Homework Assignment 5
Due: 11:59PM JUNE 9, 2023
Shinhoo Kim. 21900136@handong.ac.kr

1. (3 pt. each) Join operations.

(a) (Exercise 4.1) Consider the following SQL query that seeks to find a list of titles of all courses taught in Spring 2017 along with the name of the instructor.

instructor **NATURAL JOIN** *teaches* **NATURAL JOIN** *section* will be a desirable result. But when they are getting joined by **NATURAL JOIN** *course,* it won't be a desirable result since they need to be joined by course_id only but they will be also joined by dept_name which is from instructor table. So, applying **NATURAL JOIN** without an enough consideration is wrong with that query.

(b) (Exercise 4.16) Write an SQL query using the university schema to find the ID of each student who has never taken a course at the university. Do this using no subqueries and no set operations (use an outer join).

SELECT *ID* **FROM** *student* **NATURAL LEFT OUTER JOIN** *takes* **WHERE** *course_id* **IS NULL**;

(c) (Exercise 4.17) Express the following query in SQL using no subqueries and no set operations.

SELECT *ID*
FROM *student as s* **LEFT OUTER JOIN** *advisor as a* **ON** *s.s_id = a.s_id*
WHERE *a.i_ID* **IS NULL OR** *a.s_ID* **IS NULL**;

(d) (Exercise 4.20) Show how to define a view *tot_credits*(*year, num_credits*), giving the total number of credits taken in each year.

CREATE VIEW *tot_credits(year, num_credits)* **AS**
SELECT *year,* **SUM**(*credits)*
FROM *takes* **NATURAL JOIN** *course*
GROUP BY *year;*

(e) (Exercise 4.21) For the view that you have defined in the previous problem (Problem 1(d)), explain why the database system would not allow a tuple to be inserted into the database through this view.

Because the view is not simple. The view is not materialized view. When you attempt to update view, it does not mean you update the view but update the origin of view through the view. So only when a view is simple, it is updatable. And the view is not updatable because it is from two tables and select has an aggregation name which is **SUM**(credits) and lastly it is **GROUP BY** year.

2. Answer the following questions that are from the textbook exercise problem sets. You may refer to the Internet as well as the textbook for assistance; however, your solution should contain your own ideas in your own language.

(a) (3 pt.; Exercise 5.9) Given a relation *nyse*(*year, month, day, shares_traded, dollar_volume*) with trading data from the New York Stock Exchange, list each trading day in order of number of shared traded, and show each day's rank.

```
SELECT year,month,day,shares_traded, RANK() OVER (ORDER BY (shares_traded))
FROM nyse;
```

(b) (3 pt.; Exercise 5.23) Consider the relation from Problem 2(a). For each month of each year, show the total monthly dollar volume and the average monthly dollar volume for that month and the two prior months.
(You may want to use the hint suggested by the textbook.)

```
WITH montly_volumn(year,month,monthly_volume) AS (
   SELECT year,month,SUM(dollar_volume)
   FROM nyse
   GROUP BY year, month
) SELECT year, month, monthly_volume,
   AVG(monthly_volume) OVER (ORDER BY (year,month) ROWS BETWEEN 2 PRECEDING AND
CURRENT ROW)
FROM montly_volumn;
```

(c) (3 pt.; Exercise 5.8) Given a relation *S*(*student, subject, marks*), write a query to find the top 10 students by total marks, by using SQL ranking. Include all students tied for the final spot in the ranking, even if that results in more than 10 total students.

```
SELECT *
FROM
(WITH sub (student, sum_marks) AS (
  SELECT student, SUM(marks) AS sum_marks
  FROM S
  GROUP BY student
) SELECT student, RANK() OVER(ORDER BY sum_marks DESC) as rank
FROM sub)
WHERE rank <= 10;
```

(d) (4 pt.; Exercise 5.6) Consider the bank database of Figure 5.21. Let us define a view *branch_cust* as follows:

