6.5

Test 1
Database, advanced, 22.10.2019.
(10 12 + 5 = 27 points)

Name: EKART CABA
Neptun-code: ZWPHKP

A. Complete the following sentences/answer the following questions! (10 points)

The cost of a search in B+ tree based index depends on the depth of the tree, which in turn, can be estimated by taking logarithm (some function) of the number of values.

The size of the B+ tree based index is the same size of the table.

The size of one node of a B+ tree based index is around 4/8/32 Bytes. This is determined by size of memory block or HDD.

A B+ tree based index does not support the following types of queries efficiently:
1. lot of columns selected
2. —

List examples of pieces of information, the Cost Based Query Optimizer can use:
1. index usage
2. join order
3. join execution

Test 1
Database, advanced, 22.10.2019.
B. (12 points)

Name: EKART CABA
Neptun-code: ZULPMT

8.5

The following table is given. (A red star indicates a NOT NULL constraint. The created index with its attributes is also included.) There are also some queries and some query plans.

1. **Which query plan belongs to which query? Explain your choice!**
2. **How are the query plans to be interpreted? (Is an index used – if yes why?, how? – is the table accessed – why?)**
3. **If the amount of data increases by a factor of 1000, how will the cost of the query (~response time) change? (For some queries, no exact value, just a description, trend is expected.)**

LUKACS.AUDIO_II	
* ID	NUMBER (*,0)
TITLE	VARCHAR2 (255 BYTE)
AUTHOR	VARCHAR2 (255 BYTE)
DESCRIPTION	VARCHAR2 (2000 BYTE)
USER_ID	NUMBER (*,0)
SRC_MD5	VARCHAR2 (64 BYTE)
CREATED_AT	TIMESTAMP WITH TIME ZONE
UPDATED_AT	TIMESTAMP WITH TIME ZONE
EXPIRES_AT	TIMESTAMP WITH TIME ZONE
DURATION	NUMBER (*,0)
REMOVED_AT	TIMESTAMP WITH TIME ZONE
* CREATED_AT_ZONE	NUMBER (*,0)
* UPDATED_AT_ZONE	NUMBER (*,0)
* REMOVED_AT_ZONE	NUMBER (*,0)
* EXPIRES_AT_ZONE	NUMBER (*,0)
AUDIO_II_TITLE_AUTHOR_IX (TITLE, AUTHOR)	

```
SELECT COUNT(description)
FROM audio_ii
WHERE title > 'Y' AND title < 'Z';
```

=> (III)

=> It can check the index
There are no valid rows in the INDEX with these parameters, but these columns are not necessarily NOT NULL -s so the table is also checked for NULL values.
↳ ✓ INDEX ✓ TABLE
-> As it uses both the INDEX and the TABLE the cost increases even more logarithmically for the INDEX, linearly for the TABLE

```
SELECT COUNT(description)
FROM audio_ii
WHERE title > 'E' AND title < 'Z';
```

=> (I.) Simple range scan, will only use the INDEX,
↳ ✓ INDEX ✗ TABLE
-> Same as I describe in the 4th query. Only using the index so the cost increasing logarithmically.

```
SELECT COUNT(*)
FROM audio_ii
WHERE author LIKE 'X%';
```

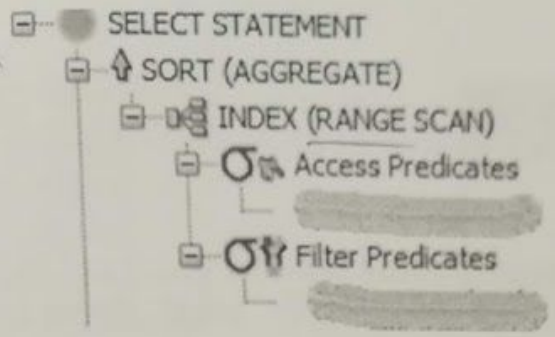
INDEX FAST FULL SCAN

=> (IV.) Needs FULL scan, because of the filter does not contain the first key of the index.
↳ ✗ INDEX ✓ TABLE
Since the query use only the table and not the index, the cost increases linearly
=> x1000

```
SELECT COUNT(*)
FROM audio_ii
WHERE title LIKE 'X%';
```

