En el capítulo 4 se utilizó una cantidad considerable de tiempo para analizar la integridad de los datos y los métodos soportados por SQL para poder asegurar tal integridad. Estos métodos incluyen la creación de restricciones, dominios y afirmaciones, todos ellos utilizados por la base de datos de una manera u otra para asegurar que los datos SQL permanezcan bien fundamentados. Sin embargo, por sí solos, estos métodos no son siempre suficientes para mantener la integridad de esos datos. Tomemos, por ejemplo, la situación que puede surgir cuando más de un usuario intenta acceder y modificar datos en la misma tabla al mismo tiempo, o cuando sus acciones se superponen e impactan los mismos datos. Algunas acciones pueden ser tomadas por un usuario basándose en los datos que ya no son válidos como resultado de las acciones tomadas por el otro usuario. Los datos pueden convertirse en inconsistentes o inexactos, sin que ninguno de los dos usuarios se dé cuenta de que el problema existe. Para encargarse de las situaciones de este tipo, SQL soporta el uso de transacciones para asegurarse que las acciones actuales no impacten la integridad de los datos que están siendo revisados por cualquier usuario. En este capítulo se describirá cómo se implementan las transacciones en un ambiente SQL y cómo puede controlarse su comportamiento. El usuario aprenderá cómo configurar las propiedades de las transacciones, iniciar transacciones, finalizarlas y utilizar otras opciones que extiendan su funcionalidad. Entender las transacciones SQL Relativamente, existen muy pocas bases de datos en las cuales solamente un usuario esté intentando acceder a los datos contenidos en ellas en algún momento dado. La mayoría de las veces, las bases de datos son utilizadas por diferentes tipos de usuarios para muy distintos propósitos, y a menudo estos usuarios están intentando acceder a los datos al mismo tiempo. Mientras mayor sea el número de usuarios, mayor será la probabilidad de que existan problemas cuando los usuarios intenten ver o modificar los mismos datos en el mismo momento. Sin embargo, dependiendo de la naturaleza de las operaciones, los problemas pueden surgir incluso si solamente dos usuarios están accediendo a los datos al mismo tiempo. Por ejemplo, un usuario pudiera estar viendo los datos en una tabla, tomar cierto tipo de acción basada en esos datos, y luego arrojarlos a la tabla para verificar los datos una vez más. Sin embargo, si otro usuario actualiza la tabla entre los dos momentos que el primer usuario la visualiza, el primer usuario encontrará datos diferentes la segunda vez que

lo hace, y pudiera incluso notar que la acción tomada por el segundo usuario invalidó los cambios que ambos hicieron después de haber visto la tabla por primera vez. Por ejemplo, el primer usuario pudiera haber notado que el número telefónico de un cliente es incorrecto y quisiera aplicar la corrección. Sin embargo, un segundo usuario pudiera estar buscando los datos del mismo cliente, y mientras éste actualiza el estatus del crédito del cliente, pudiera sin darse cuenta regresar el anterior número telefónico al registro de la base de datos (debido a que los datos que se estaban buscando contenían el número anterior), sobrescribiendo los cambios que realizó el primer usuario. Para encargarse de este tipo de inconsistencias en los datos, SQL utiliza transacciones para controlar las acciones de los usuarios individuales. Una transacción es una unidad de trabajo que se compone de una o más instrucciones SQL que realizan un conjunto de acciones relacionadas. Por ejemplo, la aplicación podría utilizar una transacción para cambiar el número de CD de las existencias. El proceso de actualizar la tabla o tablas aplicables y reportar la información actualizada de regreso al usuario es tratado como una sola transacción. La transacción puede incluir varias instrucciones SQL, realizando cada una de ellas una tarea específica. Para que un conjunto de acciones califique como una transacción, debe pasar la prueba ACID. ACID es el acrónimo comúnmente utilizado para referirse a los nombres en inglés de las cuatro características de una transacción (Atomic, Consistent, Isolated y Durable, respectivamente) que veremos a continuación: Atómica Esta característica se refiere a la naturaleza todo-o-nada de una transacción. Se realizan ya sea todas las operaciones en una transacción, o ninguna de ellas. Aunque algunas instrucciones sean ejecutadas, los resultados de éstas regresan a su punto inicial si la transacción falla en cualquier punto antes de ser completada. Solamente cuando todas las instrucciones se ejecutan apropiadamente y todas las acciones se realizan, se considera completa una transacción y sus resultados se aplican a la base de datos. Consistente La base de datos debe ser consistente al inicio y al final de la transacción. De hecho, se puede considerar una transacción como un conjunto de acciones que lleva a la base de datos de un estado consistente a otro. Todas las reglas que definen y limitan los datos deben ser aplicadas a esos datos como resultado de cualquier cambio que ocurra durante la transacción. Además, todas las estructuras dentro de la base de datos deben estar correctas al final de la transacción. Aislada (Isolated) Los datos que pudieran encontrarse en un estado inconsistente temporalmente durante una transacción no deberán estar disponibles a otras transacciones hasta que los datos sean consistentes una vez más. En otras palabras, ningún usuario deberá ser capaz de acceder a los datos inconsistentes durante una transacción implementada por otro usuario cuando los datos impactados por esa transacción están en un estado inconsistente. Además, cuando una transacción se encuentra aislada, ninguna otra transacción puede afectarla. Durable Una vez que los cambios hechos en una transacción sean completados, esos cambios deberán ser preservados, y los datos deberán estar en un estado confiable y consistente, incluso si ocurren errores de aplicación o de hardware. Si surge cualquier problema en cualquier momento durante una transacción, la transacción completa regresa a su punto inicial y la base de datos regresa al estado en que se encontraba antes de que la transacción iniciara. Cualquier acción que se tome es cancelada y los datos se restauran a su estado original. Si la transacción se completa exitosamente, entonces todos los cambios son im-

