Task 1: [5pts]
SSDs can be used as a storage layer between memory and magnetic disks, with some parts of the database (e.g., some relations) stored on SSDs and the rest on magnetic disks. Alternatively, SSDs can be used as a buffer or cache for magnetic disks; frequently used blocks would reside on the SSD layer, while infrequently used blocks would reside on magnetic disk.

A [2.5pts] Which of the two alternatives would you choose if you need to support real-time queries that must be answered within a guaranteed short period of time? Explain why.

I would choose using SSDs as a buffer or cache for magnetic disks because it takes advantage of the speed of SSDs to accelerate access to frequently used blocks, thereby improving the overall query performance. Additionally, real-time queries often involve frequent reads and writes to the database, and SSDs excel in handling random I/O operations, making them well-suited for accelerating database access

B[2.5pts] Which of the two alternatives would you choose if you had a very large customer relation, where only some disk blocks of the relation are accessed frequently, with other blocks rarely accessed.

In this case, the most suitable option would be to store the entire relation on magnetic disks and utilize SSDs as a storage layer between memory and magnetic disks because as the customer relation continues to grow, it's easier to scale the storage infrastructure by adding more magnetic disks to accommodate the increasing data volume. Additionally, storing the entire relation on magnetic disks is a lot more cost-effective, and it is a lot more efficient to allocate the SSDs for caching frequently accessed blocks.

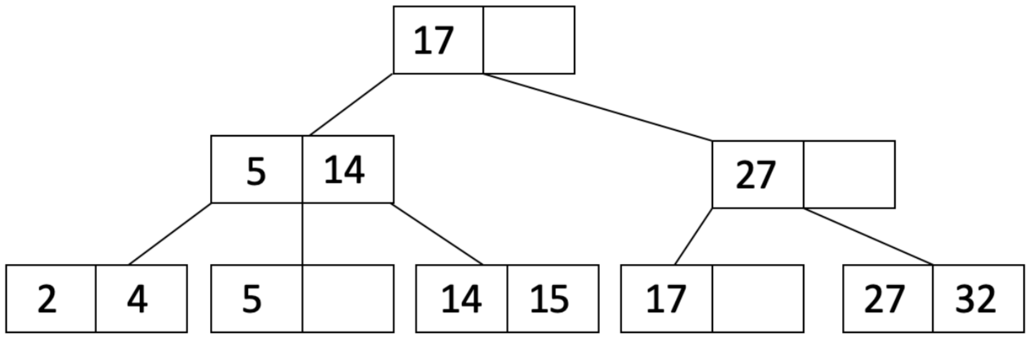
Task 2: [5pts]. Is it possible in general to have two clustering indices on the same relation for different search keys? Explain your answer.

Yes, it is possible to have two clustering indices on the same relation for different search keys. For instance, if a database requires different search keys to efficiently access the same relation, it's possible to create multiple clustering indices, each tailored to a different search key.

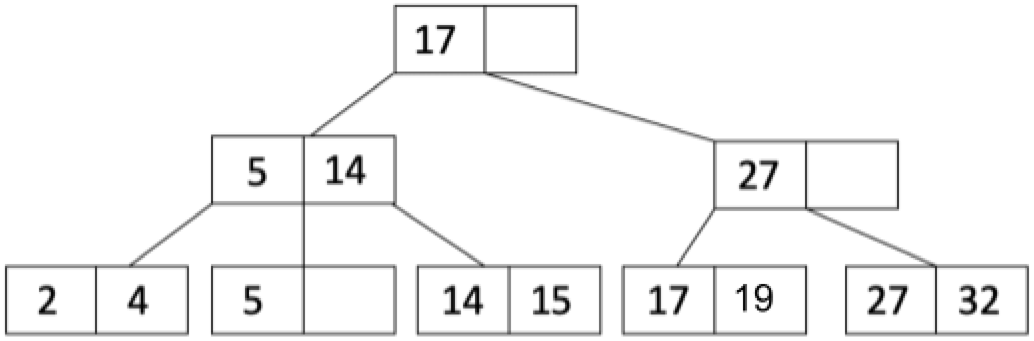
Task 3 [15 pts]
Consider a B+ tree index with the following properties:

