**414-f17 review**

1. Normal forms
   a. keys
      i. superkey: implies all of R
      ii. candidate key: minimal superkey
      iii. primary key: cand. key chosen by designer
   b. $F^+$ is closure of $F$: all dependencies that can be inferred from $F$
   c. BCNF
      i. in BCNF if for each F in F+: though equiv to use just F:
         1. A→B is trivial, or
         2. A is superkey
      ii. decomposition
         1. Choose F=A→B that breaks as above
            a. $\$R_1 = A \cup (R - B)$$
            b. $\$R_2 = A \cup (R - B)$$
         2. general test for lossless
            a. Given decomp of $\$R$ into $\$R_{1}$ and $\$R_{2}$:
               i. $\$R_{1} \cap R_{2} \rightarrow$ rightarrow R1$$
               or
               ii. $\$R_{1} \cap R_{2} \rightarrow$ rightarrow R2$$
      iii. BCNF lossless because:
         a. we decomp by some f=a→b in $F^+$ s.t. f not in BCNF
         b. a duplicated on both sides and a→b, so:
            i. $\$S_{R_{1}} \cap R_{2} \rightarrow$ rightarrow R1$$
      iv. So lossless, but not dependency-preserving
      v. example decomp. Assume $\$S=(A,B,C,D), A$ rightarrow B$$
         a. $\$S_{R_{1}} = A,B$$, carries $\$S|A$ rightarrow B$$
         b. $\$S_{R_{2}} = A,C$$, no carried depend
   d. 3NF - preserves dependencies
      i. in if for each F in F+: (though we commonly use just F):
         1. $\$S|S$$ is trivial, or
         2. $\$S$$ is superkey, or
         3. each attribute in $\$S$ in a candidate key
      ii. decompose by
         1. one subrelation for each $\$S$ in canonical cover
         2. one for a candidate key, if none of the above contains one
      iii. Decompose: $\$S=(A,B,C,D), A \cap$ rightarrow C, AB \cap rightarrow D$$
         a. Candidate keys $\$S|(A,B)$$
         b. $\$S_{R_{1}} = (A,C), R_{2} = (A,B,D)$$$, done
      iv. good because
         1. preserves dependencies
         2. lossless, why?

