DOCUMENTAÇÃO TÉCNICA | ABRIL DE 2016

Teste de ETL totalmente automatizado: um guia passo a passo

Seção 1

A função essencial de ETL na organização moderna

Desde seu rápido surgimento no mundo de data warehouse e business intelligence, o processo de ETL (Extract, Transform and Load – Extração, Transformação e Carregamento) tornou-se um processo onipresente no universo do software. Como o próprio nome sugere, uma rotina de ETL consiste em três etapas diferentes, as quais sempre ocorrem paralelamente: os dados são extraídos de uma ou mais fontes de dados, são convertidos para o estado necessário e carregados no destino desejado, geralmente, um data warehouse, um data mart ou banco de dados. Uma rotina de ETL desenvolvida geralmente também incluirá tratamento de erros, infraestrutura de log e o ambiente da rotina.[1]

Até agora, a ETL tem sido usada principalmente na preparação de dados diferentes e em grande volume para análise e business intelligence. Entretanto, seu uso está sendo ampliado para além da movimentação de dados, com migração de dados para novos sistemas, uma aplicação cada vez mais comum, além do tratamento de junções, classificações e integrações de dados.[2]

Desse modo, a ETL é um recurso essencial do ciclo de vida de desenvolvimento moderno e em constante mudança, em que várias releases e versões são desenvolvidas paralelamente em um determinado momento. As organizações devem ter a capacidade de aprimorar, integrar e inovar constantemente seu software, com dados disponíveis para testadores e desenvolvedores no estado ideal para cada iteração e release. Os dados serão extraídos das mesmas fontes, mas devem ser transformados para que correspondam aos requisitos específicos de cada uma das equipes. Isso será particularmente verdadeiro se uma organização estiver trabalhando arduamente para ser "ágil" ou para implementar a entrega contínua de modo bem-sucedido.

Um bom exemplo da função essencial da rotina de ETL na entrega contínua foi encontrado em um grande banco multinacional, com o qual a CA Technologies trabalhou. O banco estava passando por um processo de aquisição e tinha de migrar clientes, produtos e finanças dos clientes dos bancos adquiridos para sua infraestrutura existente. Isso significava que 80 arquivos de inserção tinham de ser recuperados, convertidos e validados antes de serem carregados nos sistemas de back-end dos bancos. Além disso, esses dados deveriam estar disponíveis em 47 projetos paralelamente, com a manutenção da integridade referencial dos dados. Nesse caso, a rotina de ETL foi fundamental para possibilitar o paralelismo necessário para a entrega contínua bem-sucedida.

Entretanto, apesar do aumento no uso e na importância da ETL, os testes de ETL refletem o estado do teste em geral, que é muito lento e manual e ainda possibilita um grande volume inaceitável de falhas até a produção. Este artigo apresentará os desafios enfrentados em uma abordagem típica quanto ao teste de ETL, explorando todos eles. Uma abordagem alternativa com base ampla em modelos será apresentada, considerando como é possível tornar o teste de ETL muito mais eficiente, eficaz e sistemático.

Seção 2

A abordagem típica para o teste de ETL e os desafios comuns enfrentados

Geralmente, ao validar as regras de transformação de ETL, os testadores criam um conjunto de código de sombra, usam esse conjunto para transformar os dados e depois comparam os resultados reais com os resultados esperados. Geralmente, o SQL ou os scripts de ETL são copiados manualmente nos dados de origem e depois executados, com a gravação dos resultados. Depois, o mesmo script é copiado nos dados de destino, com a gravação dos resultados. Os dois conjuntos de resultados (reais e esperados) são comparados para validar se os dados foram transformados corretamente.

O problema-raiz: complexidade e capacidade de teste

O problema subjacente nessa validação manual é que as rotinas de ETL, devido a sua natureza, rapidamente se tornam altamente complexas. À medida que os negócios progridem e que a variedade e o volume de dados coletados aumentam, as regras de ETL aumentam para que seja possível lidar com tudo isso. Na tão conhecida "era de informações", esse aumento está ocorrendo mais rápido do que os métodos de teste tradicionais conseguem lidar. Na verdade, o volume absoluto de informações coletadas pelas organizações orientadas a dados tem aumentado tão rapidamente que 90% dos dados no mundo todo foram coletados só nos últimos dois anos[3], enquanto o volume de dados coletados por uma organização típica está dobrando anualmente.[4]

A complexidade dos sistemas projetados para coletar, transferir, operar e apresentar esses dados aumenta exponencialmente com a inclusão de cada decisão. Isso inclui as regras de ETL, sendo que há inúmeros fatores que podem afetar a complexidade das transformações:

- O número e a variedade de fontes de dados envolvidas, incluindo tipos de bancos de dados relacionais e não relacionais e arquivos simples;
- O número e a variedade de destinos de dados;
- O número e a variedade de transformações simples e complexas, de pesquisas simples a uniões ativas e normalizações;
- O número de junções, trechos de código e transformações reutilizáveis;
- O número de tabelas criadas.[5]

