Consider the following relational schema for a university database:

- **Prof** (\(pid, pname, office, age, gender, specialty, dept-did\))
- **Dept** (\(did, dname, budget, num-majors, chair-pid\))

Assume the 5 following queries (of equal importance) are the only ones in the workload, and specify indexes for their efficient processing:

1. List the names, ages, and offices of professors of a user-specified gender (male or female) that have a user-specified research specialty (e.g., databases). Assume that the university has a diverse set of faculty members, making it very uncommon for more than a few professors to have the same research specialty.
2. List all the department information for departments with professors in a user-specified age range.
3. List the department id, department name, and chairperson name for departments with a user-specified number of majors.
4. List the lowest budget for a department in the university.
5. List all the information about professors who are department chairpersons.
1. List the names, ages, and offices of professors of a user-specified sex (male or female) that have a user-specified research specialty (e.g., *database query processing*). Assume that the university has a diverse set of faculty members, making it very uncommon for more than a few professors to have the same research specialty.

- Use a **hash index** for **specialty** or `<specialty, gender>` on **Prof**. The index does not have to be clustered because the query is very selective (i.e., it is expected to retrieve very few records).
2. List all the department information for departments with professors in a user-specified age range.

• This is a join query since it specifies a selection condition on professors and requires information about their departments. We need a **B+-Tree index** for **age** on Prof for finding the professors. The B+-Tree should be clustered because there may be many professors in the specified range. Then for each professor we can find his/her dept using a **hash index** for **department id** on Dept. The index does not need to be clustered since each professor belongs to exactly one department.
3. List the department id, department name, and chairperson name for departments with a user-specified number of majors.

• This is also a join query since it specifies a selection condition on departments (# of majors) and requires information about their chairpersons.
  
  - Use a **clustered B⁺-Tree** on the number of majors so that all Depts with the specified number of majors are stored together. Alternatively, since the selection condition is equality we can use **hash file organization** of Dept on major (recall that hash indexes are non-clustered). First index is better.
  
  - After we retrieve qualifying departments, we find information about their chairperson, using a **hash index** for pid on Prof.
4. List the lowest budget for a department in the university.
   • The query can be answered by index-only scan using a B+-Tree for budget on Dept. The index does not have to be clustered since it covers the query.

5. List all the information about professors who are department chairpersons.
   • We have to scan the departments table and find information about their chairperson using the hash index for pid on Prof (that we created at query 3).
Exercise #2 – Physical DB Design with Cost Estimation

Keys are underlined and foreign keys are in *italics*.

**TABLES:**

<table>
<thead>
<tr>
<th>Table</th>
<th># records</th>
<th># pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPARTMENT (DEPT_ID, NAME, HEAD_ID)</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>PROFESSOR (PROF_ID, PNAME, SALARY, DEPT_ID)</td>
<td>1,000</td>
<td>100</td>
</tr>
<tr>
<td>STUDENT (STU_ID, NAME, DEPT_ID)</td>
<td>10,000</td>
<td>1,000</td>
</tr>
<tr>
<td>COURSE (COURSE_ID, NAME, DEPT_ID)</td>
<td>1,000</td>
<td>100</td>
</tr>
<tr>
<td>OFFERING (OFF_ID, COURSE_ID, YEAR, SEMESTER, PROF_ID)</td>
<td>10,000</td>
<td>1,000</td>
</tr>
<tr>
<td>ENROLL (STU_ID, OFF_ID, GRADE)</td>
<td>100,000</td>
<td>10,000</td>
</tr>
</tbody>
</table>

• There are 500 different professor names, 1,000 different student names and 100 different department names. On the average, each student is enrolled in 10 offerings and each offering has 10 students. Each department offers 10 courses. Assume that professor salaries are uniformly distributed in the range [10,000-110,000].

• Note: For clustered indexes assume B⁺-Trees with height 2. For hash indexes assume that there are no overflow buckets (i.e., the cost of finding an entry in the index is 1). For indexes on a non-candidate key, assume that you may have multiple entries with the same search key entry (e.g., if you have an index on PROFESSOR.salary there may be many entries with the same salary value pointing to different records).

We want to optimize a set of queries. Propose indexes and estimate their cost.
Q1: Display the names of all professors whose salary is in the range [20,000-25,000).

• **Answer:** Build a **clustered index** on PROFESSOR.salary. The query is expected to retrieve 50 (5%) professors, i.e., 5 pages.

