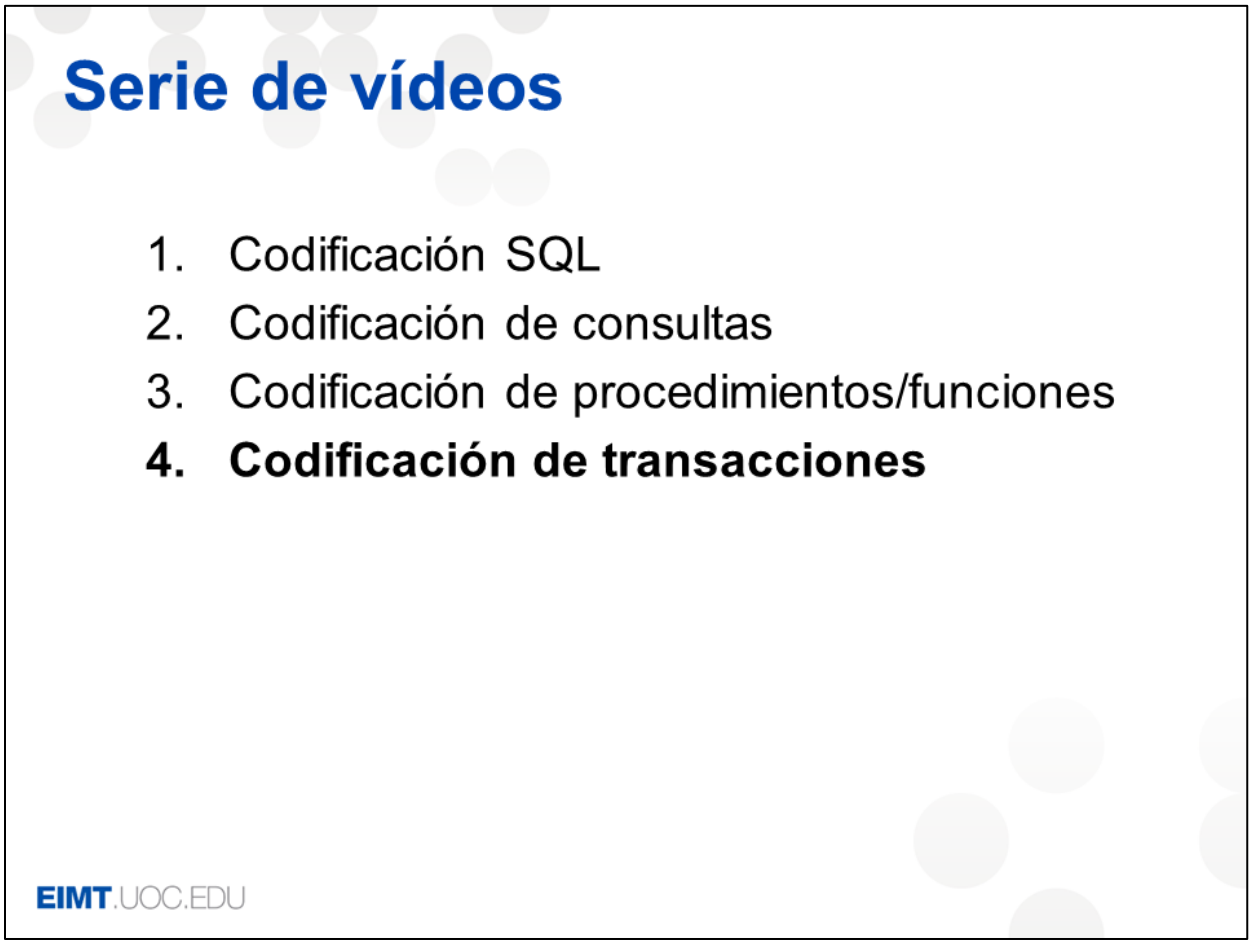
Bienvenidos al cuarto y último vídeo de la serie *Buenas prácticas en SQL* de la asignatura de *Bases de datos para Data Warehouse*, serie en la que veremos un conjunto de buenas prácticas a la hora de programar SQL en entornos de bases de datos/*Data Warehouse*.

