Issues in Informing Science and Information Technology **Volume 7, 2010**

Data Modeling for Better Performance in a Bulletin Board Application

Juan R. Bauer Mengelberg and Alejandro Hernández Negrete
Colegio de Postgraduados, Montecillo, Edo. De México, México

jbauer@colpos.mx; alehernandez@colpos.mx

Abstract

Though alternatives of the use of relational databases are usually only recommended for very large collections of data, there are situations in which they may be convenient even for smaller datasets. An electronic notice board application, which allows members of the community it serves to join several groups, or only subgroups of the members, is used to illustrate such a situation. The audience of a message also consists of certain groups, but may me limited to only some of their subgroups. Thus, at the start of a reading session by a member of the community, a query must determine the messages to be offered to the user. The situation called for a model, and the corresponding processing, that would efficiently build this list of messages to be presented to the user. Queries formulated in SQL, using a relational database model, failed to achieve satisfactory response times, even for relatively small collections of messages. So other models and processing modes were designed and tested, resulting in much better performance. A two stage query with the same model, where the second stage used the results of the first, but via programming code, though producing a thirtyfold reduction in response times, was still deemed insufficient. So a hybrid model was formulated, where bitmaps were included as fields of an otherwise relational database, resulting in an additional twentyfold diminution of the duration of the query. For example, a query took a second using the hybrid model, instead of several minutes with the initial query. Simulated data for sets with different compositions of messages and their audiences were used to evaluate the performance of the models, and the resulting response times are compared and discussed briefly, including comments on the use of relational and other database models. The materials and concepts presented could serve as part of a course in data modeling.

Keywords: data modeling, efficiency, messaging, bulletin board, hybrid models, teaching modeling, queries.

Introduction

Amongst the factors which have the greatest impact in the efficacy of computerized informing processes, are the time intervals involved in order to achieve effective communication with the information's clients. Should there be a significant delay, due either to the sending or the receiving processes, where we apply the adjective whenever the time intervals considerably exceed what the person expects or even is willing to accept, the probability of the message achieving its purpose – informing – would be considerably reduced. Additionally, user reading sessions should be considered as part of a set of possibly concurrent ones, which implies sharing computer resources with many other such processes. This in turn imposes a further constraint on the informing process: it must be efficient enough so as not to consume too many computer resources, starting with CPU usage, probably adding the use of memory, but naturally taking into account the transmission of data between resources. Thus, when defining or implementing new communication means, for example ways in which information can be posted on a computer for others to read, as in the situation described here, efficient use of computer resources should be assigned a high priority in the technical design of the processes.

With the enormously powerful computers now available, taking into account their huge RAM capacities and CPU speeds, not to speak of disk space which is very inexpensive, apparently a notion prevails in the sense that it is no longer necessary to worry about processing times. This is an example of statements found: "But does performance really matter? The answer is NO" (Berger, 2004). Actually, notwithstanding such opinions, a very important part of the research in this field is directed towards providing more efficient ways to store and process data. Of course software products such as the package we are designing for the electronic notice board have always paid attention to their performance, but additional concern by the users of the development tools is necessary.

In this paper we show one such effort. A real situation prompted the search for more efficient ways to store data, that is, to find a better data model, since the initial model did not deliver tolerable response times for the main query. First some comments on alternatives to the relational databases (RDB) are presented, since they might serve readers interested in this topic. The notice board is described in some detail in an Appendix, but its features which gave rise to the particular query are presented in the paper. The initial model formulated to store the messages, and the queries that will produce the list of messages to be offered to a user, are described next. Since there performance was unsatisfactory, a strategy of formulating alternatives, and obtaining the resulting response times via simulated data, was adopted. The subsequent models and the processing necessary to furnish the list of messages are explained, and comparisons of simulated response times for similar queries are shown. The paper concludes with some thoughts and opinions about designing data models for information systems.

Relational and Other Types of Databases

For the last 30 years, the use of relational databases (RDB) has dominated the technologies to store and use data in information systems and other applications, to the point that in most cases, not even a thought is given to an alternative. RDB's have gained this prestige and acceptance through a process of constant improvements, both in their technical aspects as in their relative ease of use. However, in the last few years, many applications have shown that there is no one-size-fits-all solution. Actually, both the use of other databases and relational database bashing has become fashionable. Many experts agree on something which might be termed obvious: were it not for the enormous popularity of RDB's, the appropriate model would depend on the particular situation.

Bain (2009) wonders if the RDB is doomed: "Even though RDBMS have provided database users with the best mix of simplicity, robustness, flexibility, performance, scalability, and compatibility, their performance in each of these areas is not necessarily better than that of an alternate solution pursuing one of these benefits in isolation". However, Bain refers mainly to scalability issues, quite different from the type of situation which presented itself in the notice board application described in this paper.

To focus on some attributes of a model where an alternative might be appropriate, Kleppmann (2009) writes: "First of all, calm down. Chances are that your current database is perfectly fine for now. But you might want to keep an eye open in case you notice some symptoms which show that you are pushing the relational model to its limits. Some symptoms relate to the structure of your data (italics are ours, indicating they apply to the situation discussed in this paper):

- Do you have tables with lots of columns, only a few of which are actually used by any particular row?
- *Do you have "attribute" tables where each row is a triple of (foreign key to row in another table, attribute name, attribute value) and you need ugly joins in your queries to deal with those tables?*
- *Have you given up on using columns for structured data, instead just serialising it (to JSON, YAML, XML or whatever) and dumping the string into your database?*
- Does your schema have a large number of many-to-many join tables or tree-like structures (a foreign key that refers to a different row in the same table)?
- Do you find yourself frequently needing to make schema changes so that you can properly represent incoming data?

Look out for these symptoms indicating that your data would better fit into a graph model:

- you find yourself writing long chains of joins (join table A to B, B to C, C to D) in your queries;
- *you are writing loops of queries in your application in order to follow a chain of relationships (particularly when you don't know in advance how long that chain is going to be);*
- you have lots of many-to-many joins or tree-like data structures;
- *your data is already in a graph form (e.g. information about who is friends with whom in a social network)".*

The term NoSQL Databases has been used to designate certain alternative databases, though there are authors do not consider some of them databases, so they prefer to call them NoSQL Systems (Kellogg, 2010). Kellogg also directed us to a very good presentation of several alternatives of RDB's by Kauhanen (2009) which we will, in the future, use in our Information Systems courses to illustrate the availability of models other than RDB's. He starts with a very simple explanation of RDB's, then introduces, one after another, Key-Value stores, Document databases, Wide column stores (Big Table and clones) and Graph databases.

