Granularity of Locks and Degrees of Consistency in a Shared Data Base

J.N. Gray, R.A. Lorie, G.R. Putzolu, I.L. Traiger

Presented By: Anuj Issar

Simon Fraser University

What we know at this point. (i.e. 1976)

- Navigational Databases were the theme of the time.
- Ted Codd, published a paper “A Relational Model of Data for Large Shared Data Banks” in 1970, which decoupled the data model from the storage being used.
- Relational databases prototypes development started to prove its viability(Ingres and System R).

What we know at this point. (i.e. 1976) (Cont.)

- A series of papers published by the IBM team working on System R, defining the concepts of transaction, consistency and granularity for the database.
  - “The Notions of Consistency and Predicate Locks in a Database System”, defining the concepts of transaction, consistency and schedule.
  - “Granularity of Locks in a Shared Data Base”, proposing a locking protocol which associates locks with sets of resources and then comparing it to the then available schemes employed by the DBMS.

Background

- A **transaction** is a series of accesses (for read or write operations) to the data base which, applied to a consistent data base, will produce a consistent data base.
  - During the execution of a transaction, the data base may be temporarily inconsistent. But multiple transactions that run simultaneously always run as they are running in isolation.
- A particular sequencing of the actions of a set of transactions is called a **schedule**. A schedule which gives each transaction a consistent view of the state is called a **consistent schedule**.

Consistency Problem

- Consider the two transactions

T1 : A + 100 -> A
B + 100 -> B

T2: A * 2 -> A
B * 2 -> B

- **Assertion** A = B.
- Although when considered alone both T1 and T2 conserve consistency, they have the following properties:
    - **temporary inconsistency**
    - **conflict**

Motivation For the Paper

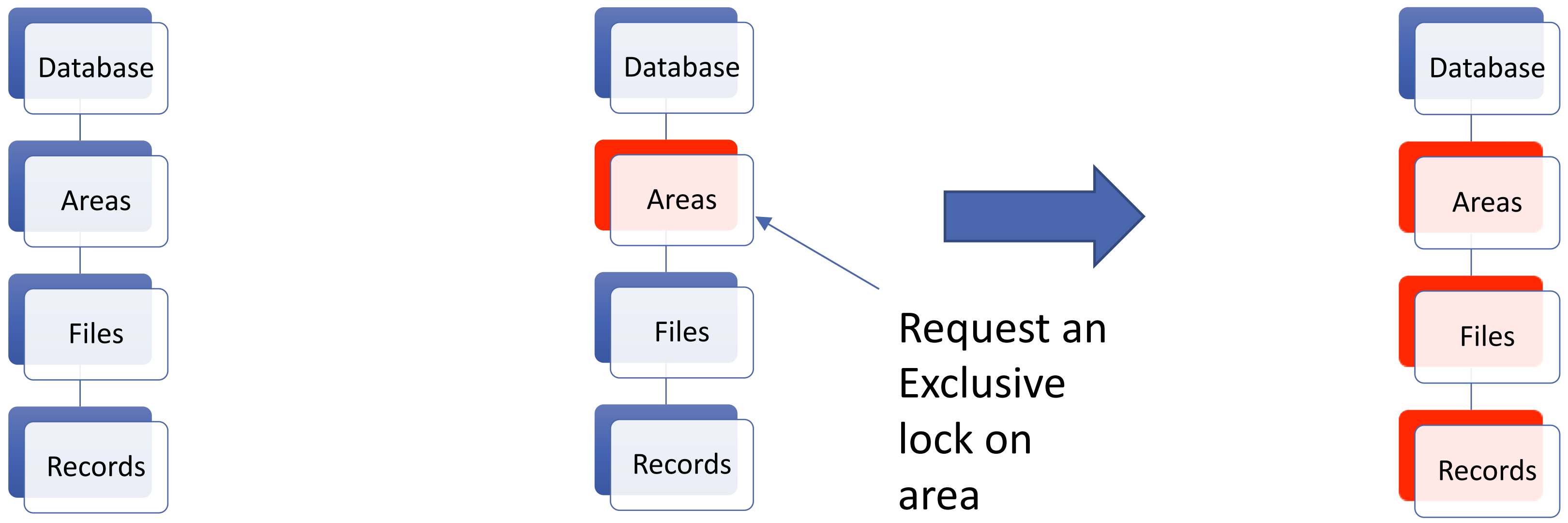
- **Problem**: What is the appropriate granularity of lockable objects in a data base?
- Small lockable objects = Increased overhead, Increased concurrency
- Larger = lower overhead, lower concurrency

