BDII 4 – CONTROL DE CONCURRENCIA

- Al ejecutar transacciones concurrentemente puede dejar de conservarse la propiedad de **aislamiento** de la bd.
  Para evitar esto, el sistema controla la interacción entre transacciones concurrentes a través del esquema de CONTROL DE CONCURRENCIA.

PROTOCOLOS BASADOS EN EL BLOQUEO

- El método más habitual para implementar la **exclusión mutua** (mientras una transacción accede a un elemento de datos, ninguna otra lo puede modificar) es permitir que una transacción acceda a un elemento de datos solo si posee un BLOQUEO sobre él.

	C	X
C	Cierto	Falso
X	Falso	Falso

- Existen varios **modos de bloqueo**:
  - Bloqueo **compartido** $(C) \longrightarrow$ la transacción con el bloqueo puede **leer** el elemento, pero no escribir sobre él.
    ↳ Varias transacciones a la vez pueden tener un bloqueo C sobre el mismo elemento.
  - Bloqueo **exclusivo** $(X) \longrightarrow$ la transacción con el bloqueo puede tanto **leer** como **escribir** sobre el elemento.
    ↳ Sólo una transacción en cada instante puede tener un bloqueo X sobre cada elemento.

1. Las transacciones deben **solicitar** un bloqueo apropiado sobre el elemento al que van a acceder en función de qué operaciones van a realizar sobre él.
   - Las instrucciones $bloquear - c(Q)$ y $bloquear - x(Q)$ permiten solicitar un bloqueo sobre un elemento de datos Q.
2. Las transacciones sólo podrán realizar la operación después de que el gestor de concurrencia les **conceda el bloqueo**.
   - Si el elemento ya tiene un **bloqueo incompatible** de otra transacción, el gestor no concederá otro hasta que se libere y la transacción tendrá que **esperar**.
3. - La instrucción $desbloquear(Q)$ permite liberar un bloqueo sobre un elemento de datos Q.

- Las transacciones deben tener un **bloqueo** sobre un elemento de datos **mientras quieran seguir accediendo** a él, pero no siempre es aconsejable **desbloquearlo inmediatamente** después de acabar de usarlo, pues puede dejar de asegurarse la **secuencialidad**.

T1	T2	Gest. cont. concurr.
bloquear-x(B)		conceder-x(B, T1)

leer(B);
B:= B – 50;
escribir(B);
desbloquear(B)		

	bloquear-c(A)	conceder-c(A, T2)
	leer(A) desbloquear(A) bloquear-c(B)	
		conceder-c(B, T2)
	leer(B) desbloquear(B) visualizar(A+B)	
bloquear-x(A)		

leer(A);
A:=A+50;
escribir(A);
desbloquear(A)
		conceder-x(A, T1)

T1	T2

bloquear-x(B)
leer(B);
B:= B – 50;
escribir(B);	

	bloquear-c(A) leer(A) bloquear-c(B)
bloquear-x(A)	

T1	T2	T3	T4	T5	
bloquear-x(A)	bloquear-c(A) leer(A)				. . .
		bloquear-c(A) leer(A)			
	desbloquear(A)				
			bloquear-c(A) leer(A)		
		desbloquear(A)			
				bloquear-c(A) leer(A)	. . .

INTERBLOQUEOS

- Se produce un INTERBLOQUEO cuando una transacción no puede avanzar porque está esperando a que otra desbloquee un elemento de datos mientras que esta otra está esperando a que la primera desbloquee otro elemento para poder continuar. En esta situación **ninguna de las dos avanza**.
  - Para solucionar un interbloqueo, el sistema **retrocede** una de las dos transacciones, liberando así sus bloqueos para que la otra continúe.

Entonces:
- Si no se usan bloqueos o se desbloquean los elementos demasiado pronto $\longrightarrow$ pueden aparecer **estados inconsistentes**.
- Si no se desbloquea un elemento antes de solicitar un bloqueo sobre otro $\longrightarrow$ pueden aparecer **interbloqueos**.
- Se prefieren los **interbloqueos**, ya que tienen una solución sencilla, mientras que ante un estado inconsistente no se puede hacer nada.

