Home | Syllabus | What's next | Grades | Reading list | Handouts | Help | View your submissions for student1 | View feedback / answers for student1 | jmajikes

Assessment due **before** Fri, 2024-05-31 00:00:00.

# COMP 421 Midterm 1 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 2023-10-17 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.

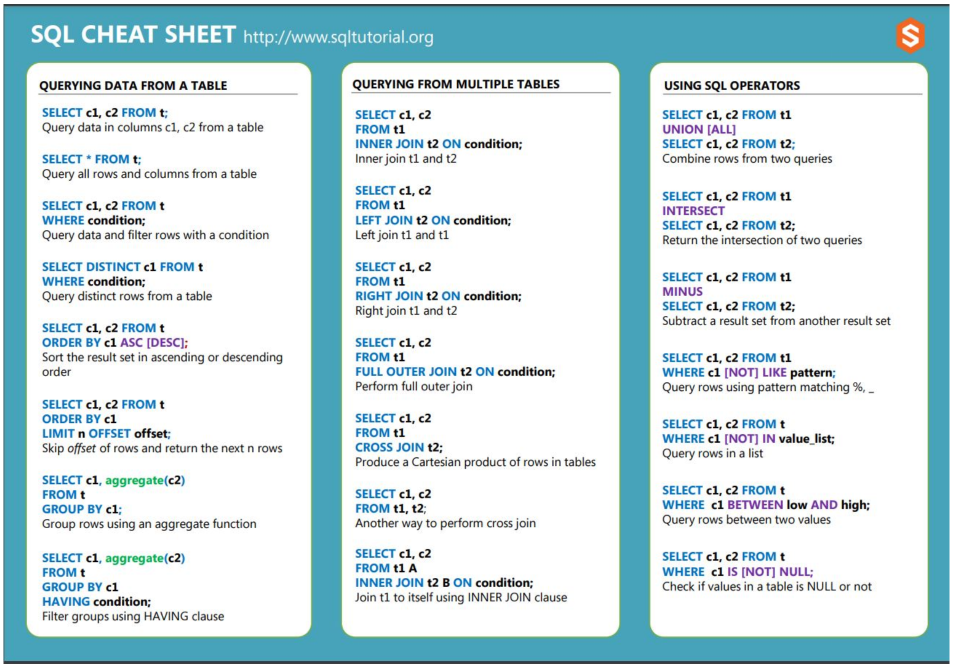
Note that HTML select statements with drop-downs are simple multiple choice questions. No highlighting of correct answers are done for select questions.

