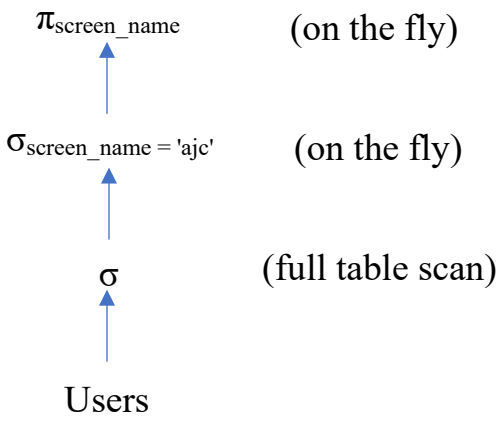
HW4 Question 2:

Full table scan

Users(screen_name varchar(80), user_name varchar(80), category varchar(80), subcategory varchar(80), state varchar(80), numFollowers int, numFollowing int, primary key(screen_name));
Tweets(tid BIGINT, post_day int, post_month int, post_year int, retweetCt int, posting_user varchar(80) not null, primary key(tid), foreign key(posting_user) references Users(screen_name));

Query 1: SELECT u.screen_name FROM Users u where u.screen_name = 'ajc';
The execution plan of the query is below. Notice that the name of the table is at the bottom of the query execution plan.



Disk I/O cost = cost of disk I/Os for doing the full table scan

In this plan, the estimated disk I/O cost is the cost of bringing all the rows in the **Users** table into memory using the full table scan algorithm to implement the selection operator. Since the data is already in memory, the projection operator can output the values of the required attributes without accessing the disk.

One disk I/O is required to retrieve one disk block/page from disk or write it back to disk as a disk block is a unit of disk retrieval.

The assumption is that the fixed-length record format is used to store a row in the **Users** table. Thus, each record has the same size and the maximum space to store all the attributes. The **Users** table has seven attributes. The maximum number of bytes per record is the sum of all the space in bytes of all the attributes.

- *bytes per record (x) = 1 (character) * 80 (maximum of 30 characters of screen_name) + 1 * 80 (user_name) + 1 * 80 (category) + 1 * 80 (subcategory) + 1 * 80 (state) + 4 (integer) + 4 (integer) = 408 bytes*
- *records per page (r) = floor(4000/x) = floor(4000/408) = 9 records due to the assumption that only 4000 bytes are used to store records and as many records as possible are stored per page.*

The floor(x) function outputs the greatest integer less than or equal to x. We use it since we cannot store a partial record in a page.

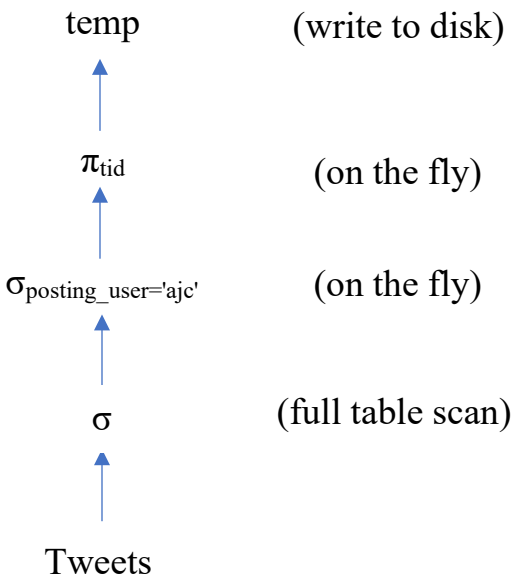
We have 5000 records. Each page can store as many as 9 records per the above calculation.

- *#total no. of pages for storing all the rows for this relation = ceil(5000/9) = 556*

The function ceil(x) returns the smallest integer greater than or equal to (x). We need the ceil() since even though the page is not full, we still need to retrieve the entire page since one page equals one disk block which is the unit of data retrieval from disk.

