WHITE PAPER | 2015 年 9 月

ETL テストの完全自動化 ステップバイステップ・ガイド

ca technologies

セクション 1

現在の企業にとってきわめて重要な ETL の役割

データ・ウェアハウスとビジネス・インテリジェンスの世界に登場して以来、ETL（抽出 / 変換 / ロード）はソフトウェア環境に偏在するプロセスとなりました。その名が示すとおり ETL のルーチンは 3 つの異なるステップで構成され、多くの場合それらが並行して実施されます。データは 1 つまたは複数のデータ・ソースから抽出され、必要な状態に変換され、通常はデータ・ウェアハウスやマート、データベースといった希望するターゲットにロードされます。また、作成された ETL ルーチンにはエラーに対応するロギング・インフラストラクチャとルーチン環境が含まれます。[1]

これまで ETL は主として、分析とビジネス・インテリジェンスに大規模な異機種のデータを用意するために使用されてきました。しかし、その使用は単純なデータの移動を超えて広がっており、一般的な用途として新規システムへのデータの移行や、データの統合、分類、結合に使用されることが増えています。[2]

したがって ETL は、常に多数のリリースとバージョンが並行して開発されている、現在の迅速に進む開発サイクルの重要な機能となっています。企業はソフトウェアを常に強化、統合、開発できることが必要で、テスト担当者と開発者がそれぞれの反復とリリースのために適切な状態でデータを利用できるようにすることが重要です。データは同じソースから抽出されますが、個々のチームの特定の要件に応じて変換する必要があります。これは特に、組織が「アジャイル」や継続的デリバリの導入成功を目指している場合に重要なことです。

CA Technologies の顧客事例で、大規模な多国籍銀行の継続的デリバリに ETL が重要な役割を果たした例があります。この銀行は買収プロセスを進めていたところで、買収した銀行の顧客、製品、財務のデータを既存のインフラストラクチャへ移行する必要がありました。これはつまり銀行のバックエンド・システムへアップロードする前に、80 の挿入ファイルを抽出、変換、検証する必要があることを意味しました。さらに、こうしたデータは 47 のプロジェクトにわたり並行して利用可能である必要があり、データの参照整合性を維持することも必要です。[3] この例では、効果的な継続的デリバリに必要な並列処理を可能にするために ETL が非常に重要な役割を果たしました。

しかし、ETL の使用とその重要性の増加にもかかわらず、速度が遅く手作業が多く、なおかつ許容できないほど多くの欠陥が本番まで持ち越されてしまう点で、ETL テストは一般的なテストの状況を反映しています。本書は ETL テストへの通常のアプローチにおいて遭遇する課題について説明し、遭遇した課題の例について検証します。その後、別の広くモデルをベースとしたアプローチについて説明し、それが ETL テストをはるかに効率的、効果的、体系的にできる点について検討します。

セクション 2

ETL テストへの通常のアプローチと共通する課題

ETL 変換ルールを検証する際、テスト担当者は通常、シャドウ・コード・セットを作成し、それを使用してデータを変換し、その後、実際の結果を期待される結果と比較します。通常、ETL スクリプトまたは SQL は手作業でソース・データにコピーされて実行され、結果が記録されます。それから同じスクリプトがターゲット・データにコピーされ、結果が記録されます。その後 2 つのテスト結果（実際と期待される結果）が比較され、データが正しく変換されたことを確認します。

根本的な問題：複雑性と試験容易性

このような手作業による検証の背後にある根本的な課題は、その本質から ETL ルーチンがあっという間に非常に複雑になってしまうことです。ビジネスが発展し、収集するデータの多様性と規模が増えるにしたがい、それに対処するために ETL ルールも拡大します。いわゆる「情報の時代」には、従来のテスト方法が対処できるよりも速いスピードでこの拡大が起きています。実際、データ中心の企業が収集する情報の純粋な量は非常に急速に増加しており、世界のデータの 90% がこの 2 年間だけで収集されたものです。[4] また、平均的な企業が収集したデータ量は毎年倍増しています。[5]

このデータを収集、転送、運用、提示するために設計されたシステムの複雑性は、意思決定を加えるごとに大幅に増大します。これには ETL ルールも含まれ、変換の複雑性に影響する可能性がある膨大な要因が存在します。

- リレーショナルまたは非リレーショナル・データベース・タイプやフラット・ファイルなど、関連するデータ・ソースの数と多様性
- データ・ターゲットの数と多様性
- 単純な検索からアクティブな結合や正規化など、簡単なものも複雑なものも含めた変換の数と多様性
- 再利用可能な変換、コード・スニペット、結合の数
- 作成する表の数 [6]

