# Homework 2 - Group 12 - 5707

| Email |
| --- |
| Name: |

@Bela D derga002@umn.edu

@Kelsey N neis@umn.edu

@Ashwin Sridhar sridh052@umn.edu

## Question 5

Suppliers( sid: integer, sname: string, address: string)
Parts(pid: integer, pname: string, color: string)
Catalog( sid: integer, pid: integer, cost: real)

### Question 5.2.1 Find the pnames of parts for which there is some supplier.

```
SELECT DISTINCT p.pname
FROM parts p, catalog c
WHERE c.pid = p.pid
```

### Question 5.2.2 Find the snames of suppliers who supply every part.

```
SELECT s.snames
FROM Suppliers s
WHERE NOT EXISTS ((SELECT p.pname
  FROM Parts p)
  EXCEPT (SELECT c.pid
    FROM Catalog c
    WHERE c.pid = p.pid
    AND c.sid = s.sid) );
```

### Question 5.2.3 Find the snames of suppliers who supply every red part.

```
SELECT s.snames
FROM Suppliers s
WHERE NOT EXISTS ((SELECT p.pname
  FROM Parts p)
  EXCEPT (SELECT c.pid
    FROM Catalog c
    WHERE c.pid = p.pid
    AND c.sid = s.sid
    AND p.color = ‘red’) );
```

### Question 5.2.4

#### Find the pnames of parts supplied by Acme Widget Suppliers and no one else.

```
SELECT p.pname
  FROM Parts p, Catalog c
  WHERE p.pid = c.pid
  AND NOT EXISTS
    (SELECT s.sid
     FROM Suppliers s
```

```
     WHERE s.sid = c.sid
     AND s.sname != 'Acme Widget Suppliers');
```

## Question 5.2.5

### Find the sids of suppliers who charge more for some part than the average cost of that part (averaged over all the suppliers who supply that part).

```
SELECT s.sid
FROM suppliers s, Catalog c
WHERE s.sid = c.sid
AND c.cost >
    (SELECT AVG(c2.cost)
     FROM Catalog c2
     WHERE c2.pid = c.pid);
```

## Question 5.2.6

### For each part, find the sname of the supplier who charges the most for that part.

```
SELECT s.sname AS Supplier, p.pid, c.cost As Cost
FROM (SELECT c2.pid, MAX (c2.cost) AS max_cost
      FROM Catalog c2
      GROUP BY c2.pid) v, Catalog c, Suppliers s, Parts p
WHERE c.cost = v.max_cost
AND c.pid = v.pid
AND c.sid = s.sid
AND v.pid = p.pid
ORDER BY p.pid ASC;
```

## Question 5.2.7

### Find the sids of suppliers who supply only red parts.

```
SELECT DISTINCT(c.sid)
FROM Catalog c, Parts p
WHERE p.color = 'Red'
AND p.pid = c.pid
AND c.sid NOT IN (SELECT c2.sid
                FROM Parts p2, Catalog c2
                WHERE p2.color <> 'Red'
                AND p2.pid = c2.pid
                )
```

## Question 5.2.8

### Find the sids of suppliers who supply a red part and a green part.

```
SELECT c.sid
FROM Catalog c, Parts p
WHERE p.pid = c.pid
AND p.color = 'Red'
AND c.sid in
(
SELECT c2.sid
FROM Catalog c2, Parts p2
WHERE p2.pid = c2.pid
AND p2.color = 'Green'
)
```

## Question 5.2.9

**Find the sids of suppliers who supply a red part or a green part.**

```
SELECT DISTINCT(c.sid)
FROM Catalog c, Parts p
WHERE p.pid = c.pid
AND p.color = 'Red'
OR c.sid in
(
SELECT c2.sid
FROM Catalog c2, Parts p2
WHERE p2.pid = c2.pid
AND p2.color = 'Green'
)
```

## Question 5.2.10

**For every supplier that only supplies green parts, print the name of the supplier and the total number of parts that she supplies.**

```
---BOTH DON"T WORK

SELECT s.sname, COUNT(c.pid)
FROM (SELECT DISTINCT(c.sid)
FROM Catalog c, Parts p
WHERE p.color = 'Green'
AND p.pid = c.pid
AND c.sid NOT IN (SELECT c2.sid
                FROM Parts p2, Catalog c2
                WHERE p2.color <> 'Green'
                AND p2.pid = c2.pid
                )) gS, Catalog c, Suppliers s
WHERE gs.sid = s.sid
GROUP BY s.sname
```

