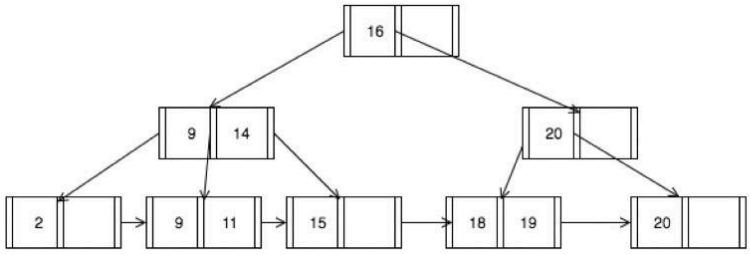
Question 1 (33 points)

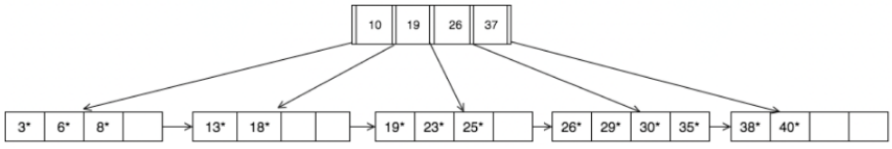
Q1.1(3 points): Consider the B+ tree in Figure 3, what is the minimum number of pointers to be followed to satisfy the query: Get all records with keys greater than 11 and less than 20?



Your answer: 5

Q1.2: Consider the B+ tree below of order d = 2 and height h = 2 levels. Please make the following assumptions:

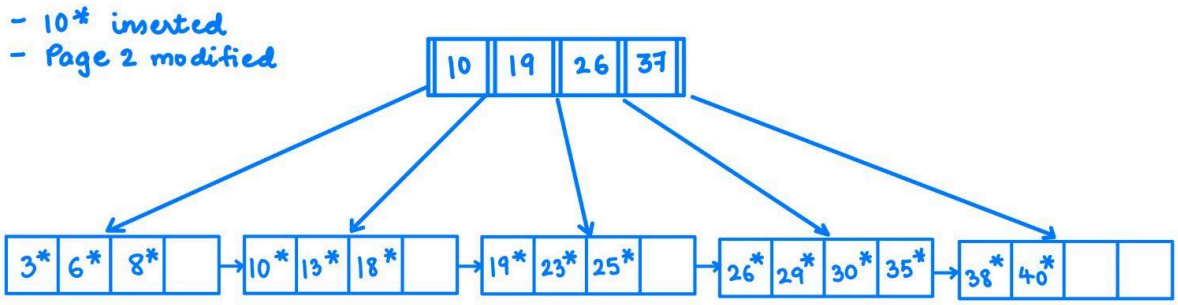
- With respect to“≥”, the left pointer is for <, the right one for ≥.
- In case of underflow, if you can borrow from both siblings, choose the one on the *right*.



For the questions below, you are allowed to only draw the pages that get modified and show the unmodified pages by their page ID (root is page 0, and page numbers increase from left to right in BFS format).

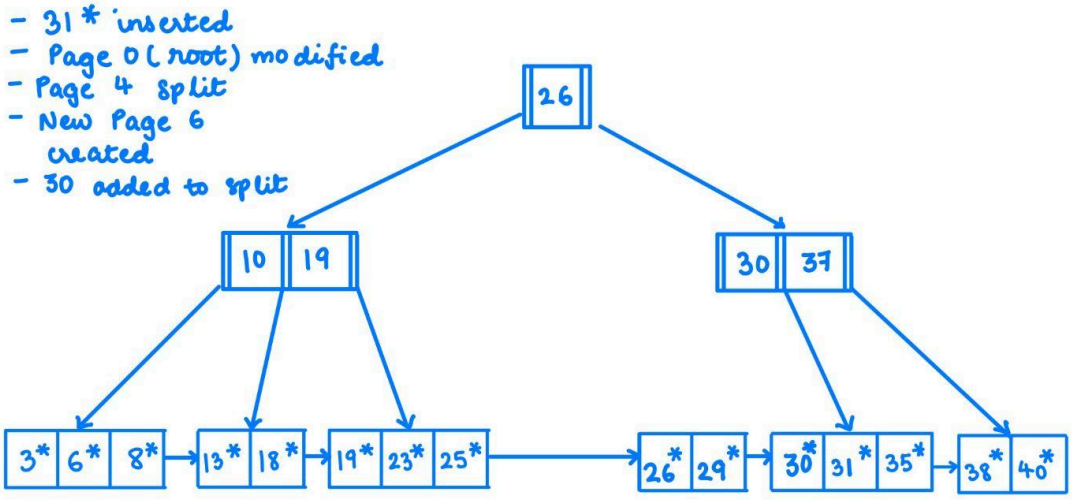
Q1.2.1(6 points): Start from the original B+ tree; insert 10∗.

