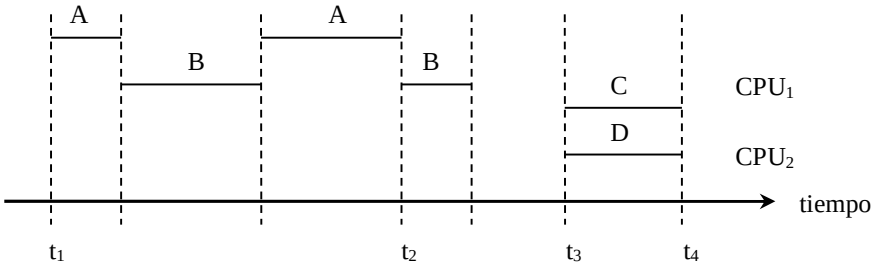
5 Transacciones y concurrencia

5.1 Introducción

Esta sección describe varios conceptos iniciales relacionados con las transacciones.

Multiprogramación

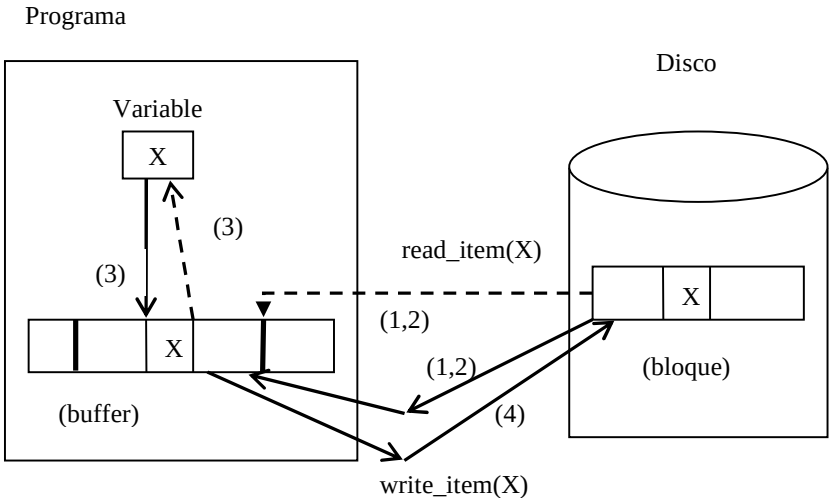
- Permite a la computadora procesar simultáneamente (concurrentemente) varios programas (o transacciones).
- Si la computadora tiene un solo procesador, la ejecución concurrente de los programas es *intercalada* (*interleaved*).
- Si la computadora tiene varios procesadores, es posible el procesamiento *simultáneo* de varios programas.
- La suspensión de la ejecución de un programa se hace por varios motivos: el programa espera la terminación de un operación de E/S, se termina su tiempo de ejecución asignado, otro programa de mayor prioridad lo interrumpe, etc.



- En bases de datos, los datos almacenados son los recursos primarios que pueden ser accedidos concurrentemente por programas. En bases de datos se le llama **transacción** a *la ejecución de un programa* que lee o modifica el contenido de la base de datos.

Operaciones de lectura/escritura de una transacción

- Las operaciones de acceso que una transacción puede incluir, a nivel de *elementos de datos*[1] y de *bloques de disco*, pueden representarse, genéricamente, como:
  - **read_item(X)**: lee un elemento X desde la base de datos en una variable de programa que, por simplicidad, también se llamará X.
  - **write_item(X)**: escribe el valor de una variable de programa X en un elemento X de la base de datos.



- Ejecutar la operación **read_item**(X) implica:
  1. Encontrar la dirección del bloque de disco que contiene al elemento X.
  2. Copiar el bloque a un buffer en memoria central (si no está ya ahí).
  3. Copiar el elemento X a la variable de programa X.
- Ejecutar la operación **write_item**(X) implica:
  1. Encontrar la dirección del bloque de disco que contiene al elemento X.
  2. Copiar el bloque a un buffer en memoria central (si no está ya ahí).
  3. Copiar la variable X al elemento X en su lugar correcto en el buffer.
  4. Regresar el bloque actualizado del buffer al disco (inmediatamente o en un tiempo posterior).

