Implementing a UUID Primary Key in a Distributed Email Client Application

Dr Tim D. Hunt
Waikato Institute of Technology
(Wintec).
Tristram Street
Private Bag 3036
Hamilton 2020
New Zealand
tim.hunt@wintec.ac.nz

This quality assured paper appeared at the 1st annual conference of Computing and Information Technology Research and Education New Zealand (CITRENZ2010) incorporating the 23rd *Annual Conference of the National Advisory Committee on Computing Qualifications*, Dunedin, New Zealand, July 6-9. Samuel Mann and Michael Verhaart (Eds).

Abstract
This paper describes the incorporation of Universal Unique Identifiers (UUIDs) into the database design for a children's email application. Implementation of the new design is described including the mechanism for sharing data between installations using an IMAP based email server. Testing of the software has demonstrated that the design results in a functional distributed application although there remains scope for improving the synchronization algorithm. The Mifrenz application is available from http://mifrenz.com for download.

Keywords
Mifrenz, Email, Children, UUID, Database, Design, Primary Key, Synchronization.

Introduction
A critical part of database design is the choice of data for a table's primary key (PK). The traditional academic view is to look at the available natural data and select an appropriate attribute or group of attributes. The alternative view is to use an artificial key, typically implemented as an automatically incrementing value. In previous work, Hunt (2010) observed that a significant gap exists between what is normally taught to students (use the natural data) and

what actually happens in practice (use an artificial value). It was noted that although this topic has been discussed at length in the online community, there seems to be a lack of coverage in the academic literature.

Hunt (2010) described the situation where data needed to be synchronized between two or more computers via an email server. A PK was required that would allow a new record to be added to a table, before the data was fully synchronized with remote data. This ruled out the use of an automatically incrementing value, as the next value in the sequence could not be reliably determined. However, the use of natural data was possible, as each user did have a unique email address and so this was chosen as the PK. A major drawback of this design was the issue of dealing with a user wishing to change their email address, as allowing the PK to change is seen as poor database design and should be avoided if possible.

Universal Unique IDentifiers (UUIDs) (Leach, Mealling & Salz, 2005) are a particular type of artificial data that can also be used as a PK. Their use is of particular significance when data needs to be synchronized between data repositories when the connection between them is not always available e.g. mobile devices may have intermittent access to the internet. Hunt (2010) discussed the possibility of using UUIDs in a children's email application and this paper describes that process of implementation.

Mifrenz email application

Mifrenz is an email application designed specifically for children to be able to send emails in a safe environment. Unlike most other solutions for children, Mifrenz is not a server based application, but instead is installed on the user's own computer. This design means that a high availability server does not have to be provided and so costs can be lower. The design does introduce the problem of how a parent can easily update the status of a child's contacts and how a child can access his or her email from multiple locations. Mifrenz solves these problems in a similar way to Microsoft Outlook in that email and contact information is stored on a server and the application synchronizes with the server. For Mifrenz the server is a third party email server, normally Gmail (Welcome to Gmail, n.d.), and the IMAP protocol (Crispin, 2003) is used to communicate with the server. Unlike an email application used by adults, an email application for children needs to filter the emails that a child sees based on the child's contacts list. Therefore, Mifrenz needs to intercept the emails that arrive at the child's IMAP account and determine what should be done with each email. The IMAP protocol does not provide for a standard way to store contact information and so Mifrenz uses a standard email to store the contact information.

What are UUIDs?

According to Leach, Mealling & Salz (2005):
"...a Uniform Resource Name namespace for UUIDs (Universally Unique IDentifier), also known as GUIDs (Globally Unique IDentifier). A UUID is 128 bits long, and can guarantee uniqueness across space and time. UUIDs were originally used in the Apollo Network Computing System and later in the Open Software Foundation's (OSF) Distributed Computing Environment (DCE), and then in Microsoft Windows platforms."

The ability to create a unique key "across space and time" makes the UUID ideally suited for use as an artificial PK in situations where remote applications cannot communicate to determine the next value in an automatically incrementing sequence.

UUID's can be considered a mature technology as they are now a critical part of various networking technologies, e.g. the Bluetooth wireless protocol (Bluetooth Special Interest Group, p83) and the Remote Procedure Call protocol (The Open Group, 1997).

The Java programming language (Java Platform Standard Ed. 6, n.d.) has incorporated the generation of UUIDs through the provision of the UUID class and so the use of UUIDs is provided for by a robust programming environment. Using Java a UUID can easily be created using the following code, myUUID = UUID.randomUUID(), where myUUID has already been declared as variable of type UUID. From then on, myUUID can be compared to other UUIDs using the UUID compareTo method and can be converted to a text representation using the UUIDs toString method. The text representation enables straightforward storage of UUIDs in a local database or in an email stored on an email server.