We are particularly interested in hybrid models, defined by Phelan (2002) as follows: "A hybrid database is usually an object-oriented framework created to act as an interface between an 'impure' object-oriented language like C++ and a relational database manager… The hybrid design allows the object-oriented programmers to use nearly any OOP feature that they want (much like an OODBMS), while keeping the database itself relational, which allows the use of commercially available and supported products, allowing the 'best of both worlds' at the cost of the run-time overhead to support the hybrid framework".

We agree with Madison (2007) who has this to say about these models: "The hybrid model is not a one-size-fits-all solution. Many projects are best served by either using only one of the traditional models or using both models separately with a feed between them. But if the objective is to create a single database that can both store data in its properly normalized form, and run aggregation queries with good performance, the hybrid model is a design pattern to consider".

This paper illustrates a situation in which an RDB could not quite satisfy the demands imposed on it by the use of the data, but the culprit was not a huge volume of data, usually the reason to invoke other structures. In the bulletin board query described, even a small number of records resulted in very long response time, due to other factors.

As will be seen below, our hybrid model (which we shall call *ad hoc* since it is a specific made-to-order design) will include objects as fields of an RDB. This solution offers most of the benefits of such a database, but at the same time, some of the tables that would have to be part of the purely relational database are replaced by objects included as fields of other tables.

The Notice Board Software and the Main Query

The context which gave rise to the data models described in this paper is a software package called BOLNO, an electronic bulletin board, designed for a community in which its members exchange messages or post notices of interest to others. A description and justification of its usefulness is presented as an Appendix. Additionally, many details about BOLNO can be found at the product's site (jbauerm.com\bolno\homepage).

Groups of members, or the registered users of an instance of BOLNO, are defined as needed. Members will join certain groups, which are not mutually exclusive. The possibility to add subgroups to these groups arose due to several reasons, amongst others, but not the only one, that a message could be directed to a group of groups (now the subgroups of the group). Thus, a member may belong only to some of the subgroups of a certain group. Figure 1 depicts groups which could be formed by members of the Country Club which is used as an example throughout the paper.

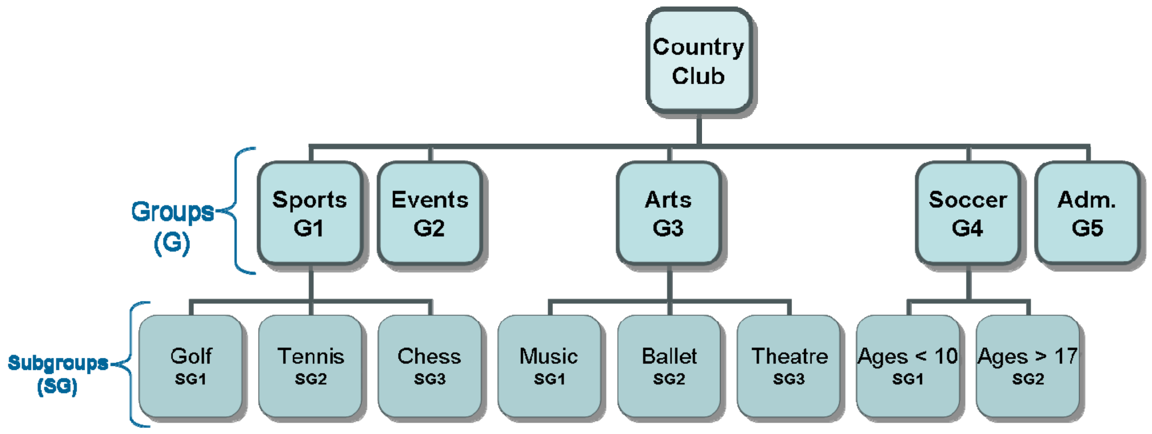


Figure 1. Groups and Subgroups of the Country Club

The salient characteristic of the messaging process, as far as its effect on the resulting queries object of this study, is that messages may be sent to any number of groups, or their subgroups. The members of these groups or subgroups constitute the *audience* of the message. Figure 2 illustrates the audience of 4 messages and the memberships of one particular user.

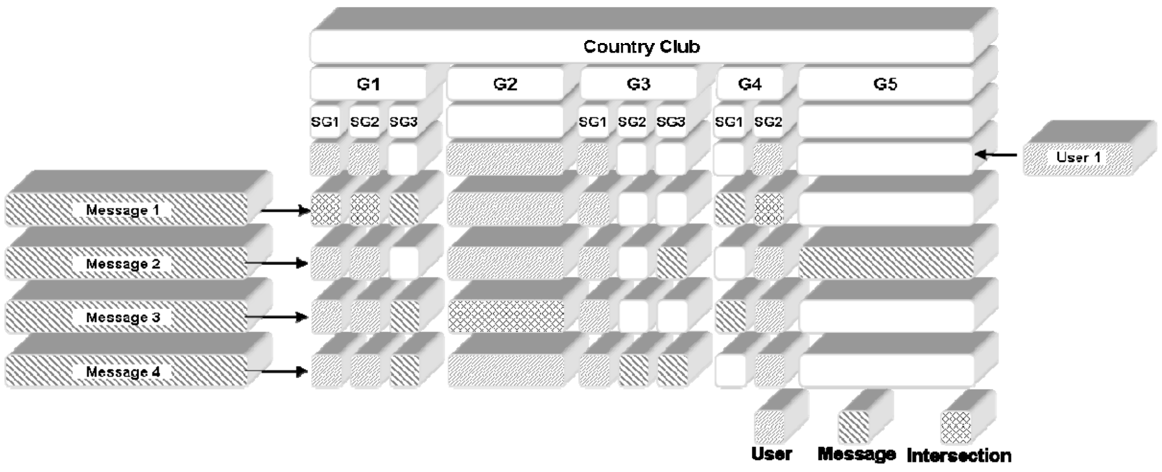


Figure 2. Audience of 4 messages and membership of a user

Note that some of the groups are not subdivided into subgroups. Messages are sent to one or more groups; for some of them, certain subgroups can be specified, so that only members of those subgroups will be included in the audience. Furthermore, the user belongs to an entire group (e.g.G2), but only to some of the other groups' subgroups (G1, G3, G4). Thus he would be offered messages 1 and 3, but he will not be able to see messages 2 and 4.

The following notation is used to describe the audience of the message as well as the user's groups (or subgroups). Groups will be represented by the letter "G" followed by the group number in brackets. Should subgroups be specified, they are added after their group, preceded by a period and indicated by the letters "SG", again with the corresponding number in brackets. Whenever there is more than one subgroup, their numbers are listed in order, separated by commas. A "0" following the period after a group specification indicates that there is no subgroup differentiation for that group. For example, the particular user of this example, as well as the audience of the messages provided to illustrate the concept in Figure 2, would be the ones reflected in Figure 3.