これらの要因はすべて、リアルタイムに近いソリューションが重視される現在の状況と、これらのソリューションがもたらす複雑性の増大によって悪化しています。[7]

役に立たないドキュメント化

この複雑性の増大は ETL ルーチンの試験容易性に直接影響を及ぼしています。変換ルールは通常、明確な期待される結果を欠いた不十分なドキュメントに保管されているため、ETL テストにとっては特に問題があります。ルールは開発フェーズそのものの間に設計されることが多く、記述されたドキュメントやスプレッドシートに保管されることがよくあります。悪くすると、開発者やテスト担当者の頭の中だけに存在していることもあります。[8] この場合、テスト・ケースを確実に抽出できる実際のドキュメントは存在しません。

CA が支援したあるビジネス・インテリジェンス・チームでは、要件はドキュメントに記述され、テスト・ケースはスプレッドシートに保存されていました。この静的なドキュメントは「言葉の壁」を提示し、ここから ETL ルーチンの論理手順を判読しなければなりませんでした。このドキュメントは「成功した経路」を中心に記録され、ネガティブな状況については含まれていなかったため、平均的なシステムでテストする必要があると考えられる論理の 80% は省略されていました。こうした不完全で曖昧なドキュメントでは、テスト担当者が ETL ルーチンを容易に、あるいは正確に理解することはできませんでした。

テスト担当者が不明点を想像で補わなければならないことがあまりに多くありましたが、そこで間違えた場合、欠陥は ETL ルーチンに入り込むことになります。そのため、コードとテスト・ケースが要件ドキュメントの妥当と思われる解釈を反映していたとしても、無効なデータがターゲットにコピーされていました。

「不完全なデータを入力すれば、不完全な答えしか得られない」– テスト・ケースと期待される結果の手作業による抽出

実際、本番に到達した欠陥のうち 56% という大きな量が、要件ドキュメントの曖昧性に起因している可能性があります。[9] このことは一部には、テスト・ケースと期待される結果が不十分なドキュメントから手作業で抽出されていることが原因です。これは非常に時間がかかるプロセスで、通常はテスト範囲の網羅が不十分になります。

品質

手作業による抽出はその場しのぎで系統立っておらず、たいていはテスト担当者の頭にたまたま浮かんだテスト・ケースが作成されることになりがちです。これまで述べてきた ETL ルーチンの複雑性と、不十分なドキュメントが多いことを考え合わせると、どんなに優秀なテスト担当者でも考えられるデータの組み合わせの検証に必要なすべてのテストを考案することは不可能だといえるでしょう。たとえば、32 個のノードと 62 個のエッジがある簡単なシステムを線形的に設計すると、その論理から 1,073,741,824 個の経路が考えられますが、これは人間が考え出せる数を超えています。

そのため、その場しのぎの抽出はテストの大幅な不足や過剰につながり、ETL ルーチンに関連すると考えられる論理のほんの一部しかテストすることができません。ネガティブなテストは特に課題で、ドキュメントと同じくテスト・ケースは通常、うまくいく経路のみに焦点を当てるからです。しかしテストする必要があるのは異常値や予期されない結果です。なぜなら ETL ルーチンはこうした「悪いデータ」を拒否しなければならないからです。

CA Technologies の顧客のある金融サービス会社は 11 のテスト・ケースを使用し、テスト網羅率は 16% でした。これは標準的な数値で、当社の監査によると機能テスト網羅率の平均は 10 ～ 20% でした。この会社の別のプロジェクトでは 18 倍も過剰にテストが行われていました。システムを十分にテストしようとしてテスト・ケースを積み重ねた結果ですが、それでも最大の網羅率を得られてはいませんでした。150 の余分なテスト・ケースを外注業者を使用して実行したところ、26,000 ドルもかかりました。[10]

このような不十分なテスト網羅率の結果、コードに欠陥が発生し、修正に高額のコストと時間がかかります。調査によると、バグをテスト中に修正するには、早期に発見した場合に比べて 40 ～ 1000 倍のリソースを必要とし[11]、50 倍の時間がかかります。[12] さらに悪い場合はバグを検出できない可能性もあり、そのため無効なデータがライブ・ターゲットにコピーされ、システムの整合性が脅かされる可能性があります。また、静的なドキュメントに直面すると、テスト担当者はテスト・ケースの網羅率を測定するための信頼できる方法がないため、任意のルーチンのどの程度をテストできているか、あるいはほとんどテストできていないかについて確実に判断できません。また、重要度に基づきテストの優先順位を決定することもできません。

時間と労力：対応できていないテスト

こうしたドキュメントからテスト・ケースを作成することもまた、多くの時間と労力を必要とします。前の例では 11 のテスト・ケースを作成するために 6 時間を要しましたが、過剰なテストのまん延によってさらに多くの時間をとられていました。手作業のテスト・ケース設計に費やされるこの時間は、その後に実際の結果と期待される結果の比較に費やさなければならない時間でさらに長くなります。

