

Contents

< コードを使用してビジネス ロジックを適用する

スケーラブルなカスタマイズ設計

データベース トランザクション

同時実行問題

自動付番の例

トランザクション設計パターン

コードを使用してビジネス ロジックの適用

2022/07/21 •

可能な限り、ビジネスロジックを定義または適用するいくつかの宣言型プロセス オプションの 1 つを適用することを最初に検討する必要があります。詳細: [Microsoft Dataverse でビジネスロジックを適用](#)

宣言型プロセスが要件を満たさない場合、開発者にはいくつかのオプションがあります。このトピックではコードを記述する共通オプションを紹介します。

プラグインの作成

プラグインと呼ばれるカスタムイベントハンドラーを作成し、Dataverse のサーバーに登録することができます。プラグインは、Dataverse データベース トランザクションの特定のイベントで実行されるように登録されています。プラグインを実行すると、現在のデータベース トランザクションで処理されているデータの作成、読み取り、変更、削除が可能になります。このように、プラグインを使用すると、Dataverse のデータ処理をカスタマイズまたは拡張できます。

詳細: [ビジネス プロセスを拡張するためのプラグインの作成](#)

ワークフロー拡張の作成

カスタム ワークフロー アクティビティを記述おとび登録することで、プロセス デザイナー内で追加のアクションを提供することができます。作成したアクションは、ワークフロー デザイナーでユーザーが適用できるようになります (条件や新しい操作など)。このようにして、ご利用の環境のユーザー向けに、プロセス デザイナーに新しいカスタムアクションを追加することができます。

詳細: [ワークフロー拡張](#)

関連項目

[Dataverse 開発者の概要](#)

NOTE

ドキュメントの言語設定についてお聞かせください。 [簡単な調査を行います](#)。(この調査は英語です)

この調査には約 7 分かかります。個人データは収集されません ([プライバシー ステートメント](#))。

Microsoft Dataverse でのスケーラブルなカスタマイズ設計

2022/07/21 •

NOTE

これはスケーラブル カスタマイズ設計に関する 1 つめのトピックです。この内容は個別のトピックに分かれていますが、スケーラブル カスタマイズの設計に関する概念、問題、および戦略の全体像を示します。各トピックの内容は、先行のトピックで説明された概念に基づいています。データをオフラインで読みたい場合は、[これらのトピックを単一の PDF 文書としてトピックをダウンロード](#) できます。

Dataverse は、要求を行っているユーザーに対する応答時間およびシステムの安定性と応答性の両方に影響を与える可能性のある長時間実行中の活動から、ユーザーやシステム自体を保護するように設計されています。

Dataverse のソリューションを実装している一部のユーザーが直面する課題として、これらの保護対策が実施された場合にプラットフォームや基盤となる Microsoft SQL Server データベースによってスローされるエラーがあります。これは多くの場合、プラットフォームが拡張できない、不正な終了、システムへの要求のスロットリングとして解釈されます。

この内容は、このような多くの主要な課題について根本的な原因を調査し、解決してきた経験に基づいています。以下のトピックでは、システムに対するこれらの要求からプラットフォームが自身を保護する方法を説明します。さらに、ほとんどの場合において、この動作がプラットフォーム内でブロックおよびトランザクションを使用することの影響を理解していないカスタム実装の結果として生じている理由について説明します。

また、このような動作を回避するためにカスタム実装を最適化することが、プラットフォーム エラーを回避するだけでなく、結果的にパフォーマンスの向上とユーザー エクスペリエンスの向上につながることも説明します。優れた設計のプラクティスを示し、回避すべき一般的なエラーを特定します。

課題

特定の種類のエラーや現象がシステム内で発生する場合は、一般的にこの領域で課題を調査し解決することから始めます。これらはプラットフォームの問題であると認識されます。必要な救済ステップは、通常、報告対象となる時間のかかる要求をトリガーするプラットフォームの制約を緩和することです。

実際に、プラットフォームの制約の一部を緩和することでエラーを短期的に回避できます。ただし、これらの制約にはもっともな理由があり、非常に時間のかかるアクションが別のユーザーやシステム パフォーマンスに影響することを防ぐために設計されています。制約を緩和してエラーを回避できますが、それでもこのユーザーにとっての応答時間が遅くなり、それがシステムのほかのユーザーのエクスペリエンスにも影響することはあります。

したがって、なぜ制約がトリガーされてエラーを引き起こしているのか、その根本的な原因を調べてから、コードのカスタマイズを最適化して回避することが推奨されます。これにより、一貫性と応答性の優れたシステムをユーザーに提供できます。

共通した現象

次の表に示すように、これらの種類の問題には共通した現象がみられます。

■	■
■	ユーザーは、システムの特定の領域 (特定のフォームやクエリなど) で応答時間が遅いことを認識します

「	「
■ SQL ■	特定のアクションは、一般的な SQL エラーを報告するプラットフォームエラー レポートで応答します。 ほとんどの場合、これはプラットフォーム層で SQL タイムアウトに変換します。
■	アクションを強制的に終了してロールバックするデッドロックが発生したことを報告するプラットフォーム エラーです。
■	特にバッチの読み込みシナリオにおいて、極端に遅いスループットが見られます。
■ / ■	このような動作の重要なインジケータは、同じアクションの処理速度が非常に速い、または極めて遅い場合は、再試行してより速く動作するか、エラーを回避します

実際、問題の発生時には、これらの現象が組み合わせて発生し、まとめて報告されます。これらの現象が必ずしも設計の問題を示すわけではありません。データベースのディスク I/O 操作の制限や製品のバグなど、そのほかの問題により、同様の現象が発生する可能性があります。ただし、これらの現象の最も一般的な原因であるため、カスタム実装の設計との直接的な関係とシステムへの影響について確認する価値はあります。

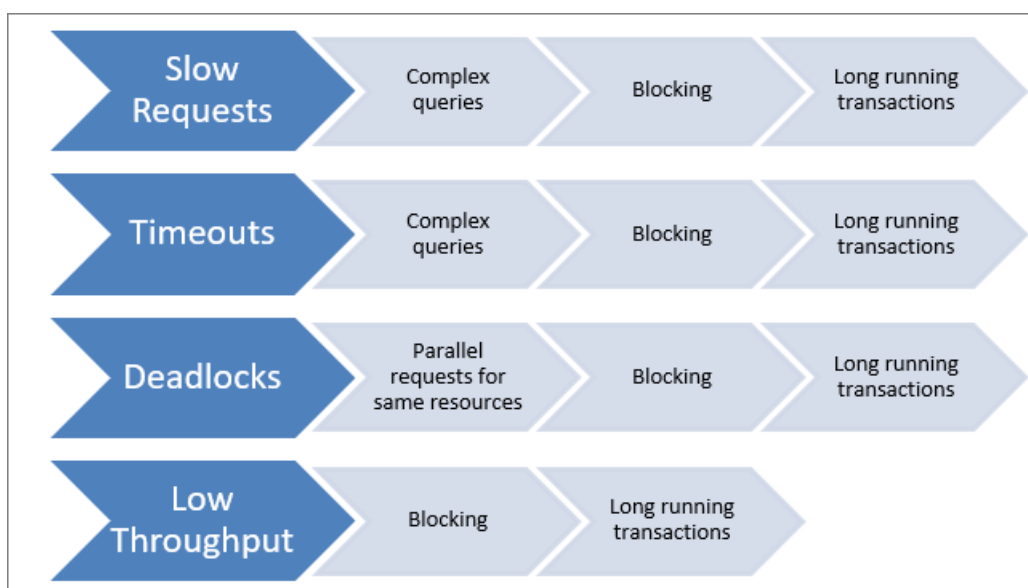
なぜ心配する必要があるのですか。Dataverse が対応してくれるのではないのですか...?

できる限り対応します... ただし、必要な場合は競合からシステムを保護するためにロックとトランザクションが使用されます。

マイクロソフトでは、データへのアクセス制御が重要な場合に、特定のシナリオについて選択して決定できるオプションも提供しています。ただし、間違った選択により、カスタム コードに意図しない結果が生じる可能性があります。通常これらの問題では、長い応答時間がユーザー エクスペリエンスに影響します。そのため、特定のアクションの影響を把握することで一貫性が向上し、結果的にユーザー エクスペリエンスも向上します。

原因の詳細

一般的な現象が原因で、特定の要求の実行速度が強制的に遅くなり、これによりプラットフォームの制約がトリガーされます。次の図は、これらの現象の一般的な根本原因の一部により発生する典型的な現象を示します。



長い時間のかかるトランザクション、データベースのブロック、複雑なクエリの潜在的な影響として、互いにオーバーラップしてこれらの現象の原因となる悪影響を増幅させる可能性があります。たとえば、互いに完全に独立した時間のかかるクエリが連続してある場合、ユーザーの応答時間が遅くなることがあります。これらのクエリが一度だけ同じリソースにアクセスする必要がある場

合、応答時間は非常に遅くなりエラーになります。

プラットフォームの制約の設計

Dataverse プラットフォームには、1 つのアクションがシステム、そしてユーザーに有害な影響を与えることを防ぐために、意図的に課すことのできる数多くの制約があります。この動作によりが特定の要求をブロックでき、制約を外すことができるかどうかという疑問につながるが多いため、いらだたく感じられることがあります。これは、幅広く影響を及ぼしたい場合は適切なアプローチではありません。

プラットフォームが目的どおりに使用されて実装が最適化されている場合、これらの制約があるシナリオがあることきわめてまれです。制約を課す状態は、システム内のリソースを過度に縛り付ける動作を示すことがほとんどです。これは、同じユーザーまたは他のユーザーからの別の要求を処理できないことを意味します。そのため、ブロックしている要求の制約を緩めることはできても、実際にはその要求が使用しているリソースはより長い時間縛り付けられるため、他のユーザーにより大きな影響が及ぶこととなります。

このような制約のため、ユーザーの要求への迅速な応答が重要度な場合に、トランザクション タイプのマルチユーザー アプリケーションをサポートするように Dataverse プラットフォームが設計されています。長い時間のかかる処理やバッチ処理のためのプラットフォームではありません。Dataverse は一連の短い要求を管理できますが、Dataverse はバッチ処理を扱うようには設計されていません。同様に、大量の反復処理を実行する活動がある場合、Dataverse はこのような反復処理を扱うように設計されていません。

次のシナリオでは、個別のサービスを使用して長い時間のかかるプロセスをホストし、短いトランザクション要求は Dataverse で管理します。たとえば、ほかの場所でフローを使用したり、Microsoft SQL Server Integration Services (SSIS) をホストして、個々の作成要求や更新要求を Dataverse で管理するパターンのほうが、Dataverse で作成される何千ものレコードをプラグインを使用してループするよりもずっと最適です。

存在しているプラットフォームの制約をアプリケーションの設計で考慮できるためには、それらの制約を認識して理解することが大切です。また、このようなエラーが発生した場合は、発生した原因を理解して発生しないように変更することができます。

■	■
■	<ul style="list-style-type: none">• プラグインは 2 分後にタイムアウトします• プラグインでは長時間の操作を実行しないでください。劣悪なユーザー エクスペリエンスからプラットフォーム、サンドボックス サービス、そして最終的にはユーザーを保護します
SQL ■	<ul style="list-style-type: none">• SQL Server へのリクエストは 30 秒でタイムアウトになります• 長い時間のかかる要求から保護します• 特定の組織内およびそのプライベート データベース内での保護を提供します。• また、プロセッサやメモリなどの共有リソースの過度な使用に対する保護をデータベース サーバー レベルで提供します。
■	<ul style="list-style-type: none">• 公平に使用するポリシーで実行• 特定の厳しい制限はないが、組織内でリソースを均等化• 要求が少ない場合、組織は使用可能なキャパシティを最大限に利用できます。要求が多い場合、リソースとスループットは共有されます。
■	<ul style="list-style-type: none">• IIS の Web サーバー接続プールからデータベースへの接続プールの上限は 100 接続というプラットフォームの既定の接続があります。Dataverse はこの値を変更しません• これに遭遇した場合、これはシステム内のエラーを示します。多くの接続がブロックされている理由を調べてください。• 複数の Web サーバーでは、各サーバーが一般的な < 10ms のデータベースに対して使用する同時接続は 100 接続です。これは各 Web サーバーの > 10k データベース要求のスループットを示します。これは必須ではなく、それよりも前にほかの問題が発生します

「	「
ExecuteMultiple	<ul style="list-style-type: none">ExecuteMultiple メッセージは、外部ソースから Dataverse にまとめて送信される操作を支援するように設計されていますこれらのリクエストの大規模なグループの処理は、ユーザーによるより多くの応答の重要なリクエストを犠牲にして、Dataverse の重要なリソースを拘束する可能性があります。
■	<ul style="list-style-type: none">すべてのユーザーに一貫した可用性とパフォーマンスを確実に提供するために、API の使用方法にいくつかの制限を適用します。これらの制限は、クライアント アプリケーションがサーバー リソースに対して異常な要求を行っていることを検出するように設計されています。詳細: サービス保護の API 制限

次の手順

プラットフォームの制約を適用する方法を理解するには、データベース トランザクションを理解する必要があります。詳細: [スケーラブル カスタマイズ設計: データベース トランザクション](#)

NOTE

ドキュメントの言語設定についてお聞かせください。 [簡単な調査を行います](#)。(この調査は英語です)

この調査には約 7 分かかります。個人データは収集されません ([プライバシー ステートメント](#))。

スケーラブル カスタマイズ設計: データベース トランザクション

2022/07/21 •

NOTE

これは、スケーラブル カスタマイズ設計に関する 2 つめのトピックです。最初から始めるには、[Microsoft Dataverse におけるスケーラブル カスタマイズ設計](#) を参照してください。

ここで直面する多くの課題の背後にある最も基本的な概念の 1 つは、データベース トランザクションの概念です。Dataverse では、データベースはシステムへのほとんどすべての要求の中心であり、データの一貫性が主として適用される場所です。

- Dataverse のデータ操作は、内部でも、コードのカスタマイズの一部でも、完全に独立して機能することはありません。
- すべての Dataverse のデータ操作は、データ レベル、またはプロセッサ、メモリ、I/O の使用状況などのインフラストラクチャ レベルのいずれかで、同じデータベース リソースと対話します。
- 不整合な変更から保護するために、各要求は表示または変更されるリソースをロックします。
- これらのロックはトランザクション内で行われ、トランザクションが確定または中止されるまで解除されません。

トランザクションおよびロックの認識

この領域で問題が発生する一般的な理由は、カスタマイズがトランザクションにどのように影響を与えるかという認識不足です。

これがどのように行われるかの詳細はこのトピックの範囲外ですが、考慮すべき最も単純な要素は、Dataverse がそのデータベース内のデータと対話することです。SQL Server は、そのデータに対してトランザクションが行う適切なロックを決定します。次に例を示します。

