Databases Autumn 2025

Hand-In Exercise 5

December 7, 2025

Aiysha Frutiger
Jannick Seper
Luis Tritschler

Total Points	

Task	**Points**

Aiysha Frutiger
Jannick Seper
Luis Tritschler

Task 1 Relational Algebra and Operator Tree

We are given the relations

$$U(a, b, c), \quad V(c, d, e), \quad W(e, f, g)$$

and the following SQL query:

```
SELECT b, f
FROM (
  SELECT V.c, V.d, W.e, W.f, W.g
  FROM V, W
  WHERE V.e = W.e
    AND f = 2
    AND g = 10
)
JOIN U ON U.c = V.c
WHERE a LIKE 'Ben';
```

a. Relational algebra and algebraic operator tree

A translation of the query into relational algebra is:

$$\pi_{b,f}\Big(\sigma_{a\,\text{LIKE}\,'Ben'}\big(\big(\pi_{V.c,\,V.d,\,W.e,\,W.f,\,W.g}\big(\sigma_{V.e=W.e\,\wedge\,W.f=2\,\wedge\,W.g=10}(\,V\times W\,)\big)\big)\bowtie_{U.c=V.c}U\big)\Big).$$

The corresponding algebraic operator tree is shown in Figure 1.

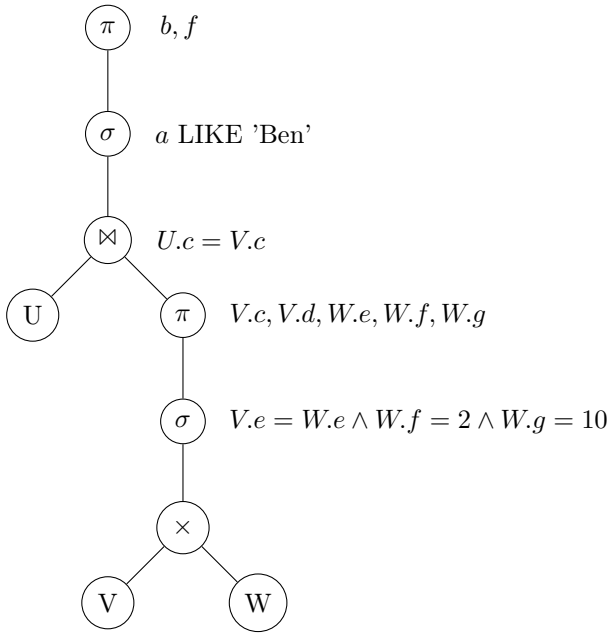

Figure 1: Algebraic operator tree for the given SQL query.

b. Query optimizer

We transform the canonical algebraic expression step by step using equivalence rules of the relational algebra. The goal is to push selections and projections as far down as possible and to replace Cartesian products plus join conditions by joins, in order to reduce the size of intermediate results.
Rule 10 – Split selection with conjunction

$$\sigma_{P_1 \wedge P_2 \wedge P_3}(R) = \sigma_{P_1}\big(\sigma_{P_2}\big(\sigma_{P_3}(R)\big)\big).$$

This rule is used to turn the single selection

$$\sigma_{V.e=W.e \wedge W.f=2 \wedge W.g=10}(V \times W)$$

into a cascade of three selections, so that the conditions can later be pushed down independently.
Rule 15 – Swap selection and Cartesian product

$$\sigma_P(R \times S) = \sigma_P(R) \times S \quad \text{if } P \text{ uses only attributes of } R.$$

This rule is used to push the selections $\sigma_{W.f=2}$ and $\sigma_{W.g=10}$ from $\sigma_{W.f=2}(\sigma_{W.g=10}(V \times W))$ down onto W, since both conditions refer only to attributes of W.
Rule 22 – Combine selection and product into join

$$\sigma_{R.A=S.B}(R \times S) = R \bowtie_{R.A=S.B} S.$$

This rule is used to replace

$$\sigma_{V.e=W.e}\big(V \times W'\big)$$

by the join

$$V \bowtie_{V.e=W.e} W',$$

where $W' = \sigma_{W.f=2}(\sigma_{W.g=10}(W))$. This eliminates the large Cartesian product in favor of a join with already filtered W.
Rule 14 – Swap selection and join

$$\sigma_P(R \bowtie S) = \sigma_P(R) \bowtie S \quad \text{if } P \text{ uses only attributes of } R.$$

