Final Quiz Review

EECS 339

Lecture 18

Topics Covered

- Disk storage and file organization
- Indexing (tree and hash)
- Query optimization
- Transactions
- Concurrency Control

Data Organization

- Many file organizations exist, each appropriate in some situation.
- If selection queries are frequent, sorting the file or building an *index* is important.
  - Hash-based indexes only good for equality search.
  - Sorted files and tree-based indexes best for range search; also good for equality search. (Files rarely kept sorted in practice; B+ tree index is better.)
- Index is a collection of data entries plus a way to quickly find entries with given key values.

Data Organization (cont'd)

- Data entries can be actual data records, <key, rid> pairs, or <key, rid-list> pairs.
  - Choice orthogonal to *indexing technique* used to locate data entries with a given key value.
- Can have several indexes on a given file of data records, each with a different search key.
- Indexes can be classified as clustered vs. unclustered, primary vs. secondary, and dense vs. sparse. Differences have important consequences for utility/performance.

Fitting a Workload

- Understanding the nature of the *workload* for the application, and the performance goals, is essential to developing a good design.
  - What are the important queries and updates? What attributes/relations are involved?
- Indexes must be chosen to speed up important queries (and perhaps some updates!).
  - Index maintenance overhead on updates to key fields.
  - Choose indexes that can help many queries, if possible.
  - Build indexes to support index-only strategies.
  - Clustering is an important decision; only one index on a given relation can be clustered!
  - Order of fields in composite index key can be important.

(Physical) Data Management

- Disks provide cheap, non-volatile storage.
  - Random access, but cost depends on location of page on disk; important to arrange data sequentially to minimize *seek* and *rotation* delays.
- Buffer manager brings pages into RAM.
  - Page stays in RAM until released by requestor.
  - Written to disk when frame chosen for replacement (which is sometime after requestor releases the page).
  - Choice of frame to replace based on *replacement policy.*
  - Tries to *pre-fetch* several pages at a time.

DBMS Storage Features

- DBMS vs. OS File Support
  - DBMS needs features not found in many OS' s, e.g., forcing a page to disk, controlling the order of page writes to disk, files spanning disks, ability to control pre-fetching and page replacement policy based on predictable access patterns, etc.
- Variable length record format with field offset directory offers support for direct access to i' th field and null values.
- Slotted page format supports variable length records and allows records to move on page.

Data Organization

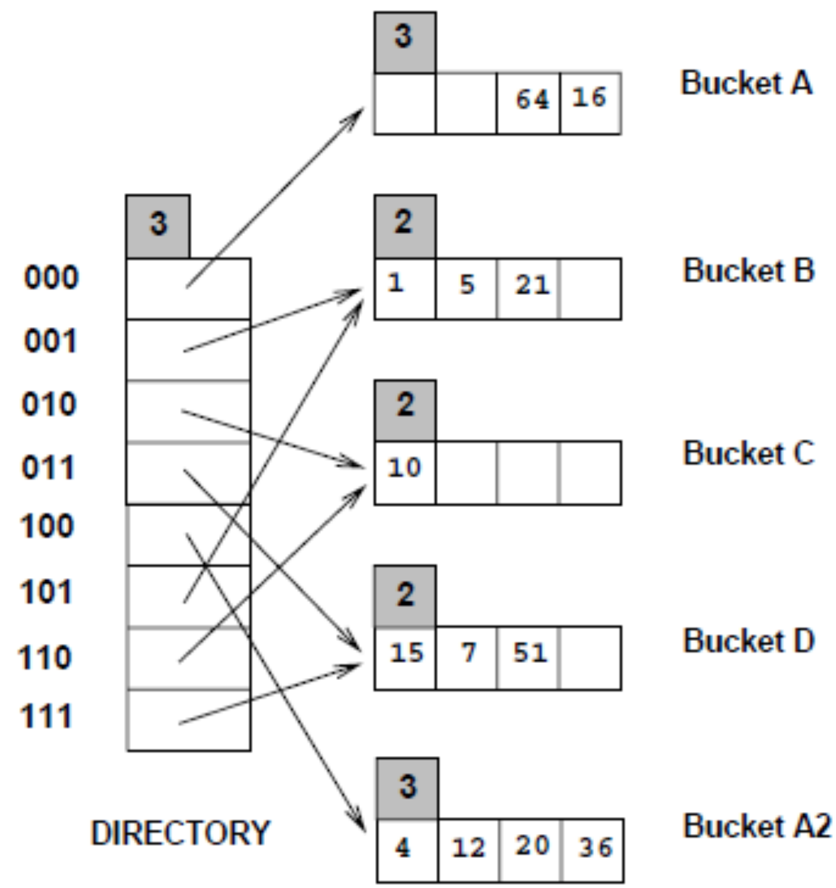
- File layer keeps track of pages in a file, and supports abstraction of a collection of records.
  - Pages with free space identified using linked list or directory structure (similar to how pages in file are kept track of).
- Indexes support efficient retrieval of records based on the values in some fields.
- Catalog relations store information about relations, indexes and views. (*Information that is common to all records in a given collection.*)