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

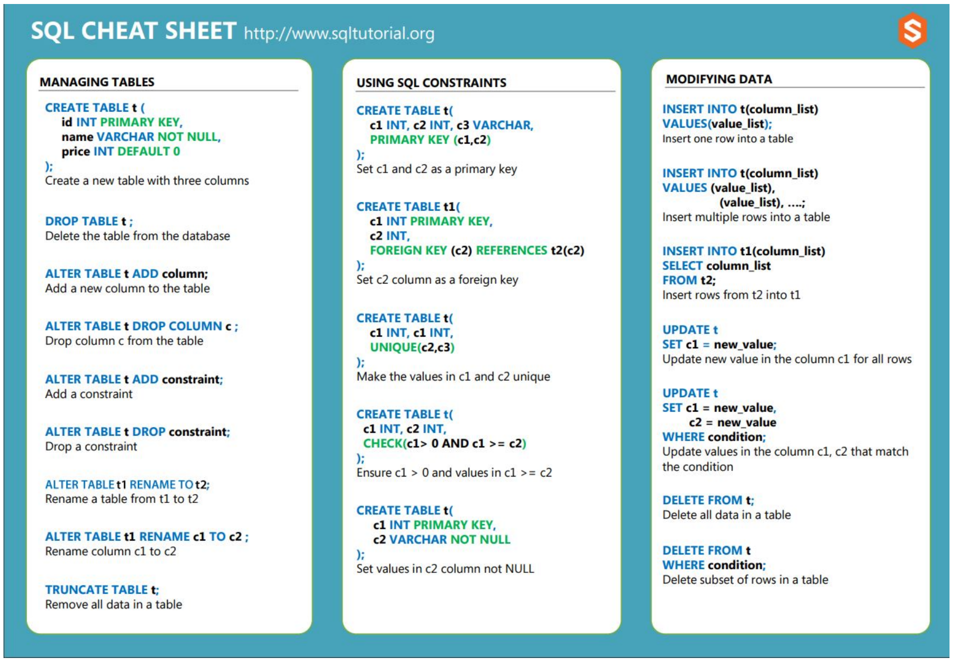
```
SELECT c1, c2 FROM t1
MINUS
SELECT c1, c2 FROM t2;
```
Subtract a result set from another result set

```
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

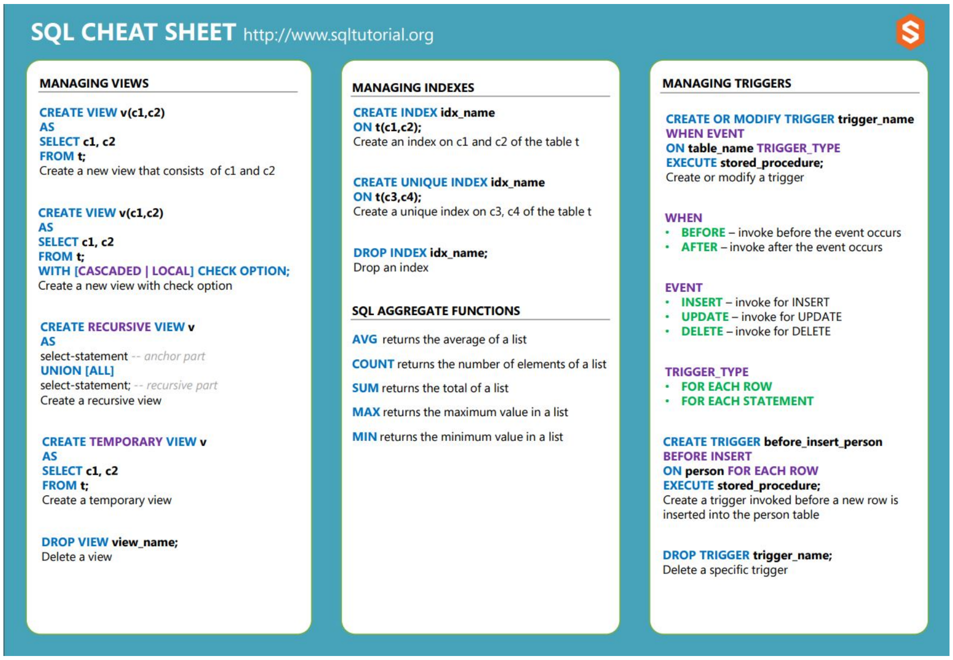
```
INSERT INTO t1(column_list)
SELECT column_list
FROM t2;
```
Insert rows from t2 into t1

```
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

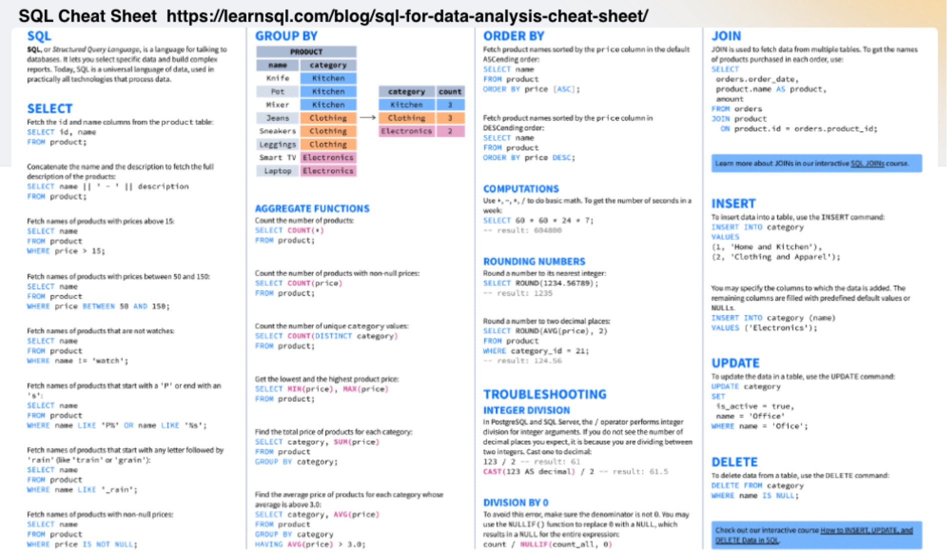
```
CREATE TRIGGER before_insert_person
BEFORE INSERT
ON person FOR EACH ROW
EXECUTE stored_procedure;
```

