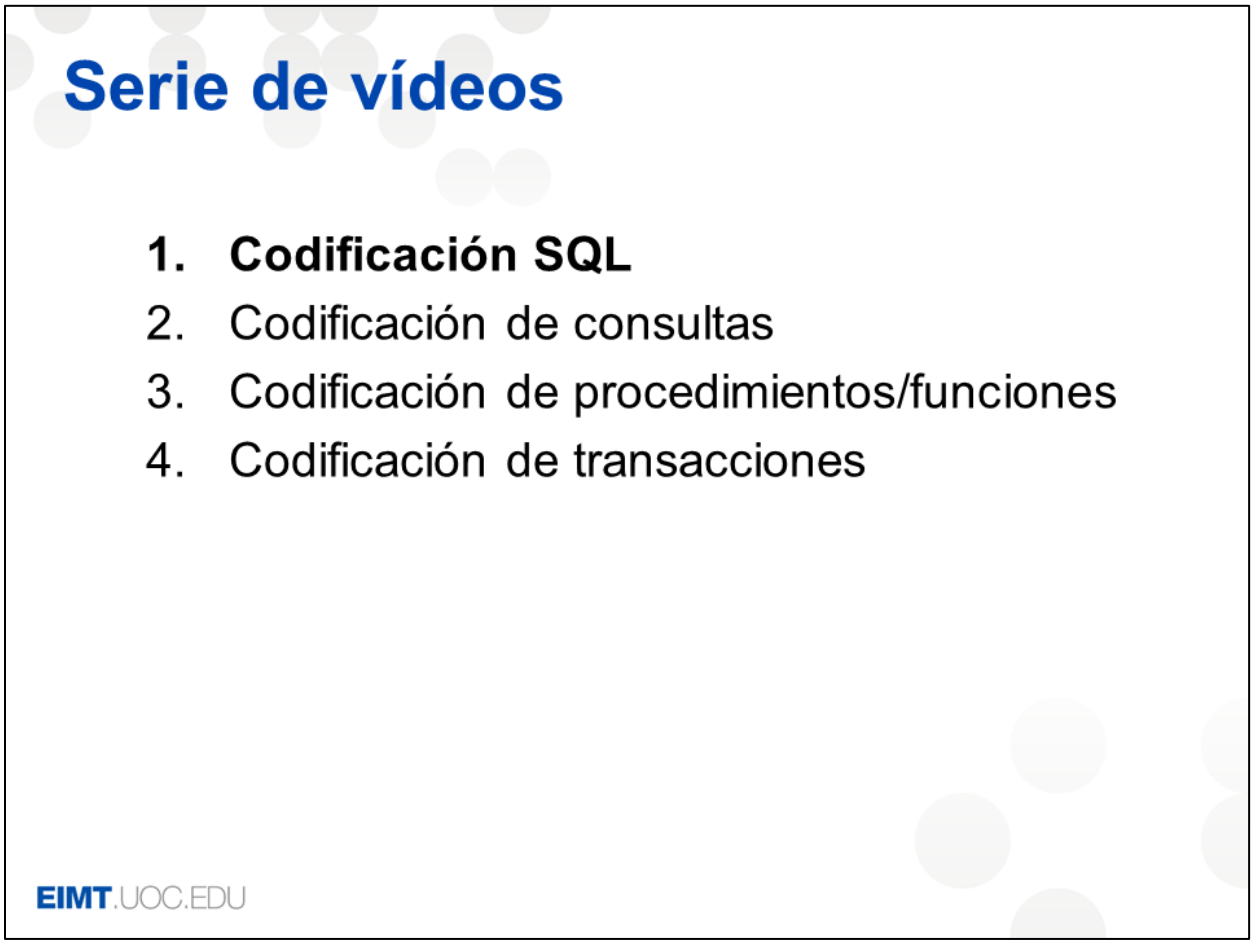
Bienvenidos al primer vídeo de la serie *Buenas prácticas en SQL* de la asignatura de *Bases de datos para Data Warehouse*, serie en la que veremos un conjunto de buenas prácticas a la hora de programar SQL en entornos de bases de datos/*Data Warehouse*.