Hash Indexing

- Hash-based indexes: best for equality searches, cannot support range searches.
- Static Hashing can lead to long overflow chains.
- Extendible Hashing avoids overflow pages by splitting a full bucket when a new data entry is to be added to it. (*Duplicates may require overflow pages.*)
  - Directory to keep track of buckets, doubles periodically.
  - Can get large with skewed data; additional I/O if this does not fit in main memory.

Study Break: Extendible Hashing

- Consider an instance of the extensible hashing index:



- Show the index after inserting entry with hash value 68 (binary: 1000100)

Study Break Solution

- Split bucket A2, doubling the directory size
- A2 is redistributed over buckets 0100 and 1100
  - 0100 contains 4, 20, 36, 68
  - 1100 contains 12

Tree Indexing

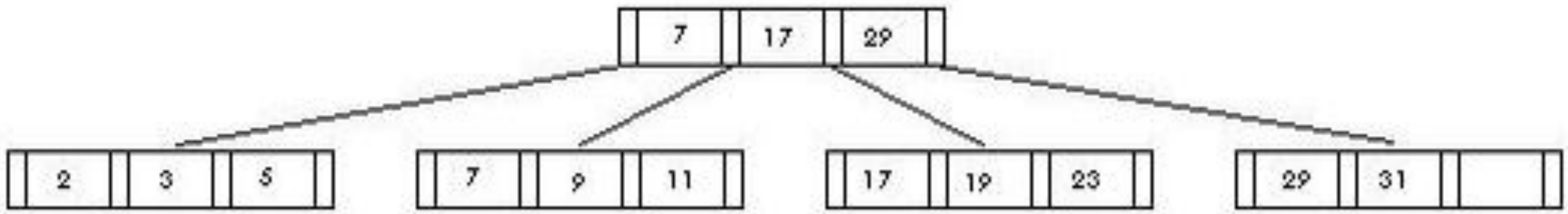
- Tree-structured indexes are ideal for range-searches, also good for equality searches.
- ISAM is a static structure.
  - Only leaf pages modified; overflow pages needed.
  - Overflow chains can degrade performance unless size of data set and data distribution stay constant.
- B+ tree is a dynamic structure.
  - Inserts/deletes leave tree height-balanced; $\log_F N$ cost.
  - High fanout (**F**) means depth rarely more than 3 or 4.
  - Almost always better than maintaining a sorted file.

B+ Trees

- Most widely used index in database management systems because of its versatility. One of the most optimized components of a DBMS.
  - Typically, 67% occupancy on average / page
  - Usually preferable to ISAM, modulo *locking* considerations; adjusts to growth gracefully.
  - If data entries are data records, splits can change rids!
- Key compression increases fanout, reduces height.