PROTOCOLOS DE BLOQUEO

- Se denomina PROTOCOLO DE BLOQUEO al conjunto de **reglas** que indica el momento en que una transacción puede **bloquear y desbloquear** cada elemento.
  ↳ Supone una restricción en el número de planificaciones secuenciables posibles.
  - Se dice que una planificación es LEGAL bajo un protocolo de bloqueo dado si es una planificación posible para un conjunto de transacciones que sigan las reglas del protocolo.
- Se dice que una transacción A PRECEDE a otra B en una planificación $(A \rightarrow B)$ cuando A realiza un bloqueo sobre un elemento de datos y posteriormente B realiza otro bloqueo en un modo **incompatible** sobre ese mismo elemento.
  - Si $A \rightarrow B$, en cualquier planificación secuencial equivalente, A debe aparecer antes de B.
  - Se dice que un protocolo de bloqueo ASEGURA LA SCC si, y solo si, todas las planificaciones legales son SCC (es decir, la relación $\rightarrow$ asociada **no tiene ciclos**).

CONCESIÓN DE BLOQUEOS

- En principio, un bloqueo sobre un elemento de datos se **concederá** si sobre dicho elemento **no hay activo ningún otro** bloqueo **conflictivo** con el solicitado.
  - Esto puede provocar que varias transacciones vayan solicitando bloqueos C sobre un elemento y que cada una lo libera poco después de que se le conceda, de manera que una transacción que desea adquirir un bloqueo X sobre él nunca lo consigue y queda en estado de INANICIÓN.
- Para evitar esto, un bloqueo sobre un elemento de datos se concederá si sobre él **no hay activo ningún otro bloqueo conflictivo** con el solicitado y además no **existe otra transacción esperando un bloqueo** sobre ese elemento y que lo haya solicitado antes.

BLOQUEO EN DOS FASES

- El PROTOCOLO DE BLOQUEO EN DOS FASES asegura la secuencialidad separando las peticiones de bloqueo y desbloqueo de cada transacción en dos fases:
  - Fase de **crecimiento** → la transacción puede obtener bloqueos, pero no puede liberarlos.
  - Fase de **decrecimiento** → la transacción puede liberar bloqueos, pero no obtenerlos.
  - Una vez una transacción libera un bloqueo, entra en la fase de decrecimiento y no puede realizar más peticiones de bloqueo.
  - PUNTO DE BLOQUEO de una transacción → punto de una planificación en el que la transacción obtiene su último bloqueo.
- ✓ Asegura la **SCC** ya que las transacciones se pueden ordenar según sus puntos de bloqueo asegurando que no se producirán conflictos.
- × **No** asegura la **ausencia de interbloqueos**.
- × **No** asegura la **ausencia de retrocesos en cascada**, pues no impide que una transacción lea datos escritos por otra que ya liberó su bloqueo sobre ellos, pero aún no se comprometió.
- × Provoca una **pérdida de concurrencia**, pues algunas transacciones deben esperar a los desbloqueos de otras para poder continuar.

PROTOCOLO DE BLOQUEO **ESTRICTO** EN DOS FASES

- Impide liberar un bloqueo X hasta que se compromete la transacción, asegurando que ninguna otra puede leer el dato modificado hasta que se comprometa la transacción que lo escribió.
- ✓ Asegura la **ausencia de retrocesos en cascada**.

PROTOCOLO DE BLOQUEO **RIGUROSO** EN DOS FASES

- Impide liberar cualquier bloqueo hasta que se compromete la transacción, permitiendo secuenciar las transacciones en el orden en que se comprometen.

PROTOCOLO DE BLOQUEO EN DOS FASES **CON CONVERSIONES DE BLOQUEO**

- CONVERSIÓN DE BLOQUEO → mecanismo que cambia el modo de un bloqueo. Se denomina SUBIR a la conversión de C a X y BAJAR a la de X a C.
- Para respetar el protocolo en dos fases, la operación de **subir** sólo se puede realizar en la **fase de crecimiento** y la de **bajar** en la de **decrecimiento**.
- ✓ **Mejora la concurrencia**, pues las conversiones facilitan la compatibilidad entre instrucciones de varias transacciones.

- Todas las planificaciones que cumplan el protocolo en dos fases son SCC, pero existen **planificaciones SCC** que **no se pueden obtener** por medio del protocolo de **bloqueo de dos fases**.
  - Para obtenerlas por medio de otros protocolos se necesita tener **información adicional** sobre las transacciones o imponer alguna **estructura u orden en el conjunto de elementos de la bd**.

