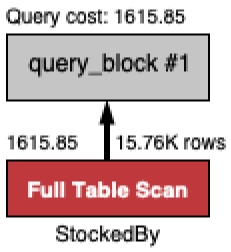
Last Name: Rachmat First Name: Anthony Student ID: 26339003

1. [12 pts] To start down the path of exploring physical database designs (indexing) and query plans, start by checking the query plans for each of the following queries against the database without any indexes using the EXPLAIN function (see instructions). For each query, take snapshots of the EXPLAIN results and paste them into your copy of the HW7 template file. (Just take a snapshot of the query plan, not the whole screen.)

a) [3 pts]

```sql
SELECT * FROM StockedBy WHERE qty BETWEEN 30 AND 35;
```

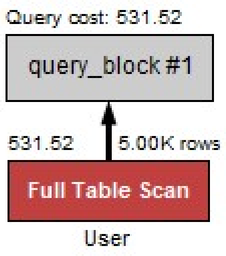
Query Plan:



b) [3 pts]

```sql
SELECT * FROM User WHERE last_name LIKE '%pot%';
```

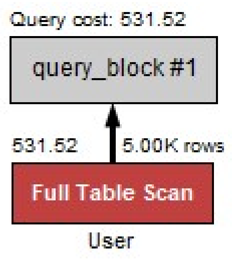
Query Plan:



c) [3 pts]

```sql
SELECT * FROM User WHERE last_name LIKE 'Dav%';
```

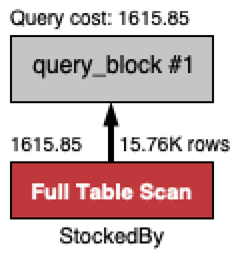
Query Plan:



d) [3 pts]

```
SELECT COUNT(*) FROM StockedBy WHERE qty = 40;
```

Query Plan:



2. [10 pts] Now create secondary indexes (which are B+ trees, under the hood of MySQL) on the User.last_name attribute and StockedBy.qty *separately*. (I.e., create two indexes, one per table.) Paste your CREATE INDEX statements below.

   CREATE INDEX lastName

   ON User(last_name)

   USING BTREE;

   CREATE INDEX stockedby_qty

   ON StockedBy(qty)

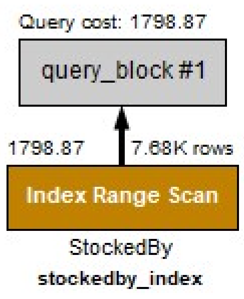
   USING BTREE;

3. [12 pts] Re-"explain" the queries in Q1 and **indicate whether the indexes you created in Q2 are used**, and if so, **whether the uses are index-only plans or not**. Copy and paste the query plan after each query, as before.

   a) [3 pts]

```
SELECT * FROM StockedBy WHERE qty BETWEEN 30 AND 35;
```

Query Plan:



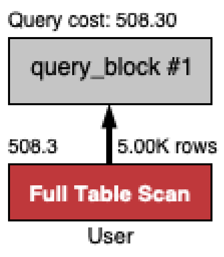
Is index used? (y/n) : Y

Is the plan index only? (y/n) : N

b) [3 pts]

```
SELECT * FROM User WHERE last_name LIKE '%pot%';
```

Query Plan:



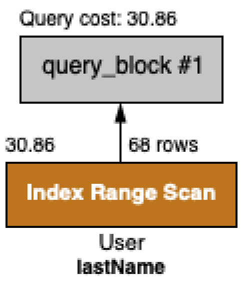
Is index used? (y/n) : N

Is the plan index only? (y/n) : N

c) [3 pts]

```
SELECT * FROM User WHERE last_name LIKE 'Dav%';
```

Query Plan:



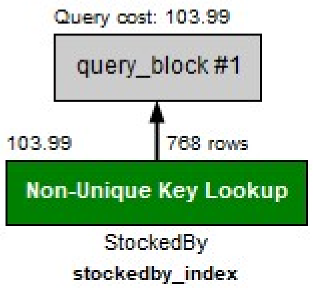
Is index used? (y/n) : Y

Is the plan index only? (y/n) : N

d) [3 pts]

```
SELECT COUNT(*) FROM StockedBy WHERE qty = 40;
```

Query Plan:



Is index used? (y/n) : Y

Is the plan index only? (y/n) : Y

4. [21 pts] Examine **each** of the above queries **with** and **without** the use of an index. Please **briefly** answer the following questions after pondering what you have seen.

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 total number of records in the file.)