Study Break: B+ Trees

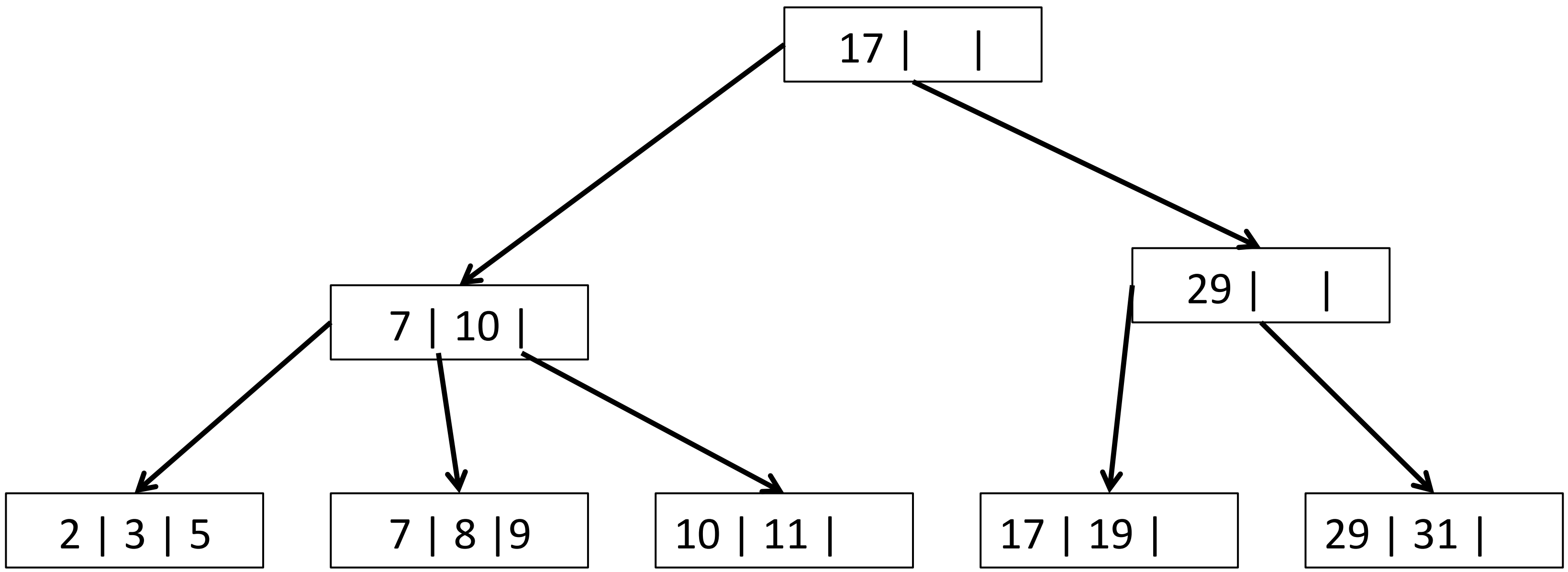
- Consider the following B+ tree:



Show the tree after the following sequence of modifications:

insert 10
insert 8
delete 23

Study Break Solution



Query Evaluation

- There are several alternative evaluation algorithms for each relational operator.
- A query is evaluated by converting it to a tree of operators and evaluating the operators in the tree.
- Must understand query optimization in order to fully understand the performance impact of a given database design (relations, indexes) on a workload (set of queries).

Query Evaluation

- A virtue of relational DBMSs: *queries are composed of a few basic operators*; the implementation of these operators can be carefully tuned (and it is important to do this!).
  - Especially joins
- Many alternative implementation techniques for each operator; no universally superior technique for most operators.
- Must consider available alternatives for each operation in a query and choose best one based on system statistics, etc. This is part of the broader task of optimizing a query composed of several ops.

Query Optimization

- Query optimization is an important task in a relational DBMS.
- Must understand optimization in order to understand the performance impact of a given database design (relations, indexes) on a workload (set of queries).
- Two parts to optimizing a query:
  - Consider a set of alternative plans.
    - Must prune search space; typically, left-deep plans only.
  - Must estimate cost of each plan that is considered.
    - Must estimate size of result and cost for each plan node.
    - *Key issues*: Statistics, indexes, operator implementations.

