Assessment was due by Tue, 2023-08-22 00:00:00

COMP 421 Midterm 2 Fall 2023

Note: There are a total of 100 points on this exams.

Don't panic!

You have 75 minutes to finish the exam.

- You *should* stay in full screen mode.
  - A <ctrl-f> find will give you a warning, which is OK
- You must hand this midterm during class or prearranged midterm timeframe.
  - Avoid accidental submissions. Fill in your name when you are ready to submit.
  - Only your **first** submission will be accepted/graded for full credit.
  - If and only if you submit your exam during class you will have a chance to resubmit it for reduced credit. See subsequent submissions on the syllabus for more information.
  - After all students have submitted and the grader has finished, the submit button will be enabled.
  - You have until 2023-11-16 11:59:59 to submit any subsequent submissions
  - Plan your time judiciously!

I recommend that you have several pieces of scrap paper to doodle notes on during the exam. I *strongly* recommend you read the whole exam and begin with questions you know how to solve quickly. Some questions will be harder or take longer than others; don't spend all your time on one question worth only a few points!

Consider this midterm **closed book**.
You can **NOT** reference other online homeworks, worksheets, etc.
You can use your notes or other things printed out. They should be on paper as you may not switch screens after starting the exam.

You **MAY NOT** Google for anything, You **MAY NOT** leave this website, you **MAY NOT** visit any websites, and you **MAY NOT** copy from a friend. Do not paste information into your midterm unless you know it came from your midterm. You **MAY NOT** receive help from anyone.

If you do not know the origin of material you should not paste it into this exam. All material pasted into this exam must originate from this exam. This implies, but is not limited to, copying from previous assignments, copying from text messages, or copying from **any** website.

You **MUST** use the Google Chrome browser.

The instruction team will **not** answer questions about course content, SQL syntax, etc. We will only deal with issues related to exam implementation.

If your browser hangs, for example because of a bad SQL query, simply kill the page and refresh. It *should* restore all your work even if it doesn't re-evaluate all answers, color-highlight boxes, etc.

You may **NOT** leave the classroom before you submit your exam. When you submit your exam you must enter the code displayed on the screen at the front of the class or given to you by ARS.`

You **must not** use your computer or phone in the classroom after you submit your exam. After submitting your exam, simply leave the classroom or ARS.

The browser will change input box color green to indicate correctness. A black or red box indicates an incorrect answer.

Note that HTML select statements with drop-downs are simple multiple choice questions. No highlighting of correct answers are done for select questions.

Green highlight should just assist you. If you believe your answer is correct and the input box did not turn green, continue on. Per the syllabus, highlighting is simply an aide not a guarantee.

Note: For database queries that are applied to two databases, **two** green lights are required to get any credit for the question.

SQL Tutorial Cheat Sheet

Following are three SQL tutorial cheat sheets available from http://www.sqltutorial.org

SQL CHEAT SHEET http://www.sqltutorial.org

QUERYING DATA FROM A TABLE

```
SELECT c1, c2 FROM t;
```
Query data in columns c1, c2 from a table

```
SELECT * FROM t;
```
Query all rows and columns from a table

```
SELECT c1, c2 FROM t
WHERE condition;
```
Query data and filter rows with a condition

```
SELECT DISTINCT c1 FROM t
WHERE condition;
```
Query distinct rows from a table

```
SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
```
Sort the result set in ascending or descending order

```
SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
```
Skip *offset* of rows and return the next n rows

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
```
Group rows using an aggregate function

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
```
Filter groups using HAVING clause

QUERYING FROM MULTIPLE TABLES

```
SELECT c1, c2
FROM t1
INNER JOIN t2 ON condition;
```
Inner join t1 and t2

```
SELECT c1, c2
FROM t1
LEFT JOIN t2 ON condition;
```
Left join t1 and t1

```
SELECT c1, c2
FROM t1
RIGHT JOIN t2 ON condition;
```
Right join t1 and t2

```
SELECT c1, c2
FROM t1
FULL OUTER JOIN t2 ON condition;
```
Perform full outer join

```
SELECT c1, c2
FROM t1
CROSS JOIN t2;
```
Produce a Cartesian product of rows in tables

```
SELECT c1, c2
FROM t1, t2;
```
Another way to perform cross join

```
SELECT c1, c2
FROM t1 A
INNER JOIN t2 B ON condition;
```
Join t1 to itself using INNER JOIN clause

USING SQL OPERATORS

```
SELECT c1, c2 FROM t1
UNION [ALL]
SELECT c1, c2 FROM t2;
```
Combine rows from two queries

```
SELECT c1, c2 FROM t1
INTERSECT
SELECT c1, c2 FROM t2;
```
Return the intersection of two queries

```
SELECT c1, c2 FROM t1
MINUS
SELECT c1, c2 FROM t2;
```
Subtract a result set from another result set

```
SELECT c1, c2 FROM t1
WHERE c1 [NOT] LIKE pattern;
```
Query rows using pattern matching %, _

```
SELECT c1, c2 FROM t
WHERE c1 [NOT] IN value_list;
```
Query rows in a list

```
SELECT c1, c2 FROM t
WHERE  c1 BETWEEN low AND high;
```
Query rows between two values

```
SELECT c1, c2 FROM t
WHERE  c1 IS [NOT] NULL;
```
Check if values in a table is NULL or not

SQL CHEAT SHEET http://www.sqltutorial.org

MANAGING TABLES

```
CREATE TABLE t (
    id INT PRIMARY KEY,
    name VARCHAR NOT NULL,
    price INT DEFAULT 0
);
```
Create a new table with three columns

```
DROP TABLE t ;
```
Delete the table from the database

```
ALTER TABLE t ADD column;
```
Add a new column to the table

```
ALTER TABLE t DROP COLUMN c ;
```
Drop column c from the table

```
ALTER TABLE t ADD constraint;
```
Add a constraint

```
ALTER TABLE t DROP constraint;
```
Drop a constraint

```
ALTER TABLE t1 RENAME TO t2;
```
Rename a table from t1 to t2

```
ALTER TABLE t1 RENAME c1 TO c2 ;
```
Rename column c1 to c2

```
TRUNCATE TABLE t;
```
Remove all data in a table

USING SQL CONSTRAINTS

```
CREATE TABLE t(
    c1 INT, c2 INT, c3 VARCHAR,
    PRIMARY KEY (c1,c2)
);
```
Set c1 and c2 as a primary key

```
CREATE TABLE t1(
    c1 INT PRIMARY KEY,
    c2 INT,
    FOREIGN KEY (c2) REFERENCES t2(c2)
);
```
Set c2 column as a foreign key

```
CREATE TABLE t(
    c1 INT, c1 INT,
    UNIQUE(c2,c3)
);
```
Make the values in c1 and c2 unique

```
CREATE TABLE t(
  c1 INT, c2 INT,
  CHECK(c1> 0 AND c1 >= c2)
);
```
Ensure c1 > 0 and values in c1 >= c2

```
CREATE TABLE t(
    c1 INT PRIMARY KEY,
    c2 VARCHAR NOT NULL
);
```
Set values in c2 column not NULL

MODIFYING DATA

```
INSERT INTO t(column_list)
VALUES(value_list);
```
Insert one row into a table

```
INSERT INTO t(column_list)
VALUES (value_list),
             (value_list), ....;
```
Insert multiple rows into a table

```
INSERT INTO t1(column_list)
SELECT column_list
FROM t2;
```
Insert rows from t2 into t1

```
UPDATE t
SET c1 = new_value;
```
Update new value in the column c1 for all rows

```
UPDATE t
SET c1 = new_value,
       c2 = new_value
WHERE condition;
```
Update values in the column c1, c2 that match the condition

```
DELETE FROM t;
```
Delete all data in a table

```
DELETE FROM t
WHERE condition;
```
Delete subset of rows in a table

SQL CHEAT SHEET http://www.sqltutorial.org

MANAGING VIEWS

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
```
Create a new view that consists of c1 and c2

```
CREATE VIEW v(c1,c2)
AS
SELECT c1, c2
FROM t;
WITH [CASCADED | LOCAL] CHECK OPTION;
```
Create a new view with check option

```
CREATE RECURSIVE VIEW v
AS
select-statement -- anchor part
UNION [ALL]
select-statement; -- recursive part
```
Create a recursive view