- The order of B+ tree is d means that every node except the root node must have x entries between d and 2d i.e. d <= x <= 2d.
- Each node has entries in sorted order.
- Inner nodes (non-leaf) nodes are called routing nodes. Every routing node has a pointer to child node(s). Since every node can have at most 2d entries, a routing node may have at most 2d+1 pointer to child nodes. This is called trees' fanout.

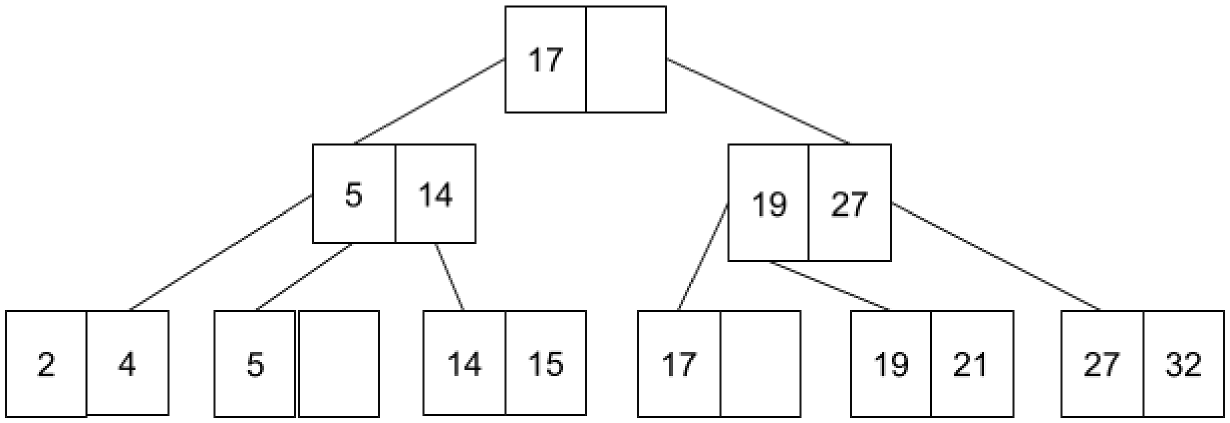
Consider the following tree with d=1



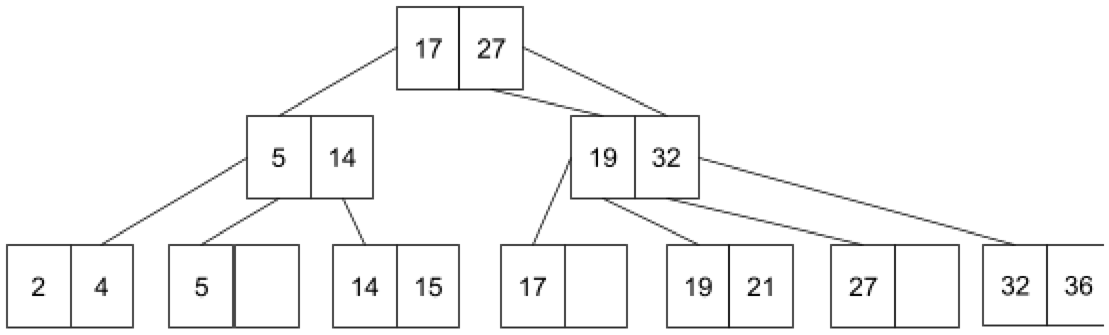
[5pts] First Insert another key 19 to this tree.



[5pts] After 19, insert 21 to the tree.

