Announcements (Thu., Feb 27)

- Private project threads created on piazza
  - Please check you are on your thread
  - Primary and secondary project mentors to be assigned soon
  - They will give you feedback on MS1, check updates, and help you as needed
  - Feel free to discuss projects in all TA office hours

- Project updates to be posted **every Monday**
  - Starts Monday 03/02: each member should say what you are supposed to do for the rest of the semester and also for MS2
  - Make sure that your primary TA says "sounds good" in the response to each update, otherwise do as they suggest
  - Other team members will also check the updates and respond to the threads as needed if there are confusions or clarifications
  - Try to resolve conflicts/concerns within group whenever possible, otherwise reach out to your TAs and me early

Announcements - contd (Thu., Feb 27)

- Homework #4 published due next Wednesday 03/04
  - One submission per project group
  - See updates on piazza about submitting the link to your website

- Let me know if you want to meet me about midterm or anything else
  - Final exam will be similar in nature (problem-solving based), but the length/difficulty/question types may vary
  - Comprehensive with more focus on topics after midterm
  - How to prepare? Think about what we discuss in class and ask me tough questions!

- Heads up: (almost) weekly quiz or lab **every Thursday**
  - For practicing problems for the final
  - In groups, but individual submissions
  - Quizzes are shorter and discussed in class, Labs are longer with extra time after class (extra credit for submitting within the class)

Today’s lecture

- Index
  - Dense vs. Sparse
  - Clustered vs. unclustered
  - Primary vs. secondary
  - Tree-based vs. Hash-index

What are indexes for?

- Given a value, locate the record(s) with this value
  \[
  \text{SELECT * FROM } R \text{ WHERE } A = \text{value}; \\
  \text{SELECT * FROM } R, S \text{ WHERE } R.A = S.B;
  \]
- Find data by other search criteria, e.g.
  - Range search
  \[
  \text{SELECT * FROM } R \text{ WHERE } A > \text{value};
  \]
  - Keyword search

Dense and sparse indexes

- Dense: one index entry for each search key value
  - One entry may “point” to multiple records (e.g., two users named Jessica)
- Sparse: one index entry for each block
  - Records must be clustered according to the search key

---

Focus of today's lecture

Focus of this lecture
**Dense versus sparse indexes**

- Index size
- Requirement on records
- Lookup
- Update

**Primary and secondary indexes**

- Primary index
  - Created for the primary key of a table
  - Records are usually clustered by the primary key
  - Can be sparse
- Secondary index
  - Usually dense
- SQL
  - PRIMARY KEY declaration automatically creates a primary index, UNIQUE key automatically creates a secondary index
  - Additional secondary index can be created on non-key attribute(s):

```
CREATE INDEX UserPopIndex ON User(pop);
```

**What if the index is too big as well?**

- ISAM (Index Sequential Access Method), more or less

Example: look up 197
Updates with ISAM

- Overflow chains and empty data blocks degrade performance
- Worst case: most records go into one long chain, so lookups require scanning all data!

B*-tree

- A hierarchy of nodes with intervals
- Balanced (more or less): good performance guarantee
- Disk-based: one node per block; large fan-out

### B*-tree balancing properties

- Height constraint: all leaves at the same lowest level
- Fan-out constraint: all nodes at least half full (except root)

#### Sample B*-tree nodes

- Non-leaf
  - to keys $100 \leq k < 120$
  - to keys $120 \leq k < 150$
  - to keys $150 \leq k < 180$
  - to keys $180 \leq k$

- Leaf
  - to keys with these $k$ values;
  - or, store records directly in leaves

#### Lookups

- SELECT * FROM R WHERE $k = 179$;
- SELECT * FROM R WHERE $k = 32$;

#### Range query

- SELECT * FROM R WHERE $k > 32$ AND $k < 179$;

- Check yourself

- Max fan-out: 4

- Max fan-out: 4

- Max fan-out: 4

- Max fan-out: 4
Insertion
• Insert a record with search key value 32

Look up where the inserted key should go...
And insert it right there

Another insertion example
• Insert a record with search key value 152

Oops, node is already full!
What are our options here?