```
CREATE TEMPORARY VIEW v
AS
SELECT c1, c2
FROM t;
```
Create a temporary view

```
DROP VIEW view_name;
```
Delete a view

MANAGING INDEXES

```
CREATE INDEX idx_name
ON t(c1,c2);
```
Create an index on c1 and c2 of the table t

```
CREATE UNIQUE INDEX idx_name
ON t(c3,c4);
```
Create a unique index on c3, c4 of the table t

```
DROP INDEX idx_name;
```
Drop an index

SQL AGGREGATE FUNCTIONS

AVG returns the average of a list

COUNT returns the number of elements of a list

SUM returns the total of a list

MAX returns the maximum value in a list

MIN returns the minimum value in a list

MANAGING TRIGGERS

```
CREATE OR MODIFY TRIGGER trigger_name
WHEN EVENT
ON table_name TRIGGER_TYPE
EXECUTE stored_procedure;
```
Create or modify a trigger

WHEN
- BEFORE – invoke before the event occurs
- AFTER – invoke after the event occurs

EVENT
- INSERT – invoke for INSERT
- UPDATE – invoke for UPDATE
- DELETE – invoke for DELETE

TRIGGER_TYPE
- FOR EACH ROW
- FOR EACH STATEMENT

```
CREATE TRIGGER before_insert_person
BEFORE INSERT
ON person FOR EACH ROW
EXECUTE stored_procedure;
```
Create a trigger invoked before a new row is inserted into the person table

```
DROP TRIGGER trigger_name;
```
Delete a specific trigger

Database Schema

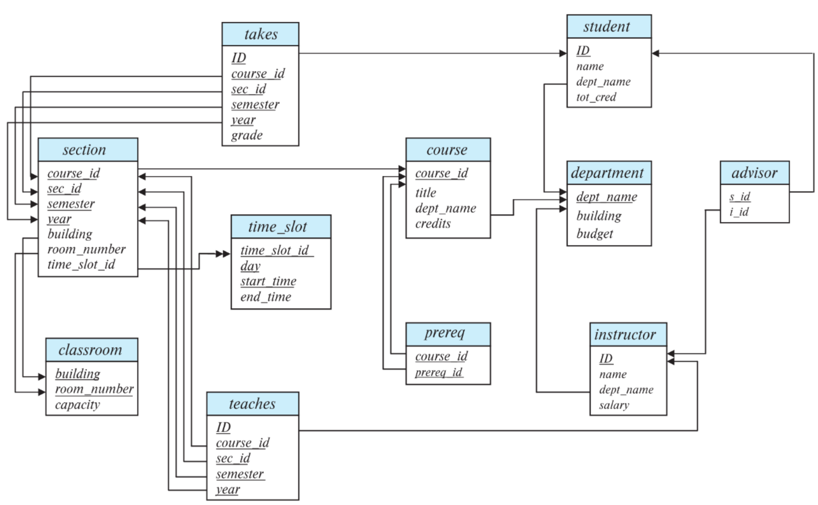
Here are the tables you'll find for the database used in the midterm. Your queries will be run against two versions of the database. One of the databases will be much smaller and only contain a subset of the information.

```
CREATE TABLE classroom
     (building      varchar(15),
     room_number  varchar(7),
     capacity     numeric(4,0),
     primary key (building, room_number)
     )
CREATE TABLE department
     (dept_name    varchar(20),
     building     varchar(15),
     budget       numeric(12,2) check (budget > 0),
     primary key (dept_name)
     )
CREATE TABLE course
     (course_id    varchar(8),
     title        varchar(50),
     dept_name    varchar(20),
     credits      numeric(2,0) check (credits > 0),
     primary key (course_id),
     foreign key (dept_name) references department (
```

