Topic Overview

While content for this test will be focused on the newer material, everything is building in this class, so still expect to see some concepts from the first test present. Major new topics though include:

- ☐ Chapter 6
  - ☐ Using joins to get information from multiple tables
  - ☐ Understanding the different types of joins
  - ☐ Utilizing self joins and understanding when they should be used
- ☐ Chapter 7
  - ☐ Understanding when double quotes are needed in identifiers
  - ☐ Creating primary and foreign keys
  - ☐ Creating other constraints with conditionals, null checks, or uniqueness checks
  - ☐ Understanding when and why you might define an index on a column
  - ☐ Creating an index on a column
- ☐ Chapter 8
  - ☐ Understanding the `GROUP BY` keywords
    - How to select entries from a grouped table
    - When you would want to use `GROUP BY`
  - ☐ Understanding the `HAVING` keyword
- ☐ Chapter 9
  - ☐ Modifying tables using `ALTER TABLE`
    - Adding columns
    - Changing column types
    - Adding primary or foreign keys
    - Removing columns
    - Renaming columns
  - ☐ Modifying table contents using `UPDATE`
  - ☐ Deleting content from tables

Question Types

The style of the test questions will follow that of Test 1 fairly closely, with a mix of:

Qualitative: In general, these wouldn't deal with direct values in a table, but are more conceptual in understanding what a particular piece of SQL is doing.

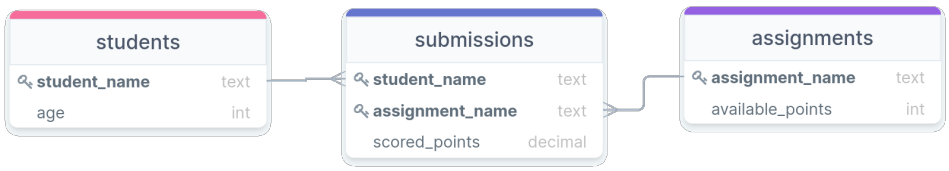
- Given a general table and query, describe what the output would look like, or what properties it might have.
- Given a desired output, what properties might the query or initial table have needed to possess?
- Given a table and desired output, what would the query need to look like?

Quantitative: These will deal more directly with sample data in a table.

- For this particular query with this tabular data, what would the output be? (These will naturally be with small and simple tables, as you won't have a computer to aid you.)

Example Questions

1. Considering the below table layout, write out queries to accomplish each of the following tasks.



(a) A table showing just names of students that are the same age as another student. They don't need to be paired up. Just a list of all the names that are the same age as someone else in the class.

Comparing ages, so self-join

```
SELECT s1.student_name
FROM students as s1
JOIN students as s2
  ON s1.age = s2.age AND s1.student_name != s2.student_name;
```

to prevent matching self
no other self-matches possible b/c names are unique b/c primary key

(b) A table showing the student and assignment name for all assignments in which a student scored a 100%.

All the JOINs

in retrospect, this is unnecessary b/c submissions already has the student names

```
SELECT st.student_name, a.assignment_name
FROM students as st
JOIN submissions as sub
  ON st.student_name = sub.student_name
JOIN assignments as a
  ON a.assignment_name = sub.assignment_name
WHERE a.available_points = sub.scored_points
```

(c) A table containing only students with missing assignments which lists the student names and missing assignments.

```
SELECT s.student_name, a.assignment_name
FROM students as st
CROSS JOIN assignments as a
LEFT JOIN submissions as s
  ON st.student_name = s.student_name AND
     a.assignment_name = s.assignment_name
WHERE s.scored_points IS NULL
```

} generates table of all student & assignment combos

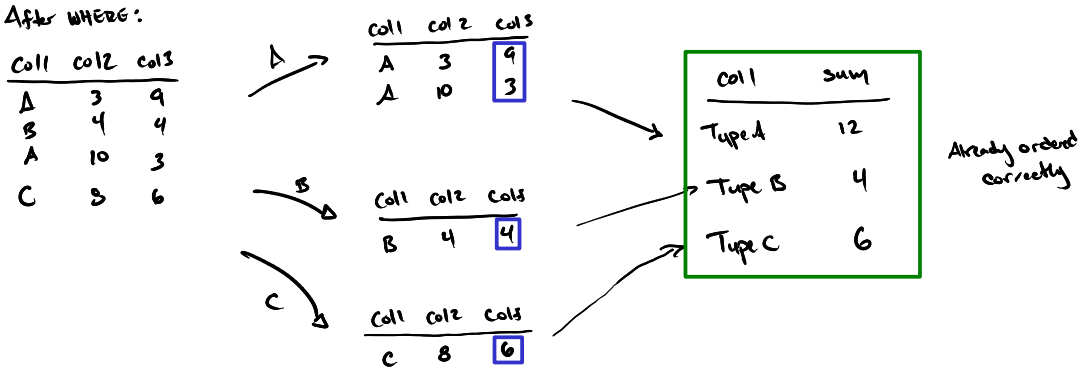
→ left join ensures that combos c/o a submission still remain in the table & NULL