- Ejemplos de dos transacciones (la primera podría representar una transferencia monetaria entre dos cuentas dadas por X y Y; y la segunda un depósito a la primera cuenta X):

T_1:	read_item(X); X= X-N;	T_2:	read_item(X);
	write_item(X);		X= X+M;
	read_item(Y); Y= Y+N;		write_item(X);
	write_item(Y);		

Nota: los mismos programas pueden ser usados para ejecutar muchas de estas transacciones T_1 y T_2. Una transacción, en este caso, es una ejecución particular de uno de los programas con una fecha, hora, cuenta(s) y monto(s) específico(s).

[1] Un elemento de datos puede ser: una tupla, un conjunto de tuplas, un bloque de disco, una tabla, un conjunto de tablas o la base de datos completa.

Problemas que pueden surgir con las transacciones

- Varios problemas importantes pueden surgir con las dos transacciones anteriores:
  - **Actualización perdida**: sucede cuando T_2 lee un valor antes de que T_1 registre en la base de datos un cambio que le hizo. El resultado es que el valor actualizado por T_1 se pierde.

T_1	T_2	
read_item(X); X= X-N;	read_item(X); X= X+M;	tiempo ↓
write_item(X); read_item(Y);		
Y= Y+N; write_item(Y);	write_item(X); ←	X tiene un valor incorrecto debido a que su actualización por T_1 se "perdió" (se sobre-escribió).

  - **Actualización temporal (dirty read)**: sucede cuando una transacción lee cambios hechos por otra transacción sin haberse confirmado. Existe el riesgo de que estos cambios puedan quedar sin confirmación, con lo que la primera transacción habrá leído y trabajado con datos incorrectos.

	T_1	T_2

tiempo ↓	read_item(X);
	X= X-N;
	write_item(X);	

		read_item(X);
		X= X+M;
		write_item(X);

	read_item(Y);	
T_1 falla y debe regresar a X a su valor anterior; mientras, T_2 ha leído un valor incorrecto "temporal" de X. →		

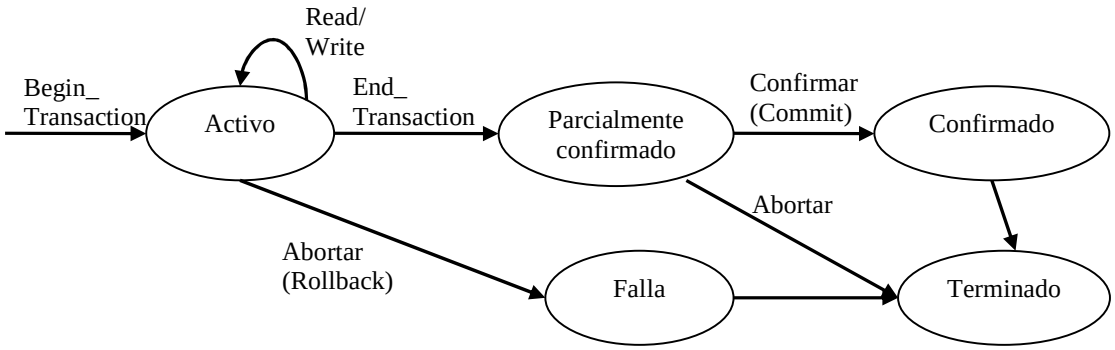
  - **Lectura repetible (repeatable read)**: las lecturas repetibles se garantizan si los datos leídos por una transacción se bloquean contra cambios hechos por otras transacciones, hasta que la transacción actual termina. Así, la transacción actual puede suponer que si lee los mismos datos dos veces, obtendrá los mismos resultados.

  - **Lectura de fantasmas (phantom read)**: los fantasmas son elementos de datos no detectados, agregados a la base de datos por otra transacción mientras la transacción actual está en progreso. Estos datos no son detectados debido a que la transacción actual está usando los resultados de una consulta hecha al inicio de la misma.

