sonSQL: An extensible relational DBMS for social network start-ups[⋆]

Zhifeng Bao   Jingbo Zhou   Y.C. Tay

National University of Singapore

Abstract. There is now a proliferation of social network start-ups. This demonstration introduces **sonSQL**, a MySQL variant that aims to be the default off-the-shelf database system for managing their social network data.

1 Introduction

The mushrooming of online social networks is now as unstoppable as the spread of the Web. Their datasets are heterogeneous and, sometimes, huge. Whatever its size, such a network needs a database system to properly manage its data. However, a small start-up with limited database expertise may pick a non-extensible design that is hard to modify when more features are included as the social network grows. The database research community should help such start-ups by designing and engineering a robust and scalable system that is customized for managing social network data.

Our contribution to solving the problem is **sonSQL** ("son" for "social network"). We plan to develop sonSQL into the default database system for social networks. We start with the MySQL codebase, and restrict the conceptual schema to **sonSchema**, which is tailored for social networks [1].

A visitor to our demonstration will see their social network design automatically transformed into relational tables for users, products, interactions, etc. The transformation has a question-and-answer interface that does not require database expertise from the user. The visitor can use the interface to (1) construct a relational database schema for the social entities, products and interactions; (2) populate the tables with synthetic data for test queries; and (3) modify or update the schema.

2 Objectives

We now state our technical objectives, and briefly say how we aim to fulfill them. For clarity, we refer to the sonSQL user as an **SNcreator**, who uses sonSQL to create a social network **SN**; we refer to the users of SN as **SNusers**. An SN typically belongs to some **SNdomain** (entertainment, education, technology, etc.).

The first objective is **(O1)** *sonSQL should be based on a database system that is freely available, yet reasonably complete*. Most models of social networks use graphs,

⋆ This research was supported in part by MOE Grant No. R-252-000-394-112 and carried out at the SeSaMe Centre, which is supported by the Singapore NRF under its IRC@SG Funding Initiative and administered by the IDMPO.

but we decided against using a graph database system. Our most compelling reason is this: A database system for social networks must have an expressive query language, query optimization, indices, integrity constraints, concurrency control, crash recovery, batch processing, etc. Implementing this list to bring a prototype to market is highly nontrivial for any database system, and the only ones to do so are relational DBMS.

We hence adopt a relational system for (O1), and start with MySQL as the codebase.

Our second objective is **(O2)** *sonSQL must be sufficiently general that it can cover most, if not all, current and foreseeable social networks*. Our strategy is to have a design that is *service-oriented*, i.e. the entities and relationships in the schema must correspond naturally to social network activity and services. We did an extensive survey of current social network services, and arrived at the following fundamental characterization [1]:
(i) An SN records (explicitly or implicitly) *social network relationships*, such as Facebook friends, Sina Weibo followers and LinkedIn groups. (ii) SNcreators and SNusers can introduce social *products*, such as Cyworld games, Flickr photos and Renren blogs. (iii) SNusers have *social interactions*. These are dynamic and cumulative, and each is related to a social *product*, like tagging a photograph, accepting an invitation, etc. (iv) Products can have *relationships*, like *coupon* for a *sale*, *poll* for a *meeting*, etc.

This characterization focuses on social interaction, and explicitly points out the role of products. We then designed sonSchema in Fig. 1 to match the characterization. Here, we highlight some entities and relationships (details in [1]):
• **social_product_activity** links a user to a product via an activity (vote in a poll, buy a coupon, etc.).
• **private_msg** is a message that is visible only to the sender and receiver(s).
• **response2post** is a relationship between a tag and an image, a comment and another comment, etc. It is a special case of **social_product_relationship**.

