Test 1 Name:
Database, advanced, 18.10.2018. Neptun-code:

Összpontszám: 43 (17+7+5+11+3)

A. "Multiple" ChoiceTest (17 pont)

1. The depth of a typical B+ tree based index is around 30.
2. The size of a node in a typical B+ tree based index corresponds to the size of a block on the hdd.
3. The typical size of block on the hdd is around 1kB.
4. A typical (B+ tree based) database index supports well queries with a high selectivity.
5. A B+ tree based index I(A,B) created for the table T(A,B,C) can be used for range queries on attribute A.
6. A B+ tree based index can only be used for exact match queries.
7. A B+ tree based index I(A,B) created for the table T(C,B,A) can be used for range query conditions on attribute A.
8. A B+ tree based index I(A,B) created for the table T(C,B,A) can be used for an exact match query condition on attribute B.
9. A function based join condition is typically executed by the sort-merge join method.
10. The join condition T1.A < T2.B is typically executed by the hash join method.
11. If a table grows by a factor of 10 000, the corresponding B+ tree based index will typically require 10 times more space.
12. A B+ tree based in index supports well queries selecting/aggregating a large number of records (queries typical for data warehouses).
13. If an INNER JOIN is replaced by an RIGHT OUTER JOIN in a query, the number of the records in the result may increase.
14. If an INNER JOIN is replaced by a FULL OUTER JOIN in a query, the number of the records in the result always increases.
15. Schema constraints (e.g. foreign key, not null) make query execution typically more expensive.
16. The joins in a query are executed in exactly the same order as they appear in the SELECT command.
17. During cost-based query optimization, the result of the query may change slightly.

	Igen	Nem
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		

B. Cost based query optimization (7 points)

The goal of the cost based query optimization is to find an execution plan with a relatively low resource requirement for a declarative query given in SQL. After a syntax check, the query is transformed into an internal representation, called …… On this, several transformations can be done, which change …… but do not affect …….

As a rule of thumb, intermediate results shall be……..
Examples of decisions a cost based query optimizer makes:

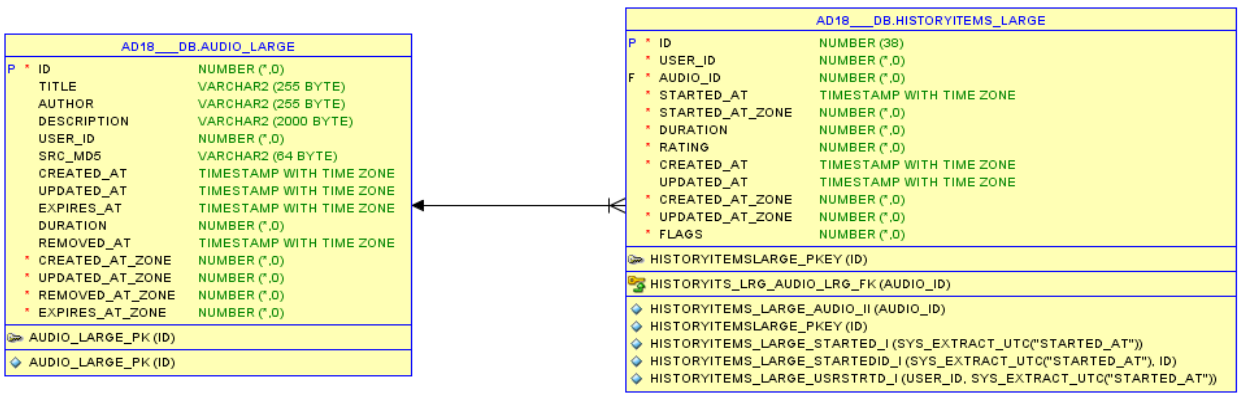
(1) …… (2) …… (3) ……

C. Index usage (5 points)

Explain index usage (and query costs) as the selectivity of a query changes. (What did you learn from the relevant tasks?)

D. (11 points)

The following 2 tables are given. (Primary and foreign keys are noted with P and F. A red star indicates a NOT NULL constraint. Indexes with their attributes are also included.)



The following queries and the execution plans created by Oracle are given:

1

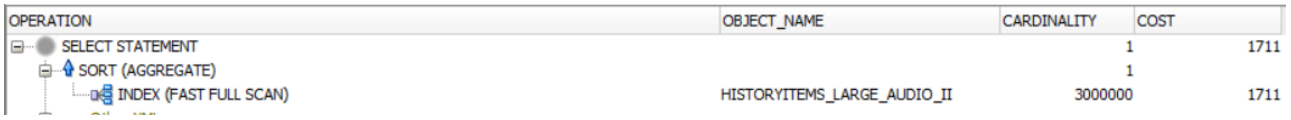
```
SELECT COUNT(*)
FROM audio_large
INNER JOIN historyitems_large
ON audio_large.id = historyitems_large.audio_id;
```

OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	1711
SORT (AGGREGATE)		1	
INDEX (FAST FULL SCAN)	HISTORYITEMS_LARGE_AUDIO_II	3000000	1711

1.1 How can it happen, that the join is not executed? (2p)
1.2 Why is it possible to answer the query by accessing only the index historyitems_large_audio_ii? (2p)
1.3 How is the index used? If the table would grow by a factor of 1000, how would "COST" change? Why (2p)

2

```
SELECT COUNT(historyitems_large.rating)
FROM audio_large
INNER JOIN historyitems_large
ON audio_large.id = historyitems_large.audio_id;
```

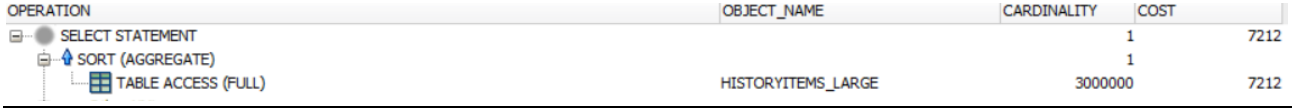


OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	1711
SORT (AGGREGATE)		1	
INDEX (FAST FULL SCAN)	HISTORYITEMS_LARGE_AUDIO_II	3000000	1711

2.1 Why is it possible to answer the query by accessing only the index historyitems_large_audio_ii (instead of the table historyitems_large)? (2p)

3

```
SELECT AVG(historyitems_large.rating)
FROM audio_large
INNER JOIN historyitems_large
ON audio_large.id = historyitems_large.audio_id;
```



OPERATION	OBJECT_NAME	CARDINALITY	COST
SELECT STATEMENT		1	7212
SORT (AGGREGATE)		1	
TABLE ACCESS (FULL)	HISTORYITEMS_LARGE	3000000	7212

3.1 What is the difference in the execution plan with respect to 1? (1)
3.2 Why did the execution plan change? (2p)

E. Covering index (3 points**)**

Create a covering index for the following query! (SQL command for creating the index.) Explain your solution!

```
SELECT user_id, AVG(rating)
FROM historyitems_large
WHERE user_id < 10
GROUP BY user_id;
```