Create a trigger invoked before a new row is inserted into the person table

```
DROP TRIGGER trigger_name;
```

Delete a specific trigger

Following is a SQL tutorial cheat sheets available from http://learnsql.com



SQL Cheat Sheet https://learnsql.com/blog/sql-for-data-analysis-cheat-sheet/

SQL

**SQL**, or *Structured Query Language*, is a language for talking to databases. It lets you select specific data and build complex reports. Today, SQL is a universal language of data, used in practically all technologies that process data.

SELECT

Fetch the id and name columns from the product table:

```
SELECT id, name
FROM product;
```

Concatenate the name and the description to fetch the full description of the products:

```
SELECT name || ' - ' || description
FROM product;
```

Fetch names of products with prices above 15:

```
SELECT name
FROM product
WHERE price > 15;
```

Fetch names of products with prices between 50 and 150:

```
SELECT name
FROM product
WHERE price BETWEEN 50 AND 150;
```

Fetch names of products that are not watches:

```
SELECT name
FROM product
WHERE name != 'watch';
```

Fetch names of products that start with a 'P' or end with an 's':

```
SELECT name
FROM product
WHERE name LIKE 'P%' OR name LIKE '%s';
```

Fetch names of products that start with any letter followed by 'rain' (like 'train' or 'grain'):

```
SELECT name
FROM product
WHERE name LIKE '_rain';
```

Fetch names of products with non-null prices:

```
SELECT name
FROM product
WHERE price IS NOT NULL;
```

GROUP BY

PRODUCT

| name | category |
|---|---|
| Knife | Kitchen |
| Pot | Kitchen |
| Mixer | Kitchen |
| Jeans | Clothing |
| Sneakers | Clothing |
| Leggings | Clothing |
| Smart TV | Electronics |
| Laptop | Electronics |

→

| category | count |
|---|---|
| Kitchen | 3 |
| Clothing | 3 |
| Electronics | 2 |

AGGREGATE FUNCTIONS

Count the number of products:

```
SELECT COUNT(*)
FROM product;
```

Count the number of products with non-null prices:

```
SELECT COUNT(price)
FROM product;
```

Count the number of unique category values:

```
SELECT COUNT(DISTINCT category)
FROM product;
```

Get the lowest and the highest product price:

```
SELECT MIN(price), MAX(price)
FROM product;
```

Find the total price of products for each category:

```
SELECT category, SUM(price)
FROM product
GROUP BY category;
```

Find the average price of products for each category whose average is above 3.0:

```
SELECT category, AVG(price)
FROM product
GROUP BY category
HAVING AVG(price) > 3.0;
```

ORDER BY

Fetch product names sorted by the price column in the default ASCending order:

```
SELECT name
FROM product
ORDER BY price [ASC];
```

Fetch product names sorted by the price column in DESCending order:

```
SELECT name
FROM product
ORDER BY price DESC;
```

COMPUTATIONS

Use +, -, *, / to do basic math. To get the number of seconds in a week:

```
SELECT 60 * 60 * 24 * 7;
-- result: 604800
```

ROUNDING NUMBERS

Round a number to its nearest integer:

```
SELECT ROUND(1234.56789);
-- result: 1235
```

Round a number to two decimal places:

```
SELECT ROUND(AVG(price), 2)
FROM product
WHERE category_id = 21;
-- result: 124.56
```

TROUBLESHOOTING

INTEGER DIVISION

In PostgreSQL and SQL Server, the / operator performs integer division for integer arguments. If you do not see the number of decimal places you expect, it is because you are dividing between two integers. Cast one to decimal:

```
123 / 2 -- result: 61
CAST(123 AS decimal) / 2 -- result: 61.5
```

DIVISION BY 0

To avoid this error, make sure the denominator is not 0. You may use the NULLIF() function to replace 0 with a NULL, which results in a NULL for the entire expression:

```
count / NULLIF(count_all, 0)
```

JOIN

JOIN is used to fetch data from multiple tables. To get the names of products purchased in each order, use:

```
SELECT
 orders.order_date,
 product.name AS product,
 amount
FROM orders
JOIN product
  ON product.id = orders.product_id;
```

Learn more about JOINs in our interactive SQL JOINs course.