複雑な ETL ルーチンによって生成されるデータの量と、ソース・データの多くが多様なデータベースやファイル・タイプに保管されていることを考えると、膨大な個別のフィールドと期待される結果との比較は非常に時間がかかる作業です。また、変換データは以下のように複数のレベルで検証する必要があるため、難度も高くなります。

- テスト担当者はデータ・ソースとターゲットの数が一致することを確認し、データの完全性を確認する必要があります。
- ターゲット・データがソース・データと一致していることを確認し、データの整合性を確保する必要があります。
- 変換はビジネス・ルールに従わなければなりません。
- 予期しない重複を識別し、データの一貫性を保証する必要があります。
- 孤立したレコードや外部キーの欠如を特定し、参照整合性を維持する必要があります。[13]

ときには妥協が行われ、サンプル・データセットだけが検証されることがあります。しかしこれは ETL テストの完全性も損ない、変換の信頼性も影響を受けます。多くの ETL ルーチンのビジネス・クリティカルな運用における役割を考えると、このような妥協は許容できません。エラーの起きやすい手作業による比較によって、品質はさらに影響を受けます。特に期待される結果の定義が不十分な場合、あるいはさらに悪いことにテストで使用されるシャドウ・コードと切り離して定義されていない場合も、品質に影響します。この場合、テスト担当者は実際の結果が特に異常なものでない限り、テストは合格したとみなす傾向があります。期待される結果を事前定義しなければ、実際の結果が期待される結果であるとみなす可能性が高いため[14]、データの妥当性を確実に判断できません。

データの問題

ここまでは、ETL ルールを検証するために必要なテスト（シャドウ・コード）を抽出する際に遭遇する課題に焦点を当ててきました。しかしテスト・ケースを抽出した後、テスト担当者はシステムに供給するためのダミーのソース・データを必要とします。これがボトルネックと欠陥の別のよくある原因となっています。

複雑な ET ルーチンをテストするために必要なすべてのデータが揃っていますか？

「悪い」データが十分あることは、効果的な ETL テストにとって非常に重要です。運用時に ETL ルールはこのデータを拒否し、適切なユーザに適切な形式で送信する点でこれは非常にすぐれているからです。「悪い」データが拒否されないとしたら、こうした悪いデータは欠陥に発展する可能性が高く、システムの崩壊にさえつながることがあります。

このコンテキストでの「悪いデータ」はいくつかの方法で定義でき、それはテスト担当者がデータを検証する必要がある方法に対応しています。これはビジネス・ルールに基づけば許容すべきではないデータである可能性があります。たとえば、オンライン・ショッピング・カートでのクーポンがないときのマイナスの値がこれにあたります。相互依存データや必須データがない、あるいは入信データ自体のデータが欠落しているなど、ウェアハウスの参照整合性を脅かすデータの可能性もあります。[15] したがって、ETL 検証ルールに供給されるテスト・データは、100% の機能テスト網羅率を可能にするために広範な無効データを含んでいる必要があります。

このようなデータは、多くの企業でいまだにテスト・チームにプロビジョニングされている本番データ・ソースにはほとんど見つかりません。これは、本番データが過去に起きた「通常のビジネス」シナリオから抽出されており、その性質上、悪いデータが排除されているためです。ETL テストに必要な予期しない結果や異常値、境界条件は含まれず、「成功した経路」を中心に構成されています。実際、本番データの監査によると、わずか 10 ～ 20% の網羅率が平均的となっています。皮肉なことに、構築されたルーチンが優れたものであればあるほど「悪いデータ」は少なくなるため、ETL ルールを完全にテストするために十分な多様性のあるデータも少なくなります。

必要としているときにデータを使用できますか？

ETL の検証に関するもう 1 つの大きな問題は、データの可用性です。ソース・データは企業全体の 50 もの異なるソースから抽出されることがあります。ETL テストおよびテスト一般の問題は、テストが一連の線形的な段階としてとらえられているため、テスト・チームは他のチームがデータを使用している間、待機しなければならないことです。

銀行の移行チェーンを例にとると、データは 1 つの銀行から抽出され、リコンサイル・ツールを使用して別のシステムに変換されます。各段階でデータを検証する必要があり、データが金融管理の枠組みに正しく変換されたこと、口座番号が抽出されたこと、時間的に正確であることなどを確認する必要があります。このプロセスは基本的な入力、重複排除、準備、伝搬、データの登録など、複数の異なるフェーズに分類されることがあります。ETL チームや ETL 以外のチーム、特にメインフレームで作業しているチームなど、さらに多くのチームが関わることもあります。

