ASE – Mini-Project 1

Task 1.3

How much experience in query development did the team have?

Prior the team had heard of SQL and took a database fundamentals course a few years ago. Hence, the team was quite rusty and would rate themselves at a beginner level.

Which steps did the query development involve in each subtask?

Without Chat-GPT:

After getting familiar with SQL again, we tried to destructure the given problem into smaller sub-problems. Tediously writing code, searching through forums such as StackOverflow and looking up documentation were involved to rather painstakingly get the syntax right. After a lot of trial and error, we furthermore consulted with an expert SQL writer to clarify some of the encountered issues with SQL.

With Chat-GPT:

Just prompting ChatGPT with the natural language specification to implement was already rather successful. The quality of the generated result improved after giving ChatGPT access to the database schema, which makes sense since, in a way, making the schema available shows ChatGPT which APIs are available to use.

How time-consuming and difficult was the development?

Without Chat-GPT:

We found the development of the query a rather challenging and hence time-consuming task. Taking a few person-hours. This was necessary especially since we were rather unfamiliar with the exact syntax and the declarative nature of SQL. Furthermore, typing everything out was time consuming as well.

With Chat-GPT:

The implementation time using ChatGPT was reduced since we already knew the common pitfalls from Task 1.1 and could prompt ChatGPT into the correct direction.

Looking up syntax and methods to solve subproblems was greatly reduced using ChatGPT. Still, checking for semantic correctness was still necessary and required some effort especially due to our unfamiliarity with SQL. Nevertheless, using ChatGPT felt like skipping the implementation step and going straight into correcting bugs and mistakes in the assumptions. Especially, since ChatGPT used a different approach to implement the query, reading, understanding, and checking the generated code took some time.

How was the quality of the resulting implementation?

Without Chat-GPT:

Our implementation produces the correct example output given by the chair. Since we did not put a focus on query runtime and are not that familiar with query optimization techniques, the runtime is rather slow: One query takes around 10 minutes.

With Chat-GPT:

The generated implementation produces the example output given by the chair correctly. However, since ChatGPT used SQL constructions that are not as beginner friendly and harder to read by humans, we are not as confident in how the query exactly works. A positive: The query runs a lot faster (i.e., a few minutes) than the hand written one.

How helpful was the LLM in general, and the additional explanations it provided (if any)?

We enjoyed working with the LLM since it especially helped with reducing tedious typing and syntax research tasks. We deemed the explanations to not always be helpful since they often seemed rather surface level, omitting the explanations for, to us, more complicated syntax.

How often did you have to prompt the LLM, and were there any misunderstandings (on your side or the LLM's)?

We had to prompt the LLM 5 times, i.e., 1 initial prompt and 4 corrections.

Corrections were mainly necessary due to ChatGPT making some of the same mistakes we made in our implementation by hand: not considering friend's friends, interpreting the "knows" relationship as bidirectional, minor syntax errors, and considering likes by the creator of their own posts not as foreign likes. (The latter seems to be an error in the example solutions given by the chair, since it is rather unintuitive that the creator themself is a foreigner.)

We can reduce the correction prompts by inserting the DB schema and known misunderstandings into the first prompt. Furthermore, we could do the last few fine corrections (e.g., removing a condition or fixing column prefixes) by hand or combine them into one prompt.

What degree of automation did the tools you used for query development achieve, both on their own and in combination?

Without Chat-GPT:

No automation besides auto-completion and search engines for syntax was used.

With Chat-GPT:

The query was generated automatically with minor corrections via additional prompts, which can be avoided by using the strategy provided above.

Task 2.3

How much experience in query development did the team have?

See Task 1.3. But now we have some basic knowledge due to answering task 1.

Which steps did the query development involve in each subtask?

Without Chat-GPT:

As in Task 1.3.

With Chat-GPT:

Learning from task 1 we directly prompted ChatGPT with the natural language addition specification, the database schema, and the current SQL query as well as the instruction to "add to and if necessary, adjust the current SQL query".

After inspecting the query, we deemed it correct. The execution of the query yielded the given example results. Hence, the code generation was one-shot.

How time-consuming and difficult was the development?

