Mini Project 1

Cedric Lorenz, Leonard Dreessen, Oliver Hess, and Raphael Reimann

Reflection on Manual Query Development

- Without…
  - some experience in query development and
  - access to the schema and additional context

  …manually writing the query would not have been possible

- Cycling development of smaller subparts and iteratively testing of those on the database
- Postgres query parser was very helpful while testing
- Manual development felt quite complex, time-consuming and frustrating

Reflection on Development with LLM

- Steps for LLM Generation
  - Provide task and schema to LLM (prompt engineering, < 1 min),
  - Run resulting query (sanity check 2 min)
  - Detailed check of query (10 min)
- LLM-generated query was of high quality
- The LLM was helpful, the result was achieved more quickly and with higher quality
- Follow-up prompts: One initial prompt, quick run of resulting query, then re-prompt to optimize the query which led to faster query execution
- Semi-automatic degree of automation, output and queries had to be manually checked

Generated Query Result (o1-preview)

```
  m_messageid  | total_likes | foreign_likes
---------------+-------------+---------------
  893353421092 |          29 |            28
  687194960067 |          24 |            17
  549756008897 |          31 |            21
 1030792501923 |          36 |            24
  687194873730 |          37 |            22
  893353322965 |          57 |            32
  962072897868 |          29 |            16
 1030792344343 |          32 |            17
 1030792345988 |          24 |            12
(9 rows)
```

Generated Extended Query Result (o1-preview)

```
  m_messageid  | total_likes | foreign_likes | first_foreign_liker_name |  first_foreign_like_time
---------------+-------------+---------------+--------------------------+----------------------------
  893353421092 |          29 |            28 | Francisco Sanchez        | 2012-05-09 04:10:33.585+02
  687194960067 |          24 |            17 | Jie Li                   | 2011-10-14 08:43:25.192+02
  549756008897 |          31 |            21 | Eun-Hye Lee              | 2011-05-29 04:53:09.868+02
 1030792501923 |          36 |            24 | Carlos Parra             | 2012-09-05 00:33:22.97+02
  687194873730 |          37 |            22 | Baby Yang                | 2011-10-13 06:24:52.569+02
  893353322965 |          57 |            32 | Miguel Gonzalez          | 2012-03-21 21:24:21.132+01
  962072897868 |          29 |            16 | Gunnar Johansson         | 2012-05-09 13:08:46.999+02
 1030792344343 |          32 |            17 | Albert Buysse            | 2012-08-19 18:19:10.971+02
 1030792345988 |          24 |            12 | Barry Wang               | 2012-08-05 19:56:31.196+02
(9 rows)
```

Execution Plans (LLM generated)

