Announcements

• HW1 Deadlines!
  – Today: parser and Q1-Q3 (last late day!)
  – Q4: next Tuesday 09/24
  – Q5 (RA questions posted on Sakai): next to next Tuesday 10/01
    • Check Piazza for submission instructions

• 2 late days with penalty apply for individual deadlines
  – It is important to start HWs from day-1!
Today’s topics

• Finish Normalization & BCNF
  – We will do 4NF later

• New topic: Database Internals

Acknowledgement:
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke
Recap: Functional Dependencies (FDs)

AB → C
ABD → C
AB → A (trivial)

But not

AB → D
A → D
A → C
Normal Forms

R is in 4NF
⇒ R is in BCNF
⇒ R is in 3NF
⇒ R is in 2NF (a historical one)
⇒ R is in 1NF (every field has atomic values)

Only BCNF and 4NF are covered in the class
Boyce-Codd Normal Form (BCNF)

• Relation $R$ with FDs $F$ is in BCNF

• if, for all $X \rightarrow A$ in $F$
  
  – $A \in X$ (called a trivial FD), or
  
  – $X$ contains a key for $R$
    
    • i.e. $X$ is a superkey

No dependencies other than from superkeys can exist!
BCNF decomposition algorithm

1. Find a BCNF violation
   - That is, a non-trivial FD $X \rightarrow Y$ in $R$ where $X$ is not a super key of $R$
2. Decompose $R$ into $R_1$ and $R_2$, where
   - $R_1$ has attributes $X \cup Y$
   - $R_2$ has attributes $X \cup Z$, where $Z$ contains all attributes of $R$ that are in neither $X$ nor $Y$
3. Repeat until all relations are in BCNF

- Also gives a lossless decomposition!
  - Check yourself
BCNF decomposition example - 1

- **CSJDPQV, key C, F = {JP → C, SD → P, J → S}**
  - To deal with SD → P, decompose into SDP, CSJDQV.
  - To deal with J → S, decompose CSJDQV into JS and CJDQV

- **Note:**
  - several dependencies may cause violation of BCNF
  - The order in which we pick them may lead to very different sets of relations
  - there may be multiple correct decompositions (can pick J → S first)

- **Is JP → C a violation of BCNF?**
BCNF decomposition example - 2

UserJoinsGroup (uid, uname, twitterid, gid, fromDate)

User (uid, uname, twitterid)

Member (uid, gid, fromDate)

BCNF violation: uid → uname, twitterid

uid → uname, twitterid
twitterid → uid
uid, gid → fromDate

BCNF
It is not enough to only look at given FDs! You need to consider the closure!

UserJointGroup (uid, uname, twitterid, gid, fromDate)

BCNF violation: twitterid → uid

UserId (twitterid, uid)

UserJoinGroup’ (twitterid, uname, gid, fromDate)

BCNF violation: twitterid → uname

UserName (twitterid, uname)

Member (twitterid, gid, fromDate)

BCNF

apply Armstrong’s axioms and rules!

uid → uname, twitterid
twitterid → uid
uid, gid → fromDate
Recap

• Functional dependencies: a generalization of the key concept

• Non-key functional dependencies: a source of redundancy

• BCNF decomposition: a method for removing redundancies
  – And gives lossless join decomposition

• BCNF = no redundancy due to FDs

But - the relation may still have redundancies! 4-NF (later)
Where are we now?

We learnt How to write queries and how to design a good schema without (some) redundancies

✓ Relational Model and Query Languages
  ✓ SQL, RA, RC
  ✓ Postgres (DBMS)
  ✓ XML (overview)
    ▪ HW1
✓ Database Normalization

Next
• DBMS Internals (i.e., how a database system works!)
  – Storage
  – Indexing
  – Query Evaluation
  – Operator Algorithms
  – External sort
  – Query Optimization
Storage
A typical DBMS has a layered architecture.

The figure does not show the concurrency control and recovery components — to be done in “transactions.”