```
    on delete set null
    )
CREATE INDEX idx_course_dept_name ON Course(dept_nam
CREATE TABLE instructor
    (ID          varchar(5),  -- instructor's ID
    name        varchar(20) not null,
    dept_name   varchar(20),
    salary      numeric(8,2) check (salary > 29000)
    primary key (ID),
    foreign key (dept_name) references department (
    on delete set null
    )
CREATE INDEX idx_instructor_id ON Instructor(ID)
CREATE INDEX idx_instructor_dept_name ON Instructor(
CREATE TABLE section
    (course_id    varchar(8),
    sec_id       varchar(8),
    semester     varchar(6)
    check (semester in ('Fall', 'Winter', 'Spring',
    year         numeric(4,0) check (year > 1701 an
    building     varchar(15),
    room_number  varchar(7),
    time_slot_id varchar(4),
    primary key (course_id, sec_id, semester, year)
    foreign key (course_id) references course (cour
    on delete cascade,
    foreign key (building, room_number) references
    on delete set null
    )
CREATE INDEX idx_section_year ON Section(year)
CREATE INDEX idx_section_semester ON Section(semeste
CREATE TABLE teaches
    (ID         varchar(5),  -- instructor's ID
    course_id  varchar(8),
    sec_id     varchar(8),
    semester   varchar(6),
    year       numeric(4,0),
    primary key (ID, course_id, sec_id, semester, y
    foreign key (course_id, sec_id, semester, year)
    references section (course_id, sec_id, semester
```

```
     on delete cascade,
     foreign key (ID) references instructor (ID)
     on delete cascade
     )
CREATE INDEX idx_teaches_id ON Teaches(ID)
CREATE INDEX idx_teaches_year ON Teaches(year)
CREATE INDEX idx_teaches_semester ON Teaches(semeste
CREATE TABLE student
     (ID         varchar(5),
     name       varchar(20) not null,
     dept_name  varchar(20),
     tot_cred   numeric(3,0) check (tot_cred >= 0),
     primary key (ID),
     foreign key (dept_name) references department (
     on delete set null
     )
CREATE UNIQUE INDEX idx_student_id ON Student(ID)
CREATE INDEX idx_student_dept_name ON Student(dept_n
CREATE TABLE takes
     (ID         varchar(5), -- Student ID
     course_id  varchar(8),
     sec_id     varchar(8),
     semester   varchar(6),
     year       numeric(4,0),
     grade      varchar(2),
     primary key (ID, course_id, sec_id, semester, y
     foreign key (course_id, sec_id, semester, year)
     references section (course_id, sec_id, semester
     on delete cascade,
     foreign key (ID) references student (ID)
     on delete cascade
     )
CREATE INDEX idx_takes_id ON Takes(ID)
CREATE INDEX idx_takes_semester ON Takes(semester)
CREATE INDEX idx_takes_year ON Takes(year)
CREATE TABLE advisor
     (s_ID       varchar(5),
     i_ID       varchar(5),
     primary key (s_ID),
     foreign key (i_ID) references instructor (ID)
```