すべてのチームが並行して企業全体のすべてのデータからデータを利用できなければ、チームは手をこまねいて待機するだけになり遅延が増大します。データを他のチームが使用しているために、テスト担当者はゴースト・コードを作成するようになり、ソース・データを使用して ETL ルールを検証することがなくなります。実際、平均的なテスト担当者は 50% の時間をデータの待機、検索、操作、作成に費やしている可能性があることがわかっています。これは SDLC 全体の 20% という大きな割合を占めている可能性があります。

ルールが変わると何が起きるか？

テスト・ケースとデータを静的な要件から手作業で抽出していると、変化への対応力が乏しくなります。ビジネスが進化すると ETL ルーチンは即座に変化し、収集するデータの規模と多様性もそれとともに拡大します。しかしこのように絶えず変化が起きていると、ETL テストはそれについていけません。

この場合、プロジェクトの遅れの唯一最大の原因は、ルーチンの変更時に既存のテスト・ケースを確認し更新しなければならないことです。テスト担当者には静的な要件とテスト・ケースに加えられた変更の影響を自動的に識別する方法がありません。網羅率が実際に維持されているかどうか測定する方法もなく、既存のテスト・ケースをすべて手作業で確認しなければなりません。

前述のビジネス・インテリジェンス・チームでは、要件とテスト・ケースがドキュメントとスプレドシートに記述されてそれぞれ保管されていましたが、変更は特に大きな問題でした。1 つの ETL ルールに変更が行われると、テスト担当者が一連のテスト・ケースを確認し更新するのに 7.5 時間もかかっていたのです。CA の別の顧客の例では、要件に変更があった場合、テスト担当者は 2 日間もかけて既存のすべてのテスト・ケースを確認していました。

セクション 3

実行可能な代替案：ETL テストの完全自動化

つまり、テスト・ケースが手作業で抽出され、結果が手作業で比較されている限り、ビジネス要件の絶え間ない変化のスピードに ETL テストが対応できないことは明らかです。以下に ETL テストの効率性と効果を改善するために考えられる戦略を示します。これはモデルベースで要件ベースの戦略で、テスト作業を前倒しし、最初から ETL ライフサイクルに品質を組み込むよう設計されています。こうしたモデルベースのアプローチはテストと開発のすべての段階に自動化を導入し、また、ETL テストが絶え間ない変化に完全に対応できるようにします。

1）正式なモデルで始める

正式なモデル化を ETL テストに導入すると、テスト作業が前倒しされ、ETL ルールのモデルへの最初のマッピング作業からその後のテスト / 開発アセットをすべて抽出できるようになります。したがって、正式なモデルは ETL 検証の完全自動化の土台となります。

ただし正式なモデル化は要件の曖昧性や不完全さという、上記に挙げたもっと固有の課題を解決するためにも役立ちます。ETL ルールの複雑性が増大し続けるにもかかわらず、正式なモデル化は試験容易性を維持できるため、テスト担当者は迅速かつ視覚的にテストの必要な論理を正確に把握できます。また、変換ルールを十分にテストするために入力すべき有効なデータと無効なデータの両方を容易に把握でき、各インスタンスで期待される結果がどのようなものであるべきかを理解できます。

たとえば、フローチャート・モデルは扱いにくい「言葉の壁」を理解しやすい塊に分割します。フローチャート・モデルは ETL を原因と結果の論理にまとめ、「もし～なら・・・する」という一連のステートメントにマッピングし、それをプロセスの階層に関連付けます。[16] 実質的にこれらの手順のそれぞれがテスト・コンポーネントとなり、テスト担当者は何を検証すべきかを正確に理解できます。したがって ETL ルーチンをフローチャートとしてモデル化すれば、要件ドキュメントから曖昧性を排除でき、曖昧性から生じる欠陥の 56% を回避するのに役立ちます。

ETL ルーチンがますます複雑化する中で、フローチャートは単一の参照ポイントとして役に立ちます。「静的」なドキュメントと図に比べ、容易に論理をモデルに追加できます。さらに、サブフロー・テクノロジを使用して高度に複雑なルーチンの抽象化が可能なため、試験容易性が向上します。低いレベルのコンポーネントはマスター・フローに組み込むことができ、高度に複雑な ETL ルールのセットを構成する膨大なルーチンを 1 つの視覚的な図に統合できます。

