



Assignment 6

Handin until: Friday, 17.06.2022, 09:00

The Summer Semester 2022 lecture evaluation is in progress — Please give us feedback. Thank you!

Please take a few minutes and report back. Every bit of feedback counts — this is especially true for text comments. The insights gained are worth their weight in gold and allow us to assess whether we are steering the DB2 lecture in the right direction and also how we can make the course even better in the remaining weeks of the semester. The feedback forms can be filled in **until June 15th**. Once again: Thank you very much!

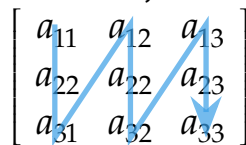
1. [15 Points] Loop Swapping Optimization

When processing a wide table stored as C array, the order of accesses to the array elements can have significant impact on performance. Consider the following table **tbl** of random integers stored *row-wise* in a C array:

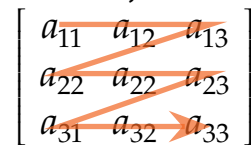
```
1 unsigned int *tbl;
2
3 /* initialize table row-by-row with random values */
4 for (r = 0; r < rows; r++)
5     for (c = 0; c < cols; c++)
6         tbl[r * cols + c] = (unsigned int) random();
```

To sum all elements of the table, the elements can be processed with two different traversal strategies:

Column-major Order



Row-major Order



The choice of loop order has significant impact on performance:

- Extend the program in **loop-swapping.c** to compute and print the overall sum of all cells in table **tbl** in two different ways: Using two nested loops processing array **tbl**
 - in *row-major* order, and
 - in *column-major* order.
- Further extend your program to measure and print the execution time of both variants.
- Compile the program with flag **-O2** and run a test with 1,000,000 rows and 100 columns. Describe the results and explain the difference in performance of both variants.

2. [15 Points] Logical Conjunction

In the lecture we learned how a MAL program would evaluate a *disjunctive* SQL predicate (connective **OR**). Based on this discussion, now implement the following SQL query featuring a *conjunctive predicate* (connective **AND**):

```
1 SELECT t.a, t.b
2 FROM ternary AS t
3 WHERE t.a % 2 = 0 AND t.c < 3;
4 --           ↑           ↑
5 --           p1         p2
```

The definition of table `ternary` with 10^7 rows is given in `/assignment06/ternary.sql`.

File `/assignment06/conjunction.mal` contains an incomplete MAL program to start with. Implement two alternative versions of the query above:

- Alternative 1: Apply predicate $p_1 = t.a \% 2$ first, filter the remaining BAT elements by predicate $p_2 = t.c < 3$ afterwards.
- Alternative 2: Apply predicate p_2 first, filter the remaining BAT elements by predicate p_1 afterwards.
- Use `mclient` with options `-l msql` and `-t clock` to run both programs on the given table. Compare the execution time of *Alternative 1* and *Alternative 2* and explain any significant difference.

Notes:

- Use `io.print(...)` to align and print the final result columns.
- Keep your MAL code comprehensible. Use comments (`#...`) and choose descriptive variable names.