3a. Database Storage

Overview

- The contents of a database must be stored on the computer.
- Want ways of storing the data to facilitate data retrieval (minimize time spent reading data).
- Overlap with operating systems:
  - Similar principles/techniques.
  - DBMSs often implement custom memory/file management procedures (overriding OS default).

Types of storage

- Traditionally, computer systems have two main types of storage.
  - “Main memory” – a computer’s fast but **volatile** storage (contents disappear if power is lost).
  - “Disk” / magnetic-disk storage – found in hard drives; non-volatile but still subject to occasional failures and data loss.
- However, recent trend is toward “flash memory” / solid-state drives – non-volatile long-term storage like disk but allowing faster access.
  - Not quite as fast or as expensive as high-end computer memory.

Primary vs. secondary storage

- “Primary storage” = memory used for performing calculations/working.
  - E.g. main memory.
- “Secondary storage” = storage used for storing data when not in use.
  - E.g. magnetic or solid-state drives.
- “Tertiary storage” = archiving data that is no longer expected to change or even be read frequently.
  - E.g. drives, tapes, CDs, etc.

Units of storage

- Block = fixed-length unit of storage; smallest unit of data that can be read or allocated.
  - Usually ~4-8 kb.
- Page = fixed-length unit of data used in memory management.
- Both blocks and pages are fixed-length chunks / units of storage. Sometimes synonymous, sometimes not.

Blocks in relational databases

- One record (row/tuple) in a relational DB is typically smaller than a block.
  - Setting aside large objects (images, files, etc.).
- Fixed-length records/fields can simplify storage/storage operations.
  - Number of records per block is known in advance.
  - New records can fill the place of deleted records if order is not important.
  - Keep track of empty slots with a "free list".

Handling variable-length records

- Records with variable-length fields can be implemented by storing two pieces of metadata per record: **offset** (location) and **length** (size).
- Variable-length records can be stored within a block (or page) with a **slotted-page structure**:
  - Block storage is "eaten" at two different ends and meets in the middle.
  - Top: header with offset/length metadata on all records in that block.
  - Bottom: the records stored contiguously.
- Operating systems also make use of this technique.

Buffers

- "Buffer" has many definitions.
    - Original meaning: Something that sits in front of something and is ideally interacted with instead of that other thing.
    - Typical CS meaning: Temporary memory storage for data that is being (1) processed, or (2) transferred elsewhere.
- In databases context: a buffer is anywhere storing (temporary) copies of data outside of the database (for reading or updating purposes).
    - Can be in primary or secondary storage.

Updating database data

- Buffers contain data temporarily loaded from database.
- If space is limited, some data (pages) not being (recently) used may be overwritten.
  - If this space contains updated database data, the updated data should be written to the database (**dirty page**).
- "Flushing buffers" means writing all data contained in buffers to long-term storage.
  - Clear out all data so that all of temporary storage is available to be used again.

Indexing and hashing

Summary so far

- Tables are stored in secondary storage ("disk") as a series of records.
- Records are organized into chunks called blocks.
- Database queries and operations require loading (parts of) the table into working memory so that it can be returned/processed.
- Reading data from secondary storage can be very slow/costly.
- Tables are often much larger than working memory.
- What if you want to just read a single row?
  - Need to find it with a minimum amount of unnecessary disk reads.

Indexes

- **Index**: a data structure to aid in the quick retrieval (random access) of specific records in a table.
- Types of indexes:
  - Clustering/clustered vs. nonclustered (or primary vs. secondary indexes).
  - For clustered indexes: Dense vs. sparse.
  - Single-level vs. multilevel.

Clustered vs. nonclustered indexes

- **Clustering** or **primary index**: the attribute or set of attributes that determine the order that records in a relation are physically stored.
  - Usually primary key, but not necessarily.
- **Nonclustered** or **secondary index**: an additional index defined on a table that differs / comes apart from the way the table is intrinsically stored.
- Can think of these as sorted vs. non-sorted attributes, where each table can only be stored sorted along one certain dimension (primary index).

Dense vs. sparse indexes

- **Dense index**: Every unique value of the index attribute(s) has a direct pointer to it.
- **Sparse index**: Not every value of the index attribute(s) has a direct pointer.
  - Sorting order + pointers to close values can be used to eventually find the search keys not covered by the sparse index.
  - Only useful for clustered indexes. All secondary indexes have to be dense to be usable.
- Common compromise: a sparse index with one entry per block.

Loading a record from a table

- In order to find a specific search key in a table:
  - Load the index.
  - While the index is in memory, peruse it for the listing you need.
  - Index tells you exact place in secondary storage to load (dense).
    - Or a range (sparse).
  - Load that block(s) from disk to get your data.
- If the index is small, can keep it around in memory and use it to retrieve other parts of the table as needed.
- But what if the index itself doesn't fit in memory?

Multilevel indexes

- If an index is too large to keep comfortably in memory at all times, can make another (sparse) index of index records.
  - Tells address of blocks that are storing ranges of index records.
  - You can instantly load the index block you need from perusing the information in the index-index.
- Create additional layers as needed.
  - Forms a tree structure.
- Keep the root of the tree in memory at all times.

B+ trees

- One of many types of **balanced tree** data structures that exist.
  - Balanced trees guarantee upper bounds on the depth of the tree.
  - Implement specialized insert/update/delete algorithms that maintain the balance of a tree in logarithmic time.
- B+ trees differ from other balanced trees because they are "short" and "fat."
  - Each node of the tree is block-sized and packed with as many pointers to other blocks as will fit in a block.
- The leaves of B+ trees are the records of the database (different from interior index nodes).
  - B+ tree also handles how to store records in blocks on disk to keep the table in sorted order as items are inserted and deleted.

Hashing

- Hashing: alternative storage and retrieval system that enables (near) instantaneous insert/update/delete.
- Use a hash function:
  - Input: Search key values.
  - Output: An address in memory.
- Use hash function to deterministically generate a storage location.
  - Store there and retrieve there later.

Desired characteristics of a hash function

- An ideal hash function for database storage has the following properties:
  - Deterministic behavior (same search key sent in twice MUST yield same output).
  - Yields an answer for all possible search keys.
  - Distributes search keys across the allowed storage space in as uniform a manner as possible.
  - Quick to calculate.

Handling overflow

- Sometimes, more data items are assigned to a storage location (bucket) by the hash function than can actually be stored there – "bucket overflow".
- Two strategies for handling overflow:
  - **Open hashing**: Use a linked list (e.g.) to provide extra storage buckets.
    - All keys will definitely be stored at that hash location (closed addressing), but you need a secondary data structure other than the hash table (open hashing).
  - **Closed hashing**: Use a method (e.g. linear probing) to determine which hash location to push the overflow items to.
    - Keys not necessarily stored in the location given by the hash function (open addressing).
    - Storage location may be dependent on the contents of the hash table at a particular time.
- Linked list type is commonly used for databases because it makes deletion easier.

Clustered index vs. hashing

Clustered index	Hashing
Data stored in sorted order. Reading data in ranges or sorted order is simple.	Data storage is in random order. Reading data in ranges or sorted order is non-trivial.
O(log n) operations.	O(1) operations in typical case.
A single implementation is appropriate for any type of data.	Performance is dependent on choice of a good hash function. Changes in the distribution of search keys may change hash function's performance.