Is an index useful? (y/n) : Y

Explanation: Yes because the index groups the data by an attribute, and if that attribute is what the table is indexed by, then because the data is connected at the leaf nodes, we can easily find all the data that satisfies the query.

b) [7 pts] For each of the LIKE queries (**b** and **c**), explain **whether** an index is useful and **why** or **why not** (<= 2 sentences per query).

a) Is an index useful? (y/n) : N

Explanation: No because the desired query result contains the letters 'pot' in a location unknown within the last name of each person. Indexing would not be cost efficient because there is still a significant amount of checking that has to be done.

b) Is an index useful? (y/n) : Y

Explanation: Yes because indexing would make the desired query result easier to obtain considering only the first three letters of the string are checked.

c) [7 pts] For equality queries (e.g., query **d**), explain **whether** an index is useful and **why** (assuming the number of selected result records is extremely small compared to the number of records in the file).

Is an index useful? (y/n) : Y

Explanation: Yes because by putting the data in index order, it will help us to more efficiently find unique values within the record without traversing through each record in the data file.

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.

Query 1)

```
SELECT * FROM Orders WHERE time_placed = '2020-05-04 15:40:51' AND
                           pickup_time = '2020-05-04 18:19:17';
```

Query 2)

```
SELECT * FROM Orders WHERE pickup_time= '2020-05-04 18:19:17';
```

a) [5 pts] Create a composite index on the attributes time_placed and pickup_time (*in that order!*) of the Orders table. Paste your CREATE INDEX statement below.

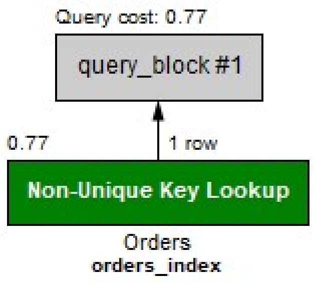
CREATE INDEX orders_index
ON orders(time_placed, pickup_time)
USING BTREE;

b) [6 pts] 'Explain' the queries 1) and 2) above. Copy and paste the query plan of each query into your response, as before. Be sure to look carefully at each plan.

Query 1)

```
SELECT * FROM Orders WHERE time_placed = '2020-05-04 15:40:51' AND
                           pickup_time = '2020-05-04 18:19:17';
```

Query Plan:

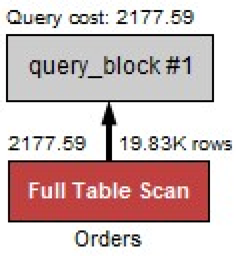


The query is traversing through all records with the specific time_placed and pick_time. The composite key is causing the result query to be ordered by time_placed with the pickup_time attached. Non-unique key look up indicates that the amount of matching rows is small (it is only returning 1 row), thus the cost is small (0.77).

Query 2)

```sql
SELECT * FROM Orders WHERE pickup_time= '2020-05-04 18:19:17';
```

Query Plan:



The query is traversing through all records in the Orders table with the specific pickup_time and is also ordered by the pickup_time. The full table scan means that the records were traversed in a sequential order to check for the condition of pickup_time. This query was costly because the large amount of rows returned (19.83k).

c) [10 pts] For each query, explain whether the composite index is **used or not** and **why**.

Query 1)

Is index used? (y/n) : Y

Explanation: Yes because the search was conducted by first finding the data that satisfies the time_placed equality and then within those orders finding the data that also satisfied the pickup_time equality.

Query 2)

Is index used? (y/n) : N

Explanation: No because the query traversed through many records by a non-unique value.

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** the index should be created on and also what the best choice between a **clustered** or **unclustered** index would be. For the purpose of this question, you can assume that the system provides that choice (even if it's MySQL and it doesn't :-)).

   a) [8 pts] (Assume that the number of result records is < 1%)

   ```
   SELECT * FROM Own WHERE user_id = "4af20bf5-80f9-4d74-8536-169fdf32ce23";
   ```

   Is index useful? (y/n) : Y

   If index is useful, what table + attributes should be indexed?: index should be made on the Owns table and the user_id attribute.

   If index is useful, should the index be clustered or unclustered?:

   Unclustered because there is no point in ordering by user_id which contains a unique combination of numbers, letters, and symbols. Ordering or grouping by user_id will not make it more organized and will make it cost more to attempt to order it.

   b) [8 pts]

   ```
   SELECT category, COUNT(*) as cnt FROM Products GROUP BY category;
   ```

   Is index useful? (y/n) : N

   If index is useful, what table + attributes should be indexed?:

   If index is useful, should the index be clustered or unclustered?:

   c) [8 pts]

   ```
   SELECT * FROM Orders;
   ```

   Is index useful? (y/n) : N

   If index is useful, what table + attributes should be indexed?:

   If index is useful, should the index be clustered or unclustered?: