1. [15 pts] To start down the path of exploring physical database designs (indexing) and query plans, start by checking the queries plans for each of the following queries without any indexes using the EXPLAIN function (see above). Report the query plan after each query by taking snapshots and pasting them into your copy of HW7 template file.
   
a. SELECT * FROM Users WHERE name_first = 'Scott';

   ![Query Plan](image1)

   **Query cost: 5.40**
   
   query_block #1
   
   5.4  22 rows
   
   Full Table Scan
   
   Users

   b. SELECT * FROM Users WHERE name_first LIKE 'Hil%';

   ![Query Plan](image2)

   **Query cost: 5.40**
   
   query_block #1
   
   5.4  22 rows
   
   Full Table Scan
   
   Users
2. [10 pts] Now create indexes (which are B+ trees, under the hood of MySQL) on the Users.name_first attribute and Users.user_since attribute separately - i.e., create two indexes, one per attribute. Paste your CREATE INDEX statements below.

    CREATE INDEX name_first_index ON Users(name_first) using btree;
    CREATE INDEX user_since_index ON Users(user_since) using btree;
3. [15 pts] Re-explain the queries in Q1. Report on the query plan after each query, as before. Be sure to look carefully at each plan - think about what you are seeing there.
   
   a. SELECT * FROM Users WHERE name_first = 'Scott';

   ![Query Plan Diagram]

   Query cost: 1.20
   query_block #1
   1 row
   Non-Unique Key Lookup
   Users
   name_first_index

   b. SELECT * FROM Users WHERE name_first LIKE 'Hil%';

   ![Query Plan Diagram]

   Query cost: 2.41
   query_block #1
   1 row
   Index Range Scan
   Users
   name_first_index

   c. SELECT * FROM Users WHERE user_since BETWEEN '2000-01-01 00:00:00'
      AND '2002-01-01 00:00:00';

   ![Query Plan Diagram]

   Query cost: 2.41
   query_block #1
   1 row
   Index Range Scan
   Users
   user_since_index
4. [20 pts] Examine the above queries with and without the use of an index. Please briefly answer the following questions.
   a. Describe the index usefulness for equality and LIKE queries (q1 and q2).

   Query1: for an equality query we see that it is beneficial to have an index to speed up the lookup of the data rather than using a full table scan.

   Query2: the query is treated as a range query due to the condition for the like operator and using an index is more useful than doing a full table scan.

   b. Describe the index usefulness for range queries (q3).

   Query3: for the range query it will search for all entries from '2000-01-01 00:00:00' to '2002-01-01 00:00:00' on B+ tree, and it will cost less than using a full table scan.

   (Optional note: If we had more entries in that range it could end up costing more than using a full table scan due to the random reads for every entry. In that case MySQL would hopefully have picked a full scan for the query plan.)
5. [40 pts] Enough about MySQL! For each of the following queries indicate whether a use of an index would be helpful or not. If so, specify which tables and columns 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. (Make your answer brief.)

   a. SELECT * FROM Users;

   We don’t need to create an index here as a full table scan is the best option.

   b. SELECT * FROM Posters WHERE userid = 3;

   An index would be helpful for this equality query. The index should be on table Posters and attribute userid and it should be an unclustered index since clustering would serve no useful purpose because userid is unique (requiring one random read regardless).

   (General note: Clustering is not “for free” and should not be “wasted” on queries where choosing it doesn’t improve performance.)

   c. SELECT name FROM Publisher WHERE name = 'Irvine Post';

   An index would be helpful for this query, and in fact it can be processed as an index-only query. The index should be on table Publisher and attribute name and it should be an unclustered index since an index-only query doesn’t touch the data itself (so the data’s clustering is irrelevant).

   d. SELECT userid,COUNT(*) FROM Has_read GROUP BY userid;

   An index would be helpful for this grouping query since it can then be executed as an index-only query. The index should be on table Has_read and attribute userid and it should be an unclustered index. (Recall that clustering is a waste for index-only queries.)