Proposal for Database

Crystal Qin

September 07, 2012

Part A: Our Current Database

Schemas:

openfire default schema for user accounts+ an additional MySQL database to store the conference scheduling table.

For information about openfire default schema, please refer to:

http://www.igniterealtime.org/builds/openfire/docs/latest/documentation/database-guide.html

Conference-scheduling table:

List of Attributes:

Attributes:	ID	Owner	Date	Start Time	End Time	Recurring	Participants
Formats:	Int	*username@server*	Dd/mm/yyyy	Unix Time	Unix Time	String	Array(size <=9)

Note: Formats are enforced when the front-end sends conference information to the server.

Server-Client Communication:

Conference creation: network.schedulingservice.pushconference() will be called and all required information is sent as a packet named “pushConference” from the current user to the server.

Conference updating: network.schedulingservice.updateconference() will be called and a new "pushConference" message with the body starts with "update conference…."will be sent to the server.

Conference retrieval: network.schedulingservice.pullConferences() will be called and a "pullConferences" message will be sent to the server to request the collection of Conference Data representing conferences the primary user is part of. The received information will then be parsed into front-end format by calling network.schedulingservice.parseConferences() .

Part B: Current Issues and Revision Suggestions

1. *Schema Incompleteness:*
    - Lacks a column to store the name of the conference
      --Suggestion: create a column "CName";
    - Lacks a column "Notes" to store the note of the conference
      --Suggestion: create a column "Notes";
2. *Conference Update Inefficiency:*
    - Now, we need to reset the whole information of a conference again even we make a really trivial update to it. This is very inefficient.
      --Suggestion: there are two possible ways: (a). Front-end could add methods to detect which fields are actually updated and which are not and send only the updated info; so in the backend, update the received info from front-end. (b). write an update trigger in the backend to update only the ones that have been changed.
3. *Conference Retrieval Inefficiency:*
    - Now, the only method for pulling conferences retrieves all the conferences the primary user takes part in. Depending on future design in the front-end, more methods that specialize in retrieving a particular range of conferences will be needed.
      --Suggestion: Views could be created to facilitate querying into some specific combination of information and hide non-public information. If according to our

statistics, some information is retrieved far more often than others, we could even create materialized views.

In addition, building indexes and choosing proper primary keys are also ways for accelerating querying.

- Even for our current retrieval-of-all method, the retrieval process is inefficient since we need to loop through the whole table.

  --Suggestion: building index over the "Owner" column is a good choice. However, two things should be taken into consideration when creating indexes. First, Index takes up a certain bit of space. Second, index speeds up finding data but slows down insert, delete and update options.[1]

4. *Integrity Constraints are not enforced.*
   - Since our database schema is very simple and basic right now. Few integrity constraints issues are touched. But as we further develop our database, integrity constraints should be forced. For example, owner in conference scheduling table should be a foreign key for the user account table, date, start and end time couldn't be null, etc.
5. *Conference Privilege Authorization Issue:*
   - This is also a design issues. For any change or query to a database, we should check if the user has the corresponding privileges. Hence, issues we should consider are: (a). who in a conference has the privilege to view the complete information about the conference. (b). who in a conference has the privilege to update the conference information. (c). who in a conference has the privilege to insert into and delete from the conference information.
   - I just read a paper about data/privilege separation. The same idea could probably apply to our application.[2]
6. *Exception Handling*
   - Concurrency Issues: we could group related operations into transactions and change the transaction isolation level to the one most suitable for our database (Read Uncommitted, Read Committed, Repeatable Read, Serializable).
   - Crash and Merge Conflict: In either case, the transaction in execution should be rolled back and the server should notify the client of any exception.

Part C: Milestones

Since we currently don't have a database running and a lot of the revision depends on design decisions on the front-end, I am not able to set exact milestones right now but I will come back to update it when everything is set to go.

References:

1. Smith, Chris. "Introduction to Databse Indexes". Interspire, web. Feb 15, 2006. <http://www.interspire.com/content/2006/02/15/introduction-to-database-indexes/>.
2. *Diesel: Applying Privilege Separation to Database Access*. EECS Department, University of California, n.d.. Internet resource.