**Scheduler**

Implements lock manager and deadlock detection

Data structures:

**lock\_table** - hash of lock entries

num\_trans = number currently holding a lock

type = 0 (shared) or 1 (exclusive)

wait\_queue = pointer to queue of pending lock requests

**wfg\_matrix** - keep track of dependencies

(TID, wait\_for\_TID)

Variables:

Functions:

// Ensures strict 2PL and then passes command on to Data Manager

// Return value passed back by Data Manager or blocked status

**handle\_command**(command, TID, data\_item) {

type = use command to determine which lock needed

if (read/mult\_read/write/delete)

if (**check\_lock**(type,TID, data\_item))

pass on to **Data Manager**

else

lock\_status = req\_lock(type, TID,data\_item)

if (lock\_status == failure)

add to lock\_table, wfg\_matrix

return blocked

else

pass on to **Data Manager**

else if (commit or abort)

pass on to **Data Manager**

**release\_locks**(TID)

else

error, unknown command

}

// Release locks TID has acquired:

Approve a pending request

Update lock\_table, wfg\_matrix

**release\_locks**(TID)

// Return true if TID has lock of type on data\_item; othewise, return false

**check\_lock**(type, TID, data\_item)

// Attempt to acquire lock of type on data\_item; return true on success, else false

**req\_lock**(type, TID, data\_item)