Un esquema de uso extendido **genera** las **peticiones** de bloqueo y desbloqueo de forma automática basándose en las peticiones de lectura y escritura:

- Se realiza una operación $\boldsymbol{leer}(Q)$ → el sistema genera una instrucción $bloquear\text{-}c(Q)$ seguida de una $leer(Q)$.
- Se realiza una operación $\boldsymbol{escribir}(Q)$:
  - Si la transacción ya tenía un bloqueo C sobre Q → el sistema genera una instrucción $subir(Q)$ seguida de una $escribir(Q)$.
  - En caso contrario → el sistema genera una instrucción $bloquear\text{-}x(Q)$ seguida de una $escribir(Q)$.
- Los **desbloqueos** se realizan cuando la transacción se **compromete o aborta**.

TRATAMIENTO DE INTERBLOQUEOS

Existen dos métodos para tratar el problema de los interbloqueos:

- **Prevención** de interbloqueos → asegura que el sistema nunca llega a un estado de interbloqueo.
  - ↳ Normalmente se usa cuando la **probabilidad** de que el sistema llegue a un estado de interbloqueo es relativamente **alta**.
- **Detección y recuperación** de interbloqueos → se permite que el sistema llegue a un estado de interbloqueo y se trata de recuperarse de él después.

PREVENCIÓN DE INTERBLOQUEOS

Existen tres grandes enfoques para la prevención de interbloqueos:

- **Asegurar que no puede haber esperas cíclicas**:
  - El esquema más simple para esta aproximación exige que las transacciones **bloqueen todos los elementos** de datos que van a usar de manera **atómica** antes de comenzar a ejecutarse.
    - × A menudo es **difícil predecir** qué elementos se necesitará bloquear antes de que comience la transacción.
    - × La **utilización de elementos puede ser muy baja**: muchos de ellos pueden estar bloqueados, pero sin usarse durante mucho tiempo.
  - Otra alternativa es imponer un **orden a todos los elementos de datos** de la bd y exigir que cada transacción **bloquee los elementos en el orden**, de manera que una vez haya bloqueado un elemento concreto, no podrá bloquear ningún otro que venga antes en la ordenación.
    - ✓ Es **fácil de implementar** cuando el conjunto de elementos a los que va a acceder una transacción **se conoce cuando comienza** la ejecución.
- **Expropiar y retroceder** transacciones → cuando una transacción solicita un bloqueo incompatible con uno de otra transacción, se puede expropiar la segunda retrocediéndola y asignándole el bloqueo deseado a la primera.
  - Para controlar la expropiación se asigna a cada transacción una **marca temporal** que se usará para decidir si debe esperar o retroceder. Cuando A solicite un bloqueo que posee actualmente B:
    - **Esperar-Morir** → evita la expropiación.
      - Si $marca(A) < marca(B)$ (A es más antigua) → A espera.
      - Si $marca(A) > marca(B)$ (B es más antigua) → A retrocede (muere).
    - **Herir-Esperar** → usa expropiación.
      - Si $marca(A) < marca(B)$ (A es más antigua) → B retrocede (es herida por A).
      - Si $marca(A) > marca(B)$ (B es más antigua) → A espera.
  - × Ambos esquemas pueden llevar a **retrocesos innecesarios**.

- **Bloqueos con límite de tiempo** ⟶ cuando una transacción solicita un bloqueo incompatible con otro de otra transacción, espera durante un intervalo de tiempo especificado. Si no se le concede el bloqueo en ese tiempo, ella misma retrocede y vuelve a comenzar.
  - Así, si había un interbloqueo, una o más de las transacciones involucradas estarán fuera de tiempo y retrocederán, permitiendo al resto continuar.
  - ✓ Funciona bien y es **fácil de implementar** cuando las transacciones son **cortas**.
  - × Es difícil decidir **cuánto debe esperar** una transacción para que esté fuera de tiempo.
    - Si la espera es demasiado larga ⟶ provocan retardos innecesarios una vez se produce un interbloqueo.
    - Si la espera es demasiado corta ⟶ producen retrocesos incluso sin haber interbloqueos.
  - × Puede provocar **inanición**.

DETECCIÓN Y PREVENCIÓN DE INTERBLOQUEOS

Para realizar la detección y prevención de interbloqueos:

1. **Mantener información** sobre la asignación de elementos a las transacciones (los **bloqueos activos**) y sobre las **peticiones pendientes**.
2. Proporcionar un **algoritmo** que use esta información para **determinar** si el sistema está en estado de **interbloqueo**.
3. **Recuperarse** del interbloqueo cuando el algoritmo de detección determine que existe.