USER MEMBERSHIP	G(1).SG(1,2)
	G(2).0
	G(3).SG(1)
	G(4).SG(2)

MESSAGE 1:	G(1).0 G(4).0
Message 2:	G(3).SG(3) G(5).0

Message 3:	G(1).SG(3)
	G(2).0
	G(4).SG(1)

Message 4:	G(1).SG(3)
	G(3).SG(2,3)

Figure 3. Notion for a user's membership and the audience of the 4 messages

Design of the Data Model

The design of the data model, besides other considerations, focused on the need to offer an efficient process to build a list of all the messages one particular member may – or should - see. The strategy used to obtain a working data model was to divide the intervening entities into three sets: 1) the names of groups and subgroups; 2) the groups and subgroups of which every user is a member; 3) the audience of a message, that is, the groups or subgroups to which it is addressed. Some of the fields of the message table and other entities used to store the contents of the message are not mentioned, since they have no effect on the processes being considered to determine the efficiency of the models.

The first part of the model had no bearing on the rest of the model, so we used two entities. In our application, a decision made during the design of the system resulted in a maximum of 75 possible groups, with no more than 15 subgroups in each of them. The two database tables are "Groups", with the group-number as its primary index, and "Subgroups", where the primary index is formed by the fields *group-number* and *subgroup-number*. Subgroups are numbered within each group. A group or subgroup name or description is added. As we have already pointed out, there may be groups which are not subdivided into subgroups: for these, no records will be included in the second of these tables.

The membership of individuals of the groups and subgroups of the community's members constitutes the second component of the design of the data model. Briefly, the subgroups of every group to which a user belongs must be available for processing. This suggests the use of two entities: a "User-Group" entity to store the groups to which each user belongs, and another "User-Group-Subgroups" for the subgroups of these groups of which the user is a member.

Finally, to store the audience of a message, a structure such as the one shown in Figure 4 seems appropriate. It reflects our initial data model, essentially because it was straightforward, easy to understand and, until proven otherwise, seemingly easy to use. The tables were named to reflect their main indices. A group index was added to both the Message-Group and the Message-Group-Subgroup tables.

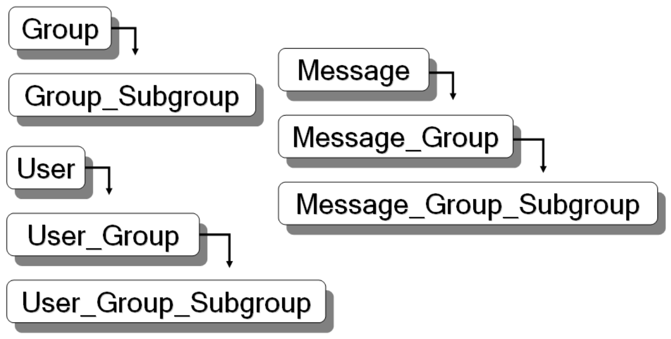


Figure 4. Hierarchy of Tables named to reflect their primary keys

"MESSAGE" is a table to store all messages with some of their attributes (title, publisher, description, show-after and expiration dates and others). Messages are numbered arbitrarily but also uniquely.

MESSAGE_GROUP is the first part of the message's audience: it contains information about the groups whose members will be allowed to read the message. Since a message may be sent to an entire group or only to some of its subgroups, an indicator of the presence (or absence) of subgroups for that group was included in the MESSAGE-GROUP table. That is, a "NO" value of this field indicates that there are no subgroups specified, which means that the message is directed to all members of the group. Of course this may or may not be due to the fact that there are no subgroups defined for that group.

MESSAGE_GROUP_SUBGROUP is the complement as far as storing the message's audience. For every group of the audience, a record will be added for every subgroup to which the message is addressed, but only if the message is not directed to the entire group.

A strategy of successive phases was adopted to obtain a model which would result in *tolerable* response times to queries of the type described above. The adjective applies to the fact that a typical user, starting a reading session of his messages, would be willing to endure the interval between asking for his messages and being presented with them. The end of every phase would be the determination that either the model was good enough for our purposes, or the next phase (the formulation of a different model) would have to be performed.

Actually, when a model proved to be unsatisfactory, instead of automatically blaming the data model, an attempt was made to make the processing more efficient – trying different algorithms or improving the code in other ways. This shows that, though correct, if may not be enough to follow advice such as: "We must address performance goals through physical database design" (Ramakrishnan & Gehrke, 2006).

In all our models and processes, the query is always performed for one particular user, so that the program will start by loading his groups and subgroups into arrays in memory. This way of starting the query, in itself, replaces a very awkward, albeit extremely inefficient, SQL code which could be used to perform all operations directly from the database tables.

The Evaluation of the Models with Generated Data

The adequacy of the combination of a model and the processing routines to produce the desired list of messages was determined by performing runs with generated data, which we refer to as simulations. Different databases were created for each model, for different numbers of messages and different numbers of groups and subgroups in their audiences. The queries were simulated for users who belonged to different numbers of groups. The duration of the process was determined by constructing a list of messages which would be shown to a particular user. In some simulations, process times were not long enough to allow the system to record the time interval accurately, so their durations were determined using the average obtained after executing the identical process several times. The interval was measured from the moment the system started the actual query, until the end of the processing which would result in displaying the data. The processing required to format and prepare the data for visualization, as well as any additional inclusion criteria, was not included in the interval, but is always insignificant compared to the other durations, besides being the same for all the models tested.

Though usually the number of groups to which a user belongs, as well as the number of groups of a message's audience, are small, the number of messages in a numerous community may be quite large, since the model will serve a package which may have different types of applications, especially regarding the number of users and messages which are interchanged amongst them. Thus, simulations were performed for 100, 1000 and 10,000 messages; somewhat larger runs, for 100,000 and 500,000 messages, were also executed, in order to know how long they might take.

Almost Normalized (SQL model)

We called the initial model the AN model, since it is *almost* normalized. Denormalization of RDB's is not considered a mistake – or sin - when there is a valid justification, such as improving performance or saving disk space. Please note that the explanation that follows is technical, provided only for readers interested in such matters. In this model, the non inclusion of sub-groups (if there are none) in the Message-Groups-Subgroups table violates the Functional dependency: Attribute B has a functional dependency on attribute A (i.e., $A \rightarrow B$) if, for each value of attribute A, there is exactly one value of attribute B. If the value of A is repeated in tuples, the value of B will also be repeated.

A query formulated in SQL, based on our AN model, would consist of a search for every subgroup of every group of the audience, that corresponds to one of the subgroups of every group of which the particular user is a member. That is, the message will be offered if there is at least one coincidence between the groups and subgroups of the message's audience and the user's groups and subgroups. As is predictable, and was confirmed by several runs, this will consume a considerabla amount of computer resources, especially should the number of subgroups in every group be high. In what follows, the term *recordset* will be used to designate a cursor, or any other data structure in memory which somehow represents data from a database. In many instances, it would probably be called *datasets*.