Modified Page 2, after insertion = 10*,13*,18*
Unmodified: Pages = 0, 1, 3, 4, 5



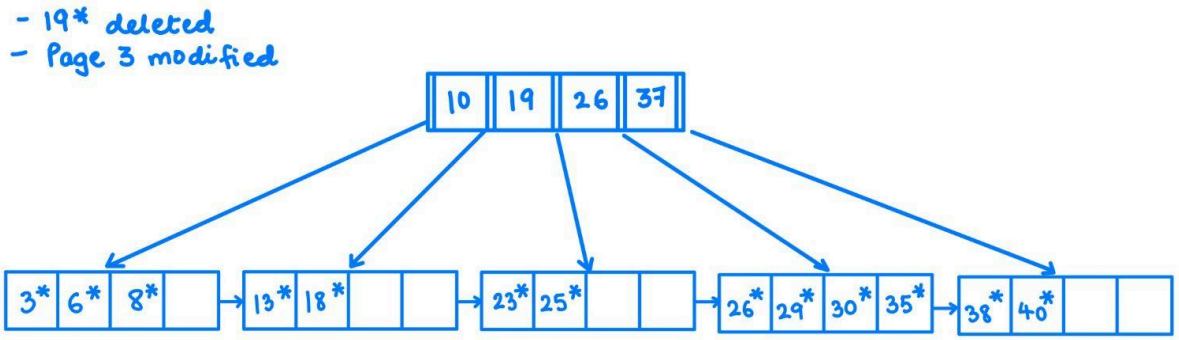
Q1.2.2(6 points): Start from the original B+ tree; insert 31∗.

Modified Page 0 (root) = 26
Modified Page 4 (split) = Left child: 26*, 29*; Right child: 30*, 31*, 35*
New Page 3 created and 30 added for the Page 4 split
Unmodified: Pages = 1, 2, 3, 5



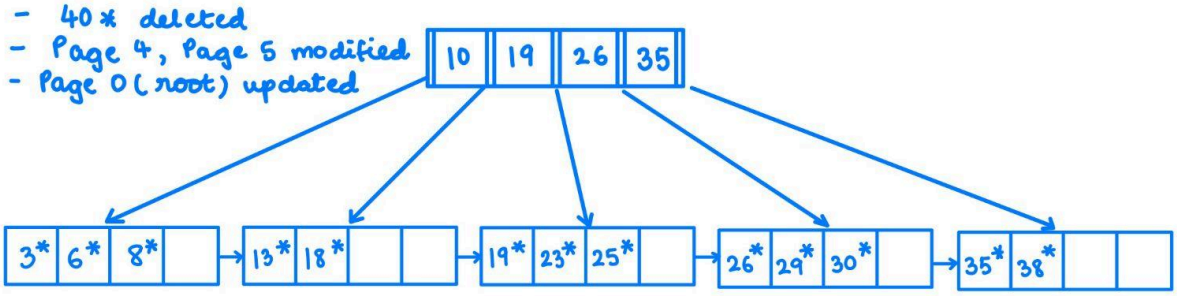
Q1.2.3 (6 points): Start from the original B+ tree; delete 19∗.

Modified Page 3, after deletion = 23*, 29*
Unmodified: Pages = 0, 1, 2, 4, 5



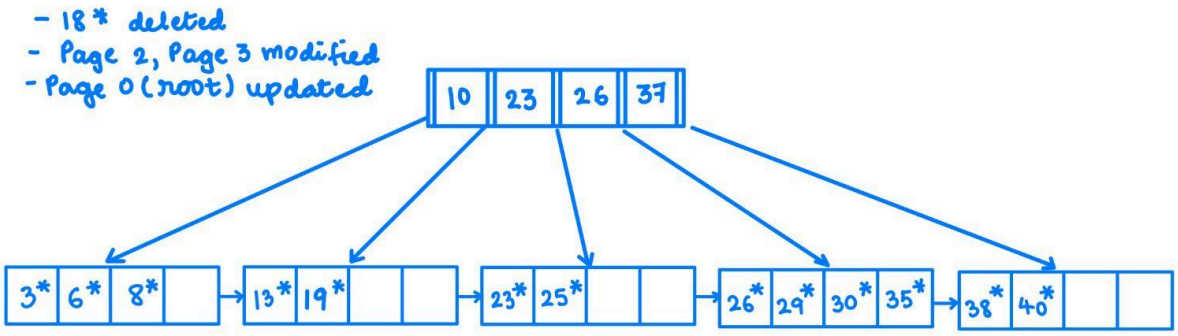
Q1.2.4(6 points): Start from the original B+ tree; delete 40∗.