Todos esses fatores são agravados pelo foco atual em soluções quase que em tempo real e na complicação agregada que elas trazem.[6]

A documentação não ajuda

Essa crescente complexidade impacta diretamente a capacidade de teste das rotinas de ETL. Ela é muito problemática para o teste de ETL, pois as regras de transformação geralmente são armazenadas em documentações insatisfatórias, gerando a falta de resultados esperados explícitos. Geralmente, as regras são projetadas durante a fase de desenvolvimento e armazenadas em documentos por escrito ou em planilhas – ou, pior ainda, talvez elas não existam fora das mentes dos desenvolvedores e testadores.[7] Nesse caso, não há uma documentação real a partir da qual os casos de teste (ou seja, o código de sombra) possam ser derivados com confiança.

Em uma equipe de business intelligence com a qual trabalhamos, os requisitos eram armazenados como documentos por escrito, sendo que os casos de teste eram armazenados em planilhas. Essa documentação estática apresentava uma "parede de palavras", da qual as etapas lógicas das rotinas de ETL tinham de ser decifradas. Os documentos eram concentrados no "caminho bem-sucedido" e não continham condições negativas, de modo que, aproximadamente, 80% da lógica possível que precisava ser testada em um sistema típico era omitida. Com essa documentação incompleta e ambígua, os testadores não tinham meios de compreender as rotinas de ETL com facilidade ou precisão.

Com muita frequência, os testadores tinham de preencher as lacunas, mas, quando faziam isso de modo incorreto, as falhas entravam nas rotinas de ETL. Dados inválidos eram então copiados para o destino, embora o código e os casos de teste refletissem uma interpretação plausível da documentação dos requisitos.

"Lixo entra, lixo sai" – derivando manualmente os casos de teste e os resultados esperados

Na verdade, um volume de 56% de falhas que chegam à produção deve-se à ambiguidade na documentação de requisitos.[8] Em partes, isso ocorre porque os casos de teste e os resultados esperados são manuais derivados da documentação insatisfatória: um processo muito demorado que geralmente leva à cobertura insuficiente do teste.

Qualidade

A derivação manual é ad hoc e não sistemática e, geralmente, leva à criação dos casos de teste que vierem à mente dos testadores. Diante da complexidade discutida das rotinas de ETL, combinada com a documentação insatisfatória oferecida, é injusto esperar até mesmo que o testador mais talentoso crie todos os testes necessários para validar as possíveis combinações de dados. Por exemplo, se um sistema simples com 32 nós e 62 extremidades for projetado de modo linear, poderá haver 1.073.741.824 rotas possíveis por meio de sua lógica – um número maior do que qualquer pessoa possa planejar.

Desse modo, a derivação ad hoc leva ao excesso e à insuficiência de testes, em que apenas uma fração da lógica possível envolvida em uma rotina de ETL é testada. O teste negativo será um desafio em particular, e os casos de teste, tais como a documentação, geralmente se concentram quase que exclusivamente em caminhos bem-sucedidos. Entretanto, são essas exceções e esses resultados inesperados que devem ser testados, pois é essencial que as rotinas de ETL rejeitem esses tipos de dados inválidos.

Por exemplo, uma empresa de serviços financeiros com a qual a CA Technologies trabalhou dependia de 11 casos de teste com cobertura de apenas 16%. Esse número é um padrão normal; nossas auditorias constataram que uma cobertura de teste funcional de 10 a 20% seja a norma. Outro projeto na empresa apresentou excesso nos testes em um fator de 18 vezes, pois os casos de teste foram acumulados na tentativa de testar o sistema completamente. No entanto, não foi alcançada a cobertura máxima. A empresa teve um custo de US$ 26.000 para executar os 150 casos de teste adicionais com o auxílio de um provedor terceirizado.[9]

O resultado de uma cobertura insatisfatória como essa é a entrada de falhas no código, algo caro e demorado para corrigir: estudos comprovaram que pode custar de 40 a 1.000 vezes mais recursos[10] e 50 vezes mais tempo[11] para corrigir erros durante os testes do que detectá-los logo no início. Pior ainda, os erros podem passar despercebidos, de modo que dados inválidos sejam copiados no destino dinâmico, podendo ameaçar a integridade do sistema. Além disso, diante de uma documentação estática, os testadores não contam com um modo confiável para avaliar a cobertura de seus casos de teste: eles não conseguem afirmar com certeza se uma rotina específica está sendo testada em excesso ou de modo insuficiente e não conseguem dar prioridade aos testes com base na importância.

Tempo e esforço: os testes não acompanham o ritmo

Criar casos de teste a partir de uma documentação é também uma tarefa muito demorada e que exige muito trabalho. No exemplo anterior, foram necessárias 6 horas para criar os 11 casos de teste, sendo que o grande número de testes em excesso na empresa demorou ainda mais tempo. Esse tempo gasto no design manual de casos de teste aumenta mais ainda se considerarmos o tempo que se leva para comparar os resultados reais e os esperados.