Granularity of Locks

- What to choose as the the lockable units. The lockable units could be areas, files, individual records, field values, intervals of field values, etc.
- Ideally it should be as small as possible.
- However, if a lot of “lockable objects” need to be examined, there is a lot of overhead
    - Takes time to set/reset locks each time you need to look at a record
    - There is a non-zero storage overhead for representing a lock in memory
- Solution: Allow for multiple granularities of locking in the same system

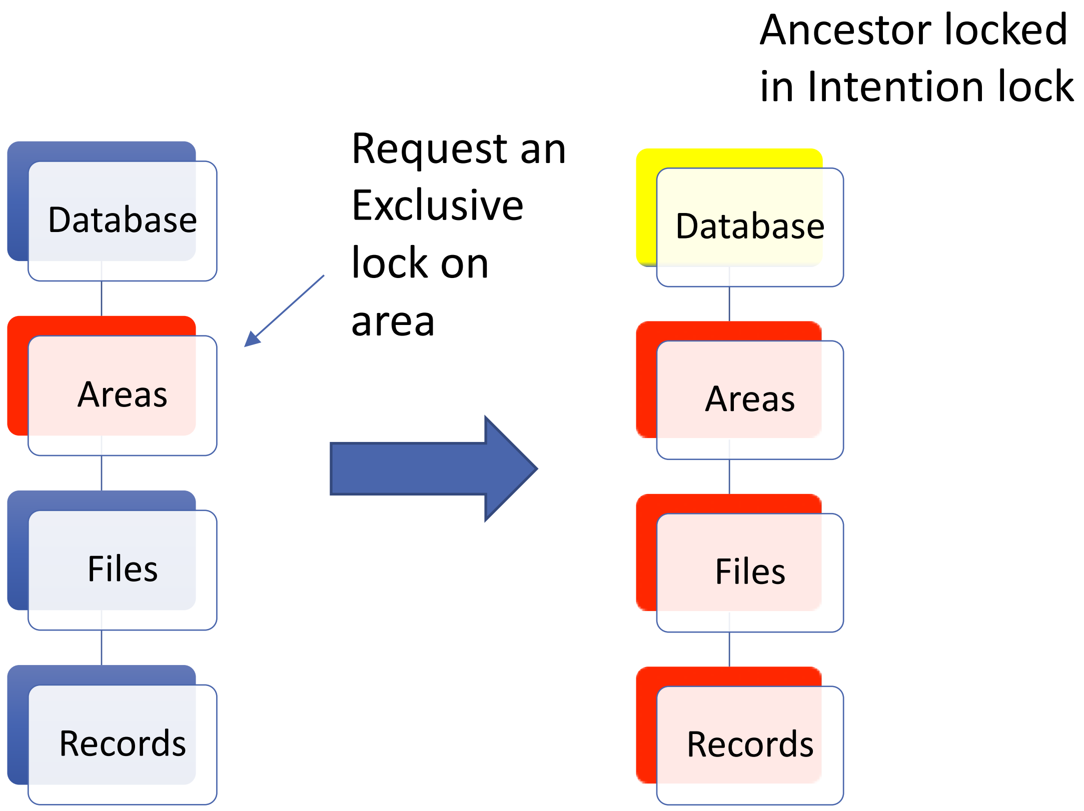
Hierarchical locks

- Each node of the hierarchy can be locked.
- There are two types of locks available
  - Shared Locks: which share the read access to locked object to other shared lock holders.
  - Exclusive Locks: which lock the object exclusively for itself
- If one requests exclusive access (X) to a particular node, then when the request is granted, the requestor has exclusive access to that node and implicitly to each of descendants.



What about ancestors?