This rule is used to pull the selection $\sigma_{a \text{ LIKE 'Ben'}}$ down onto U in

$$\sigma_{a \text{ LIKE 'Ben'}}\big(R \bowtie_{U.c=V.c} U\big) \;\equiv\; R \bowtie_{U.c=V.c} \sigma_{a \text{ LIKE 'Ben'}}(U),$$

since the predicate involves only attribute a of relation U.
Rule 16 – Swap projection and join (projection push-down)

$$\pi_Q(R \bowtie_P S) = \pi_Q\big(\pi_{Q_1}(R) \bowtie_P \pi_{Q_2}(S)\big),$$

where Q_1 and Q_2 contain the attributes from Q and from the join predicate P that belong to each side.
This rule is used to introduce projections on U, V, and W so that each relation only carries the attributes that are actually needed:

- result attributes: b (from U) and f (from W),
- join attributes: c on U and V, e on V and W,
- selection attributes: a on U; f and g on W.

Thus we choose the minimal attribute sets

$$U : \{a, b, c\}, \quad V : \{c, e\}, \quad W : \{e, f, g\},$$

and obtain the optimized algebraic expression

$$Q_{\text{opt}} = \pi_{b,f}\Big(\big(\pi_{c,e}(V) \bowtie_{V.e=W.e} \sigma_{W.f=2 \,\wedge\, W.g=10}\big(\pi_{e,f,g}(W)\big)\big) \bowtie_{U.c=V.c} \sigma_{a \text{ LIKE 'Ben'}}\big(\pi_{a,b,c}(U)\big)\Big).$$

In this form, all selections and projections are pushed down to the leaves, the Cartesian product is replaced by joins, and intermediate results are reduced.

The corresponding optimized operator tree is:

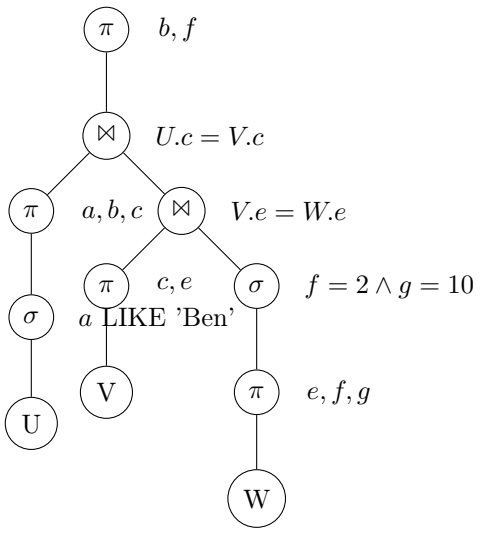

Figure 2: Optimized algebraic operator tree for the given SQL query.

c. Cardinality estimation

We assume that the attributes a, f and g are independently and uniformly distributed over 10 distinct values. Hence, a single equality condition such as $f = 2$ has selectivity $1/10$, and the conjunction $f = 2 \wedge g = 10$ has selectivity $1/10 \cdot 1/10 = 1/100$. The relation cardinalities are:

$$\mathrm{Card}(U) = 10, \quad \mathrm{Card}(V) = 100000, \quad \mathrm{Card}(W) = 1000000.$$

Non-optimized operator tree In the canonical (non-optimized) tree, the operations on V and W start with the Cartesian product:

$$V \times W, \qquad \mathrm{Card}(V \times W) = \mathrm{Card}(V) \cdot \mathrm{Card}(W) = 100000 \cdot 1000000 = 10^{11}.$$

Then the selection

$$\sigma_{V.e=W.e \,\wedge\, f=2 \,\wedge\, g=10}(V \times W)$$

is applied. We can reason about its effect as follows:

- The conditions $f = 2$ and $g = 10$ refer only to W and are independent and uniform, so together they have selectivity $1/100$. Applying them to $V \times W$ reduces the size from 10^{11} to

$$10^{11} \cdot \frac{1}{100} = 10^9.$$

- The remaining condition $V.e = W.e$ is an equality join on a uniformly distributed key. With uniformity, the probability that $V.e$ and $W.e$ match for a random pair is approximately $1/\mathrm{Card}(W)$, so we obtain

$$10^9 \cdot \frac{1}{1000000} = 1000$$

tuples after this selection.