참고한 reference: https://docs.oracle.com/javadb/10.6.2.1/ref/rrefsqlj43125.html

```
CREATE TRIGGER add_on_account_listener
AFTER INSERT ON account
REFERENCING NEW ROW AS new_record
FOR EACH STATEMENT
  INSERT INTO branch_cust
  SELECT new_record.branch_name, d.customer_name
  FROM depositor AS d
  WHERE d.account_number = new_record.account_number;
```

```
CREATE TRIGGER add_on_depositor_listener
AFTER INSERT ON depositor
REFERENCING NEW ROW AS new_record
FOR EACH STATEMENT
  INSERT INTO branch_cust
  SELECT a.branch_name, new_record.customer_name
  FROM account AS a
  WHERE new_record.account_number = account.account_number;
```

(e) (3 pt.; Exercise 5.7) Consider the bank database of Figure 5.21. Write an SQL trigger to carry out the following action: On **DELETE** of an account, for each customer-owner of the account, check if the owner has any remaining accounts, and if she does not, delete her from the depositor relation.

```
CREATE TRIGGER delete_on_account_listener
AFTER DELETE ON account
REFERENCING OLD ROW AS old_record
FOR EACH STATEMENT
DELETE FROM depositor
WHERE depositor.customer_name IN
(
SELECT depositor.customer_name
FROM depositor
WHERE depositor.account_number = old_record.account_number
);
```

(f) (Exercise 17.8) The *lost update* anomaly is said to occur if a transaction T_j reads a data item, then another transaction T_k writes the data item (possibly based on a previous read), after which T_j writes the data item. The update performed by T_k has been lost, since the update done by T_j ignored the value written by T_k.

a. (3 pt.) Give an example of a schedule shown the lost update anomaly.

In the situation that two persons are sending money to the same account at the same time, let say there are two transactions T1 and T2 and account number A1 and balance of it is b1($100). T1 reads the balance of A1 which is 100 and added 50. But before T1 update the data to the database, T2 reads the balance which is still 100 and subtract 50. After T1 finishes updating the balance, T2 update the balance to 50. Then T1's work gets lost.

b. (3 pt.) Give an example schedule to show that the lost update anomaly is possible with the **read committed** isolation level.

Let say there are two transactions accessing to the same data. T1 accesses data and update it but before T1 commits, T2 reads the original data but not updated data by T1 because it is before T1 commits it. If T2 updates the data based on what it reads earlier, then T1's work gets lost.

c. (3 pt.) Explain why the lost update anomaly is not possible with the **repeatable read** isolation level.

In repeatable read, the lost update anomaly is not possible because as soon as the data has changed by one transaction, the data that the other transactions see will be syncronized.

3. (10 pt.) Consider the following timelines where two transactions are intervening each other. The two vertical downward arrows represent the progression of time. The horizontal arrows represent the data flow between transaction and storage.

	REPEATABLE READ	READ COMMITTED	READ UNCOMMITTED
a	NULL	NULL	NULL
b	NULL	NULL	Bob
c	NULL	NULL	NULL

	REPEATABLE READ	READ COMMITTED	READ UNCOMMITTED
d	Jane	Jane	Jane
e	Jane	Jane	Mary
f	Jane	Mary	Amy
g	Jane	Mary	Mary

4. Views, procedures, and functions. Consider the following relations:

(a) (3 pt.) Assume that you have a view that has been created using the following query:
What is the result of the next query?

name	funding
Moderna	2700000000
Metronic	367000000

(b) (3 pt.) Consider a stored procedure given below:

CALL count_cpn_proc('Genetics', @tmp);

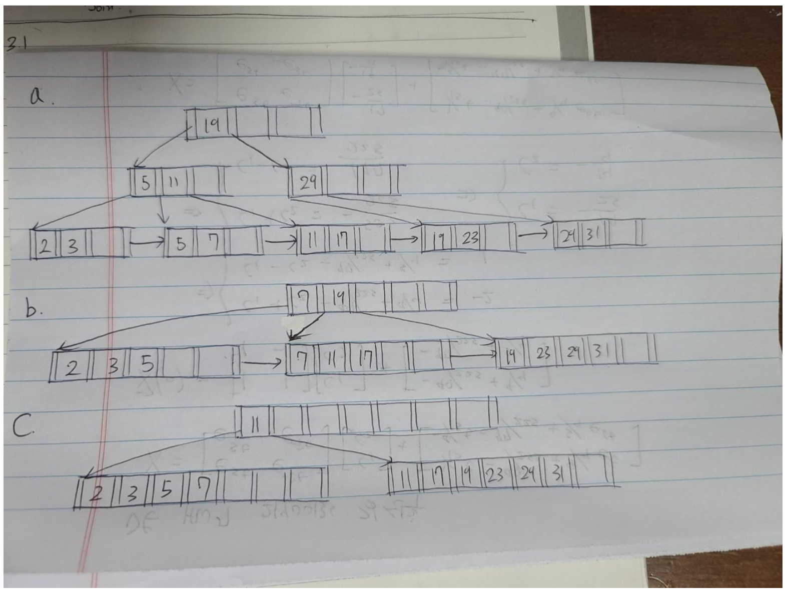
SELECT @tmp;