DETECCIÓN DE INTERBLOQUEOS

- Los interbloqueos se pueden describir mediante un grafo dirigido denominado GRAFO DE ESPERA:
  - Los vértices son todas las transacciones del sistema.
  - Se añadirá una arista $T_i \rightarrow T_j$ cuando la transacción T_i solicite un elemento de datos que posea en ese momento la transacción T_j. Esta arista solo se borrará cuando la transacción T_j deje de poseer el elemento que necesita T_i.
  - Existe un **interbloqueo** en el sistema si, y solo si, **el grafo de espera contiene un ciclo**. Se dice que toda la transacción involucrada en el ciclo está en interbloqueo.
- Para detectar los interbloqueos, el sistema debe mantener el grafo de espera e **invocar periódicamente** un **algoritmo que busque un ciclo** en él.
- × Ejecutar un algoritmo de detección de ciclos continuamente sobre un grafo que **crece con el número de transacciones** supone un **coste muy grande.**

RECUPERACIÓN DE INTERBLOQUEOS

- Cuando se detecta un interbloqueo el sistema se debe recuperar de él retrocediendo una o más de las transacciones involucradas.

Para ello se deben realizar 2 acciones:

1. **Selección de una víctima** ⟶ determinar qué transacción (o transacciones) involucradas se van a retroceder.
   - Se deben seleccionar aquellas de **coste mínimo**. El coste de un retroceso se puede determinar de muchas maneras (número de elementos de datos usados por la transacción, número de transacciones involucradas en el retroceso, etc.).
     - Si la selección de víctimas se basa principalmente en factores de coste, puede que se seleccione constantemente la misma transacción, provocando que esta muera de **inanición**.
       ↳ Para evitar esto, se incluye el número de retrocesos que ha sufrido una transacción en su factor de coste.
2. **Retroceso** ⟶ una vez decidido qué transacción (o transacciones) se van a retroceder, se debe determinar hasta dónde lo harán.
   - La solución más simple es un **retroceso total**, pero es más efectivo realizar un **retroceso parcial** en el que sólo se retrocede lo necesario.
     ↳ El retroceso parcial requiere que el sistema mantenga **información adicional** sobre el estado de las transacciones en ejecución para poder retroceder la transacción hasta el punto donde obtuvo el primero de los bloqueos responsables del interbloqueo.

PROTOCOLOS BASADOS EN VALIDACIÓN

- Si la mayoría de transacciones son de **sólo lectura**, la probabilidad de **conflictos** es baja, por lo que el sistema de **control de concurrencia** impone una **sobrecarga innecesaria**, ya que muchas veces se llegará a un estado **consistente** de la bd incluso sin él.

- Para evitar esto se necesita saber de antemano qué **transacciones** estarán **involucradas** en conflictos, por lo que se tendrá que **supervisar** el sistema con un PROTOCOLO DE VALIDACIÓN, que asume que cada transacción se ejecuta en 2 o 3 fases:
  1. **Fase de lectura** ⟶ la transacción lee los valores de varios elementos de datos y los almacena en **variables locales** sobre las que realizará todas las operaciones de escritura, sin actualizar la bd.
  2. **Fase de validación** ⟶ la transacción realiza una **prueba de validación** para determinar si puede escribir los valores de las variables locales actualizadas en la bd sin causar una violación de la secuencialidad. Si no cumple esta prueba, el sistema **aborta** la transacción.
  3. **Fase de escritura** ⟶ se escriben los valores de las variables locales de la transacción en la bd (las transacciones de sólo lectura no tienen esta fase).

- Para la **prueba de validación** se necesita conocer el momento en el que tienen lugar las distintas fases de las transacciones. Para ello, se asocian tres **marcas temporales** a cada transacción: $inicio(A)$, $validación(A)$ y $fin(A)$.
- El **orden de secuencialidad** se determina mediante la técnica de **ordenación de marcas temporales**, usando el valor de $\boldsymbol{validacion}$ como MT.
  - Así, si $MT(B) < MT(A)$ toda planificación producida debe ser equivalente a una secuencial en la que B aparezca antes que A.
- La comprobación de validación para A se cumplirá si, para toda B con $MT(B) < MT(A)$, se cumple una de las siguientes condiciones:
  - $Fin(B) < Inicio(A) \rightarrow B$ acaba de ejecutarse antes de que comience A, por lo que el orden de secuencialidad se mantiene.
  - $escritos(B) \cap leídos(A) = \emptyset$ y $Inicio(A) < Fin(B) < Validacion(A) \rightarrow$ las escrituras de B y A no se superponen y las escrituras de B no afectan a las lecturas de A y viceversa, por lo que el orden de secuencialidad se mantiene.