plementados. A través de todo el proceso, la transacción siempre asegura la integridad de la base de datos, sin importar si la transacción fue completada exitosamente o debió regresar a su punto inicial. SQL soporta diferentes instrucciones relacionadas con el proceso de transacción. Estas transacciones pueden ser utilizadas para iniciar y finalizar transacciones, configurar sus propiedades, aplazar la ejecución de las restricciones durante la transacción e identificar los puntos dentro de una transacción que actúan como puntos de recuperación cuando las transacciones vuelven a su punto inicial. En el resto de este capítulo examinaremos cómo se utiliza cada una de estas instrucciones dentro de una transacción. Sin embargo, antes de ir a una discusión más detallada de las instrucciones, sería mejor proporcionar un breve repaso de cada una de ellas para tener un mejor entendimiento de cómo funcionan las transacciones. El estándar SQL:2006 define siete instrucciones relacionadas al proceso de transacción:

SET TRANSACTION Configura las propiedades de la siguiente transacción que deberá ser ejecutada.

START TRANSACTION Configura las propiedades de una transacción e inicia esa transacción.

SET CONSTRAINTS Determina el modo de restricción dentro de una transacción actual. El modo de restricción se refiere a si una restricción es aplicada inmediatamente a los datos cuando éstos son modificados o si la aplicación de la restricción es aplazada hasta un punto posterior en la transacción.

SAVEPOINT Crea un punto de recuperación dentro de una transacción. Un punto de recuperación marca una zona dentro de la transacción que actúa como un punto para detenerse cuando una transacción tiene que regresar a su punto inicial. RELEASE SAVEPOINT Libera un punto de recuperación.

ROLLBACK Finaliza una transacción y reinvierte todos los cambios al comienzo de la transacción o a un punto de recuperación. Ç

COMMIT Finaliza una transacción y permite completar todos los cambios a la base de datos. A pesar de que todas estas siete instrucciones serán vistas con más detalle, algunas de ellas son esenciales para entender la naturaleza de una transacción. Veamos la figura 16-1 para ayudar a ilustrar este punto.

NOTA Como se verá posteriormente en este capítulo, sería raro en un ambiente SQL puro que se quisieran utilizar ambas instrucciones: SET TRANSACTION y START TRANSACTION, debido a que ambas instrucciones establecen las mismas propiedades. Sin embargo, se encontrará que las implementaciones SQL varían respecto a cuáles instrucciones relacionadas con las transacciones soportan y cómo implementan esas instrucciones. Se señalarán diferencias específicas a lo largo de este capítulo.

Cuando se inicia la transacción, la base de datos se encuentra en su estado original (los datos son consistentes y correctos).

Después se procesan las instrucciones SQL dentro de la transacción. Si este proceso es exitoso, se ejecuta una instrucción COMMIT.