```
        on delete set null,
        foreign key (s_ID) references student (ID)
        on delete cascade
        )
    CREATE INDEX idx_advisor_instructor_id ON Advisor(i_
    CREATE TABLE time_slot
        (time_slot_id    varchar(4),
        day             varchar(1),
        start_hr        numeric(2) check (start_hr >= 0
        start_min       numeric(2) check (start_min >= 0
        end_hr          numeric(2) check (end_hr >= 0 an
        end_min         numeric(2) check (end_min >= 0 a
        primary key (time_slot_id, day, start_hr, start
        )
    CREATE TABLE prereq
        (course_id       varchar(8),
        prereq_id       varchar(8),
        primary key (course_id, prereq_id),
        foreign key (course_id) references course (cour
        on delete cascade,
        foreign key (prereq_id) references course (cour
        )
```



Scratch area

The following scratch space can be used to help develop and test queries against a database described above. The database used by the exam grader will be different.

Execute Minimize Output

Questions For a total of 100 points

SQL Queries 35 points

In this section, you will write SQL queries for the university described in the book. Your queries will be tested immediately against two different databases. If your queries output matches the expected output, the displayed answers will be outlined in green. Your actual score will be determined when your query is tested against a different database but green feedback should mean that you are on track to receive full credit.

Did.Not.Take.1: List the name and total number of credits for students in the Math department who did not take the Math course named 'Computability Theory'.
Order the list in descending order by the name.

Execute 5 points Minimize Output

Teach.Most.1: Find the name of the teacher**(s)** in the Marketing department who teach the most classes.
NOTE: The classes do not have to be classes from the Marketing department.

Execute 5 points Minimize Output

Instructor.Count.1: List the names of Accounting instructors and the number of times they have taught a course in Chandler building. If they have never taught a course in that building, express the count as Null

Execute 5 points Minimize Output

All.Courses.1: List the course id and title of all the courses in the Accounting department. List them in order by their title, then course_id.

Execute 5 points Minimize Output

Student.Takes.All.1: List the names of all the students who have taken all the courses in the Accounting department. The students can be from any department; only the courses have to be from the Accounting department.

Execute 5 points Minimize Output

Courses.Always.Offered.1: List the title of the Accounting department courses that were offered each semester that this database was keeping

track of classes.
For example, for some reason this database has no classes for Summer or Winter 2001. But it does have courses for Fall and Spring semesters of that year. Find all the tuples of semesters and years that are in the database, then return the title of the Accounting courses that were offered during all of those tuples.

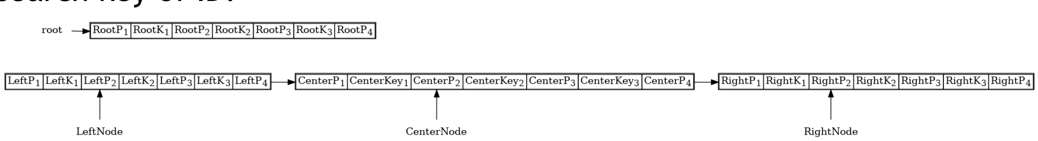
Execute 10 points Minimize Output

-

B-plus trees 50 points

In this section, you will show your knowledge of B^+ trees.

Assume the following B^+ tree with $n = 4$ and the **Takes** relation and a search key of ID:



Takes Relation

sec_id	semester	ID	course_id	year	grade
1	Fall	Student1	COMP110	2001	A+
2	Fall	Student6	COMP210	2022	F
3	Spring	Student2	COMP116	2006	C
4	Summer	Student3	COMP311	2019	A+
5	Summer	Student5	COMP110	2020	B
6	Winter	Student4	COMP283	2023	B+
7	Winter	Student6	COMP210	2012	A