フローチャートによるモデル化は曖昧性の軽減に加え、不完全性の排除にも役立ちます。フローチャートによるモデル化では、モデル作成者は制約やネガティブな条件、制限、境界条件などの観点から考え、「この原因やトリガがない場合はどうなるか」について考慮せざるを得ません。そのためモデル作成者はネガティブな経路を系統的に考案し、ETL 検証の大部分を構成すべきネガティブなテストを提供する必要があります。正式なモデルは ETL ルールの数学的に正確な図であるため、さらに完全性チェック・アルゴリズムを適用できます。

その結果「宙ぶらりん else 問題」のような論理の欠如が排除され、考えられるデータの組み合わせの 100% を網羅するテスト・ケースを抽出できます（ただしほぼ必ず、テストとして実行可能であるより多くの組み合わせが存在することに注意する必要があります。そのため後ほど最適化技術についても説明します）。もう 1 つの大きなメリットは、モデルでは期待される結果をテスト・ケースとは別に定義できることです。言い換えるとフローチャートを使用すれば、境界入力をモデルに含め、期待される結果がモデル内のさまざまなエンド・ポイントに伝搬されるようモデルを定義できます。これは検証ルールによって何が許容されるべきで何が拒否されるべきかを明確に定義します。そのため、テスト担当者は期待された結果が明確でない場合にテストが合格したとみなす間違いを回避できます。

ETL 検証にモデルベースのテストを採用するときは、企業全体のテストおよび開発に要件主導のアプローチを全面的に採用する必要がないことは特筆すべきです。大規模な変更は必要なく、当社の経験では ETL ルーチンを「アクティブ」なフローチャートとしてモデル化するのにわずか 90 分しかかかりません。その後このモデルは ETL またはテスト・チームによって、テストとそれ自体をルーチン化する再テストの目的のみに使用できます。

2）テスト・ケースをフローチャート・モデルから自動抽出

モデルベースのテストを導入すると、ETL テストの主要な手作業の要素の 1 つである、テスト・ケースの設計を自動化できます。テスト担当者はゴースト・コードを作成する必要がなくなり、ソースからターゲット・データベースへ手作業で SQL をコピーする必要もありません。その代わりフローチャートの経路がテスト・ケースになり、変換ルールにデータを供給するために使用できます。これらは系統的な抽出が可能です。これは静的要件からコードを作成していたときには不可能だったことです。

この自動抽出は、フローチャートにはシステムに含まれるすべての機能論理をオーバーレイできるため可能になります。次に自動化された数学的アルゴリズムを適用でき、モデル全体の可能な経路すべてを識別し、入力と出力のあらゆる組み合わせを網羅するテスト・ケースを生成できます（これを行うために原因と結果またはホモトピー分析を使用可能）。

テスト・ケースはモデル自体と直接関連付けられるため、モデルで定義された論理をすべて網羅できます。そのため 100% の機能網羅率の確保が可能です。つまりフローチャート・モデルを使用して完全なドキュメント化をめざして取り組むことは、ETL ルーチンを完全にテストすることをめざして取り組むことと同じなのです。この方法のさらなるメリットは、テストが測定可能になることです。考えられるあらゆるテスト・ケースを抽出できるため、テスト担当者は任意のテスト・ケースのセットが提供する機能網羅率を正確に判断できます。

最適化：より少ないテストでより多くをテスト

次に自動最適化アルゴリズムを適用することで、機能の網羅率は最大限に維持しながらテスト・ケースの数を最小限に抑えられます。このような手法の組み合わせは、フローチャートの論理構造によって可能になっており、原因結果論理（フローチャート / テスト・コンポーネントのブロック）の 1 つのステップをフロー全体の複数の経路に使用できます。その結果、ETL ルーチンを十分にテストすることは、複数の既存の最適化手法（All Edges、All Nodes、All In/Out Edges、All Pairs）のうち 1 つを使用して個別のブロック（演算子）をそれぞれテストすることになります。たとえば 3 つのテスト・ケースのセットは、実際には 5 個の経路に含まれる論理を十分にテストすれば十分な場合があります。

前述した金融サービス会社では、これは過剰なテストを減らし、テスト・サイクルを短縮し、テスト品質を改善するためにきわめて有益であることがわかりました。たとえばフローチャートのモデル化にかかった時間を含め、95% を網羅する 19 のテスト・ケースを作成するのにかかった時間は 40 分でした。以前使用していた 80% の網羅率で 18 倍も過剰なテストからなる 150 のテスト・ケースと比較すると大幅な短縮です。別のプロジェクトでは、100% の網羅率の 17 のテスト・ケースを作成するのにかかった時間は 2 時間でした。以前は 6 時間かけて 16% の網羅率でしたが、大幅に向上しました。

3）テストの実行に必要なデータの自動作成

テスト・ケースを作成したらそれを実行するために、テスト担当者は考えられるテストの 100% を網羅できるデータを必要とします。このようなデータはモデルそのものから直接抽出することも可能で、自動作成するか、あるいは同時に複数のソースから抽出することもできます。

CA Test Data Manager のような合成データの生成エンジンは、ETL テストを推進するためのモデルベースのテストの使用時に必要なデータを作成するための、複数の方法を提供します。これは、フローチャートには機能論理に加え、システムに関連するすべてのデータをオーバーレイできるからです。つまり ETL ルールがモデル化される際、個別のノードそれぞれに対して出力名、変数、デフォルト値を定義できます。テスト・ケースが作成されると、それを実行するために必要なデータが、適切な期待される結果とともにデフォルト値から自動生成されます。

また、CA Test Case Optimizer を使用すると、データ・ペインタ・ツールを使用してシステム・データを迅速に作成できます。これは結合可能なデータ生成機能、シード表、システム変数、デフォルト変数の包括的なリストを提供します。これらは「悪いデータ」やネガティブな経路を含め、あらゆる可能なシナリオを網羅するデータの作成に使用できます。それぞれの経路は別のデータ・ポイントであるため、無効なデータを拒否する ETL ルーチンの機能を系統的にテストするために必要なデータを完全に合成できます。

最後に、自動データ・マイニング機能を使用すると、既存のデータを複数のバックエンド・システムから数分で検出できます。これは統計分析を使用してデータベースのパターンを検出し、パターン、依存性、異常なレコードのグループを抽出できます。

4）正しいテストに応じたデータを数分でプロビジョニング

通常、網羅率を 100% にするために合成機能が使用される場合は、既存の本番データと合成データを組み合わせることが理想的です。効率的な ETL テストにとって重要なのは、「ゴールド・コピー」データがインテリジェントに保管され、同じデータ・セットを並行して要求、クローン化、提供できるようにすることです。これによって、データの制約が原因の遅延を排除できます。

インテリジェントなデータ・ストレージの最初の手順はテスト・マートの作成で、データが特定のテストに「適合」されます。データは特定のキーではなく、定義された安定した基準に適合され、各テストは正確なデータを割り当てられます。データをテスト・データ・ウェアハウスから自動的に抽出でき、また、数分で複数のバックエンド・システムからマイニングできるため、テストに適合したデータは大規模な本番データ・ソースの間でデータの検索に費やされる時間を節約できます。

テスト・データ・ウェアハウスは中央ライブラリとして機能し、データはその抽出に必要な関連するクエリとともに、再利用可能なアセットとして保管されます。データ・プールは要求して数分で受け取ることができ、適切なテスト・ケースと期待される結果に関連付けられます。実行されるテストが多くなればなるほど、このライブラリは大きくなり、実際にすべてのデータ要求を短時間で実行できるようになります。

重要なことに、データは複数のシステムに同時に供給され、プロビジョニング時にクローン化されます。これは、膨大なソース・データベースのデータ・セットを複数のチームが並行して利用できることを意味します。ETL テストは線形的なプロセスではなくなり、データの制約によって起きる長い遅延は排除されました。変更はモデルに対して行われるため元のデータを保持でき、チームは複数のリリースやバージョンに対して並行して作業することが可能です。また、「バージョン・コントロール」は ETL ルーチンへの変更が自動的にデータに反映され、テスト・チームに変換を厳密にテストするために必要な最新のデータを提供することを意味します。

5）ルールに対してデータを実行し、自動的に結果を比較

ETL ルーチンを十分にテストするために必要なテストと、それらを実行するために必要なデータをテスト担当者が確保したら、ETL テストが変化する要件に対応できるようにするには、検証そのものを自動化する必要があります。

データ・オーケストレーション・エンジンの使用

これを行う 1 つの方法は、テスト自動化エンジンを使用することです。CA Technologies は、データ・オーケストレーション・エンジンとして倍化するメリットを提供します。つまり、特定のテストと期待される結果に合致するデータを取得し検証ルールに供給できます。これは「データ中心の自動化」で、たとえばデータ・オーケストレーション・エンジンで作成されたテスト・ハーネスがフローチャートで定義された XML ファイルの各行を取得し、テストとしてそれを実行できます。

その後、これはモデルで定義された期待される結果に基づき、合否を提供します。これはテストの実行と、実際の結果と期待される結果の比較を自動的に行います。テスト担当者はデータ・ターゲットからデータ・ソースにスクリプトを手作業でコピーする必要がなくなり、変換によって生成された個別のフィールドをすべて比較するという手間がかかりエラーの起きやすいプロセスを回避できます。

CA Test Data Manager の使用

また、期待される結果が定義されたら、それに基づいて CA Test Data Manager を使用して合否レポートを自動生成できます。これは手作業で行っていたデータの比較を自動化しますが、ただし SQL はソースからターゲットにコピーする必要があります。