La instrucción COMMIT provoca que la implementación SQL actualice la base de datos y finalice la transacción. Si el proceso de ejecución de la instrucción no es exitoso, se ejecuta una instrucción ROLLBACK y la implementación regresa la base de datos a su estado original.

Una ejecución no exitosa no significa necesariamente que las instrucciones hayan fallado. Una instrucción ROLLBACK puede ser ejecutada de acuerdo con las condiciones de una cláusula WHERE, un error predefinido, o por cualquier otra condición que sea definida dentro de la transacción.

El punto es que, bajo ciertas circunstancias, se ejecuta la instrucción ROLLBACK, y bajo otras circunstancias se ejecuta la instrucción COMMIT.

Configurar las propiedades de la transacción La primera instrucción que veremos a detalle es la instrucción SET TRANSACTION.

La instrucción SET TRANSACTION permite configurar la mayoría de las propiedades asociadas con el proceso de transacción. Se puede ejecutar esta instrucción solamente cuando ninguna transacción está activa (o en el caso de Oracle, sólo como la primera instrucción después de COMMIT o ROLLBACK).

Cuando sí se utiliza una instrucción SET TRANSACTION, las propiedades configuradas dentro de la instrucción son aplicadas solamente a la siguiente transacción que es iniciada. Las propiedades no se trasladan de una transacción a la otra.

La instrucción SET TRANSACTION no es obligatoria para iniciar una transacción. Si la instrucción no es ejecutada, la transacción utiliza ya sea las propiedades de manera predeterminada, o las propiedades suministradas en la instrucción subsecuente START TRANSACTION. Si se ejecuta la instrucción SET TRANSACTION, la transacción utiliza las propiedades especificadas en esa instrucción. Si la instrucción se ejecuta pero no se definen todas las propiedades, la transacción utiliza los valores de forma preestablecida para las propiedades no definidas. Sin importar cuáles propiedades sean configuradas, ninguna de ellas es aplicable a ninguna transacción, excepto a la primera iniciada después de que SET TRANSACTION es ejecutada. Ahora demos un vistazo a la sintaxis utilizada para una instrucción SET TRANSACTION. En su forma más básica, la sintaxis luce de la siguiente manera: SET [LOCAL] TRANSACTION [{ , } ...]

El primer aspecto que deberá notar acerca de esta sintaxis es la palabra clave opcional LOCAL. La palabra clave LOCAL aplica solamente a las transacciones que comprenden múltiples implementaciones SQL.

Si se está trabajando con este tipo de transacciones, se puede utilizar la opción LOCAL para aplicar las propiedades a la porción local de la transacción. Para poder utilizar la opción LOCAL, la transacción debe haber sido iniciada en un servidor SQL diferente a aquél donde las propiedades de transacción locales sean configuradas.

NOTA El tema de abarcar transacciones y propiedades locales excede el campo de acción de este libro. Se mencionan aquí solamente para proporcionar una imagen completa de la instrucción SET TRANSACTION.

Como un programador SQL principiante, es muy probable que no tenga que preocuparse por abarcar las transacciones.

Además, el soporte para abarcar las transacciones varía a través de las diferentes implementaciones SQL. Por ejemplo, Oracle no las soporta, y mientras que MySQL sí, sólo utiliza los términos global y sesión.

Regresando a la sintaxis de SET TRANSACTION, se puede ver que el único tipo diferente de opción que se necesita especificar es aquel representado por el marcador de posición . Existen tres tipos de modos de transacción que se pueden especificar: Nivel de acceso Nivel de aislamiento Tamaño de diagnóstico Se deben especificar uno o más modos de transacción. Si se especifica más de uno, deben quedar separados por comas. Además, no se puede incluir más de uno del mismo tipo de los modos de transacción. Por ejemplo, es posible especificar un nivel de acceso y un nivel de aislamiento, pero no es posible especificar dos niveles de aislamiento.

La instrucción SET TRANSACTION soporta dos opciones del nivel de acceso: READ ONLY y READ WRITE. Si se selecciona la opción READ ONLY, no se puede incluir ninguna instrucción dentro de la transacción que modifique la base de datos.

Esto incluye a las instrucciones que modifican datos (por ejemplo, la instrucción UPDATE) y a las instrucciones que modifican la estructura de la base de datos (por ejemplo, la instrucción CREATE TABLE).