The subsequent projection $\pi_{V.c,V.d,W.e,W.f,W.g}$ does not change the number of tuples, so the intermediate result before joining with U still has about 1000 tuples. The join with U and the selection on a produce a very small final result compared to the sizes above.
The *dominant* intermediate result in the non-optimized tree is therefore the Cartesian product $V \times W$ with

$$\mathrm{Card}_{\max}^{(\text{non-opt})} \approx 10^{11} \text{ tuples.}$$

Optimized operator tree In the optimized tree from the previous subtask, selections and projections are pushed down:

- On U: first $\pi_{a,b,c}(U)$ keeps 10 tuples, then $\sigma_{a \text{ LIKE 'Ben'}}$ selects one out of 10 values of a:

$$\mathrm{Card}\big(\sigma_{a \text{ LIKE 'Ben'}}(\pi_{a,b,c}(U))\big) = 10 \cdot \frac{1}{10} = 1.$$

- On W: first $\pi_{e,f,g}(W)$ keeps 1000000 tuples, then $\sigma_{f=2 \wedge g=10}$ has selectivity 1/100:

$$\mathrm{Card}\big(\sigma_{f=2 \wedge g=10}(\pi_{e,f,g}(W))\big) = 1000000 \cdot \frac{1}{100} = 10000.$$

- On V: $\pi_{c,e}(V)$ keeps all 100000 tuples (only the attributes change, not the number of rows).

We then join the reduced V and W:

$$\pi_{c,e}(V) \bowtie_{V.e=W.e} \sigma_{f=2 \wedge g=10}(\pi_{e,f,g}(W)).$$

Under the uniform foreign-key assumption, the fraction of W's key values that survive the selection is

$$\frac{\mathrm{Card}(\sigma_{f=2 \wedge g=10}(W))}{\mathrm{Card}(W)} = \frac{10000}{1000000} = \frac{1}{100}.$$

Each tuple in V therefore matches with probability 1/100, so the join result has expected size

$$\mathrm{Card}(V \bowtie W') \approx \mathrm{Card}(V) \cdot \frac{1}{100} = 100000 \cdot \frac{1}{100} = 1000,$$

where $W' = \sigma_{f=2 \wedge g=10}(W)$.
Joining this intermediate result (about 1000 tuples) with $\sigma_{a \text{ LIKE 'Ben'}}(\pi_{a,b,c}(U))$ (one tuple) on c yields a very small final result. The largest intermediate results in the optimized plan are thus the base relation W with 10^6 tuples and the filtered subset of W with only

$$\mathrm{Card}\big(\sigma_{f=2 \wedge g=10}(W)\big) = 10000$$

tuples before the join.
Compared to the non-optimized tree, the *largest* intermediate result in the optimized tree is:

$$\mathrm{Card}_{\max}^{(\text{opt})} \approx 10000 \text{ tuples (on the filtered } W),$$

instead of the huge Cartesian product with $\mathrm{Card}_{\max}^{(\text{non-opt})} \approx 10^{11}$ tuples. This shows why pushing selections down and avoiding Cartesian products is crucial for query performance.

Task 2

i Parameters given in the sheet:

Parameter	Symbol	Value
Attribute size	s_a	6 B
Tuple size	s_t	12 B
Cardinality of R	Card(R)	50 000
Page size	page	8192 B
Fill degree (data pages)	f_data	0.9
Fill degree (index pages)	f_index	0.7
Page header	head	48 B
Pointer size	s_p	6 B

ii **FF(A)**
Attribute A is uniformly distributed in [0..999], therefore $\mathbf{FF(A = a) = 1/1000}$.

iii **FF(B)**
Attribute B is not uniformly distributed:
- 40 000 tuples are uniformly distributed over the values [0..99] (100 values)
The tuples are spread uniformly across 100 values so each value occurs $\frac{40\,000}{100} = 400$ times.
Therefore $\mathbf{FF(B = 10)} = \frac{\mathbf{400}}{\mathbf{50\,000}}$.
- 10 000 tuples are uniformly distributed over the values [100..999] (900 values)
Here the tuples are spread uniformly across 900 different values so each value occurs $\frac{10\,000}{900} \approx 11.11$ times.
Therefore $\mathbf{FF(B = 500)} = \frac{\mathbf{10\,000/900}}{\mathbf{50\,000}}$.