Comparar amplos campos individuais com os resultados esperados é um processo muito demorado devido ao volume de dados produzidos por uma rotina de ETL complexa e ao fato de que os dados de origem sempre serão armazenados em uma grande variedade de tipos de arquivo e bancos de dados. Isso também é muito difícil, pois os dados transformados devem ser validados em vários níveis:

- Os testadores devem verificar a completude dos dados, garantindo que o total de origens e destinos de dados corresponda;
- A integridade de dados deve ser garantida, verificando se os dados de destino são consistentes com os dados de origem;
- A transformação deve corresponder às regras de negócios;
- A consistência dos dados deve ser garantida, identificando quaisquer elementos duplicados inesperados;
- A integridade referencial deve ser mantida, com a detecção de quaisquer registros órfãos ou chaves estrangeiras ausentes.[12]

Muitas vezes, são feitas concessões e somente uma amostra do conjunto de dados é validada. Entretanto, isso também afeta a eficácia do teste de ETL, prejudicando a confiabilidade das transformações. Diante da função de muitas rotinas de ETL nas operações essenciais aos negócios, essa concessão é inaceitável. A qualidade é ainda mais afetada pela natureza sujeita a erros das comparações manuais, principalmente quando os resultados esperados não são muito bem definidos ou, pior ainda, quando não são definidos independentemente do código de sombra usado no teste. Nesse caso, os testadores tendem a supor que um teste tenha sido aprovado, a menos que o resultado real seja muito estranho: sem a predefinição de resultados esperados, é provável que eles assumam que o resultado real seja o resultado esperado[13], o que impossibilita determinar, com certeza, a validade dos dados.

O problema dos dados

Até agora, nosso foco esteve nos problemas encontrados na derivação dos testes (código de sombra) necessários para validar regras de ETL. No entanto, assim que os casos de teste forem derivados, os testadores precisarão de dados de origem fictícios para lançar no sistema. Essa é outra causa frequente de gargalos e falhas.

Você tem todos os dados necessários para testar as rotinas de ETL complexas?

Ter dados "inválidos" o suficiente é essencial para um teste de ETL eficaz, pois é fundamental que, quando em operação, a regra de ETL rejeite esse tipo de dados e envie-o para o usuário apropriado, no formato apropriado. Se esses dados não forem rejeitados, eles provavelmente gerarão falhas ou até mesmo um colapso no sistema.

Nesse contexto, pode-se definir "dados inválidos" de inúmeras maneiras, as quais correspondem às maneiras em que os dados devem ser validados pelos testadores. Podem ser dados que, com base nas regras de negócios, nunca deverão ser aceitos. Por exemplo, valores negativos em um carrinho de compras online, quando não houver nenhum voucher. Podem ser dados que ameacem a integridade referencial de um warehouse, como ausência de dados obrigatórios ou interdependentes ou ausência de dados entre os próprios dados de entrada.[14] Portanto, os dados de teste lançados por meio de uma regra de validação de ETL devem conter o conjunto completo de dados inválidos para possibilitar 100% de cobertura no teste funcional.

Raramente, esses dados são encontrados nas fontes de dados de produção que ainda são fornecidas às equipes de teste de muitas organizações. Isso ocorre porque os dados de produção são extraídos de cenários do tipo "negócios conduzidos como de costume" que já ocorreram no passado e que, portanto, são mais aceitáveis por natureza para a exclusão de dados inválidos. Eles não contêm os resultados inesperados, as exceções ou as condições de limite necessários para o teste de ETL; em vez disso, eles terão como foco o "caminho bem-sucedido". Na verdade, nossas auditorias de dados de produção constataram que uma cobertura de 10 a 20% seja a norma. A ironia disso é que, quanto melhor a criação da rotina, menor será a probabilidade de entrada de "dados inválidos". Isso significa que há menos dados de variedade suficiente para testar completamente as regras de ETL no futuro.

Os dados estão disponíveis quando você precisa deles?

Outra questão importante sobre a validação de ETL é a disponibilidade dos dados. Os dados de origem podem ser extraídos de 50 fontes diferentes em uma empresa. O problema no teste de ETL (e no teste em geral) é que ele é visto como uma série de etapas lineares. Por esse motivo, as equipes de teste são obrigadas a esperar pelos dados enquanto estes estão sendo usados por outra equipe.

Considere o exemplo de uma cadeia de migração bancária, na qual os dados são movidos de um banco e convertidos para os sistemas de outro com o uso de uma ferramenta de reconciliação. Em cada etapa, é necessário validar os dados para verificar se eles foram convertidos corretamente para a estrutura de controle financeiro, se o número da conta foi recuperado, se havia precisão de hora em hora e assim por diante. Esse processo pode englobar várias etapas diferentes, da entrada básica à deduplicação e preparação e da propagação à reserva de dados. Além disso, várias equipes poderão estar envolvidas, incluindo equipes de ETL e equipes que não trabalham com a ETL, principalmente aquelas que lidam com mainframes.

