CS 764: Topics in Database Management Systems

Lecture 8: Granularity of Locks

Xiangyao Yu

10/4/2020

Announcement

List of project topics updated on course website

- Please contract the instructor if you want to discuss project topics

Proposal due on **Oct. 25**

Today’s Paper: Granularity of Locks

Modelling in Data Base Management Systems, G.M. Nijssen, (ed.)
North Holland Publishing Company, 1976

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

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

IBM Research Laboratory
San Jose, California

The problem of choosing the appropriate granularity (size) of lockable objects is introduced and the tradeoff between concurrency and overhead is discussed. A locking protocol which allows simultaneous locking at various granularities by different transactions is presented. It is based on the introduction of additional lock modes besides the conventional share mode and exclusive mode. A proof is given of the equivalence of this protocol to a conventional one.

Next the issue of consistency in a shared environment is analyzed. This discussion is motivated by the realization that some existing data base systems use automatic lock protocols which insure protection only from certain types of inconsistencies (for instance those arising from transaction backup), thereby automatically providing a limited degree of consistency. Four degrees of consistency are introduced. They can be roughly characterized as follows: degree 0 protects others from your updates, degree 1 additionally provides protection from losing updates, degree 2 additionally provides protection from reading incorrect data items, and degree 3 additionally provides protection from reading incorrect relationships among data items (i.e. total protection). A discussion follows on the relationships of the four degrees to locking protocols, concurrency, overhead, recovery and transaction structure.

Lastly, these ideas are compared with existing data management systems.

I. GRANULARITY OF LOCKS:

An important issue which arises in the design of a data base management system is the choice of lockable units, i.e. the data aggregates which are atomically locked to insure consistency. Examples of lockable units are areas, files, individual records, field values, and intervals of field values.

The choice of lockable units presents a tradeoff between concurrency and overhead, which is related to the size or granularity of the units themselves. On the one hand, concurrency is increased if a fine lockable unit (for example a record or field) is chosen. Such unit is appropriate for a "simple" transaction which accesses few records. On the other hand a fine unit of locking would be costly for a "complex" transaction which accesses a large number of records. Such a transaction would have to set and reset a large

365

Modelling in Data Base Management Systems 1976

Agenda

Transaction basics

Locking granularity

Two-phase locking

Degree of consistency

ACID Properties in Transactions

A sequence of many actions considered to be one atomic unit of work

Atomicity: Either all operations occur, or nothing occurs (all or nothing)

Consistency: Integrity constraints are satisfied

Isolation: How operations of transactions interleave

Durability: A transaction’s updates persist when system fails

This lecture touches A, C, and I

Locking Granularity

Use locks to prevent conflicts

Locking Granularity

Use locks to prevent conflicts

Choosing a locking granularity

- Entire database
- Relation
- Records …

Locking Granularity

Use locks to prevent conflicts

Choosing a locking granularity

- Entire database
- Relation
- Records …

Increasing concurrency
Increasing overhead when many records are accessed

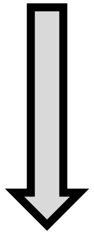
Goal: high concurrency and low cost

Locking Granularity

Use locks to prevent conflicts

Choosing a locking granularity

- Entire database
- Relation
- Records …



Increasing concurrency
Increasing overhead when many records are accessed

Goal: high concurrency and low cost

Solution: **Hierarchical locks**

Hierarchical Locks

```
   DB                        DB
    |                         |
  Areas                     Areas
    |                      /     \
  Files               Files    Indices
    |                      \     /
 Records                   Records
```

Lock a high-level node if a large number of records are accessed

- All descendants are implicitly locked in the same mode

Hierarchical Locks

```
    DB                    DB
    |                     |
  Areas                 Areas
    |                  /     \
  Files            Files    Indices
    |                  \     /
 Records               Records
```

Lock a high-level node if a large number of records are accessed

- All descendants are implicitly locked in the same mode
- **Intention lock** to avoid conflict with implicit locks

Locking Modes

Basic locking modes

- S: Shared lock
- X: Exclusive lock

Locking Modes

Basic locking modes

- S: Shared lock
- X: Exclusive lock

Intention modes:

- IS: Intention to share
- IX: Intention to acquire X lock below the lock hierarchy
- SIX: Read large portions and update a few parts

Locking Modes

Basic locking modes

- S: Shared lock
- X: Exclusive lock

Intention modes:

- IS: Intention to share
- IX: Intention to acquire X lock below the lock hierarchy
- SIX: Read large portions and update a few parts

Example: read record

DB	**IS**
Areas	**IS**
Files	**IS**
Records	**S**

Locking Modes

Basic locking modes

- S: Shared lock
- X: Exclusive lock

Intention modes:

- IS: Intention to share
- IX: Intention to acquire X lock below the lock hierarchy
- SIX: Read large portions and update a few parts

Example: read record update record

	read record	update record
DB	IS	IX
Areas	IS	IX
Files	IS	IX
Records	S	X

Locking Modes

Basic locking modes

- S: Shared lock
- X: Exclusive lock

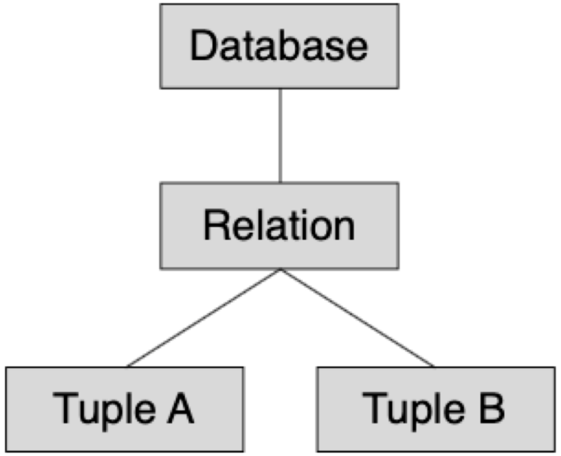
Intention modes:

- IS: Intention to share
- IX: Intention to acquire X lock below the lock hierarchy
- SIX: Read large portions and update a few parts

Example:	read record	update record	scan + occasional updates
DB	**IS**	**IX**	**IX**
Areas	**IS**	**IX**	**IX**
Files	**IS**	**IX**	**SIX**
Records	**S**	**X**	**lock specific records in X mode**

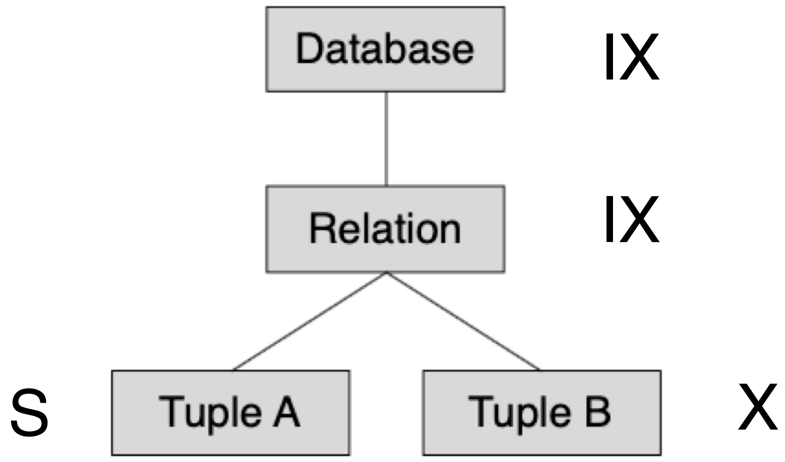
Example

a) [**10 points**] Consider the following locking hierarchy where there is a single database that contains a single table and the table contains two tuples: A and B. If a transaction T1 reads tuple A and writes tuple B, what lock modes (e.g., NL, S, X, IS, IX, SIX) will T1 hold on the tuples, the table, and the database, respectively?



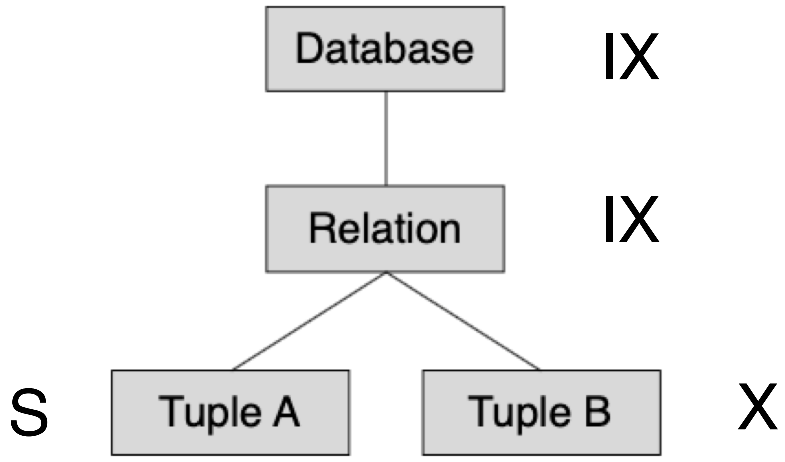
Example

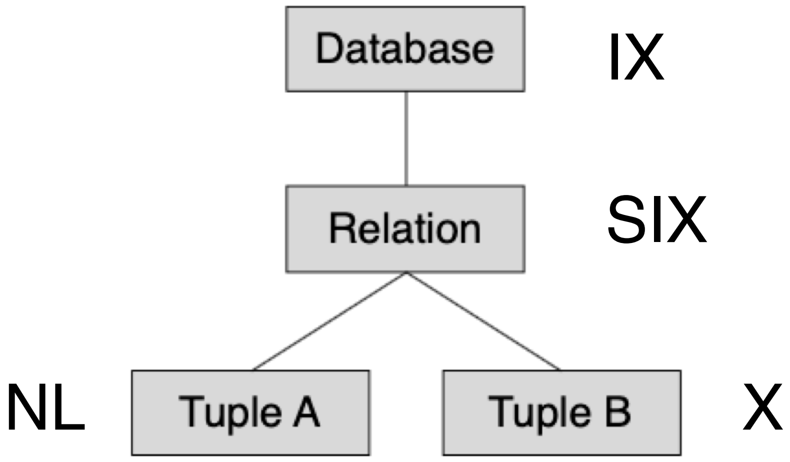
a) [**10 points**] Consider the following locking hierarchy where there is a single database that contains a single table and the table contains two tuples: A and B. If a transaction T1 reads tuple A and writes tuple B, what lock modes (e.g., NL, S, X, IS, IX, SIX) will T1 hold on the tuples, the table, and the database, respectively?



Example

a) [**10 points**] Consider the following locking hierarchy where there is a single database that contains a single table and the table contains two tuples: A and B. If a transaction T1 reads tuple A and writes tuple B, what lock modes (e.g., NL, S, X, IS, IX, SIX) will T1 hold on the tuples, the table, and the database, respectively?



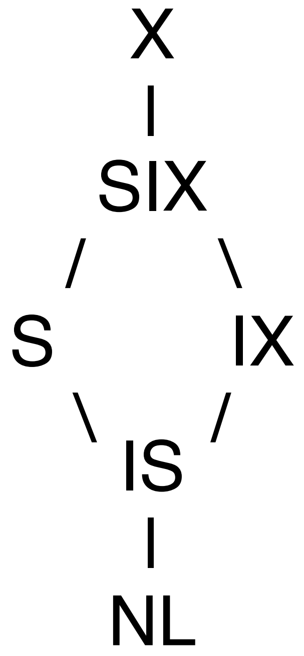


Lock Compatibility

Increasing lock strength →

	IS	IX	S	SIX	X
IS	Y	Y	Y	Y	N
IX	Y	Y	N	N	N
S	Y	N	Y	N	N
SIX	Y	N	N	N	N
X	N	N	N	N	N

Most privileged

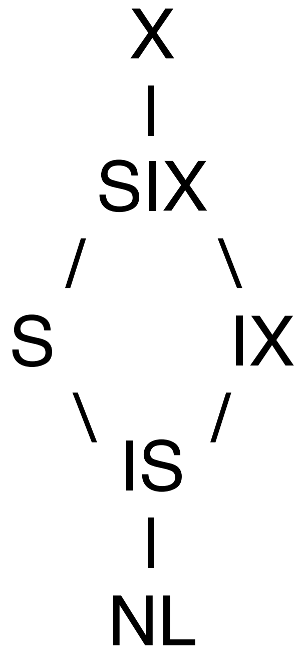


least privileged

Lock Compatibility

Increasing lock strength →

	IS	IX	S	SIX	X
IS	Y	Y	Y	Y	N
IX	Y	Y	N	N	N
S	Y	N	Y	N	N
SIX	Y	N	N	N	N
X	N	N	N	N	N



Most privileged

least privileged

Rules for Lock Requests

- Before requesting S or IS on a node, all ancestor nodes of the requested node must be held in IS or IX

Rules for Lock Requests

- Before requesting S or IS on a node, all ancestor nodes of the requested node must be held in IS or IX

- Before requesting X, SIX, or IX on a node, all ancestor nodes of the requesting node must be held in SIX or IX

Rules for Lock Requests

- Before requesting S or IS on a node, all ancestor nodes of the requested node must be held in IS or IX

- Before requesting X, SIX, or IX on a node, all ancestor nodes of the requesting node must be held in SIX or IX

- Locks requested **root to leaf**
- Locks released **leaf to root** or any order at the end of the transaction (as an atomic operation)

Extension – Semantic Locking

A system can introduce new lock types based on the operation semantics

Extension – Semantic Locking

Example: increment lock

	S	INC	X
S	Y	N	N
INC	N	Y	N
X	N	N	N

A system can introduce new lock types based on the operation semantics

Example:

- Increment and decrement values

Extension – Semantic Locking

Example: increment lock

	S	INC	X
S	Y	N	N
INC	N	Y	N
X	N	N	N

Example: compare with constant

	S	COMP	X
S	Y	Y	N
COMP	Y	Y	**depends**
X	N	**depends**	N

A system can introduce new lock types based on the operation semantics

Example:

- Increment and decrement values
- Test value is greater than X

Schedule and Granting Requests

Queue of requests

IS — IX — IS — IS — IS —S — IS— X — IS —IX

granted group waiting requests

To avoid starvation (where a transaction is delayed indefinitely), each request waits its turn in the queue

Deadlock

tuple A

T1.S — T2.X # T2 waits for T1

tuple B

T2.S — T1.X # T1 waits for T2

Deadlocks Solutions

Deadlock detection: Once a cycle is detected, abort a transaction in the cycle

Deadlocks Solutions

Deadlock detection: Once a cycle is detected, abort a transaction in the cycle

No-Wait: A transaction self-aborts when encountering a conflict

Deadlocks Solutions

Deadlock detection: Once a cycle is detected, abort a transaction in the cycle

No-Wait: A transaction self-aborts when encountering a conflict

Wait-Die: On a conflict, the requesting transaction **waits** if it has higher priority than transactions in the queue, otherwise the requesting transaction **self-aborts**

Deadlocks Solutions

Deadlock detection: Once a cycle is detected, abort a transaction in the cycle

No-Wait: A transaction self-aborts when encountering a conflict

Wait-Die: On a conflict, the requesting transaction **waits** if it has higher priority than transactions in the queue, otherwise the requesting transaction **self-aborts**

Wound-Wait: On a conflict, the requesting transaction **preemptively aborts** current owners if it has higher priority, otherwise the requesting transaction **waits**

Serializability

Concurrent execution of transactions produces the same results as some serial execution

- Intuitive and easy to reason about

Two-Phase Locking (2PL)

Two-phase locking (2PL) ensures serializability

- Growing phase: acquiring locks (no release)
- Shrinking phase: releasing locks (no acquire)

Two-Phase Locking (2PL)

Two-phase locking (2PL) ensures serializability

- Growing phase: acquiring locks (no release)
- Shrinking phase: releasing locks (no acquire)
- Serialization point: after all locks are acquired but before any release
- The equivalent serial order = order of transactions' serialization points

Two-Phase Locking (2PL)

Two-phase locking (2PL) ensures serializability

- Growing phase: acquiring locks (no release)
- Shrinking phase: releasing locks (no acquire)
- Serialization point: after all locks are acquired but before any release
- The equivalent serial order = order of transactions' serialization points

Strict 2PL: 2PL + all exclusive locks released *after* transaction commits

- Widely used scheme in practice

Degree of Consistency (Isolation)

Degree 3: Serializability (assuming no phantom effect)

- Two-phase with respective to both reads and writes

Degree of Consistency (Isolation)

Degree 3: Serializability (assuming no phantom effect)

- Two-phase with respective to both reads and writes

Degree 2: Read Committed

- Two-phase with respect to writes
- Short read locks

Degree of Consistency (Isolation)

Degree 3: Serializability (assuming no phantom effect)

- Two-phase with respective to both reads and writes

Degree 2: Read Committed

- Two-phase with respect to writes
- Short read locks

Degree 1: Read Uncommitted

- Two-phase with respect to writes
- No read locks (may observe dirty data)

Degree of Consistency (Isolation)

Degree 3: Serializability (assuming no phantom effect)

- Two-phase with respective to both reads and writes

Degree 2: Read Committed

- Two-phase with respect to writes
- Short read locks

Degree 1: Read Uncommitted

- Two-phase with respect to writes
- No read locks (may observe dirty data)

Degree 0:

- Short write locks
- No read locks

Degree of Consistency (Isolation)

Degree 3: Serializability (assuming no phantom effect)

- Two-phase with respective to both reads and writes

Degree 2: Read Committed

- Two-phase with respect to writes
- Short read locks

Degree 1: Read Uncommitted

- Two-phase with respect to writes
- No read locks (may observe dirty data)

Degree 0:

- Short write locks
- No read locks

Increasing concurrency

Weaker guarantees

Q/A – Granularity of Locks

Optimal schedule based on knowledge of the workload?

Intention locks used today?

Phantom effect?

Paper hard to follow…

Before Next Lecture

Submit review for

- Hal Berenson, et al., A Critique of ANSI SQL Isolation Levels. SIGMOD Record, 1995