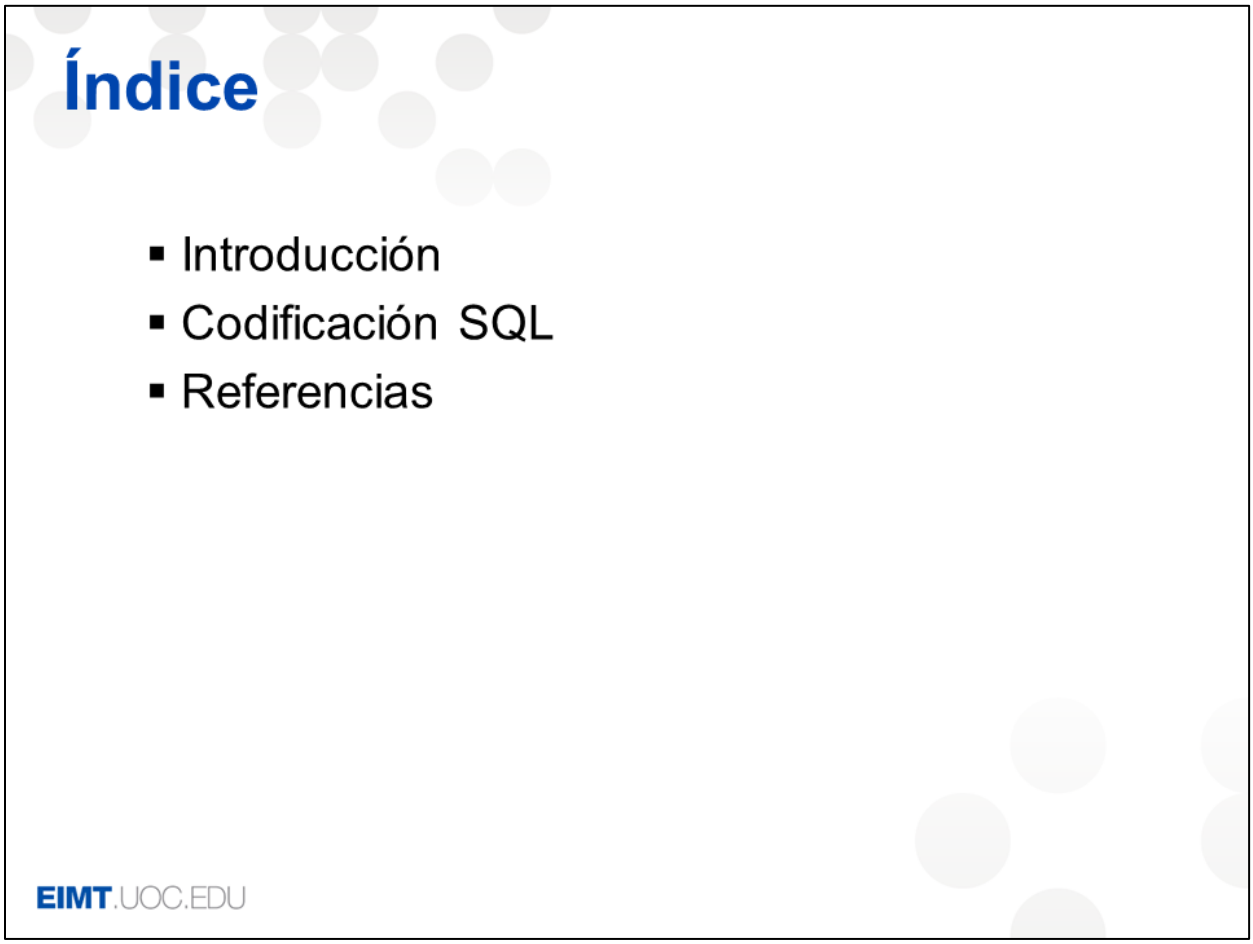


Esta serie de vídeos está dividida en 4 partes:

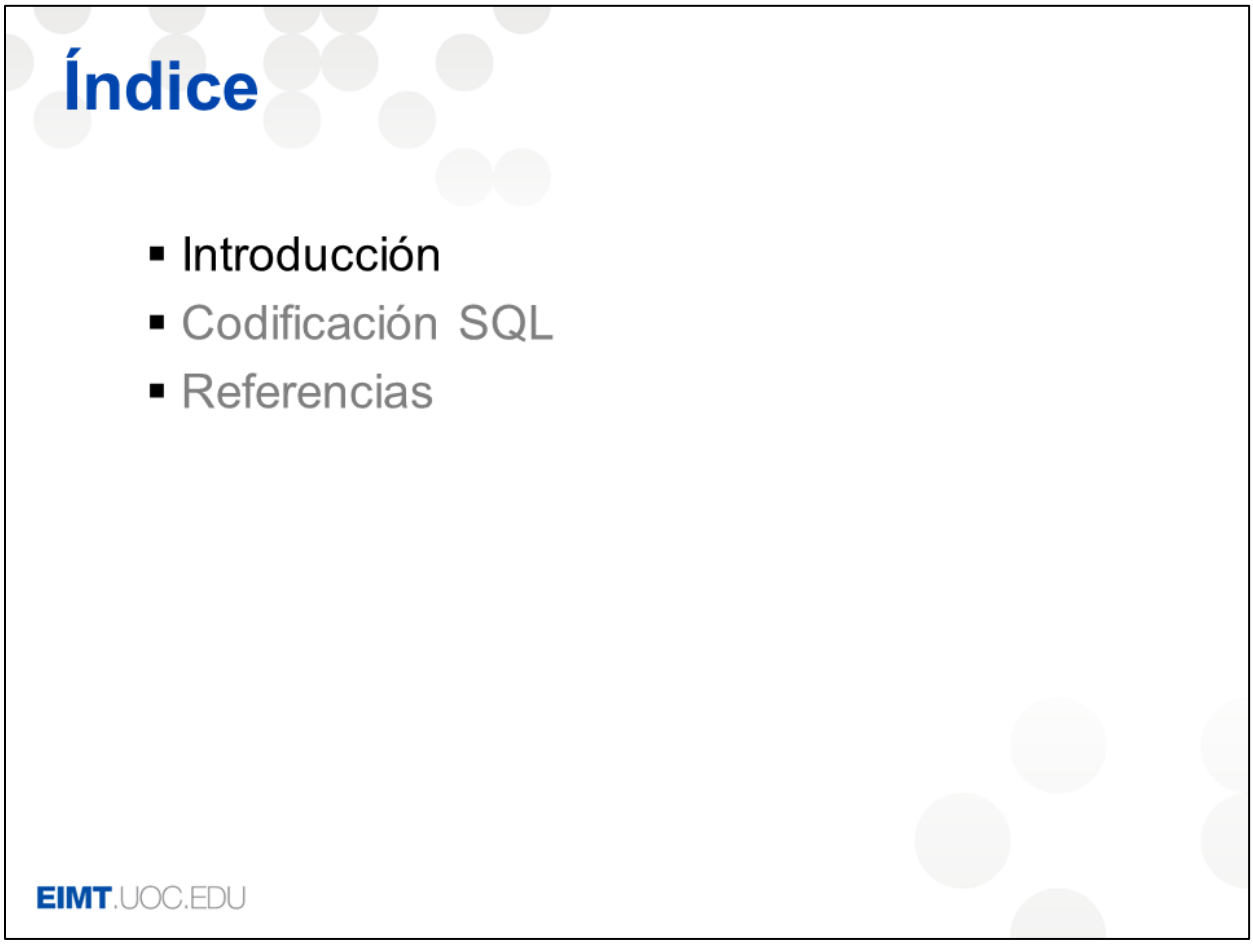
- *Codificación SQL*, centrada en aquellas prácticas desde un punto de vista genérico en SQL, que afectan a la legibilidad y portabilidad de código SQL.
- *Codificación de consultas*, en la que veremos prácticas que nos ayudarán a generar consultas SQL más eficientes y legibles.
- *Codificación de procedimientos y funciones*, donde pondremos atención en aquellas prácticas que nos permitirán escribir código SQL en servidor más eficiente y gestionar errores de forma controlada.
- *Codificación de transacciones*, en la que detallaremos prácticas para asegurar un correcto control de las transacciones que codificamos.