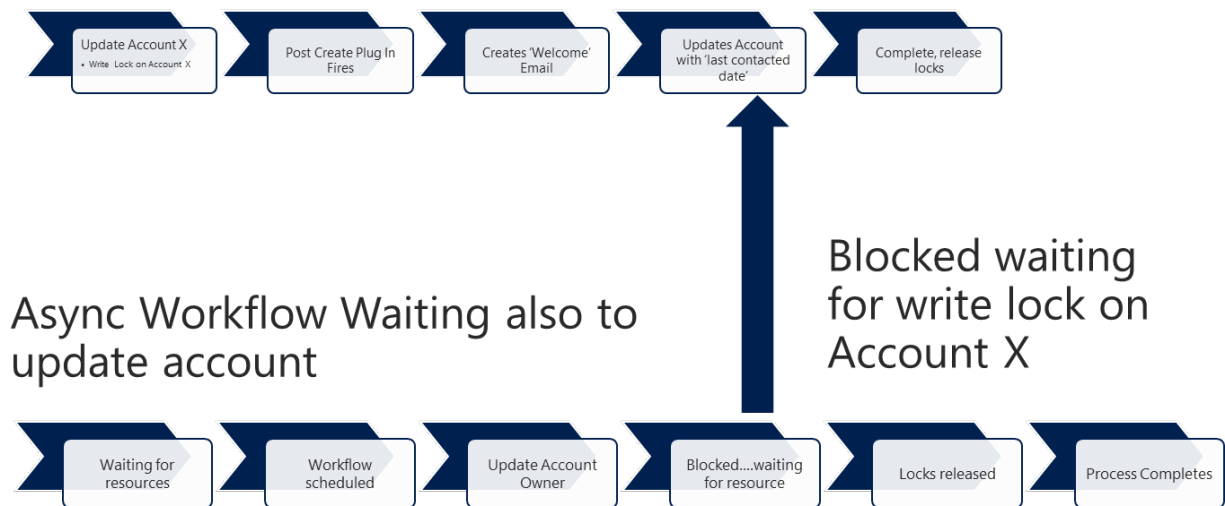
- 特定のレコードを取得するとき、SQL Server はそのレコードに対して読み取りロックをかけます。
- 一定の範囲のレコードを取得するとき、一部のシナリオでは、その範囲のレコードまたはテーブル全体に対して読み取りロックをかけることが可能です。
- レコードを作成するとき、そのレコードに対して書き込みロックを生成します。
- レコードを更新するとき、そのレコードに対して書き込みロックをかけます。
- ロックがテーブルまたはレコードに対してかけられるとき、対応するすべてのインデックス レコードに対してもかけられます。

ただし、これらのロックの範囲と期間に影響を与えることは可能です。また、SQL Server に対して、特定のシナリオにはロックは不要であると指示することも可能です。

ここでは、SQL Server データベースのロックと、同じデータにアクセスしようとする別の要求の影響について考えてみます。次の例では、アカウント作成によって一連のプロセスが設定されました。これらには、レコードが生成されたらすぐにトリガーされるプラグインを含むものや、生成時に開始される関連の非同期ワークフローに含まれるものがあります。

この例は、他の活動も同じアカウント レコードと対話している中、アカウントの更新プロセスに複雑な後処理がある場合の結果を示しています。アカウントの更新処理がまだ実行中に非同期ワークフローが処理される場合、このワークフローは、まだロックされている同じアカウント レコードを変更するための更新ロックの取得を待つブロックされる可能性があります。

User Updates an Account



トランザクションは、プラットフォームへの特定の要求の有効期間内のみに行われることに注意してください。ロックは、ユーザーセッションレベルで、または情報がユーザーインターフェイスに表示されている間は、実行されません。プラットフォームが要求を完了した直後に、データベース接続や関連のトランザクション、そしてかけられたすべてのロックを解除します。

ブロック

前の例のようなブロックは、それ自体不便ですが、Dataverse が数百の同時アクションを処理できるプラットフォームであると考え、より深刻な結果を招く可能性があります。個別のアカウントレコードをロックすることにはある程度限定的な意味合いしかないものの、リソースがより激しく争われるとどうなるでしょうか？

たとえば、各アカウントに一意な参照番号が与えられたとき、使用されている参照番号を追跡している単一のリソースが、すべてのアカウント生成プロセスによってブロックされてしまう可能性があります。[自動付番の例](#)で説明されているように、多数のアカウントが同時に生成される場合、重複している要求はすべてその自動付番のリソースにアクセスする必要があり、アクションを完了するまでそのリソースをブロックします。各アカウントの作成プロセスに時間がかかり、同時要求が多くなればなるほど、ブロックも多く実行されます。

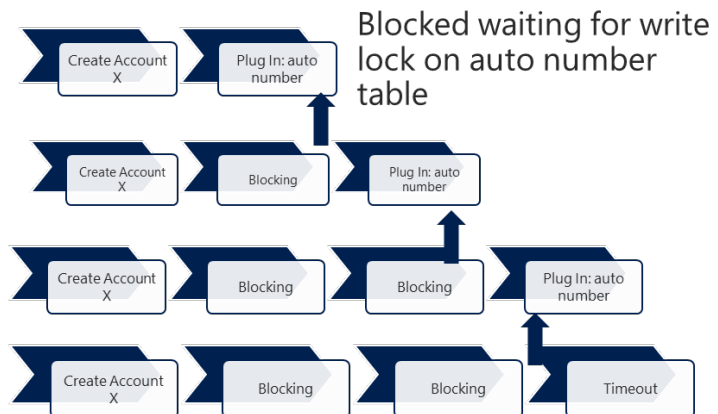
自動付番のリソースのロックを取得するための最初の要求は簡単に取得できますが、2 番目の要求は、次の一意の参照番号が何かを確認する前に最初の要求が完了するのを待つ必要があります。3 番目の要求は、1 番目と 2 番目の要求がどちらも完了するのを待つ必要があります。要求が多ければ多いほど、ブロックも長く発生します。十分な数の要求があり各要求に十分な時間がかかると、個々の要求が正しく完了したとしても、後の要求がタイムアウトするまでプッシュされる可能性があります。

User Creates an Account

All create in parallel,
because no conflicting
resource



Each blocks until auto number
lock released. If too many....can
cause a timeout



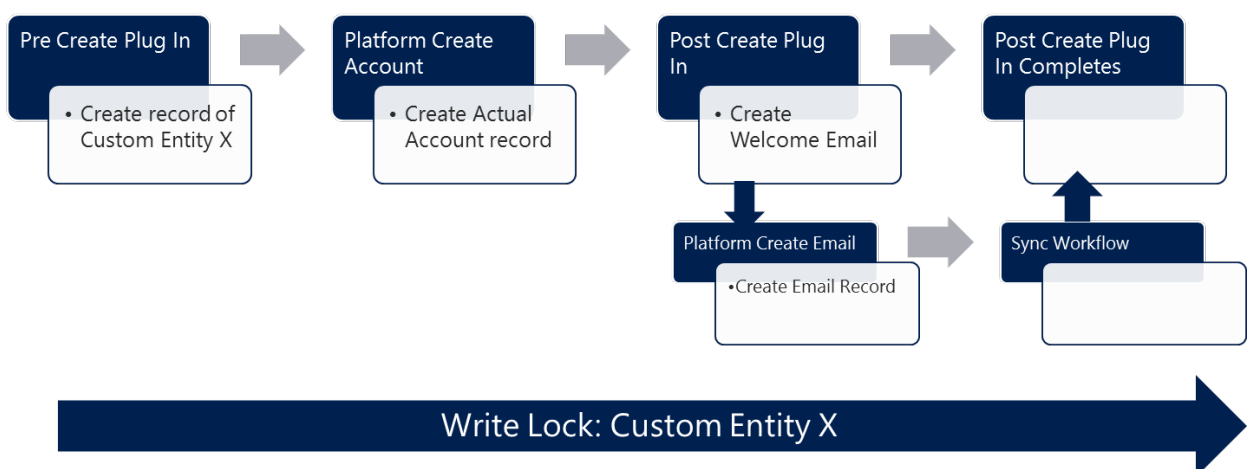
ロック解除

ロックが解除されずにトランザクションの完了まで保持される主な理由は 2 つあります。

- トランザクションが後でデータ アイテムを更新するために別要求を行う場合に備えて、データベース サーバーは一貫性のためにロックを保持します。
- またデータベース サーバーは、後で発行されるエラーまたは中止のコマンドがトランザクション全体をロールバックする可能性があるということを考慮する必要があるため、トランザクションの有効期間全体にわたりロックを保持し一貫性を確保する必要があります。

プロセスが特定のデータとの対話を完了した可能性があっても、トランザクション全体が完了し確定するまでロックが保持されることを認識しておく必要があります。トランザクションが長くなるほどロックは長く保持され、他のスレッドがそのデータと対話することを防ぎます。後に示されるように、これには同じトランザクション内で機能する関連のカスタマイズも含まれ、同期ワークフローなどのトランザクションの有効期間を大幅に延長できます。

以下の例では、アカウントの作成前プラグインのユーザー定義エンティティへの書き込みロックは、アカウント作成に関連付けられたすべてのロジックが完了するまでロックされています。



断続的エラー: タイミング

断続的な動作は、同時活動のブロックの明らかな症状です。先に失敗したときと全く同じアクションを繰り返して後で成功する場合は、同時に発生している他の何かによってエラーや遅延が引き起こされた可能性が非常に高いです。

問題のデバッグには、問題がある機能を最低限に抑えることを含める場合が多いため、これを理解することが重要です。ただし、断続的にのみ問題が発生する場合、失敗するアクションが他の活動と競合しているのはシステム内のどこかを確認しなければならない場合があり、さらに潜在的な競合ポイントを確認する必要があります。個々のプロセスを最適化することで競合を軽減できます。ただし、処理時間が短くなるほど、活動が他のプロセスと競合する可能性は低くなります。

トランザクション コントロール

多くの場合トランザクションの使用方法は単にプラットフォームに管理を委ねることができますが、必要なロジックが複雑で、結果を達成するにはトランザクションに対する理解と影響が必要だというシナリオがあります。Dataverse には、トランザクションの使用方法に異なる影響を与える、さまざまなカスタマイズ方法が用意されています。

各タイプのカスタマイズがプラットフォーム トランザクションにどのように関与するかを理解することで、Dataverse で複雑なシナリオを効果的にモデル化し、その動作を予測できます。

前述のように、トランザクションはプラットフォームへの要求の有効期間中にのみ保持され、プラットフォームのステップが完了した後に維持されるものではありません。これにより、トランザクションが外部クライアントによって長時間保持され、他のプラットフォームの活動がブロックされることを回避します。

プラットフォームのジョブは、プラットフォームのトランザクション パイプラインを通して一貫性を維持し、必要に応じてカスタマイズを同じトランザクションに参加させることです。

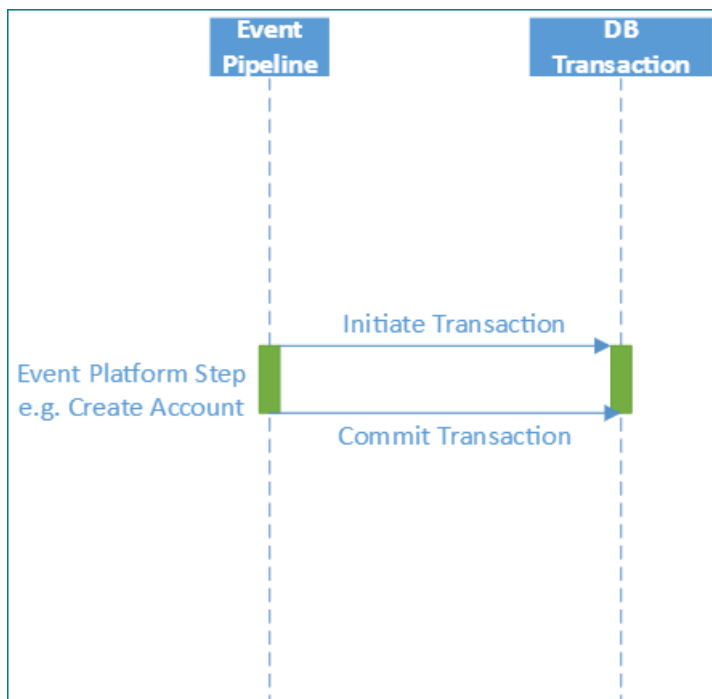
モデル駆動型アプリがトランザクションを使用する方法

カスタマイズがプラットフォームとどのように対話するかを理解する前に、モデル駆動型のアプリがプラットフォームへの要求をどのように使用し、それがどのようにトランザクションの使用へ影響を与えるかを理解しておくと有用です。

「	「
フォーム (Retrieve)	<ul style="list-style-type: none">表示されるレコードの読み取りロックを取得します。他の使用への影響は低いです。
作成	<ul style="list-style-type: none">プラットフォームを使用して作成要求を実行します新規レコードが他には何もブロックしないため、他の使用への影響は低いです完了までテーブル全体に対するロック クエリをブロックする可能性があります。カスタマイズで関連のアクションがトリガーされることが多く、それが影響を与える可能性があります。
Update	<ul style="list-style-type: none">プラットフォームを使用して更新要求を実行します。競合がある可能性が高いです。更新ロックは、そのレコードを更新したり読み取りしたりするものをブロックします。また、そのテーブルに広い読み取りロックをかけているものもすべてブロックします。他の活動をトリガーする場合があります。
表示 (RetrieveMultiple)	<ul style="list-style-type: none">これは他の多くの活動をブロックすると考えられます。不十分なクエリの最適化が DB リソースの使用状況に影響を与え、タイムアウトになる可能性はあります。

イベント パイプライン: プラットフォーム ステップ

イベントパイプラインが開始されると、プラットフォームのステップを含めるために SQL トランザクションが作成されます。これにより、プラットフォームによって実行されるすべてのデータベース活動が一貫して処理されます。トランザクションはイベントパイプラインの最初に作成され、処理が成功したかどうかによって、処理が完了したときに確定または中止されます。



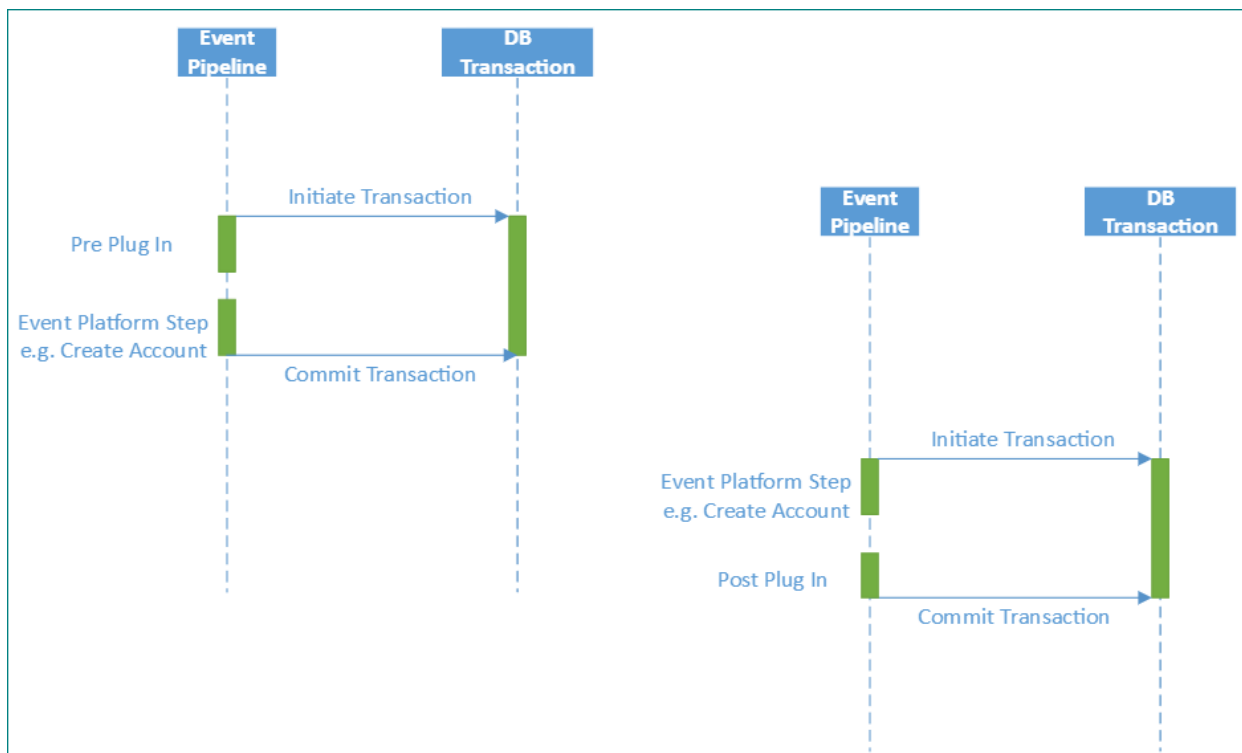
カスタマイズ要求

カスタマイズ内でプラットフォームにより開始されたトランザクションに参加することもできます。カスタマイズの種類ごとに異なる方法でトランザクションに参加します。以下のセクションでそれぞれについて順に説明します。