2. Using the below table (named `tab`), determine by hand what the output of the following queries would be.

col1 *text*	col2 *int*	col3 *int*
Type A	3	9
Type B	4	4
Type A	10	3
Type A	1	2
Type B	2	8
Type C	8	6

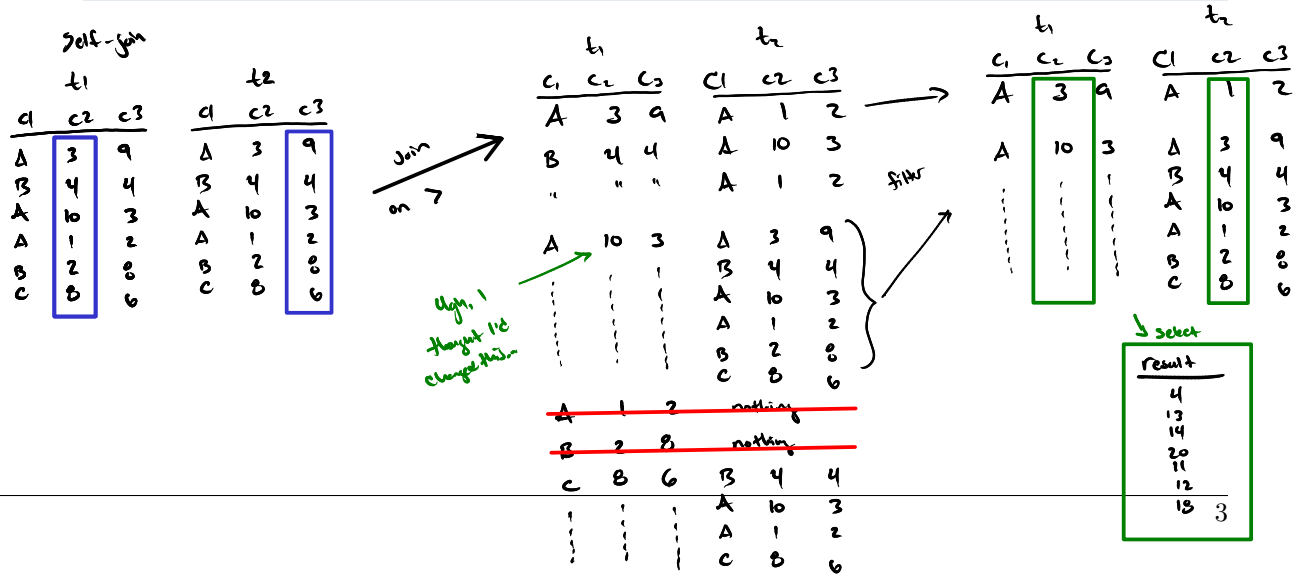
(a) What would be the output of the following query?

```
SELECT col1, SUM(col3)
FROM tab
WHERE col2 > 2
GROUP BY col1
ORDER BY col1;
```



(b) What would be the output of the following query?

```
SELECT t1.col2 + t2.col2 as result
FROM tab as t1
JOIN tab as t2
  ON t1.col2 > t2.col3
WHERE t1.col1 = 'Type A';
```



(c) What would be the final output after running the below sequence of commands?

```
① ALTER TABLE tab ADD COLUMN col4 INT;

  UPDATE tab
② SET col4 = col2 - col3
  WHERE col3 > 5;

  DELETE FROM tab
③ WHERE col4 IS NULL;

  SELECT col1, -col2*col4 as tot
④ FROM tab;
```