To provide the selection criteria, a recordset is built from the message-group table, but using the message-subgroup table as well. The resulting recordset will contain records for every group of the messages that satisfy all the criteria, i.e. the user belongs to one of its groups and, should there be subgroup constraints either in the audience of the message or concerning the user, the conditions are satisfied. For the user with the memberships shown in the example, the SQL source code would be the one shown in Figure 5. Note that, as announced, the query is not formulated using the USER-GROUPS and USER-SUBGROUPS tables. Instead, the SQL code is built by programming code so as to include the specific groups and subgroups of which the user is a member.

```
SELECT * FROM MESSAGES_GROUPS as MG WHERE
(MG.group = 1 AND MG.needs_subgroups = 0)  OR
MG.group in (SELECT MS.group FROM MESSAGES_SUBGROUPS as MS WHERE
(MS.group = 1 AND MS.subgroup = 1) OR (MS.group = 1 AND MS.subgroup = 2)) OR
MG.group = 2 OR
(MG.group = 3 AND MG.needs_subgroups = 0) OR
MG.group in (SELECT MS.group FROM MESSAGES_SUBGROUPS as MS WHERE
(MS.group = 3 AND MS.subgroup = 1)) OR
(MG.group = 4 AND MG.needs_subgroups = 0) OR
MG.group in (SELECT MS.group FROM MESSAGES_SUBGROUPS as MS WHERE
(MS.group = 4 AND MS.subgroup = 2))
```

Figure 5. The SQL source code used to process the AN model

In the simulations performed, where the data of the "club" example was not used for this purpose, even with as few as 1000 messages in the notice board, response times would be intolerable. For example, a member who belonged to 4 groups would have to wait 4 minutes, whilst another one who was in 8 groups would have to return to the terminal after about 25 minutes.

Looking for a Better Solution

As announced, before rejecting the model, its evaluation was extended: another phase was created by dividing the processing performed by the SQL statements into two parts. We will refer to this as the modified AN model. A recordset is built from the MESSAGE-GROUP table, but *without* the use of the MESSAGE-GROUP-SUBGROUP table. The resulting recordset will contain records for every group of the messages whose audience includes at least one of the user's groups. The rest of the selection is performed by special programming code.

This procedure performed much better: it produced response times of about 30 seconds for both of the cases (1000 messages shown to users who belonged to 4 and 8 groups respectively). This is an 8 fold reduction in the first case, but it is 50-fold in the second. However, should the board contain 10.000 messages, the corresponding response times would be about 2 minutes for this modified model, which the designers of the software package deemed too long. The durations obtained by several runs for both of these phases of our design process, as well as the ones for the final model to be described below, are shown later**.**

An analysis of the type of processing involved resulted in the determination of three factors, which could share the blame in some proportion. The search criteria are quite awkward, which could cause an increase in the processing required to deliver the query. In the first phase, the circumstance which takes the blame is that the query is performed using two related tables, a very convenient but expensive possibility on which the popularity of relational databases is based, but which on occasions can cause lengthy processing. In the second phase, using only the MESSAGE-GROUP table, the durations of the queries are caused by the number of records and the nature of the query – several comparisons for the same data-field.

Of course this consequence of relational modeling is well known and was briefly addressed in the section about other database models. Nevertheless, many of us keep using techniques proposed more than 30 years ago, not even taking other models into account. A search of relevant literature produced an article by McLellan (1995) about data modeling and focusing on how to make strong designs that fit our needs, though he refers only to relational models. Once again citing Ramakrishnan and Gehrke (2006): "Further, the time taken to perform **I/O** is only part of the time taken by the algorithm: we must consider CPU costs as well. Even if the time taken to do I/O accounts for most of the total time, the time taken for processing records is nontrivial and is definitely worth reducing".

Apparently, the relational model was not the solution to our search. Monash (2005) published an article called "Why Data Management Needs a New Approach", in which he states that "often, the design process in our systems must not conform to rules of a book: everything must be thought of as a new problem to be solved and the solution must agree only and just with what we need". Even before working on the design described in this paper, but especially after our study, we fully agreed with this statement.

The Ad Hoc Model

The solution we seek should enable a query of a large number of data to be performed with the least processing effort, which in turn is proportional to the number of instructions which have to be performed to achieve it, added to the time necessary for I-O activities. The first step of a solution is to enable the programmer to have control over the processing, instead of delegating it to an external handler such as a database manager. Additionally, the reduction of the number of rows of the table involved will have an almost linear effect on performance when only that table is involved and a much greater impact when two or more related tables are used in a query.

We decided to use everything we could of the RDB, but introducing certain objects – in our case, bitmaps, as fields of the tables. As stated previously, the designation of the model as being *ad hoc* reflects the fact that the model was constructed for this particular situation only, but also implies that made-to-order code must be produced to deliver the desired queries.

The concept of our design was to include the groups which constitute a message's audience as values 0 or 1 in a string of 75 (possible) groups in the Message table itself, where a "1" in position N of the string would indicate that group number N is included in the audience. Thus, we eliminated a table from the model. Furthermore, using bit strings instead of character strings obviously would help processing times: though the reduction may not show in response times, it certainly will reduce the use of computer resources, which is especially desirable in situations in which many users access the same data simultaneously.

If we represent the user's groups as a string of bits, and do likewise for the groups of the message's audience, the intersection of these strings will produce an answer to the first step - the common groups. Such intersections, implemented through AND statements, are extremely fast. We decided to use strings of 15 bits, represented as 2 byte integers. Since, as announced, a maximum of 75 groups was imposed, 5 such integers will store the 75 bits, which in turn will show if the user belongs to each of the groups. The same concept was used to indicate, for all of these 75 possible groups, whether a message is sent to a group or not. That is, instead of having a list of the actual groups involved, a positional scheme was used for the same purpose.

Table 1. The positional representation of group membership and audience

USER GROUPS (A):	000100101011001	
MESSAGE GROUPS (B):	001010010101010	
INTERSECTION (AB):	000000000001000	→ Means that this user can read the message

Thus, in the example presented in Table 1, the first 15 groups of the message and the first 15 groups of the user are represented, respectively, by the last 15 bits of two 2-byte integers. The user's first groups would be represented by 2393, whereas the value 5290 reflects the message's first 15 groups. The intersection of the bit strings is obtained executing the statement

2393 AND 5290 = 8 (which in binary notation is "1000")

The first condition for the user to be part of the message's audience is that the intersection of the bit strings is not void, that is, the resulting value is greater than zero. This means that there is at least one group present in both strings. The subgroup conditions are then executed by code: should they be needed, the subgroups of a particular group of the audience are determined with a specific query to the Message-Group table (which holds the information about the subgroups of that particular group which were included in the message's audience, once again using a bit string). This query will only be necessary if no group without subgroup constraints, regarding either the message or the user, resulted from the first step. Let us point out –probably superfluously - that this means that for every group in the intersection, subgroups for the user and subgroups for the audience are necessary as part of the determination of the user's membership of the audience. The big advantage is that, with only five AND instructions – one for each one of the 5 pairs of numbers, we are performing 75 comparisons, compared to a record by record comparison in a Relational Model.