Es importante resaltar que, dentro de cada categoría, no se proporcionan todas las posibles buenas prácticas del mercado, sino que se trata de un conjunto concreto que desde la UOC hemos considerado como más relevantes.

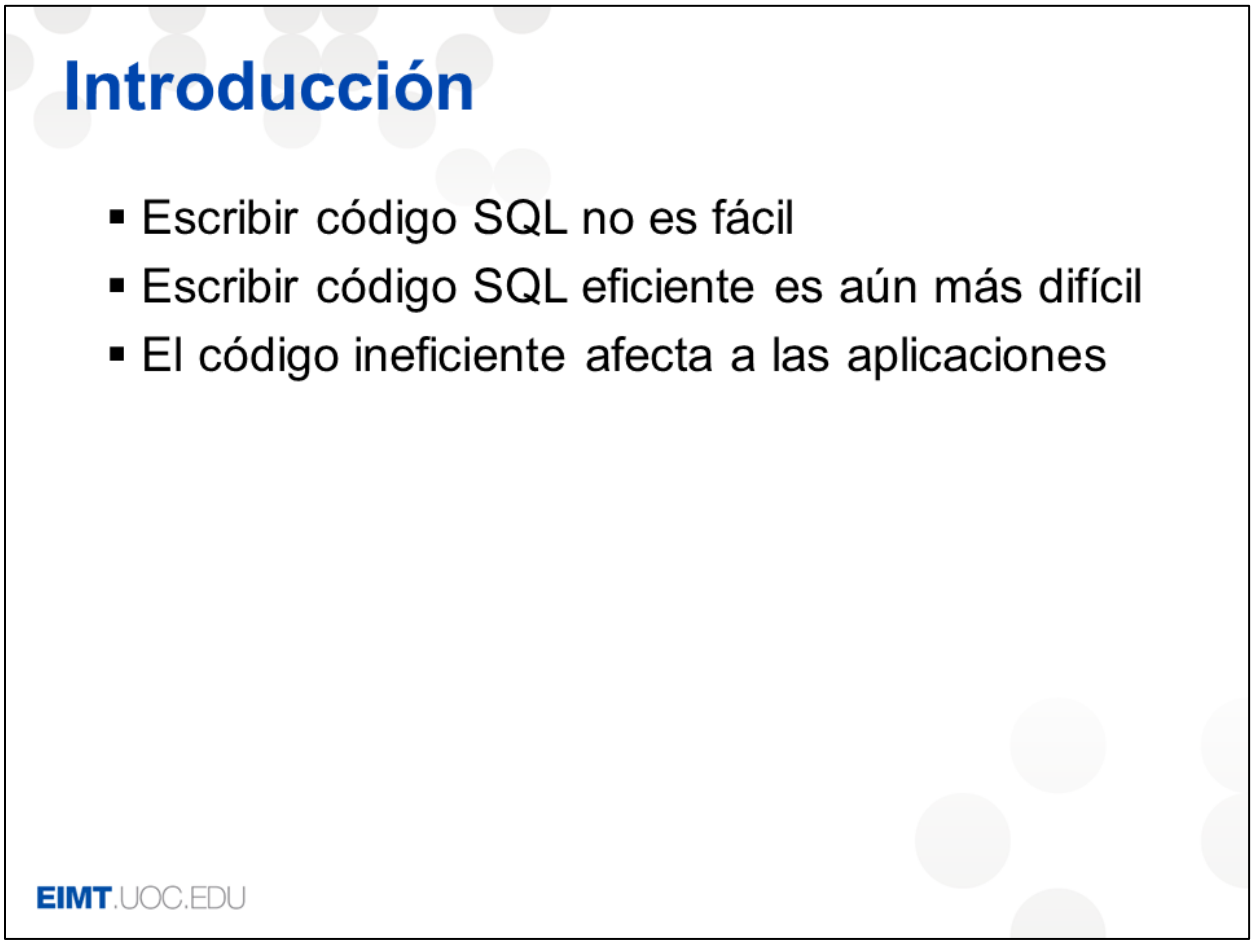
Este vídeo afrontará la primera parte, *Codificación SQL*.



