

Projeto de Banco de Dados Evolutivos

Tradução: Reinaldo Saraiva do Carmo

1

1. Introdução

Nos últimos anos temos desenvolvido uma série de técnicas que permitem que um projeto de banco de dados evolua como uma aplicação que se desenvolve. Isto é uma habilidade muito importante para a utilização de metodologias ágeis. As técnicas utilizadas dependem de integração contínua com o banco de dados e desenvolvimento de refatorações automáticas, juntamente com um estreita colaboração entre DBAs e desenvolvedores. Ambas as técnicas trabalham na pré-produção e disponibilização de sistemas.

Nos últimos anos, assistimos a ascensão de uma nova geração de metodologias de software, as metodologias ágeis. O desenvolvimento destas metodologias surgiu a partir das necessidades de avaliação constante de novos requisitos e das complexas exigências para evolução de um projeto de banco de dados. Um dos aspectos mais centrais destas exigências é a idéia da concepção evolutiva. Em um projeto ágil assume-se que não se pode fixar as exigências de um sistema “UP-FRONT”(de cima à abaixo). Como resultado, o detalhamento desarcebado na fase de concepção, torna-se impraticável no início de um projeto, porém este tem que evoluir através das várias interações do software. Os métodos ágeis, em especial Extreme Programming (XP), possuem uma série de ações que tornam esta concepção evolutiva mais prática. Muitas pessoas têm questionado se o projeto evolutivo pode ser aplicado a um sistema com uma grande componente de dados. Inevitavelmente, muitas disseram-nos que era impossível - Este pensamento perturbador embarcou na ThoughtWorks em um grande projeto de banco de dados usando muita técnicas ágeis e de XP.

Este artigo descreve as práticas que temos utilizado para nos permitir fazer algo que era impossível. Não vamos dizer que temos um banco de dados está livre de problema de evolução, mas acho que nós temos demonstrado um conjunto de técnicas que muitas pessoas podem achar útil.

2. Lidar com Mudanças

Uma das principais características dos métodos ágeis é a sua atitude perante a mudança. Para encapsular as mudanças precisamos usar uma estratégia incremental em seu próprio trabalho de desenvolvimento e mudar pequenas partes do projeto de cada vez, em vez de tentarmos concluir tudo em uma única versão. Podemos executar uma grande alteração com uma série de pequenas mudanças incrementais. Um conceito muito importante quando estivermos executando desenvolvimento ágil é que, não precisamos acertar tudo na primeira vez, além de ser muito difícil que consigamos. Mas não precisamos capturar cada detalhe do projeto, somente precisamos que ele seja bom o suficiente para o momento.

Neste momento é importante que as pessoas envolvidas no projeto tenham uma atitude diferente, tal como realizá-lo em um ciclo contínuo, intercalando concepção, desenvolvimento e teste, até mesmo na entrega do sistema. Eis ai o contraste entre projeto

planejado e evolucionário. Uma das contribuições vitais de métodos ágeis é que eles surgem com práticas que permitem a concepção evolutiva de trabalhar de uma maneira controlada. Então, ao invés do comum caos que muitas vezes ocorre no planejamento up-front, estes métodos fornecem técnicas para controlar projetos evolutivos e torná-los práticos. Embora estas técnicas têm crescido em uso e interesse, uma das maiores dúvidas consiste em trabalhá-las com banco de dados.

A maioria das pessoas consideram que um projeto de banco de dados é algo que necessita absolutamente de planejamento up-front. Atrasar alteração do esquema de dados no desenvolvimento tende a provocar uma ampla ruptura no desenvolvimento do software. Além disso uma mudança de esquema após implantação resultar em problema migração de dados dolorosos.

Ao longo dos últimos três anos temos estado envolvidos em um grande projeto (chamado Atlas) que usou projeto de dados evolutivos e fizemos funcionar. O projeto envolveu cerca de 100 pessoas em vários locais em todo o mundo (E.U.A, Austrália e Índia). Trata-se de cerca de meio milhão de linhas de código e tem mais de 200 tabelas. A base de dados evoluíram durante um ano e meio de desenvolvimento inicial e continua a evoluir, mesmo que esteja em produção para vários clientes. Durante este projeto, iniciado com iterações de um mês, mas depois de alguns meses passou a ser duas semanas iterações que funcionou melhor. As técnicas que descrevemos aqui são aquelas que estamos usando para fazer este trabalho.

Desde que iniciamos o projeto houve uma propagação para demais projeto em paralelo, ganhando assim mais experiências de mais sistemas. Nós encontramos inspiração em outros projetos ágeis.

3. Limitação

Antes de mergulhar na técnicas, é importante afirmar que não resolvemos todos os problemas no projeto de evolução dos dados. Em particular:

- Nós desenvolvemos um banco de dados para uma única aplicação, em vez de uma integração de dados que tenta integrar múltiplas bases de dados.
- Nós não temos que manter em produção base de dados de 24/7.

4. Práticas

A nossa abordagem à projeto de banco de dados evolutivos depende de um punhado de importantes práticas.

4.1. Estreita colaboração entre DBAs e Desenvolvedores

Um dos princípios dos métodos ágeis é que as pessoas com diferentes competências e backgrounds necessitam estreita a comunicação. Eles não podem comunicar principalmente através de reuniões formais e documentos. Em vez disso, precisam falar e trabalhar uns com os outros o tempo todo. Entre aqueles que são afetados estão analistas, PMs, especialistas, desenvolvedores ... e DBAs.

Qualquer tarefa que um programador trabalha necessita potencialmente da ajuda do DBAs. Tanto o DBA e os desenvolvedores precisam considerar se um desenvolvimento tarefa vai fazer uma mudança significativa para o esquema da base de dados. Se assim

for, o desenvolvedor precisa de consultar com o DBA para decidir como fazer a mudança. O desenvolvedor sabe que é necessária uma nova funcionalidade, bem como o DBA tem uma visão global dos dados da aplicação.

Para que isto aconteça, o DBA tem que fazer-se acessível e disponível. Tornar mais fácil para programador está comunicação direta que pode ser feita durante alguns minutos apenas para colocar algumas questões. Certifique-se que os DBAs e desenvolvedores estejam próximos uns dos outros para que eles possam facilmente chegar a algum acordo juntos. Garantir que as sessões do projeto facilmente estejam acessíveis aos DBAs. Estas barreiras devem ser rompidas para processo de trabalhar com projeto de banco de dados evolutivos.

4.2. Todo mundo recebe a sua própria base de dados exemplo

Projetos evolutivos reconhece que as pessoas aprendem por tentar as coisas. Em termos de programação, os programadores com experiência, como implementar um determinado recurso, e pode fazer algumas tentativas antes de decidir por uma alternativa preferida. Projeto de banco de dados pode ser assim também. Como resultado, é importante para cada desenvolvedor ter a sua própria base de dados de teste, onde poderão experimentar, e não ter que se preocupar modificações afetem ninguém.

Muitos DBAs experientes veem múltiplas bases de dados como anátema, muito de difícil de funcionar na prática, mas temos encontrado que você pode gerenciar facilmente instâncias de banco de dados. O essencial é ter uma ferramenta para permitir que você manipule dados da mesma forma que você manipularia arquivos.

4.3. Integração Contínua com desenvolvedores

Embora os programadores possam experimentar freqüentemente em sua própria área de trabalho, é importante frequentemente aproximar as diferentes abordagens em volta de um problema. Uma aplicação precisa de um banco de dados mestre compartilhado com todos os fluxos de trabalho. Quando um programador começa uma tarefa que a cópia mestre em seu próprio trabalho, manipular e então integrar as suas alterações de volta para o mestre. Como uma regra de ouro cada desenvolvedor deve integrar, uma vez por dia.

Vejamos um exemplo, quando Mike começa uma tarefa em desenvolvimento às 10h00 (supondo que ele realmente vem primeiro que os outros). Como parte desta tarefa que ele precise mudar o esquema da base de dados. Se a mudança é fácil, como adicionar uma coluna, ele acabou de decidir como fazer a mudança sozinho, Mike também garante que a coluna que deseja adicionar não existir na base de dados, com a ajuda do dicionário de dados (discutido mais tarde). Se tarefa é mais complexa, então ele conversa com o DBA para discutir sobre as prováveis mudanças com ele.

Você pode muito bem reconhecer este princípio, como semelhante à prática da Integração Contínua, que é aplicada ao gerenciamento de código-fonte. Podemos tratar o banco de dados como uma parte do código fonte. Como tal, o banco de dados mestre é mantido sob gerenciamento de configuração da mesma forma que o código-fonte. Sempre que temos êxito na implementação do banco de dados é verificada na sistema gerenciamento de configuração, juntamente com o código, de modo que temos uma versão completa e sincronizadas história de ambos.

Com o código-fonte, grande parte da dor-de-cabeça da integração é tratado pelo sistema de controle versão. Para bancos de dados, há um esforço adicional envolvido. Quaisquer alterações no banco de dados requerem correção, como refactorings automatizado de dados, que iremos discutir em breve. Além disso, o DBA precisa olhar para qualquer mudança banco de dados e garantir que ela se encaixa dentro do esquema global do banco de dados. Para que isso funcione sem problemas, grandes mudanças não devem vir surpresas como a integração temporal - daí a necessidade do estreitamento da relação entre o DBA com os desenvolvedores.

Enfatizamos integração frequentemente porque nós percebemos que ela é muito mais fácil fazer do que integrações com grandes mudanças no esquema do banco de dados. Parece que a dor de integração aumenta exponencialmente com a dimensão da integração. Ao fazer pequenas mudanças frequentemente é muito mais fácil, na prática, ainda que muitas vezes parece um contra-senso pra muitos. Este mesmo efeito tem sido notado pelos desenvolvedores que utilizam constantemente o controle de versão como ferramenta para integração contínua do código-fonte.

4.4. Um banco de dados consiste de esquema e teste de dados

Quando falamos de um banco de dados aqui, significa não apenas o esquema do banco de dados, mas também uma quantidade justa de dados. Isto consiste de dados comum para a aplicação constante de dados, tais como a inevitável lista de todos os estados do E.U.A., bem como realização de teste nos dados.

Os dados estão aí para uma série de razões. A razão principal é permitir que sejam testados. Nós acreditamos em uma grande corpo de testes automatizados para ajudar a estabilizar o desenvolvimento de uma aplicação. Essa corpo de testes é uma abordagem comum em metodologias ágeis. Para estes testes possa trabalhar de forma eficiente, faz necessário um banco de dados com algumas amostragem dos dados de exemplo, para que todos os testes podem assumir no lugar antes que seja executado.

Bem como ajudar a testar o código, estes dados de amostra para teste permite também testar as nossas migrações de dados como por exemplo, alterar o esquema da base de dados. Com dados de amostra, somos obrigados a assegurar que quaisquer alterações de esquema também sejam tratados nos dados de amostra.

Na maior parte dos projetos vimos estes dados de amostra são fictícios. No entanto, em alguns projetos vimos pessoas utilizam dados reais para as amostras. Nestes casos, estes dados foram extraídos a partir da prévia automatizada de dados com sistemas legados por migração via scripts. Obviamente você não pode migrar todos os dados imediatamente, como no início iterações apenas uma pequena parte da base de dados é embutida realmente. Mas a ideia é desenvolver iterativamente a migração scripts, assim como o pedido e os dados são desenvolvidos iterativamente. Não só faz isto ajudar flush out problemas migração precoce, que torna muito mais fácil para o domínio peritos para trabalhar com o sistema de cultivo, uma vez que esteja familiarizado com os dados que estão a olhar e muitas vezes podem ajudar a identificar casos problemáticos que podem causar problemas para a concepção e aplicação de dados. Como resultado, estamos agora de opinião de que você deve tentar introduzir dados reais desde a primeira iteração do seu projeto.

4.5. Todas as mudanças são refatoradas no banco de dados

A técnica de refactoring é sobre tudo disciplinada e aplicando técnicas de controle para alterar um código existente. Da mesma forma que identificamos vários refactorings que forneçam dados semelhantes controle e disciplina a mudança de uma base de dados. Uma das grandes diferenças sobre refatoração de banco de dados é que elas envolvem três diferentes modificações que têm de ser feito em conjunto Alterar o esquema do banco de dados Migrando dados no banco de dados Alterar o código de acesso ao banco de dados

Assim, sempre que nós descrevemos uma refatoração de banco de dados, temos de descrever todos os três aspectos da mudança e garantir que todos os três são aplicados antes de aplicar qualquer outra refatoração. Estamos ainda no processo de documentar as várias refatorações de bando de dados, então não somos capazes de entrar em detalhes sobre eles ainda. No entanto, existem algumas coisas que podemos apontar.

Como refatoração de código, refatoração de banco de dados são muito pequenas. O conceito de encadeamento juntos uma seqüência de pequenas mudanças são basicamente as mesmas para bases de dados, pois é para o código. A tríplice natureza da mudança faz com que seja ainda mais importante manter a pequenas mudanças. Muitas refatorações de banco de dados, como adicionar uma coluna, pode ser feito sem ter que atualizar todo o código que acessa o sistema. Se o código utiliza o novo esquema sem estar ciente disso, a coluna não será utilizada. Muitas mudanças, no entanto não têm esta propriedade. Chamamos a estas mudanças destrutiva, um exemplo do que está a fazer uma coluna existente não aceita valor nulo.

Mudanças destrutivas precisam de um pouco mais de cuidado, que depende do grau de destruição envolvidos. Um exemplo de uma pequena mudança destrutiva é que ao mudar uma coluna de anulável a não nulo. Neste caso, você provavelmente só pode ir em frente ao fazê-lo. O refatoração vai cuidar de todos os dados no banco de dados que é nula. Normalmente, o programador apenas se preocupa com o que esta propriedade é um dos que pediram a mudança, e que irá atualizar o banco de dados programador é código mapeamento. Como resultado, a atualização não vai quebrar o código de ninguém e, se por acaso, algo estranho acontecer, eles descobrir logo que execute um construir e utilizar os seus testes. (Do nosso grande projeto que deu-nos algum espaço extra respirar pela espera de uma semana antes de fazer a mudança de dados.)

Dividir uma tabela muito utilizada em dois, no entanto, é um pouco mais complicado caso. Neste caso, o importante é que todos sabem que a mudança está chegando para que eles possam preparar-se para ela. Além disso vale a pena esperar por um momento mais seguro para fazer a mudança. (Estes tipos de mudanças que seria adiar para o início de uma nova iteração - nós gostamos de usar iterações de duas semanas ou menos).

O importante aqui é a escolha de um procedimento que é apropriado para o tipo de mudança que você está fazendo. Se estiver em dúvida tentar enganar-se sobre o lado mais fácil de fazer mudanças. Nossa experiência é que me queimei muito menos frequência do que muitas pessoas que pensam, e com uma configuração forte controle de todo o sistema, não é difícil de reverter deveria acontecer o pior.

4.6. Automatize as refatorações

No mundo do código estamos vendo ferramentas para alguns linguagens para automatizar muitas das refatorações identificadas. Essa automatização é essencial para banco de

dados, principalmente nas áreas de alterações de esquema e migração de dados. Como um resultado cada refatoração é automatizado de dados, escrevendo-o na forma de SQL DDL (para mudar o esquema) e DML (para a migração de dados). Estas mudanças nunca são aplicadas manualmente, em vez de serem aplicadas pelo desenvolvedor, executando um pequeno script SQL para realizar as alterações.

Uma vez feito, temos que manter esses arquivos de script para produzir uma mudança completa do registro de todas as alterações feitas na base de dados como um resultado da refatorações do banco de dados. Podemos, então, atualizar toda a instância do banco de dados para a mais recente, executando a mudança no log de todas as mudanças desde que copiou o principal para produzir a instância mais antiga do banco de dados.

Esta capacidade de sequenciar mudanças automaticamente é uma ferramenta essencial tanto para integração contínua no processo de desenvolvimento software, e como também para migração de banco de dados em produção para um novo release.

Para banco de dados em produção não fazemos mudanças durante os habituais ciclos de iteração. Uma vez que fazemos, há liberação de um release, que pode ocorrer no final de uma iteração, aplicamos com várias mudanças que são registradas na refatoração do banco de dados desde do release anterior. Esta é uma grande mudança, que somente poderá ser feita com a aplicação fora-do-ar. (Temos algumas idéias para fazer isso em um ambiente 24/7 , mas ainda não foi necessário). É também aconselhável testar esta migração de esquema antes de aplicá-lo banco de dados em produção. Até agora, temos encontrado que esta técnica tem sido bastante positiva. Ao quebrar todas as mudanças no banco de dados em uma seqüência de pequenas e simples mudanças; fomos capazes de fazer grandes mudanças para banco de dados em produção sem ficar nos colocar em apuros.

Assim como automatização, encaminhar as alterações, você pode considerar reverter automatização das alterações para cada refatoração. Se você fizer isso você poderá voltar as alterações para um banco de dados da mesma maneira automática. Nós ainda não fizemos isso, como não temos tido uma grande procura por ele, mas ele é o mesmo princípio básico. (A mesma coisa que temos feito é suportar uma antiga versão de uma aplicação com uma versão atualizada do banco de dados. Isto implicou escrever uma camada de compatibilidade que permitiu a aplicação de pensar que estava falando com a versão antiga do banco de dados, apesar de ter sido realmente a falar com um dos mais novos.)

4.7. Atualização automática de todos banco de dados dos desenvolvedores

É tudo muito bem para as pessoas a fazer mudanças e atualizar o principal, mas como é que descobrir o que mudou no principal? Em um ambiente tradicional integração contínua com o código fonte, desenvolvedores atualizam o principal antes de fazer um commit. Dessa forma eles podem resolver as questões de implementação em sua própria máquina antes do commit contendo as suas alterações pode ser compartilhada com principal. Não há nenhuma razão para que você não possa fazer isso com o banco de dados, mas encontramos uma maneira melhor.

Nós automaticamente atualizamos todo o projeto, sempre que uma alteração é feita para o banco de dados principal. O mesmo script de refatoração que atualiza o principal automaticamente todos os bancos de dados. Quando temos descrever isto, as

peessoas usualmente concede esta atualização automaticamente banco de dados dos desenvolvedores debaixo deles irá causar um problema, mas achei muito bem trabalhado. Isto somente é possível quando as pessoas trabalham ligadas à rede. Se eles trabalharem offline, como em um avião, então eles tinham que sincronizar novamente com o principal manualmente uma vez que eles têm de volta para o escritório.

4.8. Claramente que todos os banco de dados separados do código de acesso à base

Para entender as conseqüências da refatoração do banco de dados, é importante ser capaz de ver como o banco de dados é utilizado pelo aplicativo. Se o SQL está disperso por bem ou por mal ao redor do código base, isto é muito difícil de fazer. Como resultado, é importante ter uma camada de acesso de dados transparente para mostrar que o banco de dados está sendo usado e como. Para isso sugerimos sequência de uma fonte de dados a partir de padrão de projeto.

