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 Post WHERE popularity > 98 AND popularity <= 100;
   
   ![Query Plan 1](image1)

   b) [3 pts] SELECT * FROM User WHERE first_name LIKE 'mil%';
   
   ![Query Plan 2](image2)

   c) [3 pts] SELECT * FROM User WHERE first_name LIKE 'Jay%';
   
   ![Query Plan 3](image3)

   d) [3 pts] SELECT COUNT(*) FROM Post WHERE popularity = 93;
   
   ![Query Plan 4](image4)
2. [10 pts] Now create indexes (which are B+ trees, under the hood of MySQL) on the User.first_name attribute and Post.popularity. (e.g., create two indexes, one per table.) Paste your CREATE INDEX statements below.

   CREATE INDEX first_name_index ON User(first_name) using btree;
   CREATE INDEX popularity_index ON Post(popularity) using btree;

3. [12 pts] Re-explain the queries in Q1 and indicate whether the indexes you created in Q2 is used, if so whether it is an index-only plan. Report on the query plan after each query, as before.

   a) [3 pts] SELECT * FROM Post WHERE popularity > 98 AND popularity <= 100;

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

   b) [3 pts] SELECT * FROM User WHERE first_name LIKE '%mil%';

   No index is used.
c) [3 pts] SELECT * FROM User WHERE first_name LIKE 'Jay%';

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

d) [3 pts] SELECT COUNT(*) FROM Post WHERE popularity = 93;

The popularity_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 (98 < popularity <= 100), it will search for all entries with popularity fall in the specific range using the popularity index, and it will cost less than using a full table scan.
(Note: If we had more entries in that range it could end up costing more than doing a full table scan due to random read for each entry. In that case, the system should choose a full table scan.)

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

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

Query c: for the range query (LIKE 'Jay%'), the first_name index is applied, because the condition is on the prefix of the key and it 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 (assume the number of selected result records is extremely small compared to the number of records in the file).

Query d: for the equality query (popularity = 93), the popularity index is applied and it is the index-only query. It is cheaper than doing a full table scan because 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 dept and cno (in that order!) of the Assignment table. Paste your CREATE INDEX statement below.

```
CREATE INDEX comp_idx ON Assignment (dept, cno);
```

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

1) [3 pts] SELECT * FROM Assignment WHERE dept = 'CS' and cno = '122A';

![Query plan for query 1](image1)

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

2) [3 pts] SELECT * FROM Assignment WHERE cno = '222';

![Query plan for query 2](image2)

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.

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 such that every field is equal to a specific value, hence using the index is 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.
6. [24 pts] For each of the following queries indicate whether a 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 purpose of this question, you can assume that the system provides that choice.

a) [8 pts] SELECT * FROM Post WHERE user_id = “998”; (Assume the number of result record is < 1%)

An index should be created here for the equality query. The index should be a clustered index on table Post and attribute user_id because user_id in Post table is not unique and clustering would make retrieving results faster because then all of a given user's Post 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 kind, COUNT(*) as cnt FROM Post GROUP BY kind;

An index should be created here for grouping query, and in fact it can be processed as an index-only query. The index should be an unclustered index on table Post 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 where it can help).

c) [8 pts] SELECT * FROM Assignment;

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