More issues with locks

Topic 6 Lesson 5
Granularity of lock, intention locks

Granularity of locked data item

Granularity: size of data items chosen as unit of protection by **concurrency control protocol**.

Ranging from coarse to fine. You can lock:

1. The entire database.
2. A file – which typically maps to a table.
3. A page (or area or collection of tuples in a table).
4. A record (one tuple in a table).
5. A cell in a record (one field in one tuple of a specific table).

Choosing a data item size

What sized data item for the locking protocol?

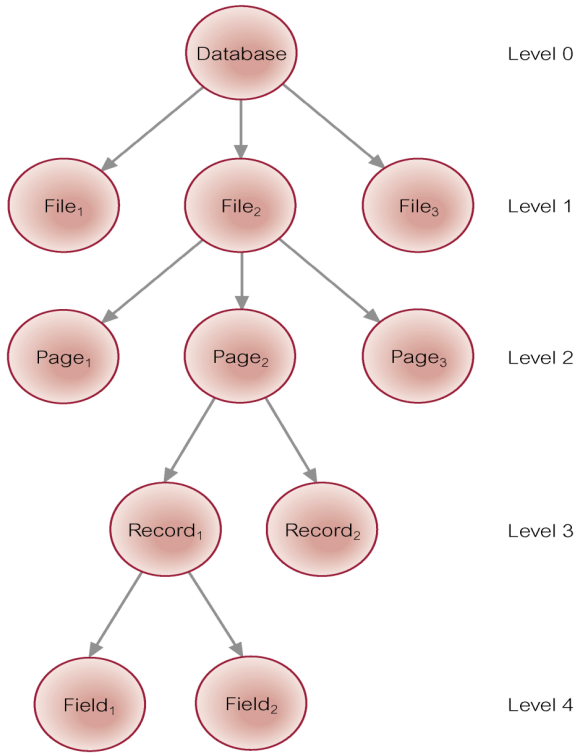
Tradeoff:

- Coarser data size leads to a lower degree of concurrency
- Finer data size leads to more locking information that is needed to be stored.

Best item size depends on the types of transactions. Most DBMS lock at the tuple level (so shared and exclusive locks per accessed tuple)

Need to choose which object level to lock

1. Top level locks the whole database.
2. Next level (file) locks a table
3. Next level (page) locks a part of a table
4. Next level locks a tuple in a table
5. Next level locks a field in a tuple



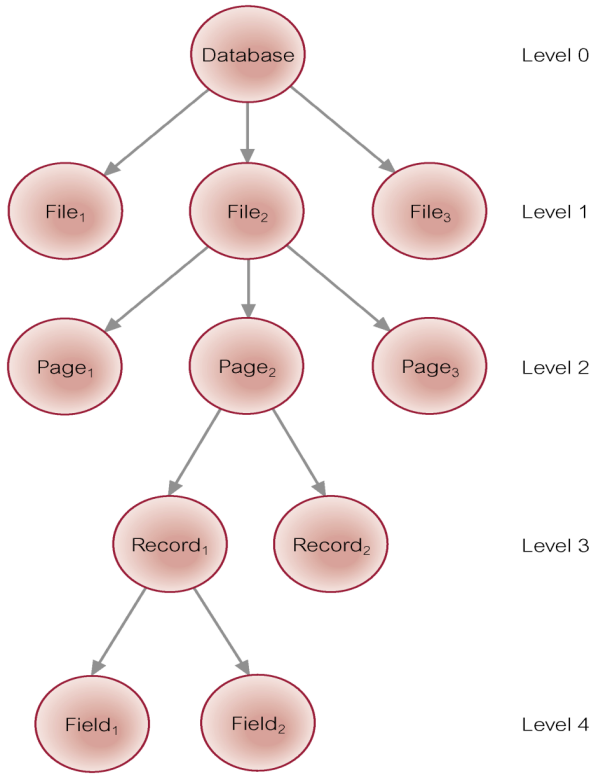
Hierarchy of database items

Since the objects are hierarchical, we could have hierarchical locks mapping to the hierarchical structure.

Root node locks the entire database, next level locks a table etc.

When node is locked, all its descendants are also locked.

The DBMS should check the complete hierarchical path to the item before granting the lock.



Addressing the hierarchical structure

One solution: **Intention lock** could be used to lock all ancestors to a soon to be locked node.

Intention locks can be shared or exclusive. They are granted top-down and released bottom-up.

	IS	IX	S	SIX	X
IS	✓	✓	✓	✓	✗
IX	✓	✓	✗	✗	✗
S	✓	✗	✓	✗	✗
SIX	✓	✗	✗	✗	✗
X	✗	✗	✗	✗	✗

✓ = compatible, ✗ = incompatible

Solving the phantom read problem

T1 reads min(bal) from Account table, A is a tuple with minimum value 10 in table Account,
T2 then writes a smaller bal to Account table
T1 repeats min(bal) and gets a different value

Time	Transaction 1 (T1)	Transaction 2 (T2)
t_1	Share_lock(min(bal) from account)	
t_2	Read(min(bal) from account) Returns(A)	No tuple to lock since B is not in the DB
t_2		Insert(B with value 5 for bal in Account)
t_4	Read(min(bal) from account) Returns(B)	

Intention locks solve phantom read problem

Time	Transaction 1	Transaction 2
t_1	IS(Account) Share_lock(Read(min(bal) from account))	
t_1	Read(min(bal) from account) Returns(A)	
		IX(Account)
t_2		WAIT
t_3	Read(min(bal) from account) Returns(A)	WAIT
t_4	Unlock(A) Unlock(Account)	Insert(B with value 5 in account)

Since a shared lock is incompatible with an intent exclusive lock T2 must wait for T1 to release its lock

Summary

Locks are a concurrency control solution; however for a complete concurrency control solution with locks we need:

1. An algorithmic solution to deadlocks
   - We can **prevent** deadlocks by placing an ordering on the transactions and allowing the order to determine which transactions can wait
   - We can **detect** a deadlock via a wait-for graph
2. A representation of the hierarchical structure of data objects in a database management system
   - Intention locks address the hierarchy, allows us to set locks on higher levels of objects in the database