まず、前述したさまざまな手法を使用して、合成データがソース・システムで定義されます。次にデータ・プールが作成され、ETL プロセスをシミュレートし、ソースからターゲットへデータがコピーされます。有効なデータのセットがコピーされて、このセットが拒否されなかったことがテストされ、同時に、無効なデータが拒否されたことを確認するためにエラーを含むデータ・セットもコピーされます。テストの条件と結果を保管するためにさらにテーブルを作成することで、テスト・ケースとデータ条件を含みデータ・プールに基づくテーブルを作成することで、期待される結果を実際の結果と自動的に比較できます。これによって、一目見てデータの欠落やエラーを識別できます。

6）変更を自動的に実装

ETL 検証のモデルベースのテストの最大のメリットの 1 つは、速度の速い変更に対応できることです。テスト・ケース、データ、要件が緊密に関連付けられているため、モデルへの変更は自動的にテスト・ケースと関連データに反映されます。この追跡可能性によって、急速に複雑性を増す ETL ルーチンにテストが対応でき、アプリケーション・デリバリ・パイプラインにボトルネックを作りません。

フローチャート・モデルを使用することで、変更の実装は新しいブロックをフローチャートに追加するだけですばやく簡単に行えるようになります。その後、完全性チェック・アルゴリズムを適用してモデルの検証を行い、論理のすべての部分が完全なフローに適切に接続されたことを確認できます。CA Test Case Optimizer を使用している場合、経路インパクト・アナライザを使用してフローチャート全体での経路への変更の影響をさらに識別できます。影響を受けたテスト・ケースは削除されるか自動的に修復され、100% の機能網羅率を維持するために必要な新しいテストが自動生成されます。

これによって手作業によるテストの確認や更新にかかる時間が排除され、また、自動重複排除によってテスト・サイクルが 30% 短縮されることが証明されています。前述したビジネス・インテリジェンス・チームでは、モデルベースのテストによって ETL プロセスにかかる時間が大幅に短縮されるようになりました。1 つの ETL ルールが変更されるとテスト・ケースの確認と変更に 7.5 時間かかっていたのに対し、CA Test Case Optimizer では 2 時間しかかかりません。また、テスト・ケース全体で、実際に影響を受け更新が必要になったテスト・ケースは 3 つしかありませんでした。

前述のとおりデータのバージョン・コントロールでは、ETL ルーチンを十分にテストするために必要な最新のデータが、変更後でもテスト担当者に提供されます。テスト・データをモデルまで追跡できるため、要件が変更されると変更は自動的に関連するデータ・セットに反映されます。データはリリースとバージョン全体で並行して利用でき、効率的な回帰テストに必要なデータはテスト・データ・ウェアハウスに保存されます。また、関心の高いデータあるいはまれに発生する「悪いデータ」をロックできるため、それが別のチームに使い尽くされたり、データ更新時に失われたりすることを防ぎます。

セクション 4

まとめ

ETL テストに高度な自動化を導入することは、高品質なソフトウェアの継続的デリバリに注力している企業にとって必須事項です。手作業による静的要件からのゴースト・コードの作成や必要なデータの抽出、結果の比較など、特に高度な手作業が ETL 検証ではまだ使用されています。モデルベースのテストとインテリジェントなテスト・データ管理機能を使用すれば、こうした手作業のそれぞれをすべて自動化でき、また、多数のチームが並行して同じデータ・ソースを使用して作業を行うことが可能です。

モデルベースのテストは ETL テストの作業を前倒しし、大部分を設計フェーズに集中させます。そこから ETL テストに必要なすべてのアセットを、わずかな時間で自動的に抽出できます。100% の機能網羅率を確保するテスト・ケースを自動的に生成し、個別に定義した期待される結果に関連付けできます。テスト・データ・ウェアハウスに保管されていれば、各テストはそれを実行するために必要な正確なソース・データにさらに「適合」され、多数のチームに並行してプロビジョニングできます。

テスト、データ、要件の間に構築された緊密なリンクのために、変更が行われた後、即座に ETL ルーチンの再テストに必要なテストを実行できます。したがって最初のモデルの作成に費やした時間よりも、手作業でテストを作成、更新、再実行しなければならない場合と比較して短縮された時間の方がすぐに重要になります。モデルベースの生成はまた、データおよび期待される結果に関連付けられた共有可能なアセットとしてテスト・コンポーネントをテスト・データ・ウェアハウスに保管できるため、再利用可能性の大きなメリットを提供します。多くのテストを実行するほどライブラリは拡大し、最終的には新規または更新された ETL ルールのテストは既存のコンポーネントからの選択と同じくらい迅速で容易に行えるようになります。