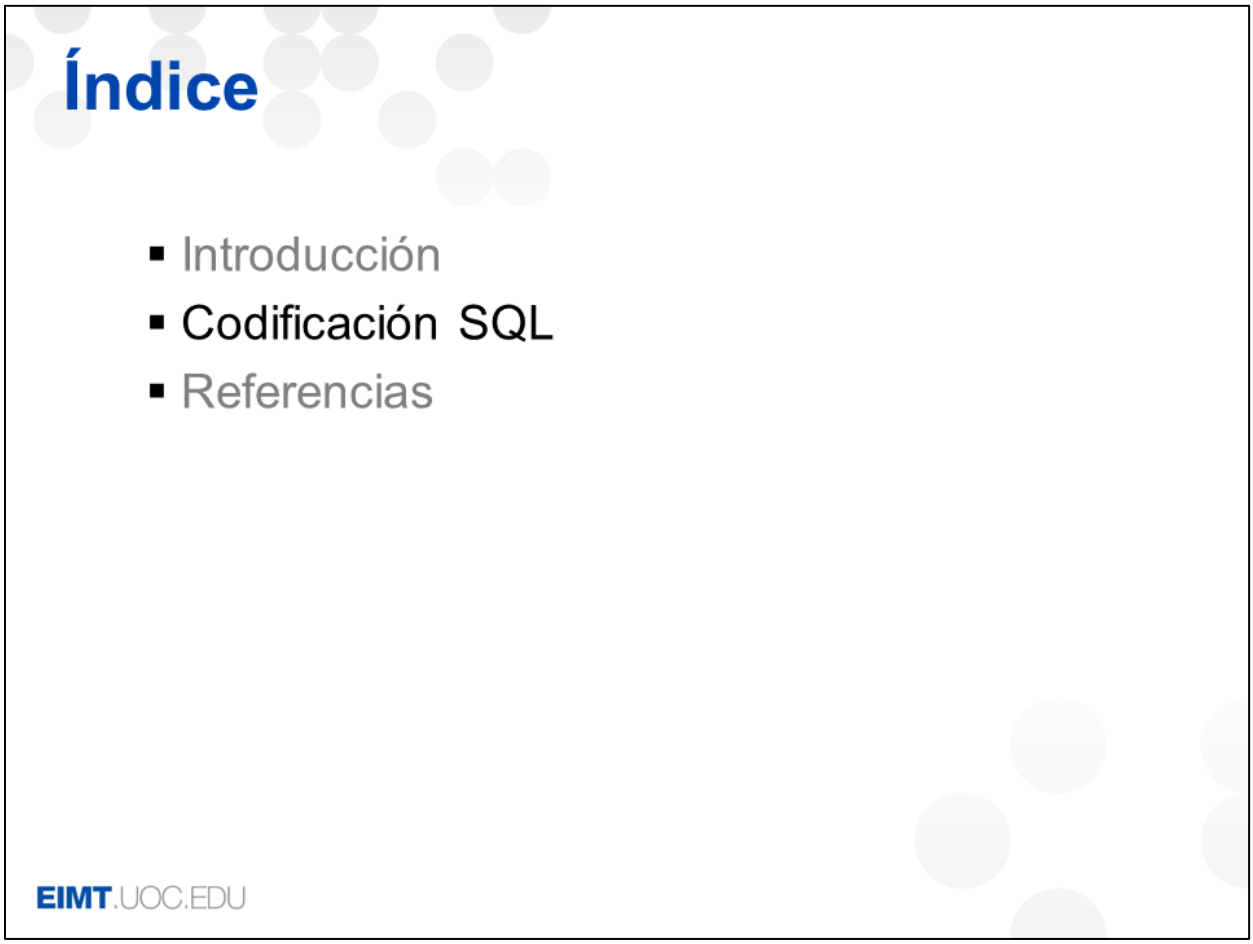
La estructura de este primer vídeo es la siguiente: primero veremos una pequeña introducción a las buenas prácticas en SQL y por qué es importante tenerlas en consideración: entre otras, nos facilitan la lectura del código, mejoran el rendimiento o gestionan los datos de forma más eficiente. Con el fin de ayudaros a escribir y programar SQL más legible y eficiente, veremos a continuación una serie de buenas prácticas de codificación SQL, que nos permitirán generar código de una forma legible y portable, y se propondrán algunos ejemplos para facilitar su entendimiento. Por último, se proporcionarán las referencias bibliográficas utilizadas.



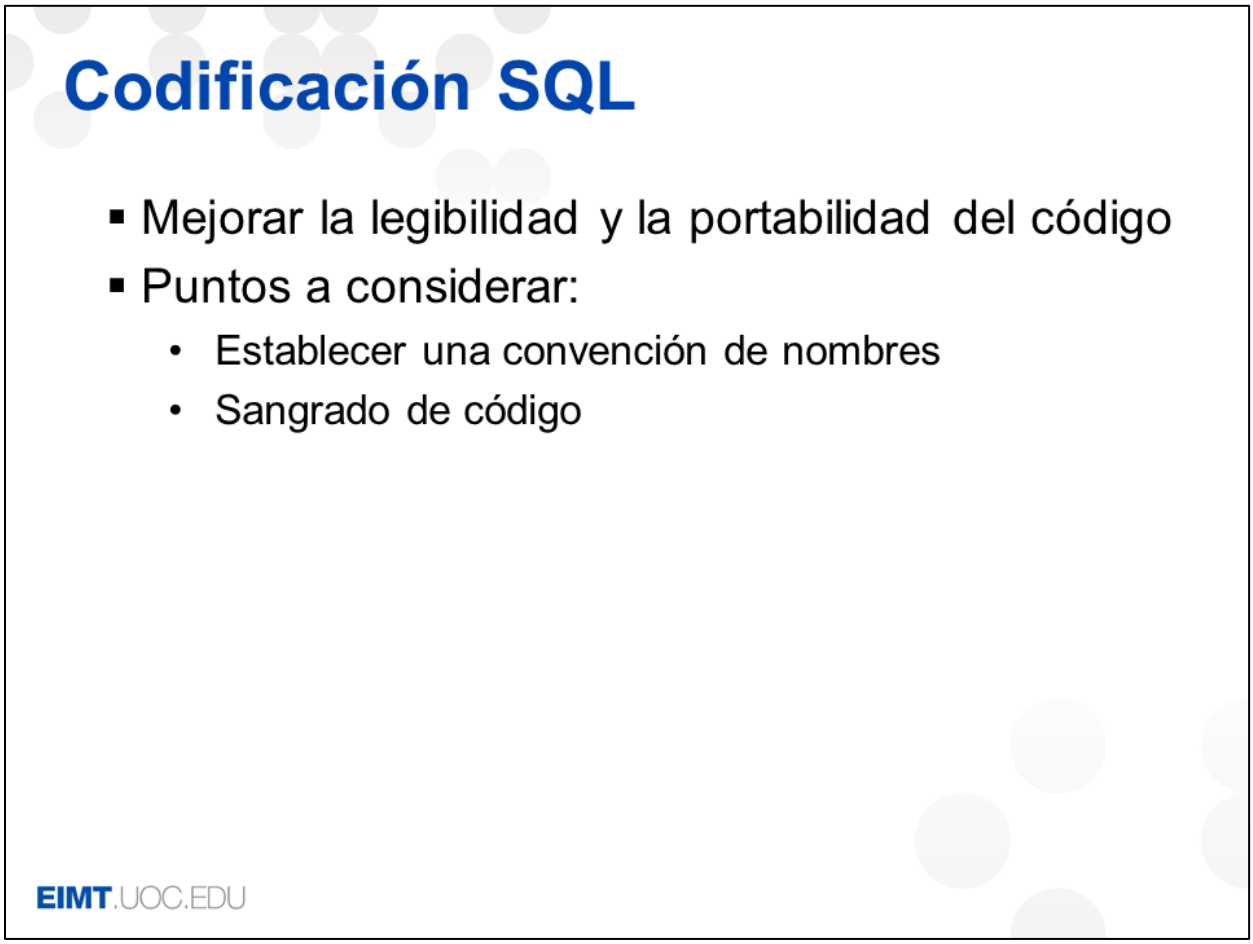
Vamos pues a ver el primer punto, que es la introducción a las buenas prácticas.