5.2 Conceptos de transacciones

Aquí se introducen conceptos más formales acerca de elementos que se utilizan para el manejo de las transacciones.

- Una **transacción** es una unidad atómica de trabajo que se realiza por completo o bien, no se efectúa en absoluto.
- **Recuperación:** es un proceso que debe efectuar el DBMS para llevar a la base de datos hasta el último *estado consistente* que tenía antes de la ocurrencia de una falla.
- Para fines de recuperación, se necesita llevar el registro de las siguientes operaciones:



  - Begin_Transaction: marca el inicio de la ejecución de la transacción.
  - Read o Write: especifican las operaciones de lectura/escritura sobre los elementos de la base de datos, que son ejecutadas como parte de la transacción.
  - End_Transaction: especifica que las operaciones anteriores han terminado y marca el final de la ejecución de la transacción. En este punto todavía falta verificar que los cambios a la base de datos puedan ser aplicados o que la transacción deba abortarse por alguna causa.
  - Commit: señala el *fin exitoso* de la transacción tal que cualquier cambio ejecutado por la transacción puede ser seguramente **confirmado (commited)** en la base de datos y no será deshecho.
  - Rollback (o Abort): señala que la transacción ha *terminado sin éxito*, tal que cualquier cambio o efecto que la transacción pudiera haber aplicado a la base de datos debe ser *deshecho*.
- **Bitácora**: registra todas las operaciones de las transacciones que afectan los valores de los elementos de la base de datos (incluyendo lecturas). Esta información puede ser necesaria para permitir la recuperación por fallas de una transacción.
- Una transacción T alcanza su *punto de confirmación (commit point)* cuando todas sus operaciones que acceden a la base de datos han sido ejecutadas exitosamente *y* el efecto de todas las operaciones de la transacción sobre la base de datos ha sido registrado en la bitácora.
  Después de este punto, la transacción se dice que está *confirmada (commited)* y se considera que sus efectos están *permanentemente registrados* en la base de datos.

5.2.1 Propiedades deseables de una transacción

- Propiedades **ACID**
  - **Atomicity**: una transacción es una unidad atómica de procesamiento; se ejecuta totalmente o no se ejecuta en absoluto.
  - **Consistency preservation**: una ejecución correcta de la transacción debe llevar a la base de datos de un estado consistente a otro.
  - **Isolation**: una transacción no debe hacer visibles sus actualizaciones a otras transacciones hasta que sea confirmada (esta propiedad resuelve la actualización temporal y las reversiones (rollback) en cascada).
  - **Durability**: una vez que una transacción cambia la base de datos y los cambios son confirmados, éstos nunca deben perderse por fallas subsecuentes.

5.2.2 Planes y seriabilidad

Un **plan** (o **historia**) S de n transacciones T_1, T_2,…, T_n es un ordenamiento de las operaciones de las transacciones sujeto a la restricción de que, para cada transacción T_i que participe en el plan S, las operaciones de T_i en S deben aparecer en el mismo orden en el cual ellas ocurren en T_i.

Los ejemplos anteriores de actualización perdida y lectura sucia son dos posibles planes para las transacciones T_1 y T_2.

Un plan S es **serial** si, para cada transacción T participante en el plan, todas las operaciones de T son ejecutadas consecutivamente en el plan; en caso contrario, el plan es llamado **no serial**.

Ejemplo serial: primero todo T_1 y luego todo T_2 (o primero todo T_2 y luego todo T_1):

T_1	T_2

read_item(X);
X= X-N;
write_item(X);
read_item(Y);
Y= Y+N;
write_item(Y);
	
	read_item(X);
	X= X+M;
	write_item(X);

Ejemplos no seriales: el de actualización perdida o el de lectura sucia.

