Lab 7: SQL Design and Indexing (100 points)

***Due Date:*** **Sunday, March 16th (11:59 PM)**

### Submission

All lab assignments should contain both your student ID and your name and must be submitted online (e.g., 12345678_John_Doe_Lab7.pdf) via the Lab7 assignment on Gradescope. See the table below for the Lab 7submission opportunities. Note that after 11:59 PM on Monday the 17th$^{th}$ no further Lab 7 submissions will be accepted.

| **Date / Time** | **Grade Implications** |
|---|---|
| Sunday, March 16th (11:59 PM) | Full credit will be available |
| Monday, March 17th (11:59PM) | 10% will be deducted |

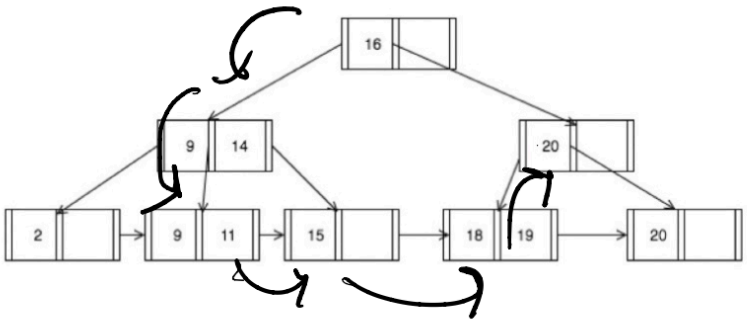
## SQL Design and Indexing **(SQL) [100 pts]**

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. MySQL 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.

**Note:** For this assignment, you can turn in a PDF by cutting/pasting from MySQL into a copy of the Lab7 template (this file itself) and then **PDF-printing** the results. **Make sure all screenshots are properly included. Do not snapshot the whole screen.Only the green area is needed.**

Question 1 (33 points)

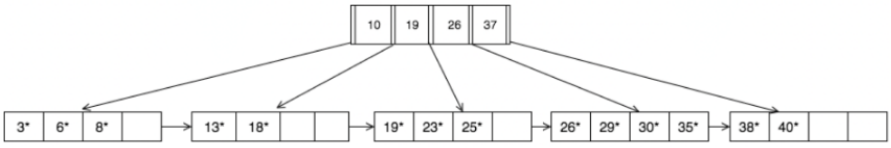
Q1.1(3 points): Consider the B+ tree in Figure 3, what is the minimum number of pointers to be followed to satisfy the query: Get all records with keys greater than 11 and less than 20?




Your answer: 5

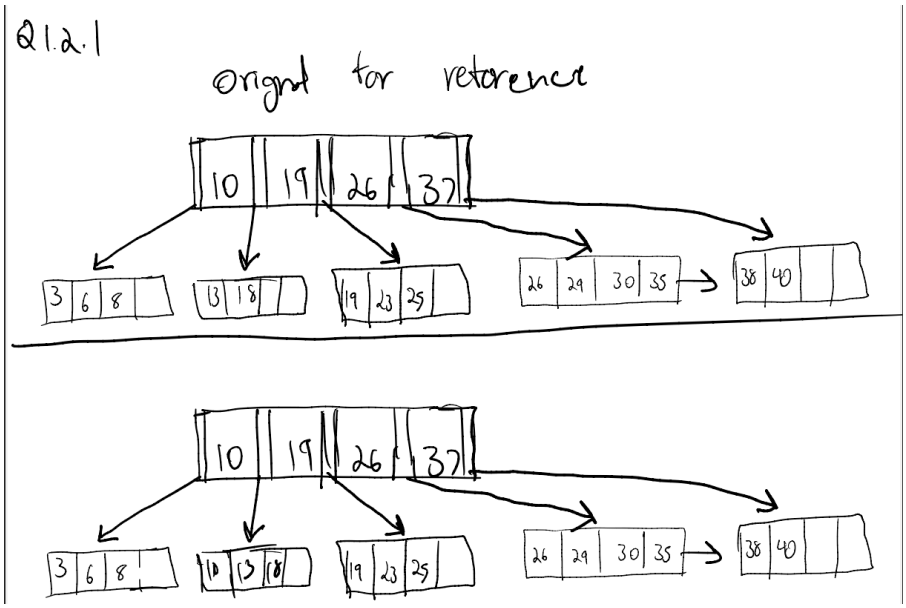
Q1.2: Consider the B+ tree below of order d = 2 and height h = 2 levels. Please make the following assumptions:

- 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*.

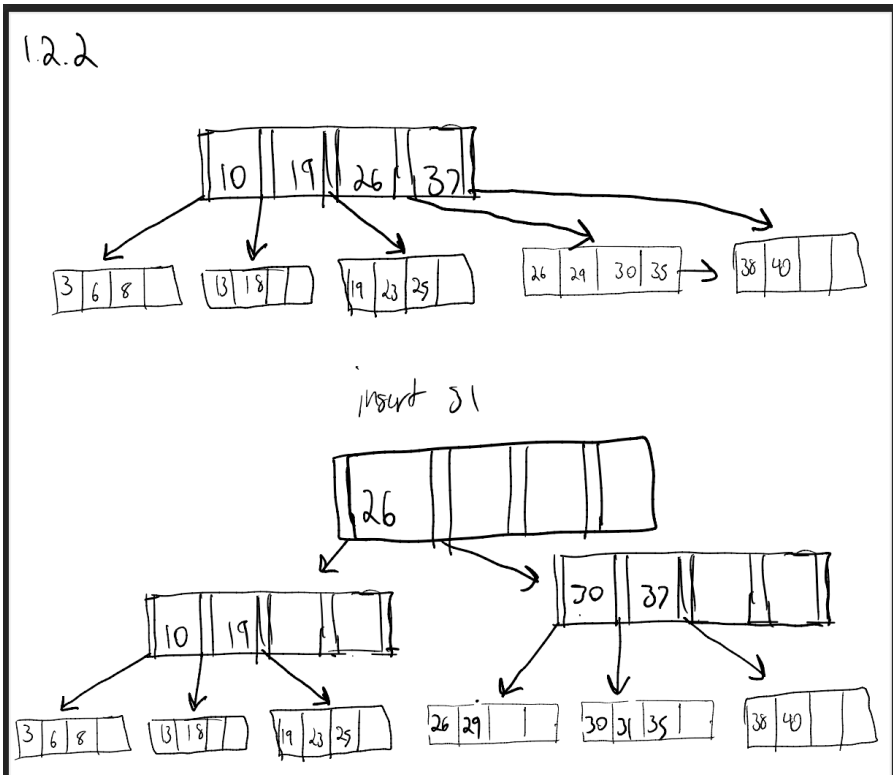



For the questions below, you are allowed to only draw the pages that get modified and show the unmodified pages by their page id (root is page 0, and page numbers increase from left to right in BFS format).

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

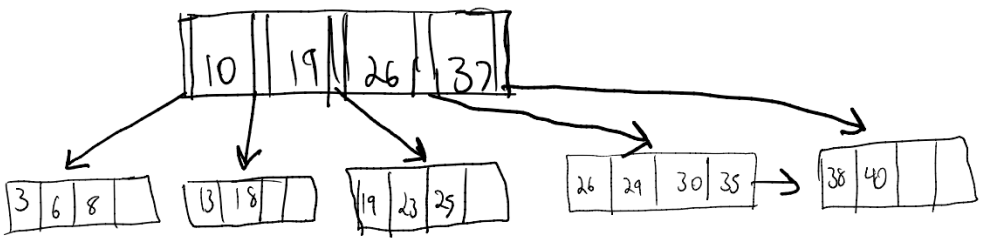



Q1.2.2(6 points): Start from the original B+ tree; insert 31∗.

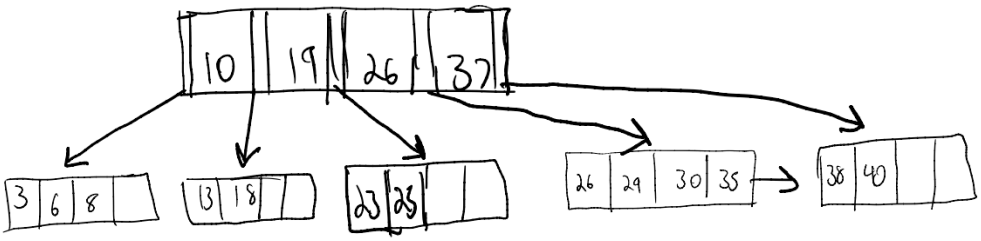


Q1.2.3 (6 points): Start from the original B+ tree; delete 19∗.

1.2.3

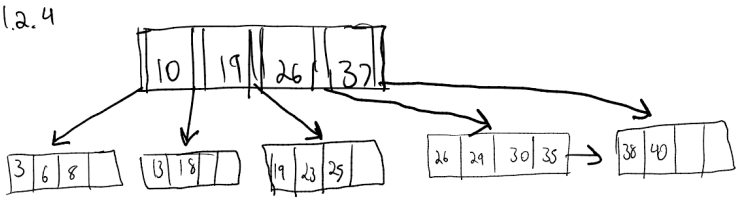



delete 19

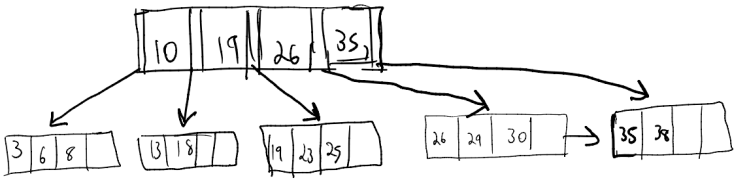



Q1.2.4(6 points): Start from the original B+ tree; delete 40∗.