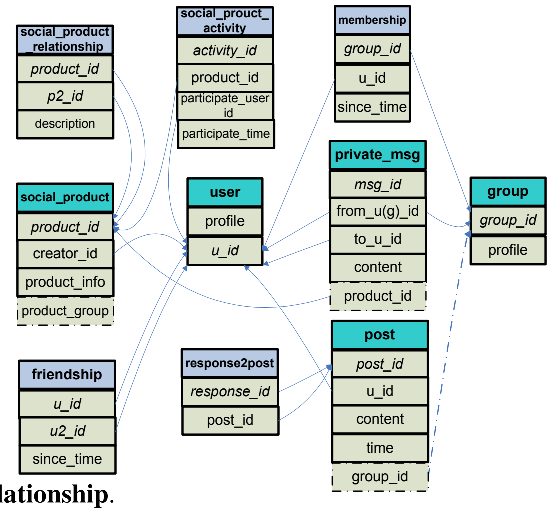


Fig. 1. The sonSchema conceptual schema. Table names are in bold, and edges point from foreign key to primary key. The 5 green tables are for entities, the 5 blue tables for relationships. Each table can have multiple logical instantiations.

sonSchema fulfils objective (O2) because it exhausts the list of entities (users and products) and relationships (user-user, user-product and product-product). It is a conceptual schema: **user** in Fig. 1 can be instantiated as a table for retailers and another for advertisers, while buying a coupon and registering for a course can be different instances of **social_product_activity**. For contrast, consider Drupal Gardens (http://drupalgardens.com), which is a software-as-a-service for designing and hosting websites. Its interface for constructing an SN has a fixed list of products (blog, forum, etc.) and services (follow, share, etc.) for SNcreator to choose from; she cannot customize these, nor create new ones, and the tables are pre-defined.

Many web services now offer social network applications. Such application data can be contained in a sonSchema instantiation that is separate from the legacy schema.

sonSchema's extensibility adds to our confidence in its generality (O2).

Our third objective is **(O3)** *a database novice should find it easy to use sonSQL to construct a schema for her social network design*. How can we provide an interface that requires little database expertise and minimal SNcreator effort, yet constructs a technically sound schema to match the SNcreator specification?

Fig. 2 shows the sonSQL architecture, with the interface at the top and MySQL at the bottom. The middle layer contains the *SN Constructor*, a *Module Tester* for the SNcreator to test the SN, and a *Data Generator* to populate the SN with test data.

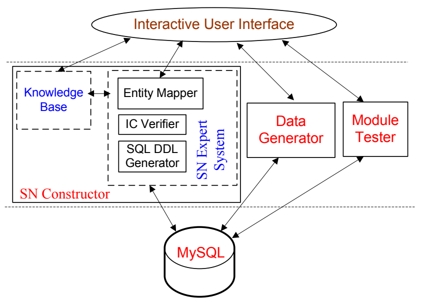


Fig. 2. sonSQL Architecture

Our solution to (O3) lies in form-based interaction. The forms are generated by a rule-based expert system, with help from a knowledge base. It has an inference engine called Entity Mapper that maps SNcreator's input into sonSchema entities and relationships. Its IC Verifier checks that the SNcreator's specifications satisfy sonSchema's integrity constraints. It then generates SQL DDL (or DML) scripts to construct (or update) MySQL tables.

Our fourth objective is **(O4)** *the schema should facilitate engineering for scalability*. sonSchema is in Boyce-Codd normal form, so its tables can be updated without requiring integrity checks that may be prohibitive in a distributed system under heavy workload. sonSchema is also hypergraph-acyclic, so it has a full reducer [2]. We are now studying the structure imposed by sonSchema on the space of all join trees, to identify bushy strategies for multi-way joins that execute faster than those produced by current optimizers [3]. We will also study the use of sonSchema's structure to design a concurrency control that provides strong consistency but without the ACID bottleneck.

