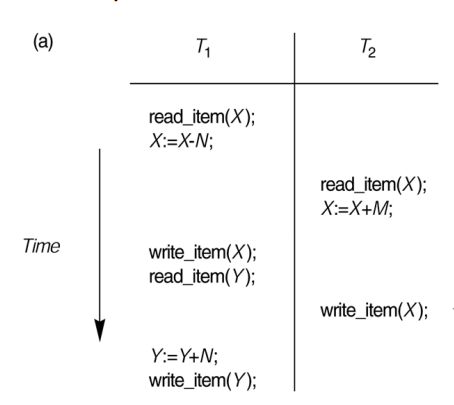
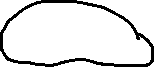
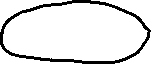
**The problems encountered with the transactions running concurrently if uncontrolled:**

1. **Lost update problem**

* Vấn đề này xảy ra khi 2 transactions truy cập vào cùng databases items có operations interleaved làm cho giá trị của database items bị sai.

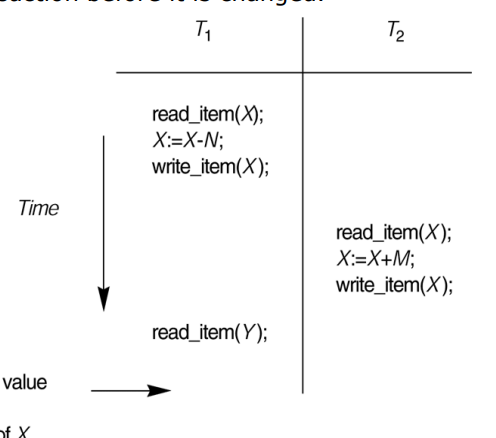
****

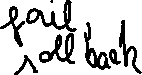


* Với trình tự thực thi trên, thì cập nhật write trên giao tác T1 lên item(X) bị ghi đè bởi giao tác T2. Nghĩa là khi giao tác T1 cập nhật giá trị cho item X và commit, thì khi xem lại value của item X nó sẽ khác với giá trị mong muốn của người thực hiện giao tác T1.
* Ví dụ ban đầu giá trị X = 5, giao tác T1 cập nhật X = X – 2, lẻ ra kết quả sau khi thực thi giao tác sẽ là X = 3. Tuy nhiên do bị giao tác T2 cập nhật X trước khi giao tác T1 commit X = X + 1. Nên kết quả hiện ra sẽ là X = 4 thay vì phải là X = 3 như theo mong muốn của người thực thi giao tác T1. Và ngược lại người thực thi giao tác T2 cũng sẽ thấy X = 4 thay vì là X = 6 như mong muốn.

1. **Temporary update** (or **dirty read**) problem

* Xảy ra khi một transaction cập nhật database item và sau đó thì transaction fails vì một lý do nào đó. Giá trị được cập nhật của transaction đó lại được đọc từ một transaction khác.

****

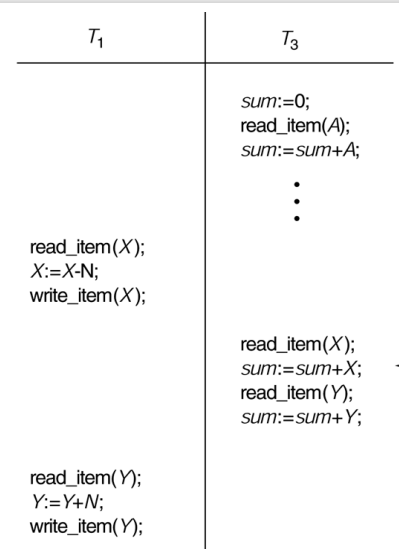


* Theo trình tự thực thi trên, T1 cập nhật giá trị X, và giá trị đó được T2 sử dụng. Tuy nhiên vì một lý do nào đó T1 bị fails và phải quay lui -> gỡ bỏ giá trị cập nhật đó trở về giá trị cũ -> vậy là T2 đã sử dụng giá trị cập nhật tạm thời (không chính xác).



1. **Incorrect summary** problem

* Nếu một transaction đang thực hiện aggregate summary function trên một số bản ghi, đồng thời trong lúc đó có những giao tác khác cập nhật giá trị cho những bản ghi này, aggregation function có thể calculate values trước khi nó được cập nhật hoặc sau khi nó được cập nhật.

****



1. **Unrepeatable (nonrepeatable)** problem

* Transaction T đọc một item 2 lần và item đó bị thay đổi giá trị bởi một giao tác T’ khác giữa 2 lần đọc. Do đó, T nhận 2 giá trị khác nhau cho 2 lần đọc trên cùng 1 item.

