# Data Modeling - Project 1 - Spring 2014

*Sawyer Jager, Rees Klintworth, Derek Nordgren, Brad Steiner*

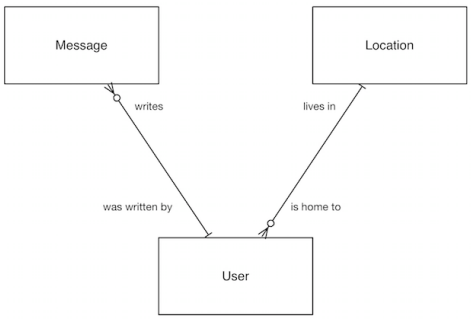
## Conceptual Modeling

- 1st conceptual model: This model provides all basic functionality without introducing any redundancies or unnecessary complexities. Users may be associated with many messages and must be associated with a single location. A location *may* be associated with a user. This relationship is not required; while a location could only be initially introduced by the presence of a user residing there, the removal of the user need not require the removal of the location record from the location table. Although this model introduces slight redundancy over the 3rd conceptual model, this model is also much less complex. For the purposes of the data we analyzed, the simpler data model made the C implementation much easier and more efficient.

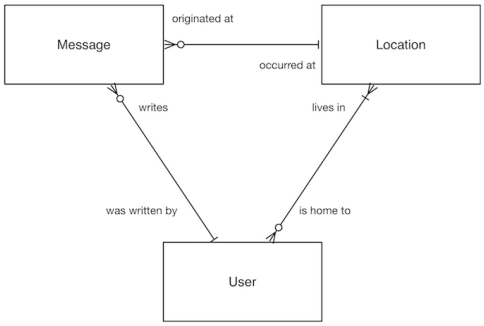
### Alternative Models

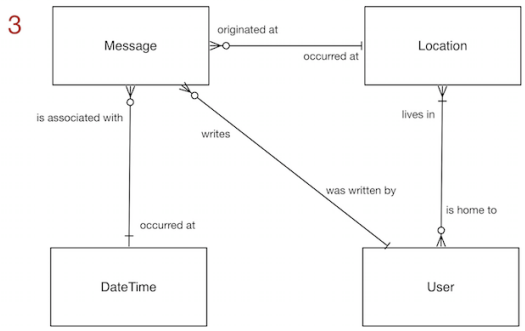
- 2nd conceptual model: in this model, messages are tied to locations. A location may be associated with many messages, but each message must have a location. A user may be associated with multiple locations, but must be associated with at least one.
- 3rd conceptual model: this model is similar to the 1st conceptual model, except the DateTime is split out from the message table. A message must be associated with one DateTime, while a DateTime *may* be associated with multiple messages. This relationship is not required; while a DateTime could only be initially introduced a message with that DateTime, the removal of the message need not require the removal of the DateTime from the DateTime table.

1




2






## Logical Modeling

### Relational Notation

**USER**(UserID, UserName, UserScreenName, *LocationID*)
**MESSAGE**(MessageID, *UserID*, Text, Time)
**LOCATION**(LocationID, City, State)

We decided to go with this implementation because it provided all the necessary functionality without making too many generalizations. It also eliminated the additional complexity of Conceptual Model 2 (presented above). Our current data only has one location per user, meaning that adding the functionality of Conceptual Model 2 would be redundant. We are asked to perform queries for messages from certain locations, but since we are restricting a user to only one location, including the link from messages to locations would result in a redundant loop.

## Preliminary Physical Modeling I

*Please see user.h, location.h, message.h for table structures.*

## Preliminary Physical Modeling II

Table 1: Operation timing

| Operation | Parameter | Operation time (s) |
|---|---|---|
| Generate tables | - | 1426.15 |
| Sort Users | locationID | 19.96 |
| Sort Users | ID | 24.31 |
| Sort Users | message_num | 25.84 |
| Sort Locations | state | 5.87 |
| Sort Locations | locationID | 6.01 |
| Sort Messages | hour:minute | 700.78 |
| Sort Messages | userID | 678.75 |

Sorting Users by locationID is required for Queries 1, 3, and 4. Sorting Locations by state is also required for Queries 1, 3 and 4. Sorting Messages by time is required for Query 2, while sorting Messages by userID is required for Queries 3 and 4. See Table 1.

## Querying

Table 2: Query timing

| Query | Sort table | By | Query time (s) | Total time (with necessary sorting included) (s) |
|---|---|---|---|---|
| 1 | Locations | state | 0.15 | 25.98 |
| " | Users | locationID | | |
| 2 | Messages | hour:minute | 0.16 | 700.94 |
| 3 | Messages | userID | 0.58 | 705.16 |
| " | Locations | state | | |
| " | Users | locationID | | |
| 4 | Locations | state | 0.92 | 705.5 |
| " | Users | locationID | | |
| " | Messages | userID | | |

## Algorithm Complexities and Comparison

Table 3: Query algorithm complexities

| Query | Intermediate Step | Complexity | Realistic Complexity |
|---|---|---|---|
| 1 | 2*log(L)* + l + 2*log(U) | O(L + log(U)) | O(l + log(U)) |
| 2 | 2*log(M) + 2m + m^2 | O(M^2) | O(mlog(m))** |
| 3 | 2*log(L) + l + 2*log(U) + u + 2*log(M) + m | O(L + U + M) | O(l + u + m) |
| 4 | 2*log(L) + l + 2*log(U) + u + 2*log(M) + m | O(L + U + M) | O(l + u + m) |

*L is the number of total locations, M is the number of total messages, U is the number of total users, l is a subset of the total locations, m is a subset of the total messages, u is a subset of the total users.
** while in the worst case M = m, in most cases m will be significantly smaller than M, so complexity will be in relation to m rather than M

quicksort of messages has a worst-case complexity of n^2; however, only in rare cases will quicksort yield this complexity. Quicksort's average case complexity is nlog(n), so Query 2's complexity is more realistically 2*log(M) + 2m + m*log(m).

Algorithm complexities can be viewed in Table 3. It is difficult to compare the complexities outright, without knowing the values of l, u, and m. It is interesting that, although queries 3 and 4 have the same complexity, query 4 took almost twice as long as query 2.

The times taken for each of the queries, which can be viewed in Table 2, are consistent with what is expected from each. It should be noted that the limiting factor, in terms of time, was the sorting of the tables to prepare for each query. The sorting took *significantly* longer than the actually query.

## Project 1 and Assignment 1 timing Comparison

Table 4: Project 1 and Assignment 1 Query timing comparison

| Query | Assignment 1 Time (s) | Project 1 time (s) |
|---|---|---|
| 1 | 1.923 | 0.15 |
| 2 | 2.004 | 0.16 |
| 3 | 2.013 | 0.58 |
| 4 | 2.013 | 0.92 |

Querying in Project 1 was far more efficient than querying in Assignment 1. By devising a data model that significantly reduced the redundancies of the provided data set, we were able to drastically improve querying times. However, these query times do not take into account the time it took to pre-generate and pre-sort the tables required for Project 1 queries to be completed - steps not required to complete Assignment 1 queries. The conclusion one might draw from this observation is that if such queries had to be run only a few times, it would be less expensive to run Assignment 1 queries. However, if such queries were required to run often, it would be more efficient in the long run to pre-generate and pre-sort all tables once, and then have much more efficient querying.