Original database design using natural data

Original database design using natural data

Figure 1 shows the original Entity-Relationship Diagram (ERD) for the database. The email address was used as the PK for the User entity as this could be guaranteed to be unique for each user. This immediately constrained the application, making email address changes problematic, and also restricting each user to only one email address (although it was never intended to allow a user to have multiple email addresses). User data is not actually shared between separate installations of Mifrenz, and so using the email address as a PK for the User data was not in itself necessary. However, the Email, Contact and Joke entities can now use the email address as a foreign key relating back to the correct user. The PK of the email entity is a composite of the user email address and the time that the email was entered into the database. The software logic assures that for location no two emails are inserted at exactly the same time. It is assumed that a user is not using the software at two locations concurrently, because if they were, it is possible that two records could have the same values of email address and creation time. The Email entity also requires a foreign key (contact email address) from the Contact entity so that an email can be related to the contact that an email was either sent to or received from.

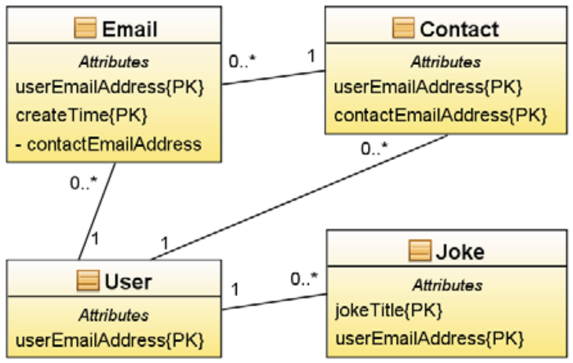


Figure 1. ERD of the original Mifrenz database design. It can be seen that each table uses an email address as either the PK, or as part of a composite PK. Only the attributes that form part of the relationships between entities are shown.

The Contact entity uses a composite PK consisting of the user email address and the contact email address. This allows contacts for all the users to be stored in the same table, yet a simple SQL query can retrieve just the contacts for the current user. Here the use of the natural data, i.e. the contact email address, again puts a restriction on the application: this time according to good design principles the email address of a contact should not change. Initially it was thought that this was not a significant restriction as the purpose of the software is to control emails based on the email address rather than a person's name. A contact is approved by a parent based on the email address and so a change in email address was initially thought of as just a new contact being created. This concept was later changed to the idea that a contact could change their email address with parental approval. As the software does not display email addresses to the user, but rather displays the contact name, there would have to be a rather complicated process of showing that emails from two different email addresses were really from the same contact.

A feature of the Mifrenz application is the creation and sharing of jokes. Each joke needs to be uniquely identified and associated with a particular user. It was decided that for a user, each of their jokes should have a unique title and therefore a composite PK of joke title and user email address was chosen. Again, it can be seen that a users email address should never change, and not so obviously, a joke cannot be received (or at least inserted into the database) with an identical title to one already present in the database.

In summary, the use of the email address was chosen as the whole or part of a PK after following the standard database design 'rules' that are taught to students. It has been shown that there are major drawbacks with this approach for this particular application. Yet, the alternative approach of using auto numbers (auto number) as the PK is not a solution when the data is distributed and offers no guarantee that the sequence of numbers can be synchronized before the need to know the next number of the sequence. This is obvious when you consider the following example.

Two siblings share their time between two households. Mifrenz is installed on PCs at both locations and there is currently a child at each location using the application. Each child creates a new email – what should the next automatically incremented number be? The database can only be synchronized between locations via the child's IMAP server (typically their Gmail account), and this only happens once they use the software at the other location. In fact in an extended family situation, they may never use the software at the other location.

A place for auto numbers

The use of UUIDs for PKs for the entities that share data between locations is discussed in the next section, but first it is worth considering the situation where data is not shared, that is the User entity. In the previous design, the user email address was used as the PK, but this has been shown to have drawbacks. The use of a UUID seemed unnecessary, as there was no need to share this data with other locations. When a child uses the application, the application knows who is logged on, and so any data that is downloaded from the users IMAP account, is obviously for that user. It was decided that it was probably straight forward to use the correct user identification number (userId) when a row of data, that has just been retrieved from the IMAP account, is inserted into the database for that user. This userId will not be stored with any data, for any of the entities, on the IMAP account as each user's data is stored in their own IMAP account and so belongs to them. This means that there is no attempt to synchronize userIds between installations of the application, which results in the possibility that a user has a different userId at each location. The userId is never shown to the user (as good database design suggests) and so the user is not going to be confused by this situation.

Referring back to Figure 1, the userId can now just replace the user email address attribute in each of the entities.