Se os dados de todos os dados da empresa não estiverem disponíveis paralelamente a todas as equipes, os atrasos se acumularão porque as equipes ficarão ociosas aguardando pela disponibilidade. Os testadores escreverão seu código fantasma e não terão os dados de origem para validar uma regra de ETL, pois eles estão sendo usados por outra equipe. Na verdade, constatamos que um testador típico consegue gastar 50% de seu tempo pesquisando, manipulando e criando dados ou esperando por eles. Isso pode representar 20% do SDLC total.

O que ocorre quando as regras mudam?

Derivar manualmente casos de teste e dados de requisitos estáticos é algo que não reage muito às alterações. As rotinas de ETL mudam com a mesma rapidez com que uma empresa progride, e o volume e a variedade de dados coletados aumentam com tudo isso. Quando ocorrem essas mudanças constantes, entretanto, o teste de ETL não consegue acompanhar esse ritmo.

Possivelmente, a única grande causa de atrasos nos projetos nesse caso é a necessidade de verificar e atualizar os casos de teste existentes quando as rotinas mudarem. Os testadores não têm meios para identificar automaticamente o impacto de uma alteração feita nos requisitos estáticos e nos casos de teste. Em vez disso, eles precisam verificar manualmente todos os casos de teste existentes, sem nenhuma forma de avaliar se a cobertura realmente foi mantida.

Com a equipe de business intelligence mencionada anteriormente, na qual os requisitos e os casos de teste eram armazenados em documentos por escrito e em planilhas, respectivamente, as alterações eram muito problemáticas. Um testador levava 7,5 horas para verificar e atualizar um conjunto de casos de teste quando ocorria uma alteração em uma única regra de ETL. Em outra organização com que trabalhamos, dois testadores levaram dois dias para verificar cada um dos casos de teste existentes quando foi realizada uma alteração nos requisitos.

Seção 3

A alternativa viável: teste de ETL totalmente automatizado

Está claro que, quando os casos de teste forem derivados e os resultados comparados manualmente, o teste de ETL não conseguirá acompanhar o ritmo de mudanças constantes dos requisitos corporativos. A seguir, apresentamos uma estratégia possível para aumentar a eficiência e a eficácia do teste de ETL. Trata-se de uma estratégia com base em modelos e em requisitos, projetada para priorizar o esforço dos testes e agregar qualidade ao ciclo de vida de ETL logo no início. Essa abordagem com base em modelos insere automação em todas as etapas de teste e desenvolvimento, além de fazer com que o teste de ETL seja totalmente reativo às constantes mudanças.

1) Comece com um modelo formal

Inserir um modelo formal no teste de ETL proporciona a vantagem fundamental de que ele prioriza o esforço de teste, sendo que todos os ativos de desenvolvimento e teste subsequentes podem ser derivados do esforço inicial de mapear uma regra de ETL para um modelo. Desse modo, o modelo formal se torna o pilar da validação de ETL totalmente automatizada.

Entretanto, o modelo formal também ajuda a solucionar o problema mais específico mencionado anteriormente quanto à ambiguidade e incompletude nos requisitos. Ele ajuda a manter a capacidade de teste apesar da crescente complexidade das regras de ETL, de modo que os testadores possam compreender exatamente a lógica que precisa ser testada, tudo de modo rápido e visual. Eles podem facilmente saber quais dados válidos e inválidos deverão ser inseridos para testar completamente uma regra de transformação e qual deverá ser o resultado esperado em cada caso.

Por exemplo, um modelo de fluxograma detalha a complicada documentação em forma de "parede de palavras" usando blocos mais compreensíveis. Ele reduz a ETL a sua lógica de causa e efeito, mapeando-a para uma série de instruções do tipo "se..., então...", vinculadas a uma hierarquia de processos.[15] Cada uma dessas etapas em vigor se tornará um componente de teste, informando ao testador exatamente o que deve ser validado. Desse modo, criar as rotinas de ETL como um fluxograma elimina a ambiguidade da documentação de requisitos, trabalhando para evitar os 56% de falhas que têm origem dela.

À medida que as rotinas de ETL tornam-se mais complexas, o fluxograma atua como um ponto único de referência. Em contraste aos diagramas e documentos por escrito "estáticos", é possível adicionar lógicas facilmente ao modelo. Além disso, existe a possibilidade de abstração de rotinas altamente complicadas por meio do uso da tecnologia de subfluxo, o que aumenta a capacidade de teste. Componentes de nível inferior podem ser incorporados a fluxos principais, de modo que as inúmeras rotinas que compõem um conjunto altamente complexo de regras de ETL possam ser consolidadas em um único diagrama visual.

