# Introduction to Database Systems
# Practice Questions for LO4

Eleni Tzirita Zacharatou

## August 2020: Adapted Question

Consider the Likes relation from the social media database:

Likes(userID, postID)

Assume here that the Likes relation has billions of entries. Now consider the following three SQL queries:

**Query 1:**
```
select count(*)
from Likes
where userID = 25;
```
unclustered covering index on Likes(userID)

**Query 2:**
```
select max(postID)
from Likes
where userID = 25;
```
a covering index or a clustered index on Likes(userID, postID)

**Query 3:**
```
select count(distinct userID)
from Likes
where postID = (select max(postID) from Likes);
```
A clustered index on Likes(postID, userID)

Answer each of the following questions:

(a) Indicate for each query whether a clustered index should be defined (i.e., would be preferable to a non-clustered index or no index at all). Explain your answer and define the indexes you consider.

(b) Indicate for each query whether a covering index could be defined (i.e., would be preferable to a clustered index). Explain your answer and define the indexes you consider.

(c) Considering all three queries, which clustered index would you define on Likes? Explain your answer.

## August 2020: Example Answer

(a) (Q1) Since we only wish to know the number of records, not their contents, an index on Likes(userID) would be covering, and a clustered index would not improve performance. (Q2) A clustered index on Likes(userID) would perform better than an unclustered index on Likes(userID), as it would result in (fewer) sequential reads, while the unlustered index would have (more) random reads. Since the relation only has two columns, a covering index or a clustered index on Likes(userID, postID) would both be optimal for this query. (Q3) Since this query is effectively two queries, we need to consider them separately. For the inner query, an index on Likes(postID) is effective: since the query only considers one column and returns one value, any index will be covering, so creating a clustered index does not improve performance. The outer query can be expected to return few rows, but a clustered index on Likes(postID) would perform better than an unclustered index, as decribed above, and a clustered index on Likes(postID, userID) would be optimal for this query. Note that the index for the outer query could also be used for the inner query.

(b) (Q1) The proposed index is covering. (Q2) A covering index on Likes(userID, postID) would perform the same as a clustering index. (Q3) A covering index on Likes(postID, userID) would perform the same as a clustering index, and could in fact be used .

(c) In this case, there are two candidates for clustered index: Likes(userID, postID), which is more effective for Q2, or Likes(postID, userID), which is more effective for Q3. We can choose either, and then make a covering index for the other.

*Note: It is important to understand why Likes(postID, userID) is not effective for Q2, and why Likes(userID, postID) is not effective for Q3!*

## January 2021: Adapted Question

Consider the following large relation with web logging data:

Requests(ID, userID, baseURL, options, timestamp, <detailed logging info>)

Assume that the relation has request data for the last 5 years with 10 billion requests, that requests are uniformly distributed over each day of those 5 years, that there are 100 million users, but only 10 possible base URLs (the variability is in the options). Now consider the following three SQL queries:

**Query 1:**
```
select *
from Requests
where baseURL = 'http://www.ThisURLisOneOfTheTen.com';
```
clustered index on Requests(baseURL)

**Query 2:**
```
select max(ID)
from Requests;
```
covering index on Requests(ID)

**Query 3:**
```
select count(distinct userID)
from Requests
where options like '%include%';
```
covering index on Requests(options, userID)

Answer each of the following questions:

(a) Indicate for each query whether a clustered index should be defined (i.e., would be preferable to a non-clustered index or no index at all). Explain your answer and define the indexes you consider.

(b) Indicate for each query whether a covering index could be defined (i.e., would be preferable to a clustered index). Explain your answer and define the indexes you consider.

(c) Considering all three queries, which clustered index would you define on Requests? Explain your answer.

## January 2021: Example Answer

(a) (Q1) A clustered index on Requests(baseURL) would result in reading 10% of the relation sequentially, while an unclustered index would result in very many random reads. Such an index is therefore better than both unclustered index and no index. (Q2) This query returns a single value, which can be read from the index itself. A clustered index is therefore not useful. (Q3) Due to the nature of the LIKE expression, starting with %, a clustered index could not reduce the amount of data read, and hence is no better than no index.

are all index are covering index: depend on query

(b) (Q1) Since whole records are returned, a covering index does not improve over a clustered index. (Q2) An index on Requests(ID) is already covering. (Q3) A covering index on Requests(options, userID) would result in reading much less data than having no index, and is thus preferred.

(c) Since (a) covering indexes can be used effectively for queries Q2 and Q3, but not for Q1, while (b) a clustered index is very effective for Q1, the best clustered index for these three queries would be on Requests(baseURL).

## March 2021: Adapted Question

Consider the following large relation with phone call data:

Calls(fromnum, tonum, timestamp, duration, <detailed call info>)

Assume that the relation has call data for the last 10 years with 10 billion requests, that calls are uniformly distributed over each day of those 10 years, that there are 100 million telephone numbers (tonum and fromnum) that are used uniformly. Now consider the following three SQL queries:

**Query 1:**

```
select fromnum
from Calls
where duration = (select max(duration) from Calls);
```

covering (unclustered) index on Calls(duration, fromnum)

covering index on Calls(duration)

**Query 2:**

```
select sum(duration)
from Calls
where fromnum = 12345678;
```

covering index on Calls(fromnum, duration)

**Query 3:**

```
select *
from Calls
where tonum = 87654321 and fromnum = 12345678;
```

clustered index on Calls(tonum, fromnum)

Answer each of the following questions:

(a) Indicate for each query whether a clustered index should be defined (i.e., would be preferable to a non-clustered index or no index at all). Explain your answer and define the indexes you consider.

(b) Indicate for each query whether a covering index could be defined (i.e., would be preferable to a clustered index). Explain your answer and define the indexes you consider.

(c) Considering all three queries, which clustered index would you define on Calls? Explain your answer.

## March 2021: Example Answer

(a) (Q1) Since this query is effectively two queries, we need to consider them separately. For the inner query, an index on Calls(duration) is effective. Since the query only considers one column and returns one value, any index will be covering, so creating a clustered index does not improve performance. The outer query can be expected to return very few rows, and hence a clustered index does not improve performance much over an unclustered index. An unclustered index on Calls(duration), or a covering (unclustered) index on Calls(duration, fromnum) would be preferable. (Q2) A clustered index on Calls(fromnum) would perform better than an unclustered index, as with an unclustered index, each row would result in a random disk read. (Q3) This query should return relatively few calls on average, given the assumptions. A clustered index on Calls(tonum, fromnum) would return all rows in one disk read (or a few sequential reads), while an unclustered index on Calls(tonum, fromnum) would result in one random disk read for each, so the clustered index is preferred. If indexes on only one attribute are considered, then the clustered index would perform much better than an unclustered index.

(b) (Q1) As discussed above, the index for the inner query will be covering, as it contains only one attribute. A covering index for the outer query, on Calls(duration, fromnum), will perform better than a clustered index. (Q2) A covering index on Calls(fromnum, duration) would outperform a clustered index. (Q3) Since all attributes are returned by this query, a covering index would need to replicate the whole relation, and hence a clustered index is a better choice.

(c) Since (a) covering indexes can be used effectively for queries Q1 and Q2, but not for Q3, while (b) a clustered index is effective for Q3, the best clustered index for these three queries would be on Calls(tonum, fromnum).