Distributed database design using UUID's

The case has been made for using UUID's for the entities that are shared and therefore it was decided to reengineer the Mifrenz application to use a UUID as the PK for the Email, Contact and Joke entities. Figure 2 shows the new ERD with UUIDs being used for the PK for the Email, Contact and Joke entities. As discussed above, the User entity now uses an auto number, the userId, for its PK. It was also decided that it would be useful to keep track of who created a particular joke, and so the Joke entity also uses the contactUUID as a foreign key. If the Joke was created by the user, the contactUUID can be left blank, hence the 0..1 multiplicity at the Contact end of the Contact-Joke relationship.

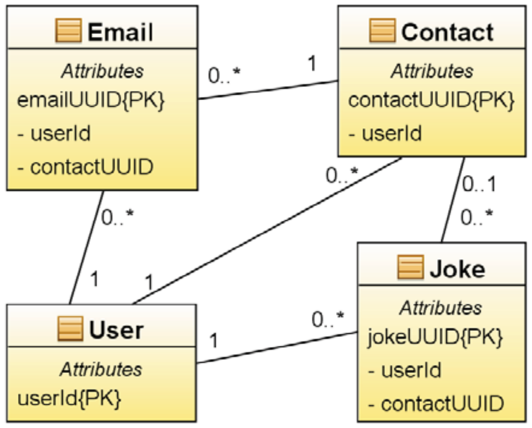
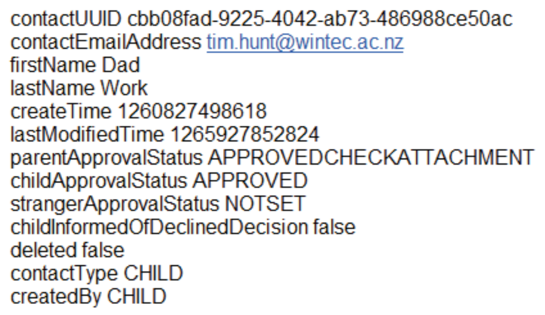


Figure 2. The reengineered database design that uses UUID's for the PKs of the Email, Contact and Joke entities. The User entity uses an automatically incrementing value, userId for the PK.

As the original design (Figure 1) used the user email address for the PK of the User entity, the user email address was unique. This uniqueness is still required (each user should have a different email address) and so a unique constraint is placed on the email address attribute in the User entity. Also, each contact should only occur once for each user, and so the Contact entity has a unique constraint consisting jointly of the contact email address and the userId attributes.

Implementation

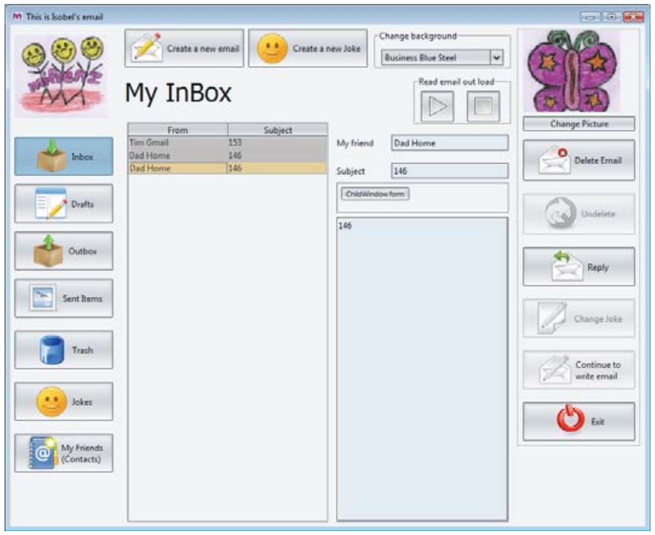
Mifrenz is written in the Java programming language and uses the HyperSQL database (Hsqldb – 100% java database, n.d.) for data storage on the local computer and connects to an IMAP email store (currently only Gmail). IMAP does not provide a standard for access to contact information or for that matter jokes. In any case, Mifrenz requires additional attributes for contacts such as the status of a contact's approval. These limitations were circumvented by using a normal email for the storage of contact and joke data. The data was stored in plain text in the body of an email, one email being used for each contact or joke. Figure 3 shows the contents of an email being used to store contact data. The main points of significance from this paper's perspective are the first line that holds the contactUUID and the deliberate absence of user identification as described above.



```
contactUUID cbb08fad-9225-4042-ab73-486988ce50ac
contactEmailAddress tim.hunt@wintec.ac.nz
firstName Dad
lastName Work
createTime 1260827498618
lastModifiedTime 1265927852824
parentApprovalStatus APPROVEDCHECKATTACHMENT
childApprovalStatus APPROVED
strangerApprovalStatus NOTSET
childInformedOfDeclinedDecision false
deleted false
contactType CHILD
createdBy CHILD
```

Figure 3. The text contents of an email stored on an IMAP account. This email is being used to hold the data of a single contact.

