File Structures and Physical Database Design, Query Processing, and Transaction Management

Week 6

COS60009: Data Management for the Big Data Age

1

Learning Objectives

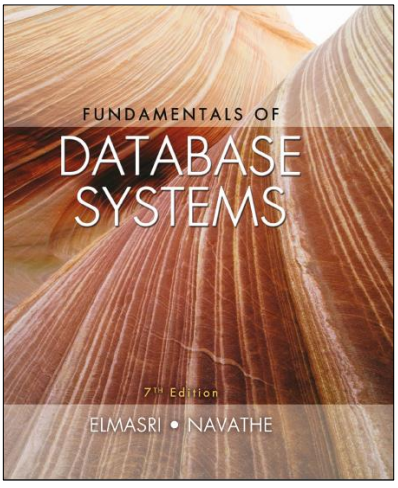
- Storage Organization and Index
- Physical Database Design in Relational Databases
- Query Processing and Optimization
- Transactions
  - Concepts
  - ACID Properties
  - Recoverability
  - Serializability
- Concurrency Control
  - 2PL Protocol
- Recovery
  - Deferred and Immediate Update Techniques

Pearson

2

Fundamentals of Database Systems

Seventh Edition



Chapter 16 - 22

File Structures and Physical Database Design, Query Processing, and Transaction Management

Pearson

3

Storage Organization of Databases

- Databases typically stored on magnetic disks
  - Accessed using physical database file structures
- Storage hierarchy
  - Primary storage: CPU main memory, cache memory
  - Secondary storage: magnetic disks, flash memory, solid-state drives
- Persistent data - most databases
- Transient data - exists only during program execution
- File organization
  - Determines how records are physically placed on the disk
    - Record: collection of related data values or items
    - Values correspond to record field
    - Data types: Numeric, String, Boolean, Date/time, Binary large objects (BLOBs) for unstructured objects
  - Determines how records are accessed
    - Retrieval operations - No change to file data, records selected based on selection condition
    - Update operations - File change by insertion, deletion, or modification
    - Operations for accessing file records - Open, Find, Read, FindNext, Delete, Insert, Close, Scan

Pearson

4

Index

- Indexes used to speed up record retrieval in response to certain search conditions
- Index structures provide secondary access paths
- Any field can be used to create an index
  - Multiple indexes can be constructed
- Most indexes based on ordered files
- General form of the command to create an index
  - Unique and cluster keywords optional
  - Order can be ASC or DESC

```
CREATE [ UNIQUE ] INDEX <index name>
ON <table name> ( <column name> [ <order> ] { , <column name> [ <order> ] } )
[ CLUSTER ] ;
```

Pearson

5

Types of Single-Level Ordered Indexes

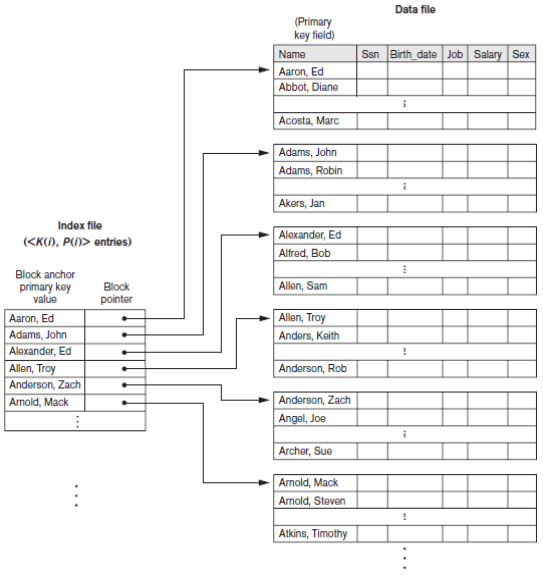
- Ordered index similar to index in a textbook
- Indexing field (attribute)
  - Index stores each value of the index field with list of pointers to all disk blocks that contain records with that field value
- Values in index are ordered
- Primary index
  - Specified on the *ordering key field* of *ordered file of records*
- Clustering index
  - Used if numerous records can have the same value for the *ordering field* of the file
- A data file can have only one primary index or clustering index, not both, because there is only one *ordering field* of the file
- Secondary index
  - Can be specified on any *nonordering field* of the file
  - A data file can have several secondary indexes