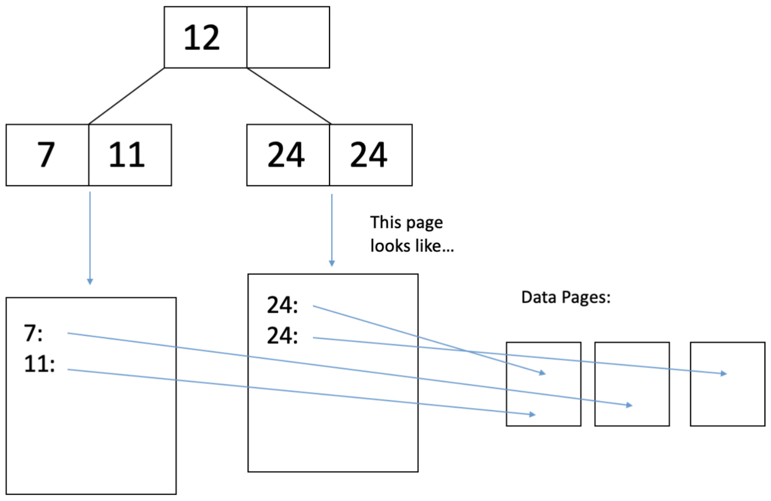


[5pts] Insert 36 to the tree resulted after inserting 21.



Task 4 [10 pts]

Consider the following B+ tree index where leaf nodes of the tree are pointing to data pages on disk. Accessing every index tree node (reading or writing) accounts for 1 I/O operation.
For example: reading 7 from disk will involve 3 reads(one read each for reading root, leaf node and read from disk).
If we want to delete 11 from index and record on disk, how many I/O will it take? Note: this require reading and also updating the index and data pages as well.



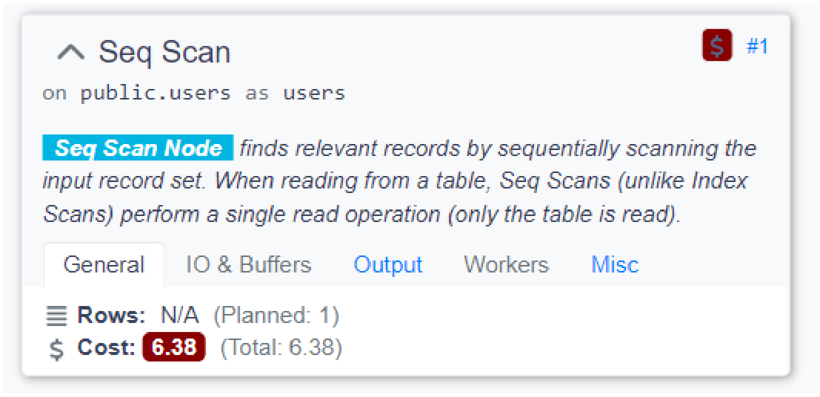
3 for reading 11 from disk (reading root, leaf node, and read from disk)
1 for writing the modified data page back to the disk
1 for writing the modified leaf node back to disk after deleting 11 key
Total: 5 I/O

Task 5 [15 pts]

A physical query plan is like a routine (or a relational algebra expression) that the DBMS follows to assemble the requested query results from the underlying base tables. Different queries will have different physical plans. In fact, the same query may be translated into different physical plans depending on the physical database design. Postgres SQL provides an ***'EXPLAIN'*** function, as shown in the figure below, to help you to check the query plan of your query. You can use this function to examine how your query will be executed internally, what indexes are being used, and the total cost of your query.

start down the path of exploring physical database designs (indexing) and query plans, start by checking the queries plans for each of the following queries without any indexes using the EXPLAIN function Report the query plan of each query by taking snapshots and pasting them(Just take a snapshot of the query plan, not the whole screen.)

b) [5 pts] SELECT * FROM Users WHERE first_name LIKE '%mil%';