Além de reduzir a ambiguidade, o modelo de fluxograma ajuda a evitar a incompletude. Ele faz com que o criador de modelos pense em termos de restrições, condições negativas, limitações e condições de limite, com a seguinte pergunta: "o que acontecerá se essa causa ou esse gatilho não estiver presente?" Desse modo, é necessário sistematicamente elaborar caminhos negativos, acomodando o teste negativo que deverá compor grande parte da validação de ETL. Algoritmos de verificação de completude também podem ser aplicados, pois o modelo formal é um diagrama da regra de ETL com precisão matemática.

Isso elimina lógicas ausentes, como os "elses pendentes", de modo que os casos de teste que englobam 100% das combinações de dados possíveis possam ser derivados. (No entanto, deve-se observar que quase sempre haverá mais combinações que poderão ser viavelmente executadas como testes. Por esse motivo, as técnicas de otimização serão discutidas posteriormente.) Outra vantagem importante é que é possível definir os resultados esperados no modelo, independentemente dos casos de teste. Em outras palavras, com um fluxograma, o usuário pode definir o modelo que incluirá as entradas de limite e propagará os resultados esperados para diferentes pontos finais no modelo. Isso define claramente o que uma regra de validação deverá aceitar e rejeitar, de modo que os testadores não assumam incorretamente que os testes foram aprovados quando o resultado esperado não é explícito.

Deve-se observar que a adoção do teste com base em modelos para a validação de ETL não exige a adoção completa de uma abordagem direcionada a requisitos quanto ao teste e ao desenvolvimento em toda a empresa. Não é necessária uma mudança significativa e, de acordo com nossa experiência, leva-se menos de 90 minutos para criar uma rotina de ETL como um fluxograma "ativo". Esse modelo pode então ser usado pela equipe de teste ou de ETL com a exclusiva finalidade de testar e testar novamente essa rotina.

2) Derive casos de teste automaticamente a partir do modelo de fluxograma

A introdução do teste com base em modelos pode automatizar um dos principais elementos manuais do teste de ETL: o design de casos de teste. Os testadores não precisam mais escrever códigos fantasmas ou copiar manualmente o SQL do banco de dados de origem para o de destino. Em vez disso, os caminhos pelo fluxograma tornam-se os casos de teste, que podem ser usados para lançar dados por meio das regras de transformação. Isso pode ser sistematicamente derivado de um modo que não é possível ao escrever códigos a partir de requisitos estáticos.

Essa derivação automática é possível porque o fluxograma pode ser sobreposto por toda a lógica funcional envolvida em um sistema. Algoritmos matemáticos automatizados podem então ser aplicados para identificar todos os caminhos possíveis pelo modelo, gerando casos de teste que englobam todas as combinações de entradas e saídas (a análise de homotopia ou de causa e efeito pode ser usada para fazer isso).

Como os casos de teste estão vinculados diretamente ao modelo em si, eles englobam toda a lógica definida nele. Portanto, eles fornecem 100% de cobertura funcional, de modo que usar um modelo de fluxograma para trabalhar e ter uma documentação completa seja o mesmo que trabalhar para testar completamente uma rotina de ETL. Uma vantagem adicional desse método é que o teste torna-se mensurável. Como é possível derivar todos os casos de teste possíveis, os testadores podem determinar exatamente qual será a cobertura funcional fornecida por um conjunto de casos de teste específico.

Otimização: teste mais com menos testes

Os algoritmos de otimização automatizada podem ser aplicados para reduzir o número de casos de teste ao mínimo possível, além de reter o máximo de cobertura funcional. Essas técnicas combinatórias são possíveis devido à estrutura lógica do fluxograma, no qual uma única etapa na lógica de causa e efeito (um bloco no componente de teste/fluxograma) pode aparecer em vários caminhos pelo fluxo. Com isso, testar completamente a rotina de ETL torna-se uma questão de testar cada bloco (operador) individualmente, usando uma das várias técnicas de otimização existentes (All Edges [Todas as extremidades], All Nodes [Todos os nós], All In/Out Edges [Todas as extremidades de entrada e saída], All Pairs [Todos os pares]). Por exemplo, um conjunto de 3 casos de teste pode, na verdade, ser suficiente para testar completamente a lógica envolvida em 5 caminhos.

Na empresa de serviços financeiros mencionada anteriormente, isso provou ser extremamente importante para reduzir o excesso de testes e os ciclos dos testes e aprimorar a qualidade deles. Por exemplo, incluindo o tempo necessário para elaborar o fluxograma, foram necessários 40 minutos para criar 19 casos de teste com 95% de cobertura, em contraste aos 150 casos de teste com 80% de cobertura e um excesso de 18 vezes nos testes que eram usados antes. Em outro projeto, foram necessárias 2 horas para criar 17 casos de teste com 100% de cobertura: uma melhoria drástica com relação aos 16% de cobertura que eram obtidos anteriormente em 6 horas.