Pearson

6

Primary Indexes

- Ordered file with two fields
  - Primary key, $K(i)$
  - Pointer to a disk block, $P(i)$
- One index entry in the index file for each block in the data file
- Indexes may be dense or sparse. Dense index has an index entry for every search key value in the data file. Sparse index has entries for only some search values
- Major problem: insertion and deletion of records need to move records around and change index values
  - Solutions: Use unordered overflow file or linked list of overflow records

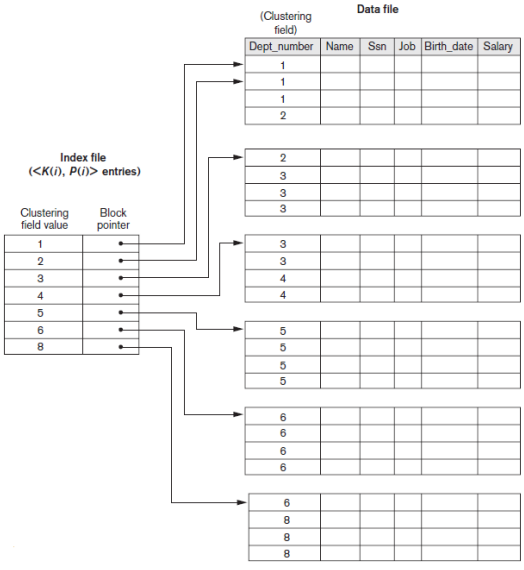


Pearson

7

Clustering Indexes

- Clustering field
  - Data file records are physically ordered on a nonkey field (as clustering field) without a distinct value for each record
- Ordered index file with two fields
  - Same type as clustering field
  - Disk block pointer

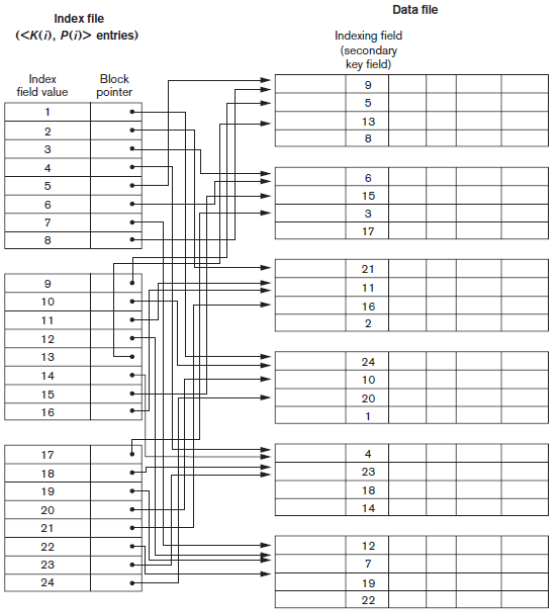


Pearson

8

Secondary Indexes

- Provide secondary means of accessing a data file
  - Some primary access exists
- Data file records are not physically ordered on the index field
- Ordered index file with two fields
  - Indexing field, $K(i)$
  - Block pointer or record pointer, $P(i)$
- Usually need more storage space and longer search time than primary index
  - Improved search time for arbitrary record



Pearson

9

Dynamic Multilevel Indexes Using B-Trees and B+-Trees

- Tree data structure terminology
  - Tree is formed of nodes
  - Each node (except root) has one parent and zero or more child nodes
  - Leaf node has no child nodes
    - Unbalanced if leaf nodes occur at different levels
  - Nonleaf node called internal node
  - Subtree of node consists of node and all descendant nodes