## Question 5.2.11

**For every supplier that supplies a green part and a red part, print the name and price of the most expensive part that she supplies.**

```
SELECT p.pname, MAX(c.cost)
 FROM catalog c, parts p
 WHERE c.pid = p.pid
 AND p.color = 'Red'
 AND c.sid IN
   (SELECT c2.sid
    FROM catalog c2, parts p2
    WHERE c2.pid = p2.pid
      AND p2.color = 'Green')
 Group By p.pname
```

## Question 5.8.1

**Student (snum: integer, sname: string, major: string,**
**level: string, age: integer)**
**Class(name: string, meets_at: time, room: string, fid: integer)**
**Enrolled(snum: integer, cname: string)**
**Faculty(fid: integer, fname: string, deptid: integer)**

```
CREATE TABLE Student (
  snum int,
  sname char(20),
  major char(20),
  level char(20),
  age int,
  PRIMARY KEY(snum)
);


CREATE TABLE Class (
  name char(20),
  meets_at time,
  room char(20),
  fid int,
  PRIMARY KEY(name),
  FOREIGN KEY(fid) REFERENCES Faculty
  ON DELETE SET NULL
);

CREATE TABLE Enrolled (
  snum int,
  cname char(20),
  PRIMARY KEY(snum, cname),
  FOREIGN KEY(snum) REFERENCES Student(snum)
  ON DELETE CASCADE,
  FOREIGN KEY(cname) REFERENCES Class(name)
  ON DELETE CASCADE
);

CREATE TABLE Faculty (
  fid int,
  fname char(20),
  deptid int,
  PRIMARY KEY(fid),
);
```

### Question 5.8.2

### (a) Every class has a minimum enrollment of 5 students and a maximum enrollment of 30 students.

```
--- Do we have to specify whether the constraint triggers on update or delete?
--- TRIGGER on Insert into Enrolled
CONSTRAINT max30Enrolees
CHECK (0 = (SELECT COUNT(e.cname) AS ccount
          FROM Class c, Enrolled e
          WHERE c.cname = e.cname
          GROUP BY c.cname)
          WHERE ccount > 30
        ))

--- TRIGGER on Delete into Enrolled
CONSTRAINT min5Enrolees
CHECK (0 = (SELECT COUNT(e.cname) AS ccount
          FROM Class c, Enrolled e
          WHERE c.cname = e.cname
          GROUP BY c.cname)
          WHERE ccount < 5
        ))
```

### (b) At least one class meets in each room.

`room` doesn't have its own relation, so we cannot tell whether there are rooms where no classes meet. We also can't tell how many rooms there are in total to choose from. Lastly, for a valid schema at the moment of creating, all the classes would have to span all the rooms at least once per room.

### (c) Every faculty member must teach at least two courses.

Can not guarantee that each faculty teaches at least 2 classes as the trigger has to be on the Class table. Can only guarantee that existing teaching arrangements don't fall below minimum.

```
--- Trigger on DELETE on Class table
CONSTRAINT min_appointment
CHECK (
 NOT EXISTS ( SELECT c.fid
      FROM Class c,
      GROUP BY c.fid
      HAVING COUNT(*) < 2
  )
)

---Alternate solution
CREATE ASSERTION Teach Two
CHECK(
(SELECT COUNT(*)
FROM Faculty F, Class C
WHERE F.fid = C.fid
GROUP BY C.fid
HAVING COUNT(*)<2) = 0)
```

### (d) Only faculty in the department with deptid=33 teach more than three courses.

```
--- Trigger on INSERT on Class table
CONSTRAINT dept_33_exception
CHECK (
 NOT EXISTS ( SELECT c.fid
      FROM Class c, Faculty f
      WHERE f.deptid <> 33
      AND c.fid = f.fid
      GROUP BY c.fid
      HAVING COUNT(*) > 3
  )
)
```

### (e) Every student must be enrolled in the course called Math 101.

```
--- Trigger on DELETE of Enrollment table
CONSTRAINT math101
CHECK NOT EXISTS(
      (
        SELECT s.snum
        FROM Students s
        WHERE NOT EXISTS (
            SELECT e.snum
            FROM Enrollments e
            WHERE s.snum = e.snum
            AND e.cname = 'Math101'
        )
      )
)
```

### (f) The room in which the earliest scheduled class (i.e., the class with the smallest meets_at value) meets should not be the same as the room in which the latest scheduled class meets.

```
--- Trigger on INSERT and/or DELETE of Class table
CONSTRAINT meeting_time
CHECK NOT EXISTS (
    SELECT MIN(c.meets_at) AS earliest, MAX(c.meets_at) AS latest
    FROM Class c
    WHERE earliest.room = latest.room
   )
```

### (g) Two classes cannot meet in the same room at the same time.

```
--- Trigger on INSERT of Class table
CONSTRAINT diff_room_req
```

```
CHECK NOT EXISTS (
    SELECT c.cname, c.meets_at
    FROM Class c
    WHERE (
          SELECT c2.name, c2.meets_at
          FROM Class c2,
          WHERE c.meets_at = c2.meets_at
          AND c.room = c2.room
    )
   );
```