(c) (3 pt.) Assume that you have given a function defined as below:

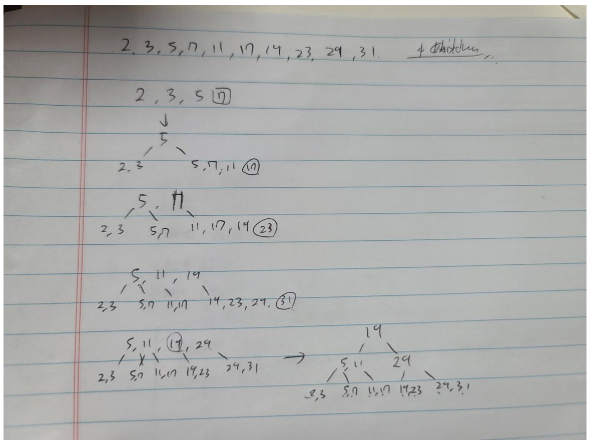
name
Moderna
Pacific Biosciences
Metronic

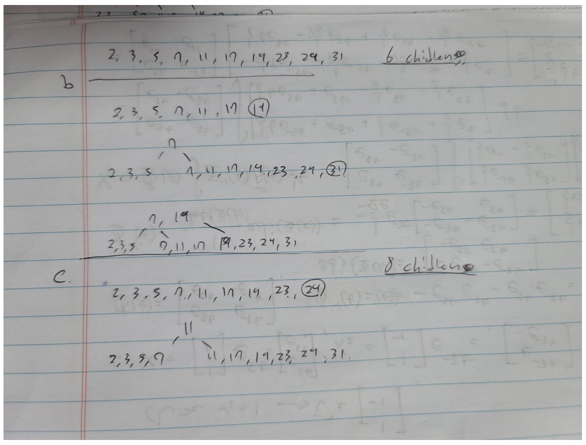
5. Indexes. Answer the following questions that are from the textbook exercise problem sets. You may refer to the Internet as well as the textbook for assistance; however, your solution should contain your own ideas in your own language.

(a) (4 pt. each; Exercise 14.3) Construct a B+tree for the following set of key values:



<풀이과정>





(b) (2 pt. each; Exercise 14.18) For each B+tree of Exercise 14.3a (not b and c), show the steps involved in the following queries:

a. Find records with a search-key value of 11.