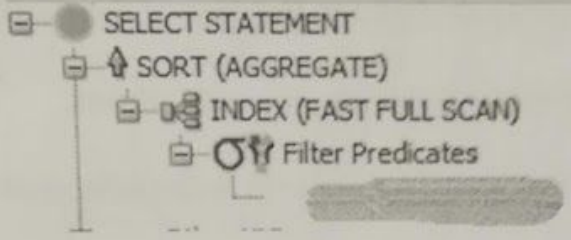
=> (II.) It can use the index, but still has to go through all elements to count them. => FAST FULL SCAN
RANGE
↳ ✓ INDEX ✗ TABLE
logarithmic

(I)



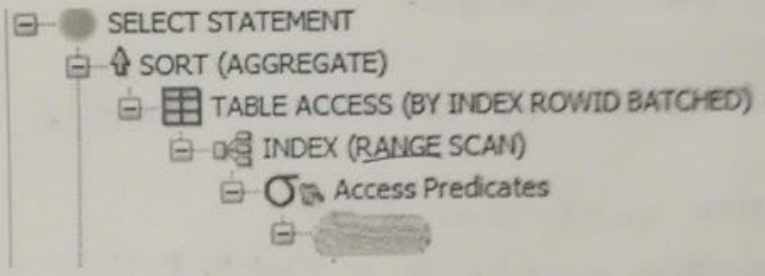
AUDIO_ST_TITLE_AUTHOR_IX

(II)



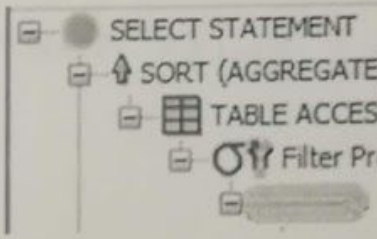
AUDIO_ST_TITLE_AUTHOR_IX

(III)



AUDIO_ST
AUDIO_ST_TITLE_AUTHOR_IX

(IV)



AUDIO_ST

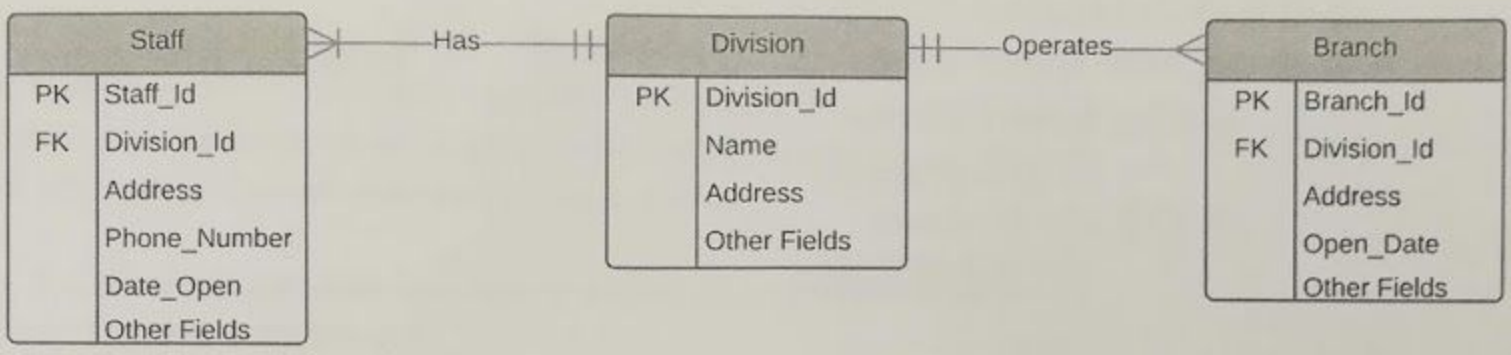
1

Basically I used this column to answer the 1st question, and the FULL, FAST SCAN, AND RANGE SCAN columns

0.5

(C) (5 points)

The following three tables are given.



Write an SQL statement, that calculates for each division the size of the staff and the number of branches.

- **We only need three columns in the result: division_id, staff_count, branch_count.**
- **The values should also be correct if there is no staff for a division, and/or there is no branch for a division.**

SELECT ~~COUNT (DIVISION_ID) RECORDS ALIAS~~ 1

SELECT DIVISION.DIVISION_ID, COUNT(STAFF.STAFF_ID), COUNT(BRANCH. BRANCH

FROM DIVISION INNER JOIN STAFF

ON DIVISION.DIVISION_ID = STAFF.DIVISION_ID

INNER JOIN BRANCH

ON ~~DIVISION~~.
DIVISION.DIVISION_ID = BRANCH.BRANCH_ID;