3) Crie automaticamente os dados necessários para a execução dos testes

Assim que os casos de teste são criados, os testadores precisam de dados que possam englobar 100% dos testes possíveis a fim de executá-los. Esses dados podem ser derivados diretamente do próprio modelo e podem ser criados automaticamente ou extraídos de várias fontes simultaneamente.

Um mecanismo de geração de dados sintéticos, como o CA Test Data Manager, oferece várias maneiras de criar os dados necessários ao usar o teste com base em modelos para impulsionar o teste de ETL. Isso ocorre porque, além da lógica funcional, um fluxograma também pode ser sobreposto por todos os dados envolvidos em um sistema. Em outras palavras, à medida que as regras de ETL forem elaboradas, será possível definir nomes de saída, variáveis e valores padrão para cada nó individualmente. Quando os casos de teste são criados, os dados necessários para executá-los podem ser gerados automaticamente a partir de valores padrão, em conjunto com os resultados esperados relevantes.

Como alternativa, usando o CA Agile Requirements Designer (anteriormente, Grid Tools Agile Designer), os dados do sistema podem ser rapidamente criados com a ferramenta Data Painter. Isso fornece uma lista abrangente de funções de geração de dados combináveis, tabelas de propagação, variáveis de sistema e variáveis padrão. Esses elementos poderão ser usados para criar dados que englobem todos os possíveis cenários, incluindo "dados inválidos" e caminhos negativos. Como cada caminho é simplesmente outro ponto de dados, será possível criar, de modo totalmente sintético, os dados necessários para testar sistematicamente a capacidade de uma rotina de ETL de rejeitar dados inválidos.

Por fim, os dados existentes podem ser localizados em vários sistemas de back-end em questão de minutos, usando a mineração de dados automatizada. Esse processo usa a análise estatística para detectar padrões em bancos de dados, de modo que seja possível extrair grupos de padrões, dependências ou registros incomuns.

4) Provisione os dados em questão de minutos, associados aos testes apropriados

Geralmente, será dada preferência para uma combinação de dados sintéticos e dados de produção existentes, em que a geração sintética é usada para fazer com que haja 100% de cobertura. O essencial para o teste eficiente de ETL é que os dados da "cópia de ouro" sejam armazenados de modo inteligente para que seja possível solicitar, clonar e entregar paralelamente os mesmos conjuntos de dados. Isso elimina os atrasos causados pelas restrições de dados.

A primeira etapa para o armazenamento de dados inteligente é a criação de um data mart de teste, onde é feita a associação dos dados a testes específicos. Para cada teste, são atribuídos dados exatos, sendo que os dados são associados a critérios definidos e estáveis, e não a chaves específicas. Os dados de associação de teste podem eliminar o tempo que seria gasto ao pesquisar dados em grandes fontes de dados de produção, pois é possível recuperar os dados automaticamente a partir do data warehouse de teste ou fazer a mineração deles em questão de minutos a partir de vários sistemas de back-end.

O data warehouse de teste atua como uma biblioteca central, onde os dados são armazenados como ativos reutilizáveis, em conjunto com as consultas associadas que são necessárias para extraí-los. Com isso, os pools de dados podem ser solicitados e recebidos em questão de minutos, vinculados aos casos de teste apropriados e aos resultados esperados. Quanto mais testes forem executados, maior ficará a biblioteca, fazendo com que praticamente toda solicitação de dados seja realizada em uma fração do tempo.

Essencialmente, é possível inserir os dados em vários sistemas simultaneamente e cloná-los à medida que forem provisionados. Isso significa que os conjuntos de dados de inúmeros bancos de dados de origem estão disponíveis a várias equipes paralelamente. O teste de ETL já não é mais um processo linear, e os demorados atrasos gerados pelas restrições de dados são eliminados. Os dados originais poderão ser mantidos à medida que ocorrerem alterações no modelo, possibilitando que as equipes trabalhem em várias releases e versões paralelamente. Esse "controle de versão" também significa que as alterações realizadas nas rotinas de ETL refletem automaticamente nos dados, fornecendo às equipes de teste os dados atualizados de que precisam para testar minuciosamente as transformações.

5) Execute os dados de acordo com as regras e compare os resultados automaticamente

Assim que os testadores já tiverem os testes necessários para testar uma rotina de ETL completamente e os dados necessários para executá-los, a validação também deverá ser automatizada se o teste de ETL tiver a capacidade de acompanhar o ritmo de mudanças constantes dos requisitos.

Usando o mecanismo de orquestração de dados

Uma forma de fazer isso é usando um mecanismo de automação de teste. A CA Technologies oferece a vantagem de atuar como um mecanismo de orquestração de dados, o que significa que os dados, associados aos testes específicos e aos resultados esperados, podem ser obtidos e lançados por meio de uma regra de validação. Essa é a "automação orientada a dados", na qual, por exemplo, um conjunto de testes criado em um mecanismo de orquestração de dados pode extrair cada linha de um arquivo XML, definido no fluxograma, e executá-la como um teste.