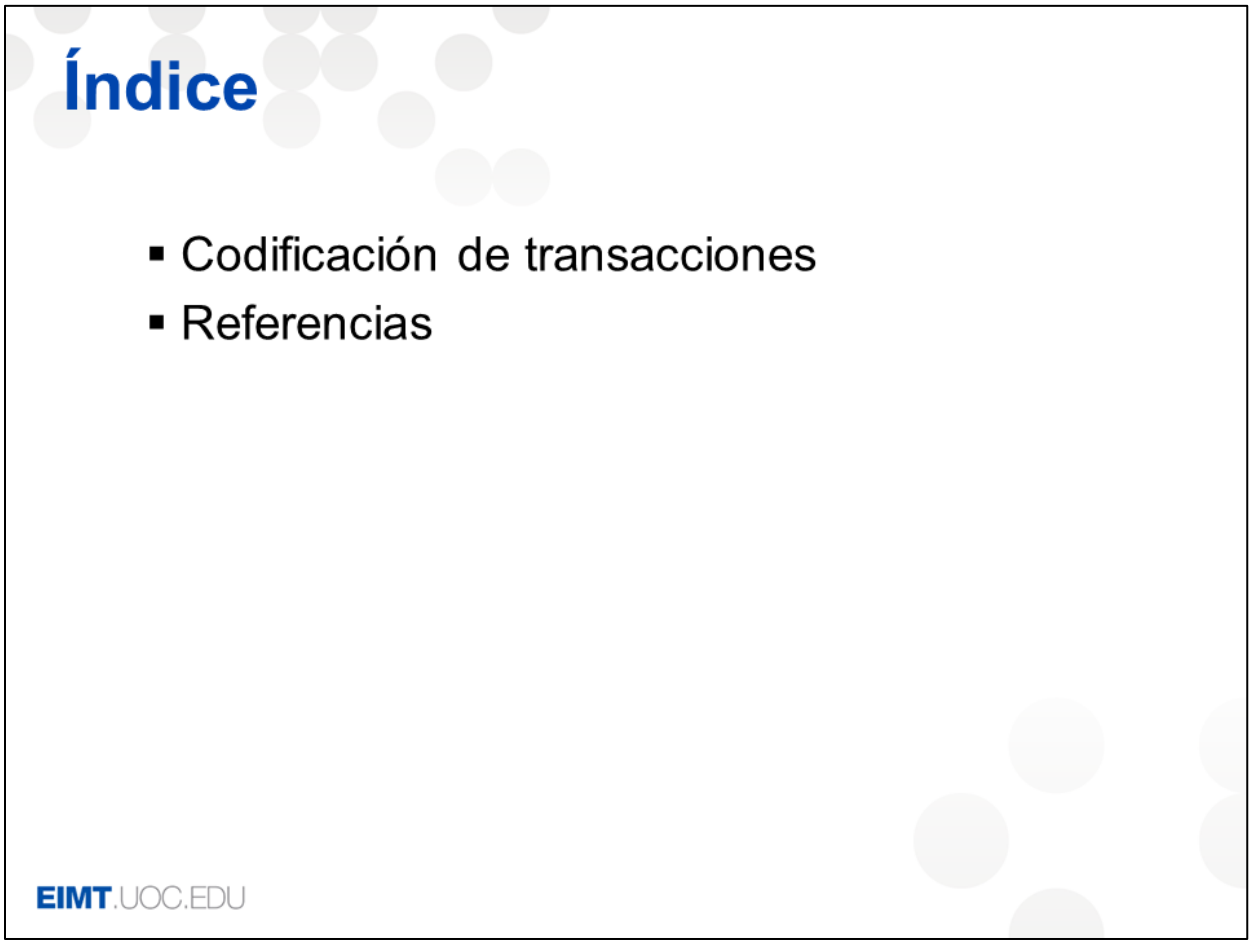


Esta serie de vídeos está dividida en 4 partes:

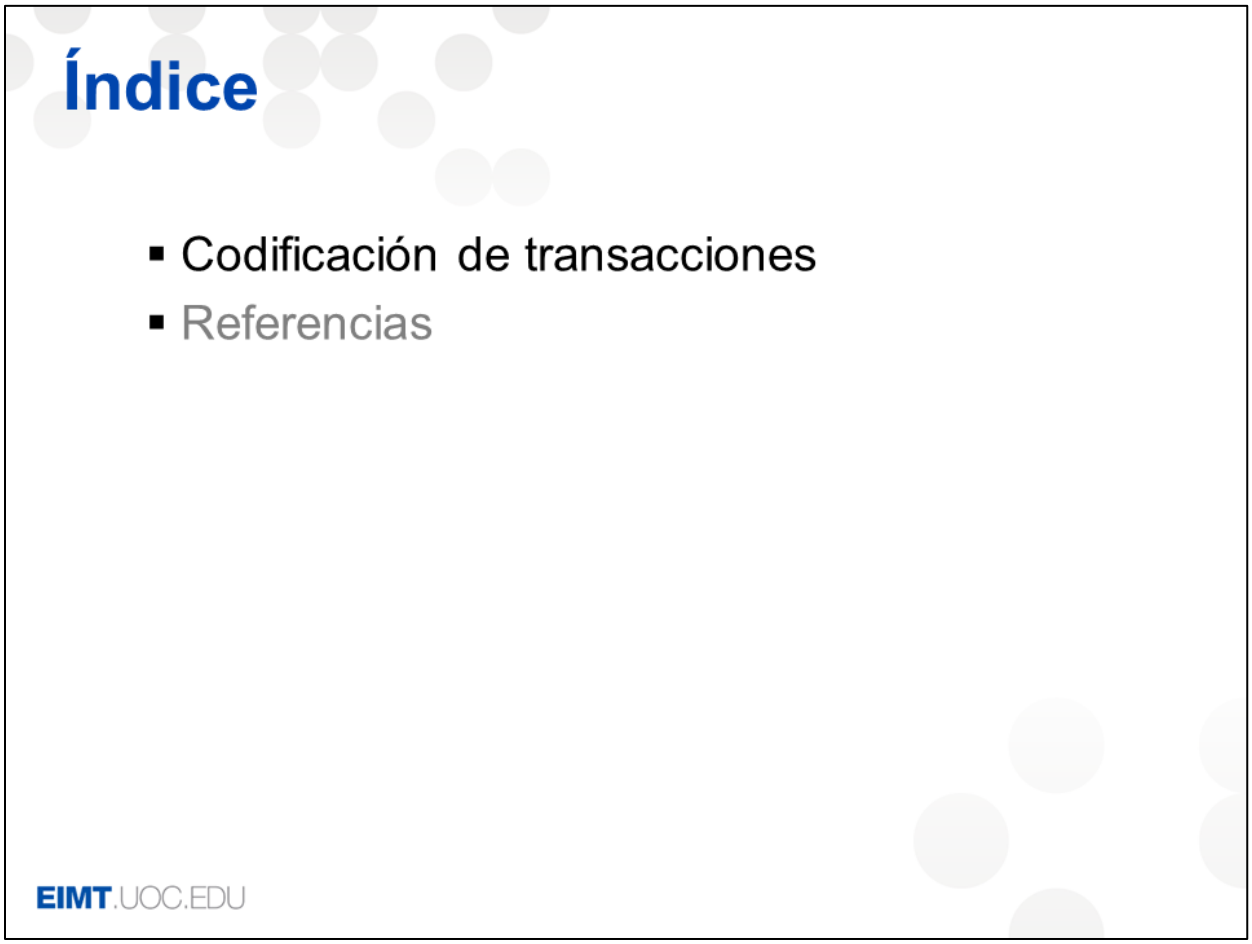
- *Codificación SQL*, centrada en aquellas prácticas desde un punto de vista genérico en SQL, que afectan a la legibilidad y portabilidad de código SQL.
- *Codificación de consultas*, en la que veremos prácticas que nos ayudarán a generar consultas SQL más eficientes y legibles.
- *Codificación de procedimientos 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.