## (h) The department with the most faculty members must have fewer than twice the number of faculty members in the department with the fewest faculty members.

```
--- Trigger on INSERT or DELETE of Faculty Table
CONSTRAINT dept_levels
CHECK NOT EXISTS (
    SELECT f.cname, c.meets_at
    FROM Class c
    WHERE (
          SELECT MAX
          (SELECT f.deptid Count(*) biggest
          FROM Faculty f,
          Group By f.deptid) as dept_id_totals
    )
   );
```

### (i) No department can have more than 10 faculty members.

```
CONSTRAINT faculty_limit
CHECK(
  (SELECT COUNT(*)
  FROM Faculty F
  GROUP BY F.deptid
  Having COUNT(*)>10) = 0)
);
```

### (j) A student cannot add more than two courses at a time (i.e., in a single update)

Each db write is single write. There is no way to limit updates to instances of user or attribute within some period of time. This check has to be application level.

### (k) The number of CS majors must be more than the number of Math majors.

```
CONSTRAINT cs_majors_more_than_math_majors
CHECK (
(SELECT COUNT(*)
FROM Students S
WHERE S.major='CS' > (SELECT COUNT(*)
FROM Students S
WHERE S.major='Math'))
);
```

### (l) The number of distinct courses in which CS majors are enrolled is greater than the number of distinct courses in which Math majors are enrolled.

```
CONSTRAINT distinct_cs_greater_than_math
CHECK(
  (SELECT COUNT(*) FROM(
    SELECT E.cname, COUNT(E.cname)
    FROM Enrollment E, Students S
    WHERE S.snum = E.snum
    AND S.major = 'CS'
```

```
    GROUP BY E.cname)) > (SELECT COUNT(*)
    FROM (SELECT E.cname, COUNT(E.cname)
        FROM Enrollment E, Students S
        WHERE S.snum = E.snum
        AND S.major = 'Math'
      CHECK(( SELECT COUNT(S.snum)FROM Students SWHERE S.major='CS')>0))CHECK(( SELECT COUNT(S.snum)FROM Students SWHERE S.major='CS')>0))
);
```

### (m) The total enrollment in courses taught by faculty in the department with deptid=33 is greater than the number of math majors.

```
CONSTRAINT dept33_greater_than_math_majors
CHECK(
  (SELECT COUNT(E.snum)
  FROM Enrollment E, Faculty F, Class C
  WHERE E.cname = C.name
  AND C.fid = F.fid
  AND F.deptid = 33) > (SELECT COUNT(E.snum)
    FROM Student S
    WHERE S.Major = 'Math'));
```

### (n) There must be at least one CS major if there are any students whatsoever.

```
--- Trigger on insert or delete of Student
CONSTRAINT one_cs_major
CHECK(
  ((SELECT COUNT(S.snum)
  FROM Students S
  WHERE S.major = 'CS') > 0) OR
  (SELECT COUNT(*)
  FROM Students S) = 0)
);
```

### (o) Faculty members from different departments cannot teach in the same room.

```
--- Constraint on Insert of Class table
CONSTRAINT dept_room
CHECK(
( SELECT COUNT(*)
FROM Faculty F1, Faculty F2, Class C1, Class C2
WHERE F1.fid = C1.fid
AND F2.fid = C2.fid
AND C1.room = C2.room
AND F1.deptid <> F2.deptid) = 0)
```

## Question 8.6

**I/O Cost Comparisons**

| File type | Scan | Equality search | Range search | Insert | Delete |
|---|---|---|---|---|---|
| Heap file | BD | 0.5BD | BD | 2D | Search + D |
| Sorted file | BD | Dlog2(B) | Dlog2(B) + # matches | Search + BD | Search + BD |
| Clustered file | 1.5BD | DLogF(1.5B) | DlogF(1.5B) + # matches | Search + D | Search + D |
| Unclustered tree index | BD(R+0.15) | D(1 + logF(0.15B) | DlogF(0.15B) + # matches | D(3 + logF(0.15B) | Search + 2D |
| Unclustered hash index | BD(R + 0.125) | 2D | BD | 4D | Search + 2D |
| Untitled | | | | | |

### Key:

B - number of data pages when records are packed onto pages with no wasted space

C - avg time to process a record (compare a field value to a selection constant) = 100 nanoseconds

D - avg I/O cost = 15 miliseconds

F - fan out of a tree (avg: 100)

H - avg time to apply hash function to a record = 100 nanoseconds

R - number of records per page

## Question 8.10

### Emp( eid: integer, sal: integer l age: real, did: integer)

There is a clustered index on cid and an unclustered index on age.

