Mark Shinozaki
Cpts 451
HW 6 – Indexing

Question 1: Indexing

1. List the names, ages, and offices of professors of a user-specified sex (male or female) who have a user-specified research specialty (e.g., artificial intelligence). Assume that he university has a diverse set of faculty members, making it very uncommon for more than a few professors to have the same research specialty.

    - Attributes: \`sex\`, \`specialty\`
    - Index Type: Composite index on (\`sex\`, \`specialty\`)
    - Unclustered, updates are not frequent
    - Index structure: B+ tree (range queries)
        - **Query:** CREATE INDEX idx_prof_sex_specialty ON Prof(sex, specialty);

2. List all the department information for departments with professors in a user specified age range.
    - Attributes: \`age\`, \`dept_did\`
    - Index Type: Index on \`age\`
    - It is Unclustered
    - Index structure: B++ tree
        - **Query:** CREATE INDEX idx_prof_age ON Prof(age);

3. List the department id, department name, and chairperson name for departments with a user specified age range.
    - Attributes: \`num_majors\`, \`did\`, \`dname\`, \`chair_ssno\`
    - Index Type: Composite index on \`num_majors\`
    - It is Unclustered
    - Index structure: B++ tree
        - **Query:** CREATE INDEX idx_dept_num_majors ON Dept(num_majors);

4. List the lowest budget for a department in the university.
    - Attributes: \`budget\`
    - Index Type: Index on \`budget\`
    - It is Unclustered
    - Index structure: B++ tree
        - **Query:** CREATE INDEX idx_dept_budget ON Dept(budget);

5. List all the information about professors who are department chairpersons.
    - Attributes: \`chair_ssno\`
    - Index Type: Index on \`chair_ssno\`
    - It is Unclustered
    - Index structure: B++ tree
        - **Query:** CREATE INDEX idx_dept_chair_ssno ON Dept(chair_ssno);

Mark Shinozaki
Cpts 451
HW 6 – Indexing

Question 2: Storage and Indexing

<u>Given</u>

- Relation: \`Student(sid, sname, major, email)\`
- \`sid\` is the key
- \`sid\` values: uniformly distributed between \`100\` and \`204,900\`
- Attributes: \`char(40)\`
- 100,000 records
- Block size: 16KB + 8 bytes
- Record pointer size: 8 bytes

(a) Heap File Costs

a. File Scan:
  - # of records per block = $\frac{16 \times 1024}{40 \times 4} = 1024$
  - # of blocks = $\frac{100{,}000}{1024} = 98$
  - Cost = 98D

b. Equality Search (\` sid=\`25700\`)
  - Average case: Half the blocks need to be scanned
  - Cost = $\frac{98}{2} = 49D$

c. Range Search (\` sid <= \`25700\`):
  - Selectivity = $\frac{25700-100}{20480} = 0.125$
  - Number of records to scan $= 0.125 \times 100{,}000 = 12{,}500$
  - # of blocks to scan $= \frac{12{,}500}{1024} \approx 13$
  - Cost = 13D

(b) Clustered B++ Tree Index Costs

1. File Scan:
   - # of blocks with 67% occupancy = $1.5 \ \times 98 \approx 147$
   - Cost = 147D
2. Equality Search (\` sid=\`'25700'\`):
   - Height of B+ tree = 3
   - Cost =$Height + 1\ for\ data = 3 + 1 = 4D$
3. Range Search (\` sid <= '25700'\`):
   - Selectivity - .125
   - Cost for leaf pages = 13D
   - Total cost = 4D (for tree) + 13D (for data) = 17D

Mark Shinozaki
Cpts 451
HW 6 – Indexing

(c) Unclustered B+ Tree Index Costs
1. File Scan:
   - Same as heap file = 98D

2. Equality Search(`sid=`25700’`):
   - Height of B+ tree = 3
   - Cost = Height + 1 for data = 3 + 1 = 4D

3. Range Search (`sid <= ‘25700’`):
   - Selectivity = 0.125
   - # of records to scan = 12,500
   - Each look up needs 1 I/O for index + 1 I/O for data
   - Total cost = 12,500 I/Os (assuming random I/O)