4.9. Variações

Como qualquer conjunto de práticas, estas devem ser variadas, dependendo da sua situação específica. Estas práticas são ainda muito novas, por isso não temos que entrar em toda a muitas variações, mas aqui estão alguns que temos. Mantendo a base de dados com múltiplas linhagens Um simples projeto pode sobreviver com apenas um único banco de dados de dados principal no repositório. Com projetos mais complexos, há uma necessidade de suportar multiplas variedades de projecto banco de dados, o que chamamos de linhagens de banco de dados. Podemos criar uma nova linhagem se temos um branch da aplicação para colocar em produção. No essencial a criação de uma nova linhagem de dados é muito idêntica à ramificação do código-fonte sobre o pedido, com o twist acrescentado que você também faça uma linhagem quando você precisa de um conjunto diferente de amostra de dados, como se você precisar de um monte de dados para desempenho testando.

4.10. Você não precisa DBA

Tudo isso soa como seria de muito trabalho, mas, na verdade, ela não requer uma quantidade enorme de recursos humanos. Sobre o projeto Atlas tínhamos trinta desenvolvedores e uma equipe com cerca de uma centena(incluindo, GQ, analistas e gestão). Em um determinado dia, tínhamos uma centena de várias linhagens saída de estação de trabalho de uma pessoa. Mas toda esta atividade necessita apenas de um DBA (Pramod) com desenvolvedores para dá assistência e cobertura.

Em pequenos projectos, mesmo que não seja necessário. Temos visto a utilização das técnicas em uma série de pequenos projetos (cerca de uma dúzia de pessoas) e nós encontramos nestes projetos não necessitam de um tempo inteiro DBA. Em vez disso, dependem de um jovem de desenvolvedores com um interesse no PO questões que lidar com o DBA funções a tempo parcial. A razão para isto é automação. Se você é determinado para automatizar cada tarefa, você pode lidar com um lote trabalhar com muito menos pessoas.

4.11. Ferramentas de Ajuda

Fazendo este tipo de coisa que exige uma série de tarefas repetitivas. A boa notícia é que sempre que você caia em executar tarefas repetitivas no desenvolvimento de software que

you are in an ideal position to automate them. As a result we have developed a just quantity of simple tools to help us. One of the most valuable pieces of the automation is a simple set of scripts for common tasks in a data bank.

- Bring a user the updated data
- Create a new user
- Copy the database schema, for example, consider that there is an error with your database, now Mike can copy Sue's database and try to debug the application
- Move the database, for example, from one workstation to another, that is, essentially copy and delete the database from one place to another
- Delete a user
- Export a user so a team member can work offline with the database and continue working.
- Import a user, for this, if the team members have a backup of the database, they can import the security copy and create a new schema.
- Export a reference scenario - make a backup of the main database. This is a case of specialized export of a user
- Create a report of differences between any schemas, so that Mike can discover what is structurally and what the difference is between his database and Sue's.
- Differentiate a schema from the main one, so that developers can compare their local copy against the main one.
- List of all users

Analysts and people in management often have to look at test data in the database to be able to change it easily. For this, I created an Excel application with VBA scripts to pull the data from the database into an Excel file, allowing people to edit the file and send the data back to the database. Although there are other tools to visualize and edit the content of a database, Excel works well, because many people are familiar with it.

Everyone on the project needs to be able to explore the database easily, in a way that they can know which tables are available and how they are used. We implemented a tool based on HTML to do this, we use servlets to consult the metadata of the database. So Mike, before adding a column to a table, could look and see if the column already exists, by searching the metadata of the tables and columns. We did the data modeling using Erwin and pulled Erwin's data into our own metadata tables.

4.12. New measures and other information

There is no last word on the topic of database evolution. We certainly want to see and how we can expand these techniques for the database integration that works 24/7, and other problems that we have not yet solved.

References

- [1] Martin Fowler **Evolutionary Database Design.** Site Martin Fowler.
<http://martinfowler.com/articles/evodb.html>. 2003