- [同期プラグイン \(事前または事後操作: トランザクション コンテキスト内\)](#)
- [同期プラグイン \(事前および事後操作: トランザクション コンテキスト内\)](#)
- [同期プラグイン \(PreValidation: トランザクション コンテキスト外\)](#)
- [同期プラグイン \(PreValidation: トランザクション コンテキスト内\)](#)
- [非同期プラグイン](#)
- [プラグイン トランザクションの使用概要](#)
- [同期ワークフロー](#)
- [非同期ワークフロー](#)
- [ユーザー定義ワークフロー活動](#)
- [ユーザー定義アクション](#)
- [Web サービス要求](#)

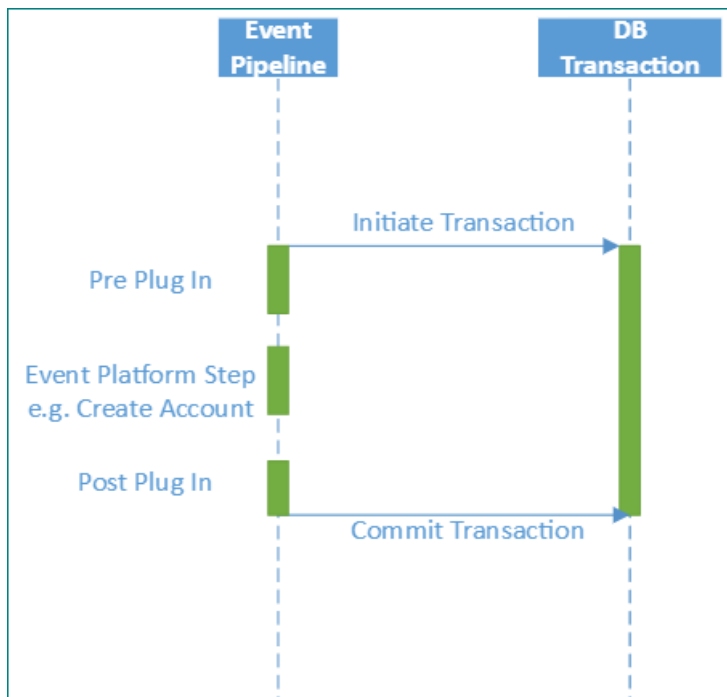
同期プラグイン (事前または事後操作: トランザクション コンテキスト内)

プラグインがイベントに登録されると、トランザクション内にある **PreOperation** または **PostOperation** ステージに対して登録できます。すべてのプラグインからのメッセージ要求がトランザクションの内部で実行されます。これはトランザクションの有効期間を意味し、実行されたすべてのロックが延長されます。



同期プラグイン (事前および事後操作: トランザクション コンテキスト内)

プラグインは PreOperation および PostOperation ステージの両方に対して登録できます。この場合は、トランザクションが PreOperation プラグインの開始から PostOperation プラグインの完了まで延長するため、トランザクションはさらに延長できます。

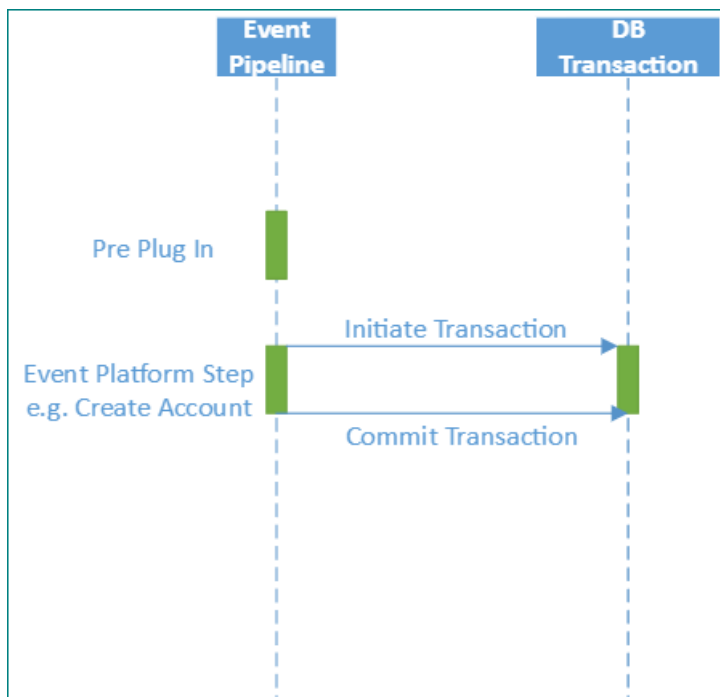


同期プラグイン (PreValidation: トランザクション コンテキスト外)

プラグインは、PreValidation ステージに登録されることで、プラットフォーム トランザクションの外部で機能するように登録することも可能です。

NOTE

これは独自のトランザクションを作成しません。そのため、プラグイン内の各メッセージ要求はデータベース内で単独で処理されます。



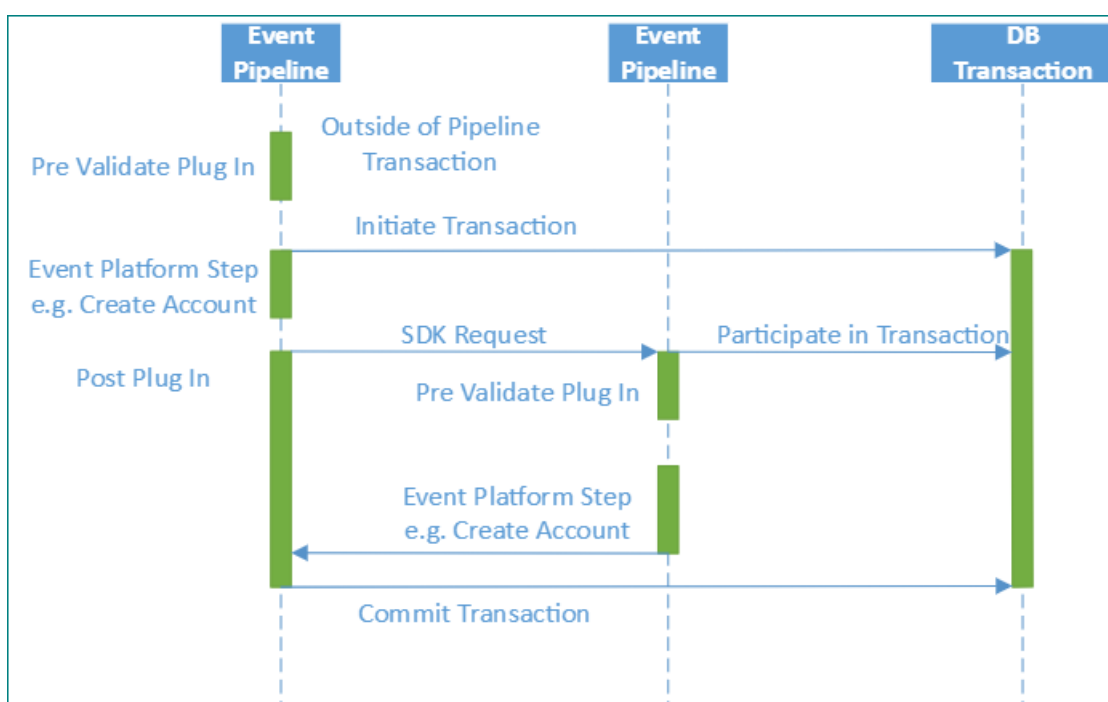
このシナリオは、**PreValidation** がイベント パイプラインの最初のステージとして呼び出されたときにのみ適用されます。プラグインが **PreValidation** ステージで登録されても、次のセクションで示すように、トランザクションに参加することは可能です。**PreValidation** プラグインがトランザクションに参加していないとは限りませんが、その場合は実行コンテキストから確認できます。

同期プラグイン (**PreValidation**: トランザクション コンテキスト内)

関連のシナリオは、**PreValidation** プラグインが登録されたものの、関連するイベント パイプラインが既存のトランザクション内からのメッセージ要求によってトリガーされたときに発生します。

次の図で示すように、アカウントを作成すると、初回作成の実行時に **PreValidation** プラグインがトランザクションの外部で実行される可能性があります。プラグイン後の一部として、2 番目のイベント パイプラインが親パイプライン内から開始されるために関連の子アカウントを作成するというメッセージ要求が行われた場合、それは同じトランザクションに参加します。

その場合、**PreValidation** プラグインはトランザクションがすでに存在することを確認し、**PreValidation** ステージに登録されている場合でもそのトランザクションに参加します。



前述のとおり、プラグインは **IsInTransaction** プロパティの実行コンテキストを確認します。それは、このプラグインがトランザクシ

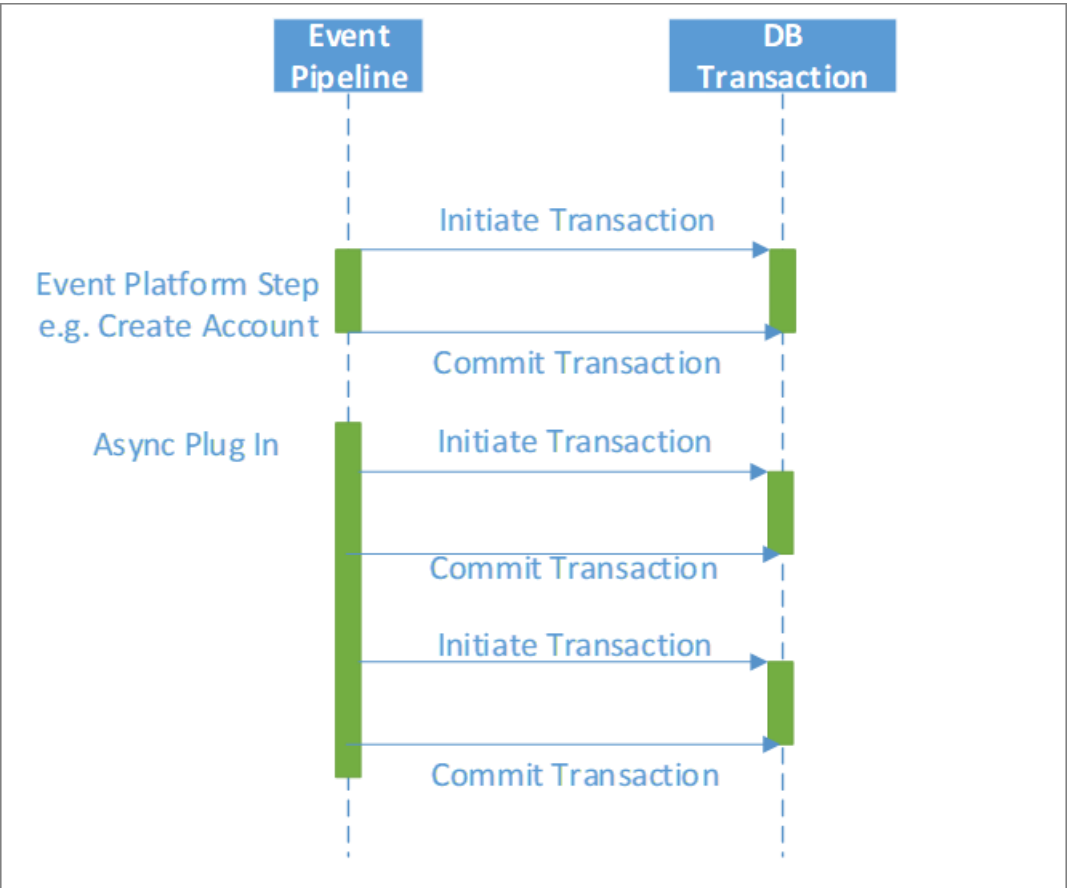
ン内で実行されているかどうかを示します。

非同期プラグイン

プラグインは、非同期で機能するように登録することもできます。この場合、プラグインは、プラットフォーム トランザクションの外部でも機能します。

NOTE

プラグインは、自身のトランザクションは作成しません。プラグイン内の各メッセージ要求は独立して処理されます。



プラグイン トランザクションの使用概要

要約:

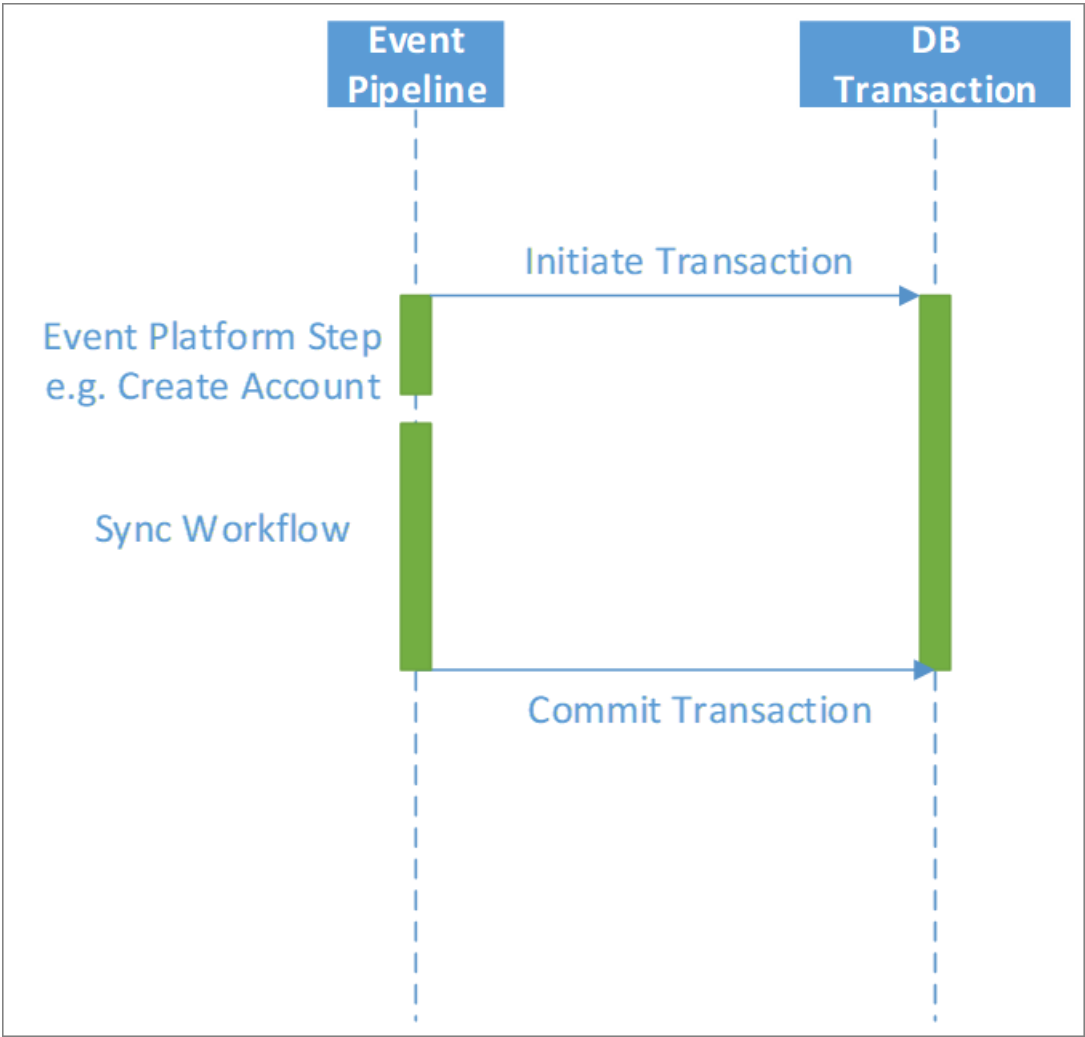
- 同期プラグインは通常トランザクションに参加します。
- 非同期プラグインは、プラットフォームのトランザクションには一切参加しません。各要求は独立して実行されます。
- PreValidation プラグインは、トランザクションがすでに存在する場合、トランザクションを作成せず参加します。

イベント前	■	トランザクションは作成されません。トランザクションに参加しません。各要求は、データベースへの独立したトランザクションを使用します	既存のトランザクションに参加します
イベント前	■	既存のトランザクションに参加します	既存のトランザクションに参加します
イベント後	PostOperation	既存のトランザクションに参加します	既存のトランザクションに参加します

同期	該当なし	トランザクションは作成されません。トランザクションに参加しません。各要求は、データベースへの独立したトランザクションを使用します	該当なし
----	------	--	------

同期ワークフロー

トランザクションの観点から見たとき、同期ワークフローは事前または事後操作のプラグインとして機能します。したがって、それらはプラットフォーム パイプライン トランザクション内で機能し、トランザクション全体の長さと同じ影響を与える可能性があります。



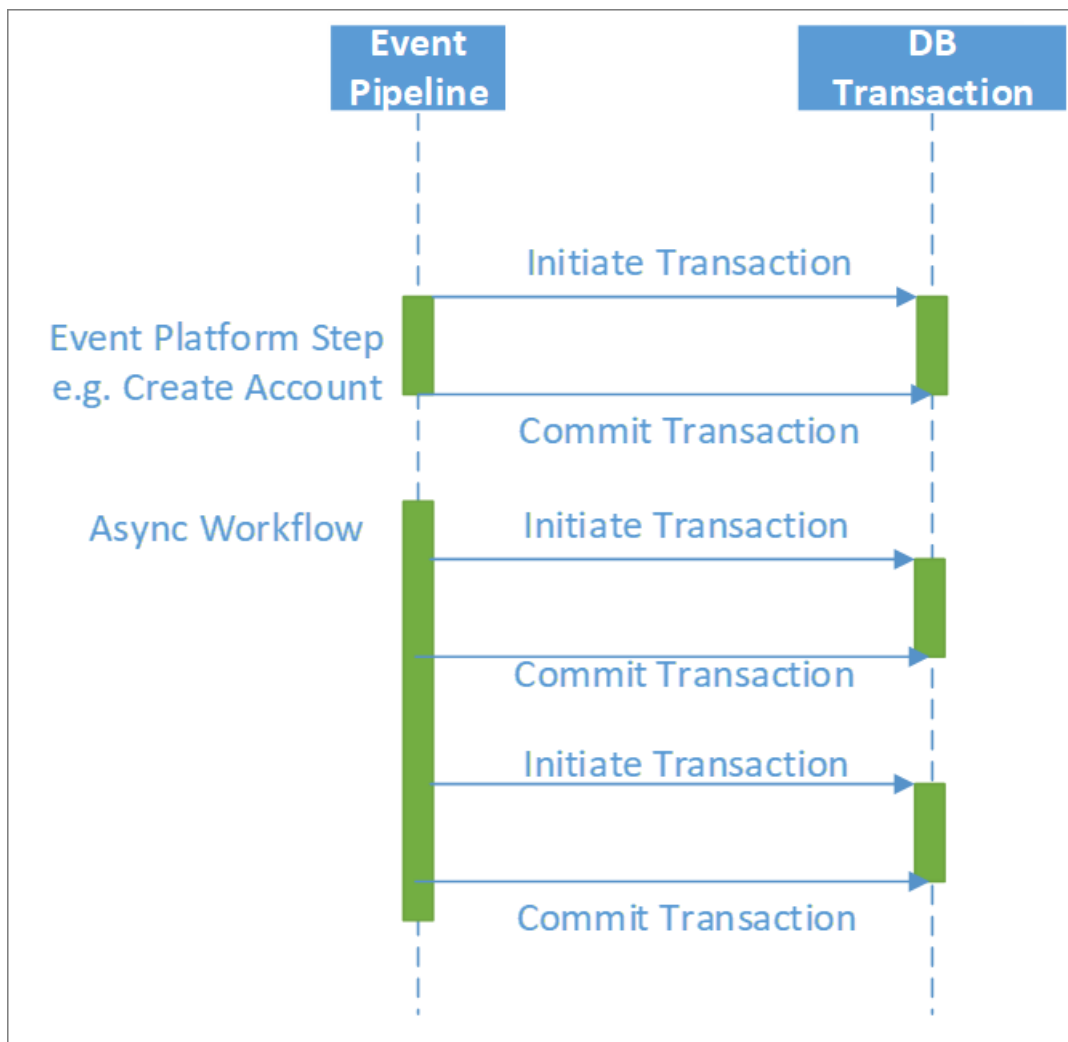
非同期ワークフロー

非同期ワークフローがプラットフォーム トランザクションの外でトリガーされます。

NOTE

ワークフローは独自のトランザクションも生成しないため、ワークフロー内の各メッセージ要求は独立して処理されます。

次の図は、プラットフォーム トランザクションの外で処理される非同期ワークフローと、独立したトランザクションの開始における各ステップを示します。

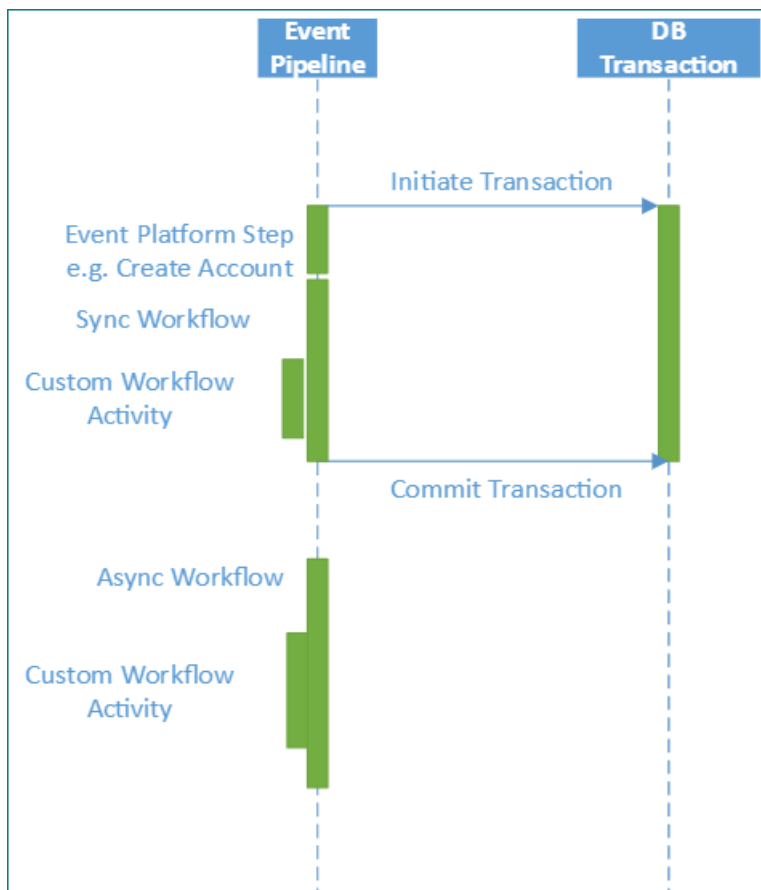


ユーザー定義ワークフロー活動

ユーザー定義ワークフロー活動は、親ワークフローのコンテキスト内で機能します。

- 同期ワークフロー: トランザクション内で機能
- 非同期ワークフロー: トランザクション外で機能

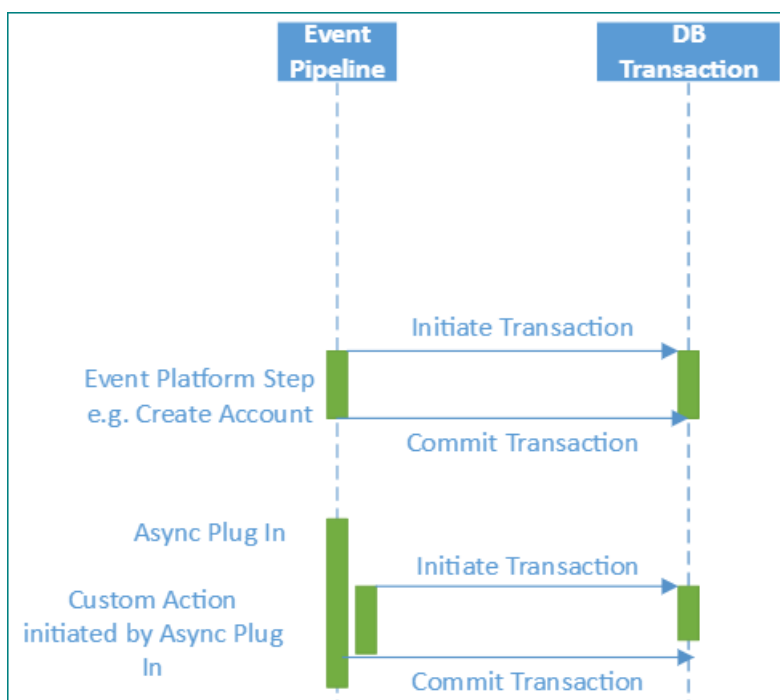
次の図は、まず同期ワークフロー内で、次に非同期ワークフロー内で機能するユーザー定義活動を示します。



カスタム アクション

ユーザー定義アクションは、独自のトランザクションを作成できます。これは主要な機能です。ユーザー定義アクションは、「ロールバックを有効にする」に設定されているかどうかに応じて、プラットフォームのステップ外に別のトランザクションを作成できます。

- 「ロールバックを有効にする」が設定
 - トランザクション内で実行中のプラグインからのメッセージ要求によって呼び出され、「ロールバックを有効にする」が設定されている場合、ユーザー定義アクションは既存のトランザクション内で機能します。
 - それ以外の場合、ユーザー定義アクションは新しいトランザクションを作成しその中で実行します。
- 「ロールバックを有効にする」が未設定
 - ユーザー定義アクションは、トランザクションの内部で機能しません。



Web サービス要求

要求が Web サービスを介して外部に作られると、前に説明したように、パイプラインが作成されパイプライン内で処理されるトランザクションが発生しますが、応答が返されたらトランザクションは維持されません。次の要求までどのくらいかかるかがわからないため、プラットフォームは他の活動をブロックするリソースのロックを許可しません。

同じ実行コンテキストを使用してプラグイン内で複数の要求が作られる場合、トランザクション参照を維持し、同期プラグインでは各要求が同じトランザクション内で行われるようにするのは、共通の実行コンテキストです。要求間で実行コンテキストを維持する能力はプラグインの外部では使用できないため、トランザクションは外部で作られた別々の要求間では維持されません。

1 つの Web サービス要求の一部として複数のアクションを Dataverse プラットフォームに渡すことができる 2 つの特別なメッセージがあります。

MESSAGE	⌘
<code>ExecuteMultiple</code>	これにより、複数の独立したアクションを同じ Web サービス要求内で渡すことができます。これらの各要求はプラットフォーム内で個別に実行されるため、要求間で保持されるトランザクション コンテキストはありません。
<code>ExecuteTransaction</code>	<p>これにより、同期プラグイン内で作られた複数のメッセージ要求と同様の方法で、同じデータベース トランザクション内で複数のアクションを処理できます。</p> <p>またこの能力は、複数のメッセージ要求と同様の意味を持ちます。つまり、アクションごとに大幅な時間がかかる場合は (高クエリを作成したり、関連の同期プラグインまたはワークフローの長いチェーンをトリガーしたりする場合)、より幅広いプラットフォームでブロックの問題が発生する可能性があります。</p>

プラグインの Web API (OData) 要求

プラグイン内の Web API (OData) 要求を、同じ組織でプラグインとして使用しないでください。常に [IOrganizationService](#) メソッドを使用します。これにより、トランザクション コンテキストが渡され、パイプライン トランザクションに操作を参加させることが可能になります。

次のステップ

データベース トランザクションに加えて、複数のデータ操作がシステムに与える影響を理解することが重要です。詳細: [スケーラブル カスタマイズ設計: 同時実行問題](#)

NOTE

ドキュメントの言語設定についてお聞かせください。 [簡単な調査を行います](#)。(この調査は英語です)

この調査には約 7 分かかります。個人データは収集されません ([プライバシー ステートメント](#))。

スケーラブル カスタマイズ設計:同時実行の問題

2022/07/21 •

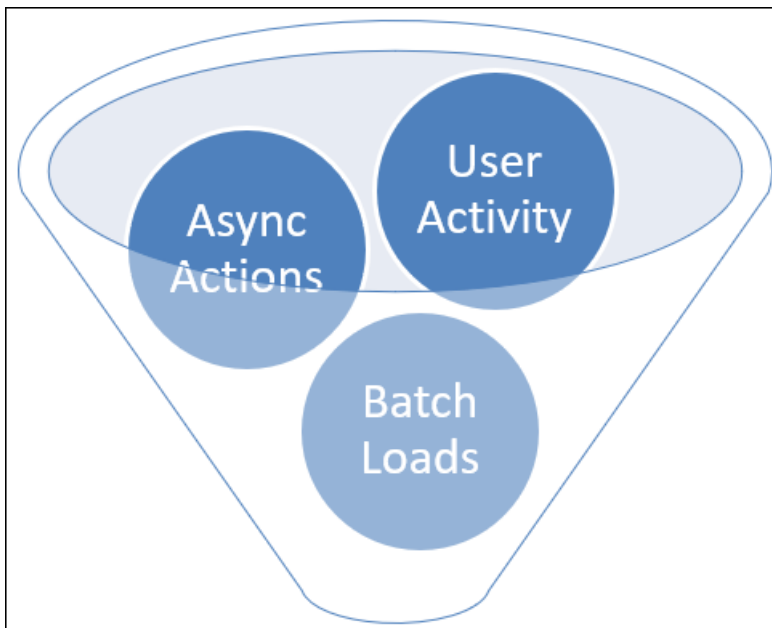
NOTE

これはスケーラブル カスタマイズ設計に関する 3 つめのトピックです。最初から始めるには、[Microsoft Dataverse におけるスケーラブル カスタマイズ設計](#) を参照してください。前のトピック [スケーラブル カスタマイズ 設計: データベース トランザクション](#) ではデータベース トランザクションが適用される方法、およびそれらがさまざまな種類のカスタマイズに与える影響について説明しました。