1. **Phantom** problem

* Transaction T1 đọc một tập các hàng từ một bảng, chắc chắn dựa trên một vài điều kiện cụ thể trong SQL Where-clause. Giả sử rằng một transaction T2 cập nhật một hàng mới thỏa điều kiện Where-clause trong T1, vào trong table mà T1 đang sử dụng. Nếu T1 lập lại, T1 sẽ nhìn thấy phantom, một hàng mà lần đọc trước không tồn tại.
* **Đó là lý do vì sao concurrency control quan trọng.**

1. **Concurrency Control**

* Buộc ***isolation*** (through mutual exclusion) giữa những conflicting transactions
* Bảo toàn ***database consistency*** through consistency preserving execution of transactions
* Resolve ***read-write and write-write conflicts.***
* Các kỹ thuật Concurrency control:
* The two-phase locking technique of locking data items (cơ chế khóa 2 pha) để ngăn chặn nhiều transactions truy cập vào items concurrently.

+ Multiversion concurrency control (điều khiển concurrency thông đa pha bản thông qua khóa)

* Concurrency control protocols sử dụng timestamps

+ Multiversion concurrency control

1. **Two-phase locking techniques**

* Locking là hành động bảo vệ permission to *read* or *write* a data item cho 1 transaction
* Unlocking là hành độgn bỏ permissions từ data item.
* Lock và Unlock là atomic operations
* **Two phases:** expanding or growing (first) phase shrinking (second) phase:

**+** một transactions được gọi là tuân theo ***two-phase locking*** protocol (i.e. well-formed) nếu như ***all locking*** operations đứng trước ***first unlock*** operation trong transaction.

* Xảy ra vấn đề với lock:

+ deadlock: các transaction đợi resource lẫn nhau

+ starvation: transaction unlimitedly waits for resource.

* Các kiểu locks: binary locks, shared/exclusive (or read/write) locks, certify locks
  1. **Binary locks**
* Two states (values): locked (1) và unlocked (0)
* Nếu giá trị lock trên data item X là 1, thì item X không thể được accessed bởi database operation (read/write).
* Nếu giá trị của lock trên X là 0, item có thể được accessed.
* Binary lock buộc ***mutual exclution***trên data item: chỉ có 1 transaction được giữ lock trên một data item
* 2 operations được sử dụng với binary locking: **lock\_item, unlock\_item**
  1. **Shared/Exclusive**
* 3 trạng thái: read-locked, write-locked, unlocked

+ Read-locked: ***share-***locked -> các transactions khác vẫn có thể read item

+ write-locked: ***exclusive***-locked -> chỉ có một transaction được giữa lock trên item.

* *Less restrictive* than the binary locking scheme
* 3 locking operations: **read\_lock(X)**, **write\_lock(X), unlock(X)**
* Các rules trong **shared/exclusive locking scheme:**

1. Một transaction t phải yêu cầu read\_lock(x) hoặc write\_lock(x) trước khi read\_item(x) operation.
2. Một transaction T phải yêu cầu operation write\_lock(x) trước khi write\_item(x) operation.
3. Một transaction T phải yêu cầu unlock(x) sau khi tất cả read\_item(x) và write\_item(X) operations hoàn thành.
4. Một transaction T sẽ không thể yêu cầu read\_lock(X) operation nếu như nó đã giữ read (shared) lock hoặc write (exclusive) lock trên item X
5. Một transaction T sẽ không thể yêu cầu write\_lock(X) nếu như nó đã giữ read (shared) lock hoặc write (exclusive) lock trên item X
6. Một transaction T sẽ không thể yêu cầu unlock(X) nếu như nó không giữ read(shared) hay write(exclusive) lock trên item X

Note: luật số 4 và 5 sẽ không có cho ***lock conversion.***

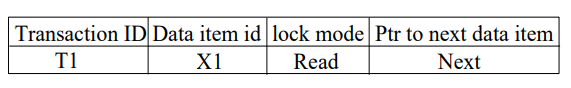
* **Conversion of locks**
* Một transaction đã giữ lock trên item X vẫn được cho phép chuyển đổi lock từ state (read) sang state (write) hoặc ngược lại.
* **Upgrade:** read\_lock -> write\_lock