The Ad hoc model is depicted in Figure 6. For groups which do not require subgroup distinctions, no record is added to the corresponding table neither in the user nor in the messages Group tables.

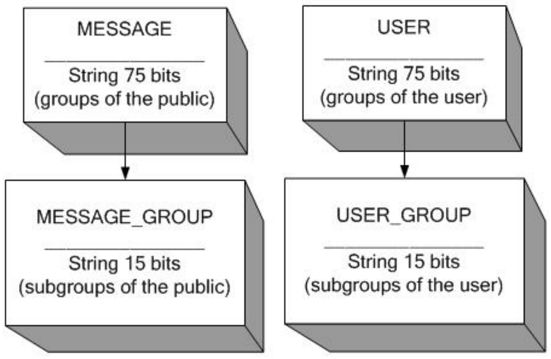
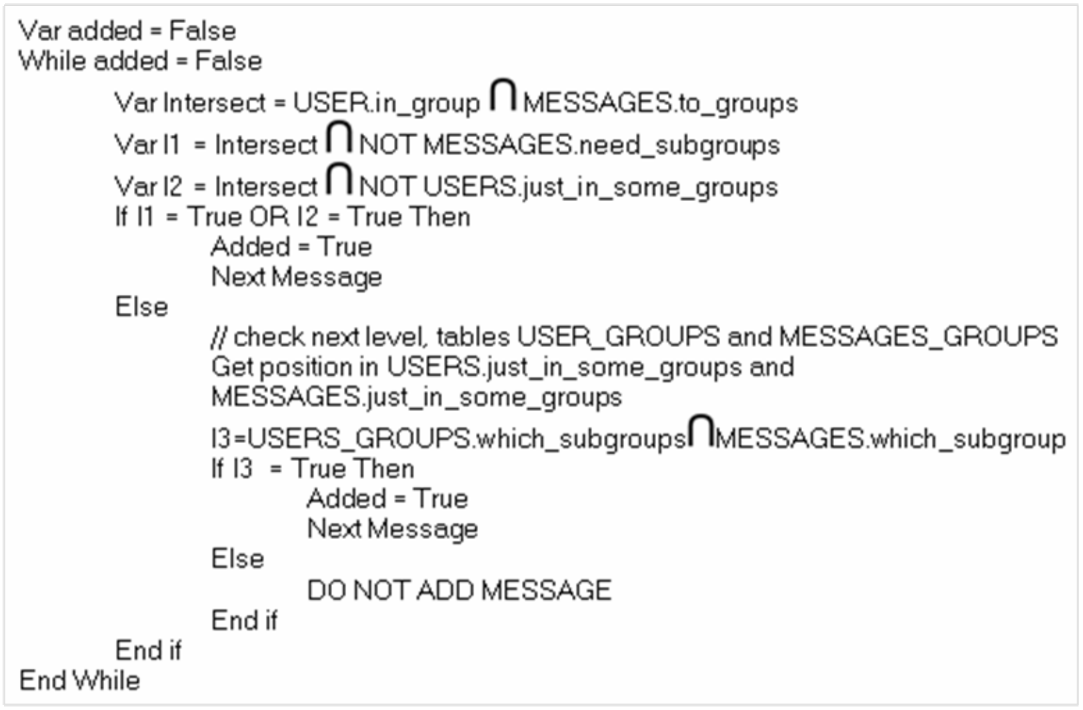


Figure 6. The Ad Hoc Model



```
Var added = False
While added = False
        Var Intersect = USER.in_group ∩ MESSAGES.to_groups
        Var I1  = Intersect ∩ NOT MESSAGES.need_subgroups
        Var I2  = Intersect ∩ NOT USERS.just_in_some_groups
        If I1  = True OR I2 = True Then
                Added = True
                Next Message
        Else
                // check next level, tables USER_GROUPS and MESSAGES_GROUPS
                Get position in USERS.just_in_some_groups and
                MESSAGES.just_in_some_groups
                I3=USERS_GROUPS.which_subgroups∩MESSAGES.which_subgroup
                If I3  = True Then
                        Added = True
                        Next Message
                Else
                        DO NOT ADD MESSAGE
                End if
        End if
End While
```

Figure 7. Algorithm used in the Ad Hoc model

Figure 7 shows the algorithm used to implement the query for this model.

The same runs that were used to evaluate the previous models (a set of 1000 messages for a user in 4 groups and another one in 8 groups) were performed using the AH model. This resulted in response times of 1.67 and 1.75 seconds respectively.

Table 2 shows how the processing was performed and evaluated in the three phases of our design process.

Table 2. How the data was obtained, stored and used in every Phase.

MODEL	Group selection	Subgroup selections
Almost Normalized	Sql	Sql
Almost Normalized Modified	Sql	Processing
Ad Hoc	Memory	Processing

Results Obtained Simulating the Query for the Three Models

The queries were performed for three different users, who belonged to 4, 8 and 29 groups respectively, shown in the first column of Table 3, which shows the durations of the query obtained from the simulations. The different trial databases contained 100, 1000 and 10,000 messages. Perhaps it should be pointed out that these samples were not random, since this would not have contributed anything to the interpretation or validity of the expected results. As can be seen, in column 3 of this table, the response times for runs with 10,000 messages were not available, since the corresponding runs were aborted after several hours. The exact resulting times did not seem important, neither for illustration purposes nor for the selection of the model.

All simulations were performed on the same computer, and programmed in Visual Basic.NET of Visual Studio 2005.

Table 3. Response Times in minutes and seconds

	Model	100 Messages	1000 Messages	10000 Messages
4 Groups	AN	00:02.43	03:59.98	-----------
	AN Modified	00:00.97	00:31.70	01:56.23
	Ad Hoc	00:00.30	00:01.67	00:06.66
8 Groups	AN	00:15.58	25:47.45	-----------
	AN Modified	00:00.98	00:32.06	02:01.06
	Ad Hoc	00:00.30	00:01.75	00:06.89
29 Groups	AN	00:46.95	01:04:23.11	-----------
	AN Modified	00:03.04	00:33.24	02:15.87
	Ad Hoc	00:00.88	00:01.92	00:7.01

Actually, we wanted to see how our ad hoc model behaved for a large number of messages: it turned out that, with this model, the run took 25 seconds for a set of 500.000 messages. As the reader can probably figure out, the other 2 processes would have taken quite a long time. Here it is important to point out that extreme data was generated to produce the worst scenario cases, initially to determine the performance of the AH model, but as it turned out, to demerit the other ones.