Escribir código SQL no es una tarea fácil. La construcción de consultas y bloques de código requiere un cierto nivel de destreza que se obtiene mediante el estudio y uso del lenguaje. Codificar código SQL eficiente y legible es aún más difícil. No solamente es importante conocer el lenguaje a fondo sino que requiere de un conocimiento práctico que se adquiere mediante el uso constante de este lenguaje y de una experiencia en el desarrollo de código para aplicaciones. Este conocimiento práctico es indispensable cuando trabajamos con sistemas gestores de bases de datos (SGBD): a pesar de que la configuración de un SGBD y el diseño de la base de datos puedan considerarse óptimos, el código SQL puede haberse escrito de manera ineficiente, causando así un problema de rendimiento, afectando a las aplicaciones de usuario.



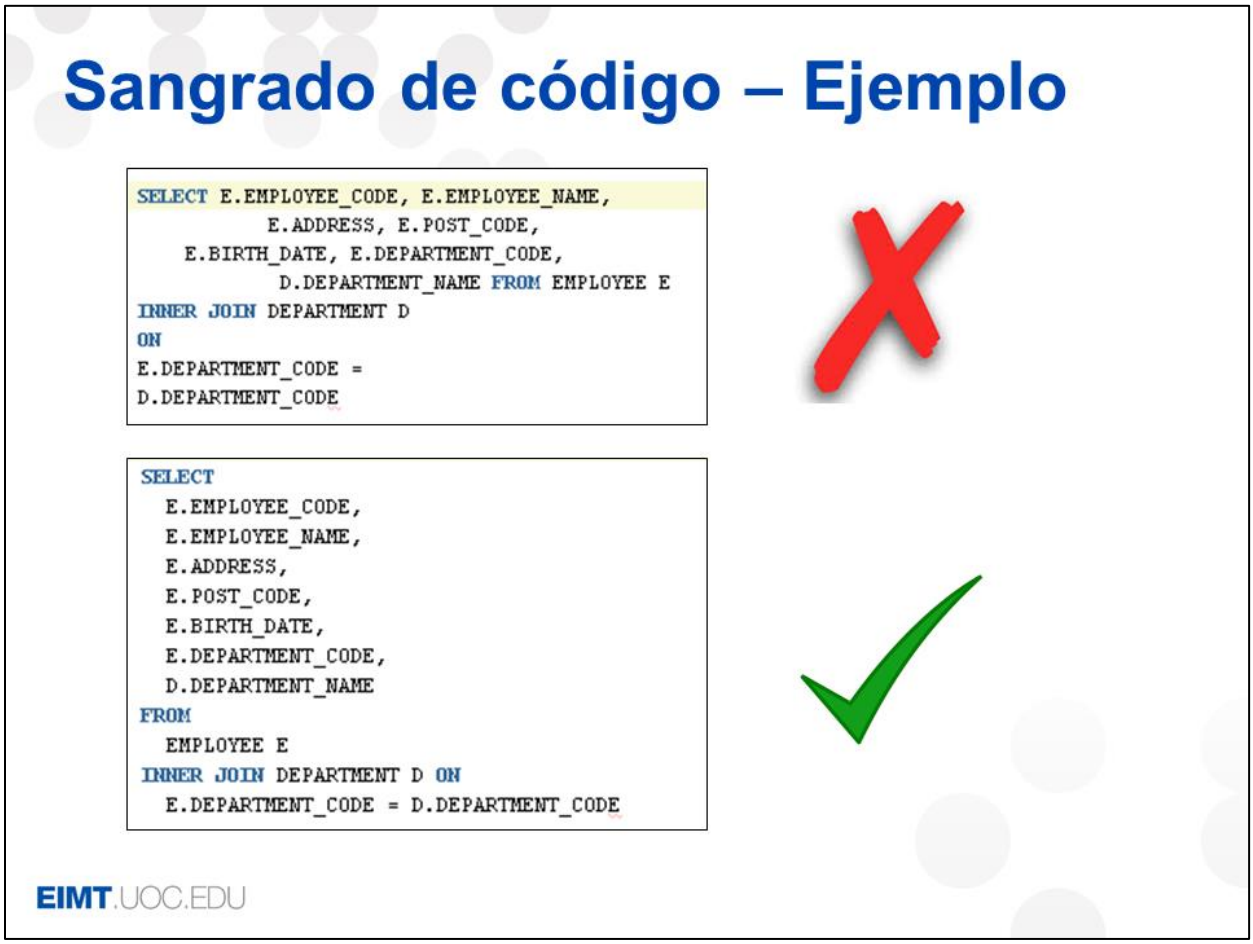
Después de esta introducción, vamos a ver la primera categoría de buenas prácticas: codificación de código SQL.



En esta sección veremos una serie de puntos genéricos que afectan especialmente a cómo debemos codificar SQL de forma que éste sea lo más legible posible y, además, facilite la portabilidad de código en caso de posibles migraciones a otros SGBD. Desde este punto de vista, se recomienda seguir los siguientes puntos:

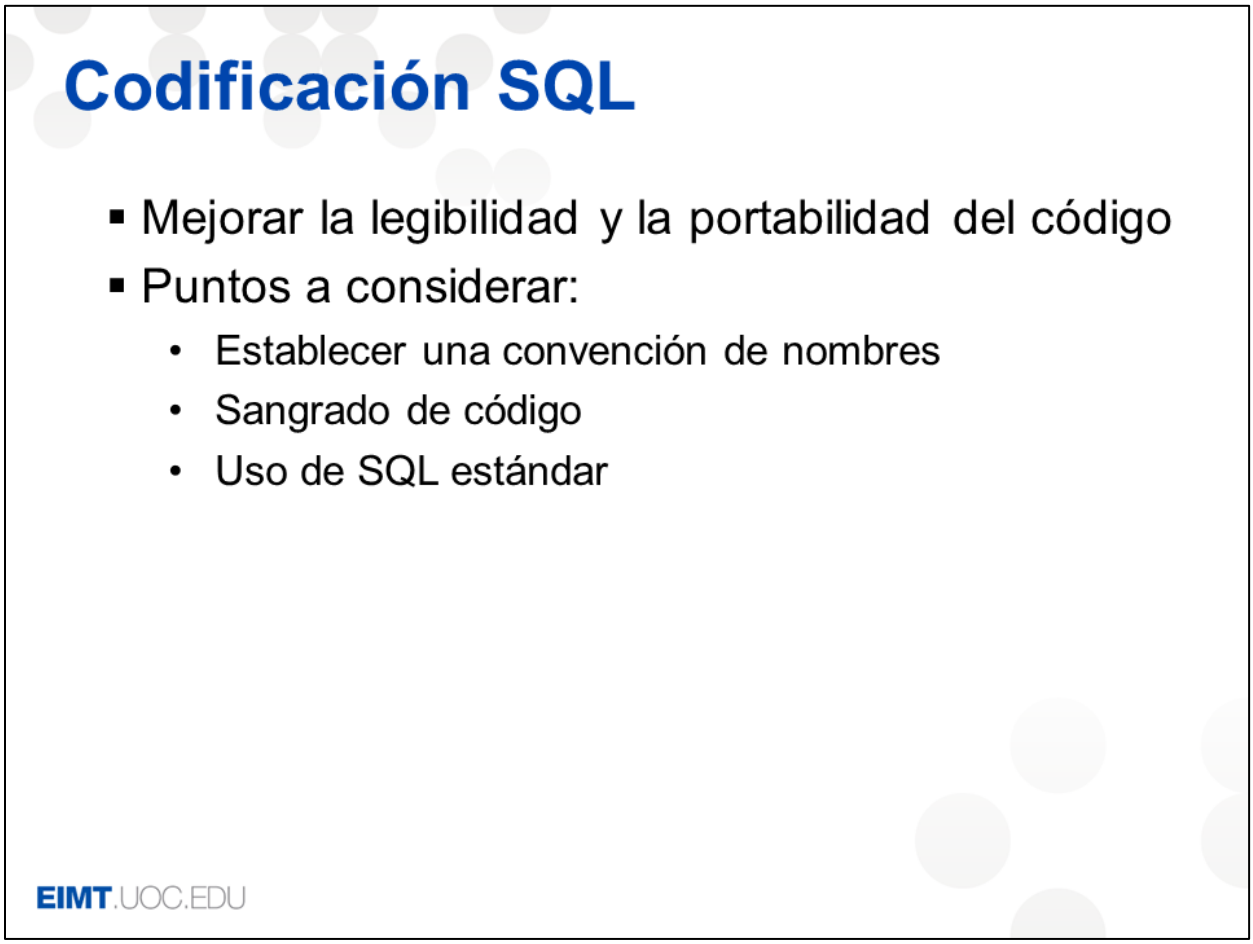
- *Establecer una convención de nombres*: el hecho de nombrar objetos de forma consistente mejora la apariencia del código, facilita su legibilidad e interpretación, y el mantenimiento de éste, además de proporcionar información adicional sobre el uso de dichos objetos en las aplicaciones.

- *Sangrado de código*: entendemos por sangrado el movimiento de bloques de texto hacia la derecha mediante espacios o tabulaciones para separarlo del margen izquierdo, y así distinguirlo del texto adyacente. En informática, se suele utilizar más el término *indentación*, aunque éste es considerado un anglicismo. El uso del sangrado en código SQL mejora la legibilidad por parte de los programadores y facilita el mantenimiento.



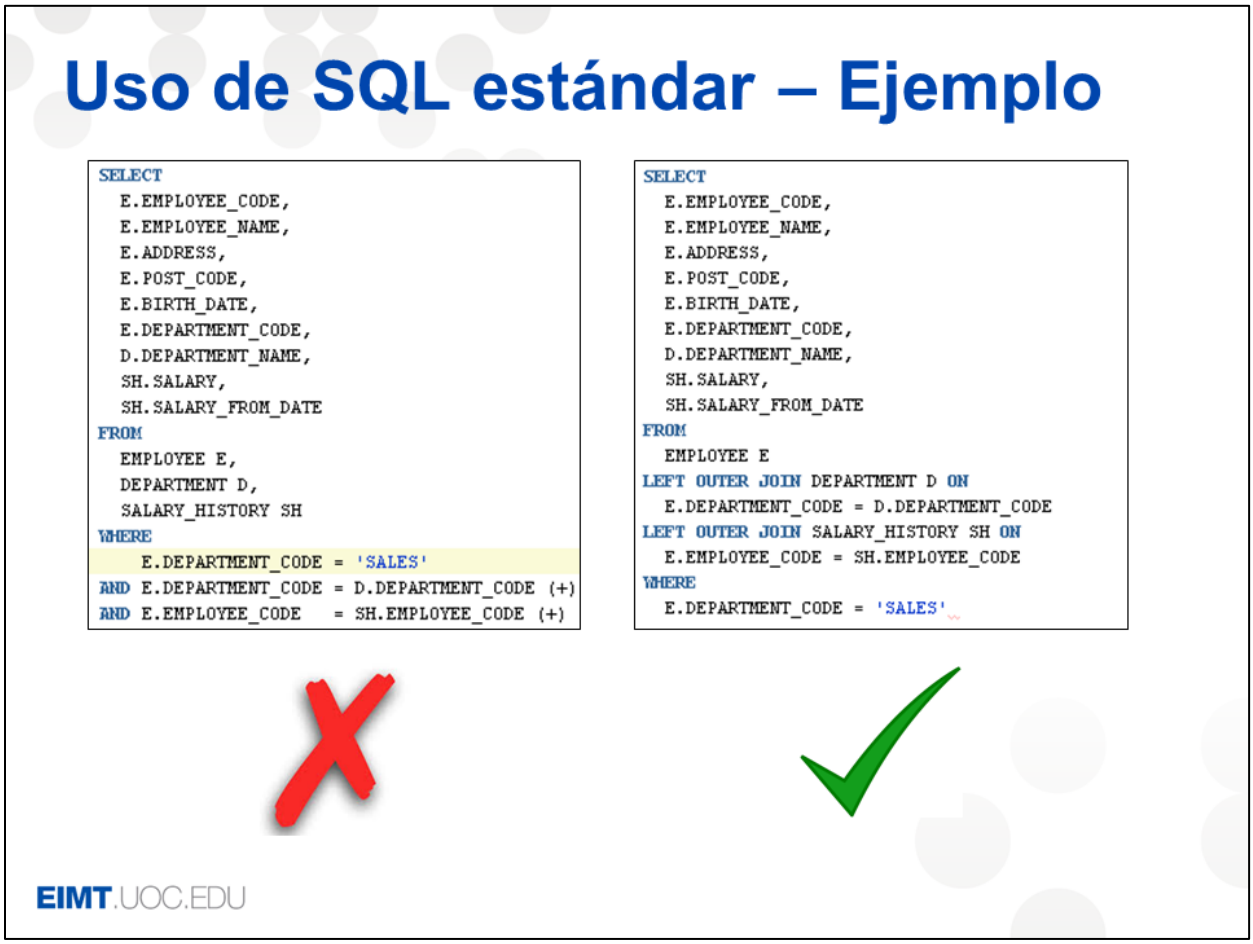
Veamos el siguiente ejemplo de sangrado. La siguiente consulta SQL obtiene información de empleados y el departamento al que pertenecen. Como podemos ver en la imagen, aun siendo una consulta relativamente simple, ésta es compleja de entender: es difícil distinguir qué columnas se obtienen y qué tablas se utilizan. ¡Imaginaos si se tratase de una consulta con 20 tablas y 80 columnas en el `SELECT`! La forma en la que esta consulta está codificada es un ejemplo claro de cómo no debemos codificar SQL.