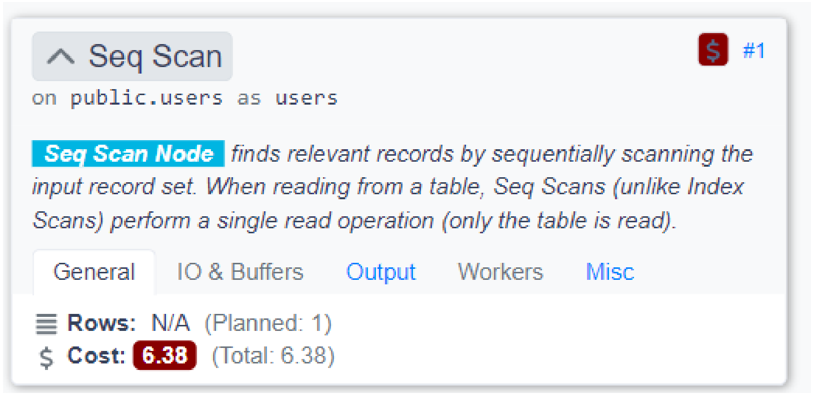

Seq Scan

on public.users as users

Seq Scan Node *finds relevant records by sequentially scanning the input record set. When reading from a table, Seq Scans (unlike Index Scans) perform a single read operation (only the table is read).*

Rows: N/A (Planned: 1)
Cost: 6.38 (Total: 6.38)

c) [5 pts] SELECT * FROM Users WHERE first_name LIKE 'Jay%';

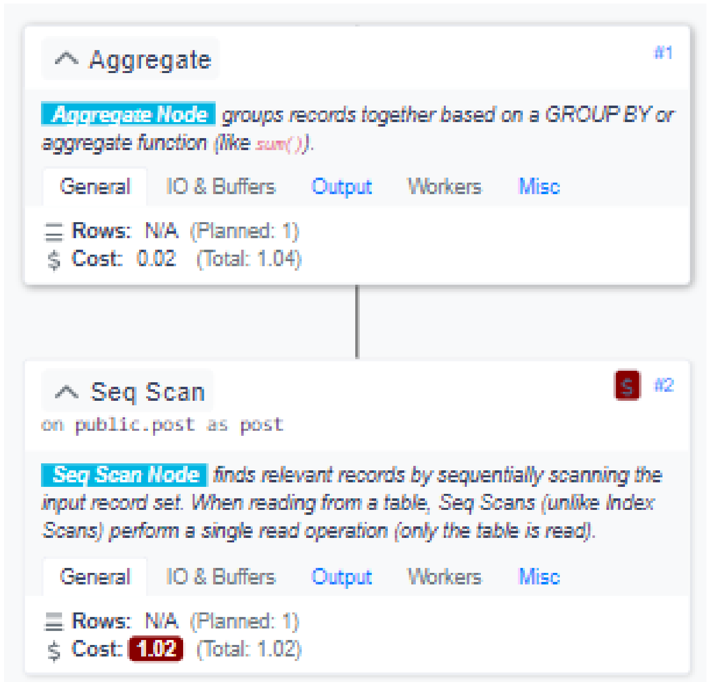

Seq Scan

on public.users as users

Seq Scan Node *finds relevant records by sequentially scanning the input record set. When reading from a table, Seq Scans (unlike Index Scans) perform a single read operation (only the table is read).*

Rows: N/A (Planned: 1)
Cost: 6.38 (Total: 6.38)

d) [5 pts] SELECT COUNT(*) FROM Post WHERE popularity = 93;


Aggregate #1

Aggregate Node groups records together based on a GROUP BY or aggregate function (like sum()).

General | IO & Buffers | Output | Workers | Misc

Rows: N/A (Planned: 1)
Cost: 0.02 (Total: 1.04)

Seq Scan #2
on public.post as post

Seq Scan Node finds relevant records by sequentially scanning the input record set. When reading from a table, Seq Scans (unlike Index Scans) perform a single read operation (only the table is read).