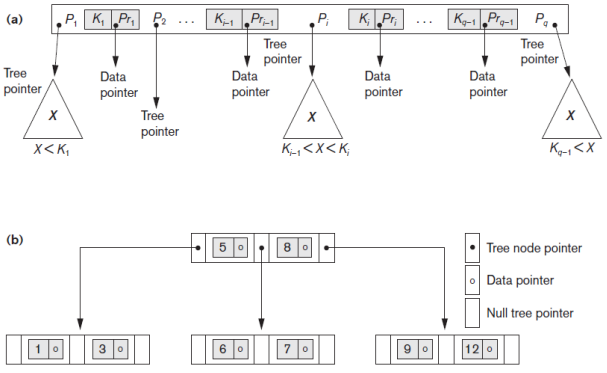
Pearson

10

B-Trees

- Provide multi-level access structure
- Tree is always balanced
- Space wasted by deletion never becomes excessive
  - Each node is at least half-full
- Each node in a B-tree of order *p* can have at most *p−1* search values

Figure 17.10 B-tree structures (a) A node in a B-tree with *q−1* search values (b) A B-tree of order *p=3*. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6



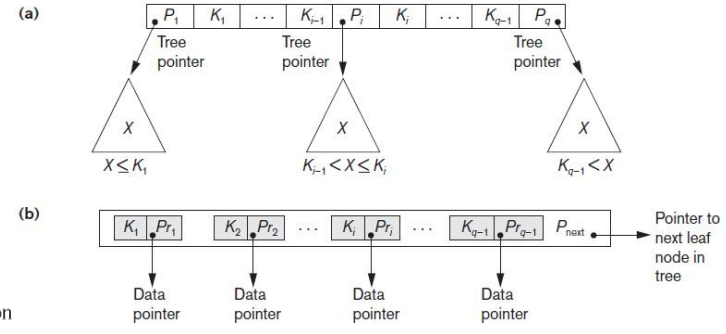
Pearson

11

B+ -Trees

- Data pointers stored only at the leaf nodes
  - Leaf nodes have an entry for every value of the search field, and a data pointer to the record if search field is a key field
  - For a nonkey search field, the pointer points to a block containing pointers to the data file records
- Internal nodes
  - Some search field values from the leaf nodes repeated to guide search

Figure 17.11 The nodes of a B+-tree (a) Internal node of a B+-tree with *q−1* search values (b) Leaf node of a B+-tree with *q−1* search values and *q−1* data pointers

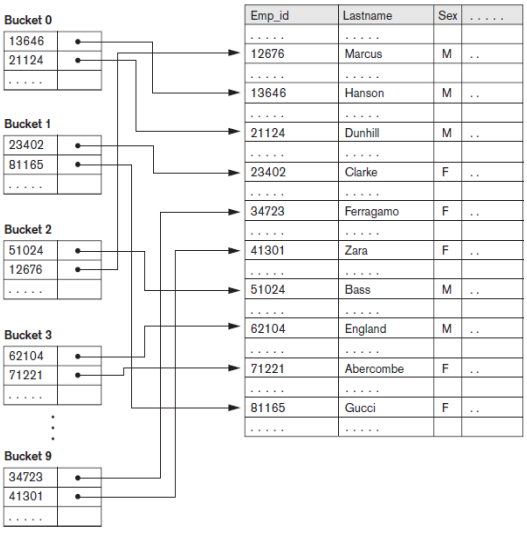


Pearson

12

Hash indexes

- Apply hash function (randomizing function) to hash key value of a record, yield address of the disk block of stored record
- Index file entries of form (K, P_r) or (K, P)
  - P_r: pointer to the record containing the key
  - P: pointer to the block containing the record for that key
- Collision
  - Hash key value for inserted record hashes to address already containing a different record
  - Resolution: Open addressing, Chaining, Multiple hashing