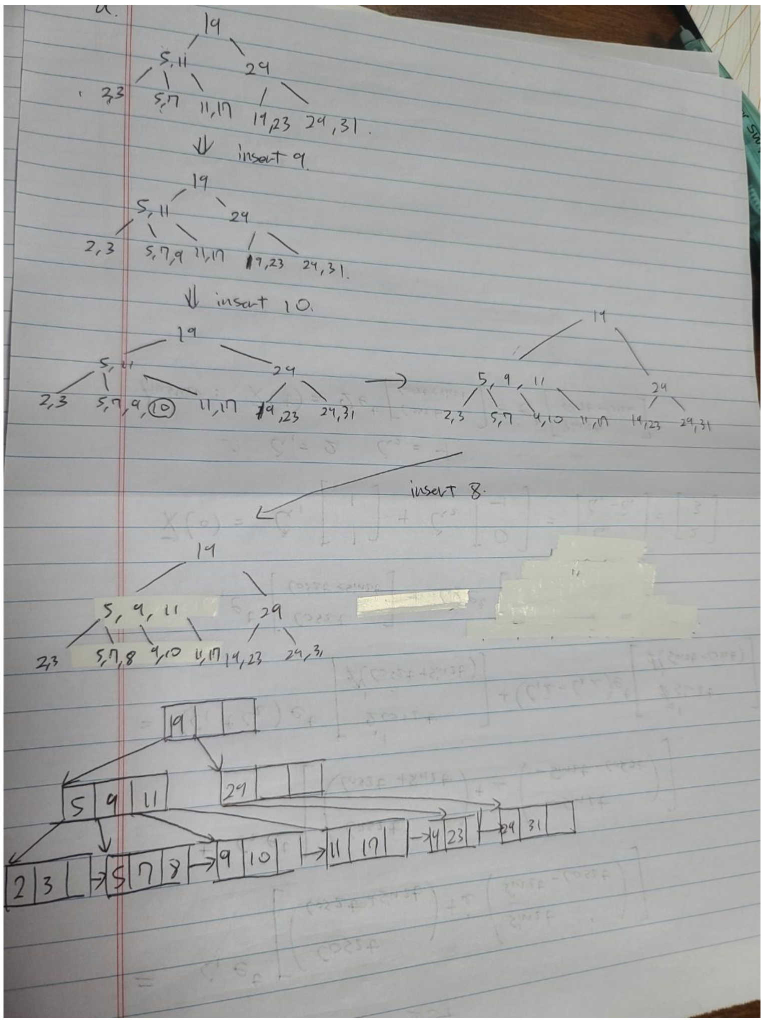
Step1. Look up the root node which is 19 and found out 11 is less than 19. So make a decision to move to left child of node 19.
Step2. Search if there is 11 in the node and found out 11 at the second node. Since the left of the node 11 should be less than 11, move to the right child of node 11.
Step3. Found the record that contains a search-key value of 11 by traveling the leaf node from the first pointer of it.

b. Find records with a search-key value between 7 and 17, inclusive.

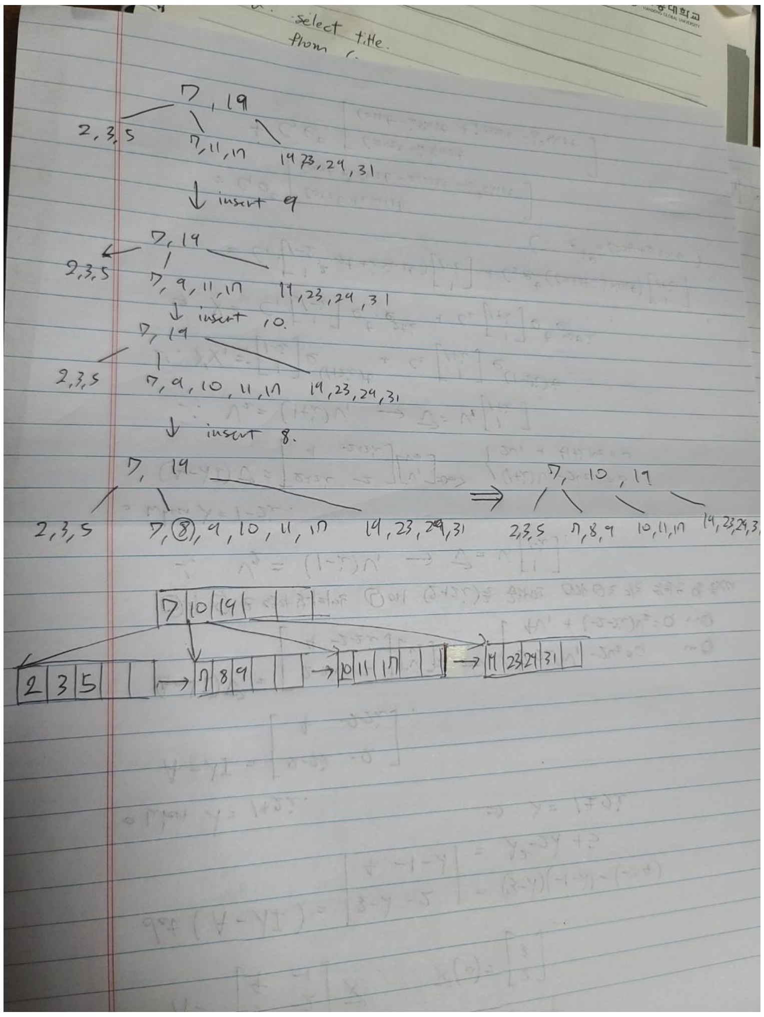
Step1. Look up the root node which is 19 and found out 7 is less than 19. So make a decision to move to the left child of node 19.
Step2. Found out the child node contains 5 and 11 which means the right child node of 5 will have values greater than 5 and less than 11. So make a decision to move to the right child of node 5.
Step3. Reached to the left node and travel from the first pointer of it.
Step4. Found 5 -> just pass it. Found 7 -> store it. Move to the next leaf Node. Found 11 -> store it. Found 17 -> store it. Move to the next leaf Node. Found 19 -> leave it and ends finding process.