同時実行の要求がある場合は、ロックで衝突する可能性が高くなります。トランザクションに時間がかかるほどロックが長く保持されます。衝突の可能性がさらに高まり、エンドユーザーの全体的な影響は大きくなります。

アクティビティがアプリケーションで駆動する複数の方法についても認識する必要があり、システム内の他のアクションと競合する可能性があるそれぞれにロックをかけています。このような場合、ロックは同じデータに対して重複するアクションが発生したときに発生するデータの不整合を防止します。

設計を検討し問題があるか確認するために重要な領域は下記です:



- **ユーザー駆動の活動:** 直接ユーザーインターフェイスを介して。
- **非同期の操作:** 他のアクションの結果として後に発生する活動。この活動がいつ処理されるかは、開始アクションがトリガーされた時点では不明です。
- **バッチ 活動:** 一括削除ジョブやサーバー側の同期処理などの Dataverse、または他のシステムからの統合などの外部ソースからのいずれかで駆動されます。

並行処理される非同期操作

よくある誤解は、非同期のワークフローまたはプラグインはキューから連続的に処理され、それらの間に競合は生じないということです。Dataverse は、各非同期サービス インスタンス内でも、スループット向上のため異なるサーバーに分散した非同期サービス インスタンス間でも、同時に複数の非同期活動を並列処理するため、これは正確ではありません。各非同期サービスは、構成と負荷に基づいてサービスごとに約 20 のバッチで実行されるジョブを実際に取得します。

同じレコードの同じイベントから複数の非同期活動を開始した場合、それらは並行して処理される可能性があります。それらが同じレコードで開始したとき、共通のパターンは同じ親レコードに戻って更新されます; そのため競合する機会が高いです。

アカウントの作成などトリガーとなるイベントが発生すると、Dataverse の非同期ロジックは各プロセスまたは行われる操作ごとに **非同期処理 (システム ジョブ) エンティティ** にエントリを作成する場合があります。非同期サービスはこのテーブルを監視し、待機中の要求をバッチとして取得して処理します。ワークフローは同時にトリガーされるため、同じバッチに取得され同時に処理される可能性が非常に高いです。

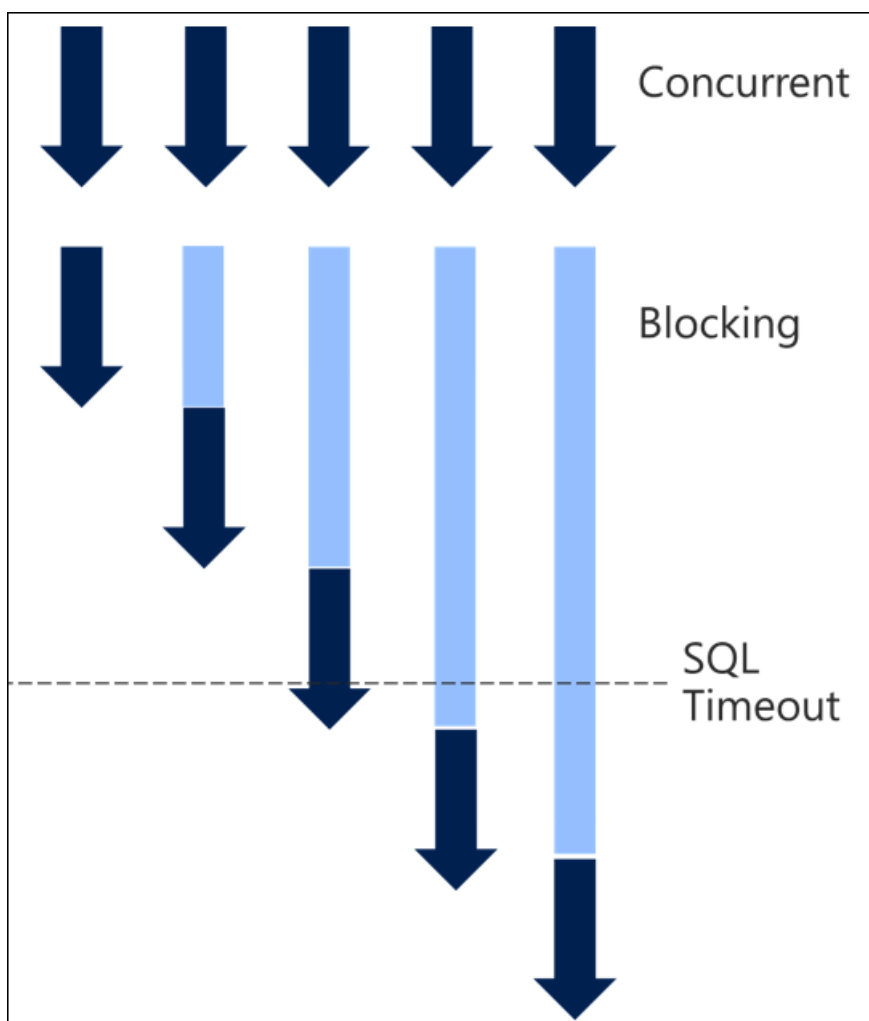
トランザクションの理解が重要な理由

スケーラブルなカスタマイズを設計するときに **自動付番の例** はデータベース トランザクションと同時実行の問題を考慮する方法を示すシナリオを提供します。

シリアル化とタイムアウト

高度なシリアル化とは通常、ブロックをタイムアウトと貧弱なスループットに変えるものです。多数の同時実行要求がある場合にそれらがシリアル化されて処理に時間がかかると、各要求の順番待ちが長くなりタイムアウトやエラーが発生してしまいます。

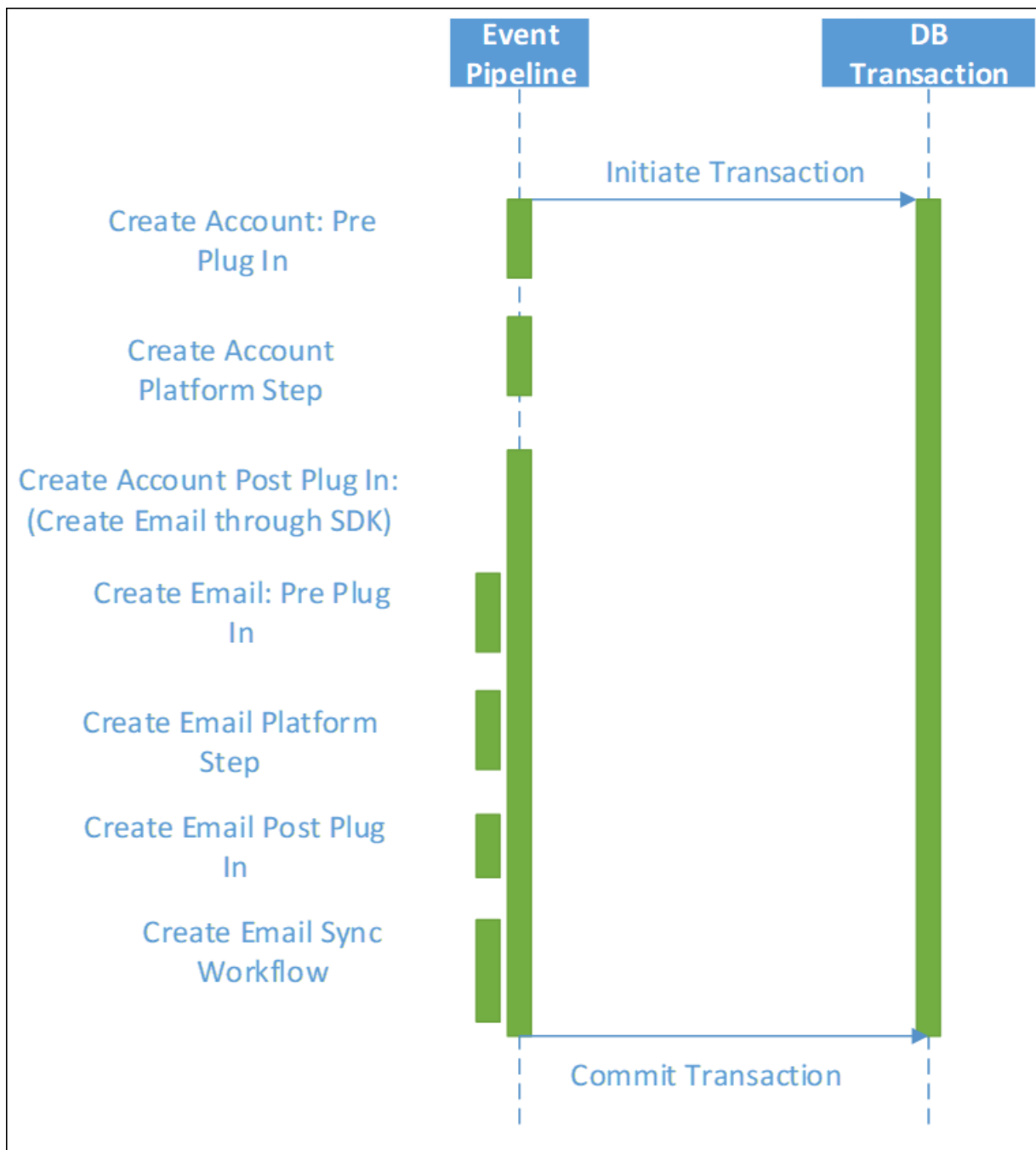
プラグインのタイムアウトは開始されたときから始まります。SQL タイムアウトはデータベース要求で計算されるため、クエリブロックが長時間待機するとタイムアウトになる可能性があります。



一連の操作

直接トリガーされた活動で特定のクエリを理解するとともに、関連する一連のイベントがどこで発生するのかを検討することも必要です。

プラグインまたは同期ワークフローのステップとして作成された各メッセージ要求は、直接操作をトリガーするだけでなく、他の同期プラグインやワークフローを起動させる可能性もあります。これらの同期活動それぞれは同じトランザクションで発生し、そのトランザクションと保持されたすべてのロックの寿命を、認識よりはるかに長く延長する可能性があります。



全体的な影響は最初の認識よりはるかに大きい可能性があります。これは複数人で実装を構築した場合や、または時間をかけて拡張した場合に、意図せず頻繁に発生する可能性があります。

プラットフォームの制約に遭遇

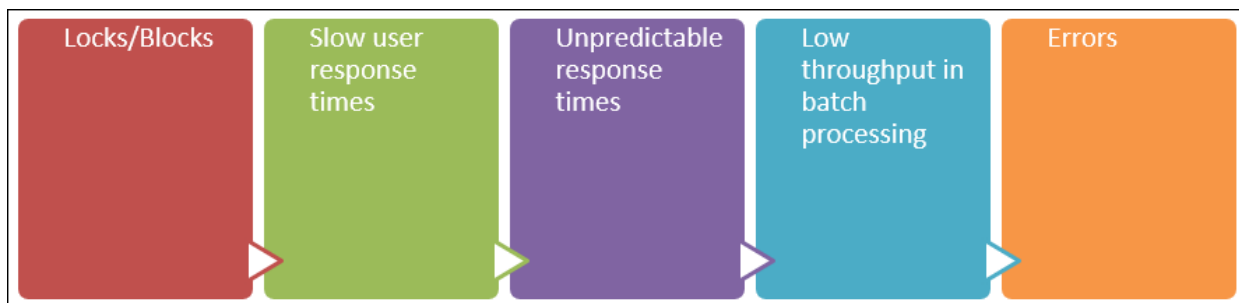
これがプラットフォーム制約が生じる可能性がある場所です。そして実際にこの種の振る舞いからまさに広大なシステムを保護するために制約が存在します。

このレベルの処理遅延が発生するたびに、システムの他の領域と他のユーザーに対して意図しない結果が生じます。したがって、この種の活動がシステム パフォーマンスを妨げるのを防ぐことが重要です。

そのエラーを回避する安易な方法はプラットフォーム制約の緩和ですが、これは全体的なスケーラビリティとシステムのパフォーマンスに対するより根本的な影響に対処するものではありません。そもそもこれは制約をトリガーする動作を修正および防止することで対処する必要があります。

使用の影響

使用にも影響することが多いのは、この動作の連鎖的な一連の影響です。



最初の問題はロックであり、したがってシステム内のブロックです。これはユーザー応答時間の遅延につながり、多くの場合システムの特定の領域で、予測不能で信頼できないユーザーの応答時間に増幅されます。

極端なケースや通常の負荷より重い場合、これはバックグラウンドのバッチ処理でのスループット低下として現れる可能性があります。結局それはすべてシステムで発生するエラーに拡大する可能性があります。

SQL タイムアウトエラーを調査するときは、ユーザーが貧弱で予測不能な応答時間も報告しており、関連する問題としてこれらの間で接続が確立されていないのが一般的です。

次のステップ

パフォーマンスの問題を最小限に抑えるために適用（または回避）できる設計パターンの理解。詳細: [スケーラブル カスタマイズ設計: トランザクション設計パターン](#)

NOTE

ドキュメントの言語設定についてお聞かせください。 [簡単な調査を行います](#)。(この調査は英語です)

この調査には約 7 分かかります。個人データは収集されません ([プライバシー ステートメント](#))。

スケーラブル カスタマイズ設計: 自動付番の例

2022/07/21 •

NOTE

この例では、スケーラブル カスタマイズ設計に関する一連のトピックをサポートします。最初から始めるには、[Microsoft Dataverse におけるスケーラブル カスタマイズ設計](#) を参照してください。

Dataverse でトランザクションが処理される方法について一般的な誤解を示すシナリオは、自動付番スキームを実装することです。

このシナリオにおける通常の要件:

- 特定のパターンに従って一意の番号を生成します。
- 関連するレコードの種類を生成するために多数の同時実行要求を許可します; 例えば一意な参照を必要とするアカウント。
- 一意な番号の連番を使用できます。
- 番号の生成に一貫性があり、スケーラブルで、ロード中にエラーが発生しないことを確認してください。重複した番号が生成されないことも確認する必要があります。
- 関連するレコードの作成で番号を生成します。

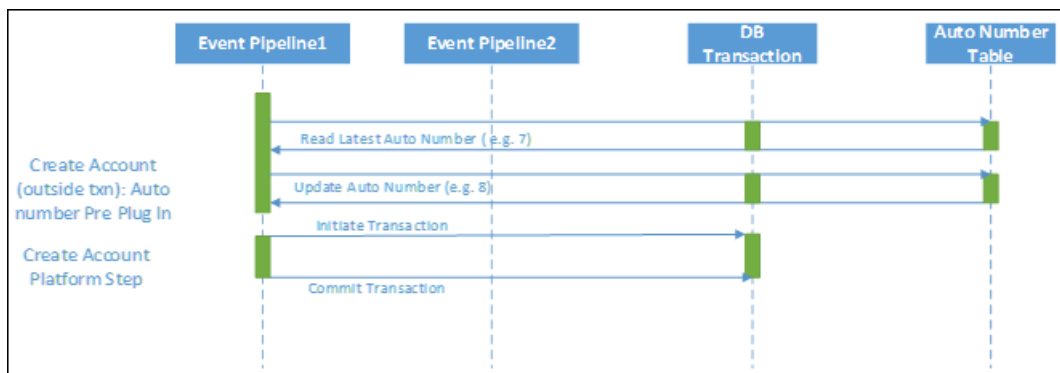
一般的な方法には次の種類が含まれます:

- 自動付番インデックスのデータストアで最後に使用された番号を保存します; 例えばデータの種類ごとの行とユーザー定義エンティティ。
- 最後に使用された番号を取得し、その番号を増やします。
- 新しく作成されたレコードに対して新しい番号を記録します。
- 新しい番号を自動付番インデックス ストアで最後に使用された番号として保存します。

次のセクションでは Dataverse で取られるさまざまな方法を説明し、その影響を強調して、トランザクションの利用方法を理解する重要性和利点の両方を示します。

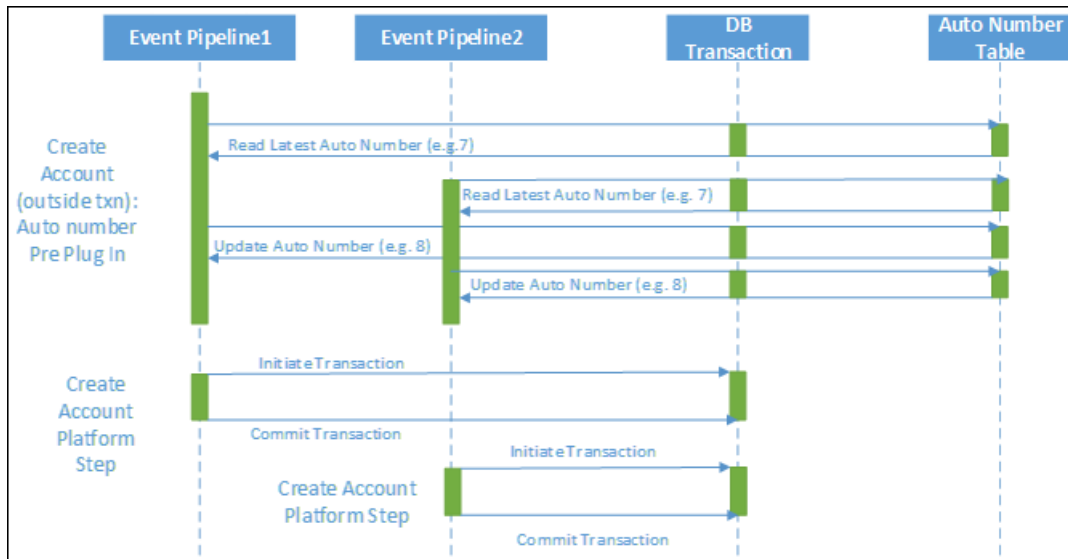
方法1: トランザクションの外へ

最も簡単な方法は、一般的に要求されるリソースを使用するとブロックされる可能性が生じることを認識することです。これはスケーラビリティに影響を与えるため、自動付番を生成するときにプラットフォーム トランザクションを回避べきと決定する可能性があります。検証前プラグインのパイプライン トランザクションの外側で自動付番を生成する方法を考えてみましょう。



これを個別に実行すると正常に動作します。ただし、実際には同時実行エラーから保護しません。下記の図が示すように、2つの要求が両方とも並行して最新の数値を要求し、両方とも値を増加させ更新すると、数値が重複することになります。取

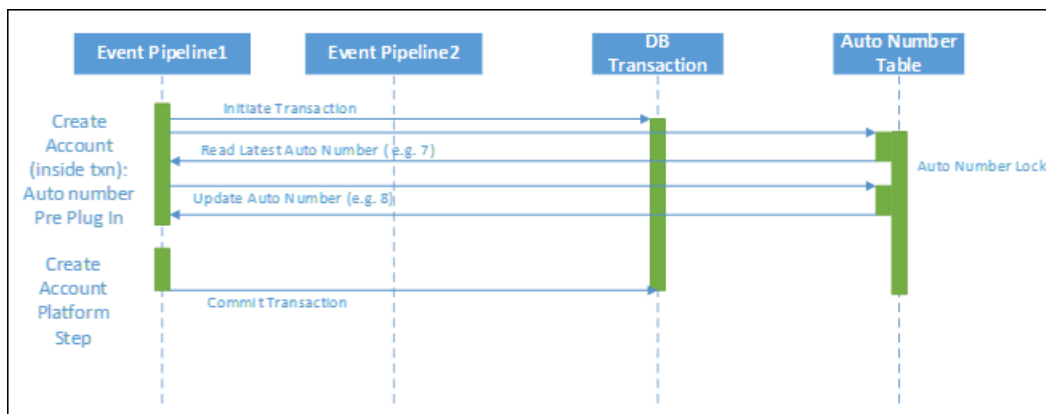
得した数に対してロックが保持されないため、競合状態が発生し両方のスレッドが同じ値になってしまう可能性があります。



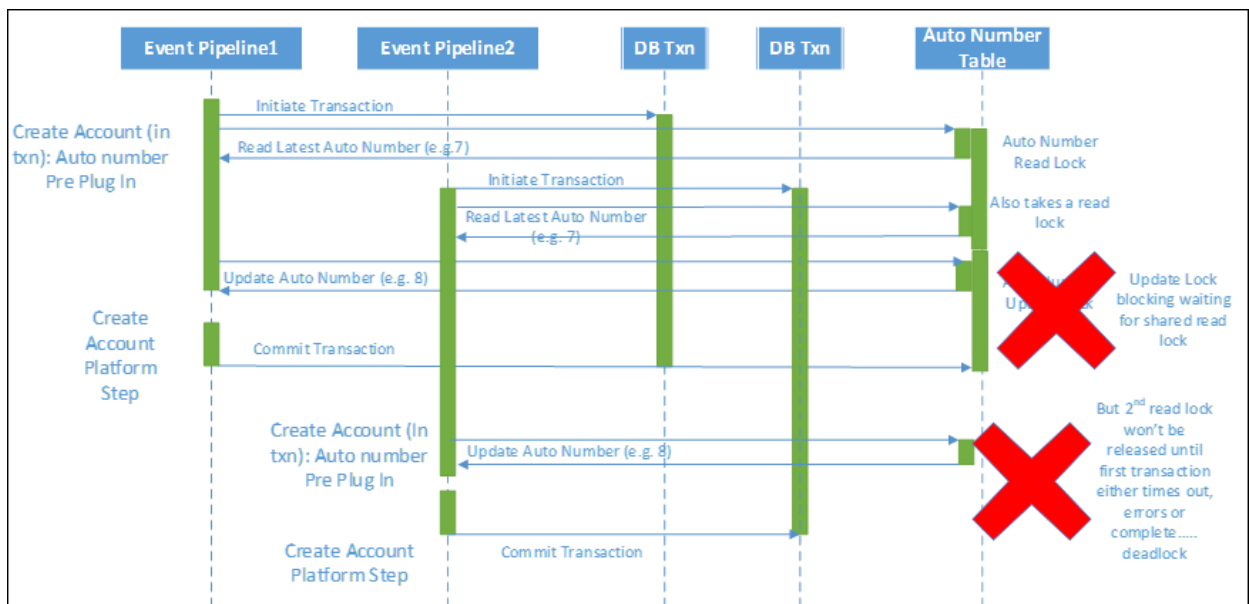
多くの場合、複数の要求が発生していても重複するウィンドウが限られているため、うまく機能する可能性があります、それは重複を防ぐための良い設計ではなく運に頼っています。

アプローチ 2: プラグイン トランザクション内で

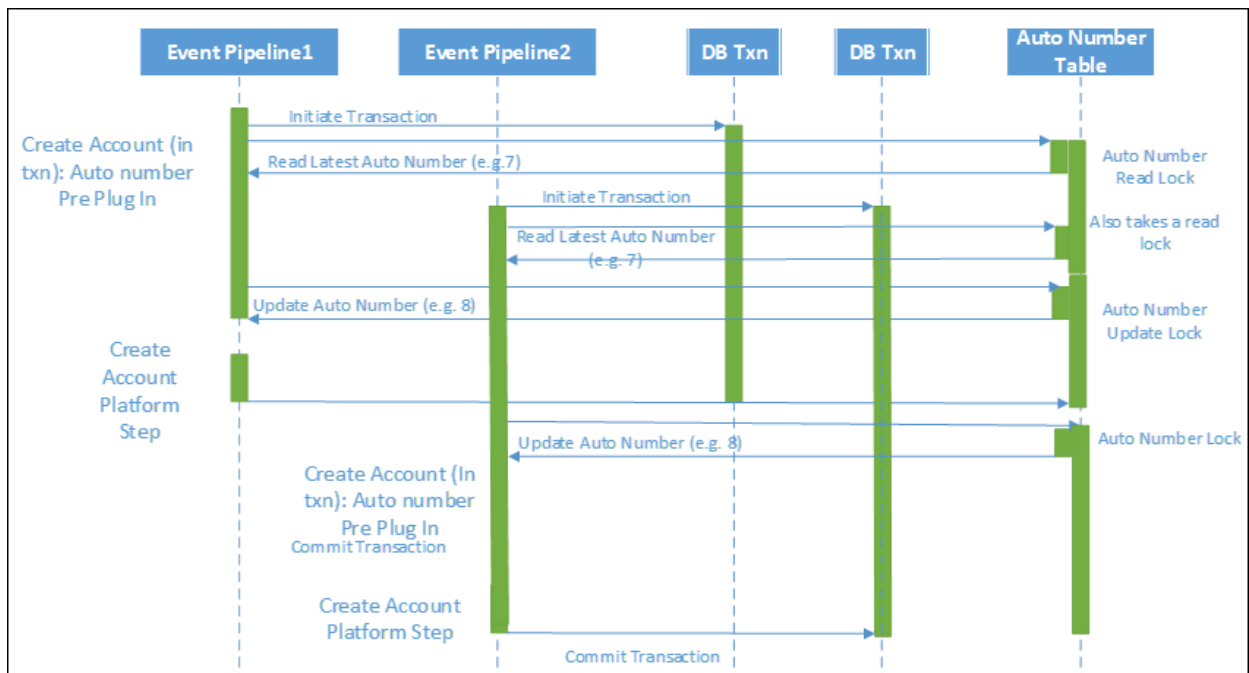
トランザクション (txn) に登録されているプラグインから自動付番をする場合、これは確かにうまくいきます。



重複した要求が同時に番号を生成しようとするのと同じ状況では、両方の要求に自動付番テーブルに共有読み取りロックを付与することが可能です。残念ながら、アプリケーションがこれを排他ロックにアップグレードしようとする時点では、別の共有読み取りロックが妨げるためこれは不可能です。



クエリが生成されている方法によって正確な振る舞いが異なる可能性があります、それらの条件に頼り一意性が不可欠であるという結果が信頼できないのは理想的ではありません。これにより障害が発生しないとしても、分離モードが正しくない場合は共有読み取り機能によって重複した番号が生成される可能性があります。次の図が示すように、両方のレコードが自動付番で同じ値 8 になってしまいます。

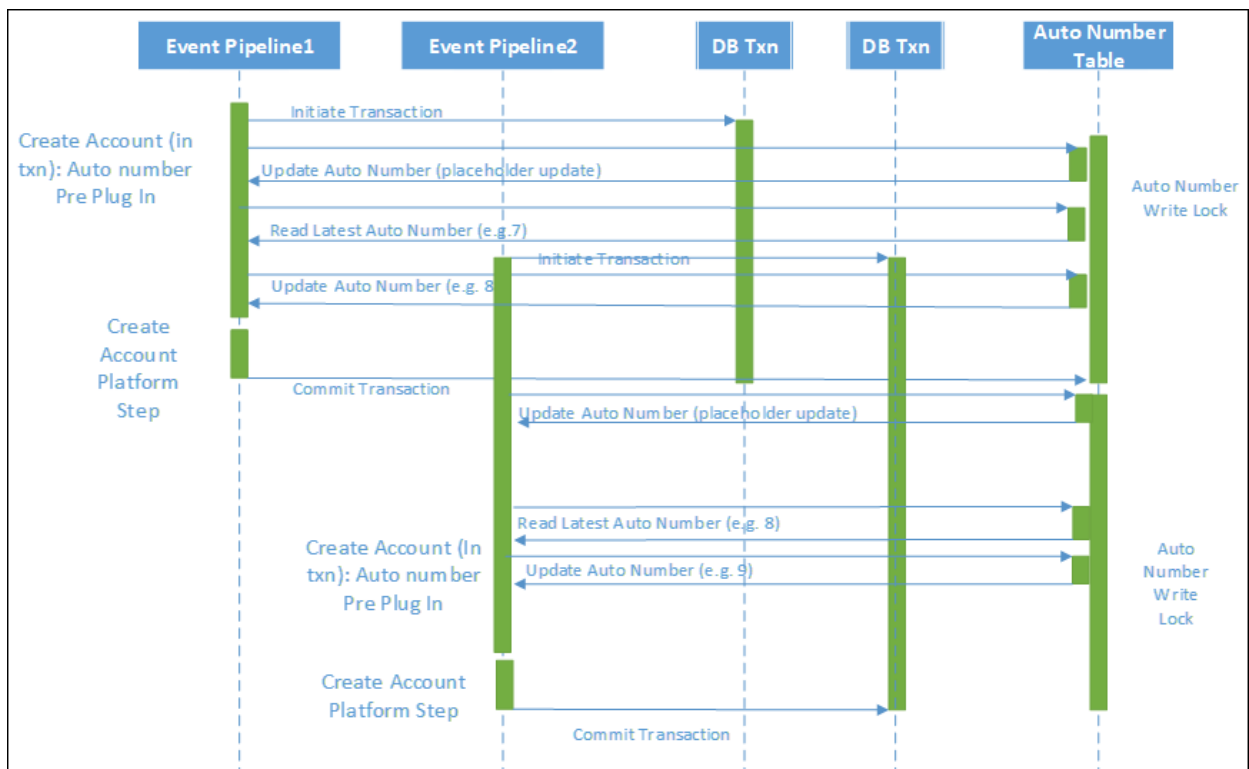


アプローチ 3: プラグイン トランザクションの Pre-lock

トランザクションが機能する方法を理解することは、これを行う安全な方法を生み出すことにつながります。

このアプローチでは、純粋に一貫性を維持する目的で使用するフィールド(たとえば UpdateInProgress)に対して、トランザクションの開始時から自動付番レコードでプレースホルダの更新が実行されます。更新が開始されようとしていることを示す更新を書き込むことによってこれを行います。次に、このプロセスは自動付番テーブルのその行に排他的書き込みロックを要求および実施して、他のプロセスが自動付番アプローチを開始するのをブロックします。