Si se selecciona la opción READ WRITE, se pueden ejecutar ambos tipos de instrucciones en la transacción. Como se verá en la siguiente sección, “Especificar un nivel de aislamiento”, el nivel de acceso depende predeterminadamente del nivel de aislamiento. Sin embargo, si no se especifica ningún nivel de aislamiento y ningún nivel de acceso, el nivel de acceso por defecto es READ WRITE.

Especificar un nivel de aislamiento Cuando se crea una instrucción SET TRANSACTION, se pueden especificar los niveles de aislamiento cero o uno. Un nivel de aislamiento define cómo sería aislar una transacción desde las acciones de otras transacciones. Una instrucción SET TRANSACTION soporta cuatro opciones del nivel de aislamiento: READ UNCOMMITTED READ COMMITTED

REPEATABLE READ SERIALIZABLE Los niveles de aislamiento están enlistados del menos restrictivo al más restrictivo, siendo la opción READ UNCOMMITTED la menos efectiva en términos de aislar los datos, y la opción SERIALIZABLE la más efectiva.

Si no se especifica ningún nivel de aislamiento, se asume SERIALIZABLE de manera preestablecida. NOTA El soporte para los niveles de aislamiento varía entre las diferentes implementaciones de SQL.

Por ejemplo, Oracle soporta solamente las opciones READ COMMITTED y SERIALIZABLE, mientras que SQL Server soporta las cuatro opciones definidas en el estándar SQL, además de añadir una nueva llamada SNAPSHOT que proporciona consistencia de lectura para todas las instrucciones en una transacción al igual que al inicio de la transacción. Irregularidades de los datos La mejor manera de comprender los niveles de aislamiento es dar un vistazo a los tipos básicos de irregularidades que pueden ocurrir a los datos, dependiendo de qué tan aislada está una transacción de la otra.

En general, pueden ocurrir tres tipos de irregularidades: Lecturas sucias Lecturas no repetibles Lecturas fantasma El tipo de irregularidad de datos que puede experimentarse durante una transacción depende de cuál nivel de aislamiento se configure para la transacción. Sin embargo, antes de internarnos en mayores especificaciones acerca de los niveles de aislamiento, demos un vistazo a estos tres tipos de irregularidades. Lecturas sucias La primera irregularidad que veremos es la lectura sucia.

Una lectura sucia puede ocurrir cuando una transacción modifica los datos, una segunda transacción ve esas modificaciones antes de que sean realmente completadas en la base de datos, y después la primera transacción retira las modificaciones, regresando a la base de datos a su estado original.

Sin embargo, la segunda transacción, habiendo leído los datos modificados, pudo haber tomado alguna acción basada en los datos incorrectos. Para ayudar a ilustrar el concepto de una lectura sucia, veamos la figura 16-2, que muestra dos transacciones operando al mismo tiempo. Cuando comienza Transacción 1, ésta lee la tabla en su estado original. Luego, la transacción actualiza la tabla, cambiando el valor EXISTENCIA de cada fila. Después de que se realizan los cambios, Transacción 2 es iniciada y lee los datos actualizados. Por ejemplo, Transacción 2 verá que el valor EXISTENCIA para la fila Past Light es 11. Basada en esa información, Transacción 2 toma algún tipo de acción, por ejemplo, ordenar los CD adicionales de Past Light. Después de que Transacción 2 ha leído los datos de la tabla, Transacción 1, por una u otra razón, reinvierte la actualización, y la base de datos regresa a su estado original. Como resultado, la fila Past Light ahora tiene en realidad un valor EXISTENCIA de 22, a pesar de que Transacción 2 piensa que ésta tiene

Lecturas no repetibles La siguiente irregularidad que puede ocurrir cuando se inician transacciones simultáneas es la lectura no repetible. La lectura no repetible puede ocurrir cuando una transacción lee datos desde una tabla, luego otra transacción actualiza la tabla, y la primera transacción vuelve a leer los datos, sólo para descubrir que los datos han sido cambiados.

Como resultado, la primera lectura es no repetible. Veamos la figura 16-3 para comprender mejor este concepto. Cuando se inicia Transacción 1, ésta lee los datos en la tabla. En ese punto, la transacción podría estar involucrada en otros procesos o estar esperando la respuesta de un usuario. Por ejemplo, el usuario pudiera recibir una llamada de un administrador que está tratando de averiguar cuántas copias hay en existencia de un CD en particular.