c. (5 pt. each; Exercise 14.4) For each B+tree of Exercise 14.3, show the form of the tree after each of the following series of operations:

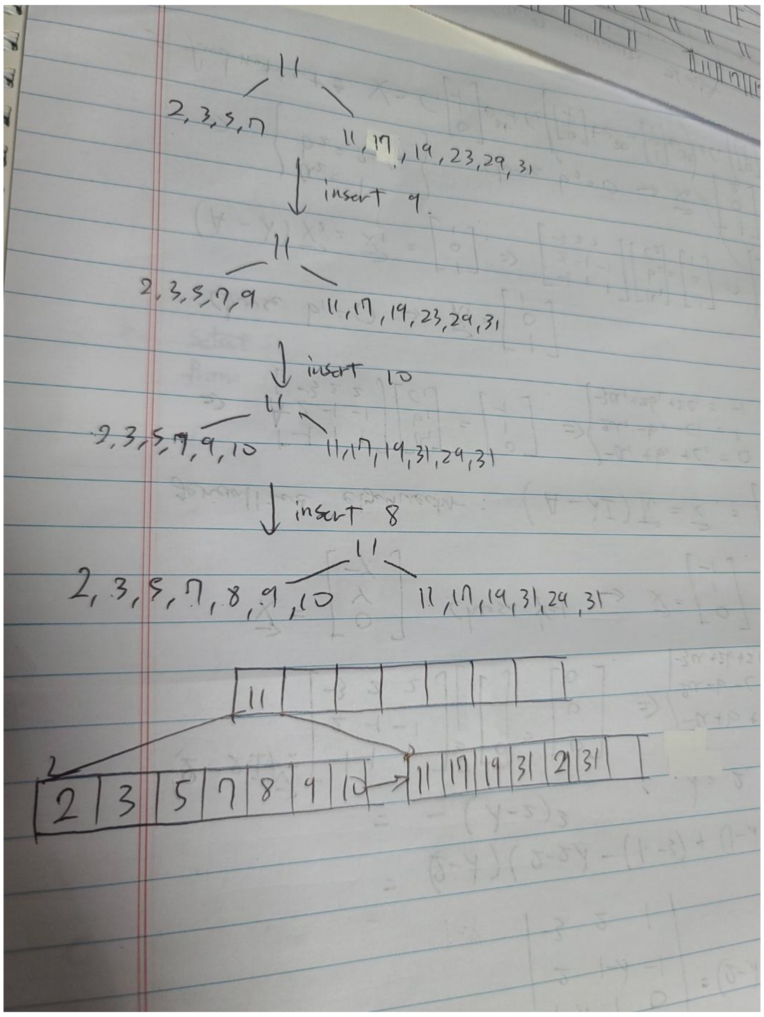
<b+ tree with 3 keys>



<b+ tree with 5 keys>


7, 19
2, 3, 5
7, 11, 17
19, 23, 24, 31
insert 9
7, 19
2, 3, 5
7, 9, 11, 17
19, 23, 24, 31
insert 10.
7, 19
2, 3, 5
7, 9, 10, 11, 17
19, 23, 24, 31
insert 8.
7, 19
2, 3, 5
7, 8, 9, 10, 11, 17
19, 23, 24, 31
7, 10, 19
2, 3, 5
7, 8, 9
10, 11, 17
19, 23, 24, 31
7 10 19
2 3 5
7 8 9
10 11 17
19 23 24 31

<b+ tree with 7 keys>


11
2,3,5,7
11,17,19,23,29,31
insert 9
11
2,3,5,7,9
11,17,19,23,29,31
insert 10
11
2,3,5,7,9,10
11,17,19,31,29,31
insert 8
11
2,3,5,7,8,9,10
11,17,19,31,29,31
11
2
2 3 5 7 8 9 10
11 17 19 31 29 31