Outline

- Physical organization of data on disk
- Indexing (and SQL)
- Indexing sequential files
  - Primary, secondary
  - Clustering, non-clustering
  - Dense, sparse.
  - Multi-level
- Other indexing structures
  - Linear, B-tree, hashing

Disk Surface

- Sectors and Blocks
  - Sector: smallest physical unit of data transferred between disk and main memory.
  - Block: logical unit of data, consists of several consecutive sectors.
  - Databases deal with blocks.

Data Layout

- Each block contains:
  - Block header (meta data)
  - Records (corresponding to tuples)
- Each record contains:
  - Record header
  - Fields (attributes)

Indexing

- Get a particular record (or several records) given a value for some field.
  - Read all blocks with records.
  - Use an index to locate block(s) with record(s).
Indexes in SQL

CREATE INDEX index_name
ON table(attr1, attr2,...);

CREATE INDEX bar_idx
ON Sells(bar);

- Indexes can be included in the table declaration.

MySQL Indexes

CREATE TABLE Sells(
    bar varchar(20),
    beer varchar(20),
    price real,
    INDEX (bar),
    INDEX beerIdx (beer)
);

DROP INDEX beerIdx ON Sells;
SHOW INDEX FROM Sells;

Using Indexes: Selection

SELECT beer FROM Sells WHERE bar = 'Level';
SELECT price FROM Sells WHERE beer = 'Bud';
SELECT price FROM Sells WHERE beer = 'Bud' AND bar = 'Rainbo';
SELECT MAX(price) FROM Sells WHERE bar <> 'Cans';

Using Indexes: Joins

SELECT beer FROM Sells AS S, Frequents AS F
WHERE S.bar = F.bar AND drinker = 'Sally';

SELECT beer FROM Sells AS S, Frequents AS F
WHERE S.bar = F.bar AND price < 10;

Indexing Sequential Files

- Records stored in a sorted order
  - often by primary key
- Primary index
  - on a sorting field
  - determines record location

Clustering

- Clustering index packs records with the same values of indexed attributes in as few blocks as possible
  - Not necessarily sorted
Dense Indexes
- Record pointer for each key value
- Number of index entries = number of records
  - Is it worth it?
  - Example
  - Block vs. Record pointers

Sparse Indexes
- Index only the first record in a block.
- Example.
- Always better than dense indexes?
- Records must be sorted.

Multiple Level Indexes
- What if indexes occupy many blocks?
- First-level index can be sparse or dense
- Higher level indexes must be sparse.
- Example.

Record Modifications
- Deletion
- Insertion
- Updates
- Reorganization policy
  - immediate
  - postpone (overflow blocks)
- Examples

Secondary Indexes
- Records not sorted (in order of indexing field).
- First level index must be dense.
- Higher levels indexes can be (must be?) sparse.

Applications
- Multiple keys, only one can be primary
  - Only one primary index!
- Non-key fields
- Clustering
  - Store records of two different types on the same block
Buckets

- Buckets of record pointers
- Index points to buckets
- Another level of indirection
- Example
- Is it worth it?
  - Efficient joins.

B-trees

- Balanced trees
- Each node is at least half full.
- Find any record with fixed number of I/O
  - In most cases 1 or 2
  - Many variants: B+ trees

B-tree Structure

- Every node is stored in a block
- Each node has space for
  - $n$ values
  - $n+1$ pointer
- Three types of nodes:
  - Root
  - Interior
  - Leaf nodes

Leaf Nodes

- All leaf nodes are chain-linked together
  - One pointer per node.
  - Number of pairs of value and record pointer:
    - Max: $n$
    - Min: $\lfloor (n+1)/2 \rfloor$
  - Example

Interior Node

- Values and pointers to nodes of the next (lower) level:
  - Max: $n$ values, $n+1$ pointers
  - Min: $\lceil (n-1)/2 \rceil$ values, $\lceil (n+1)/2 \rceil$ pointers

Root Node

- Pointer(s) to next level
- Min: 2
- Max: $n+1$
- Example
- Extreme case: the root is also a leaf
Lookup

- Given a key $x$,
- Start at the root node.
- Follow the pointer before the smallest value that is strictly greater than $x$, or last pointer if there's no such value.
- Repeat until you reach a leaf node.
- If $x$ exists in the leaf node follow pointer to record, otherwise there's no such record.

Range Queries

```
SELECT beer
FROM Sells
WHERE beer < 'Corona';
```

```
SELECT drinker
FROM Drinkers
WHERE drinker > 'Amy'
AND drinker < 'Rick';
```

Insertion

- Possible cases:
  1. No structural change
  2. Leaf node overflow
  3. Interior node overflow
  4. Root overflow
- A single insertion can trigger cases 2, 3, and 4!

Deletion

- Possible cases:
  1. No structural changes
     - But we may update a value in a higher level
  2. Leaf node underflow
  3. Interior node underflow
  4. Root underflow
     - Often deletion reorganization is ignored.

Performance

- Reorganization is rare
- Lookup, insert, delete take $k$ I/Os, where $k$ is the depth of the tree.
- $k$ is at most 4
  - For less than 4 billion records
  - The root is often kept in memory
    - And possibly (part) of second level
  - So, operations take 1-3 I/Os.

Hashing

- Main memory hashing
- Secondary storage hashing
- Static hashing
- Extensible hashing
  - Double the number of buckets
- Linear hashing
  - Increase the number of buckets by 1