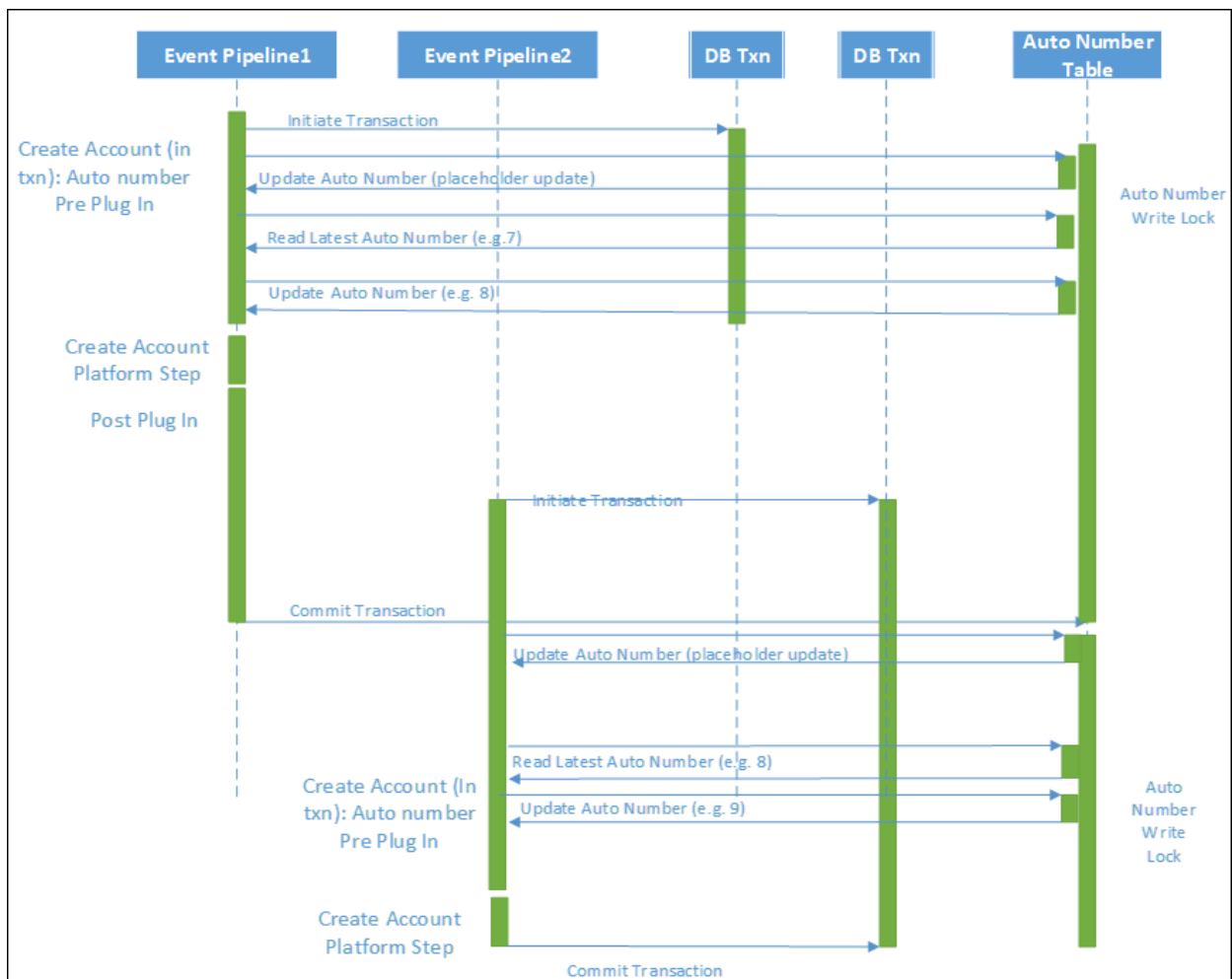
これにより他のプロセスが干渉することなく、更新された自動付番を安全に増分して書き戻すことができます。



手順が同じプラットフォーム トランザクションで発生するため、自動付番の更新だけでなくアカウント作成要求もシリアル化する影響があります。アカウントの作成がクイック アクションならば、それは完全に良いアプローチである可能性があり、それはアカウントの作成と自動付番が一貫して実行されることを保証します; もし一方が失敗した場合は、両方とも失敗してロールバックします。

実際、トランザクションで他の処理が速い場合は、これがカスタマイズで自動付番を実装する最も一貫性があり効率的なアプローチです。

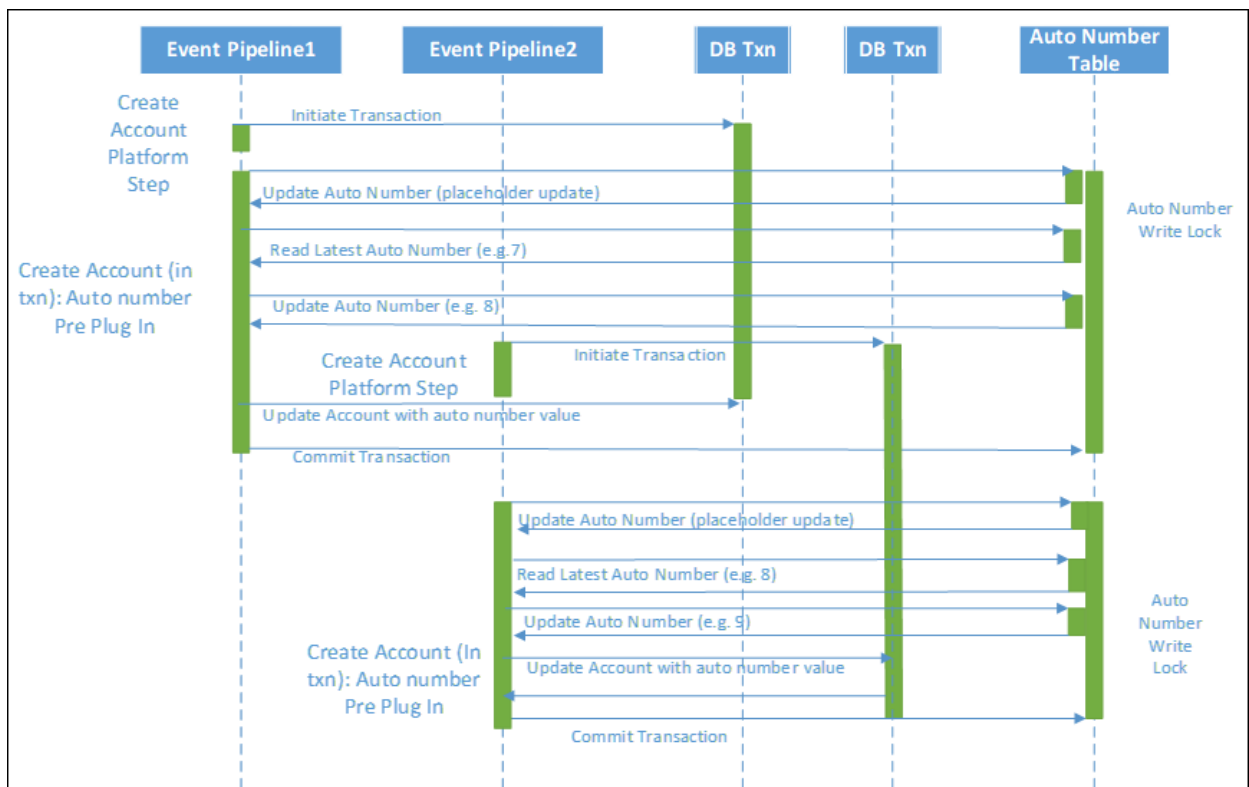
ただし、完了までの時間を伸ばす他の同期プラグインやワークフローを導入する場合、自動付番のプロセスがそれ自体をブロックするだけでなく他の活動の完了待機もブロックするため、シリアル化は現実にはスケーラビリティの課題になります。



通常、自動付番の生成はイベント前プラグインで行われます。作成ステップの入力パラメータに番号を含め、アカウントに対して生成された自動付番を記録する後処理で 2 回目の更新を回避します。

スケーラビリティの影響を念頭に、アカウント作成プロセスに他の複雑な処理がある場合、代替策は自動付番の生成を作成後プロセスに移動することで、これも一貫した更新プロセスを保証します。その利点は、ロックがプロセスの終わりに向かってのみ行われるため、自動付番レコードのロックが保持されるトランザクションの時間を短縮することです。自動付番のテーブルが最も競合の激しいリソースである場合、それにアクセスするすべてのプロセスに対してこのアプローチが取られれば全体の競合量が減少します。

ここでのトレードオフは、自動付番記録の待機をブロックしている全体の時間を減らす間に、アカウントに追加の更新を実行する必要があることです。



NOTE

ドキュメントの言語設定についてお聞かせください。 [簡単な調査を行います](#)。(この調査は英語です)

この調査には約 7 分かかります。個人データは収集されません ([プライバシー ステートメント](#))。

スケーラブル カスタマイズ設計: トランザクション デザイン パターン

2022/07/21 •

NOTE

これは、スケーラブル カスタマイズ設計に関する 4 つめのトピックです。最初から始めるには、[Microsoft Dataverse におけるスケーラブル カスタマイズ設計](#) を参照してください。

このセクションでは、回避または最小化する設計パターンとその影響について説明します。それぞれの設計パターンは、解決されるビジネス上の課題という観点から考慮する必要があり、調査するための選択肢として役立ちます。

ロックを避けないようにしてください。

ロックは SQL Server と Dataverse の非常に重要なコンポーネントであり、システムの正常な動作と一貫性を保つために不可欠です。このため、特にスケールで、デザインへの影響を理解することが重要です。

トランザクションの使用: No-lock のヒント

ビューで頻繁に使用される Dataverse プラットフォームの 1 つの機能は、クエリを no-lock のヒントで実行できることを指定する機能です。これにより、このクエリにロックが不要であることをデータベースに伝えます。

ビューは、ビューおよびそれ以降のアクションのトップに間に直接リンクがないため、この方法を使用します。そのユーザー、またはその間にいる他のユーザーによって、他にも多くのアクティビティが発生する可能性があります。ビューに表示されるデータのテーブル全体をロックして、ユーザーが移動するまで待機するのは実用的ではなく、利益もありません。

大規模なデータセットに対するクエリは、そのデータのいずれかと対話しようとしている他のユーザーに影響を与える可能性があるため、ロックが不要であると指定できるということは、システムのスケーラビリティに対して大きなメリットがあります。

SDK を介してプラットフォームのクエリを作成するときに、no-lock が使用できるように指定することは有益です。これは、このクエリがデータベース内で読み取りロックを取得する必要があることを認識していることを示しています。クエリに特に有効なのは、次の場合です。

- スコープのデータ全体があるとき
- 競合の激しいリソースでクエリが実行されたとき
- シリアル化されたは重要ではないとき

前述の自動付番のロックの例のように、後の操作が結果の変更依存しない場合は、No-lock を使用しないでください。

それが役に立つ可能性のあるシナリオの例としては、電子メールが既存のケースに関連しているかどうかを判断することが挙げられます。他のユーザーが新しいケースを作成するのをブロックして、電子メールが確実に作成されないようにすることは、一貫性のコントロールに対して有益だとは言えないでしょう。

代わりに、関連するケースにクエリを実行して既存のケースに電子メールを添付するか、他のケースを生成できるようにする一方で新しいケースを作成するといった、妥当な努力をするほうがより適切です。特に、これら 2 つのアクションの間のタイミングには固有のリンクがないため、電子メールが数秒早く受信してリンクが検出されなかった可能性があります。

ロックのヒントが特定のシナリオに有効であるかどうかは、通常、競合が発生する可能性と影響、および取得と後続の間にあるアクションの一貫性を保証しないことによるビジネス上の影響の判断に基づきます。ロックを回避してもビジネス上の影響が生じない場合は、no-lock を使用することが最適化の良い選択肢となります。ビジネスに潜在的な影響がある場合は、その影響について、ロックを回避することによるパフォーマンスと拡張性のメリットを比較検討することができます。

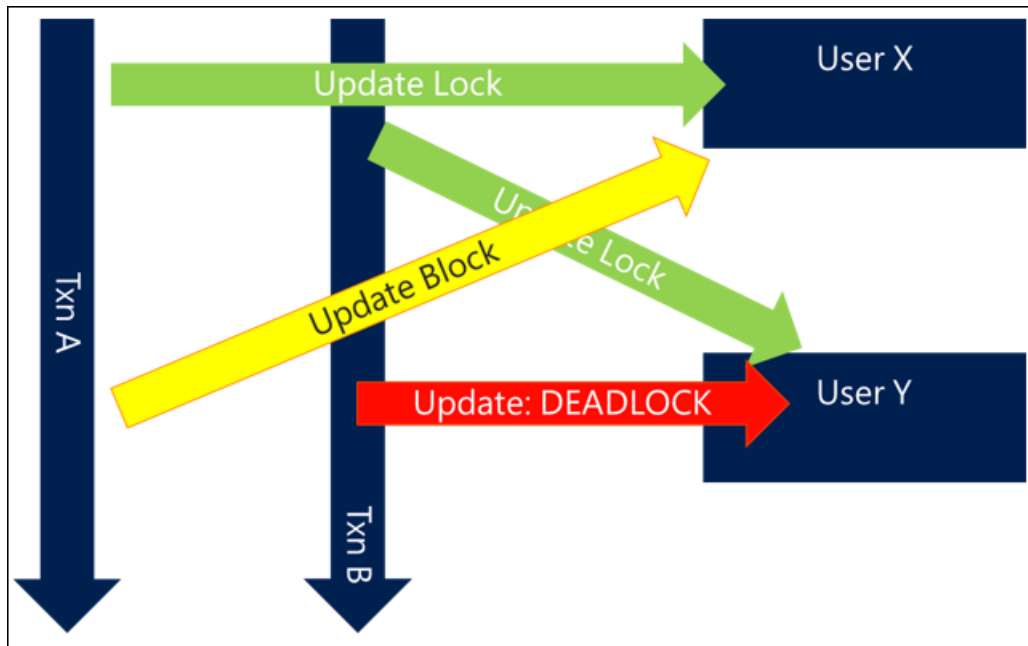
ロックの順番を考える

ブロックの影響を減らす、特にデッドロックを回避するのに役立つ可能性があるもう 1 つの方法は、実装内でのロックの順序付けに対する一貫性のあるアプローチです。

単純で一般的な例は、ユーザーのグループを更新または対話するときです。関連ユーザーを更新するリクエスト (チームへのメンバーの追加やアクティビティ内のすべての参加者の更新など) がある場合に、2 つの同時アクティビティが同じユーザーを更新しようとすると、次のような動作になる可能性があり、その結果、デッドロックが発生します。