- What if some requests locks the ancestor of already locked node?
- Intention mode(I) is used to “tag” (lock) all ancestors of a node to be locked in share or exclusive mode.
    - These tags signal the fact that locking is being done at a "finer” level.


Ancestor locked in Intention lock
Request an Exclusive lock on area
Database
Areas
Files
Records
Database
Areas
Files
Records

Intention Locking

Performance improvement: if lock on parent is weak

- **Intention shared (IS)**: to get an S lock on an item, T must first get IS locks on all containing items (↑root)
- **Intention exclusive (IX):** to get an X lock on an item, T must first get IX locks on all containing items (↑root)
- **Shared Intention Exclusive (SIX):** Equivalent to an S lock and an IX lock on an item

Intention lock indicates transaction’s intention to acquire conventional lock on a contained item

Modes of access to a Resource

- **NL** – no locks held
- **IS** – allows requestor to lock descendants in S or IS mode, does no actual locking
- **IX** – allows requestor to lock descendants in X, S, IX, IS, SIX mode, does no actual locking
- **S** – grants shared access to the node and all descendants of the node without requesting any further locks
- **SIX** – gives explicit shared access to the requested node and all descendants, also allows the requestor to further lock a descendant node in X, SIX, or IX mode
- **X** – gives explicit exclusive access to the requested node and all descendant nodes

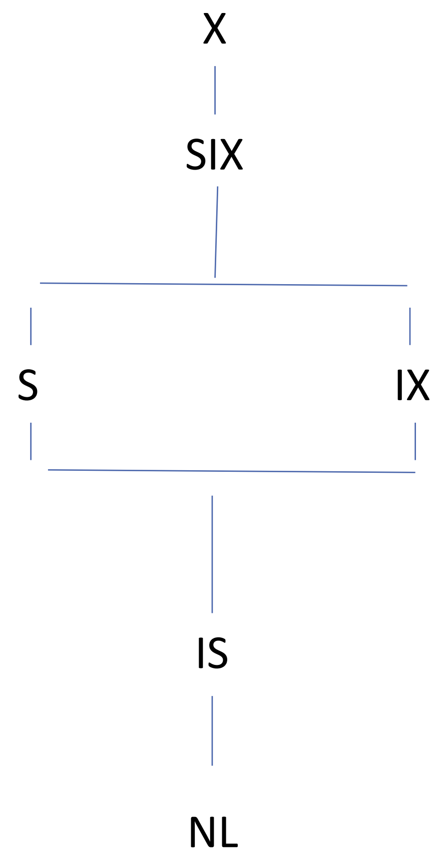
Conflict Table

Requested mode	Granted mode IS	IX	S	X	SIX
IS				x	
IX			x	x	x
S		x		x	x
X	x	x	x	x	x
SIX		x	x	x	x

- Example 1: T_2 denied an IX lock (intends to update some contained items) since T_1 is reading all contained items
- Example 2: T_2 granted IS lock even though T_1 holds IX lock (since they may be accessing different subsets of contained items)

Modes Ordering by their Privileges

- IS mode is the weakest non-null form of access to a resource.
- IX mode allows IS, IX, S, SIX and X mode locks to be set on the descendants.
- S only allows read only access to descendants.
- SIX carries privileges of both S and IX
- X is most privileged form.



Rules for requesting nodes

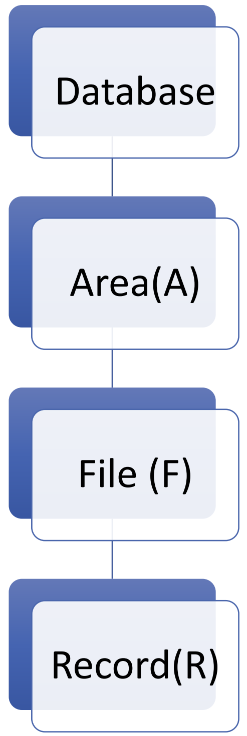
a) Before requesting an S or IS lock on a node, all ancestor nodes of the requested node must be held in IX or IS mode.

b) Before requesting an X, SIX or IX lock an a node, all ancestor nodes of the requested node must be held in SIX or IX mode.

c) Locks should be released either at the end of the transaction (in any order) or in leaf to root order.