Se puede suponer razonablemente, si se considera que las transacciones son *independientes*, que *cada plan serial es considerado correcto*.
Un plan S de *n* transacciones es **seriable** si es *equivalente a algún plan serial* de las mismas *n* transacciones; esto es, si produce los mismos resultados que el plan serial.
El siguiente esquema muestra un plan no serial de las anteriores transacciones T_1 y T_2 que es *seriable,* o sea, es equivalente al plan serial mostrado en el esquema del párrafo anterior:

T_1	T_2

read_item(X);
X= X-N;
write_item(X);
 	
read_item(Y);
Y= Y+N;
write_item(Y);
	
	read_item(X);
	X= X+M;
	write_item(X);

Decir que un plan S no serial es **seriable** es equivalente a decir que es **correcto**, debido a que es equivalente a un plan serial, el cual es considerado correcto.

5.3 Transacciones en Oracle

El estándar de SQL92 define cuatro niveles de aislamiento de transacciones con diferentes grados de impacto sobre el rendimiento del procesamiento de las transacciones. Estos cuatro niveles se muestran en la siguiente tabla en términos de los posibles problemas que se pueden presentar durante una transacción.

Nivel de aislamiento	Lectura sucia	Lectura no repetible	Lectura fantasma
Read uncommitted	Posible	Posible	Posible
Read committed	No posible	Posible	Posible
Repeatable read	No posible	No posible	Posible
Serializable	No Posible	No Posible	No Posible

Oracle ofrece los niveles *Read committed* y *Serializable*, siendo el primero el default; ofrece también un modo *read-only* que no es parte del estándar.

Consistencia de lectura a nivel de instrucción

Es un término que se usa en Oracle para indicar que los datos regresados por una **sola** consulta provienen desde un solo punto en el tiempo, el tiempo en que la consulta inicia. Por lo tanto, una consulta nunca ve datos sucios, ni cambios hechos por transacciones que se confirman durante la ejecución de la consulta. Otra forma de decir lo anterior es que las consultas se ven como transacciones atómicas, esto es, que no son interrumpidas: se inician y confirman como si fueran indivisibles.

Consistencia de lectura a nivel de transacción

Cuando una transacción se ejecuta en modo *serializable*, todos los accesos a datos reflejan el estado de la base de datos tal como está al momento de que la transacción inicia. Esto significa que los datos vistos por todas las consultas dentro de la misma transacción son consistentes con respecto a un solo punto en el tiempo, salvo que las consultas pueden ver los cambios hechos dentro de la misma transacción. Este nivel de transacción produce lecturas repetibles y no expone a las consultas a fantasmas.

Transacciones read-only

Permiten ver aquellos cambios que fueron confirmados al momento en que la transacción inició y no permiten ejecutar instrucciones *insert, delete* o *update.*

Establecimiento del nivel de aislamiento

El nivel default en Oracle es el *read committed,* pero se puede establecer explícitamente un nivel de la siguiente manera:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET TRANSACTION ISOLATION LEVEL READ ONLY;
```

En forma alterna se puede establecer un nivel de aislamiento para un conjunto subsecuente de transacciones con:

```
ALTER SESSION SET ISOLATION_LEVEL SERIALIZABLE;
ALTER SESSION SET ISOLATION_LEVEL READ COMMITTED;
```

El nivel de *read committed* es adecuado para ambientes donde hay pocas transacciones con probabilidad de entrar en conflicto. Aunque se permiten lecturas no repetibles y fantasmas, el rendimiento de las transacciones es de alto potencial.

El nivel *serializable* es apropiado para ambientes:

- con bases de datos grandes y transacciones cortas que actualizan pocas filas
- donde es baja la probabilidad de que dos transacciones concurrentes modifiquen las mismas filas
- donde hay transacciones relativamente largas (en ejecución) y básicamente son de sólo lectura (por ejemplo, impresiones de mucha información).

5.4 Control de concurrencia

Asociadas con el concepto de transacciones se tienen varias técnicas de **control de concurrencia**, usadas para asegurar la no interferencia o aislamiento de transacciones ejecutadas concurrentemente. A continuación se describen algunos aspectos relacionados con esto.

Granularidad de los datos

- Un elemento de la base de datos puede ser: una tupla de una tabla, un conjunto de tuplas, un bloque de disco, una tabla, un conjunto de tablas o toda la base de datos.
- Entre más grande es el tamaño del elemento, más pequeño es el grado de concurrencia permitido.
- Por otro lado, entre más pequeño es el tamaño de los elementos, más elementos (bloqueos) existirán en la BD (más trabajo para el manejador de los bloqueos).
- Al tamaño de los elementos de datos frecuentemente se le llama **granularidad de los elementos de datos.**
- ¿Cuál es el mejor tamaño de los elementos? Si una transacción típica accede a un pequeño número de registros, es ventajoso que la granularidad de los elementos sea de un registro; si la transacción accede a muchos registros de la misma tabla, puede ser mejor tener una granularidad de bloque o de tabla tal que considere a todos esos registros como uno (o pocos) elementos de datos.

Técnicas de bloqueo

Un **candado** o **bloqueo (lock)** es una variable asociada con un elemento de datos en la base de datos y describe el status de ese elemento con respecto a posibles operaciones que pueden ser aplicadas al mismo.

En general, puede existir un candado por cada elemento de datos en la base de datos. Se usan los candados como un medio de sincronizar el acceso a los elementos de la base de datos por las transacciones concurrentes.

Tipos de candados

Candados binarios. Pueden tener dos estados o valores: bloqueado (1) y desbloqueado (0). Un candado distinto puede ser asociado con cada elemento X de la base de datos. Se usa lock(X) para referirse al *valor* del candado asociado a X.

Dos operaciones, lock_item(X) y unlock_item(X), deben ser incluidas en las transacciones cuando los candados binarios son usados:

```
lock_item(X):
   B:   if lock(X) = 0     //El elemento no está bloqueado.
        then lock(X) ← 1;          //Bloquea el elemento.
        else begin
           wait (until lock(X) = 0 y el lock manager despierte a la transacción);
           goto B;
        end;