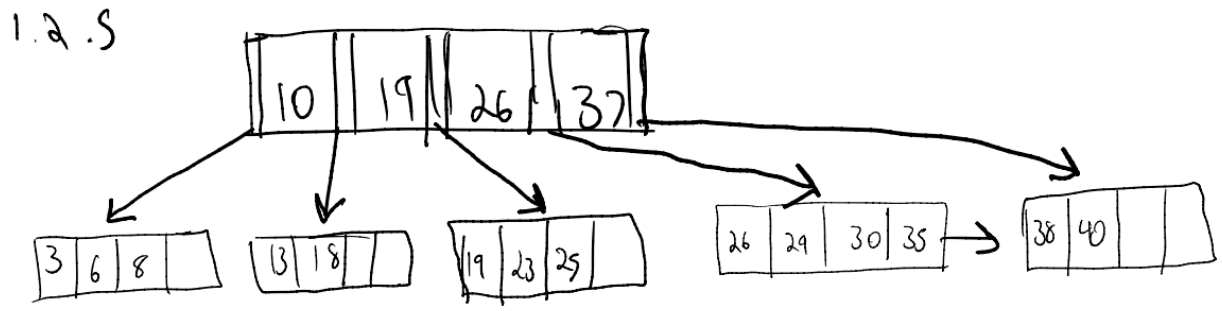
1.2.4



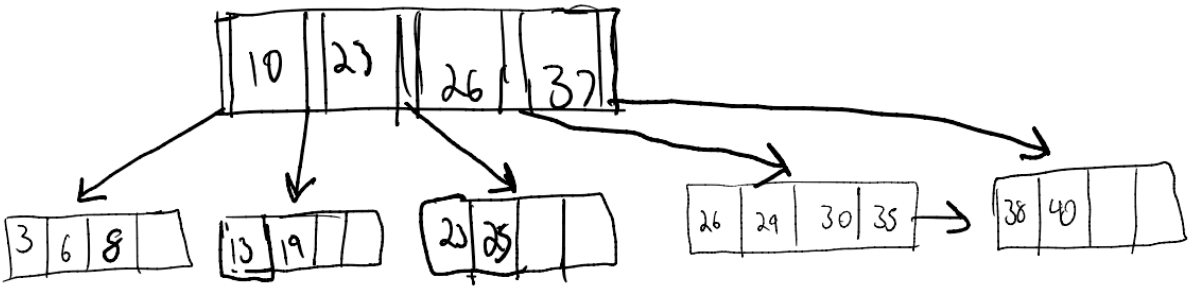

delete 40

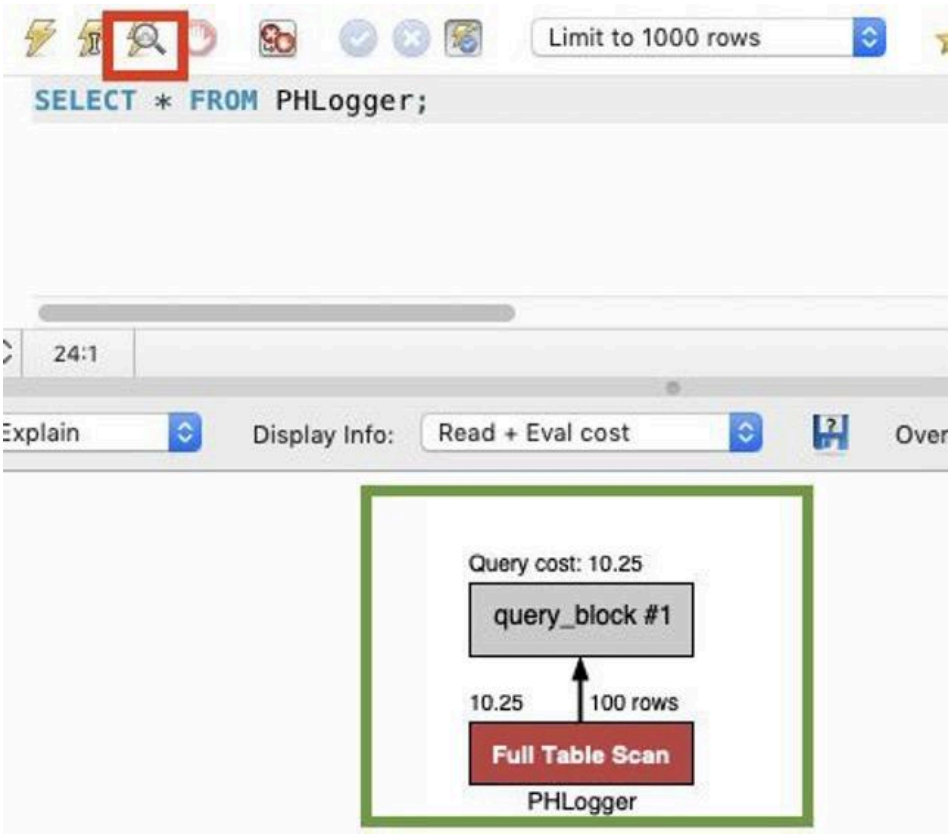


Q1.2.5(6 points): Start from the original B+ tree; delete 18∗.




delete 18



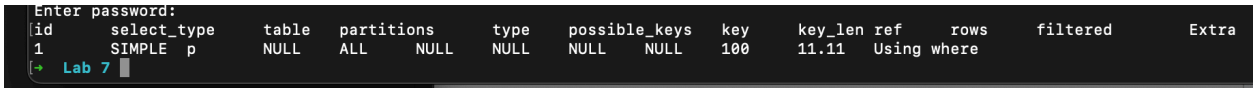



2. [12 pts] Report on the query plan of each query. (Snapshot of the query plan, not the whole screen)