After ①

c1	c2	c3	c4
A	3	9	NULL
B	4	4	NULL
A	10	3	NULL
A	1	2	NULL
B	2	8	NULL
C	8	6	NULL

After ②

c1	c2	c3	c4
A	3	9	-6
B	4	4	NULL
A	10	3	NULL
A	1	2	NULL
B	2	8	-6
C	8	6	2

After ③

c1	c2	c3	c4
A	3	9	-6
B	2	8	-6
C	8	6	2

After ④

col1	tot
A	18
B	12
C	-16

3. Considering the following table definitions and initial contents, would the below SQL commands result in an error? If so, specifically what is causing the error?

```
CREATE TABLE tab1 (
  col1 INT PRIMARY KEY,
  col2 TEXT NOT NULL,
  col3 INT


);
```

col1	col2	col3
1	red	2
2	blue	NULL
4	green	6
5	green	6
6	red	10

```
CREATE TABLE tab2 (
  col1 INT,
  col2 INT,
  col3 DATE NOT NULL,
  col4 INT CHECK (col4 > 0),
  PRIMARY KEY (col1, col2)
);
```

col1	col2	col3	col4
1	5	2022-10-02	1
2	4	2021-08-01	2
3	3	2022-03-01	10
4	2	2022-01-06	3
5	1	2022-04-01	4

```
CREATE TABLE tab3 (
  col1 SERIAL PRIMARY KEY,
  col2 INT,
  col3 INT,
  col4 INT REFERENCES tab1,
  col5 INT,
  FOREIGN KEY (col2, col3) REFERENCES tab2,
  CHECK (col5 > col4)
);
```

col1	col2	col3	col4	col5
1	1	5	1	5
2	3	3	1	6
3	4	2	4	5
4	4	2	5	10
5	5	1	2	8

(a)
```
INSERT INTO tab3 (col2, col3, col4, col5) VALUES
  (2,4,2,4);
```

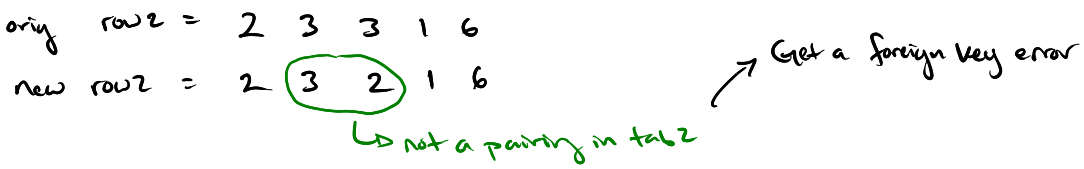
(2,4) is in tab2 ✓
2 is in tab1 ✓ This would work.
4>2 ✓

(b)
```
DELETE FROM tab1
WHERE col1 = 6;
```

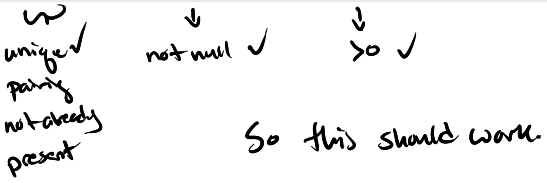
There is no row in tab3 col4 w/ a value of 6, so this is safe to delete.

This would work!

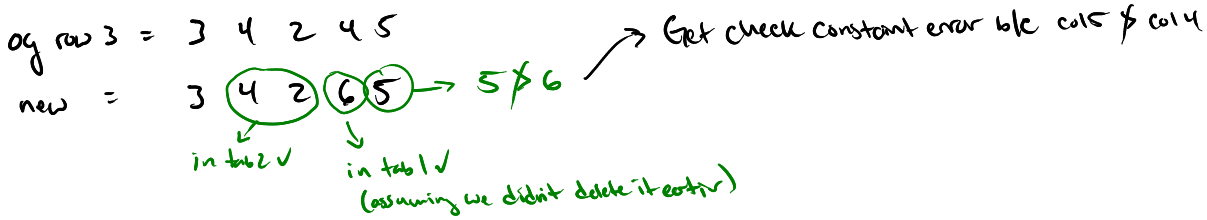
(c)
```
UPDATE tab3
SET col3 = 2
WHERE col1 = 2;
```



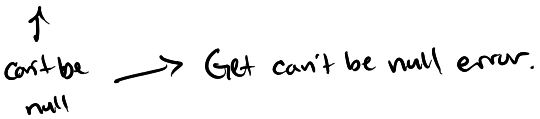
(d)
```
INSERT INTO tab2 VALUES
  (6,6, '2022-04-01', 3);
```



(e)
```
UPDATE tab3
SET col4 = 6
WHERE col1 = 3;
```



(f)
```
INSERT INTO tab1 VALUES
  (3, NULL, NULL);
```



(g)
```
UPDATE tab2
SET col4 = col1 - col2
WHERE col3 >= '2022-05-10'
```