2. FD goodness
   a. Armstrongs Axioms
      i. reflexivity: if $\$S|\alpha$$ is set of attribute, and $\$S|\beta$$ subseteq\alpha$$, then $\$S|\alpha$ rightarrow $\beta$$
      ii. augmentation: if $\$S|\alpha$ rightarrow $\beta$$, then $\$S|\alpha, \gamma$ rightarrow $\beta$$, $\gamma$$
      iii. transitivity: if $\$S|\alpha$ rightarrow $\beta$$, $\beta$$ rightarrow $\gamma$$, then $\$S|\alpha$ rightarrow $\gamma$$
   b. Other axioms:
      i. union: $\$S|\alpha$ rightarrow $\beta$$ and $\$S|\alpha$$ rightarrow $\gamma$$ implies $\$S|\alpha$ rightarrow $\beta, \gamma$$
      ii. decomposition: $\$S|\alpha, \gamma$ rightarrow $\beta$$, $\gamma$$ implies $\$S|\alpha$$ rightarrow $\beta$$, $\gamma$$
      iii. pseudotransitivity: if $\$S|\alpha$ rightarrow $\beta$$ and $\$S|\gamma$$ rightarrow $\beta$$, then $\$S|\alpha, \gamma$ rightarrow $\beta$$
   c. attribute closures:
      1. start with A, continually add to set via FDs
      2. extraneous attributes
         i. $\$S$ extra in $\$S$ if $\$S|\alpha$$ in $\$S$ and $\$S$$ implies $\$S|\beta$$, $\Gamma$$
         ii. $\$S$ extra in $\$S$ if $\$S|\alpha$$ in $\$S$$ and $\$S$$ implies $\$S|\beta$$, $\Gamma$$
      iii. canonical closure
         i. use union to combine RHS's
         ii. eliminate extra attributes
        iii. repeat
   d. Storage
      a. cost for single random read: $\$c = t_s \cdot t_{(rot+rt)} \cdot t_s$$
      b. RAID
         i. 0 - striped - both reads and writes faster
         ii. 1 - mirrored - reads faster, redundant, much space overhead
         iii. 5 - distributed parity blocks - not that fast, but space overhead low, durable
      c. Records
         i. fixed length - use free lists
         ii. variable length - slotted page (header up front w/ ptrs to where record is)
      d. Record to file mapping
         i. heap - no sort
         ii. sequential - sorted by key
            1. binary searches, still slow...
         iii. hash - hash to a block number, overflows
4 Indexes
   a Types
      i sparse - each key doesn't have to appear in index
      ii dense - does
   b Types
      i primary - relation sorted on search key of index, can be sparse
      ii secondary - not, can't
   c B+tree
      i short, large fanout minimize block reads
      ii balanced
      iii each node at least $$\frac{n}{2}$$ ptrs
      iv use: costs $\log_{\text{fanout}}$ for read of tree, plus read of data. Upper levels cached in memory
   v example: midterm 2 answers
5 Query execution
   a selections
     i seq scan
     ii bin search
     iii B+tree
        1 Example: Cost of reading tuples, assuming primary index, not key, equality
           a first leaf: $$h_i*(t_T+t_S)$$
           b tuples: $$t_S + b*t_T$$
   b external merge sorts
   c joins (minimal, fits in memory, one fits in memory)
     i nested loop
     ii block nested loop
        1 Example: 50 blocks
           a foreach group of 48 blocks in $$\ldots$$
              i foreach block in $$\ldots$$
              1 output tuples that match
           b Read $$\ldots$$ a total of $$\ldots$$ times
              i index nested loop
              iv merge join
                 1 $$b_r + b_s$$ blocks, seeks dependent on memory
                 v hash join
                    1 difficult can is neither fits in mem, write out corresponding partitions
              d blocking vs pipelined
   6 Query optimization
      a selectivity estimates
      b histograms
         i equi-width vs equi-depth
      c general scheme:
      d
6 Indexes
   a Types
      i sparse - each key doesn't have to appear in index
      ii dense - does
   b Types
      i primary - relation sorted on search key of index, can be sparse
      ii secondary - not, can't
   c B+tree
      i short, large fanout minimize block reads
      ii balanced
      iii each node at least $$\frac{n}{2}$$ ptrs
      iv use: costs $\log_{\text{fanout}}$ for read of tree, plus read of data. Upper levels cached in memory
   v example: midterm 2 answers
5 Query execution
   a selections
     i seq scan
     ii bin search
     iii B+tree
        1 Example: Cost of reading tuples, assuming primary index, not key, equality
           a first leaf: $$h_i*(t_T+t_S)$$
           b tuples: $$t_S + b*t_T$$
   b external merge sorts
   c joins (minimal, fits in memory, one fits in memory)
     i nested loop
     ii block nested loop
        1 Example: 50 blocks
           a foreach group of 48 blocks in $$\ldots$$
              i foreach block in $$\ldots$$
              1 output tuples that match
           b Read $$\ldots$$ a total of $$\ldots$$ times
              i index nested loop
              iv merge join
                 1 $$b_r + b_s$$ blocks, seeks dependent on memory
                 v hash join
                    1 difficult can is neither fits in mem, write out corresponding partitions
              d blocking vs pipelined
   6 Query optimization
      a selectivity estimates
      b histograms
         i equi-width vs equi-depth
      c general scheme:
b locks
   i approaches
      1 two-phase
         a conflict-serial, not recoverable
      2 strict two-phase
         a conflict-serial, cascade-less and recoverable
      3 rigorous
   ii granularity
      1 grab intention locks on higher level first
         a S, X, IX, IS, SIX
   iii problems
      1 deadlock
         a prevent - order
         b deal with - waits-for graph
            i wait-die (older waits, younger quits)
            ii wound-wait (older kills younger, younger waits)
   c Time-stamp based
      i trans issued TS when they enter system
      ii TS determine serial order
      iii reject reads, writes that would violate, abort
      iv restart aborted trans
      v Thomas’s Write Rule: ignore a write that would have caused an abort because should be overwritten

10 Security
   a SQL injection
      i blacklisting
      ii whitelisting
      iii prepared statements/bound-variables
   b Cryptdb
      i deterministic encryption \(\rightarrow\) equality checks
      ii order-preserving encryption \(\rightarrow\) range queries
      iii fully homomorphic encryption \(\rightarrow\) arbitrary operations on encrypted data
      iv encryption must match the intended use
   c Differential privacy
      i differential privacy aims to maximize the accuracy of queries from statistical databases while minimizing the chances of leaking information
      ii Example: protecting privacy vs allowing data mining

11 Machine learning
   a Classification
   b Clustering
   c Associations for correlation/causation