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

```
Enter password:
id      select_type     table   partitions      type    possible_keys   key     key_len ref     rows    filtered        Extra
1       SIMPLE  o       NULL    ALL     NULL    NULL    NULL    NULL    2010    11.11   Using where
→  Lab 7
```

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 PHLogger_name_index
    ON PHLogger(name);

CREATE INDEX Observable_rate_index
    ON Observable(rate);
```

4. [12 pts] Re-explain the queries in Q2 and indicate whether the indexes you created in Q3 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;

```
(➜  Lab 7 mysql -u root -p lab6 < q2a.sql
Enter password:
id      select_type     table   partitions      type    possible_keys   key     key_len ref     rows    filtered        Extra
1       SIMPLE  o       NULL    range   Observable_rate_index   Observable_rate_index   5       NULL    207     100.00  Using index condition
```

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

```
Enter password:
id      select_type     table   partitions      type    possible_keys   key     key_len ref     rows    filtered        Extra
1       SIMPLE  p       NULL    range   PHLogger_name_index     PHLogger_name_index     202     NULL    1       100.00  Using index condition
```

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

```
id      select_type     table   partitions      type    possible_keys   key     key_len ref     rows    filtered        Extra
1       SIMPLE  p       NULL    ALL     NULL    NULL    NULL    NULL    100     11.11   Using where
```

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

```
Enter password:
id      select_type     table   partitions      type    possible_keys   key     key_len ref     rows    filtered        Extra
1       SIMPLE  o       NULL    ref     Observable_rate_index   Observable_rate_index   5       const   24      100.00  Using index
```

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 resultrecords in the selected range is extremely small compared to the number of records in the file).

Indexes would be very useful for range queries. Since SQL is essentially a b+ tree behind the scene we can use the example, a B+ tree can quickly guide us to the correct leaf node, so we only scan the portion of data that falls within the specified range. This approach lets us jump straight to the start of the range and then read until we reach the end, rather than scanning the entire table. It's especially beneficial when the range is small compared to the total number of records.

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

When the query uses a trailing wildcard like in query b, MySQL can leverage an index on p.name to treat it like a range lookup, starting at "Adeline" and scanning until the next larger prefix. This avoids a full table scan and speeds up the search considerably.

However on query c since it has a leading wildcard, it prevents MySQL from identifying a specific starting position in the index, making a full table scan necessary.

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 equality queries like WHERE o.rate = 74, a B+ tree index can guide us directly to the matching record(s) without scanning the entire table. This approach quickly locates the needed rows and is especially helpful when only a small fraction of rows match the condition.

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_observer_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';

| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | SIMPLE | o | NULL | ref | composite_observer_index | composite_observer_index | 404 | const,const | 7 | 100.00 | NULL |

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

| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | SIMPLE | o | NULL | ALL | NULL | NULL | NULL | 200 | 10.00 | Using where | |

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

The composite index is used in query 1 because the query constrains both columns in the same order (manufacturer first, then model). This lets MySQL jump directly to the matching rows instead of scanning the entire table.

The composite index is not used for query 2 because the leading column is not given, so MySQL cannot do a direct range lookup on just model. As a result, it falls back to a full table scan, applying the WHERE condition row by row.

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 inthe system -- consider each query in isolation.

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

An index would be very useful here, specifically a clustered index. This is because of the search on the primary key of phild. So the index must be o.phlid to have an index on phild in the observer table

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

An index will be useful here to leverage its grouping abilities. The index would need to be made on kind in the observer table. Since we do not fetch records from the disk it doesn't matter if we cluster or uncluster the index. However to keep efficiency and ease of code we do not do a clustered and do an unclustered index

h. [4 pts] SELECT * FROM PHLG_obs;

An index would not be beneficial here, as we need to scan every row since there is no clause.