1. [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;

   ![Query Plan for a]

   Query cost: 203.00
   query_block #1
   203.0 2.01K rows
   Full Table Scan
   o

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

   ![Query Plan for b]

   Query cost: 10.25
   query_block #1
   10.25 100 rows
   Full Table Scan
   p
c) [3 pts] SELECT * FROM PHLogger p WHERE p.name LIKE "%eiten%";

Query cost: 10.25

```
query_block #1
```

```
10.25
100 rows
```

```
Full Table Scan
p
```

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

Query cost: 203.00

```
query_block #1
```

```
203.0
2.01K rows
```

```
Full Table Scan
o
```
2. [10 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.

```sql
CREATE INDEX name_index ON PHLogger(name) using btree;
CREATE INDEX heart_rate_index ON Observable(rate) using btree;
```
3. [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;

The heart_rate_index is used. Not an index-only plan.

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

The name_index is used. Not an index-only plan.
c) [3 pts] SELECT * FROM PHLogger p WHERE p.name LIKE "%eiten%";

No index is used.

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

The heart_rate_index is used. It is an index-only plan.
4. [21 pts] Examine the above queries with and without the use of an index. Please briefly answer the following questions.

a) [7 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).

Query a: for the range query (60 < rate <= 70), it will search for all entries with heart rate fall in the specific range using the heart rate index, and it will cost less than using a full table scan.

(Note (not part of students’ answers): If we had many entries in that range it could end up costing more than doing a full table scan due to a random read for each entry. In that case, the system should choose a full table scan.)

b) [7 pts] For each LIKE query (b and c), explain whether an index is useful and why (<= 2 sentences per query).

Query b: the query (LIKE '%eiten%') still needs a full table scan because it isn’t a range query that can be handled by a B+ tree index. This is because the condition is not on the prefix but any part of the key.

Query c: for the range query (LIKE 'Adeline%'), the first_name index is applicable because the condition is on the prefix of the key and using the index will cost less than using a full table scan.

c) [7 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).

Query d: for the equality query (rate = 74), the heart rate index is applied and -- better yet -- it is an index-only query. It is much, much cheaper than doing a full table scan because the index-only plan only touches the index entries and does not touch the data records.
5. [21 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.

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

   ```sql
   CREATE INDEX comp_idx ON Observer (manufacturer, model);
   ```

b) [6 pts] ‘Explain’ the queries 1 and 2 below. Report on the query plan of each query, as before.

1) [3 pts] SELECT * FROM Observer o WHERE o.manufacturer = 'Google' and o.model = 'Model 3';

   ![Query 1 Diagram]

   Query cost: 1.45
   query_block #1
   1.45 7 rows
   Non-Unique Key Lookup
   0 comp_idx

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

   ![Query 2 Diagram]

   Query cost: 20.25
   query_block #1
   20.25 200 rows
   Full Table Scan
   0
c. [10 pts] Report for each query whether the composite index is used or not and why (<= 2 sentences per query).

Query 1: the composite index is used because it’s an equality search where every field is equal to a specific value, hence using the index is possible and faster than a full table scan.

Query 2: the composite index is not used because values in the additional composite index columns are grouped by the order of the first column. In order to find values in the second column without having an index on the first column, it would have to scan the entire index.

(Note (not part of students’ answers): Some systems, e.g., DB2 from IBM for one, might still choose a full index scan after estimating the costs of all alternatives.)
6. [24 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.

a) [8 pts] SELECT * FROM Observer o WHERE o.phlid = 1;

An index should be created here for the equality query. The index should be a **clustered** index on table Observer and attribute phlid because phlid in Post table is not unique and clustering would thus make retrieving results faster because then all of a given user's Observer records will be co-located on disk. (This is assuming the number of result records is low comparing to the number of records in the file.)

b) [8 pts] SELECT o.kind, count(*) AS cnt FROM Observer o GROUP BY o.kind;

An index should be created here for grouping query, and it can be processed as an index-only query. The index could be an **unclustered** index on table Observer and attribute kind because index-only queries do not touch the data (so clustering is irrelevant and it should thus be used for something else!).

c) [8 pts] SELECT * FROM PHLG_obs;

No need to create an index here. A full table scan is the best option.