Lecture 45
Multiple Granularity Locking Protocol

Multiple Granularity Locking Protocol

- All concurrency control techniques assume that the database is formed of a number of named data items. A database item could be chosen to be one of the following:
  - A database record
  - A field value of a database record
  - A disk block
  - A whole file
  - The whole database
- The particular choice of data item type can affect the performance of concurrency control and recovery.
- The size of data items is often called the **data item granularity**.
- ***Fine granularity*** refers to small item sizes, whereas ***coarse granularity*** refers to large item sizes.

Multiple Granularity Locking Protocol

- First, notice that the larger the data item size is, the lower the degree of concurrency permitted.
- For example, if the data item size is a disk block, a transaction *T* that needs to lock a single record *B* must lock the whole disk block *X* that contains *B* because a lock is associated with the whole data item (block).
- Now, if another transaction *S* wants to lock a different record *C* that happens to reside in the same disk block *X* in a conflicting lock mode, it is forced to wait.
- If the data item size was a single record instead of a disk block, transaction *S* would be able to proceed, because it would be locking a different data item (record).

Multiple Granularity Locking Protocol

- On the other hand, the smaller the data item size is, the more the number of items in the database. Because every item is associated with a lock, the system will have a larger number of active locks to be handled by the lock manager.
- More lock and unlock operations will be performed, causing a higher overhead.
- In addition, more storage space will be required for the lock table.
- For timestamps, storage is required for the read_TS an write_TS for each data item, and there will be similar overhead for handling a large number of items.