Examples

- Lock record R for read
- Lock record R for write-exclusive access
- Lock a file F for read and write access
- Lock a file F for complete scan and occasional update

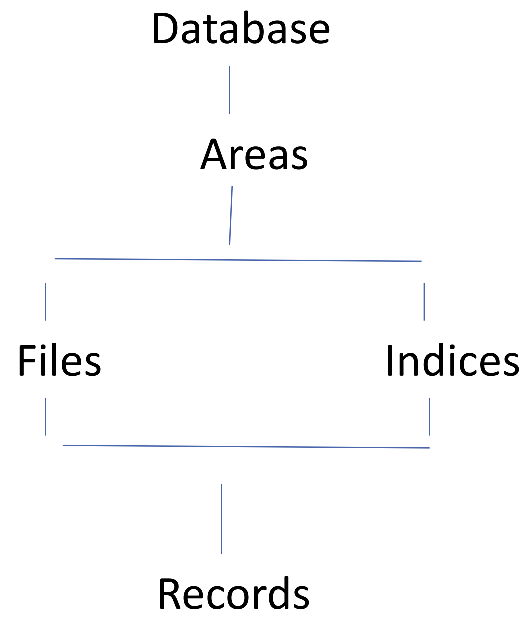


🟦 NL 🟨 IS 🟩 IX 🟦 S 🟪 SIX 🟥 X

Directed Acyclic graphs of Locks

- The tree locking hierarchy can be generalized for DAGs
- To lock a node, one should lock all the parents in the DAG

- A node is implicitly granted in S mode to a transaction if at least one of its parents is (implicitly or explicitly) granted to the transaction in S, SIX or X mode.
- A node is implicitly granted a X mode if all of its parents are (implicitly or explicitly) granted to the transaction in X node.



Requesting locks on DAG

a) Before requesting an S or IS lock on a node, one should request at least one parent (and by induction a path to a root) in IS (or greater) mode.

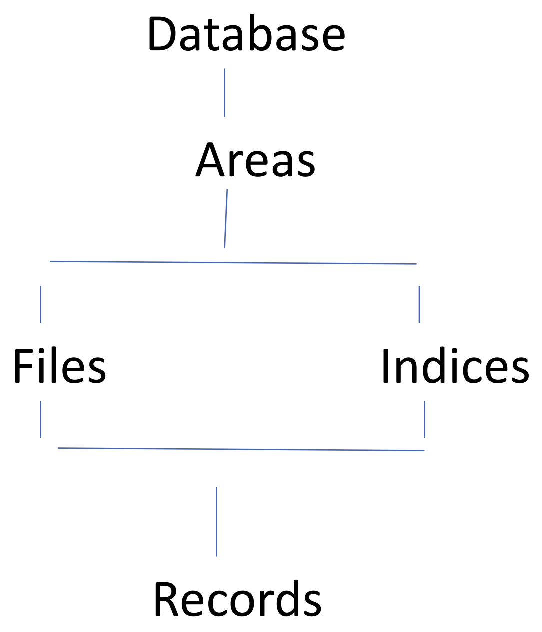
b) Before requesting IX, SIX or X mods access to a node, one should request all parents of the node in IX (or greater) mode.

c) Locks should be released either at the end of the transaction (in any order) or in leaf to root order.

"... AND THAT SHOULD COVER ALL MY RULES FOR THE CLASS."