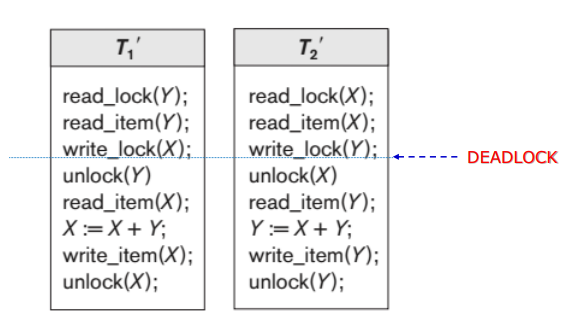
+ lưu ý chỉ thực hiện được khi không có transactions nào giữ lock trên X

* **Downgrade:** write\_lock -> read->lock
* Khi upgrade hay downgrade locks được sử dụng, lock table phải lưu ***transaction identifiers*** trong record structure cho mỗi lock để lưu trữ thông tin transactions nào giữ locks trên item
* Nếu lock conversion được sử dụng, upgrading of locks phải trong expanding phase và downgrading of lock phải trong shrinking phase.
* Essential components:

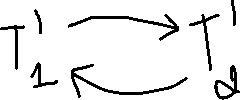
+ Lock Manager: Managing locks on data items.

+ Lock Table: một bảng lưu trữ identity of transaction locking (data item, lock mode and pointer to the next data item locked). Mộ cách đơn giản để hiện thực lock table là sử dụng linked lists





* Transaction T1’ và T2’ tuân theo two-phase locking protocol và xảy ra deadlock: T1’ yêu cầu write\_lock(x) tuy nhiên hiện tại T2’ đang giữ read\_lock(X) nên T1’ đợi T2’ (T1’ -> T2’). Cùng lúc đó T2’ yêu cầu write\_lock(Y) nhưng T1’ đang giữ read\_lock(Y) nên T2’ đợi T1’ (T1’->T2’) => trên đồ thị wait-for graph có vòng lặp tuần hoàn



* Bắt buộc phải abort, quay lui một giao tác để cấp resource cho giao tác còn lại
* Các biến thể của two-phase locking (2PL)
* Basic 2PL
* Conservative 2PL (static 2PL):

+ transaction locks tất cả items mà nó accesses ***trước khi*** transaction bắt đầu thực thi, bằng cách khai bao trước tập read-set và write-set. Nếu như một trong những predeclared items cần nhưng không thể locked, transaction sẽ không lock item nào. -> deadlock-free protocol.

* Strict 2PL

+ transaction T không release bất kỳ ***exclusive (write)*** locks nào cho đến khi ***sau khi*** nó commits hoặc aborts. Dó đó, những transaction khác không thể đọc hay write an item mà đang được written bởi T nếu như T chưa committed, dẫn đến strict schedule for recoverability. Và đây không phải là deadlock-free.

* Rigorous 2PL

+ transaction T không release bất kỳ lock nào ***(kể cả exclusive or shared)*** cho đến khi ***after*** nó commits hoặc aborts, dẫn đến strict schedule for recoverability nhưng dễ hiện thực hơn. Và là not deadlock-free.

1. **Concurrency Control Based On Timestamp Ordering**

* Bảo đảm serializability (khả tuần tự hóa) bằng cách sử dụng transaction timestamps để sắp thứ tự thực thi của transaction cho một trình tự thực thi tương đương với serial schedule.
* ***Timestamp ordering***: sắp xếp transactions dựa trên timestamps
* Ensure: mỗi item accessed by conflicting operations, thứ tự thực thi mà không violate the ***serializability order.***
* Deadlock-free
* **Giải thuật timestamp ordering algorithm**
* Kiểm tra xem có conflicting operations violate the timestamp ordering
* Schedules are guaranteed to be conflict serializable

1. Transaction T issues write\_item(X) operation

* If read\_TS(X) > TS(T) or if write\_TS(X) > TS(T), then abort and roll back T and reject the operation because some younger transaction with a timestamp greater than TS(T) – and hence, after T in the timestamp ordering – has already read or written the value of item X before T had a chance to write X, thus violating the timestamp ordering.
* Otherwise, execute the write\_item(X) operation of T and set write\_TS(X) to TS(T)

1. Transaction T issues a read\_item(X) operation

* If write\_TS(X) > TS(T), then abort and roll back T and reject the operation because some younger transaction with timestamp greater than TS(T) – and hence, after T in the timestamp ordering – has already written the value of item X before T had a chance to read X.
* Otherwise, execute the read\_item(X) operation of T and set read\_TS(X) to the larger of TS(T) and the current read\_TS(X).

**Ngoài ra còn có phần Multiversion concurrency control techniques trên Timestamp và two-phase locking, cũng như kiến thức về recovery, schedule em cũng đã tìm hiểu.**