General | IO & Buffers | Output | Workers | Misc

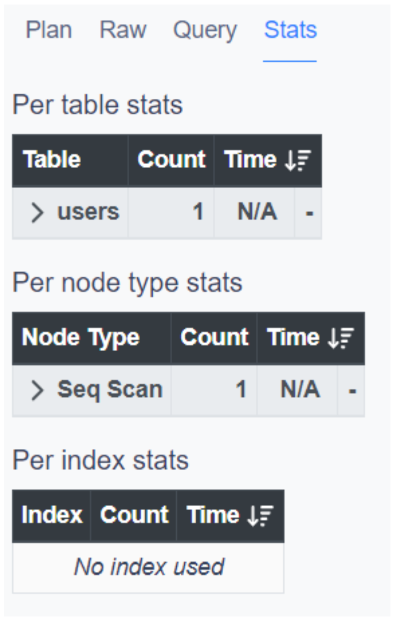
Rows: N/A (Planned: 1)
Cost: 1.02 (Total: 1.02)

Task 6 [10 pts]
Now create indexes (which are B+ trees under the hood of Postgres SQL) on the User.first_name attribute and Post.popularity separately (I.e., create two indexes, one per table.) Paste your CREATE INDEX statements below.

```
CREATE INDEX idx_users_first_name ON Users (first_name);
CREATE INDEX idx_post_popularity ON Post (popularity);
```

Task 7 [15 pts]
[15 pts(5pts each query)] Re Explain the 3 queries in Task 5 and indicate whether the indexes you created in Q2 are used, and if so, whether the uses are index-only plans or not. Report the query plan after each query, as before.

First 2 Queries:


Plan Raw Query Stats

Per table stats

Table	Count	Time	
users	1	N/A	-

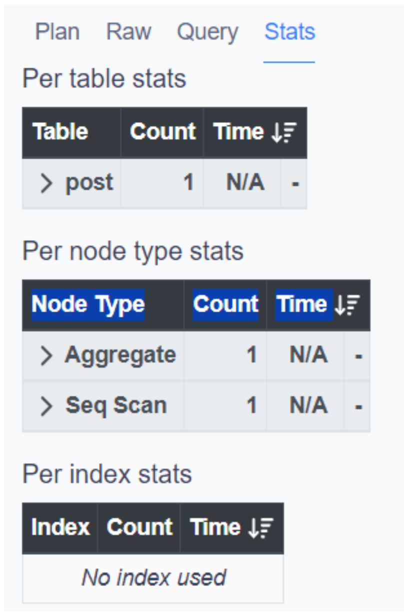
Per node type stats

Node Type	Count	Time	
Seq Scan	1	N/A	-

Per index stats

Index	Count	Time
No index used		

Last query:


Plan Raw Query Stats

Per table stats

Table	Count	Time	
post	1	N/A	-

Per node type stats

Node Type	Count	Time	
Aggregate	1	N/A	-
Seq Scan	1	N/A	-

Per index stats

Index	Count	Time
No index used		

All of the results were the same, none of the indices were used.

Task 8 [15 pts]
Examine each of the above queries with and without the use of an index. Please Briefly answer the following questions.

a) [5 pts] For range queries (e.g., query a), explain whether an index is useful and why. (Assume the number of result records in the selected range is extremely small compared to the total number of records in the file.)

For a range query, having an index can significantly improve performance, allows for efficient range scans, reduces I/O operations, optimizes query performance, and leverages index statistics to choose the best execution plan.

b) [5 pts] For each of the LIKE queries (b and c), explain whether an index is useful and why or why not (<=2 sentences per query).

For query b *SELECT * FROM Users WHERE first_name LIKE '%mil%'*, an index on the first_name column may not be very useful because the leading wildcard % prevents the effective use of the index.

However, for query c *SELECT * FROM Users WHERE first_name LIKE 'Jay%'*, an index on the first_name column can be very useful because the pattern starts with a string ('Jay') followed by a wildcard (%), allowing the database engine to efficiently utilize the index to quickly locate and retrieve matching records

c) [5 pts] For equality queries (e.g., query d), explain whether an index is useful and why (assuming the number of selected result records is extremely small compared to the number of records in the file).
For equality queries like query D *SELECT COUNT(*) FROM Post WHERE popularity = 93*, an index on the popularity column can be useful because it allows the database engine to quickly locate and count the matching records without having to scan the entire table, since the number of selected result records is extremely small compared to the total number of records.