This is one of several possible architectures — each system has its own variations.

These layers must consider concurrency control and recovery.
Data on External Storage

• Data must persist on disk across program executions in a DBMS
  – Data is huge
  – Must persist across executions
  – But has to be fetched into main memory when DBMS processes the data

• The unit of information for reading data from disk, or writing data to disk, is a page

• Disks: Can retrieve random page at fixed cost
  – But reading several consecutive pages is much cheaper than reading them in random order
Disk Space Management

• Lowest layer of DBMS software manages space on disk

• Higher levels call upon this layer to:
  – allocate/de-allocate a page
  – read/write a page

• Size of a page = size of a disk block
  = data unit

• Request for a sequence of pages often satisfied by allocating contiguous blocks on disk

• Space on disk managed by Disk-space Manager
  – Higher levels don’t need to know how this is done, or how free space is managed
Buffer Management

Suppose
• 1 million pages in db, but only space for 1000 in memory
• A query needs to scan the entire file
• DBMS has to
  – bring pages into main memory
  – decide which existing pages to replace to make room for a new page
  – called Replacement Policy
• Managed by the Buffer manager
  – Files and access methods ask the buffer manager to access a page mentioning the “record id” (soon)
  – Buffer manager loads the page if not already there
Buffer Management

Buffer pool = main memory is partitioned into frames either contains a page from disk or is a free frame

Page Requests from Higher Levels

MAIN MEMORY

DISK

BUFFER POOL

disk page

free frame

choice of frame dictated by replacement policy

• Data must be in RAM for DBMS to operate on it
• Table of <frame#, pageid> pairs is maintained
When a Page is Requested ...

For every frame, store

• a dirty bit:
  – whether the page in the frame has been modified since it has been brought to memory
  – initially 0 or off

• a pin-count:
  – the number of times the page in the frame has been requested but not released (and no. of current users)
  – initially 0
  – when a page is requested, the count in incremented
  – when the requestor releases the page, count is decremented
  – buffer manager only reads a page into a frame when its pin-count is 0
  – if no frame with pin-count 0, buffer manager has to wait (or a transaction is aborted -- later)
When a Page is Requested ...

- Check if the page is already in the buffer pool
- if yes, increment the pin-count of that frame
- If no,
  - Choose a frame for replacement using the replacement policy
  - If the chosen frame is dirty (has been modified), write it to disk
  - Read requested page into chosen frame
- Pin (increase pin-count of) the page and return its address to the requestor

If requests can be predicted (e.g., sequential scans), pages can be pre-fetched several pages at a time
- Concurrency Control & recovery may entail additional I/O when a frame is chosen for replacement
Buffer Replacement Policy

• Frame is chosen for replacement by a replacement policy

• Least-recently-used (LRU)
  – add frames with pin-count 0 to the end of a queue
  – choose from head

• Clock (an efficient implementation of LRU)
• First In First Out (FIFO)
• Most-Recently-Used (MRU) etc.
Buffer Replacement Policy

- Policy can have big impact on # of I/O’s
- Depends on the access pattern
- **Sequential flooding:** Nasty situation caused by LRU + repeated sequential scans
  - What happens with 10 frames and 9 pages?
  - What happens with 10 frames and 11 pages? *(check yourself!)*
  - # buffer frames < # pages in file means each page request in each scan causes an I/O
  - MRU much better in this situation (but not in all situations, of course)
DBMS vs. OS File System

• Operating Systems do disk space and buffer management too:
  • Why not let OS manage these tasks?

• DBMS can predict the page reference patterns much more accurately
  – can optimize
  – adjust replacement policy
  – pre-fetch pages – already in buffer + contiguous allocation
  – pin a page in buffer pool, force a page to disk (important for implementing Transactions concurrency control & recovery)

• Differences in OS support: portability issues

• Some limitations, e.g., files can’t span disks
Next..

• 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
  – Next lecture!
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
Heap File Using a Page Directory

- 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!