Pearson

13

Physical Database Design in Relational Databases

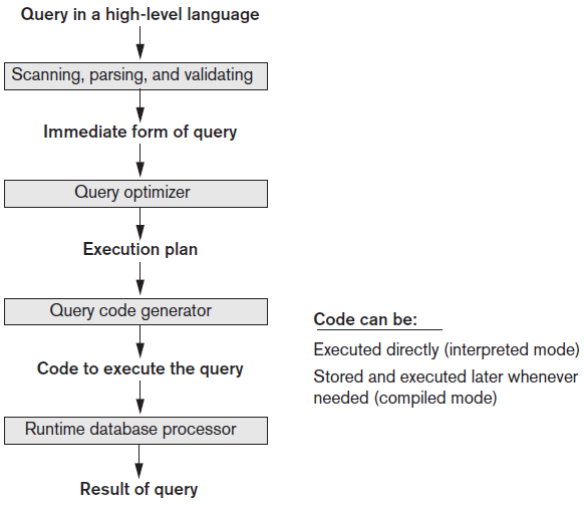
- Physical design goals
  - Create appropriate structure for data in storage
  - Guarantee good performance
- Analyzing the database queries and transactions
  - Information about each retrieval query
  - Information about each update transaction
- Analyzing the expected frequency of invocation of queries and transactions
  - Expected frequency of using each attribute as a selection or join attribute
  - 80-20 rule: 80 percent of processing accounted for by only 20 percent of queries and transactions
- Analyzing the time constraints of queries and transactions
  - Selection attributes associated with time constraints are candidates for primary access structures
- Analyzing the expected frequency of update operations
  - Minimize number of access paths for a frequently-updated file
- Analyzing the uniqueness constraints on attributes
  - Access paths should be specified on all **candidate key** attributes that are either the primary key of a file or unique attributes

Pearson

14

Query Processing and Optimization

- DBMS techniques to process a query
  - Scanner identifies query tokens
  - Parser checks the query syntax
  - Validation checks all attribute and relation names
  - Query tree (or query graph) created
  - Execution strategy or query plan devised
- Query optimization
  - Planning a good execution strategy



Pearson

15

Query Optimization

- Conducted by a query optimizer in a DBMS
- Goal: select best available strategy for executing query
  - Based on information available
- Most RDBMSs use a tree as the internal representation of a query
- Query Trees and Heuristics for Query Optimization
  - Step 1: scanner and parser generate initial query representation
  - Step 2: representation is optimized according to heuristic rules (e.g., Apply SELECT and PROJECT before JOIN to reduce size of files to be joined)
  - Step 3: query execution plan is developed
    - Execute groups of operations based on access paths available and files involved

Pearson

16

Query Trees and Query Graph for Query Optimization

Q2: **SELECT** P.Pnumber, P.Dnum, E.Lname, E.Address, E.Bdate
FROM PROJECT P, DEPARTMENT D, EMPLOYEE E
WHERE P.Dnum=D.Dnumber **AND** D.Mgr_ssn=E.Ssn **AND** P.Plocation= 'Stafford';

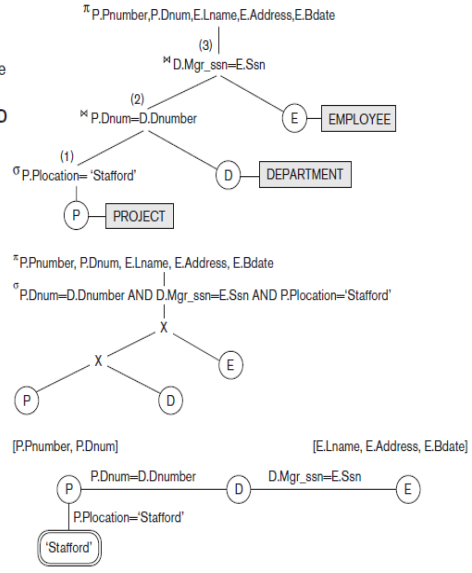


Figure 19.1 Two query trees for the query Q2.
(a) Query tree corresponding to the relational algebra expression for Q2.
(b) Initial (canonical) query tree for SQL query Q2.
(c) Query graph for Q2.