En la segunda imagen, vemos la misma consulta a la cual se ha aplicado un sangrado de código. En este caso, vemos que es mucho más sencillo identificar las diferentes partes de la consulta (`SELECT`, `FROM`, `INNER JOIN`). Este es un ejemplo de cómo sí debemos codificar SQL.

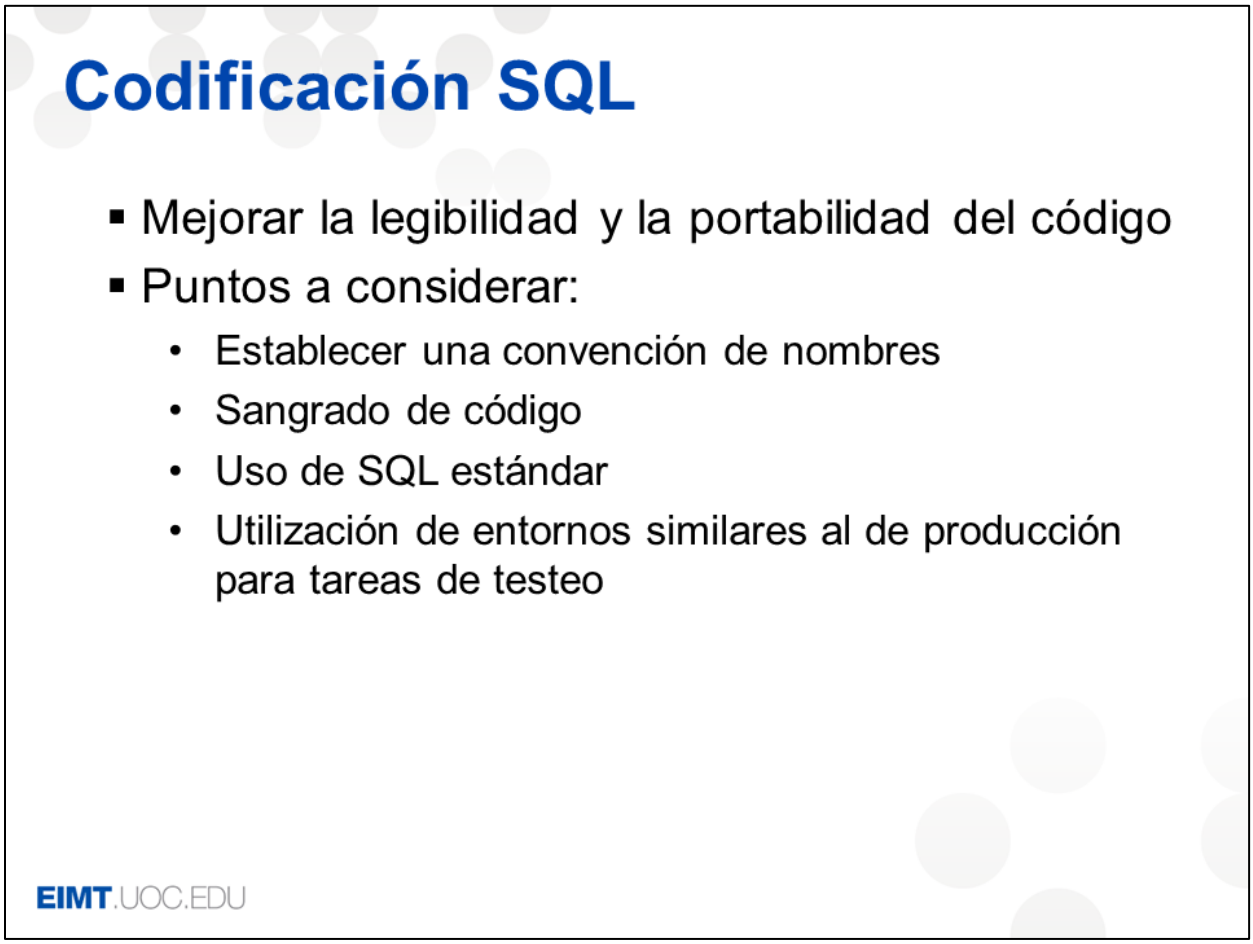


Continuamos el siguiente punto a considerar en la codificación SQL:

- *Uso de SQL estándar*: muchos de los SGBD actuales implementan cláusulas que no forman parte del ANSI estándar. El uso de estas notaciones propias no solamente dificultan la lectura del código, sino que además complica la migración de código de un SGBD a otro. Además de lo mencionado, el uso de cláusulas estándar como `JOIN` en lugar de la forma propuesta en SQL:1989 facilita que en la cláusula `WHERE` solamente se añadan condiciones y no código para la realización de operaciones de combinación.

