Broker Portal

David Södermark, Fredrik Söderlund, Markus Söderholm

Department of Computing Science
Umeå University, Sweden
{dw02sdd, dva96fsd, dv01mm}@cs.umu.se

Abstract. Today, managing insurances for small businesses is an expensive thing for insurance companies. This is why an Insurance Broker Portal would be useful for these companies. Placing the management of the small customers in the hands of Insurance Brokers will lead to reduced costs for managing small businesses. We made a web interface to the Portal, because it is more accessible than a stationary system. The Portal would be useful for brokers, claim handlers, insurance companies and site administrators. We have implemented a part of the broker section of the system using PHP and PostgreSQL. A large part of the application logic is put in the database to ensure the consistency in the database and to ease the front-end implementation.

1 Introduction

Broker Portal\footnote{http://www.cs.umu.se/~dw02sdd/db2/php/index.php} is a portal where a broker can, at a customers expense, buy and sign new insurance policies. Brokers enter this site via login and access all of their customers and manage each customer’s insurance policy. The most important aspect of the site is to make it all easy and accessible to the broker, although the database back-end is very complex. The database is a temporal database which contains history information of policies. It also contains a lot of integrity reassuring functionality as well as calculations of policy premiums and keeping track of policy payments. The site will assist the broker with reminders of premium payments for policies that has not been sufficiently paid for. The system also calculates commissions for four different entities (broker, insurance company, administrative\footnote{http://www.cs.umu.se/~dw02sdd/db2/php/index.php}). All commissions are added to the premium.

2 Fast Insurance Intro

- **Policy** agreement between an insurance company and a customer.
- **Base Package** A policy contains a base package, or a policy is of a base package type. A base package defines the base rules for the policy.
- **Addition** An addition is an extension of the policy, extending the coverage defined by the base package.
- **Commission** The fee a broker gets of the policy premium.
- **Rate** The part of the turnover that the insurance company needs to cover a risk. Rate could be interpreted as the probability of an event to occur (although it considers the impact as well).
- **Premium** The cost for an insurance company to cover the risk. In this system it is calculated as rate * turnover. That calculation is not a entirely correct way to do it when considering all additions.
3 EER diagram

The EER diagram in fig. 2 is a model describing the database. This is a working model of the more complex model in fig. 1 for this system. Many concepts of the actual model is lost when transferring the database to the working model. Examples of such concepts are: base packages, different types of additions, multiple commission takers, different levels of commission and damage history. This model can, however, do many things. We can represent the policy additions as an instance in the addition table even though each addition type has different properties. For example, Addition Tool has only an amount as a parameter, but the Addition Building has building type, address etc.

The working model in fig. 2 defines the tables: policy, company, broker, product, payment and addition. A broker manages companies and their policies. A policy contains one base package that is a product. A policy can contain zero to many additions and each addition has a product type. All products have a rate. All additions have a premium associated with it which is calculated from the rate and the turnover. The policies also have a premium associated with them. It is the sum of all addition premiums and the base package premium. Brokers can manage several companies and policies but a company can only have one policy. Each year the policy is renewed and is given a new revision number. A policy is renewed if it is not terminated. More formal requirements for the system are specified in section 4.

![Complex EER database model](image-url)
4 System Constraints

4.1 Broker
A broker has a broker code (one code for each broker individual, not one code per broker firm). This code is an actual real-life code that identifies each broker (and broker firm). For example, in asRW12/02, the asRW12 identifies the broker firm and the 02 identifies the broker at that firm. The broker manages policies for the customer (company) and the brokers can not access other brokers’ customers.

4.2 Company
A company (insurance holder) is a customer and it is managed by a broker. It contains an organization number in the format xxxxxxxxxx (x is a number between 0 and 9) and a company name. Each customer can only have one active policy, but several non-active policies.

4.3 Policy
A policy contains a reference to the organization number of the company that hold that policy. Each policy has a policy code and a revision number. Each year a policy is renewed and is given a new revision number. Valid start and end times are used to define when a policy is valid. Transaction times control whether the policy is active or if it is terminated. A policy always consists of one base package and zero to many additions. Premiums for each policy are calculated in triggers and stored in the policy.

4.4 BasePackage
Base package is a reference code that identifies the package that is included within the policy. The base package includes turnover amount and product code. The product code uniquely identifies the legal text associated with the package.
4.5 Commission

There are four commission holders: two administrators, one broker and one insurance company. They all get a certain percentage of the gross premium.

4.6 Addition

An addition extends the base package and contains a premium. Each addition is of a specific product type and have a rate associated with it.

4.7 Rate

Rates are used, together with a specific sub rate select field, to calculate the product premium. Rates are represented along with products and are decimal values between 0.0 and 1.0. Rates are referenced by a product code and a specific sub rate select field. The sub rate select field is used to vary the rates of a product. The rate of a product can vary depending on, for example, where you live.

4.8 Insurance letter

The insurance letter contains information about the whole policy. All additions are included and specified with net premiums. The total premium is specified with a total net premium and all the commissions are explicitly stated as numeric amounts. Finally, the total gross premium is presented.

5 Discussion

We have chosen to develop our Broker Portal with PostgreSQL as the database back-end. This gave us the possibility to put a lot of the application logic into the database. Since the web interface is meant to be very easy to use, we wanted that the brokers would have to do as little as possible while using the system. For the web interface we used the server side scripting language PHP. The PHP manual found at [?] came in handy several times during the creation of the web interface.

Having a lot of application logic in the database also made it easier to implement the web interface. In some cases we want to present information that has to be collected from several tables in the database. So instead of doing complex joins for retrieving this information we have created a set of views in the database. For instance, we have a view active_policies that gather all active policies and how much the company that has the policy should pay before the end of the current month. In the web interface, this view is used on the broker home page to list all companies that has unpaid policies. Additionally, this view is used in the first step of the wizard for creating a new policy. There it is used to get a list of companies that has no active policy.

The database is a temporal database. All policies have a valid start time and end time describing the time interval when the policy is valid. The policies also have a transaction start time set to when it was first inserted into the database. Their transaction end time is set when the policy becomes inactive. There are two ways for a policy to become inactive: when it is renewed (a new tuple with
a new revision number is inserted keeping the history of the policies); when it is cancelled.

Other database features we have used are triggers and we have also written some useful functions in PL/pgSQL. Two functions that is very useful for the policy wizard are `last_insert_chr()` and `last_insert_int()`. These functions saves us from having to do two additional SELECT statements on several pages.

6 Conclusions

In the creation of the web interface we were happy to have much of the application logic in the database. This is something we will recommend to other developers and take with us in our future endeavors. However, the database design took a lot of time. So, let us say that a company hire you to do a system in which a complex database would be preferred. After a couple of weeks the company want a preview of the system. But all you have done is designing the database back-end, so you have nothing to show your employer. That is also a scenario that should be considered.