Once again, from a comparison of the times for the same model (AN) using the subgroup table, and without accessing this table (AN Modified), it seems the culprit is the multi-table query. Eliminating one of the tables from the initial query, as we did both in the AH as in the AN modified models, reduced the time intervals by a huge factor. This in itself is a valuable clue to enhance performance in systems that require such an improvement.

Additional Comparisons of the Two Models

The difference between the latter two models was much larger than expected. Instead of rejoicing over this fact, we analyzed the trials and discovered that they were not fair: they supposed most messages were directed to several groups, quite the contrary of what would happen in a real world situation. So the next modification of the process was performed, namely separating the messages directed to a unique group from those which had more than 1 group in their audience. A field was added to the MESSAGES table in both models: *the_only_group*, which is assigned a value of zero whenever there is no such a unique group. The rest of the fields were left intact, and used in the same manner as before in the AH model, but with a two part query in the modified AN model. Now the "where the_only_group > 0 and the_only_group in (the list of the user's groups)" is added to both queries, naturally complemented by the other part "and the_only_group = 0 AND (the rest of the SQL source)". Furthermore, a sample of messages directed to different numbers of groups was created: 80% of them were only directed to a single group. The immediate effect was to reduce the difference between the two models (only the MODIFIED AN was run, since it would definitely outperform the AN model by a huge proportion, much greater than before).

Since one could suspect, as we did, that the models would behave differently when running on a network, meaning the database resides on a different computer than the one on which the process is being executed, some additional runs were made. The literature had already pointed to the fact that the new technologies reduced the time to transfer this type of data very significantly compared to previous ones, specifically using the DotNet languages and procedures as compared to its predecessors in Visual Studio, such as Visual Basic 6.0. According to Exforsys Inc (2007) and Microsoft (2007), the use of DataSets in ADO.NET, instead of recordsets in the previous version, will greatly enhance performance, especially regarding the durations involved in building such datasets from the original data.

Some of the results of this batch of trials are shown in Table 4: They reflect the fact that the runs were performed on the computer which also contained the database (LOCAL) or not (LAN), and the use of the additional "the_only_group" field. This is indicated as the new IF, that is, with or without the separation of the query using the new field. It is important to note that the number of characters transferred for each message is very small, since only some fields will be required in the rest of the selection process to be performed by the receiving program. The runs were performed using 133,003 messages, of which 100.000 had only one group in their audience, and the simulated user was in 20,178 of them. The rest of the messages were distributed as follows: 7,000 were directed to 2 groups, each with 3 subgroups, another 7.000 to 5 groups with 5 subgroups, 8,000 to seven groups again with 5 subgroups, 8.000 to seven groups with 7 subgroups and finally, 3.003 to only 2 groups with 1 subgroup each in their audience (which is senseless, but was included to provide a variety of situations, since the number of subgroups could have a significant effect on processing times). We thought that the resulting set of messages would work "against" the ad hoc model, since it would enhance the effect of including the separation into two groups by reducing the transmission of data to the program.

Table 4. Comparative LOCAL and LAN Response Times rounded to minutes and seconds

Model	With new "if" LOCAL	With new "if" LAN	Without new "IF" LOCAL	Without new "IF" LAN
AN (mod)	1' 1"	2' 33"	6' 49"	Variable
AH	1' 1"	1' 57"	2' 31"	Variable

The omission of specific values reported in Table 4 when running on a LAN is due to the extreme variability of these numbers. To determine the effect of different traffic situations in the network used for the runs, they were performed several times with the same programs and the same data, on our local – most of the time quite busy, but very often, saturated – network. Table 5 shows the durations of exactly the same runs performed three times, since the last suspicion of some existing bias in our trial data was to be eliminated.

Table 5. Durations (rounded to minutes and seconds) for repeat runs on the same LAN

Model	First trial	Second trial	Third trial
AN(mod)	10' 2"	7' 52"	8' 22"
AH	5' 42"	9' 42 "	8' 42"

If should be pointed out that the very long durations are not typical at all, since not only the busy network was involved, but also the number of messages included in the runs was purposely exaggerated. One would simply not implement a system which had that many messages directed to some of its users: imagine a person getting a list of twenty thousand messages he should read! Previously in this paper, we reported that a run of 500.000 messages in an unfavorable situation – namely, most of the messages were meant for the user, and without the "if" which reduces processing time considerably, took 25 seconds when performed on a PC which also contained the database. As will probably be pointed out in the literature of the eventual product itself (BOLNO), should the notice board have such numbers of active messages, the processes should be run on the computer which holds the data. When accessing data on the Internet, for example, this is often the case since the programs will be executed on the same computer as the data itself.

Conclusions

The model to be used in the eventual package, with particular structures included as fields of some of the relational database tables, was designed and selected from its competitors. Since the reduction in response times after changing the model was even greater than our most optimistic forecasts, situations were studied in which this reduction would not be so spectacular. When running the queries on a busy LAN, or by using other tricks, the model chosen for the product still outperformed the other one, but not by such a big factor.

The other announced purpose of the paper was to show that the search for more efficient models and processing schemata is in no way an obsolete task, as voiced by some authors, nor is it confined to situations involving huge data collections. The reduction in processing and, of course, in its main reflection, namely the response time, and the effect perceived by the user, is not only welcome: it is essential, since long response times render the service useless. Another objective of the paper was to show that these types of activities during the modeling stages of information systems, should be taught and used whenever there is a suspicion of exaggerated processing needs which could –or should - be significantly reduced. Of course, as in all similar activities, the main indicators of the need to do these types of modeling tasks are the cardinalities involved: the more data we have, the likelier it is that we might suffer from the sheer size of our information mass.

In the model obtained and shown in this paper, we used the principle of record cardinality reduction. That is, the main reason the queries are faster is that they process fewer records. Of course the simplification of the SQL code has a huge effect as well. The main objective of the modeling task was to furnish "tolerable" response times, and the results were satisfactory in this context.

References

Bain, T. (2009). *Is the relational database doomed?* Retrieved March 8, 2010 from http://www.readwriteweb.com/enterprise/2009/02/is-the-relational-database-doomed.php

Berger, E. (2004). *Advanced compilers*. University of Massachusetts, Amherst Department of Computer Science CMPSCI 710 Spring 2004.