Node splitting
Max fan-out: 4

Oops, that node becomes full!
Need to add a pointer to the newly created node

More node splitting
Max fan-out: 4

Need to add to parent node a pointer to the newly created node
• In the worst case, node splitting can “propagate” all the way up to the root of the tree (not illustrated here)
• Splitting the root introduces a new root of fan-out 2 and causes the tree to grow “up” by one level

Deletion
• Delete a record with search key value 130

Look up the key to be deleted...
And delete it
Oops, node is too empty!

Stealing from a sibling
Max fan-out: 4

Remember to fix the key in the least common ancestor of the affected nodes
Another deletion example
• Delete a record with search key value 179

Coalescing
• Deletion can “propagate” all the way up to the root of the tree (not illustrated here)
• When the root becomes empty, the tree “shrinks” by one level

Performance analysis
• How many I/O’s are required for each operation?
  • $h$, the height of the tree (more or less)
  • Plus one or two to manipulate actual records
  • Plus $O(h)$ for reorganization (rare if $f$ is large)
  • Minus one if we cache the root in memory
• How big is $h$?

B*-tree in practice
• Complex reorganization for deletion often is not implemented (e.g., Oracle)
  • Leave nodes less than half full and periodically reorganize
• Most commercial DBMS use B*-tree instead of hashing-based indexes because B*-tree handles range queries
  • A key difference between hash and tree indexes!

The Halloween Problem
• Story from the early days of System R...
  UPDATE Payroll
  SET salary = salary * 1.1
  WHERE salary >= 100000;
  • There is a B*-tree index on Payroll(salary)
  • The update never stopped (why?)
• Solutions?

B*-tree versus ISAM
• ISAM is more static; B*-tree is more dynamic
• ISAM can be more compact (at least initially)
  • Fewer levels and I/O’s than B*-tree
• Overtime, ISAM may not be balanced
  • Cannot provide guaranteed performance as B*-tree does

B⁺-tree versus B-tree

- B-tree: why not store records (or record pointers) in non-leaf nodes?
  - These records can be accessed with fewer I/O's
- Problems?

Beyond ISAM, B-, and B⁺-trees

- Other tree-based indexes: R-trees and variants, GiST, etc.
  - How about binary tree?
- Hashing-based indexes: extensible hashing, linear hashing, etc.
- Text indexes: inverted-list index, suffix arrays, etc.
- Other tricks: bitmap index, bit-sliced index, etc.

Clustered vs. Unclustered Index

- If order of data records in a file is the same as, or 'close to', order of data entries in an index, then clustered, otherwise unclustered
- How does it affect # of page accesses? (in class)

Hash vs. Tree Index

- Hash indexes can only handle equality queries
  - SELECT * FROM R WHERE age = 5 (requires hash index on (age))
  - SELECT * FROM B, S WHERE R.A = S.A (requires hash index on R.A or S.A)
  - SELECT * FROM R WHERE age = 5 and name = 'Bart' (requires hash index on (age, name))
- Cannot handle range queries
  - SELECT * FROM R WHERE age >= 5
    - need to use tree indexes (more common)
  - Tree index on (age), or (age, name) works, but not (name, age) - why?
- But are more amenable to parallel processing
  - late hash-based join
- Performance depends on how good the hash function is (whether the hash function distributes data uniformly and whether data has skew)
- Details of hash-based dynamic index (extendible hashing, linear hashing) not covered in this class

Trade-offs for Indexes

- Should we use as many indexes as possible?
Trade-offs for Indexes

• Should we use as many indexes as possible?

Index-Only Plans

• A number of queries can be answered without retrieving any tuples from one or more of the relations involved if a suitable index is available

```sql
SELECT E.dno, COUNT(*)
FROM Emp E
GROUP BY E.dno
```

```sql
SELECT E.dno, MIN(E.sal)
FROM Emp E
GROUP BY E.dno
```

```sql
SELECT AVG(E.sal)
FROM Emp E
WHERE E.age=25 AND E.sal BETWEEN 3000 AND 5000
```

• For index-only strategies, clustering is not important