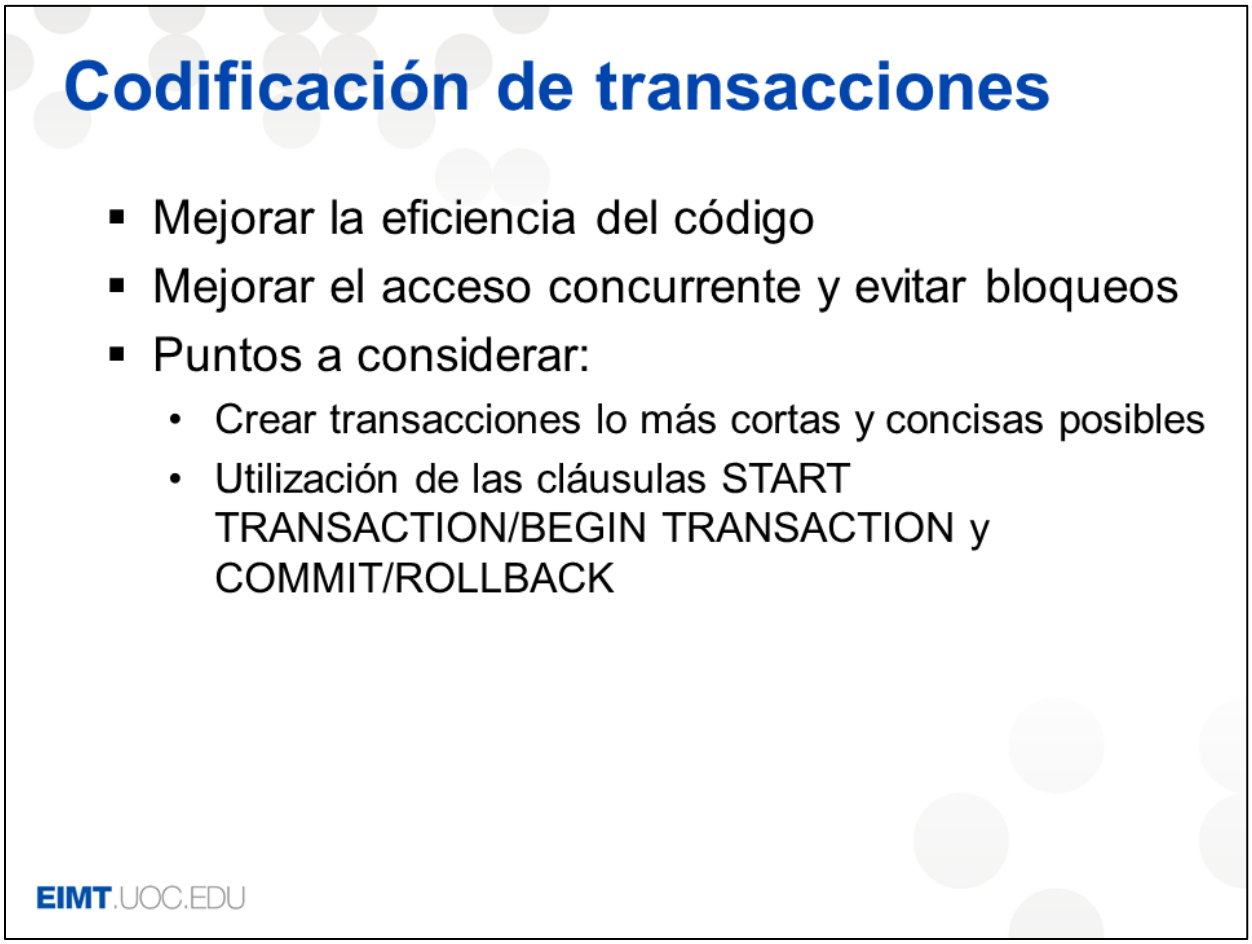
Este vídeo afrontará la última parte, *Codificación de transacciones*.



