Group **A**

Please fill in your name and registration number (Matrikelnr.) **immediately**.

Exam 1 ON	**SOLUTION KEY**	02.05.2019
	Advanced Database Systems (**184.780**)	**GROUP A**
Matrikelnr.	Last Name	First Name

Duration: 60 minutes. Provide the solutions on the designated pages. **Good Luck!**

Question 1: (4)

For each of the statements below, decide if it is true or false and tick the corresponding circle. You get +1 credit for each correct answer, −1 credit for each wrong answer and 0 credit if you leave the answer open. In total, you always get ≥ 0 credits on the entire exam question 1.

1. Suppose that there exists a hash index with static hashing on some integer attribute A. Then it always suffices to read 1 page in order to check if a tuple satisfying the condition $A = 17$ exists. true ◯ false ⊗
2. If there are m buffer frames available for sorting, then the DBMS can combine up to $m + 1$ runs in each pass of the merge phase. true ◯ false ⊗
3. For all relations R and S and for all selection predicates P, the following equalities hold: $\sigma_P(R \cup S) = \sigma_P(R) \cup \sigma_P(S)$ and $\sigma_P(R \cap S) = \sigma_P(R) \cap \sigma_P(S)$. true ⊗ false ◯
4. For all relations R and S and for all selection predicates P, the following equality holds: $\sigma_P(R \bowtie S) = \sigma_P(R) \bowtie \sigma_P(S)$. true ◯ false ⊗

Question 2: (4)

Suppose that the database of a car rental company contains the following relations:

`Cars(Id, Brand, Category, KW, Price)` (short c)
`Assistants(Id, Name, Salary, Address)` (short a)
`Rentings(Id, CarId, CustomerId, AssistantId, BeginDate, EndDate)` (short r).

Further, suppose that the following query has to be processed:

```
SELECT a.Name, a.Salary, c.Brand, c.Price
FROM   Cars c, Assistants a, Rentings r
WHERE  c.Price  > 150
AND    a.Salary < 1200
AND    c.Id = r.CarId
AND    a.Id = r.AssistantId
```

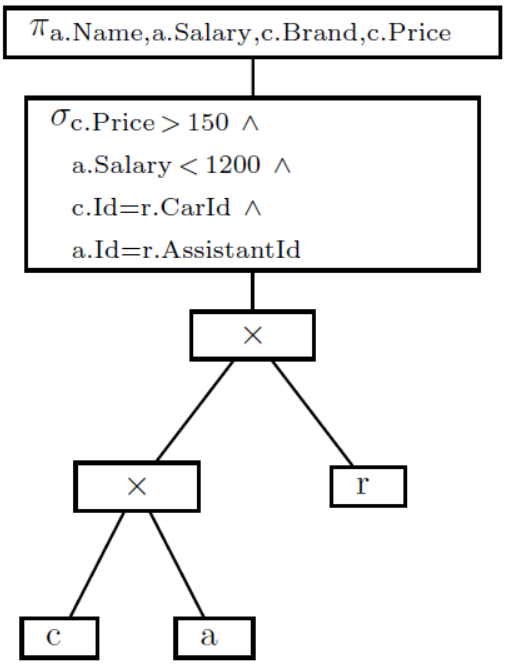
i.e., we are interested in information on assistants who let expensive cars and whose salary is rather low.

a) In the first box below, draw the query tree of the canonical translation. To save space, you may use the abbreviations c, a, and r, for the relations Cars, Assistants, and Rentings, respectively.