8	Spring	Student7	COMP421	2022	A
9	Fall	Student8	COMP550	2021	C+

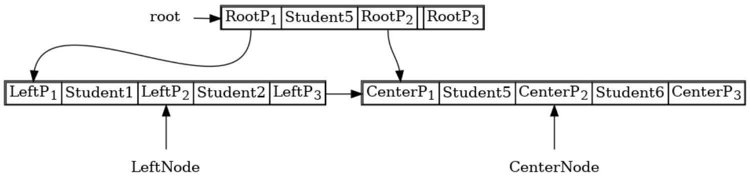
Complete.B.Plus.Tree.1: Complete the indices for relation Takes using a search key of ID. You should assume the same notation used in class and the same algorithm used in the book.

To facilitate having only one single correct answer, field CenterK3 must be 'Student6'. It may help you to start building your tree from field CenterK3.

Field	Value
RootP1	
RootK1	
RootP2	
RootK2	
RootP3	
RootK3	
RootP4	
LeftP1	
LeftK1	
LeftP2	
LeftK2	
LeftP3	
LeftK3	
LeftP4	
CenterP1	
CenterK1	
CenterP2	
CenterK2	
CenterP3	
CenterK3	
CenterP4	
RightP1	
RightK1	
RightP2	
RightK2	
RightP3	
RightK3	
RightP4	

20 points

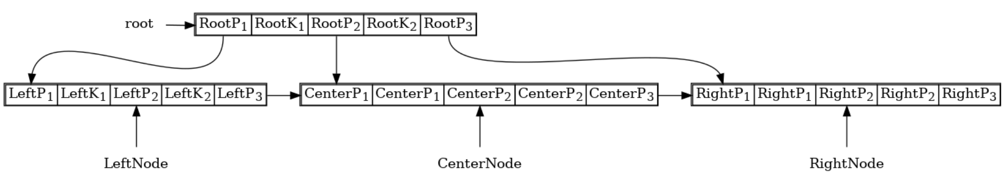
Assume the following B$^+$ tree with $n = 3$ and the **Takes** relation and a search key of ID:



Takes Relation →

sec_id	semester	ID	course_id	year	grade
1	Fall	Student1	COMP110	2001	A+
2	Fall	Student6	COMP210	2022	F
3	Spring	Student2	COMP116	2006	C
5	Summer	Student5	COMP110	2020	B

Insert.B.Plus.Tree.1: Complete the following table of ID search keys and pointers after an insert of record '6 Winter Student4 COMP283 2023 B+'. **Note:** You can be assured that RootP1 points to LeftNode, RootP2 points to CenterNode, and RootP3 points to RightNode.



Takes Relation →

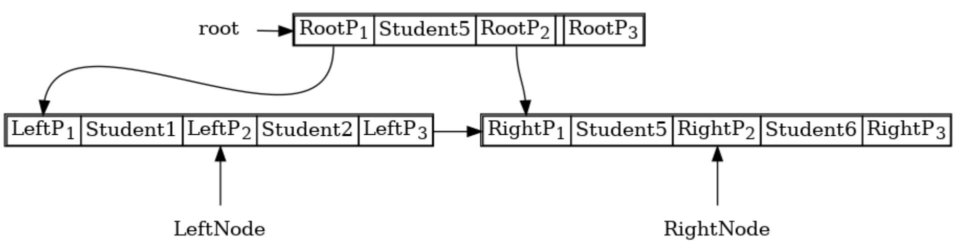
sec_id	semester	ID	course_id	year	grade
1	Fall	Student1	COMP110	2001	A+
2	Fall	Student6	COMP210	2022	F
3	Spring	Student2	COMP116	2006	C
5	Summer	Student5	COMP110	2020	B
6	Winter	Student4	COMP283	2023	B+

Field	Value
RootP1	
RootK1	
RootP2	
RootK2	
RootP3	
LeftP1	
LeftK1	
LeftP2	
LeftK2	
LeftP3	
CenterP1	

CenterK1	
CenterP2	
CenterK2	
CenterP3	
RightP1	
RightK1	
RightP2	
RightK2	
RightP3	

20 points

Assume the following B$^+$ tree with $n = 3$ and the **Takes** relation and a search key of ID:



Takes Relation →

sec_id	semester	ID	course_id	year	grade
1	Fall	Student1	COMP110	2001	A+
2	Fall	Student6	COMP210	2022	F
3	Spring	Student2	COMP116	2006	C
5	Summer	Student5	COMP110	2020	B

Delete.B.Plus.Tree.1: Complete the following table of ID search keys and pointers after the delete of record '5 Summer Student5 COMP110 2020 B'.

Field	Value
RootP1	
RootK1	
RootP2	
RootK2	
RootP3	
LeftP1	
LeftK1	

LeftP2	
LeftK2	
LeftP3	
RightP1	
RightK1	
RightP2	
RightK2	
RightP3	

10 points

Chapter Reading Review 15 points

Following are statements about a B$^+$-tree index with $P_1, P_2, \ldots, P_n$ pointers in each index node for a relation of M tuples.

A Every path from the root of the tree to the leaf of the tree is the same length

B Each non-leaf node stores n keys

C The height of the tree is proportional to the logarithm of the base n of M

D Lookup (search) on B$^+$-trees is straightforward and efficient

E B$^+$ trees are much shorter than other binary-tree structures such as AVL trees

B.Plus.Booleans.1: Given the book's discussion of B$^+$ trees in chapter 14, which of the above is a false statement?

5 points

Disk.I.O.Keys.1: For the following question, assume the same M tuples, n tuples per index, with 100 tuples per 4K block, the disk blocks are 4K in size, and assume the dense index B$^+$ search tree of ID kept entirely in memory such that no I/Os are required to read the B$^+$-tree, how many I/Os are required for `SELECT DISTINCT ID FROM Instructor`?

A M I/Os

B Zero I/Os

C n I/Os

D M/100 I/Os

E 100/M I/Os

F None of the above.

5 points

Disk.1: Assume a disk spins at 5400 revolutions per minute, transfer time of one block of t milliseconds, and a average seek time of s milliseconds. Write an expression that calculates the average time in milliseconds required to complete an I/O for one block of data.

Enter an expression 5 points