When a user logs on to Mifrenz, the contents of the local database are immediately displayed for the logged on user. This is significant, as a previous design did not store any data locally, but the time it took to retrieve and display the user's data was found to be unacceptable. Mifrenz then connects to the user's IMAP store and synchronizes the contacts there with the contacts stored locally. It is important that this is performed before any new emails are downloaded, as an email will only be displayed to the user if it has come from an approved contact. If an email is from someone who is not yet a contact, a challenge email is sent back to that person as the first step of an authentication procedure. To avoid a challenge email being sent multiple times, it is important that synchronization of contact information occurs first. Once contact synchronization has occurred, new email can be downloaded, processed and displayed if appropriate. Finally synchronization of the already processed emails and jokes can occur e.g. deleting emails that were deleted at another location. Figure 4 shows a screen shot of the user interface as seen by a child. Information such as when an email was sent or received is deliberately missing in order to create a simpler interface.



My InBox

Change background: Business Blue Steel

From	Subject
Tim Gmail	153
Dad Home	146
Dad Home	146

My friend: Dad Home

Subject: 146

146

Figure 4. The child user interface of the Mifrenz application. The interface deliberately displays less information than a normal email application intended for adults.

Testing and future work

The use of UUIDs in this application has proven to be a robust and successful technique for synchronizing data across multiple installations. The software has been installed in two locations and functional testing has shown that the data can be successfully replicated. To aid with ongoing testing, a log file of which functions are used and errors that occur is created and emailed to the developer each time the software is used (by the developer's children). Mifrenz is also available to download from http://mifrenz.com and can be used for a trial (with data collection) or can be purchased for actual use, in which case logging of data ceases.

Testing has highlighted the current poor performance of the synchronization process. This normally is not an issue as it occurs as a background process of which the user is unaware. However, when the user closes the application, a final synchronization occurs which the user has to wait for before shutting down the computer. A similar situation seems to happen with Microsoft Outlook which manifests (at least with the Vista operating system) when it is shut down immediately after sending an email. However, it is believed that the current algorithm used for synchronizing can be improved and this is expected to be the subject of future work.

Future work includes the plan to produce a multi-lingual version of Mifrenz utilizing the inbuilt Java capabilities for internationalization support.

Discussion

This work was undertaken in the context of teaching and indeed learning database design. It has taken the author on a journey from repeating the mantra of using natural data for PKs, to considering auto numbers, and finally to UUIDs. It is difficult to envisage this journey happening if it was not for the author working on an application that he wishes to share with a wider community: working through text book examples would unlikely have had a similar result. The author believes this experience will greatly enrich the 'real world' practitioner expectations of teaching and learning at his institution.

Summary and Conclusions

Previous work had highlighted the design issues of choosing a database table's PK for a distributed email application called Mifrenz and concluded with the suggestion of using a UUID. This work has used a combination of UUIDs for shared data, and auto numbers for non shared data. Initial testing has demonstrated the new design to be robust and it has been incorporated in the latest version of the software.

Acknowledgements

I would like to thank my two daughters who have continued to test Mifrenz and given expert feedback on how it is performing and new desirable features. Thanks also to my colleagues Bruce Ferguson and Hami Te Momo for their very useful feedback.

References

Crispin, M (2003). Internet message access protocol – version 4rev1. Network Working Group. Retrieved February 19, 2010, from http://tools.ietf.org/html/rfc3501

Hsqldb – 100% java database. (n.d.). Retrieved February 18, 2010, from http://hsqldb.org/

Hunt, T.D. (2010). Natural or artificial primary key. New Zealand Journal of Applied Computing and Information Technology, 14(1).

Leach, P., Mealling, M. & Salz, R. (2005). A universally unique identifier (UUID) URN namespace. Retrieved September 15, 2009, from http://www.ietf.org/rfc/rfc4122.txt

Java Platform Standard Ed. 6 (n.d.). Retrieved February 19, 2010, from http://java.sun.com/javase/6/docs/api/java/util/UUID .html

The Bluetooth Special Interest Group (2001). Bluetooth Profile Specifications: Part K:2 Service discovery application profile. Retrieved February 19, 2010, from http://www.bluetooth.com/NR/rdonlyres/52B76D51-B8B9-44AA-879B-2E7D90060A23/985/SDAP_SPEC_V11.pdf

The Open Group (1997). CAE specification: DCE 1.1: Remote procedure call: Document number: C706. Retrieved February 19, 2010, from http://www.opengroup.org/onlinepubs/9629399/apdx a.htm

Welcome to Gmail (n.d.). Retrieved February 19, 2010, from http://mail.google.com/