- ✓ **Evita los retrocesos en cascada** pues las escrituras reales se realizan después de que la transacción se haya comprometido.
- × **No evita la inanición** de transacciones largas debido a que una secuencia de transacciones cortas puede causar repetidos fallos de validación de una larga.
  ↳ Para evitarlo es necesario bloquear temporalmente las transacciones cortas para permitir que la larga termine.

- Los protocolos basados en denominación son **esquemas de control de concurrencia optimistas**, ya que las transacciones se ejecutan de forma optimista, asumiendo que serán capaces de finalizar la ejecución y validar al final.
- En cambio, los bloqueos y la ordenación por marcas temporales son **esquemas de control de concurrencia pesimistas** pues fuerzan una espera o un retroceso cada vez que se detecta un conflicto, aun cuando existe una posibilidad de que la planificación sea secuenciable en cuanto a conflictos.

AISLAMIENTO DE INSTANTÁNEAS

- Un AISLAMIENTO DE INSTANTÁNEA implica dar a una transacción una "instantánea" de la bd en el momento en que comienza su ejecución para que trabaje sobre ella en completo aislamiento de otras transacciones concurrentes.
  - Las **actualizaciones** se mantienen en la instantánea de la transacción hasta que esta se **compromete**, momento en el que se escriben en la bd.
    - ↳ La transición de la transacción al estado comprometido y la escritura de sus actualizaciones en la bd se deben realizar de forma **atómica**, de manera que cualquier instantánea que cree otra transacción o incluye todas las actualizaciones o ninguna.
- Es ideal para las transacciones de **sólo lectura** ya que nunca tendrán que esperar y nunca serán abortadas por el gestor de concurrencia.

VALIDACIÓN PARA LAS TRANSACCIONES DE ACTUALIZACIÓN

- El problema de la ACTUALIZACIÓN PERDIDA sucede cuando dos transacciones que se ejecutan concurrentemente intentan actualizar el mismo elemento. Como trabajan en sus instantáneas, ninguna verá las actualizaciones realizadas por la otra, por lo que, si a ambas se les permite escribir en la bd, la **primera actualización será sobrescrita por la segunda**.

Para solucionar este problema existen 2 variantes del protocolo de aislamiento de instantáneas:

- **Primer compromiso gana** ⟶ cuando una transacción A entra en el estado **parcialmente comprometida**, se toman las siguientes medidas **atómicamente**:
  - Se comprueba si alguna transacción concurrente con A ha escrito alguna actualización en la bd en algún elemento de los que se pretenden escribir:
    - Si no se encuentra dicha transacción ⟶ A se compromete y sus actualizaciones se escriben en la bd.
    - Si se encuentra dicha transacción ⟶ se aborta A.
  - ▸ Si varias transacciones entran en conflicto, la primera que se compruebe consigue escribir en la bd, mientras que las siguientes son abortadas.
- **Primera actualización gana** ⟶ cuando una transacción A intenta actualizar un elemento de datos, solicita un **bloqueo de escritura sobre él**.
  - Se comprueba si alguna transacción concurrente con A tiene un bloqueo sobre el mismo elemento:
    - Si no se encuentra dicha transacción:
      - Si el elemento lo ha actualizado cualquier transacción concurrente ⟶ A aborta.
      - En otro caso ⟶ A puede continuar con su ejecución (incluso comprometerse).
    - Si se encuentra dicha transacción B ⟶ A no puede continuar y espera hasta que:
      - B aborte ⟶ se libera el bloqueo y lo adquiere A, que se comportará como si el bloqueo no lo hubiera tenido nadie antes.
      - B se comprometa ⟶ se aborta A.
  - ▸ Si varias transacciones entran en conflicto, la primera que obtenga el bloqueo consigue escribir en la bd, mientras que las siguientes son abortadas (a no ser que la primera aborte).