Task 9 [20 pts].
It's time to go one step further and explore the notion of a "composite Index", which is an index that covers several fields together.

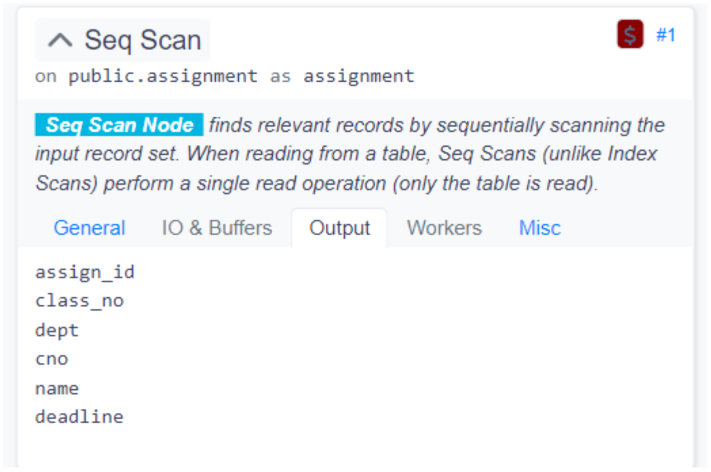
1) SELECT * FROM Assignment WHERE dept = 'COEN' and cno = '178';

2) SELECT * FROM Assignment WHERE cno = '222';

a) [5 pts] Create a composite index on the attributes dept and cno (in that order!) of the Assignment table. Paste your CREATE INDEX statement below.

CREATE INDEX idx_assignment_dept_cno ON Assignment (dept, cno);

b) [5 pts] 'Explain' the queries 1 and 2 above. Report on the query plan of each query, as before. Be sure to look carefully at each plan.


Seq Scan

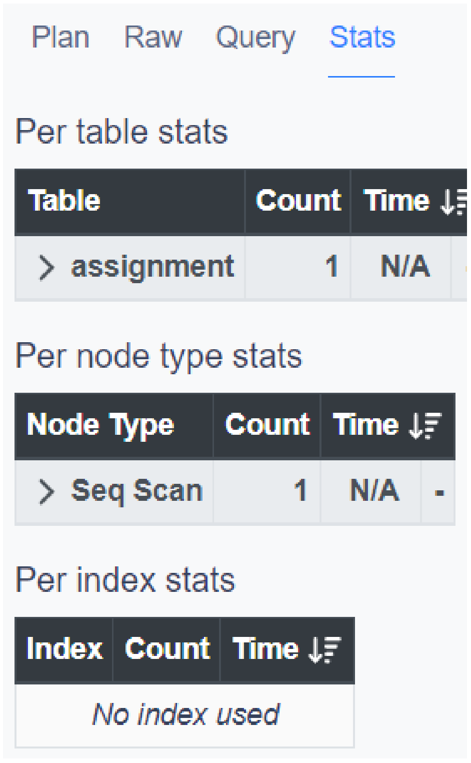
on public.assignment as assignment

Seq Scan Node finds relevant records by sequentially scanning the input record set. When reading from a table, Seq Scans (unlike Index Scans) perform a single read operation (only the table is read).

```
assign_id
class_no
dept
cno
name
deadline
```

c) [10 pts] Report for each query whether the composite index is used or not and why.

When I create the index, I am receiving the error ERROR: syntax error at or near ";" (SQLSTATE 42601). Thus, I assume that the composite index isn't being used. Additionally, when I explain when I run the index first and simply explain the 2 queries again this is the result:



Per table stats

Table	Count	Time
assignment	1	N/A

Per node type stats

Node Type	Count	Time	
Seq Scan	1	N/A	-

Per index stats

Index	Count	Time
No index used		

Thus either way, the composite index isn't being used.