ETL テストはアプリケーション・デリバリのボトルネックを引き起こさなくなり、データ中心のビジネスの拡大速度に対応できるようになりました。かつてないほど複雑化したルーチンの試験容易性が維持されるため、テストは収集したデータの多様性と規模に対処でき、高品質なアプリケーションの継続的デリバリを妨げることがありません。

セクション 5

CA Technologies のメリット

CA Technologies（NASDAQ：CA）は、複雑な IT 環境の管理と保護に役立つ IT 管理ソリューションを提供し、アジャイル開発のビジネス・サービスを支援します。CA Technologies のソフトウェアと SaaS ソリューションを活用することで、データセンタからクラウドに至るまで革新を加速し、インフラストラクチャを変革し、データとアイデンティティを保護できます。CA Technologies はそのテクノロジにより、お客様が必要な成果と期待どおりのビジネス・バリューを実現できるようにします。お客様を成功に導くプログラムの詳細については、**ca.com/jp/customer-success** をご覧ください。CA Technologies の詳細については、**ca.com/jp** をご覧ください。

ca.com/jp/でCA Technologiesにアクセスしてください

CA Technologies（NASDAQ:CA）は、企業の変革を推進するソフトウェアを作成し、アプリケーション・エコノミーにおいて企業がビジネス・チャンスを獲得できるよう支援します。ソフトウェアはあらゆる業界であらゆるビジネスの中核を担っています。プランニングから開発、管理、セキュリティまで、CA は世界中の企業と協力し、モバイル、プライベート・クラウドやパブリック・クラウド、分散環境、メインフレーム環境にわたって、人々の生活やビジネス、コミュニケーションの方法に変化をもたらしています。詳細については **ca.com/jp** をご覧ください。

1 Jean-Pierre Dijcks、「**Why does it take forever to build ETL processes?**」、**https://blogs.oracle.com/datawarehousing/entry/why_does_it_take_forever_to_bu** より 2015 年 7 月 24 日に取得

2 Alan R. Earls、「**The State of ETL:Extract, Transform and Load Technology**」、**http://data-informed.com/the-state-of-etl-extract-transform-and-load-technology/** より 2015 年 7 月 21 日に取得

3 Grid-Tools、「**Synthetic Data Creation at a Multinational Bank**」、**https://www.grid-tools.com/resources/synthetic-data-creation-multinational-bank/** より 2015 年 7 月 21 日に取得

4 IBM、**http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html** より 2015 年 7 月 20 日に取得

5 Jacek Becla および Daniel L. Wang、「Lessons Learned from managing a Petabyte」、P.4、**http://www.slac.stanford.edu/BFROOT/www/Public/Computing/Databases/proceedings/** より 2015 年 2 月 19 日に取得

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

7 Jean-Pierre Dijcks、「**Why does it take forever to build ETL processes?**」、**https://blogs.oracle.com/datawarehousing/entry/why_does_it_take_forever_to_bu** より 2015 年 7 月 24 日に取得

8 ETL Guru、「**ETL Startegy to store data validation rules**」 **http://etlguru.com/blog/?p=22** より 2015 年 7 月 22 日に取得

9 Bender RBT、「**Requirements Based Testing Process Overview**」、**http://benderrbt.com/Bender-Requirements%20Based%20Testing%20Process%20Overview.pdf** より 2015 年 3 月 5 日に取得

10 Huw Price、「**Test Case Calamity**」、**https://www.grid-tools.com/test-case-calamity/** より 2015 年 7 月 21 日に取得

11 Bender RBT、「**Requirements Based Testing Process Overview**」

12 Software Testing Class、「**Why testing should start early in software development life cycle?**」、**http://www.softwaretestingclass.com/why-testing-should-start-early-in-software-development-life-cycle/** より 2015 年 3 月 6 日に取得

13 datagaps、「**ETL Testing Challenges**」、**http://www.datagaps.com/etl-testing-challenges** より 2015 年 7 月 24 日に取得

14 Robin F. Goldsmith、「 **Four Tips for Effective Software Testing**」、**http://searchsoftwarequality.techtarget.com/photostory/4500248704/Four-tips-for-effective-software-testing/2/Define-expected-software-testing-results-independently** より 2015 年 7 月 20 日に取得

15 Jagdish Malani、「ETL:**How to handle bad data**」、**http://blog.aditi.com/data/etl-how-to-handle-bad-data/** より 2015 年 7 月 24 日に取得

16 Philip Howard、「**Automated Test Data Generation Report**」、P.6、**http://www.agile-designer.com/wpcms/wp-content/uploads/2014/10/00002233-Automated-test-case-generation.pdf** より 2015 年 7 月 22 日に取得

CS200-154751