INSERT

To insert data into a table, use the INSERT command:

```
INSERT INTO category
VALUES
(1, 'Home and Kitchen'),
(2, 'Clothing and Apparel');
```

You may specify the columns to which the data is added. The remaining columns are filled with predefined default values or NULLs.

```
INSERT INTO category (name)
VALUES ('Electronics');
```

UPDATE

To update the data in a table, use the UPDATE command:

```
UPDATE category
SET
 is_active = true,
 name = 'Office'
WHERE name = 'Ofice';
```

DELETE

To delete data from a table, use the DELETE command:

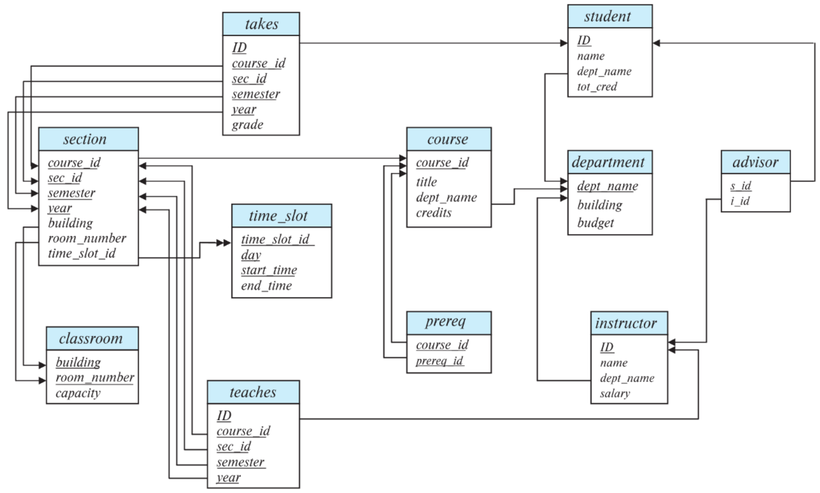
```
DELETE FROM category
WHERE name IS NULL;
```

Check out our interactive course How to INSERT, UPDATE, and DELETE Data in SQL.

## 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 65 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.

**List.All.1:** Create a SQL query for the relational expression:
$\Pi_{\text{name}} (\sigma_{\text{dept\_name} = \text{'Comp. Sci.'}} (\text{ student }))$

Execute 4 points Minimize Output

**Relational.Algebra.1:** Create a SQL query for the relational expression:

$$\Pi_{\text{title}}(\sigma_{\text{semester}='Fall' \wedge \text{dept\_name}='Astronomy'}(\rho_S(\text{Section}) \bowtie_{\text{C.course\_id}=\text{S.course\_id}} \rho_C(\text{Course})))$$
$$-$$
$$\Pi_{\text{title}}(\sigma_{\text{semester}='Winter' \wedge \text{dept\_name}='Astronomy'}(\rho_S(\text{Section}) \bowtie_{\text{C.course\_id}=\text{S.course\_id}} \rho_C(\text{Course})))$$
$$-$$
$$\Pi_{\text{title}}(\sigma_{\text{semester}='Spring' \wedge \text{dept\_name}='Astronomy'}(\rho_S(\text{Section}) \bowtie_{\text{C.course\_id}=\text{S.course\_id}} \rho_C(\text{Course})))$$
$$-$$
$$\Pi_{\text{title}}(\sigma_{\text{semester}='Summer' \wedge \text{dept\_name}='Astronomy'}(\rho_S(\text{Section}) \bowtie_{\text{C.course\_id}=\text{S.course\_id}} \rho_C(\text{Course})))$$

Execute 4 points Minimize Output

**Relational.Algebra.In.Words.1:** Which of the following best represents in words what **Relational.Algebra.1** is asking for?
**A** List the courses that are offered by the Astronomy department Fall semester
**B** List the unique course titles that are offered by the Astronomy department all semesters
**C** List the unique course titles that are offered by the Astronomy department Fall semester but not at other times
**D** List the courses that are offered by the Astronomy department all semesters
**E** List the unique course titles that are offered by the Astronomy department Fall but not Spring, Summer semester(s)
**F** List the unique course titles that are offered by the Astronomy department Fall but not Winter semester(s)
**H** All of the above
**I** None of the above

4 points

**Did.Not.Take.1:** List the name and total number of credits for students in the Comp. Sci. department who did not take the comp. sci. course named 'Game Design'.
Order the list in descending order by the name.

