**Overall Explanation**

I have tested each index separately. I have used scale factor as 5 to see the difference clearly. I have conducted 2 experiment for each index.

The first one is, running all statements with and without the index.

The second one is only running the related query, with and without the index.

At the end of this document, you can see the total difference, i.e. executing all queries without indexes and with all indexes.

## **Query studentCountForDormitoryRoom**

**Explanation:** I have chosen index as dormitory and room pairs, because the query depends on those. I have tried hash index on those columns (dorm, room) but psql does not allow multicolumn indexes.

**Used index**: CREATE INDEX dorm\_room ON STUDENT(room, dormitory);

**Test 1:** With all statements running

Result without index: 19080 ms

Result with index: 15900 ms

**Test 2:** Using only studentCountForDormitoryRoomStatement running

Result without index: 4000 ms

Result with index (dorm,room) B+: 500 ms

Result with index (dorm) HASH: 1000 ms

Result with index (dorm) B+: 1000 ms

As a side node, I have tried 3 different index. All 3 indexes give similar results (~15500 ms), that is because of other queries' overhead.

**Other indexes that I have tried:**

CREATE INDEX dorm\_room\_hash on STUDENT USING HASH (dormitory);

and

CREATE INDEX dorm\_room ON STUDENT(dormitory);

## **Query studentCountForAgeRange**

**Explanation:** I have chosen B+ tree index because of the range query

**Used index:** CREATE INDEX age\_range ON STUDENT(age);

**Test 1:** All statements are running

Result without index: 20900 ms

Result with the index: 16470 ms

**Test 2:** Only studentCountForAgeRange running

Result without index: 3550 ms

Result with the index below: 1100 ms

## **Query professorCountForDepartment**

**Explanation:** I have tried both B+, Hash and cluster around department in Faculty table indexes on this query. With or without indexes they gave similar results. This might be because of even I use the scaling factor as 5 there are 294 entries in the Faculty relation which is a small dataset. The query plan does not change when I add the indexes.

**Used index:** CREATE INDEX prof\_department ON FACULTY(department);

**Test 1:** All statements are running

Result without index: 20000 ms

Result with the index below: 19400 ms

**Test 2:** Only professorCountForDepartment running

Result without index: 657 ms

Result with the index B+: 650 ms

Result with the index HASH: 660 ms

**Other indexes that I have tried:**

CREATE INDEX prof\_department\_hash on FACULTY USING HASH(department);

And

CREATE INDEX prof\_department ON FACULTY(department);

CLUSTER prof\_department ON FACULTY;

CLUSTER Faculty;

## **Query studentCountForDepartment**

**Explanation**: Since the dataset is larger than the professor/faculty relation, I can see the difference between using index and not using index. Also, clustering the relation Department around “department” field increased speed by 150ms. I can see that, without index, the plan is sequential scan. After the index with clustering, the plain is bitmap index scan (I assume that it is a kind of index scan).

**Used index**:

CREATE INDEX student\_department ON Registered(department);

CLUSTER student\_department ON Registered;

CLUSTER Registered;

**Test 1:** All statements are running

Result without index: 20000 ms

Result with the index and clustering: 16700 ms

**Test 2:** Only studentCountForDepartment running

Result without index: 3200 ms

Result with the index and clustering: 700 ms

Result with the index B+: 850 ms

Result with the index HASH: 850 ms

**Other indexes that I have tried**

CREATE INDEX student\_department\_hash ON Registered USING HASH (department);

AND

CREATE INDEX student\_department ON Registered(department);

## **Query studentsForProfessor**

**Explanation**: In the default query plan, it uses hash joins and sequential scan on Advisor (professor id) and sequential scan on Professor on first name and last name. Since the advisor has 3210 rows, I will do an index on professor id attribute. After the index on professor id, the plan is changed to bitmap heap scan on advisor.

**Used index:** CREATE INDEX advisor\_professor ON Advisor(professor\_id, student\_id);

**Test 1:** All statements are running

Result without index: 20000 ms

Result with the index: 17400 ms

**Test 2:** Only studentsForProfessor running

Result without index: 2600 ms

Result with index: 935 ms

## **Query professorsForStudent**

**Explanation**: In the default query plan, it uses sequential scan on student first name and last name. Others were bitmap heap scan, so I thought I cannot increase them further. I have added the index on first name and last name to change sequential scan to the index scan.

**Used index:** CREATE INDEX student\_name\_surname ON Student(first\_name, last\_name);

**Test 1:** All statements are running

Result without index: 20000 ms

Result with the index: 16300 ms

**Test 2:** Only studentsForProfessor running

Result without index: 3300 ms

Result with index: 650 ms

## **Query professorTitle**

**Explanation**: The default query plan is sequential scan. I have tried to add both B+ index and Hash index to the relation Professor on attribute title. The query execution plan did not change. I thought about changing clustering from professor id to title but this will affect heavily other queries (2 of the queries uses professor id as a join attribute). So, I have decided not to use any index on this query to not increase the complexity of the database.

**Test 1:** All statements are running

Result without index: 20000 ms

Result with the index: 19500 ms

**Test 2:** Only studentsForProfessor running

Result without index: 800 ms

Result with index: 800 ms

**Other indexes that I have tried:**

CREATE INDEX prof\_title ON Professor USING HASH (title);

AND

CREATE INDEX prof\_title ON Professor (title);

## **Overall performance changing**

Result without indexes: 19300 ms

Result with indexes: 4700 ms