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);

1. 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);

1. 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);

1. 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);

1. 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);

Question 2: Storage and Indexing

Given

* 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

1. Heap File Costs
   1. File Scan:

* # of records per block =
* # of blocks =
* Cost = 98D
  1. Equality Search (`sid=`25700`)
* Average case: Half the blocks need to be scanned
* Cost =
  1. Range Search (`sid <= `25700`):
* Selectivity =
* Number of records to scan
* # of blocks to scan
* Cost = 13D

1. Clustered B++ Tree Index Costs
2. File Scan:

* # of blocks with 67% occupancy =
* Cost = 147D

1. Equality Search (`sid=`’25700’`):

* Height of B+ tree = 3
* Cost =

1. Range Search (`sid <= ‘25700’`):

* Selectivity - .125
* Cost for leaf pages = 13D
* Total cost = 4D (for tree) + 13D (for data) = 17D

1. Unclustered B+ Tree Index Costs
2. File Scan:

* Same as heap file = 98D

1. Equality Search(`sid=`25700’`):

* Height of B+ tree = 3
* Cost = Height + 1 for data = 3 + 1 = 4D

1. 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)