Execute 9 points Minimize Output

**How.Many.Students.1:** Find the number of students in the Comp. Sci. department.

Execute 4 points Minimize Output

**New.Instructor.1:** Add a new instructor named 'Pauli Murray', ID 'pauli', with a salary of 75,000 in the department of History.
**NOTE$_1$:** You may be assured that no other instructor has the ID pauli and so you may hardcode your query using it.
**NOTE$_2$:** With all SQL statements like this that modify the database, you either need to refresh the web page before rerunning or handle the fact that the statement was previously run.

Execute 9 points Minimize Output

**New.Advisor.1:** Make the new instructor from the previous question, Pauli Murray, the advisor to student 'Bates'.

**NOTE$_1$:** You need to get the previous question correct to get this question correct.

**NOTE$_2$:** You can hardcode the advisor id of pauli, but you MUST programmatically determine student Bates's ID. In other words, do not print out Bates's ID and then hardcode it. Programmatically determine it within the SQL code!

**NOTE$_3$:** You can be assured that the student name Bates is unique.

**NOTE$_4$:** With all SQL statements like this that modify the database, you either need to refresh the web page before rerunning or handle the fact that the statement was previously run.

Execute 9 points Minimize Output

**Strings.1:** Select all the student names who have a name with second character of a.

Order the result alphabetically by the student's department name. For all students within each department sort by the student's name.

Execute 4 points Minimize Output

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

Execute 9 points Minimize Output

**Student.Takes.All.1:** List the names of all the students who have taken all the courses in the Cybernetics department.

Execute 9 points Minimize Output

-

## Joins 20 points

In this section, you will show your knowledge of the different join types.

Relation instance A. Attribute names are in uppercase, **bold**, and blue in the first row. Tuples are in the rows that follow.

| A1 | A2 | A3 | A4 |
|---|---|---|---|
| a11 | a21 | a31 | a41 |
| a12 | a22 | a32 | a42 |
| a13 | a23 | a33 | a43 |

Relation instance B. Attribute names are in uppercase, **bold**, and blue in the first row. Tuples are in the rows that follow.

| B1 | B2 | B3 | B4 | B5 | B6 |
|---|---|---|---|---|---|
| b11 | b21 | b31 | a41 | b51 | b61 |
| b12 | b22 | b32 | a42 | b52 | b62 |

**Cartesian.Product.1:** Given the relational instances above, what is the result of the following join?

```
SELECT DISTINCT A2, B2, B5
  FROM A, B
 ORDER BY A2, B2, B5
```

**NOTE:** If after completing the rows returned from the query and there are rows remaining in the table, entry ROWUNUSED for the unused rows.

| A2 | B2 | B5 |
|---|---|---|
| a21 | b21 | b51 |
| a21 | b22 | b52 |
| a22 | b21 | b51 |
| a22 | b22 | b52 |
| a23 | b21 | b51 |
| a23 | b22 | b52 |

5 points

---

**Inner.Join.1:** Given the relational instances above, what is the result of the following join?

```
SELECT DISTINCT A2, B2, B5
  FROM A, B
 WHERE A4 = B4
 ORDER BY A2, B2, B5
```

**NOTE:** If after completing the rows returned from the query and there are rows remaining in the table, entry ROWUNUSED for the unused rows.

| A2 | B2 | B5 |
|---|---|---|
| a21 | b21 | b51 |
| a22 | b22 | b52 |
| ROWUNUSED | ROWUNUSED | ROWUNUSED |
| ROWUNUSED | ROWUNUSED | ROWUNUSED |
| ROWUNUSED | ROWUNUSED | ROWUNUSED |
| ROWUNUSED | ROWUNUSED | ROWUNUSED |

5 points

---

**Left.Join.1:** Given the relational instances above, what is the result of the following join?

```
 SELECT DISTINCT A2, B2, B5
   FROM A LEFT JOIN B
                 ON A4 = B4
  ORDER BY A2, B2, B5
```

**NOTE:** If after completing the rows returned from the query and there are rows remaining in the table, entry ROWUNUSED for the unused rows.

| A2 | B2 | B5 |
|---|---|---|
| a21 | b21 | b51 |
| a22 | b22 | b52 |
| a23 | NULL | NULL |
| ROWUNUSED | ROWUNUSED | ROWUNUSED |
| ROWUNUSED | ROWUNUSED | ROWUNUSED |
| ROWUNUSED | ROWUNUSED | ROWUNUSED |

5 points

**Full.Outer.Join.1:** Given the relational instances above, what is the result of the following join?

```
 SELECT DISTINCT A2, B2, B5
   FROM A FULL OUTER JOIN B
                 ON A4 = B4
  ORDER BY A2, B2, B5
```

**NOTE:** If after completing the rows returned from the query and there are rows remaining in the table, entry ROWUNUSED for the unused rows.

| A2 | B2 | B5 |
|---|---|---|
| a21 | b21 | b51 |
| a22 | b22 | b52 |
| a23 | NULL | NULL |
| ROWUNUSED | ROWUNUSED | ROWUNUSED |
| ROWUNUSED | ROWUNUSED | ROWUNUSED |
| ROWUNUSED | ROWUNUSED | ROWUNUSED |

5 points

## Chapter Reading Review 15 points

**Databases.Are.1:** Select the letter of the statement below that is false.
**A** Database systems are designed to store large bodies of information
**B** The relational data model is the only model used for storing data in databases
**C** The management of data involves both the definition of structures for the storage of information and the provision of mechanisms for the manipulation of information
**D** The database system must provide for the safety of the information stored in the face of system crashes or attempts at unauthorized access
**E** If data is shared among users, the database system must avoid possible anomalous results

▾ 1 point

---

**Data.Manipulation.Language.1:** Select the letter of the data manipulation language statement below that is false or select **E** if none of them are false.
**A** The data manipulation language is used to specify the database schema.
**B** The data manipulation language enables users to access data.
**C** The INSERT statement is a data manipulation language statement.
**D** The data manipulation language is widely used in retrieving data from relational databases.
**E** None of the above are false.

▾ 1 point

---

For the following fill-in-the-blank-questions match the letter for the phrase that best answers the following questions.

**A** A collection of iterrelated data and the programs to access the data
**B** Edgar "Ted" Codd
**C** Data documentation language
**D** A collection of data
**E** Standard query language
**F** Fred Brooks
**G** Data manipulation language
**H** None of these

**Word.Definition.1:** Letter representing the best answer to the question:

| Question | Letter of definition |
|---|---|

| | |
|---|---|
| What is a database? | |
| What is a database management system? | |
| Who proposed and later won a Turing award for the relational database model? | |
| What is an example of a database language? | |

4 points

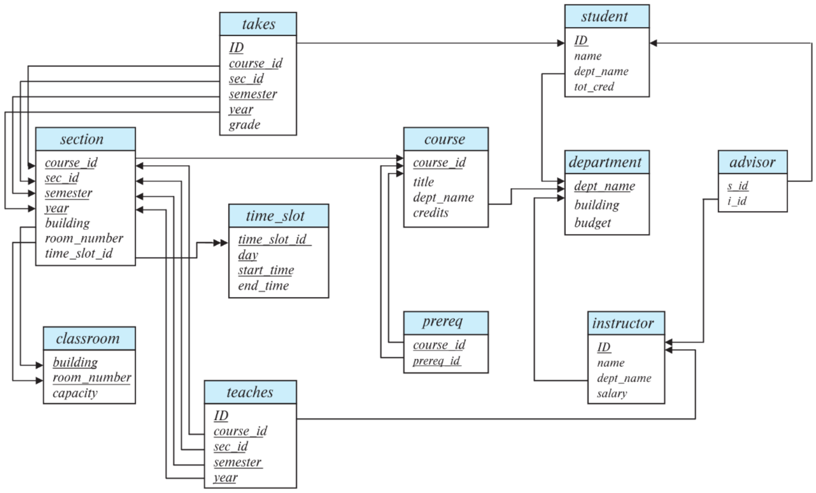
Each of the items in below describe a problem solved by a database. The table contains terms of features of a database. For each database feature, select the letter of the problem that it solves.