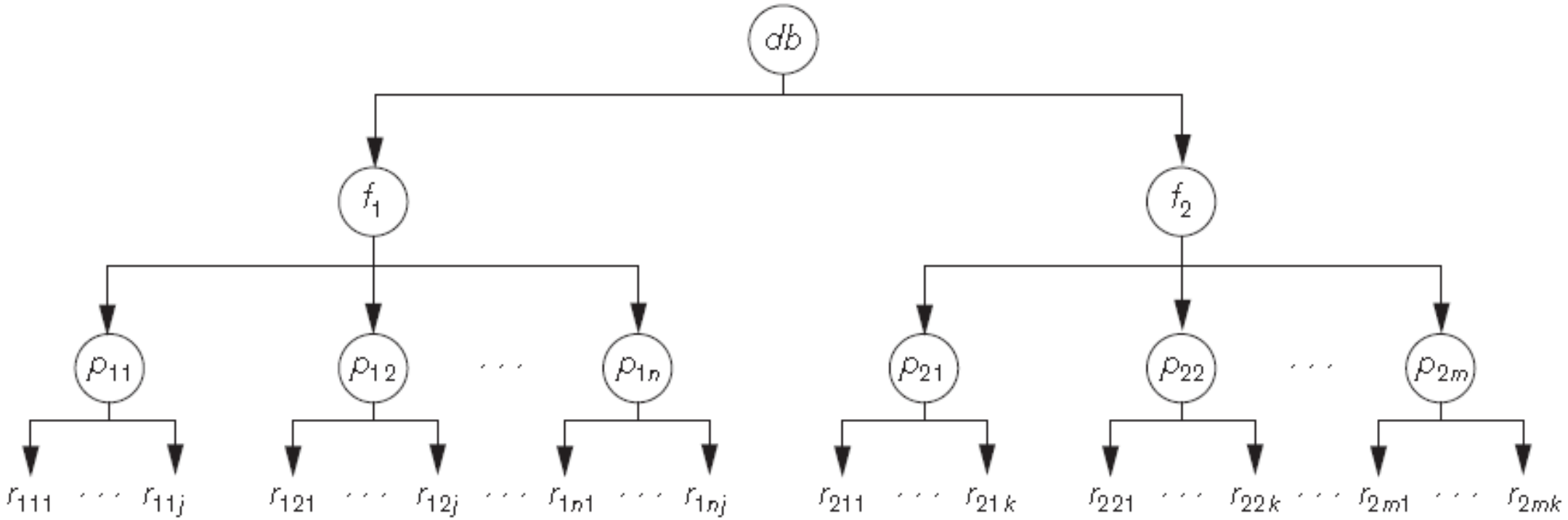
Multiple Granularity Locking Protocol

- Given the above tradeoffs, an obvious question can be asked:
  - What is the best item size?
  - The answer is that it *depends on the types of transactions involved*.
- If a typical transaction accesses a small number of records, it is advantageous to have the data item granularity be one record.
- On the other hand, if a transaction typically accesses many records in the same file, it may be better to have block or file granularity so that the transaction will consider all those records as one (or a few) data items.

Multiple Granularity Locking Protocol

- Since the best granularity size depends on the given transaction, it seems appropriate that a database system should support multiple levels of granularity, where the granularity level can be adjusted dynamically for various mixes of transactions.
- Figure below shows a simple granularity hierarchy with a database containing two files, each file containing several disk pages, and each page containing several records.
- This can be used to illustrate a **multiple granularity level** 2PL protocol, with shared/exclusive locking modes, where a lock can be requested at any level. However, additional types of locks will be needed to support such a protocol efficiently.

Multiple Granularity Locking Protocol



Multiple Granularity Locking Protocol

- To make multiple granularity level locking practical, additional types of locks, called **intention locks**, are needed.
- The idea behind intention locks is for a transaction to indicate, along the path from the root to the desired node, what type of lock (shared or exclusive) it will require from one of the node's descendants.
- There are three types of intention locks:
  1. **Intention-shared (IS)** indicates that one or more shared locks will be requested on some descendant node(s).
  2. **Intention-exclusive (IX)** indicates that one or more exclusive locks will be requested on some descendant node(s).
  3. **Shared-intention-exclusive (SIX)** indicates that the current node is locked in shared mode but that one or more exclusive locks will be requested on some descendant node(s).

Multiple Granularity Locking Protocol

- The compatibility table of the three intention locks, and the actual shared and exclusive locks, is shown in Figure below.