SOBRE LA SECUENCIALIDAD

- × El aislamiento de instantáneas **no asegura la secuencialidad**. Por ejemplo:
  - ▷ Dadas dos transacciones concurrentes que leen los elementos A y B pero una actualiza A y la otra B, como actualizan diferentes elementos se permite a ambas comprometerse, pero el grafo de precedencia tiene un ciclo.
  - ▷ Dadas dos transacciones concurrentes donde una lee B y lo actualiza y la otra lee A y B y actualiza A. En principio, no existe ningún problema entre ellas pues el grafo de precedencia no tiene ciclos, pero si una tercera transacción lee A y B después de que la segunda transacción se comprometa, pero antes de que lo haga la primera, se permite su compromiso, aunque el grafo de precedencia tiene un ciclo.
  - La razón de la secuencialidad es asegurar la consistencia cuando se ejecutan transacciones concurrentemente, por lo que estas anomalías **no son tan graves**. Si la ejecución termina en un **estado consistente**, es **aceptable** que no sea secuenciable.
    - La bd comprueba las **restricciones de integridad** en el momento del **compromiso** y no en la instantánea, lo que ayuda a evitar inconsistencias.
- Para evitar algunas anomalías del aislamiento de instantáneas se puede añadir la cláusula $for\ update$ a las consultas de SQL.
  Esta cláusula provoca que el sistema trate los **datos leídos** en la consulta **como si hubieran sido actualizados**.
  - × Se debe usar con cuidado, ya que puede **reducir** considerablemente la **concurrencia**.
- Existen **métodos formales** para determinar si un conjunto de transacciones corre el **riesgo de una ejecución no secuenciable** bajo un aislamiento de instantánea y para decidir en qué **partes conflictivas hacer modificaciones** (por ejemplo, añadir $for\ update$) para asegurar la secuencialidad.
  - ↳ Por supuesto, estos métodos solo se pueden aplicar si se **conoce con antelación qué transacciones se van a ejecutar**.

OPERACIONES PARA INSERTAR Y BORRAR

BORRADO

- Para comprender cómo la presencia de instrucciones $borrar(Q)$ afecta al control de concurrencia se debe decidir cuándo entran en conflicto con otra instrucción:
  - $leer(Q)$ ⟶ entra en conflicto si se realiza **después** de $borrar(Q)$.
  - $escribir(Q)$ ⟶ entra en conflicto si se realiza **después** de $borrar(Q)$.
  - $borrar(Q)$ ⟶ entra en conflicto si se realiza **después** de $borrar(Q)$.
  - $insertar(Q)$ ⟶ entra en conflicto si se realiza **después** de $borrar(Q)$ si Q no existía en la bd en un primer lugar.
- ▸ En conclusión, en el protocolo de bloqueo en dos fases, se necesita un **bloqueo X** en un elemento antes de borrarlo.

INSERCIÓN

- Para comprender cómo la presencia de instrucciones $insertar(Q)$ afecta al control de concurrencia se debe decidir cuándo entran en conflicto con otra instrucción:
  - $leer(Q)$ ⟶ entra en conflicto si se realiza **antes** que $insertar(Q)$ si Q no existía en la bd en un primer lugar.
  - $escribir(Q)$ ⟶ entra en conflicto si se realiza **antes** que $insertar(Q)$ si Q no existía en la bd en un primer lugar.
  - $borrar(Q)$ ⟶ entra en conflicto si se realiza **antes** que $insertar(Q)$ si Q no existía en la bd en un primer lugar.
  - En conclusión, en el protocolo de bloqueo en dos fases, una transacción que inserta un nuevo elemento en la bd obtiene un **bloqueo X** sobre él.

LECTURA DE PREDICADOS Y EL FENÓMENO FANTASMA

- El problema del FENÓMENO FANTASMA sucede cuando se pretende realizar una instrucción de lectura e inserción sobre la misma relación. El resultado será distinto si se ejecuta primero una o la otra, por lo que las instrucciones entran en **conflicto** por una **tupla que todavía no existe** en la bd.
- Entonces, no es suficiente bloquear a las tuplas a las que se accede, también se necesita **bloquear la información utilizada para encontrarlas**.