unlock_item(X):
        lock(X) ← 0;      //Desbloquea el elemento.
        if hay transacciones esperando
        then despierta a una de las transacciones que están esperando;
```

El candado binario fuerza la **exclusión mutua** sobre los elementos de datos. Estas operaciones deben ser implementadas como unidades indivisibles; esto es, no deben interrumpirse en su ejecución.

Cada transacción debe ajustarse a las siguientes reglas:
- Una transacción T debe emitir la operación lock_item(X) antes de que cualquier read_item(X) o write_item(X) sean ejecutados en T.
- Una transacción T debe emitir la operación unlock_item(X) después de que sean completados todos los read_item(X) y write_item(X) en T.
- Una transacción T no emitirá una operación lock_item(X) si ya tiene un candado sobre X.
- Una transacción T no emitirá una operación unlock_item(X) a menos que ya tenga un candado sobre X.

Candados compartidos y exclusivos. Los candados anteriores son muy restrictivos. Por esta razón se puede usar un **candado de modo múltiple**.

Estos candados pueden tener tres tipos de valores: "read-locked", "write-locked" o "unlocked".
En este esquema hay tres operaciones de bloqueo: read_lock(X), write_lock(X) y unlock(X).

Un elemento read-locked también se conoce como **bloqueado-compartido (share-locked)** porque se permite que otras transacciones lo lean; un elemento write-locked se conoce como **bloqueado-exclusivo (exclusive-locked)** porque una sola transacción debe tener en exclusiva el candado sobre el elemento.

Usar estos tipos de candados en las transacciones *no garantiza la seriabilidad* de los planes en los cuales aquellas participan. Para garantizarla se necesita seguir *un protocolo adicional*; uno de los mejores es el mencionado en seguida.

Protocolo de bloqueo de dos fases

Se dice que una transacción sigue el **protocolo de bloqueo de dos fases** si *todas* las operaciones de bloqueo (read_lock, write_lock) preceden a la *primera* operación de desbloqueo en la transacción.

Las transacciones se dividen en dos fases: una de **expansión**, durante la cual se pueden adquirir nuevos candados sobre elementos, pero ninguno puede ser liberado; y otra de **contracción**, durante la cual se pueden liberar candados existentes, pero ninguno nuevo puede ser adquirido. Ejemplo:

T_3	T_4

read_lock(Y);
read_item(Y);
write_lock(X);
unlock(Y);
read_item(X);
X= X+Y;
write_item(X);
unlock(X);
	read_lock(X);
	read_item(X);
	write_lock(Y);
	unlock(X);
	read_item(Y);
	Y= X+Y;
	write_item(Y);
	unlock(Y);

Se puede probar que si *cada* transacción en un plan sigue este protocolo, se garantiza que el plan es **seriable**, evitando la necesidad de probar la seriabilidad del mismo.

El empleo de este protocolo, para garantizar que un plan sea seriable, no evita los **bloqueos mortales (deadlocks)** los cuales ocurren cuando cada una de dos transacciones está esperando para que la otra libere el candado sobre un elemento:

T_3	T_4

read_lock(Y);
read_item(Y);
 	
write_lock(X);
	
	read_lock(X);
	read_item(X);
	
	write_lock(Y);

Hay varias formas de prevenir este problema: **protocolo de prevención** (todos los elementos se tratan de bloquear anticipadamente; si alguno falla, ninguno se bloquea), **no espera** (si un candado no se obtiene, la transacción se aborta), usando **timeouts**, con la construcción de un **grafo de espera** (si hay un ciclo en el grafo, existe un deadlock).

5.5 Bloqueos en Oracle

Oracle **automáticamente obtiene los bloqueos necesarios** cuando ejecuta las instrucciones de SQL, de tal manera que los usuarios no necesitan preocuparse acerca de ellos. Oracle automáticamente usa el nivel más bajo de bloqueo para proveer el nivel más alto de concurrencia y aún así proporcionar seguridad en la integridad de los datos. Oracle también permite que **el usuario bloquee manualmente** los datos.

Modos de bloqueo

Oracle usa dos modos de bloqueo en una base de datos multiusuario:

- **Bloqueo compartido (share lock)**, que permite compartir el recurso dependiendo de las operaciones involucradas. El candado permite que varios usuarios puedan leer y compartir el recurso, previniendo que alguien pueda aplicar un bloqueo exclusivo sobre el mismo.
- **Bloqueo exclusivo (exclusive lock)**, el cual evita compartir el recurso y permite modificarlo. La primera transacción que bloquea un recurso de manera exclusiva es la única que puede alterarlo hasta que el bloqueo se libera.

Todos **los candados adquiridos** dentro de una transacción **son mantenidos hasta la terminación** de la misma, esto es, hasta que se aplica *commit* o *rollback* a la transacción. Esto previene lecturas sucias, actualizaciones perdidas o la aplicación de instrucciones de SQL que modifiquen la estructura de la base de datos.

Tipos de candados

Oracle tiene tres categorías de candados: **candados DML, candados DDL** y **candados internos**.

Candados DML. Permiten **proteger la integridad de los datos**. Por ejemplo, un candado DML garantiza que una fila específica en una tabla pueda ser actualizada por **sólo** una transacción a la vez, y que una tabla no pueda ser eliminada si una transacción no confirmada contiene una inserción en dicha tabla.

Existen dos tipos de candados en esta categoría:

Row lock (TX)[2]. Se aplica **al nivel de filas** de una tabla. Es la **granularidad más fina** que posee Oracle y no hay límite en la cantidad que puede tener una instrucción o una transacción.
Una transacción adquiere automáticamente un candado de este tipo **en modo exclusivo** para cada fila individual modificada por un *insert, delete, update* o *select… for update*[3].
Si una transacción adquiere un *row lock* para una fila, también adquiere un *table lock* para la tabla correspondiente a fin de evitar conflictos con operaciones DDL.

Table lock (TM). Una transacción adquiere un *table lock* cuando una tabla es modificada con las siguientes instrucciones: *insert, delete, update, select… for update* o *lock table*. Este tipo de candado permite reservar una tabla dentro de una transacción y evitar que sea alterada o eliminada por otras, si aún no se confirma esta transacción.

La tabla se puede bloquear en alguno de los siguientes modos:

Row share (RS). Indica que se tienen filas bloqueadas en la tabla y que se pretende actualizarlas. Es el modo menos restrictivo de un *table lock*, ofreciendo el grado más alto de concurrencia para una tabla.

Row exclusive (RX). Indica que la transacción ha hecho una o más actualizaciones a filas de la tabla.

Share table (S). Es adquirido para la *tabla* especificada con la siguiente instrucción:

```
LOCK TABLE tabla IN SHARE MODE;
```

Share row exclusive (SRX). Es adquirido para la *tabla* especificada con la siguiente instrucción:

```
LOCK TABLE tabla IN SHARE ROW EXCLUSIVE MODE;
```

Exclusive (X). Es el modo más restrictivo de bloqueo de una tabla. Es adquirido para la *tabla* especificada con la siguiente instrucción:

```
LOCK TABLE tabla IN EXCLUSIVE MODE;
```

La siguiente tabla muestra los modos *table lock* que las instrucciones adquieren y las operaciones que estos bloqueos permiten y prohíben:

		¿Bloqueo permitido?				
Instrucción SQL	Modo del *table lock*	RS	RX	S	SRX	X
`SELECT...FROM table...`	Ninguno	Y	Y	Y	Y	Y
`INSERT INTO table ...`	RX	Y	Y	N	N	N
`UPDATE table ...`	RX	Y*	Y*	N	N	N
`DELETE FROM table ...`	RX	Y*	Y*	N	N	N
`SELECT ... FROM table FOR UPDATE OF ...`	RS	Y*	Y*	Y*	Y*	N
`LOCK TABLE table IN ROW SHARE MODE`	RS	Y	Y	Y	Y	N
`LOCK TABLE table IN ROW EXCLUSIVE MODE`	RX	Y	Y	N	N	N

[2] Las siglas en paréntesis indican la abreviación usada para el candado en cuestión por el monitor de candados de Oracle.
[3] Esta forma del *select* se utiliza para bloquear tuplas de una o varias tablas para su actualización posterior. El bloqueo termina cuando finaliza la transacción actual.

LOCK TABLE *table* IN SHARE MODE	S	Y	N	Y	N	N
LOCK TABLE *table* IN SHARE ROW EXCLUSIVE MODE	SRX	Y	N	N	N	N
LOCK TABLE *table* IN EXCLUSIVE MODE	X	N	N	N	N	N
Y*, si no hay *row locks* conflictivos con otras transacciones; en caso contrario, una espera ocurre.						

Candados DDL. Permiten proteger la estructura de la base de datos, por ejemplo, las definiciones de las tablas y de las vistas. No se detallan en estas notas.

Candados internos y latches. Permiten proteger estructuras internas (catálogo) de la base y son manejados de manera totalmente automática. Los usuarios no tienen acceso a ellos.

Bloqueo de datos explícito (manual)

Oracle siempre ejecuta los bloqueos automáticamente para asegurar la concurrencia, la integridad de los datos y la consistencia de lectura a nivel de instrucción. Sin embargo, se pueden anular estos mecanismos de bloqueo default de Oracle. Esto es útil en las siguientes situaciones:

- La aplicación requiere una consistencia de lectura a nivel de transacción o lecturas repetibles. Esto se puede lograr usando bloqueos explícitos, transacciones de sólo lectura o transacciones seriables.
- La aplicación requiere que una transacción tenga acceso exclusivo a un recurso sin que tenga que esperar a que otras transacciones terminen.

Las transacciones que incluyen las siguientes instrucciones de SQL anulan los bloqueos default de Oracle:

SET TRANSACTION ISOLATION LEVEL.
LOCK TABLE.
SELECT … FOR UPDATE

Los bloqueos adquiridos por estas instrucciones son liberados después de que la transacción termina (por *commit* o *rollback*).

5.6 Consideraciones sobre las transacciones y concurrencia en Oracle

Bloqueo default para las consultas. Las consultas son las instrucciones que tienen menos probabilidad de interferir con otras instrucciones de SQL debido a que sólo leen datos. Las instrucciones de **insert, update** y **delete** pueden tener consultas implícitas como parte de ellas. Las siguientes características son verdaderas para todas las consultas que no usan la cláusula **for update**:

- Una consulta no adquiere candados sobre los datos. Por lo tanto, otras transacciones pueden consultar y actualizar una tabla que está siendo consultada, inclusive usando las tuplas que están siendo consultadas por la transacción actual.