Modified Page 5, after deletion = 35*, 38*
Modified Page 4 = 26*, 29*, 30*
Page 0 updated
Unmodified: Pages = 1, 2, 3



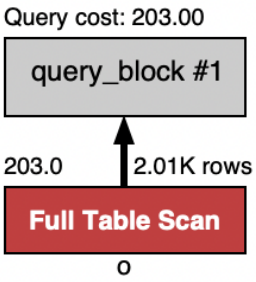
Q1.2.5(6 points): Start from the original B+ tree; delete 18∗.

Modified Page 2, after deletion = 13*, 19*
Modified Page 3 = 23*, 25*
Page 0 updated
Unmodified: Pages = 1, 4, 5

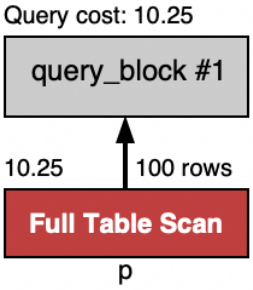


2. [12 pts] Report on the query plan of each query. (Snapshot of the query plan, not the whole screen)

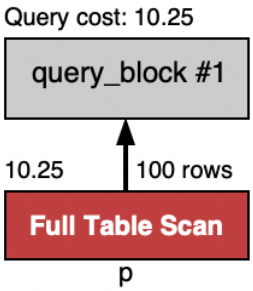
a) [3 pts] SELECT * FROM Observable o WHERE o.rate > 60 AND o.rate < 70;



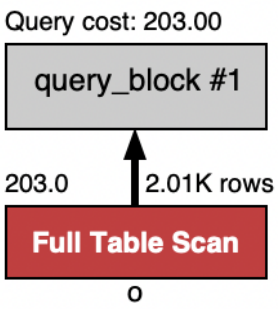
b) [3 pts] SELECT * FROM PHLogger p WHERE p.name LIKE "Adeline%";



c) [3 pts] SELECT * FROM PHLogger p WHERE p.name LIKE "%eiten%";



d) [3 pts] SELECT count(*) FROM Observable o WHERE o.rate = 74;

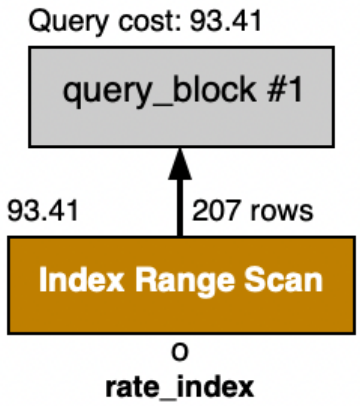


3. [4 pts] Now create indexes (which are B+ trees, under the hood of MySQL) on the PHLogger.name attribute and Observable.rate. (e.g., create two indexes, one per table.) Paste your CREATE INDEX statements below.

```
CREATE INDEX name_index ON PHLogger(name) using btree;
CREATE INDEX rate_index ON Observable(rate) using btree;
```

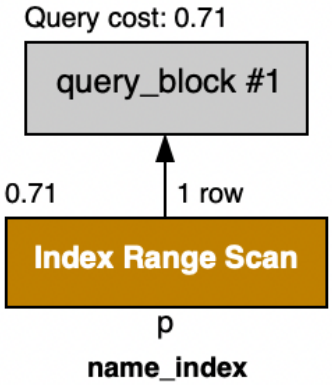
4. [12 pts] Re-explain the queries in Q1 and indicate whether the indexes you created in Q2 are used, and **if so whether it is an index-only plan**. Report on the query plan after each query, as before.

a. [3 pts] SELECT * FROM Observable o WHERE o.rate > 60 AND o.rate < 70;



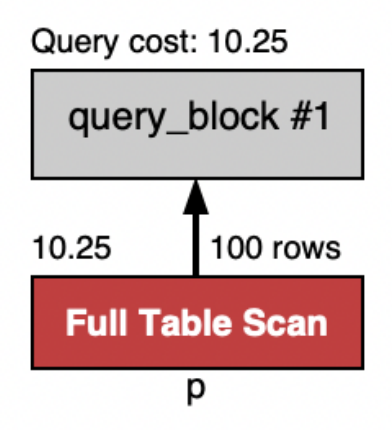
Created index **rate_index** is used. Not an index-only plan.

b. [3 pts] SELECT * FROM PHLogger p WHERE p.name LIKE "Adeline%";