BLOQUEO DE RELACIÓN

- Se asocia un elemento de datos con la relación, de manera que este represente la información utilizada para encontrar las tuplas en la relación.
  - Las transacciones que **lean** información sobre las tuplas de una relación tendrán que **bloquear** su elemento de datos en modo C.
  - Las transacciones que **actualicen** información sobre las tuplas de una relación tendrán que **bloquear** su elemento de datos en modo X.
  - Las transacciones que **usen un índice** para recuperar las tuplas de una relación deben **bloquear** el **propio índice**.
- × El bloqueo de este elemento de datos **no impide el acceso a una tupla específica**, sólo evita que otras transacciones actualicen la información acerca de qué tuplas pertenecen a la relación. Por lo tanto, pueden aparecer **inconsistencias** igualmente.
- × Se impide que **dos transacciones que inserten** distintas tuplas en la relación se ejecuten **concurrentemente**.

BLOQUEO DE ÍNDICE

- Se usan los índices para convertir las apariciones del fenómeno fantasma en conflictos de bloqueo sobre los nodos hoja índice.
  Requisitos:
  - Toda relación debe tener **al menos un índice**.
  - Una transacción puede **acceder** a las tuplas de una relación únicamente después de haberlas encontrado a través de uno o más **índices**.
- Una transacción que realiza una **búsqueda** debe bloquear en modo C todos los nodos hoja índice a los que accede.
- Una transacción no puede **insertar**, **borrar** ni **actualizar** una tupla en una relación sin **actualizar** todos los **índices** de la relación.
  La transacción debe obtener bloqueos en modo X sobre todos los nodos hoja índice afectados.
  - Para la **inserción** y **borrado** ⟶ los afectados son los que contienen o han contenido el valor de la clave de búsqueda.
  - Para las **actualizaciones** ⟶ los afectados son los que contenían el valor antiguo de la clave de búsqueda y los que contienen el nuevo.
- ▸ Los **bloqueos** se obtienen sobre las **tuplas** de forma usual, cumpliendo las reglas del protocolo del **bloqueo en dos fases**.
- × **No** trata el **control de concurrencia** sobre los **nodos internos** del índice.
- × El bloqueo de un nodo hoja índice **evita cualquier actualización** del nodo, incluso si la actualización **no entra en conflicto** con el predicado.
  - ↳ El **bloqueo por valor de clave** minimiza estos problemas de bloqueos falsos.

BLOQUEO DE PREDICADO

- Se adquieren bloqueos C sobre los predicados de una consulta.
- Se deben comprobar las **inserciones** y **borrados** sobre la relación para ver si satisfacen el predicado.
  Si es así, hay un **conflicto de bloqueo**, por lo que se **espera** hasta que se libere el predicado.
- En las **actualizaciones** se debe comprobar tanto el valor **inicial** como el **final** para el predicado.
- × No se usa en la práctica pues es más **costoso** de implementar que el protocolo de bloqueo de índices y no aporta ventajas significativas.

CONTROL DE CONCURRENCIA EN INTERACCIONES DE USUARIO

- ▸ Los protocolos de control de concurrencia normalmente consideran que las transacciones **no implican interacción con el usuario**.
- El sistema de CONTROL DE CONCURRENCIA OPTIMISTA SIN VALIDACIÓN DE LECTURA se usa para manejar transacciones que implican interacción con el usuario.
  Usa **números de versión** almacenados en las tuplas.
  - Se añade al esquema de cada relación un **atributo adicional** para almacenar el número de versión de cada tupla, que se **inicia en 0 cuando se crea.**
  - Las **actualizaciones** se realizan **localmente** y se copian a la bd al **comprometer** la transacción siguiendo los siguientes pasos:
    1. Para cada tupla actualizada, la transacción comprueba si el número de versión actual es el mismo que cuando lo leyó:
       - Si coinciden ⟶ se realiza la actualización de la tupla en la bd y se incrementa su número de versión en 1.
       - Si no coinciden ⟶ la transacción se aborta, retrocediendo todas las actualizaciones realizadas.
    2. Si la comprobación del número de versión es correcta para todas las tuplas actualizadas, la transacción se compromete.
  - ↳ Se puede usar una **marca de tiempo** en lugar de un número de versión.

- × **No** asegura **la secuencialidad**.
- ✓ Se puede **implementar fácilmente sobre el SGBD**, realizando la validación y actualización como parte del proceso de compromiso usando el esquema de control de concurrencia de la bd para asegurar la atomicidad de dicho proceso.