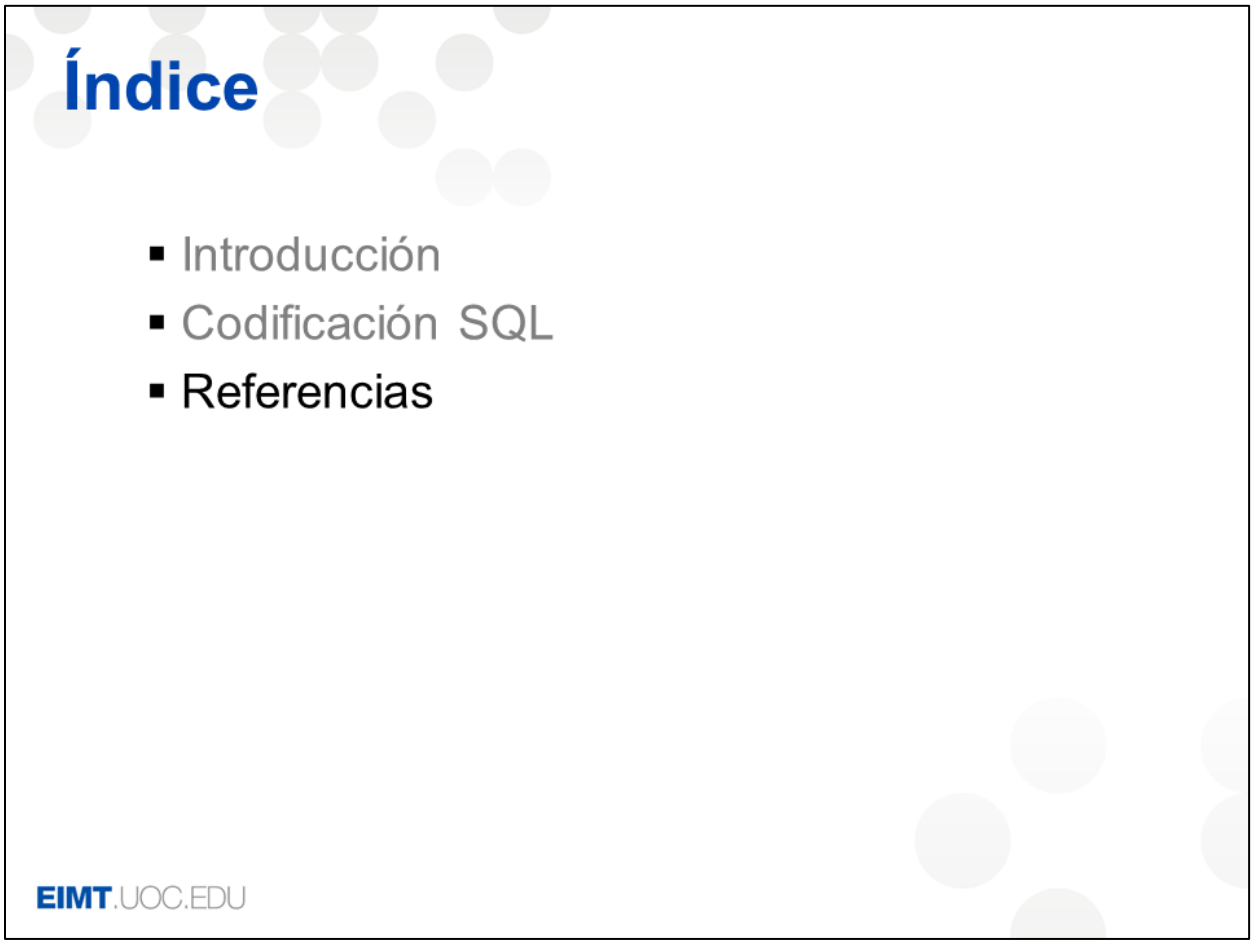


Veamos el siguiente ejemplo de uso de SQL estándar. En el caso de Oracle, la cláusula `OUTER JOIN` se puede implementar mediante la notación `(+)`. Esta notación es propia de Oracle y no es válida para otros SGBD como PostgreSQL, SQL Server o DB2. En estos casos, se recomienda utilizar la cláusula de SQL estándar `OUTER JOIN` (como se puede ver en la segunda imagen), ya que no solamente facilita la lectura de la consulta, sino que además nos evitará problemas en el caso de una posible migración a un SGBD diferente. En la medida de lo posible, se recomienda siempre el uso de SQL estándar.



Como último punto de la lista a considerar en codificación SQL:

- *Utilización de entornos similares al de producción para el testeo*: cuando se crean aplicaciones que utilizan bases de datos, suele existir una separación de entornos para facilitar el desarrollo de la fase de testeo, y éste del entorno de producción. Estos entornos de testeo deberían ser, en la medida de los posible y desde un punto de vista de hardware y software, similares al entorno de producción. Una de las razones principales es la de garantizar que la aplicación se ejecute de forma correcta, pero también es muy importante que aquellas consultas implementadas se ejecuten de forma correcta. La disponibilidad de un entorno similar al de producción nos facilita realizar pruebas de rendimiento en el SGBD y verificar que el código SQL generado es eficiente según las necesidades de la aplicación.



Y hasta aquí hemos llegado con este primer vídeo de la serie *Buenas prácticas en SQL*. Esperamos que os haya gustado la presentación y que esta os haya servido de mucha ayuda.

Presentaremos ahora un conjunto de enlaces y referencias de interés acerca de este tema.

Referencias

Rankins, R.; Bertucci, P.; Gallelli, C.; Silverstein, A.. (2013). *Microsoft® SQL Server 2012 Unleashed*. Sams.

PostgreSQL:

http://www.postgresql.org/docs/9.3/

SQL Server Performance:

http://www.sql-server-performance.com/2001/sql-best-practices/

EIMT.UOC.EDU

¡Que tengáis un buen día!