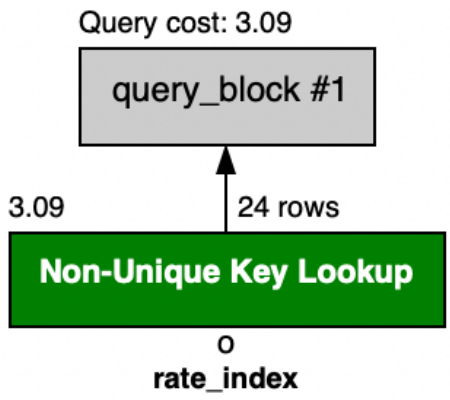
Created index **name_index** is used. Not an index-only plan.

c. [3 pts] SELECT * FROM PHLogger p WHERE p.name LIKE "%eiten%";



No created index is used.

d. [3 pts] SELECT count(*) FROM Observable o WHERE o.rate = 74;



Created index **rate_index** is used. This is an index-only plan.

5. [15 pts] Examine 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 number of records in the file).

For query A, the database will use the rate index to find all the entries where the heart rate is between 60 and 70. This is faster and cheaper than scanning the entire table. This way is quicker, and it is easier than going through the whole table.

b. [5 pts] For each LIKE query (b and c), explain whether an index is useful and why (<= 2 sentences per query).

For query B (LIKE 'Adeline%'), the first_name index works because the search starts from the beginning of the name. Using the index is faster and cheaper than scanning the whole table. This way is quicker, and it is easier than going through the whole table.

For query C, LIKE '%eiten%' has to do a full table scan because it's not a range query like query A that the B+ tree index can handle. The search is looking for 'eiten' anywhere in the text, so the index doesn't really help.

c. [5 pts] For equality queries (e.g., query d), explain whether an index is useful and why (again assuming that the number of selected result records is small compared to the number of records in the file).

For query D (rate = 74), the rate index is used, and it's even better because it's an index-only query. That means it only looks at the index itself and doesn't have to go into the actual data records, which makes it way cheaper than scanning the whole table. This way is quicker, and it is easier than going through the whole table.

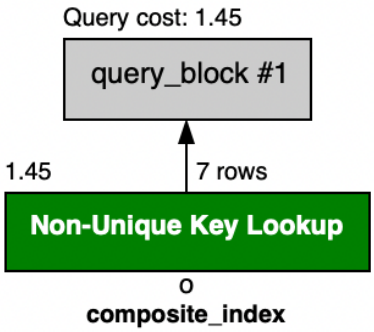
6.[12 pts] It's time to go one step further and explore the notion of a "composite Index", which is an index that covers several fields together.

d. [4 pts] Create a composite index on the attributes manufacturer and model (in that order!) of the Observer table. Paste your CREATE INDEX statement below.

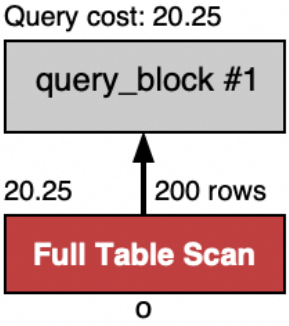
```
CREATE INDEX composite_index ON Observer (manufacturer, model);
```

e. [4 pts] 'Explain' the queries 1 and 2 below. Report on the query plan of each query, as before.

i. [2 pts] SELECT * FROM Observer o WHERE o.manufacturer = 'Google' and o.model = 'Model 3';



ii. [2 pts] SELECT * FROM Observer o WHERE o.model = 'Model 3';



c. [4 pts] Report for each query whether the composite index is used or not and why (<= 2 sentences per query).

For query A, the composite index works here because it's an equality search. Every column in the index is matched to a specific value. That makes using the index possible and faster than scanning the whole table.

For query B, the composite index doesn't work for this one because the columns in the index are ordered a certain way. Since the query is trying to find values in the second column without filtering by the first column, it would basically have to scan the whole index anyway so the index doesn't really help.

7.[12 pts] For each of the following queries, indicate whether the use of an index would be helpful or not. If so, specify which tables and attributes an index should be created on and the best choice between a clustered or unclustered index. For the sake of this question, you don't have to worry about how your choice of the index would affect other queries running in the system -- consider each query in isolation.

f. [4 pts] SELECT * FROM Observer o WHERE o.phlid = 1;

We should create an index here since it's an equality query. It would be a clustered index on the Observer table for the phlid column.

g. [4 pts] SELECT o.kind, count(*) AS cnt FROM Observer o GROUP BY o.kind;

We should create an index for the grouping query. The index would be unclustered on the Observer table for the kind attribute.

h. [4 pts] SELECT * FROM PHLG_obs;

Since it is a SELECT * query, we don't need to create an index here.