Este cuarto y último vídeo se centrará en presentar una serie de buenas prácticas de codificación de transacciones, que nos permitirán generar código eficiente y evitar problemas de concurrencia y bloqueos. Al final del vídeo, se proporcionarán las referencias bibliográficas utilizadas.



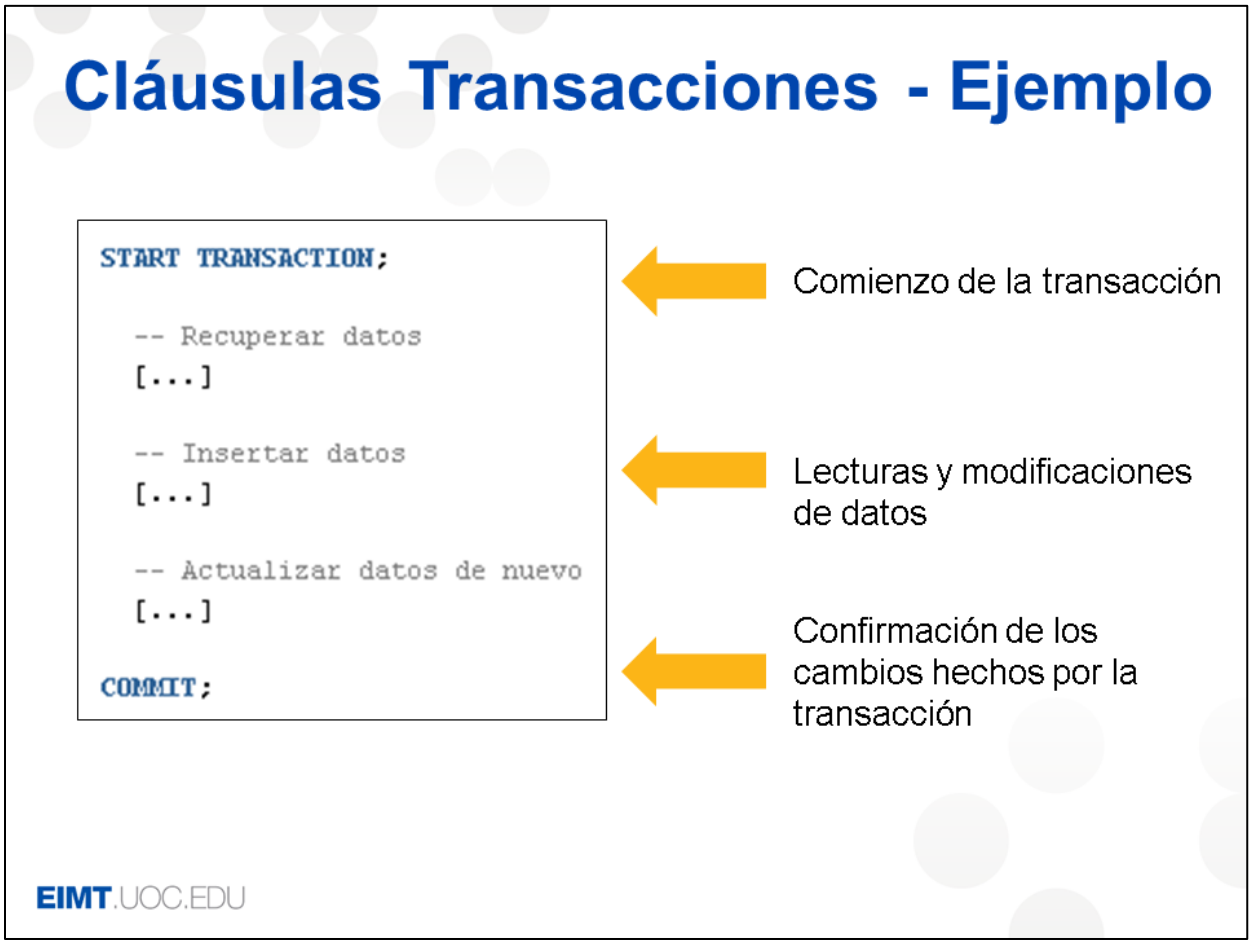
Vamos pues a ver la última categoría de buenas prácticas: codificación de transacciones.