### 1. How would you use the indexes to enforce the constraint that eid is a key?

To enforce the constraint that eid is a key it is important to ensure that the index is unique and dense. Since the index is clustered, it is sorted, so it is dense. On insertion of the new record the new eid must be unique. In other words no key of the index can have a duplicate. Mapping of key to tuple must be 1:1.

### 2. Give an example of an update that is definitely speeded up because of the available indexes. (English description is sufficient.)

The nature of the sorted and indexed storage of eids means that updating did of a range of employees would be speeded up, as looking up the employees we can take the sorted nature of the clustered index. Because we are not removing or adding new employees or changing their ages, the index does not have to be updated, we just take advantage of its speed for look-up (SCAN)

### 3. Give an example of an update that is definitely slowed down because of the indexes. (English description is sufficient.)

Any operation that requires update of the index will be slowed because of the index. For example, adding new employees. While they will be appended to the bottom of the range (continuing the sorted arranged of clustered index), this will still require the eid index to be updated. The same with updating the age of employees.

### 4. Can you give an example of an update that is neither speeded up nor slowed down by the indexes?

An update of a specific department's salaries by 10%. No benefit in looking up the department or no slow down in updating the salaries since its not indexed.

## Question 12.2

**Consider a relation R( a,b,c,d,e) containing 5,000,000 records, where each data page of the relation holds 10 records. R is organized as a sorted file with secondary indexes. Assume that R.a is a candidate key for R, with values lying in the range 0 to 4,999,999, and that R is stored in R.a order. For each of the following relational algebra queries, state which of the following three approaches is most likely to be the cheapest:**
**• Access the sorted file for R directly.**
**• Use a (clustered) B+ tree index on attribute R.a.**
**• Use a linear hashed index on attribute R.a.**

**1.** $\sigma_{a<50{,}000}(R)$ **- Sorted is the cheapest choice due to the cost of traversing to the B+ tree leaf.**
**2.** $\sigma_{a=50{,}000}(R)$ **- direct hash key - value calculation is cheapest. So Linear HashIndex**
**3.** $\sigma_{a>50{,}000 \cap a<50{,}010}(R)$ **B+ cheapest since it bypasses scanning the uneneded first 50K**
**4.** $\sigma_{a\neq 50{,}000}(R)$ **Sorted is cheapest, remove unwanted from fetched page (processing, cheap cost)**

## Question 12.4 Consider the following schema with the Sailors relation:

Sailors(sid: integer, sname: string, rating: integer, age: real)
For each of the following indexes, list whether the index matches the given selection conditions.
If there is a match, list the primary conjuncts.

1. A B+-tree index on the search key ( Sailors.sid ).
   (a) $\sigma_{sailors.sid<50{,}000}(Sailors)$ - Match, primary conjunct: sailors.sid < 50,000
   (b) $\sigma_{sailors.sid=50{,}000}(Sailors)$ - Match, primary conjunct: sailors.sid = 50,000
2. A hash index on the search key ( Sailors.sid ).
   (a) $\sigma_{sailors.sid<50{,}000}(Sailors)$ - Not a match - range queries can't be applied to Hash Indexes
   (b) $\sigma_{sailors.sid=50{,}000}(Sailors)$ - Match, primary conjunct: sailors.sid = 50,000
3. A B+-tree index on the search key ( Sailors.sid, Sailors.age ).
   (a) $\sigma_{sailors.sid<50{,}000 \wedge sailors.age=21}(Sailors)$ - Match, primary conjuncts sailors.sid < 50,000 and sailors.age=21
   (b) $\sigma_{sailors.sid=50{,}000 \wedge sailors.age>21}(Sailors)$ - Match, primary conjuncts sailors.sid = 50,000 and sailors.age>21
   (c) $\sigma_{sailors.sid=50{,}000}(Sailors)$ - Match, primary conjunct: sailors.sid=50,000
   (d) $\sigma_{sailors.age=21}(Sailors)$ - Not a match, because age is not a prefix of the search key
4. A hashtree index on the search key ( Sailors.sid, Sailors.age ).
   (a) $\sigma_{sailors.sid=50{,}000 \wedge sailors.age=21}(Sailors)$ - Match, primary conjuncts: sailors.sid = 50,000 and sailors.age=21
   (b) $\sigma_{sailors.sid=50{,}000 \wedge sailors.age>21}(Sailors)$ - Not a match, range queries can't be applied to Hash Indexes
   (c) $\sigma_{sailors.sid=50{,}000}(Sailors)$ - Match, primary conjunct: sailors.sid = 50,000
   (d) $\sigma_{sailors.age=21}(Sailors)$ - Not a match, because age is not a prefix of the search key