CompSci 516
Data Intensive Computing Systems

Lecture 7
Storage and Index

Instructor: Sudeepa Roy
Announcements

• HW1 deadline this week:
  – Due on 09/21 (Thurs), 11:55 pm, no late days

• Project proposal deadline:
  – Preliminary idea and team members due by tonight
    09/18 (Mon), 11:55 pm by email to the instructor
  – Proposal due on sakai by 09/25 (Mon), 11:55 pm

• Everyone should be in a group now
  – otherwise let the instructor know asap
Reading Material

• [RG]
  – Storage: Chapters 8.1, 8.2, 8.4, 9.4-9.7
  – Index: 8.3, 8.5
  – Tree-based index: Chapter 10.1-10.7
  – Hash-based index: Chapter 11

Additional reading
• [GUW]
  – Chapters 8.3, 14.1-14.4

Acknowledgement:
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.
Storage
(contd. from Lecture 6)
Recap

• Typical DBMS hierarchy
• Disk and main memory/buffer pool
• Unit = page or block
  – page replacement strategies
  – dirty bit
  – pin
Today

• How are pages stored in a file?
• How are records stored in a page?
  – Fixed length records
  – Variable length records
• How are fields stored in a record?
  – Fixed length fields/records
  – Variable length fields/records
Files of Records

• Page or block is OK when doing I/O, but higher levels of DBMS operate on records, and files of records

• **FILE**: A collection of pages, each containing a collection of records

• Must support:
  – insert/delete/modify record
  – read a particular record (specified using record id)
  – scan all records (possibly with some conditions on the records to be retrieved)
File Organization

• **File organization**: Method of arranging a file of records on external storage
  – One file can have multiple pages
  – **Record id** (rid) is sufficient to physically locate the page containing the record on disk
  – **Indexes** are data structures that allow us to find the record ids of records with given values in index search key fields

• **NOTE**: Several uses of “keys” in a database
  – Primary/foreign/candidate/super keys
  – Index search keys
Alternative File Organizations

Many alternatives exist, each ideal for some situations, and not so good in others:

• **Heap (random order) files:** Suitable when typical access is a file scan retrieving all records

• **Sorted Files:** Best if records must be retrieved in some order, or only a “range” of records is needed.

• **Indexes:** Data structures to organize records via trees or hashing
  
  – Like sorted files, they speed up searches for a subset of records, based on values in certain (“search key”) fields
  
  – Updates are much faster than in sorted files
Unordered (Heap) Files

• Simplest file structure contains records in no particular order
• As file grows and shrinks, disk pages are allocated and de-allocated
• To support record level operations, we must:
  – keep track of the *pages* in a file
  – keep track of *free space* on pages
  – keep track of the *records* on a page
• There are many alternatives for keeping track of this
Heap File Implemented as a List

- The header page id and Heap file name must be stored someplace
- Each page contains 2 `pointers’ plus data
- Problem?
  - to insert a new record, we may need to scan several pages on the free list to find one with sufficient space
• The entry for a page can include the number of free bytes on the page.
• The directory is a collection of pages
  – linked list implementation of directory is just one alternative
  – Much smaller than linked list of all heap file pages!
How do we arrange a collection of records on a page?

• Each page contains several slots
  – one for each record

• Record is identified by \(<\text{page-id, slot-number}>\)

• Fixed-Length Records
• Variable-Length Records

• For both, there are options for
  – Record formats (how to organize the fields within a record)
  – Page formats (how to organize the records within a page)
Page Formats: Fixed Length Records

- Record id = <page id, slot #>
- Packed: moving records for free space management changes rid; may not be acceptable
- Unpacked: use a bitmap – scan the bit array to find an empty slot
- Each page also may contain additional info like the id of the next page (not shown)
Page Formats: Variable Length Records

• Need to find a page with the right amount of space
  – Too small – cannot insert
  – Too large – waste of space

• if a record is deleted, need to move the records so that all free space is contiguous
  – need ability to move records within a page

• Can maintain a directory of slots (next slide)
  – Slot contains <record-offset, record-length>
  – deletion = set record-offset to -1

• Record-id rid = <page, slot-in-directory> remains unchanged
Page Formats: Variable Length Records

- Can move records on page without changing rid
  - so, attractive for fixed-length records too
- Store (record-offset, record-length) in each slot
- rid-s unaffected by rearranging records in a page
Record Formats: Fixed Length

- Each field has a fixed length
  - for all records
  - the number of fields is also fixed
  - fields can be stored consecutively
- Information about field types same for all records in a file
  - stored in system catalogs
- Finding i-th field does not require scan of record
  - given the address of the record, address of a field can be obtained easily

Base address (B)  Address = B+L1+L2
Record Formats: Variable Length

- Cannot use fixed-length slots for records
- Two alternative formats (# fields is fixed):
  - Second offers direct access to i-th field, efficient storage of nulls (special don’t know value); small directory overhead
  - Modification may be costly (may grow the field and not fit in the page)
Indexes
Indexes

• An index on a file speeds up selections on the search key fields for the index
  – Any subset of the fields of a relation can be the search key for an index on the relation.
  – “Search key” is not the same as “key”
    key = minimal set of fields that uniquely identify a tuple

• An index contains a collection of data entries, and supports efficient retrieval of all data entries $k^*$ with a given key value $k$
Remember Terminology

• **Index search key (key):** k
  – Used to search a record

• **Data entry:** k*
  – Pointed to by k
  –Contains record id(s) or record itself

• **Records or data**
  – Actual tuples
  – Pointed to by record ids
Alternatives for Data Entry $k^*$ in Index $k$

- In a data entry $k^*$ we can store:
  1. (Alternative 1) The actual data record with key value $k$, or
  2. (Alternative 2) $<k, \text{rid}>$
     - $\text{rid} = \text{record of data record with search key value } k$, or
  3. (Alternative 3) $<k, \text{rid-list}>$
     - list of record ids of data records with search key $k$

- Choice of alternative for data entries is orthogonal to the indexing technique used to locate data entries with a given key value $k$
### Alternatives for Data Entries: Alternative 1

- **In a data entry $k^*$ we can store:**
  1. The actual data record with key value $k$
  2. $<k, \text{rid}>$
    - rid = record of data record with search key value $k$
  3. $<k, \text{rid-list}>$
    - list of record ids of data records with search key $k$

- **Index structure is a file organization for data records**
  - instead of a Heap file or sorted file
- **How many different indexes can use Alternative 1?**
- **At most one index can use Alternative 1**
  - Otherwise, data records are duplicated, leading to redundant storage and potential inconsistency
- **If data records are very large, #pages with data entries is high**
  - Implies size of auxiliary information in the index is also large

---

Advantages/ Disadvantages?
Alternatives for Data Entries: Alternative 2, 3

- In a data entry $k^*$ we can store:
  1. The actual data record with key value $k$
  2. $\langle k, \text{rid} \rangle$
     - rid = record of data record with search key value $k$
  3. $\langle k, \text{rid-list} \rangle$
     - list of record ids of data records with search key $k$

- Data entries typically much smaller than data records
  - So, better than Alternative 1 with large data records
  - Especially if search keys are small.

- Alternative 3 more compact than Alternative 2
  - but leads to variable-size data entries even if search keys have fixed length.
Index Classification

- Primary vs. secondary
- Clustered vs. unclustered
- Tree-based vs. Hash-based
Primary vs. Secondary Index

• If search key contains primary key, then called primary index, otherwise secondary
  – Unique index: Search key contains a candidate key

• Duplicate data entries:
  – if they have the same value of search key field \( k \)
  – Primary/unique index never has a duplicate
  – Other secondary index can have duplicates
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
  - Alternative 1 implies clustered
  - Alternative 2, 3 are typically unclustered
    - unless sorted according to the search key
  - Sometimes, clustered also implies Alternative 1
    - since sorted files are rare
  - A file can be clustered on at most one search key
  - Cost of retrieving data records (range queries) through index varies greatly based on whether index is clustered or not
Clustered vs. Unclustered Index

- Suppose that Alternative (2) is used for data entries, and that the data records are stored in a Heap file.
- To build clustered index, first sort the Heap file:
  - with some free space on each page for future inserts
  - Overflow pages may be needed for inserts
  - Thus, data records are ‘close to’, but not identical to, sorted
Methods for indexing

• Tree-based
• Hash-based

• (in detail later)
System Catalogs

- For each index:
  - structure (e.g., B+ tree) and search key fields
- For each relation:
  - name, file name, file structure (e.g., Heap file)
  - attribute name and type, for each attribute
  - index name, for each index
  - integrity constraints
- For each view:
  - view name and definition
- Plus statistics, authorization, buffer pool size, etc.
- (described in [RG] 12.1)

Catalogs are themselves stored as relations!