Exforsys Inc. (2007). *Access and manipulate data - The ADO .NET Object Model* [A VB.NET 2005 Tutorials:]. Retrieved November 28, 2007 from http://www.exforsys.com/tutorials/vb.net-2005/access-and-manipulate-data-the-ado-.net-object-model.html

Kauhanen, H. (2009). *NoSQL databases.* Accessed March 10, 2010 at http://www.slideshare.net/harrikauhanen/nosql-3376398

Kellogg, D. (2010). *IEEE Computer Society Article on NoSQL; An executive-level overview*. March 10th, 2010 . Retrieved March 11, 2010 from http://www.kellblog.com/2010/03/10/ieee-computer-society-article-on-nosql-an-executive-level-overview/

Kleppmann, M. (2009). *Should you go beyond relational databases?* Retrieved March 2, 2010 from http://carsonified.com/blog/dev/should-you-go-beyond-relational-databases/

Madison, J. (2007). *Building a hybrid data warehouse model*. Retrieved March 6, 2010 from http://www.oracle.com/technology/pub/articles/madison-models.html

McLellan T. (1995). *Data modeling: Finding the perfect fit. An introduction to data modeling*. Retrieved April 15, 2007 from http://www.islandnet.com/~tmc/html/articles/datamodl.htm

Microsoft. (2007). *DataSet Class*. Retrieved November 26 from http://msdn2.microsoft.com/en-us/library/system.data.dataset.aspx

Monash, C.A. (2005, September 19). Why data management needs a new approach. *Computerworld.com*. Retrieved April 7, 2007 from http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=104652&pageNumber=1

Phelan, P. (2002). *Hybrid databases explained*. Retrieved February 22, 2010 from http://searchoracle.techtarget.com/answer/Hybrid-databases-explained

Ramakrishnan, R. & Gehrke, J. (2006). *Databases management systems* (2nd ed.). McGrawHill Higher Education, ISBN 007246535-2 p.457

Appendix

BOLNO – An Electronic Bulletin Board

Notice: since the paper about BOLNO (Boletín electrónico de noticias) has never been published, but is still being considered for publication, the information contained in this section is provided exclusively as a complement of the paper. Its inclusion was a way to satisfy reviewer's remarks: some suggested that the underlying product should be described in more detail, whilst other reviewers pointed out that too much was included in the paper about this topic..

Why yet another Electronic Bulletin Board

Since it is not obvious why such a notice board is necessary, there being many alternatives, we present some of the motives which caused this product to be defined and developed. BOLNO was conceived several years ago, and somehow anticipated many features of modern communication tools. One of its objectives was to combine a messaging service with a notice board, two somewhat similar terms (to the point that they are often used interchangeably), but with different implications. Finally, a very exhaustive search of existing products (conducted between 1997 and 2007) failed to produce a service that included all of the features we considered essential (actually, all of those included and described below).

Some Features of the Notice and Messaging Board

The possibility of a user to select what type of notices he wishes to receive, via the groups described above, or of somebody sending a message or notification only to interested parties –its audience, is perhaps the salient feature of BOLNO. This feature was described in the paper, so it is only mentioned here. Perhaps we should add that these groups might reflect characteristics of their members (e.g. children, adults, men, women, teachers), but they could also be created according to subject matter or interest of their members (films, tournaments, lectures).

The specification of the audience of the message, besides or instead of groups, may include either a list of registered members (of the community, i.e. of the instance of BOLNO) who may receive the message regardless of their belonging to any of the groups and subgroups specified: The same is true for a list of members who may not see the message, regardless of their membership in some of those groups. Of course, not both are possible simultaneously. This allows sending a message to a group but excluding one or more of its members specifically. A notice of a surprise party is a harmless use of this feature. But in certain applications, a manager (or supervisor, or whatever) may want to receive ALL messages of a certain type. Instead of him joining every group, which will cause him to receive a lot of messages, he will ask his staff to include him "by exception" in the messages involved.

BOLNO keeps track of the readers that have seen a message. This not only allows BOLNO to offer only unread messages to every one of their potential readers (unless they ask to see them again), but also as a way to determine if a particular member of the community has seen the message. It should be noted that the "sender" of a message will never be shown his own messages, unless he specifically asks this feature to be overridden.

A message may be posted as public: it will be offered to all users who start a session (and have not yet seen the message). At the start of a session, a user may choose not to receive such messages.

All of BOLNO's functions are protected by a role-based access control system, which allows the users to perform only certain functions. Roles are defined dynamically, that is, new roles (with a new set of allowed functions) can be defined by users who have the authorization to do so.

Additionally, members of a group have a role for that group, besides the role they have as users of the software itself. The same applies to its subgroups. Thus, besides being able to receive messages (which they can do as long as they belong to the group) they may be allowed to answer messages – in the name of the group's other members, post messages, authorize other member's messages and finally, to include new members in their group.

To increase the credibility of the messages, messages sent from a group or subgroup (messages are always posted by such a member) may require an authorization by a distinguished member of the group or subgroup.

Messages posted may include multi-media materials, and their inclusion does not require any specialized knowledge on the part of the generator of the message. The administrators (or anybody else) will offer "formats" which are pre-designed containers, Posting involves providing their contents, which might be texts (usually as plain text or in RTF format), images, soundtracks, videos or any other material. Additionally, messages can have an associated file of any type (think of it as an attachment). Alternately, a message may be a PowerPoint or Flash presentation. Selection of the appropriate format is easy and fast (a list of formats is presented to the "poster of a message). He may filter this list in several ways (for example, indicate that he wishes to include 2 images, or the message should consist of two texts and a sound-track). The user of the program is then prompted to furnish the contents corresponding to every object of that format.

The use of these formats allows a certain standardization of the messages, which might contribute to their comprehension or increase their expected effect, and – why not – to some aesthetic aspects of the messages.

BOLNO allows a message to either require or allow a response (from a member of one of its audience). Usually this feature will only be used when a message is directed to a specific group or subgroup.

BOLNO allows the same message to be posted in several languages and versions. For example, the message "Come to the concert tomorrow" would, tomorrow, read "come to the concert today". Or "your lunch is in GREEN LOUNGE" at 2pm", will be read by another recipient as "Comida a las 15 horas en GREEN LOUNGE" (note that here both the language and version changed). See the "hotels" application below.

Since the notice board was conceived for a large community (such as a university, a large firm or government agency, even a country), the possibility to post a message in several languages – with little effort, since one only has to translate the texts or furnish the sound tracks or images in other languages – is attractive. This feature is particularly significant in the "hotel" application described below, which actually takes the blame for many of BOLNO's features.

BOLNO allows the preparation and display of what are called Spontaneous Sessions. They constitute a way to display a sequence of messages simultaneously on several terminals of a network, allowing different terminals to receive the messages in a particular language or version: see the hotels and airport applications.