El usuario verifica esa información. Entonces el administrador pone al usuario en espera durante un corto tiempo, por lo que el usuario debe esperar para completar la transacción. Durante ese tiempo, se inicia Transacción 2 y actualiza la tabla.

Después de la actualización, Transacción 1 vuelve a leer los datos (el administrador regresa al teléfono) y encuentra información diferente a la de la primera lectura, dando como resultado una lectura no repetible.

Lecturas fantasma La última irregularidad que veremos es la lectura fantasma. A pesar de ser parecida a la lectura no repetible, la lectura fantasma tiene algunas diferencias sutiles, que se convierten en factores cuando se intenta determinar un nivel de aislamiento. Una lectura fantasma puede ocurrir cuando una transacción lee una tabla basada en algún tipo de condición de búsqueda, después una segunda transacción actualiza los datos en la tabla, y la primera transacción intenta volver a leer los datos, sólo que esta vez se arrojan diferentes filas debido a como está definida la condición de búsqueda. Para dejar esto más claro, demos un vistazo a la figura 16-4.

Cuando se inicia Transacción 1, ésta lee los datos en la tabla ejecutando una instrucción SELECT que consulta los datos. La instrucción incluye una cláusula WHERE que arroja solamente aquellas filas con un valor EXISTENCIA mayor a 20. Eso significa que la fila Court and Spark y la fila Past Light son arrojadas. Después de que Transacción 1 recupera (lee) los datos, se inicia Transacción 2 y actualiza la tabla. Ahora, cuando Transacción 1 vuelve a leer los datos (utilizando los mismos criterios de búsqueda), solamente la fila Famous Blue Raincoat es arrojada debido a que ahora es la única fila con un valor EXISTENCIA mayor a 20. Como resultado, la transacción ha experimentado una lectura fantasma, y las filas que está leyendo no son las mismas filas que ésta había visto anteriormente. Escoger un nivel de aislamiento Ahora que se tiene una vista más general acerca de los tipos de irregularidades de datos con los que es posible toparse cuando se tienen transacciones simultáneas, se deberá estar mejor preparado para escoger un nivel de aislamiento para la transacción. El punto importante a recordar es que mientras más restrictivo sea el nivel de aislamiento, más tipos de irregularidades pueden ser eliminadas. Demos un vistazo al nivel de aislamiento READ UNCOMMITTED, el menos restrictivo de los cuatro niveles. Una transacción configurada con esta opción puede experimentar cualquiera de las irregularidades de datos que se vieron anteriormente (lectura sucia, lectura no repetible y lectura fantasma). Como es fácil imaginarse, normalmente éste no es un estado deseable. De hecho, si se define una transacción con la opción READ UNCOMMITTED, la transacción no puede incluir instrucciones que modifiquen datos. Estas transacciones, de manera predeterminada, solamente

tienen un nivel de acceso de sólo lectura (READ ONLY), y no es posible especificar un nivel READ WRITE. (Todos los demás niveles de aislamiento tienen un nivel de acceso preestablecido de READ WRITE.) El nivel de aislamiento READ UNCOMMITTED deberá utilizarse solamente para transacciones que generen información aproximada, por ejemplo, algunos tipos de datos estadísticos en los que los resultados no son muy críticos en términos de precisión. El nivel de aislamiento READ COMMITTED es sólo ligeramente más restrictivo que READ UNCOMMITTED. La opción READ COMMITTED evita las lecturas sucias, pero las lecturas no repetibles y las lecturas fantasma aún pueden ocurrir. La siguiente opción, REPEATABLE READ, es más restrictiva que READ COMMITTED. Evita las lecturas sucias y las lecturas no repetibles, pero no evita las lecturas fantasma. La única opción que evita los tres tipos de irregularidades en los datos es SERIALIZABLE. Una transacción que está definida con el nivel de aislamiento SERIALIZABLE puede aislar completamente a esa transacción de todas las otras transacciones. Como resultado, se dice que la transacción se puede serializar, lo que significa que interactúa con transacciones simultáneas en una forma en que ordena las transacciones secuencialmente para que una transacción no pueda impactar a la otra. Esto no significa que una transacción deba cerrarse antes de que otra pueda abrirse, pero sí significa que los resultados de esas transacciones tienen que ser iguales a los resultados de las operaciones que se realizan una a la vez. Mientras ninguna transacción que se puede serializar puede influir sobre otra transacción que se puede serializar, las transacciones estarán en conformidad con el nivel de aislamiento SERIALIZABLE.