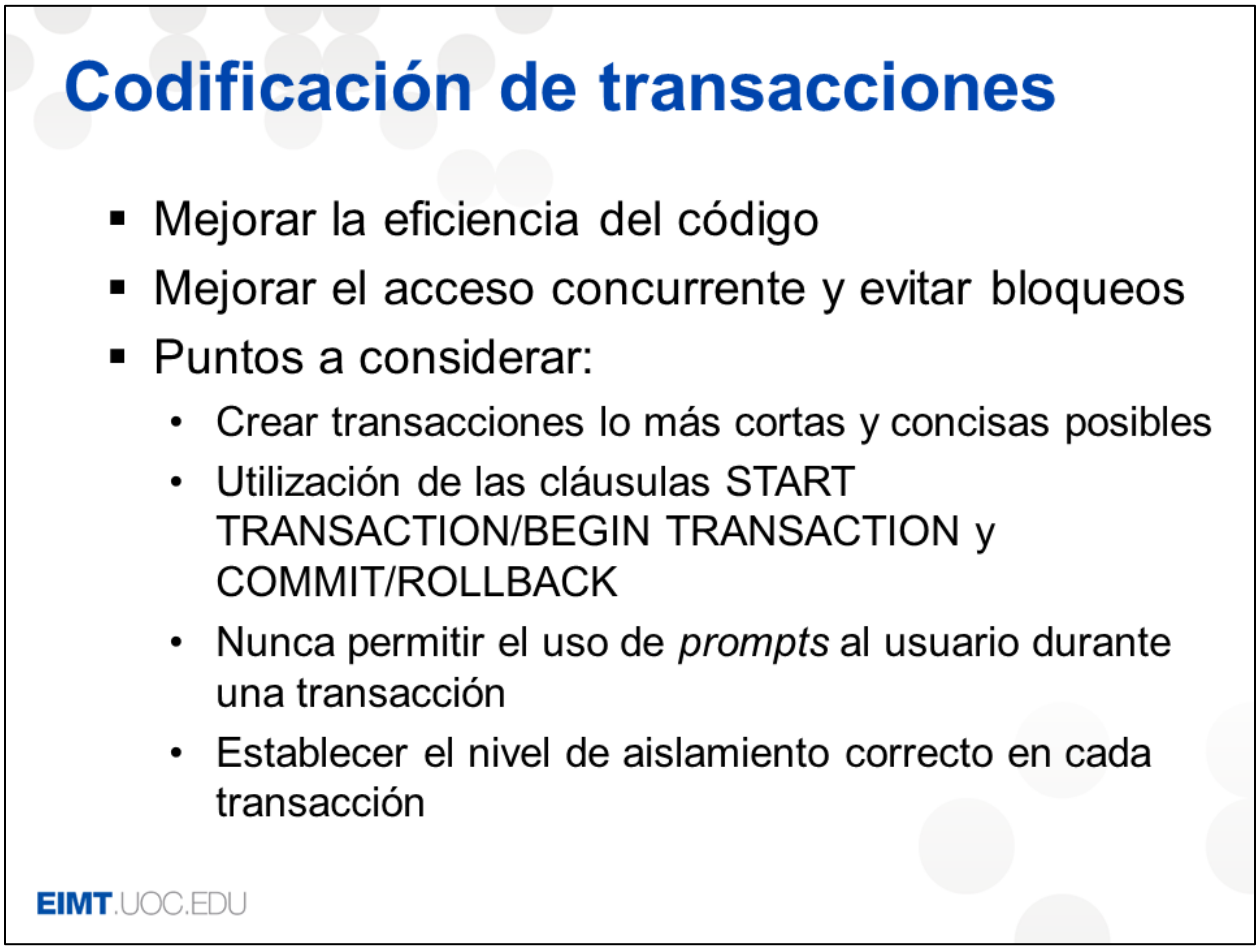
Cuando codificamos transacciones de manera ineficiente, esto puede tener un impacto en el acceso concurrente a los datos y causar así un pobre rendimiento global de la aplicación. Con el fin de mejorar la eficiencia del código y evitar problemas en el acceso concurrente, veremos a continuación una serie de puntos a tener en consideración a la hora de codificar transacciones en SQL:

- *Mantener las transacciones lo más cortas y concisas posibles*: la creación de transacciones cortas en tiempo nos evita problemas de bloqueos y dependencias con otras transacciones. Se recomienda no incluir código SQL dentro de una transacción que no es esencial para llevar a cabo dicha transacción, como por ejemplo consultas SQL de recuperación de datos que no se utilizan, ya que esto prolongará la transacción de forma innecesaria.

- *Utilización de las cláusulas START TRANSACTION/BEGIN TRANSACTION y COMMIT/ROLLBACK*: es muy importante que las transacciones esté bien delimitadas con los comandos `BEGIN` y `COMMIT`, o en su defecto, `ROLLBACK`. De esta forma, evitamos dejar transacciones abiertas y sufrir posibles pérdidas de datos.



Veamos un ejemplo de cómo utilizar estas cláusulas de manera sencilla. Como podemos ver en la imagen, vemos que se inicia una transacción con `START TRANSACTION`, en la que se ejecuta un código específico: recuperación de datos, inserción de datos y actualización de datos. Si este código se ejecuta de forma correcta, los cambios realizados por la transacción se confirmarán mediante la cláusula `COMMIT`, terminando así la transacción.



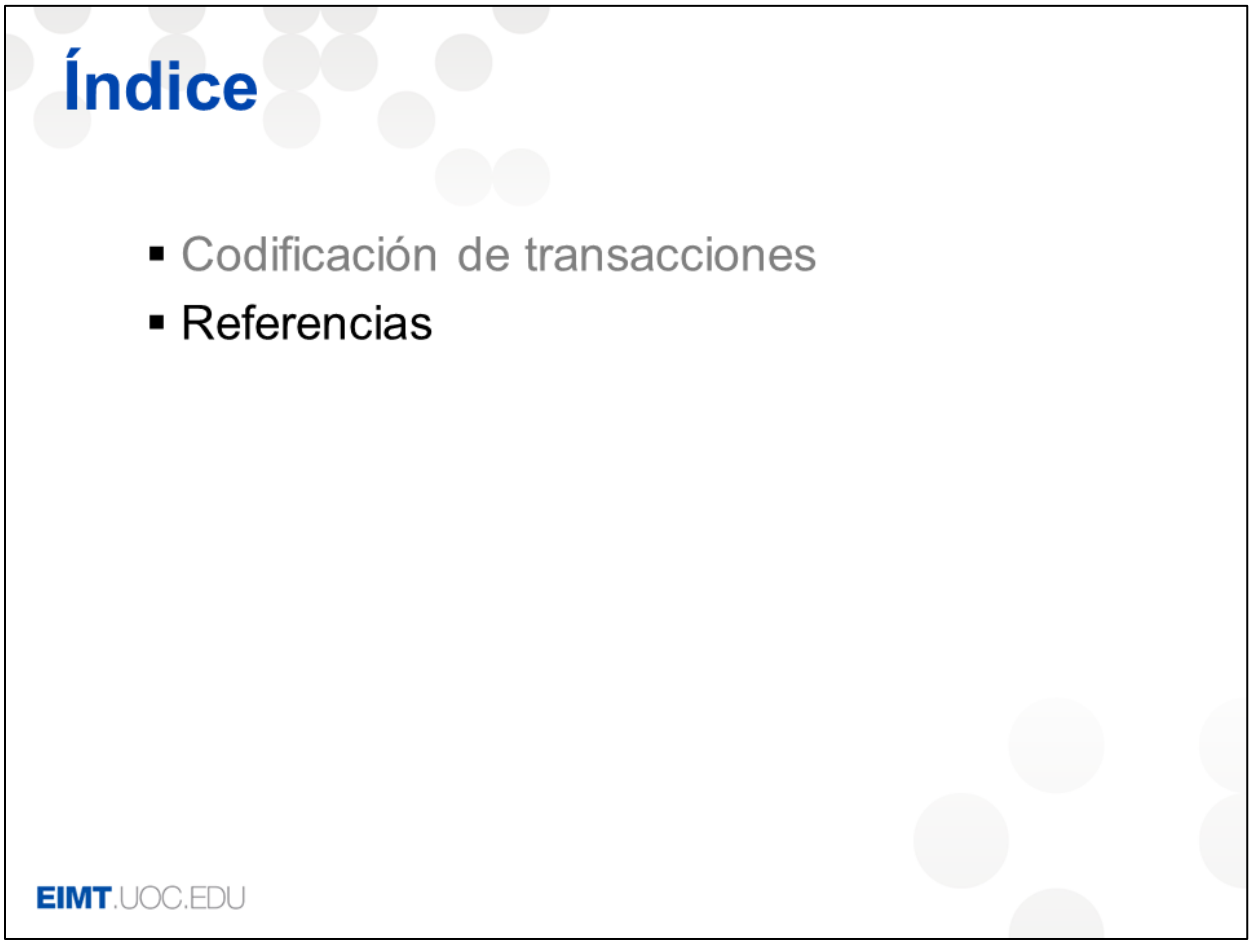
Como último punto a destacar:

- *Nunca permitir el uso de prompts al usuario en transacciones*: el hecho de requerir al usuario que proporcione algún valor en medio de una transacción causa que perdamos el control de la duración de la transacción, por lo que nunca debemos de permitirlo. Este tipo de acciones debe de realizarse fuera de una transacción.