Pearson

17

Transaction

- Transaction
  - Describes local unit of database processing
  - Begin and end transaction statements specify transaction boundaries
  - Read-only transaction
  - Read-write transaction
- Transaction processing systems
  - Systems with large databases and hundreds of concurrent users
  - Require high availability and fast response time

Pearson

18

Read and Write Operations

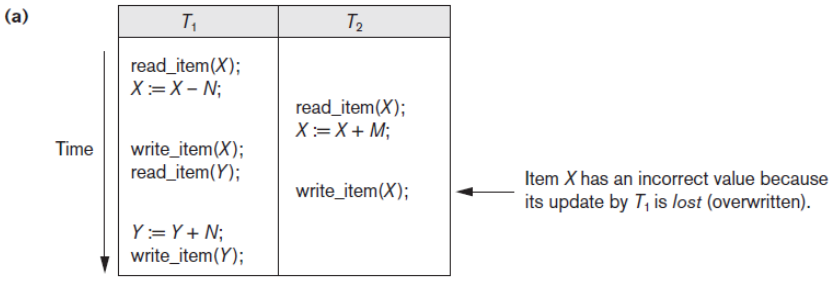
- read_item(*X*)
  - Reads a database item named *X* into a program variable named *X*
  - Process includes finding the address of the disk block, and copying to and from a memory buffer
  - Read set of a transaction: set of all items read
- write_item(*X*)
  - Writes the value of program variable *X* into the database item named *X*
  - Process includes finding the address of the disk block, copying to and from a memory buffer, and storing the updated disk block back to disk
  - Write set of a transaction: set of all items written

Pearson

19

Why Concurrency Control is Needed

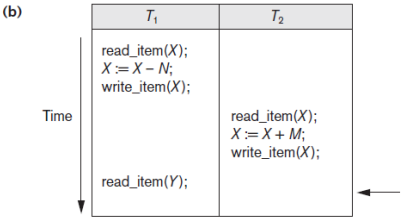
- Transactions submitted by various users may execute concurrently
  - Access and update the same database items
  - Some form of concurrency control is needed
- Various update problems
  - Occur when two transactions that access the same database items have operations interleaved
  - Results in incorrect value of some database items
- The update lost problem (a)



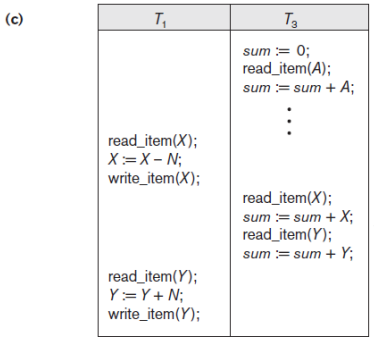
Pearson

20

Why Concurrency Control is Needed (cont'd)



Transaction T_1 fails and must change the value of X back to its old value; meanwhile T_2 has read the *temporary* incorrect value of X.



T_3 reads X after N is subtracted and reads Y before N is added; a wrong summary is the result (off by N).

- The Dirty Read (Temporary Update) Problem (b)
- The Incorrect Summary Problem (c)
- The Unrepeatable Read Problem
  - Transaction T reads the same item twice, Value is changed by another transaction T′ between the two reads, T receives different values for the two reads of the same item

21

Why Recovery is Needed

- Committed transaction
  - Effect recorded permanently in the database
- Aborted transaction
  - Does not affect the database
- Types of transaction failures
  - Computer failure (system crash)
  - Transaction or system error
  - Local errors or exception conditions detected by the transaction
  - Concurrency control enforcement
  - Disk failure
  - Physical problems or catastrophes
- System must keep sufficient information to recover quickly from the failure
  - Disk failure or other catastrophes have long recovery times