Query Optimization (cont'd)

- Single-relation queries:
  - All access paths considered, cheapest is chosen.
  - *Issues*: Selections that *match* index, whether index key has all needed fields and/or provides tuples in a desired order.
- Multiple-relation queries:
  - All single-relation plans are first enumerated.
    - Selections/projections considered as early as possible.
  - Next, for each 1-relation plan, all ways of joining another relation (as inner) are considered.
  - Next, for each 2-relation plan that is `retained', all ways of joining another relation (as inner) are considered, etc.
  - At each level, for each subset of relations, only best plan for each interesting order of tuples is `retained'.

Transactions

- Concurrency control and recovery are among the most important functions provided by a DBMS.
- Users need not worry about concurrency.
  - System automatically inserts lock/unlock requests and schedules actions of different Xacts in such a way as to ensure that the resulting execution is equivalent to executing the Xacts one after the other in some order.
- Write-ahead logging (WAL) is used to undo the actions of aborted transactions and to restore the system to a consistent state after a crash.
  - *Consistent state*: Only the effects of commited Xacts seen.

Study Break: Serializable Schedules

For the following schedules, state whether they are conflict serializable. If so, give a schedule. Otherwise, identify the conflicting ops.

Transaction 1	Transaction 2
1. RA	
2. WA	
	3. RB
	4. RC
	5. WA
6. RB	
7. RC	
8. **commit**	
	9. WC
	10. **commit**

Transaction 1	Transaction 2
1. RA	
	2. RB
3. WA	
	4. WB
5. RB	
	6. RA
7. WB	
8. **commit**	
	9. WA
	10. **commit**

Study Break Solution

- Yes, it serializes to T1 T2 if we swap (6,7,8) with (3, 4, 5)
- No, this schedule is not serializable. Example conflicts: (1, 9) (3, 6) (3,9)

Two Phase Locking

- “Pessimistic” concurrency control
- Transactions acquire a **S**hared lock for reads, e**X**clusive lock for writes on an object
- Phases:
  - Growth (lock acquisition)
  - Shrinking (releasing locks)
- Use intention locks to address phantom problem

Study Break: Phantom Problem

- For the following xactions, state whether they are possibly affected by the phantom problem and why.

Transaction 1	Transaction 2
`SELECT * FROM employees;`	
	`INSERT into employees VALUES ('JR', 'Art Dept');`
`SELECT count(*) FROM employees;`	

Transaction 1	Transaction 2
`SELECT room_number FROM classrooms, buildings WHERE building.b_name = 'Tech' AND building.b_id = classrooms.b_id;`	
	`INSERT into classrooms VALUES (1, 'Tech', 414);`
`SELECT max(room_number) FROM classrooms, buildings WHERE building.b_name = 'Tech' AND building.b_id = classrooms.b_id;`	

Study Break Solution

- Yes, the count may increase with the addition of a new employee.
- Yes, the first query accesses all room numbers in Tech, and the second part of Transaction 1 may access a larger set of tuples than the one returned in the earlier query.

Optimistic Concurrency Control

- Optimistic concurrency control detects conflicts when or shortly after they happen
- Maintain read and write lists
- 3 phases: read, validation write
- Best CC scheme depends on conflict rate
  - OCC best for low conflict – 2PL for mostly overlapping xactions

Study Break: OCC

- For each of the following, indicate whether T2 will commit or abort using OCC with serial validation:

1. T1: Read Set {A,B}, Write Set {A},
   T2: Read set: {A,B}, Write Set {B}
2. T1: Read Set {A,C}, Write Set {A}
   T2: Read Set {B,C}, Write Set {A}
3. T1: Read Set {A,C}, Write Set {A}
   T2: Read Set {B,C}, Write Set {A,B}

Study Break Solution

1. ABORT – read and write sets intersect
2. COMMIT – By rule 2 of OCC, the WS of T1 does not intersect RS of T2, and T1 completes its write phase before T2 starts its write phase, due to the use of serial validation.

- COMMIT – for the same reason as above; adding B to the write set of T2 doesn't change it.

Additional Remarks

- Quiz
  - is open book, open notes (no laptops or phones)
  - Takes place on 3/16 at 9 am in Tech M152
  - Is not cumulative
- Check out Canvas for old exams and more example questions
- Good luck and thank you!