	IS	IX	S	SIX	X
IS	YES	YES	YES	YES	NO
IX	YES	YES	NO	NO	NO
S	YES	NO	YES	NO	NO
SIX	YES	NO	NO	NO	NO
X	NO	NO	NO	NO	NO

Multiple Granularity Locking Protocol

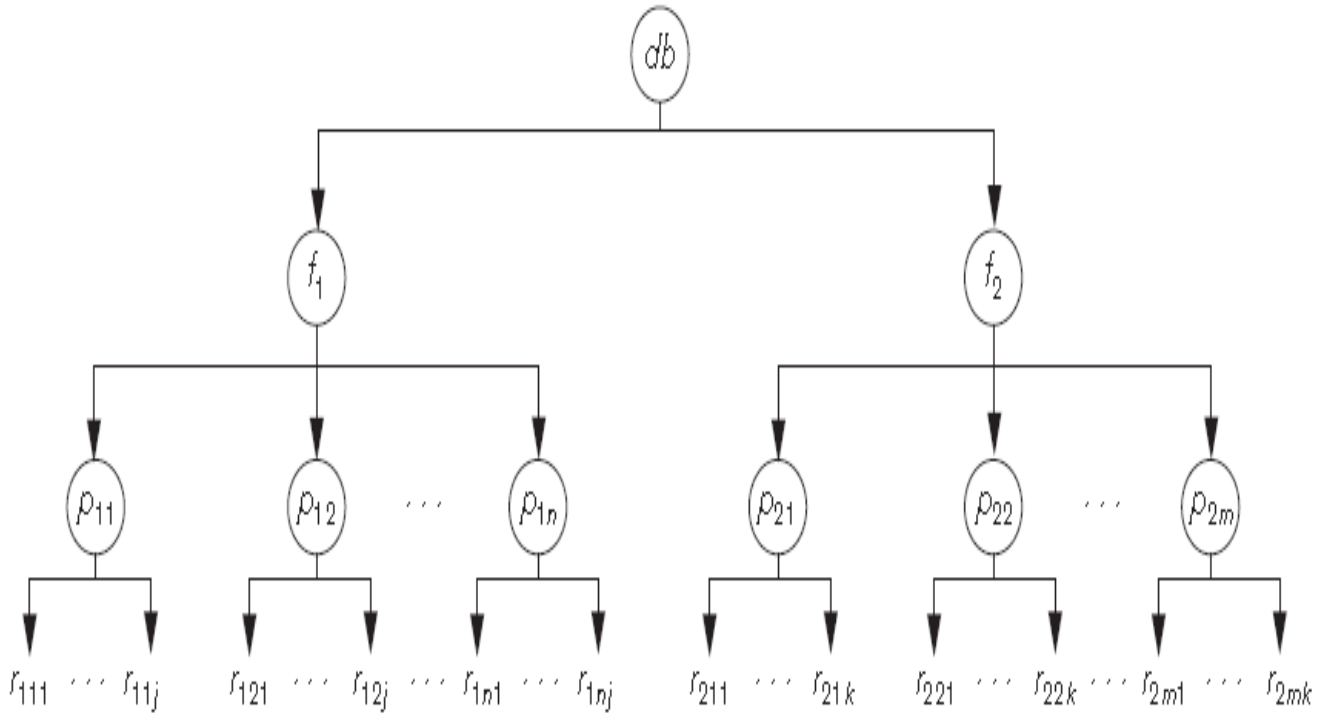
- In addition to the three types of intention locks, an appropriate locking protocol must be used. The **multiple granularity locking (MGL)** protocol consists of the following rules :
  1. The lock compatibility must be adhered to.
  2. The root of the tree must be locked first, in any mode.
  3. A node *N* can be locked by a transaction *T* in S or IS mode only if the parent node *N* is already locked by transaction *T* in either IS or IX mode.
  4. A node *N* can be locked by a transaction *T* in X, IX, or SIX mode only if the parent of node *N* is already locked by transaction *T* in either IX or SIX mode.
  5. A transaction *T* can lock a node only if it has not unlocked any node (to enforce the 2PL protocol).
  6. A transaction *T* can unlock a node, *N*, only if none of the children of node *N* are currently locked by *T.*

Multiple Granularity Locking Protocol

- Rule 1 simply states that conflicting locks cannot be granted.
- Rules 2, 3, and 4 state the conditions when a transaction may lock a given node in any of the lock modes.
- Rules 5 and 6 of the MGL protocol enforce 2PL rules to produce serializable schedules.
- Basically, the locking *starts from the root* and goes down the tree until the node that needs to be locked is encountered, whereas unlocking *starts from the locked node* and goes up the tree until the root itself is unlocked.

Multiple Granularity Locking Protocol

1. *T*1 wants to update record *r*111 and record *r*211.
2. *T*2 wants to update all records on page *p*12.
3. *T*3 wants to read record *r*11*j* and the entire *f*2



T_1	T_2	T_3
IX(*db*)		
IX(f_1)		
	IX(*db*)	
		IS(*db*)
		IS(f_1)
		IS(p_{11})
IX(p_{11})		
X(r_{111})		
	IX(f_1)	
	X(p_{12})	
		S(r_{11j})
IX(f_2)		
IX(p_{21})		
X(p_{211})		
unlock(r_{211})		
unlock(p_{21})		
unlock(f_2)		
		S(f_2)
	unlock(p_{12})	
	unlock(f_1)	
	unlock(*db*)	
unlock(r_{111})		
unlock(p_{11})		
unlock(f_1)		
unlock(*db*)		
		unlock(r_{11j})
		unlock(p_{11})
		unlock(f_1)
		unlock(f_2)
		unlock(*db*)

For Video lecture on this topic please subscribe to my youtube channel.

The link for my youtube channel is

https://www.youtube.com/channel/UCRWGtE76JlTp1iim6aOTRuw?sub_confirmation=1