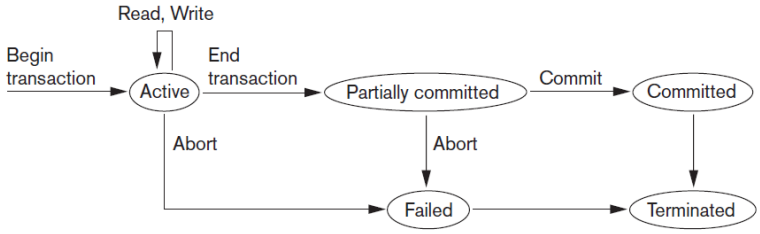
Pearson

22

Transaction and System Concepts

- System must keep track of when each transaction starts, terminates, commits, and/or aborts
  - BEGIN_TRANSACTION
  - READ or WRITE
  - END_TRANSACTION
  - COMMIT_TRANSACTION
  - ROLLBACK (or ABORT)

Figure 20.4 State transition diagram illustrating the states for transaction execution



Pearson

23

The System Log

- System log keeps track of transaction operations
- Sequential, append-only file
- Not affected by failure (except disk or catastrophic failure)
- Log buffer
  - Main memory buffer
  - When full, appended to end of log file on disk
- Log file is backed up periodically
- Undo and redo operations based on log are possible
- Commit Point of a Transaction
  - Occurs when all operations that access the database have completed successfully and effect of operations recorded in the log
  - Transaction writes a commit record into the log - If system failure occurs, can search for transactions with recorded start_transaction but no commit record
  - Force-writing the log buffer to disk - Writing log buffer to disk before transaction reaches commit point

Pearson

24

Desirable Properties of Transactions

- **ACID** properties
  - **A**tomicity: transaction performed in its entirety or not at all
  - **C**onsistency (preservation): Takes database from one consistent state to another
  - **I**solation: Not interfered with by other transactions
  - **D**urability or permanency: Changes must persist in the database

Pearson

25

Characterizing Schedules Based on Recoverability

- Schedule or history
  - Order of execution of operations from all transactions
  - Operations from different transactions can be interleaved in the schedule
- Total ordering of operations in a schedule
  - For any two operations in the schedule, one must occur before the other
- Two conflicting operations in a schedule: belong to different transactions, access the same item *X,* at least one of the operations is a write_item(*X*)
- Two operations conflict if changing their order results in a different outcome
- Two types of conflict: Read-write conflict, Write-write conflict
- Recoverable schedules - Recovery is possible
- Non-recoverable schedules should not be permitted by the DBMS
- No committed transaction ever needs to be rolled back
- Cascading rollback may occur in some recoverable schedules
  - Uncommitted transaction may need to be rolled back
- Cascadeless schedule - Avoids cascading rollback
- Strict schedule - Transactions can neither read nor write an item *X* until the last transaction that wrote *X* has committed or aborted

Pearson

26

Characterizing Schedules Based on Serializability

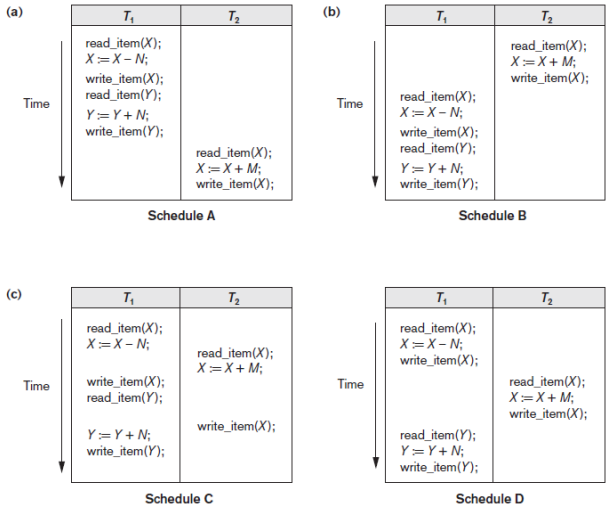
- Serial schedules
  - Places simultaneous transactions in series
    - Transaction T_1 before T_2, or vice versa
- Problem with serial schedules
  - Limit concurrency by prohibiting interleaving of operations
  - Unacceptable in practice
  - Solution: determine which schedules are equivalent to a serial schedule and allow those to occur
- Serializable schedule of *n* transactions
  - Equivalent to some serial schedule of same *n* transactions
- Result equivalent schedules
  - Produce the same final state of the database
    - May be accidental
  - Cannot be used alone to define equivalence of schedules
- Conflict equivalence
  - Relative order of any two conflicting operations is the same in both schedules
- Serializable schedules
  - Schedule *S* is serializable if it is conflict equivalent to some serial schedule *S'*.