Example DAG

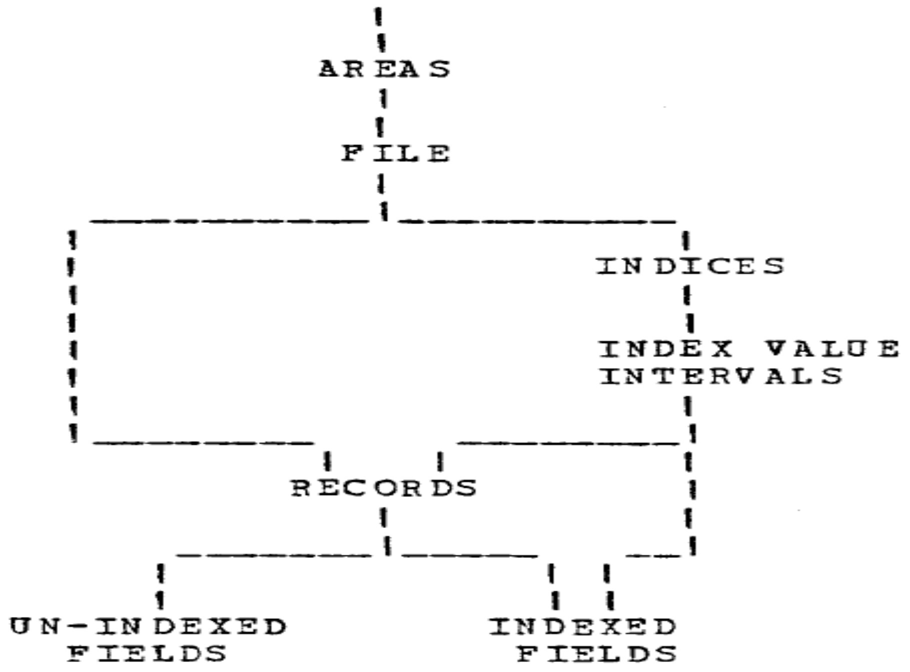
- To write a record R in file F with index I:
    - lock data base with mode = IX
    - lock area containing F with mode = IX
    - lock file F with node = IX
    - lock index I with mode = IX lock
    - record R with mode = X



But Database are Dynamic

- So far we have assumed a static database
- This is not useful, because if the database were static, locks would be unnecessary
- It is often convenient to lock one particular value of an indexed attribute
  - Index Interval locks can be used to do this
- Assumes that the indexed fields are stored separately from the unindexed fields
  - Can read the indexed values directly (without touching the actual record)

Example: Dynamic DAG



AID	BAL
1	1500
2	15000
3	800
4	2000
5	4000
6	14500

AID	LOC
4	BOS
2	CHI
1	NY
3	NY
5	NY
6	NY

Index Value Intervals

- Normal Locking protocol for DAG is extended
- When an indexed field is changed, it must “leave” the index value interval it was in and “join” a new one
- Before moving a node, the node must be locked in X mode in both its old and new position on the lock graph
  - Example: To move an account from the NY branch to the BOS branch, both the NY and BOS index value intervals would need to be locked

Equivalence of lock protocol

- The discussed lock protocol is equivalent to a conventional one, which uses only two modes (S and X) , and which locks only atomic resources (leaves of a tree or a directed graph)
- Terminology
  - Node
  - Parent
  - Source
  - Sink
  - Node-Slice

Equivalence of lock protocol

- A lock-graph L : N->M, is a mapping of explicit locks held by a particular transaction
- Projection of a lock-graph l：SI->m is set of implicit locks on atomic resources correspondingly acquired by a transaction
- Theorem:

  If two lock-graphs L1 and L2 are compatible then their projections l1 and l2 are compatible.

  In other words if the explicit locks set by two transactions are not conflicting then also the three-state locks implicitly acquired are not conflicting.

Proof (Basic Idea)

- Assume: l1 and l2 are incompatible.
- Prove: L1 and L2 are incompatible.
- Incompatibility: a sink (n) such that l1 (n) =X and l2 (n) $\in$ {S, X} (or vice versa) .
- By definition of projection there must exist a node-slice {n1 . . . ns} of n such that L1 (n1) =. . .=L1 (ns) =X.
- Also there must exist an ancestor n_0 of n such that L2 (n0) $\in$ {S,SIX,X}

Proof (Basic Idea)

- There is a path P1 from a source to n_0 on which L2 does not take the value NL.
- If P1 intersects the node-slice at n_i, then L1 and L2 are incompatible since L1(n_i) = X which is incompatible with the non null values of L2(n_i).
- Hence the theorem is proved.

Summary

- Greater lock granularity gives you finer sharing and concurrency but higher overhead.
- Providing intension locking to the ancestors improves performance.
- Basic shared and exclusive lock modes can be extended without taking a performance hit.

THANK YOU FOR YOUR ATTENTION!
ANY QUESTIONS?
memegenerator.net