1. **No index (Layout R)**

   Page-layout: $\frac{(\text{page}-\text{head})\cdot f_\text{data}}{r_\text{avg}+p_\text{slot}} \Longrightarrow \underline{x = 407}$
   (with $r_\text{avg} = s_t$ and $p_\text{slot} = s_p$)

   Number of data pages: $\lceil Card(R)/x \rceil \Longrightarrow \underline{NPages(R) = 123}$

   Since there is no index: $\Longrightarrow \mathbf{C(A = 10)} = \mathbf{C(A = 500)} = \mathbf{C(B = 10)} = \mathbf{C(B = 500)} = \mathbf{123}$

2. **Indirect B+ tree on A (Layout RA)**

   Leaf capacity: $\left\lceil \frac{(page-head-2\cdot p_{leaf})\cdot f_\text{index}}{k+r_k\cdot p_{rec}} \right\rceil \Longrightarrow \underline{t_\text{leaf} = 18}$
   (with $k = s_a$, $r_k = 50, p_{leaf} = p_{rec} = s_p$)

   Number of leaf pages: $\lceil \frac{n_k}{t_\text{leaf}} \rceil \Longrightarrow \underline{n_\text{leaf} = 56}$
   (with $n_k = 1000$)

   Inner node capacity: $\left\lceil \frac{((page-head)\cdot f_\text{index})-p}{k+p} \right\rceil \Longrightarrow \underline{e_i = 474}$
   (with $p = s_p$)

   Height: $\lceil \log_{e_i+1}(n_\text{leaf}) \rceil + 1 \Longrightarrow \mathbf{h = 2}$

   Index-only selection cost: $(h-1) + \lceil FF(A = a) \cdot n_\text{leaf} \rceil \Longrightarrow \mathbf{C(A = 10)} = \mathbf{C(A = 500)} = \mathbf{2}$

   Queries on B require table scan: $\Longrightarrow \mathbf{C(B = 10)} = \mathbf{C(B = 500)} = \mathbf{123}$

3. **Indirect B+ tree on B (Layout RB)**

   The index structure parameters are the same as RA:
   $\Longrightarrow \underline{t_{\text{leaf}} = 18,\ n_{\text{leaf}} = 56,\ e_i = 474,}$
   Height: $\lceil \log_{e_i+1}(n_{\text{leaf}}) \rceil + 1 \Longrightarrow \mathbf{h = 2}$
   Queries on A require table scan: $\Longrightarrow \mathbf{C(A = 10) = C(A = 500) = 123}$
   Index-only selection cost: $(h-1) + \lceil FF(B = b) \cdot n_{\text{leaf}} \rceil \Longrightarrow \mathbf{C(B = 10) = C(B = 500) = 2}$

4. **Two indirect indexes on A and B (Layout RAB)**

   Both RA and RB exist.
   Heights and Cost identical (to calculation of indexes):
   $\Longrightarrow \mathbf{h = 2}$
   $\Longrightarrow \mathbf{C(A = 10) = C(A = 500) = 2}$
   $\Longrightarrow \mathbf{C(B = 10) = C(B = 500) = 2}$

5. **Clustered, direct index on A (Layout RA$)**

   Means physically sorted by A $\rightarrow$ Leaf pages = table pages.
   The parameters are the same as before:
   $\Longrightarrow \underline{x = 407,\ n_{\text{leaf}} \rightarrow NPages(R) = 123,\ e_i = 474,}$
   Height: $\lceil \log_{e_i+1}(n_{\text{leaf}}) \rceil + 1 \Longrightarrow \mathbf{h = 2}$
   Cost for A: $(h-1) + \lceil FF(A = a) \cdot n_{\text{leaf}} \rceil \Longrightarrow \mathbf{C(A = 10) = C(A = 500) = 2}$
   B has no usable index: $\Longrightarrow \mathbf{C(B = 10) = C(B = 500) = 123}$