  **Total Cost:** 2+5

• **Alternative answer:** We can use a **multi-attribute index** on PROFESSOR<salary, name> and do index-only scan. Since each index entry contains only 2 of the 4 PROFESSOR attributes, we can assume that the index leaf level contains 50 pages (half of the file). The cost is 2 pages to find the first entry satisfying the input condition, and then 2 more pages for reading the remaining entries (recall that the query will retrieve 5% of professors, so it has to read 5% of the index leaf nodes, i.e., 3 pages). **Total cost:** 2+2
Q2: Given a professor name, display the name of the department where the professor belongs.

- **Answer**: Build a hash index on PROFESSOR.pname. Use the index to find the two professors with the input name (cost 1 for locating the pname entry in the index + 2 for following the two pointers in the file). For each dept_id of these professors, use a hash index on DEPARTMENT.dept_id to retrieve the department record (and name), with cost 2. **Total cost**: 3 + 2*2 = 7

- **Alternative answer**: Instead of a hash index on DEPARTMENT.dept_id, use a hash file organization on DEPARTMENT, where the records are partitioned in 10 buckets (pages) based on their dept_id. In this case, we can retrieve each dept record with cost 1. **Total cost**: 3 + 2 = 5
Q3: Given a department name, display all the course names offered by the department.

• Answer: Use a hash index on DEPARTMENT.dept_name to retrieve the id of the department (cost 1+1). Use a clustering index on COURSE.dept_id to retrieve the names of all courses offered by the department (cost 2+1). Total cost 2+3=5.
Q4: Given a student name, display the name of the department where the student belongs.

• **Answer**: Use a clustering index on STUDENT.stu_name to retrieve the 10 students with the given name (cost 2+1). Find the dept_name where the student belongs using a hash index on DEPARTMENT.dept_id (cost 1+1 for each student). Total cost $3+2\times10=23$.

• **Alternative answer**: Similar to query 2, instead of a hash index on DEPARTMENT.dept_id, use a hash file organization on DEPARTMENT, where the records are partitioned in 10 buckets (pages) based on their dept_id. Total cost $3+10=13$. 

STUDENT (STU_ID, NAME, DEPT_ID): 10,000 recs, 1,000 names
DEPARTMENT (DEPT_ID, NAME, HEAD_ID): 100 recs
Q5: Given a student id, display the id of the courses taken by the student and his/her average grade.

• Answer (using a clustering index on ENROLL.stu_id): There are 100,000 records in ENROLL. Since there are 10,000 students, each student is enrolled in 10 offerings on the average. Use a clustering index on ENROLL.stu_id to retrieve the 10 offerings taken by the student with cost 2+1 (assuming that all the 10 offerings are in the same page). We can immediately compute the average of the student (based on these 10 offerings). For each of the offerings, we retrieve the corresponding course_id using a hash index on OFFERING.off_id (cost 1+1 for each offering). Total cost 3+10*2=23.

• Alternative answer (using a hash index on ENROLL.stu_id): If we do not use a clustering index on ENROLL.stu_id (because we will need another clustering index for query Q6), we use hash index, so that we retrieve the 10 offerings taken by the student with cost 11 (we need 1 page access for retrieving the index bucket and 10 accesses for following the pointers). The rest of the solution remains the same and the total cost is: 11+20=31.
Q6: Given a course id, display the average grade for each offering of the course. Each course has on the average 10 offerings. For example, if the input is id COMP231, the query will display the average grade group-by 10 offerings of COMP231. Each offering has on the average 10 students.

Answer (using a hash index on ENROLL.offering id): Use a clustering index on OFFERING.course id so that we can retrieve the 10 offerings of the course with cost 2+1 (assuming that all 10 offerings are in the same page). If we have already assumed a clustering index ENROLL.studentid (for Q5), we cannot have another clustering index on the same file. Instead, we use a hash index on ENROLL.offering id, to retrieve the records of 10 students that took the offering and compute the average grade for the offering (for each offering we need 1 page access in the index and 10 in the file, i.e., the cost is 11). We repeat the same process for each of the 10 offerings. Total cost 3+ 10(11)=113.

Alternative answer (using a clustering index on ENROLL.offering id): First we retrieve the 10 offerings of the course using a clustering index on OFFERING.course id (with cost 3 as explained in the first solution). Then, we use a clustering index on ENROLL.offering id so that all the records of students who took an offering are stored together and retrieved with cost 2+1. Therefore, the cost of this process for 10 offerings is 30 and the total cost is 33.