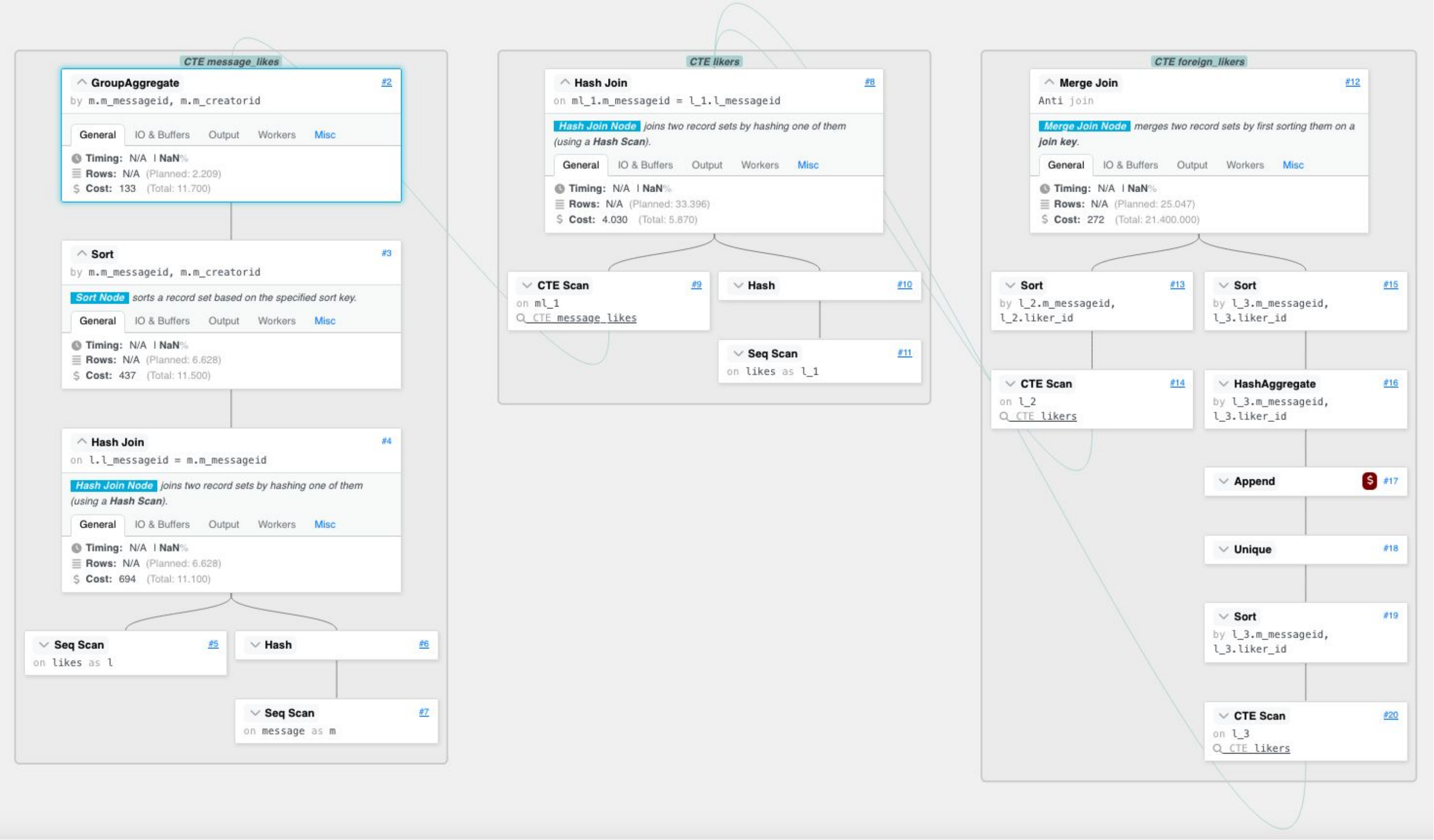
LLM-generated

```
Nested Loop Left Join  (cost=... rows=... width=...)
  Join Filter: (COALESCE(flc.foreign_like_count, 0) >= (ml.total_likes / 2.0))
  ->  Gather  (cost=... rows=... width=...)
        Workers Planned: X
        ->  Hash Join  (cost=... rows=... width=...)
              Hash Cond: (ml.m_messageid = flc.m_messageid)
              ->  HashAggregate  (cost=... rows=... width=...)
                    Group Key: ml.m_messageid
                    ->  Nested Loop  (cost=... rows=... width=...)
                          ->  HashAggregate  (cost=... rows=... width=...)
                                Group Key: m.m_messageid, m.m_creatorid
                                Filter: (COUNT(DISTINCT l.l_personid) >= 20)
                                ->  Hash Join  (cost=... rows=... width=...)
                                      Hash Cond: (l.l_messageid = m.m_messageid)
                                      ->  Seq Scan on likes l  (cost=... rows=... width=...)
                                      ->  Seq Scan on message m  (cost=... rows=... width=...)
                                            Filter: (m_length > 100)
                          ->  Index Scan using likes_pkey on likes l  (cost=... rows=1 width=...)
                                Index Cond: (l_messageid = ml.m_messageid)
              ->  HashAggregate  (cost=... rows=... width=...)
                    Group Key: flc.m_messageid
                    ->  Hash Join  (cost=... rows=... width=...)
                          Hash Cond: (fl.m_messageid = flc.m_messageid)
                          ->  HashAggregate  (cost=... rows=... width=...)
                                Group Key: fl.m_messageid, fl.liker_id
                                ->  Hash Join  (cost=... rows=... width=...)
                                      Hash Cond: (l.m_messageid = fl.m_messageid)
                                      ->  Seq Scan on likes l  (cost=... rows=... width=...)
                                      ->  Seq Scan on foreign_likers fl  (cost=... rows=... width=...)
                          ->  Seq Scan on foreign_likes_counts flc  (cost=... rows=... width=...)
  ->  Left Join  (cost=... rows=... width=...)
        ->  Nested Loop Left Join  (cost=... rows=... width=...)
              ->  Nested Loop  (cost=... rows=... width=...)
                    ->  Seq Scan on first_foreign_liker ffl  (cost=... rows=... width=...)
                    ->  Index Scan using person_pkey on person_names pn  (cost=... rows=1 width=...)
                          Index Cond: (p_personid = ffl.liker_id)
        ->  Seq Scan on person_names pn  (cost=... rows=... width=...)
```