Isso fornecerá um resultado de aprovação/reprovação, com base nos resultados esperados definidos no modelo. Isso automatizará a execução de testes e a comparação dos resultados reais com os resultados esperados. Os testadores não precisam mais copiar scripts manualmente do destino para a fonte de dados. Além disso, evita-se o árduo processo sujeito a erros de ter de comparar individualmente cada campo produzido pela transformação.

Usando o CA Test Data Manager

Como alternativa, assim que os resultados esperados forem definidos, o CA Test Data Manager poderá ser usado para criar relatórios de aprovação/reprovação com base nesses resultados automaticamente. Isso automatiza as comparações de dados manuais, porém, o SQL ainda deve ser copiado da origem para o destino.

Primeiro, os dados sintéticos são definidos no sistema de origem, usando as diversas técnicas descritas anteriormente. Depois, será possível criar um pool de dados a fim de simular o processo de ETL, copiando os dados da origem para o destino. É possível copiar um conjunto válido de dados para testar se ele não será rejeitado, bem como copiar um conjunto de dados com erros a fim de garantir que ocorra a rejeição de dados inválidos. Com a criação de uma tabela adicional para armazenar as condições de teste e os resultados e de uma tabela com base em pool de dados com casos de teste e condições de dados, será possível comparar os resultados esperados com os resultados reais automaticamente. Quaisquer dados ausentes ou inválidos poderão ser identificados imediatamente.

6) Implemente as alterações automaticamente

Uma das maiores vantagens do teste com base em modelos para a validação de ETL é a capacidade de reagir a rápidas alterações. Como os casos de teste, os dados e os requisitos estão estreitamente vinculados, uma alteração realizada no modelo poderá refletir automaticamente nos casos de teste e nos dados associados. Essa capacidade de rastreamento significa que, à medida que as rotinas de ETL se tornam cada vez mais complexas, os testes poderão acompanhar esse ritmo e não criarão gargalos no pipeline da entrega de aplicativos.

Com o modelo de fluxograma, a implementação de uma alteração se torna tão rápida e simples quanto a adição de um novo bloco no fluxograma. Os algoritmos de verificação de completude podem ser aplicados a fim de validar o modelo e garantir que cada parte da lógica foi vinculada adequadamente a um fluxo completo. Com o CA Agile Requirements Designer, será possível usar o Path Impact Analyzer para identificar o impacto da alteração nos caminhos pelo fluxograma. Os casos de teste afetados poderão ser removidos ou reparados automaticamente, com quaisquer novos testes necessários para reter a cobertura funcional de 100% gerada automaticamente.

Isso elimina o tempo gasto para verificar e atualizar testes manualmente, sendo que a deduplicação automatizada demonstrou reduzir os ciclos de teste em 30%. Na equipe de business intelligence mencionada anteriormente, o teste com base em modelos proporcionou muita economia de tempo para o processo de ETL. Para verificar e atualizar casos de teste quando uma única regra de ETL era alterada, eram gastas 7,5 horas, sendo que, com o CA Agile Requirements Designer, esse processo demorou apenas 2 minutos. Além disso, do total de casos de teste, apenas 3 realmente tinham sido afetados e precisaram ser atualizados.

Conforme mencionado, com o controle de versão de dados, os testadores também recebem os dados atualizados de que precisam para testar completamente uma rotina de ETL, mesmo depois das alterações. Devido à possibilidade de os dados de teste serem rastreados para o modelo, quando os requisitos forem alterados, essas alterações refletirão automaticamente nos conjuntos de dados relevantes. Os dados ficam disponíveis em todas as releases e versões paralelamente, enquanto os dados necessários para o teste de regressão eficiente são preservados no data warehouse de teste. Os "dados inválidos" raros ou importantes também poderão ser bloqueados para impedir que sejam utilizados por outra equipe e para evitar a perda durante uma atualização.

Seção 4

Resumo

Introduzir um nível maior de automação no teste de ETL é essencial para qualquer organização que trabalha muito para alcançar a entrega contínua de software de alta qualidade. Ainda há um nível muito alto de esforço manual na validação de ETL, da escrita manual de código fantasma aos requisitos estáticos e da aquisição dos dados necessários à comparação dos resultados. O teste com base em modelos e o gerenciamento inteligente de dados de teste podem ser usados para automatizar cada uma dessas tarefas, possibilitando que inúmeras equipes trabalhem paralelamente com as mesmas fontes de dados.