  Imaginad que lanzáis una transacción que, en mitad de ésta, lanza un *prompt* que pide al usuario un código. Como no sabéis cuándo la transacción os pedirá el código, os vais a tomar un café con vuestros compañeros mientras la transacción se ejecuta, con tan mala suerte que justo cuando habéis dejado vuestro PC, la transacción os pide la entrada del código. Esta transacción estará abierta y esperando a que volváis, por lo que aquellas estructuras que habéis modificado estarán bloqueadas hasta que vuestra transacción finalice, por lo que otras transacciones estarán a la espera de los recursos que vuestra transacción ha bloqueado, causando problemas de concurrencia en la base de datos.

- *Establecer el nivel de aislamiento correcto en cada transacción*: una buena práctica a la hora de implementar transacciones es la de establecer el nivel de aislamiento correcto. En aquellos casos en

los que no sabemos si nuestras transacciones se ejecutarán concurrentemente con otras y el aislamiento es necesario, entonces se recomienda usar el nivel máximo. En cambio, si el aislamiento no nos preocupa o si bien sabemos de antemano qué otras transacciones se van a ejecutar con las nuestras, entonces siempre podemos definir el nivel de aislamiento más adecuado. Si lo podemos relajar a un nivel inferior, podremos mejorar el rendimiento de nuestra base de datos o aplicaciones.



Y hasta aquí hemos llegado con este último vídeo de la serie *Buenas prácticas en SQL*. Esperamos que os haya gustado la presentación, y en general la serie de vídeos, y que 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!