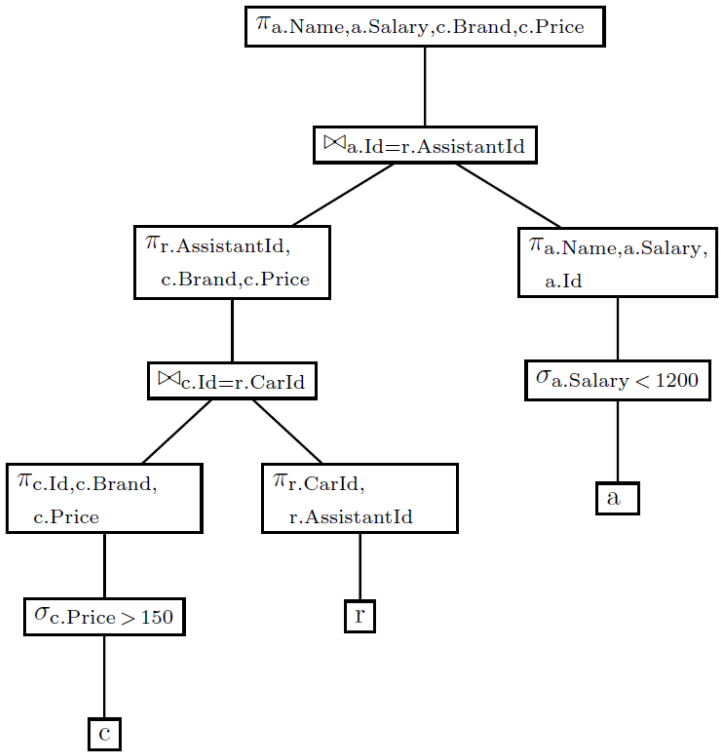
b) In the second box below, draw the query tree of the optimized Relational Algebra expression. For the optimization, apply the following rules:

- Push selections as deep in the tree as possible,
- project out attributes as soon as they are not needed anymore,
- replace cross products by joins.

(a) Canonical Translation:



(b) Optimized Expression:



Question 3: (3)

Consider a magnetic disk with the following characteristics:

- Average seek time: 3ms
- Track-to-track seek time: 0.2ms
- Transfer time (for 1 block): 1ms
- Rotational speed: 5000 rpm
- Block size: 15 000 byte
- Average track size 450 000 byte

Assume a file `books` with 1400 fixed-length records with a record size of 2000 bytes is stored on the disk in an unspanned organization.

a) Compute the the time required to read the whole `books` file from disk if you can make no assumptions on the layout of its blocks on the disk.

$$
\begin{aligned}
&bfr = \lfloor 15000/2000 \rfloor = 7 \\
&blocks = 1400/bfr = 200 \\
&t_r = 1/2 \cdot 1/5000 \cdot 60 \cdot 1000 = 6ms \\
&t_a = t_s + t_{tr} + t_r = 3ms + 1ms + 6ms = 10ms \\
&access\ \ time = t_a \cdot 200 = 2000ms = 2s
\end{aligned}
$$

b) Now, assume the file `books` is stored on consecutive blocks of the disk. Compute the time required to read the whole `books` file from disk again under these new assumptions. (For simplicity you can also assume that no rotational delay occurs during track changes.)

$$
\begin{aligned}
&block\ \ per\ \ track = \frac{15000}{450000} = 30 \\
&tracks\ \ to\ \ read = \lceil 200/30 \rceil = 7 \\
&access\ \ time = t_s + t_r + 200t_{tr} + 6 \cdot t_{t2t} \\
&\qquad = 9ms + 200 \cdot 1ms + 6 \cdot 0.2ms = 210.2ms
\end{aligned}
$$

c) There always exists a percentage p such that when reading $> p\%$ of a file with random access it is faster to read the entire file instead. Estimate this percentage for the given scenario. (You do not need to calculate it exactly, a rough calculation is sufficient.)

$$
\begin{aligned}
p &= \frac{\texttt{Sequential Time}}{\texttt{Random Access Time}} \\
p &= \frac{210.2}{2000} \approx 0.1 = 10\%
\end{aligned}
$$