The heap file format needs 1 header page and #pages to keep all the records. However, the assumption asks to ignore the header page, so the ***disk I/O cost is 556***.

Query 2: INSERT into temp SELECT tid FROM Tweets where posting_user='ajc';



Disk I/O cost = cost of disk I/Os for doing the full table scan + cost of writing the content in *temp* to disk

Similar to the estimation of the cost of disk I/Os for doing the full table scan of Users table, we have the estimated cost of disk I/Os for doing the full table scan of Tweets table as follows:

- *bytes per record = 8 + 4 + 4 + 4 + 4 + 80 = 104*
- *records per page = floor(4000/104) = 38 rows*
- *total no. of pages for storing all the rows for this relation = ceil(10000/37) = 264*

To calculate the disk I/O cost for writing, we need to know how many rows are output and what is the number of bytes required for each row. The assumption is that only 10% of the input rows from Tweets satisfy the where clause:

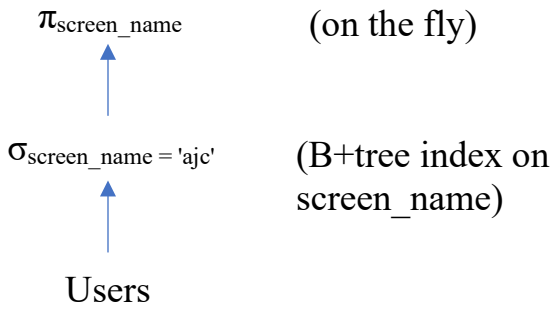
- *rows to write to disk = 0.1*10,000 = 1000 rows*
- *bytes per record for temp = 1 * 8 (BigInt) = 8 Bytes*
- *records per page = floor(4000/8) = 500 rows*
- *data pages to write 1000 rows = ceil(1000/500) = 2*

The heap file format needs 1 header page and #pages to keep all the records. However, the assumption asks to ignore the header page, so the ***disk I/O cost is* 264 + 2 = 266**.

Single index access path plan

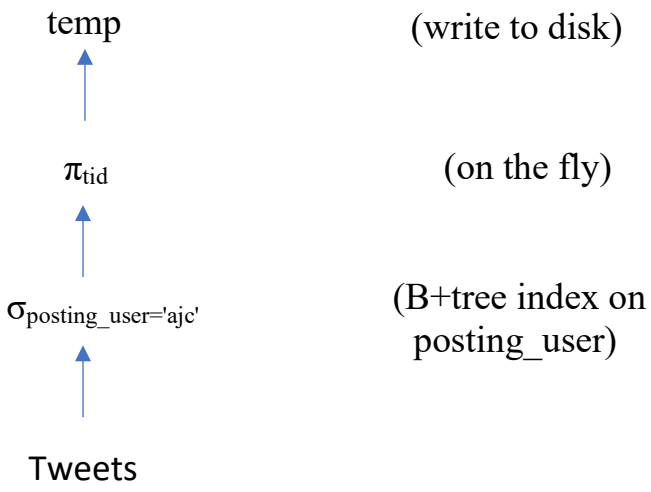
a) There is only one index relevant to the condition in the where clause and the attribute in the condition is the first attribute of the index and the attribute in the where clause is not inside a function. Hence, there is only one single index access path plan. If there were another index on screen_name, DBMS would consider the plan using that index as well.

SELECT u.screen_name FROM Users u where u.screen_name = 'ajc';



b) There is only one index relevant to the condition in the where clause and the attribute in the condition is the first attribute of the index and the attribute in the where clause is not inside a function. Hence, there is only one single index access path plan. If there were another index on posting_user, DBMS would consider the plan using that index as well.

INSERT into temp SELECT tid FROM Tweets where posting_user='ajc';



Since the only indexes available in this problem are B+Tree indexes, it is ok if the solution does not specify “B+Tree”, but the word “index” on which attribute is required.