O teste com base em modelos prioriza o esforço do teste de ETL, concentrando a carga do trabalho na fase de projeto. A partir desse ponto, todos os ativos necessários para o teste de ETL podem ser derivados automaticamente em uma fração do tempo. Os casos de teste que fornecem 100% de cobertura funcional podem ser gerados automaticamente e vinculados a resultados esperados definidos de modo independente. Cada teste é "associado" aos dados de origem exatos necessários para executá-lo. Se esses dados forem armazenados no data warehouse de teste, será possível provisioná-los a inúmeras equipes paralelamente.

Devido ao estreito vínculo criado entre os testes, os dados e os requisitos, os testes necessários para testar uma rotina de ETL novamente podem ser rapidamente executados após a realização de uma alteração. O tempo gasto na criação do modelo inicial é, portanto, rapidamente ponderado pelo tempo que se economiza em comparação com o processo manual de criar, atualizar e executar testes novamente. A geração com base em modelos também oferece o excelente benefício de reutilização, em que os componentes de teste podem ser armazenados como ativos compartilháveis e vinculados aos dados e resultados esperados no data warehouse de teste. Quanto maior o número de testes executados, maior ficará a biblioteca, fazendo com que o teste de regras de ETL novas ou atualizadas se torne tão rápido e fácil quanto fazer a seleção a partir de componentes existentes.

O teste de ETL já não cria mais gargalos na entrega de aplicativos e consegue acompanhar o ritmo de evolução das empresas orientadas a dados. A capacidade de teste de rotinas cada vez mais complexas é mantida, de modo que os testes possam lidar com a variedade e o volume de dados coletados. Além disso, ela não impede a entrega contínua de aplicativos de alta qualidade.

Conecte-se à CA Technologies em ca.com/br

A CA Technologies (NASDAQ: CA) cria software que acelera a transformação das empresas e permite que elas aproveitem as oportunidades da economia dos aplicativos. O software está no cerne de todas as empresas, em todos os setores. Do planejamento ao desenvolvimento e do gerenciamento à segurança, a CA está trabalhando com empresas de todo o mundo para mudar a maneira como vivemos, fazemos negócios e nos comunicamos – usando dispositivos móveis, as nuvens privada e pública e os ambientes distribuídos e de mainframe. Para saber mais sobre os programas de sucesso de nossos clientes, visite **ca.com/customer-success**. Obtenha mais informações em **ca.com/br**.

1 Jean-Pierre Dijcks, *Why does it take forever to build ETL processes?*, obtido em 24/07/2015, de: **https://blogs.oracle.com/datawarehousing/entry/why_does_it_take_forever_to_bu**

2 Alan R. Earls, *The State of ETL: Extract, Transform and Load Technology*, obtido em 21/07/2015, de: **http://data-informed.com/the-state-of-etl-extract-transform-and-load-technology/**

3 IBM, obtido em 20/07/2015, de: **http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html**

4 Jacek Becla and Daniel L. Wang, Lessons Learned from managing a Petabyte, P. 4. Obtido em 19/02/2015, de: **http://www.slac.stanford.edu/BFROOT/www/Public/Computing/Databases/proceedings/**

5 **http://etlcode.com/index.php/utility/etl_complexity_calculator**

6 Jean-Pierre Dijcks, *Why does it take forever to build ETL processes?*, obtido em 24/07/2015, de: **https://blogs.oracle.com/datawarehousing/entry/why_does_it_take_forever_to_bu**

7 ETL Guru, *ETL Startegy to store data validation rules*, obtido em 22/07/2015, de: **http://etlguru.com/?p=22**

8 Bender RBT, *Requirements Based Testing Process Overview*, obtido em 05/03/2015, de: **http://benderrbt.com/Bender-Requirements%20Based%20Testing%20Process%20Overview.pdf**

9 Huw Price, *Test Case Calamity*, obtido em 21/07/2015, de: **https://communities.ca.com/community/ca-agile-requirements-designer/blog/2016/02/24/test-case-calamity**

10 Bender RBT, *Requirements Based Testing Process Overview*

11 Software Testing Class, *Why testing should start early in software development life cycle?*, obtido em 06/03/2015, de: **http://www.softwaretestingclass.com/why-testing-should-start-early-in-software-development-life-cycle/**

12 datagaps, *ETL Testing Challenges*, obtido em 24/07/2015, de: **http://www.datagaps.com/etl-testing-challenges**

13 Robin F. Goldsmith, *Four Tips for Effective Software Testing*, obtido em 20/07/2015, de: **http://searchsoftwarequality.techtarget.com/photostory/4500248704/Four-tips-for-effective-software-testing/2/Define-expected-software-testing-results-independently**

14 Jagdish Malani, ETL: *How to handle bad data*, obtido em 24/07/2015, de: **http://blog.aditi.com/data/etl-how-to-handle-bad-data/**

15 Philip Howard, *Automated Test Data Generation Report*, P. 6. Obtido em 22/07/2015, de: **http://www.agile-designer.com/wpcms/wp-content/uploads/2014/10/00002233-Automated-test-case-generation.pdf**

CS200-196717_0416