3 The Demonstration

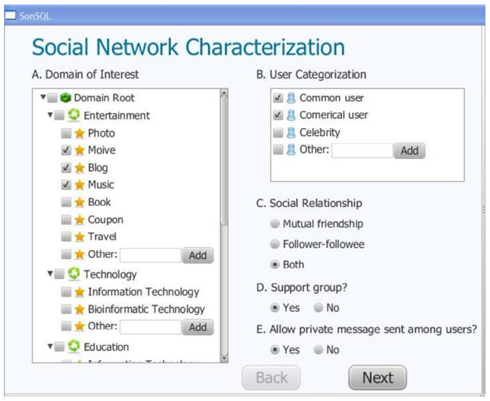
A visitor to our demonstration will be invited to assume the role of an SNcreator (see `http://sonsql.comp.nus.edu.sg/`).

Initial SN Characterization

sonSQL first presents a form like Fig. 3 for the SNcreator to browse a tree and identify the SNdomain (e.g. sports news or comic books) for her SN design. She can then specify the user categories (celebrities, authors, etc.) and social relationships (friendships, followers, etc.). With this information, sonSQL can create tables for **user**, **friendship**, **group** and **membership** in sonSchema.

Social Product Creation

Next, SNcreator sees a form for specifying products, including new ones (textbox in Fig. 3). When she selects a product (e.g. *coupon*), sonSQL checks its knowledge base and responds with options for *Producer* (advertiser? retailer?), *Consumer* (common SNuser? group?), possible related activities (*disseminate_coupon*?), and a high-level view of the SN (top-left in Fig. 4). Clicking on $\otimes$ shows a mid-level pop-up view of relationships (top-right of Fig. 4). Clicking on an entity box shows a low-level view of the tables (bottom of Fig. 4). In this way, SNcreator iterates through the products in her SN design, thus specifying the details to sonSQL.



SonSQL

Social Network Characterization

A. Domain of Interest

- Domain Root
  - Entertainment
    - ☐ Photo
    - ☑ Moive
    - ☑ Blog
    - ☑ Music
    - ☐ Book
    - ☐ Coupon
    - ☐ Travel
    - ☐ Other: Add
  - Technology
    - ☐ Information Technology
    - ☐ Bioinformatic Technology
    - ☐ Other: Add
  - Education

B. User Categorization

- ☑ Common user
- ☑ Comerical user
- ☐ Celebrity
- ☐ Other: Add

C. Social Relationship

Mutual friendship / Follower-followee / [✓ Both]

D. Support group?

[✓ Yes] No

E. Allow private message sent among users?

[✓ Yes] No

Back Next

Fig. 3. Form for initial SN characterization

Modifications and Updates

Throughout, SNcreator can click on any part of the 3-level view to undo the latest change. After the changes are committed and even after the SN is deployed, SNcreator can call up similar forms to add new activities or remove some products, etc. sonSQL translates SNcreator's forms into SQL DDL or DML transactions, verifies that integrity constraints are satisfied, then sends the transactions to the MySQL backend for execution.

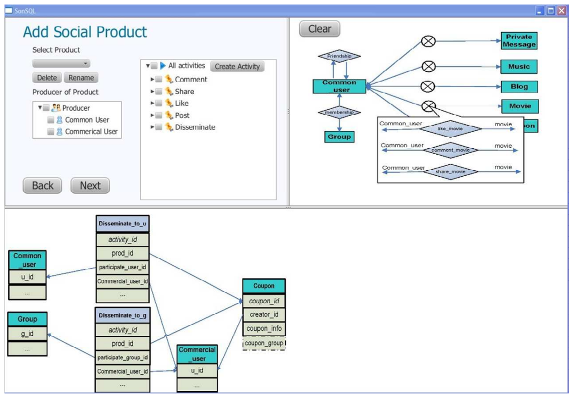


Fig. 4. 3-level views of the SN at the stage of adding social product

Testing the SN

sonSQL also has a form for the SNcreator to generate synthetic data, so she can run SQL queries to test her SN design.

References

1. Z. Bao, Y. C. Tay, and J. Zhou. sonSchema: A conceptual schema for social networks (under review). http://sonsql.comp.nus.edu.sg/rsn.pdf.
2. C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, July 1983.
3. Q. Huang. Optimizing PostgreSQL for social network databases. (FYP report), Dec. 2012.