Question 4: (5)

a) Your DBMS maintains equi-depth histograms to support selectivity calculation for query planning. In particular, for the column `credits` of **integers** of a table `students` with 5000 rows, the following 4 values divide the column values into 5 groups of equal size: $34, 61, 181, 320$. Furthermore, the maximum value present in the column `credits` is known to be 500. **Assume that the dividing values in the histogram are always part of the left bucket, i.e., the value** 34 **is in the first bucket,** 61 **in the second, and so on.**

Estimate the selectivity for the following two predicates as accurately as possible. Assume that values are evenly distributed inside the buckets.

i) `credits` $\leq$ `410`
ii) `credits > 141`

i) votes $\leq$ 410 : 4/5 + part of last bucket ($\frac{1}{5} \cdot r$).

$$r = 1 - \frac{500 - 410}{500 - 320} = 0,5$$

Thus, the selectivity is $\frac{4.5}{5} (= 0.9)$.

ii) votes > 141: 2/5 + part of 3rd bucket ($\frac{1}{5} \cdot r$).

$$r = \frac{181 - 141}{181 - 61} = 40/120 = 1/3$$

b) You are given the following query in relational algebra, with relation sizes and *uncorrelated* selectivities as in the tables below. Assume that S and T only share the column name a which is a foreign key from S to T. T and U only share the column name b which is a foreign key from T to U. S and U do not share any column names.

$$\sigma_\varphi S \bowtie T \bowtie \sigma_\psi U$$

Estimate the number of tuples in the result and determine the join order that minimizes the size of the intermediate results. *Don't forget about the uniformity assumption!*

Relation	Size
S	400
T	10 000
U	20

Selection	Selectivity
σ_φ	0.25
σ_ψ	0.6

Three candidates for first join, compute all result sizes to decide the order:
$|\sigma_\varphi S \bowtie \sigma_\psi U| = 1200$ No attributs in common $\Rightarrow$ cross product
$|T \bowtie \sigma_\psi U| = \sigma_\psi |T| = 6000$
Foreign key & Uniformity $\Rightarrow$ only 60% of the keys referenced in T remain after σ_ψ.

$|\sigma_\varphi S \bowtie T| = \sigma_\varphi |S| = 100$ Smallest $\Rightarrow$ first join to compute

Final result size:
$|(\sigma_\varphi S \bowtie T) \bowtie \sigma_\psi U| = \sigma_\psi |\sigma_\varphi S \bowtie T| = 60$

Therefore, first join T with S, then join in U. Estimated output size is 60 tuples. Argument for the computation of the final size is the same as with the size of $T \bowtie \sigma_\psi U$.

Question 5: (4)

As so often, you are casually discussing database systems with some colleagues at a party. The night turns serious as one of your colleagues suddenly claims:

"If I join a very small relation R with a very large relation S, there are B^+-tree indices on all join attributes and I expect that the result $R \bowtie S$ will be empty, then I should always use a sort-merge join."

Do you agree? Briefly argue your position.

(You should assume that *very large* relations don't fit into the buffer, while *very small* relations always do.)

Good points to discuss:

- Index nested loop join runs in $O(|R| \log |S|)$. Is this better than $O(|R| + |S|)$? It depends on the concrete sizes of the very small relation and the very large relation.
- To test for join partners, one may traverse the leaf nodes of the B^+-tree rather than the data file. In general, the number of pages of the leaf nodes of a B^+-tree should be smaller than the number of pages of the corresponding relation.
- Why are other join implementations most probably worse?
  - (hypbrid) hash join: we may assume that the small relation R entirely fits into memory. Hence, the number of disk I/O is $p_R + p_S$. Asymptotically this is the same as the sort merge join. However, why do the hashing if we already have the sorted attribute values in the B^+-tree indexes.
  - (block) nested loops join: again, since we may assume that the small relation R entirely fits into memory, we can choose S as the outer relation and the outer loop is iterated only once. Hence, we again end up with $p_R + p_S$ disk I/O. However, as mentioned above, the number of pages of the relations is probably bigger than the number of pages of the leaf nodes of the B^+-tree indexes.

Overall: 20 points