**A** Larger amounts of storage and the cost to access data in various formats
**B** Confusion of physical versus logical layer
**C** Efficient and convenient access to data
**D** All portions or no portions of a logical unit of work completes. The all-or-none requirement
**E** Physical layer data abstraction
**F** University payroll personnel should see financial information but not academic or grade records
**G** Multiple data updates to a field that occur near simultaneously to each updater without inconsistent results
**H** None of these

**DB.Solves.1** For each database feature below, select the letter of the problem that it solves.

| Database feature | Problem solved |
|---|---|
| Atomicity | |
| Security | |
| Concurrency | |

4 points



Following are relational algebra statements:

Note: During the test, this question incorrectly had $\sigma$ where it should have had $\Pi$.

**A** $\Pi_{\text{course\_id, title}}(Teaches \bowtie \text{Course})$

**B** $(\Pi_{\text{course\_id, title}} Teaches) \bowtie (\Pi_{\text{dept\_name, title}} \text{Course})$

**C** $\Pi_{\text{course\_id, title}}(Teaches \bowtie_{\text{Teaches.course\_id=Course.course\_id}} \text{Course})$

**D** $\Pi_{\text{course\_id, title}}((\Pi_{\text{course\_id,year}} Teaches) \bowtie_{\text{Teaches.course\_id=Course.course\_id}} \text{Course})$

**E** $\Pi_{\text{course\_id, title}}(Teaches \bowtie_{\text{Teaches.course\_id=Course.course\_id}} (\Pi_{\text{coures\_id,title}} \text{Course}))$

**F** All of the above relational algebra produces the same results

**G** There are more than two different results

**Teachers.Teach.1:** Which of the above relational algebra statements produce different results than all the others?

Select **F** or **G**, respectively, if they all produce the same results or more than two different results.

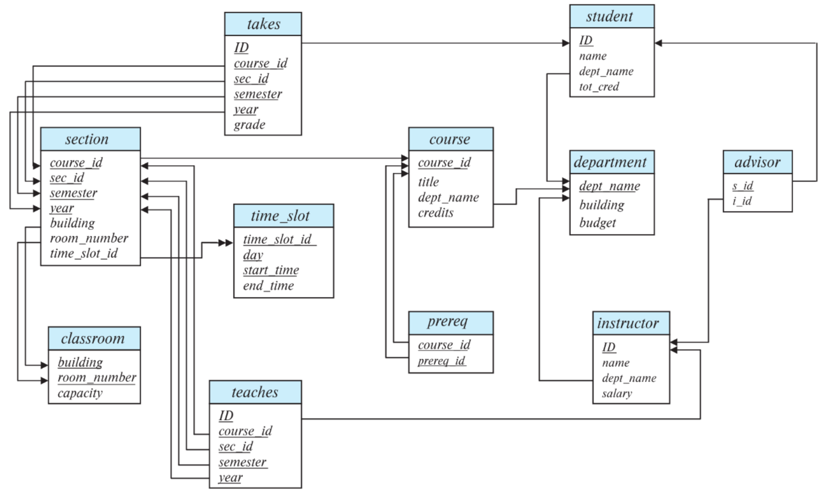
3 points

---

**Relational.Algebra.Operation.1:** What relational algebra operation is represented by the Greek letter sigma, $\sigma$?

1 point

---



**Reading.Schema.1:** Given the schema diagram above, can a department be in two different buildings?

1 point

---

## Pledge your worksheet

**DO NOT PUT YOUR NAME ON THE WORKSHEET UNTIL YOU ARE READY TO SUBMIT IT.**
You will be given grace on any two assignments that you want to resubmit. But you should consider that only the first submission of a worksheet or any assignment will be graded.

**UNC Honor Pledge:** I certify that no unauthorized assistance has been received or given in the completion of this work. This unauthorized assistance includes, but is not limited to, copying of another student's answers.

Enter your full name

If other students helped in the completion of this worksheet, give attribution to them by entering their Onyen below. Help can be in the form of a work group, online conversation, etc. It must **not** include copying answers. If no other student helped with the completion of this worksheet, **None** should be entered.

Enter the Onyen of any students who assisted in this worksheet's cc

## Done!

## Submit your work

Did you **pledge your work** above?

Did you **acknowledge any student assistance** above?

Submit