Some Remarks about the Notice Board

It is important to note that BOLNO was not created for the Internet – though it may serve as such a tool. Users would typically have access to a computer terminal, on which they could quickly receive the messages they need or would like to read. It includes many confidentiality and credibility features.

A registered user has a "preferred language" and a "preferred version". Occasional (non-registered) users, called visitors, may indicate the language and version they prefer at the start of the session (registered users can use this to change their version, either permanently or just for that session). Incidentally, it is possible to block such changes of versions: the user will not be able to change the version. This is meant for certain uses where the message is conveyed differently to some of its recipients, but this distinction has some reason which is not to be removed by the reader.

The language selected will not only have an effect on the messages' language, but also the program functions will be offered in that language. To enable this feature, the programs are "translatable", meaning that somebody can prepare the texts, help features and other information in any language, without any knowledge of computing and, especially, without access to the source code of the programs.

BOLNO allows the posting of personal messages (directed to <u>*one*</u> user). who may not be to a registered user. In this case, he/she is a *visitor*, and might have to provide some authentication to enable BOLNO to decide which message is directed to him/her. The messages may also be posted by visitors (the occasional user, who may also see PUBLIC messages).

All types of messages may be stored in a deformed way (either scrambled, encoded or a combination of both). This allows posting a message in such a way that nobody can see - or understand - it, including the managers of the service. We have used this to post exams as messages!

Some of the features included in the design have not been implemented at this time, but are planned for the new version of the notice board. For example, a member can "read" a message from his cell-phone, and even post a text or voice message. This is not totally equivalent to sending an SMS or voice message via cell-phone: here the message is included as a message of the bulletin board, so several users can see it, with the same constraints as messages published using the package's normal software. A variation of this feature is the inclusion of procedures which enable a reader to ask, from his cell phone, for an email containing the entire message sent to his email address. This would enhance the previous one, since he could obtain the message in the chosen format used by its originator. It should be stated that these features were designed several years ago, so that some of them will no longer be necessary, given the variety of products and technologies that offer similar – or better – services.

Possible uses of BOLNO

A partial list of uses of this notice board application is presented because several of them furnished desirable features of the product.

- The message board of a department, enterprise, social club or other such community. Here the spontaneous sessions may be useful, for example, to allow messages to be displayed in idle times (nobody is using the terminal). When we conceived this feature, we thought of it as an advertising tool.

- To display important messages in shopping centers, airports: arrivals, delays, messages to certain passengers (instead of the often not very clear PA) ,which can be prepared and delivered in several languages or versions (for example, according to the location of the terminal involved). "To your left, you will see …" or "Right in front of you, you will see…"
- Notices to travelers (especially large groups) at hotels (as mentioned before, this is actually one of the applications which contributed the most to the features included in BOLNO). Let us expand the description of this use. A thousand members of a group check into a hotel. The organizers reserve lunch at a restaurant, but of course both transportation as well as lunch itself will be offered in batches. Furthermore, many of the passengers do not speak the common language. So the message informing them of their lunch will have several versions ("you board the green bus at 2", and in several languages). Of course BOLNO also has the features which enable such sessions to be prepared and distributed (sent) to many terminals. This may happen individually (that is, the passengers are asked to go to a terminal and find out the details of *their* lunch) or may be displayed on the computers installed in every room.
- Another distinguished use of BOLNO is in large meetings, such as shows or conferences. Here the inclusion of personal messages is significant, though many of the other features will serve other communication needs that arise in such events.
- For before-and-after communications referring to an event, such as a conference: announcements, remarks, notices of last minute changes, suggestions and other messages can be posted making use of the "group" feature, so that persons not interested in certain topics or notices will not be bothered by them when they use the message board.
- Confidentiality and personal messages allow some strange applications. For example, a person must make an unplanned trip, and does not have his passport and credit cards with him. So he asks somebody to take the stuff to the airport, and leave it somewhere "protected by BOLNO". They agree on a password, and the user may not see the message (which will tell him where to pick up the package) unless he provides this (temporary) password. Even visitors may use this feature, both as a sender or as a recipient of the message.
- Arrivals at an airport or train station can also use BOLNO in obvious ways. Instead of standing in a visible place waving a cardboard sign, the "collector" of the passenger will post a message on BOLNO, and the arriving passenger will go the nearest terminal and find out where he will be met. Of course this can happen in reverse order: a passenger arrives, an not finding the person who is supposed to meet him, will post a message, so that he can be found. In many airports, nowadays phones are provided for precisely that purpose.
- We have used BOLNO as a way to assign homework, and receive the completed tasks from our students, a matter which can easily be solved by a number of (now) existing products, but which at the time was a novelty. We have even posted examinations this way, which allowed us to formulate "an exam" with different questions for certain subgroups of our students (naturally, they were the members of the corresponding groups). Remember that messages can be scrambled.
- We have also used it as a "change of shift" device. A new batch of nurses, teachers, office assistants, arrives, and the previous ones will post the necessary remarks, even if they don't know who will replace them.
- Finally, BOLNO can become and important communication device in families or small groups (e.g. your office, team or classmates).. Though the current social network software

products, added to messaging and chat, have pretty much taken care of such needs, the confidentiality and possibility to subset the recipients of a message is, if not unique, at least worthy of consideration. Additional information about BOLNO, and the new product which will replace it, is available directly with the authors.

A closing remark: BOLNO has never been marketed nor widely distributed. Since the reasons are not clear, they are not stated here.

Biographies

John Bauer Mengelberg, after obtaining a degree in Mathematics from the Universidad de Buenos Aires, Argentina, got a PhD in Statistics and Operations Research from the University of Wisconsin, at Madison, where he also taught courses in the area of Stochastic Programming. He has since worked in Mexico. Besides teaching at the Colegio de Postgraduados, a school primarily involved in the field of Agronomy but which has both Statistics and Applied Computing departments, he has held several positions, always connected with the field of Information Systems, in which he has also been a consultant all his professional life. He is primarily concerned with the subject of "systems that work", a concept he has extended to signify that they work even under abnormal circumstances. He has created and implemented many computer packages, and is currently working on several software products regarding publishing papers or books in the electronic media, and what he calls sub-setting in very large data collections, specifically designing a different type of database. He has often complained he has to work by himself, and his main interest in attending conferences seems to consist in finding ways to change this.

Alejandro Hernández Negrete, after obtaining a degree in Computing Engineering at Universidad Autónoma del Estado de México, Campus Texcoco, got his Masters degree in Applied Computing at the Colegio de Postgraduados, Campus Montecillo. He has worked with Dr. Bauer for 2 years developing several systems in the fields of education and technology, including the Electronic Bulletin Board package. Currently he works for Appliance Technologies Mexico, a company that sells Netezza Performance Server for large data-warehouses and its uses in data mining and business analytics.