Without Chat-GPT:

Code from 1.1

Adding to our own solution was rather straightforward. Still, we had to look up syntax and methods we could use. Some typing and debugging later, the solution was correct and we can easily reconcile it with our mental model.

Code from 1.2:

It is rather hard to add functionality in the generated code due to two factors: Firstly, the code readability is not great. Secondly, to adjust the code, we have to build a new mental model of how it works.

With Chat-GPT:

Nearly no effort and only little time was necessary to implement the new requirements. Testing if the generated code is correct took the longest time since the runtime of the query was rather long.

How was the quality of the resulting implementation?

Without Chat-GPT:

Code from 1.1

Good code readability and still easy to extend. The runtime was, however, very long, likely due to the use of multiple joins.

Code from 1.2:

Redundancies in the code and bad readability. The performance was still better than our own original query as we picked up some seemingly faster performing tricks from Chat GPTs query.

With Chat-GPT:

Code from 1.1

Since ChatGPT adjusted very little of our original code and mostly added to it, we had some issues with this query at first. Once the functionality was correct, we had to ask it again for improvements as the query resulted in an out of memory error. The final query was still not very fast but remained easily readable.

Code from 1.2:

The implementation seemed correct (i.e., from manual code inspection and comparing example output data and computed results) and still performed decently fast. Code readability is not good.

How helpful was the LLM in general, and the additional explanations it provided (if any)?

In both cases the generated additional explanations provided only little value and were rather high level. But we actually did not necessarily need explanations since the query for adjusting the code from 1.2 worked on the first try.

How often did you have to prompt the LLM, and were there any misunderstandings (on your side or the LLM's)?

Reprompting was only sparingly necessary. This time there were no misunderstandings neither for the LLM nor us.

What degree of automation did the tools you used for query development achieve, both on their own and in combination?

See 1.3.

How could automation as provided by the LLM could be integrated holistically into an iterative query development process?

The chat interface works quite well. Integration into an IDE could look something like this: When writing code you can select code blocks, weblinks, or files as an additional context to a prompt. The LLM generates a suggestion you can accept or write another prompt to improve.

Note: OpenAI offers the Canvas function in a beta version, where after writing a prompt the generated code is shown in a sub-window and subsequently adjusted by the LLM in place when prompted to do so. Additionally, similar functionality as described above are available using GitHub Copilot. The new o1 model family provides self-reprompting and enables internal “thinking” trying to solve a problem before outputting a result. In initial experiments by us these models produced even more often than older generation models quite readable 1-shot results. For example, in one shot the o1 preview generated the working Browser extension used to copy the ChatGPT results into an .md file for this exercise.

Task 3.2

Is the execution plan generated by the LLM plausible?

The first generated query plan was not very plausible as it did not match the format provided by postgresql. After additional prompting the generated query plan had a similar format, however the content still did not seem very plausible. For example, it was a lot shorter than the Postgres query plan. Given that we have little experience with SQL it was hard to determine exact problems with the plan and formulate prompts to improve it. Overall, we assume that the plan could be plausible, just not for our query.

How helpful were additional explanations provided by the LLM (if any)?

Additional explanations helped us gain a shallow understanding of how certain SQL operations we chose might be mapped to a query plan and how it might be executed.

To what degree is query execution already automated in PostgreSQL?

From our understanding and perspective, query execution is already highly automated. The planning and optimization of queries happens automatically. Queries are parsed, the syntax validated and further processed as well.

Are there any fundamental differences between the query development tasks from Tasks 1 and 2 and the creation of a query execution plan that may have implications regarding automation?

Task 1&2 are tasks that the LLM is particularly trained for. It has seen a lot of data from this domain and is used to generating code from natural language descriptions. Additionally, there are currently no other methods known to translate a natural language prompt into working code for various domains, languages etc. with a similarly low effort. Generating a query execution plan is an uncommon task for ChatGPT to do given that it takes only one command and seconds of execution time to get a correct solution with PotgreSQL. Furthermore, there is a clear algorithm for translating SQL to a query execution plan and existing optimization infrastructure. It seems like a similar problem as compiling a higher level programming language to a lower level one: An existing compiler we can trust exists, works well, and is efficient; why use a non-deterministic, cost-intensive, and not specifically trained LLM for this task, that is also really hard to check for correctness.