Pearson

27

Figure 20.5 Examples of Serial and Nonserial Schedules Involving Transactions T_1 and T_2

(a) Serial schedule A: T_1 followed by T_2 (b) Serial schedule B: T_2 followed by T_1
(c) Two nonserial schedules C and D with interleaving of operations



Pearson

28

Characterizing Schedules Based on Serializability

- Testing for serializability of a schedule
  1. For each transaction T_i participating in schedule S, create a node labeled T_i in the precedence graph.
  2. For each case in S where T_j executes a read_item(X) after T_i executes a write_item(X), create an edge $(T_i \rightarrow T_j)$ in the precedence graph.
  3. For each case in S where T_j executes a write_item(X) after T_i executes a read_item(X), create an edge $(T_i \rightarrow T_j)$ in the precedence graph.
  4. For each case in S where T_j executes a write_item(X) after T_i executes a write_item(X), create an edge $(T_i \rightarrow T_j)$ in the precedence graph.
  5. The schedule S is serializable if and only if the precedence graph has no cycles.

$r_1(x), r_2(y), w_1(x), w_2(y), r_2(y), c_2, w_1(z), c_1, r_3(z), r_3(x), c_3$ ✓

$r_1(x), r_2(y), r_3(x), w_1(x), w_2(y), r_2(y), c_2, w_1(z), r_3(z), c_3, c_1$ ✗

Pearson

29

Concurrency Control

- Concurrency control protocols
  - Set of rules to guarantee serializability
- Two-phase locking protocols
  - Lock data items to prevent concurrent access
- Other protocols
  - Timestamp
    - Unique identifier for each transaction
  - Multiversion currency control protocols
    - Use multiple versions of a data item
  - Validation or certification of a transaction

Pearson

30

Two-Phase Locking Techniques for Concurrency Control

- Lock: variable associated with a data item describing status for operations that can be applied, one lock for each item in the database
- Shared/exclusive or read/write locks
  - Read operations on the same item are not conflicting
  - Must have exclusive lock to write
  - Three locking operations: read_lock(X), write_lock(X), unlock(X)
- Two-phase locking protocol
  - All locking operations precede the first unlock operation in the transaction
  - Phases
    - Expanding (growing) phase
      - New locks can be acquired but none can be released
    - Shrinking phase
      - Existing locks can be released but none can be acquired
- If every transaction in a schedule follows the two-phase locking protocol, schedule guaranteed to be serializable
- Two-phase locking may limit the amount of concurrency that can occur in a schedule
- Some serializable schedules will be prohibited by two-phase locking protocol

Pearson

31

Figure 21.3 Transactions That Do Not Obey Two-Phase Locking (a) Two Transactions *T*1 and *T*2 (b) Results of Possible Serial Schedules of *T*1 and *T*2 (c) A Nonserializable Schedule *S* That Uses Locks

(a)

T_1	T_2

read_lock(*Y*);
read_item(*Y*);
unlock(*Y*);
write_lock(*X*);
read_item(*X*);
X := *X* + *Y*;
write_item(*X*);
unlock(*X*);
	read_lock(*X*);
	read_item(*X*);
	unlock(*X*);
	write_lock(*Y*);
	read_item(*Y*);
	Y := *X* + *Y*;
	write_item(*Y*);
	unlock(*Y*);

(b) Initial values: *X*=20, *Y*=30

Result serial schedule T_1 followed by T_2: *X*=50, *Y*=80

Result of serial schedule T_2 followed by T_1: *X*=70, *Y*=50

(c)