- Una consulta no tiene que esperar a que se libere candado alguno; siempre puede proceder.

Boqueo default para insert, update, delete y select … for update. Las características de bloqueo para estas instrucciones son como sigue:

- La transacción que contiene una de estas instrucciones (llamadas también instrucciones DML) adquiere candados de fila (row lock) exclusivos sobre las tuplas modificadas por la instrucción. Otras transacciones no pueden actualizar o borrar las tuplas bloqueadas hasta que la transacción actual se confirme (commit) o se revierta (roll back).
- La transacción que contiene una instrucción DML no necesita adquirir candados de fila sobre las filas seleccionadas con una subconsulta que la instrucción contenga.
- Una consulta en una transacción puede ver los cambios hechos por instrucciones DML previas dentro de la misma transacción, pero no puede ver los cambios hechos por otras transacciones que iniciaron después de su propia transacción.
- En adición a los candados de fila necesarios, una transacción que contiene una instrucción DML adquiere al menos un *row exclusive table lock* sobre la tabla que contiene las filas afectadas.

5.7 Transacciones y programación concurrente en Visual C#

En esta sección se describen los elementos que se pueden usar, a nivel de programación y de bases de datos, para implementar el manejo de las transacciones y la concurrencia desde Visual C#.

Niveles de aislamiento definidos por ADO.Net

Los identificadores usados para los niveles de aislamiento de las transacciones son:

`IsolationLevel.ReadUncommitted`: lecturas sucias, lecturas no repetibles y lecturas fantasma pueden ocurrir.

`IsolationLevel.ReadCommitted`: las lecturas sucias se previenen; las lecturas no repetibles y las fantasma, no.

`IsolationLevel.RepeatableRead`: las dos primeras se previenen, las lecturas fantasmas no.

`IsolationLevel.Serializable`: los tres casos se previenen.

A continuación se explica donde se deben especificar estos valores.

Realización de la transacción

Los pasos que se deben seguir para ejecutar un conjunto de instrucciones como una transacción son los siguientes:

1. Se abre una conexión a la base de datos, con un objeto **Connection**, si es que no ha sido establecida previamente (se emplea el método **Open()**).
2. Se llama al método **BeginTransaction**, de ese objeto **Connection**, para iniciar la transacción. En este momento debe especificarse como parámetro el nivel de aislamiento que se quiere para la transacción, dando alguno de los valores indicados en el párrafo anterior. El método regresa una referencia a un objeto **Transaction**.
3. Este objeto se asigna a las propiedades: **InsertCommand, DeleteCommand, UpdateCommand** y **SelectCommand**, de los **DataAdapter**'s asociados a la tablas que se van a usar durante la transacción.
4. A continuación se ejecutan dentro de la transacción todas las instrucciones de lectura/escritura que se requieran.
5. Se termina la transacción llamando a los métodos **Commit()** o **Rollback()**, dentro de **Try … Catch … End Try**, según que ésta termine exitosamente o fracase.
6. Se cierra la conexión a la base de datos (se usa el método **Close()**).

Se puede usar la propiedad **IsolationLevel**, del objeto **Transaction**, para saber el nivel actual de aislamiento de una transacción.