PostgreSQL

```
Sort  (cost=21390981.25..21390983.09 rows=736 width=72)
  Sort Key: (((COALESCE(flc.foreign_like_count, '0'::bigint))::double precision / (ml.total_likes)::double precision)) DESC
  CTE message_likes
    ->  GroupAggregate  (cost=11530.16..11679.29 rows=2209 width=24)
          Group Key: m.m_messageid, m.m_creatorid
          Filter: (count(DISTINCT l.l_personid) >= 20)
          ->  Sort  (cost=11530.16..11546.73 rows=6628 width=24)
                Sort Key: m.m_messageid, m.m_creatorid
                ->  Hash Join  (cost=8840.39..11109.47 rows=6628 width=24)
                      Hash Cond: (l.l_messageid = m.m_messageid)
                      ->  Seq Scan on likes l  (cost=0.00..1792.40 rows=109440 width=16)
                      ->  Hash  (cost=8623.30..8623.30 rows=17367 width=16)
                            ->  Seq Scan on message m  (cost=0.00..8623.30 rows=17367 width=16)
                                  Filter: (m_length > 100)
  CTE likers
    ->  Hash Join  (cost=3802.40..5866.20 rows=33396 width=32)
          Hash Cond: (ml_1.m_messageid = l_1.l_messageid)
          ->  CTE Scan on message_likes ml_1  (cost=0.00..44.18 rows=2209 width=16)
          ->  Hash  (cost=1792.40..1792.40 rows=109440 width=24)
                ->  Seq Scan on likes l_1  (cost=0.00..1792.40 rows=109440 width=24)
  CTE foreign_likers
    ->  Merge Anti Join  (cost=21367216.05..21367591.16 rows=25047 width=24)
          Merge Cond: ((l_2.m_messageid = l_3.m_messageid) AND (l_2.liker_id = l_3.liker_id))
          ->  Sort  (cost=3177.19..3260.68 rows=33396 width=24)
                Sort Key: l_2.m_messageid, l_2.liker_id
                ->  CTE Scan on likers l_2  (cost=0.00..667.92 rows=33396 width=24)
          ->  Sort  (cost=21364038.85..21364058.51 rows=7864 width=16)
                Sort Key: l_3.m_messageid, l_3.liker_id
                ->  HashAggregate  (cost=21363372.73..21363451.37 rows=7864 width=16)
                      Group Key: l_3.m_messageid, l_3.liker_id
                      ->  Append  (cost=757.58..21363333.41 rows=7864 width=16)
                            ->  Unique  (cost=757.58..758.83 rows=163 width=16)
                                  ->  Sort  (cost=757.58..757.99 rows=167 width=16)
                                        Sort Key: l_3.m_messageid, l_3.liker_id
                                        ->  CTE Scan on likers l_3  (cost=0.00..751.41 rows=167 width=16)
                                              Filter: (liker_id = m_creatorid)
```

PostgreSQL execution plan visualized



CTE message_likes

GroupAggregate #2
by m.m_messageid, m.m_creatorid
Timing: N/A | NaN%
Rows: N/A (Planned: 2.209)
Cost: 133 (Total: 11.700)

Sort #3
by m.m_messageid, m.m_creatorid
Sort Node sorts a record set based on the specified sort key.
Timing: N/A | NaN%
Rows: N/A (Planned: 6.628)
Cost: 437 (Total: 11.500)

Hash Join #4
on l.l_messageid = m.m_messageid
Hash Join Node joins two record sets by hashing one of them (using a **Hash Scan**).
Timing: N/A | NaN%
Rows: N/A (Planned: 6.628)
Cost: 694 (Total: 11.100)

Seq Scan #5
on likes as l

Hash #6

Seq Scan #7
on message as m

CTE likers

Hash Join #8
on ml_1.m_messageid = l_1.l_messageid
Hash Join Node joins two record sets by hashing one of them (using a **Hash Scan**).
Timing: N/A | NaN%
Rows: N/A (Planned: 33.396)
Cost: 4.030 (Total: 5.870)

CTE Scan #9
on ml_1
CTE message_likes

Hash #10

Seq Scan #11
on likes as l_1

CTE foreign_likers

Merge Join #12
Anti join
Merge Join Node merges two record sets by first sorting them on a **join key**.
Timing: N/A | NaN%
Rows: N/A (Planned: 25.047)
Cost: 272 (Total: 21.400.000)

Sort #13
by l_2.m_messageid, l_2.liker_id

CTE Scan #14
on l_2
CTE likers

Sort #15
by l_3.m_messageid, l_3.liker_id

HashAggregate #16
by l_3.m_messageid, l_3.liker_id

Append $ #17

Unique #18

Sort #19
by l_3.m_messageid, l_3.liker_id

CTE Scan #20
on l_3
CTE likers

LLM-generated could not be parsed by explain.dalibo.com

Reflection on the LLM’s EXPLAIN output

- LLM faces greater difficulty generating precise, measurement-oriented outputs like PostgreSQL’s EXPLAIN statement, compared to the previous tasks

- LLM fails to correctly emulate PostgreSQL’s format

- LLM provides broad explanations of identified execution steps, however the output is hardly comprehensible because of lack of visualization