Time

T_1	T_2

read_lock(*Y*);
read_item(*Y*);
unlock(*Y*);	

	read_lock(*X*);
	read_item(*X*);
	unlock(*X*);
	write_lock(*Y*);
	read_item(*Y*);
	Y := *X* + *Y*;
	write_item(*Y*);
	unlock(*Y*);

write_lock(*X*);
read_item(*X*);
X := *X* + *Y*;
write_item(*X*);
unlock(*X*);	

Pearson

32

Variations of Two-Phase Locking

- Basic 2PL
  - Technique described on previous slides
- Conservative (static) 2PL
  - Requires a transaction to lock all the items it accesses before the transaction begins
    - Predeclare read-set and write-set
  - Deadlock-free protocol
- Strict 2PL
  - Transaction does not release exclusive locks until after it commits or aborts
- Rigorous 2PL
  - Transaction does not release any locks until after it commits or aborts
- Concurrency control subsystem responsible for generating read_lock and write_lock requests
- Locking generally considered to have high overhead

Pearson

33

Recovery Concepts

- Recovery process restores database to most recent consistent state before time of failure
- Information kept in system log
- Deferred update techniques
  - Do not physically update the database until after transaction commits
  - Undo is not needed; redo may be needed
- Immediate update techniques
  - Database may be updated by some operations of a transaction before it reaches commit point
  - Operations also recorded in log
- Undo and redo operations required to be idempotent
  - Executing operations multiple times equivalent to executing just once
- Caching (buffering) of disk blocks
- Before-image: old value of data item
- After-image: new value of data item

Pearson

34

Recovery Concepts (Cont'd)

- Steal/no-steal and force/no-force
  - Specify rules that govern when a page from the database cache can be written to disk
- No-steal approach
  - Cache buffer page updated by a transaction cannot be written to disk before the transaction commits
- Steal approach
  - Recovery protocol allows writing an updated buffer before the transaction commits
- Force/no-force approaches
  - All pages updated by a transaction are immediately written to disk before the transaction commits
  - Otherwise, no-force
- Typical database systems employ a steal/no-force strategy
  - Avoids need for very large buffer space
  - Reduces disk I/O operations for heavily updated pages

Recovery Concepts (Cont'd)

- Write-ahead logging protocol for recovery algorithm requiring both UNDO and REDO
  - Before-image of an item cannot be overwritten by its after-image until all UNDO-type log entries have been force-written to disk
  - Commit operation of a transaction cannot be completed until all REDO-type and UNDO-type log records for that transaction have been force-written to disk
- Transaction rollback – transaction failure after update but before commit
  - Necessary to roll back the transaction
  - Old data values restored using undo-type log entries
- Cascading rollback
  - If transaction T is rolled back, any transaction S that has read value of item written by T must also be rolled back
  - Almost all recovery mechanisms designed to avoid this

NO-UNDO/REDO Recovery Based on Deferred Update

- Deferred update concept
  - Postpone updates to the database on disk until the transaction completes successfully and reaches its commit point
  - Redo-type log entries are needed
  - Undo-type log entries not necessary
  - Can only be used for short transactions and transactions that change few items
    - Buffer space an issue with longer transactions
- Deferred update protocol
  - Transaction cannot change the database on disk until it reaches its commit point
    - All buffers changed by the transaction must be pinned until the transaction commits (no-steal policy)
  - Transaction does not reach its commit point until all its REDO-type log entries are recorded in log and log buffer is force-written to disk

Pearson

37

Recovery Techniques Based on Immediate Update

- Database can be updated immediately
  - No need to wait for transaction to reach commit point
  - Not a requirement that every update be immediate
- UNDO-type log entries must be stored
- Recovery algorithms
  - UNDO/NO-REDO (steal/force strategy)
  - UNDO/REDO (steal/no-force strategy)

Pearson

38

Copyright

Pearson

39