6. **Combined B+ tree on (A,B) (Layout RC)**

   Combined search key: $s_A + s_B \Longrightarrow \underline{k = 12}$.
   Number of distinct key pairs: $1000 \cdot 1000 \Longrightarrow \underline{n_k = 1\,000\,000}$.
   Average number of tuples per key pair: $\frac{Card(R)}{n_k} \Longrightarrow \underline{r_k = 0.05}$.
   Leaf capacity (page-layout indirect index): $\left\lceil \frac{(\text{page}-\text{head}-2 \cdot p_{\text{leaf}}) \cdot f_{\text{index}}}{k + r_k \cdot p_{\text{rec}}} \right\rceil \Longrightarrow \underline{t_{\text{leaf}} = 462}$.
   Number of leaf pages: $= \left\lceil \frac{n_k}{t_{\text{leaf}}} \right\rceil \Longrightarrow \underline{n_{\text{leaf}} = 2165}$
   Inner node capacity: $\left\lceil \frac{((\text{page}-\text{head}) \cdot f_{\text{index}}) - p}{k+p} \right\rceil \Longrightarrow \underline{e_i = 316}$.
   Height of the index: $\lceil \log_{e_i+1}(n_{\text{leaf}}) \rceil + 1 \Longrightarrow \mathbf{h = 3}$
   Cost for queries on A: $(h-1) + \lceil FF(A = a) \cdot n_{\text{leaf}} \rceil = \Longrightarrow \mathbf{C(A = 10) = C(A = 500) = 5}$.
   B alone cannot use the combined index (no leading A): $\Longrightarrow \mathbf{C(B = 10) = C(B = 500) = 123}$.

Summary:

Layout	A=10	B=10	A=500	B=500	Height
R	123	123	123	123	no index
RA	2	123	2	123	2
RB	123	2	123	2	2
RAB	2	2	2	2	2
RA$	2	123	2	123	2
RC	5	123	5	123	3

Task 3

a. Page accesses for complex query

We consider

$$Q: \texttt{SELECT COUNT(*) FROM R WHERE A = 321 AND B <> 50;}$$

We approximate the combined filter factor assuming independence and uniformity for A and B:

$$FF(A = 321 \wedge B \neq 50) = FF(A = 321) \cdot FF(B \neq 50) = \frac{1}{1000} \cdot \left(1 - \frac{1}{1000}\right) \approx 0.000999.$$

With $\text{Card}(R) = 50\,000$ this gives an expected

$$\text{Card}(R \mid A = 321 \wedge B \neq 50) \approx 50\,000 \cdot 0.000999 \approx 50$$

matching tuples.

i **R (no index)**
No index on A or $B \Rightarrow$ full table scan:

$$C_Q(\texttt{R}) = NPages(R) = \mathbf{123}.$$

ii **RA (unclustered index on A)**
Use index on A for $A = 321$. Index search cost: $C_{IS} = h = 2$ (height from Task 2). Unclustered index $\Rightarrow$ in the worst case one data page per qualifying tuple:

$$C_Q(\texttt{RA}) = C_{IS} + \#\text{data pages} \approx 2 + 50 = \mathbf{52}.$$

iii **RB (unclustered index on B)**
Condition $B \neq 50$ is almost unselective (matches nearly all tuples), so using the index would touch almost all data pages. Cheaper is a full table scan:

$$C_Q(\texttt{RB}) \approx NPages(R) = \mathbf{123}.$$

iv **RAB (indexes on A and B)**
Index on A is useful (selective), index on B is not (due to $B \neq 50$). We use the same strategy as for `RA`:

$$C_Q(\texttt{RAB}) \approx C_Q(\texttt{RA}) = \mathbf{52}.$$

v **RA$ (clustered, direct index on A)**
Tuples with the same value of A are stored contiguously. The number of tuples with $A = 321$ and $B \neq 50$ is still about 50, but all of them fit into *one* data page (each page holds 407 tuples). Hence:

$$C_Q(\texttt{RA\$}) \approx C_{IS} + \#\text{data pages} \approx 2 + 1 = \mathbf{3}.$$

Aiysha Frutiger
Jannick Seper
Luis Tritschler

vi **RC (combined index on (A,B))**

Combined key (A, B) allows us to jump directly to the range $A = 321$ in the index and then scan only the leaf entries for that value, filtering by $B \neq 50$. As in Task 2, the index cost for a query on A is

$$C(A = 321) = (h - 1) + \left\lceil FF(A = 321) \cdot n_{\text{leaf}} \right\rceil = 5,$$

and the additional $B \neq 50$ condition is applied while scanning those leaf pages. We thus approximate:

$$C_Q(\texttt{RC}) \approx \mathbf{5}.$$

Layout	# page accesses for Q
`R`	123
`RA`	52
`RB`	123
`RAB`	52
`RA$`	3
`RC`	5