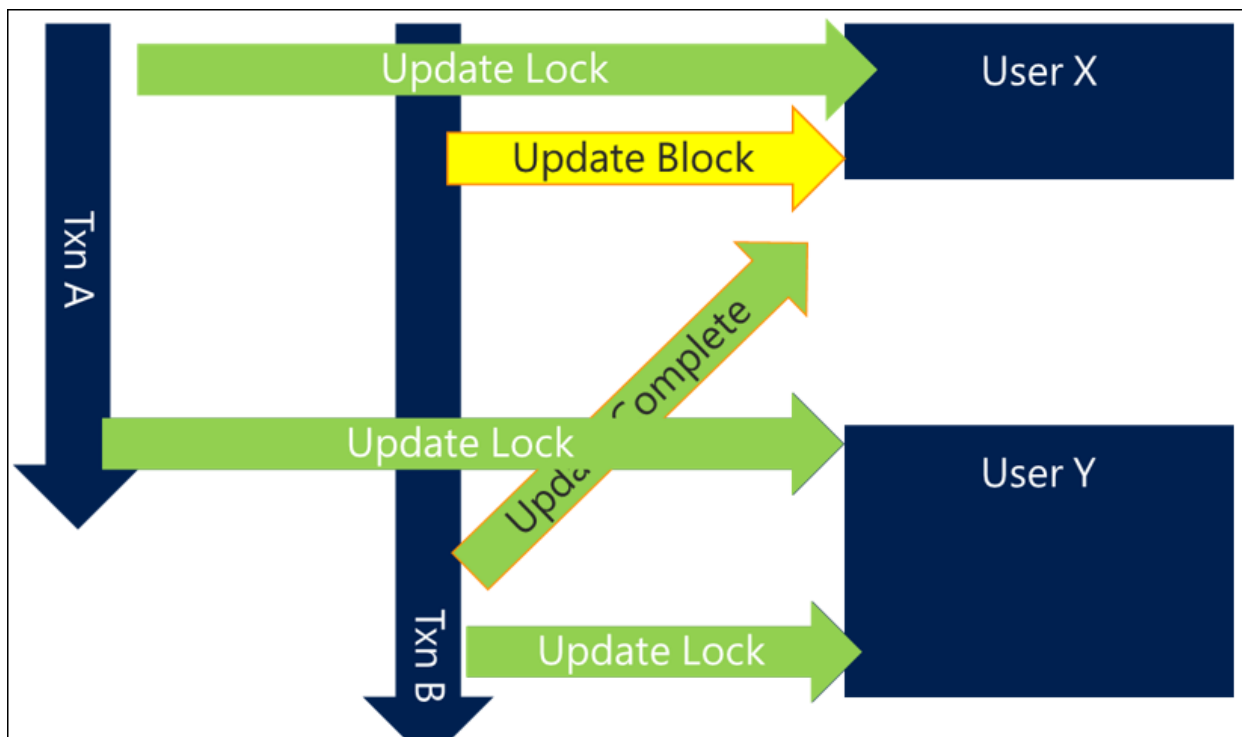
- トランザクション A はユーザー X を更新してからユーザー Y を更新しようとします
- トランザクション B はユーザー Y を更新してからユーザー X を更新しようとします

両方の要求が同時に開始されるため、トランザクション A はユーザー X をロックし、トランザクション B はユーザー Y をロックすることができますが、各ユーザーが他のユーザーをロックしようとするとすぐにブロックされ、その後デッドロックが発生します。



一貫した方法でアクセスするリソースを順番に並べるだけで、多くのデッドロックされる状況を防ぐことができます。順序付けメカニズムは、一貫性があり、できるだけ効率的に実行できる限り、多くの場合重要ではありません。たとえば、名前または GUID でユーザーを並べ替えることで、少なくともデッドロックを回避する一定レベルの一貫性を確保できます。

このアプローチを使用するシナリオでは、トランザクション A はユーザー X を取得しますが、トランザクション B もユーザー Y よりもユーザー X を取得しようとします。これは、トランザクション A が完了するまでトランザクション B がブロックされることを意味しますが、このシナリオではデッドロックが回避され、正常に完了します。



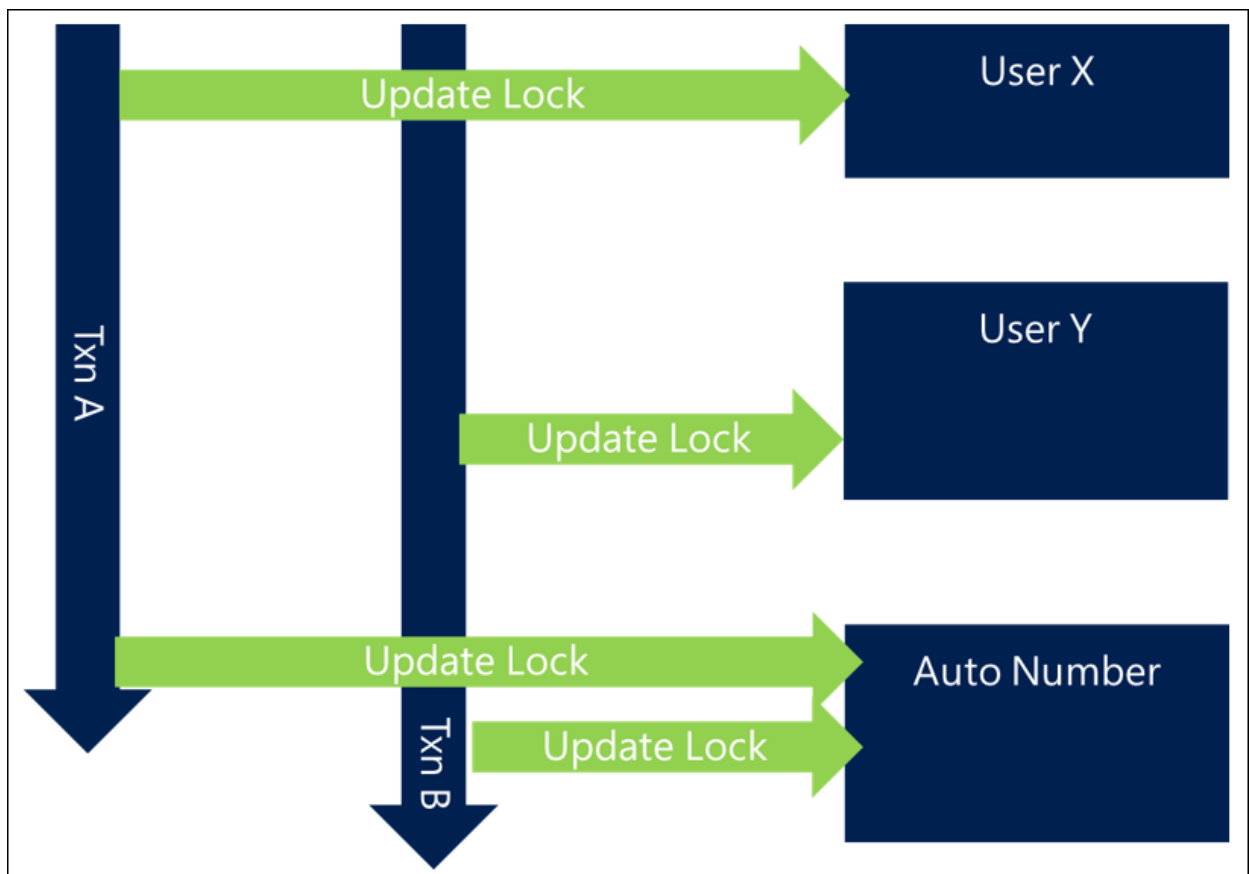
より複雑で効率的なシナリオでは、最初に最も一般的に参照されていないユーザーをロックし、最後により頻繁に参照されているユーザーをロックすることです。これにより、次の設計パターンにつながります。

最短期間、競合するロックを保持する

自動付番アプローチのような、ロックが必要な競合の激しいリソースの存在を回避する方法がないというシナリオがあります。その場合、ブロックの問題を完全に回避することはできませんが、最小限に抑えることはできます。

競合の激しいリソースがある場合は、プロセス内の機能的に論理的なポイントにそのリソースとの対話を含めないで、できるだけトランザクションの終わりで、そのトランザクションとの対話を行うことをお勧めします。

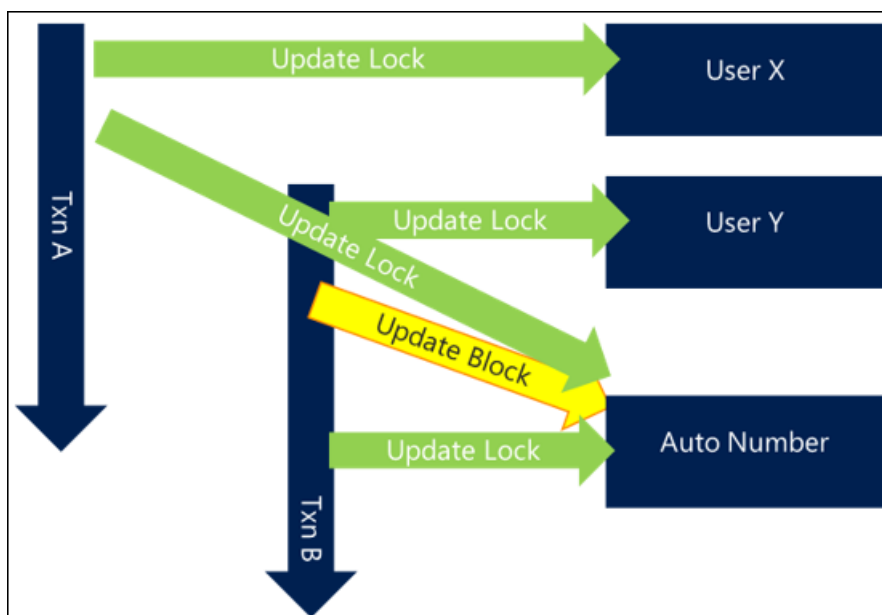
この方法では、まだこのリソースをある程度ブロックすることになりますが、リソースがロックされる時間が短縮されるため、リソースを待っている間に他の要求がブロックされる可能性と時間が減少します。



対話の長さを減らす

同様に、ロックは、2つのプロセスが同時に同じリソースにアクセスする必要がある場合にのみブロックの問題になります。ロックを保持するトランザクションが短くなればなるほど、2つのプロセスが同じリソースにアクセスしたとしても、それらがまったく同時に同じリソースを必要とし、衝突を引き起こす可能性は低くなります。トランザクションの保持時間が短いほど、ブロックが問題になる可能性は低くなります。

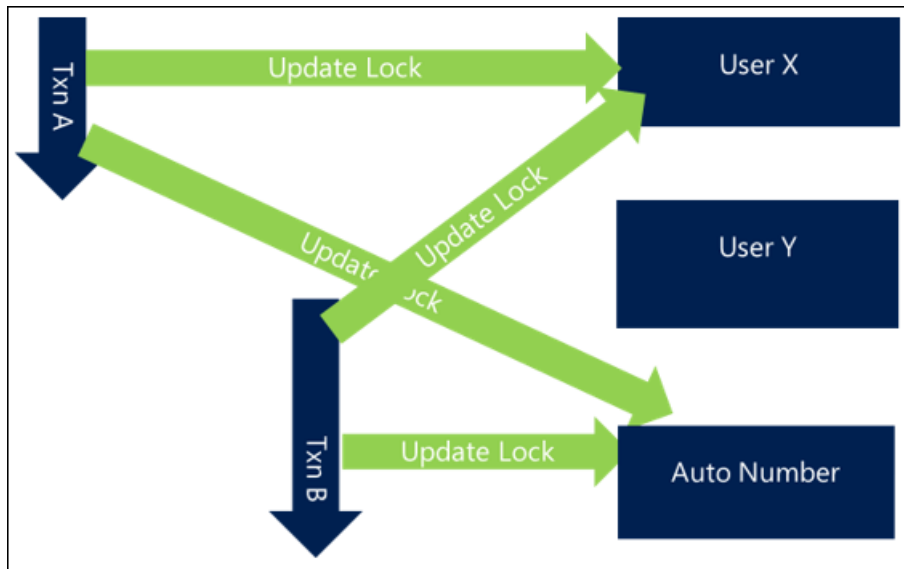
次の例では、同じロックが選択されますが、トランザクション内の他の処理は、トランザクションの全体の長さが拡張され、同じリソースに対する要求が重複することを意味します。つまり、ブロックが発生し、各リクエストは全体的に遅くなるということです。



トランザクションの全長を短くすることで、最初のトランザクションは2番目の要求が開始される前にロックを完了して解放します。つまり、ブロックがなく、両方のトランザクションが効率的に完了します。

トランザクションの寿命を延ばすリクエスト内の他のアクティビティは、特に複数のリクエストが重複している場合にブロックの可能

性を高め、システムが大幅に遅くなる可能性があります。



トランザクションを短くする方法はいくつかあります。

最適化要求

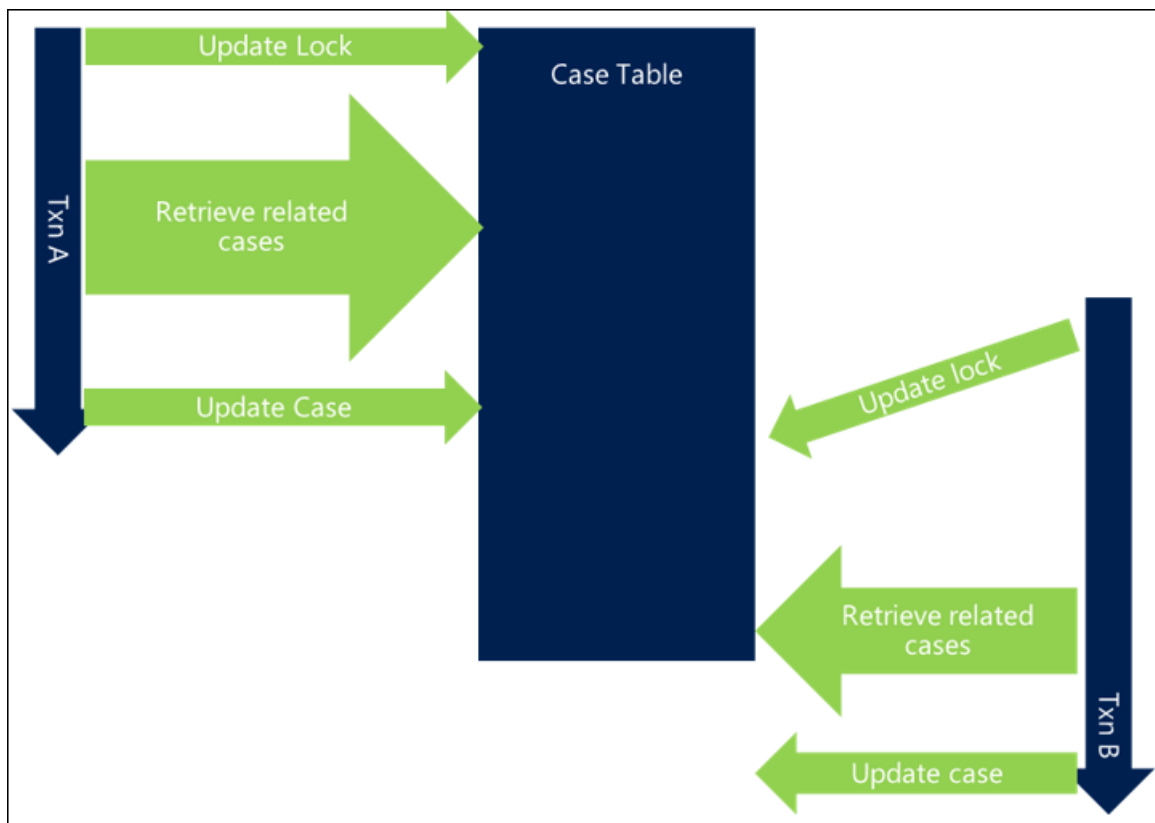
各トランザクションは一連のデータベース要求で構成されています。各要求ができるだけ高く効率的になされれば、トランザクション全体の長さを減らして競合する可能性を低減します。

各クエリを確認して、以下の事を判断します。

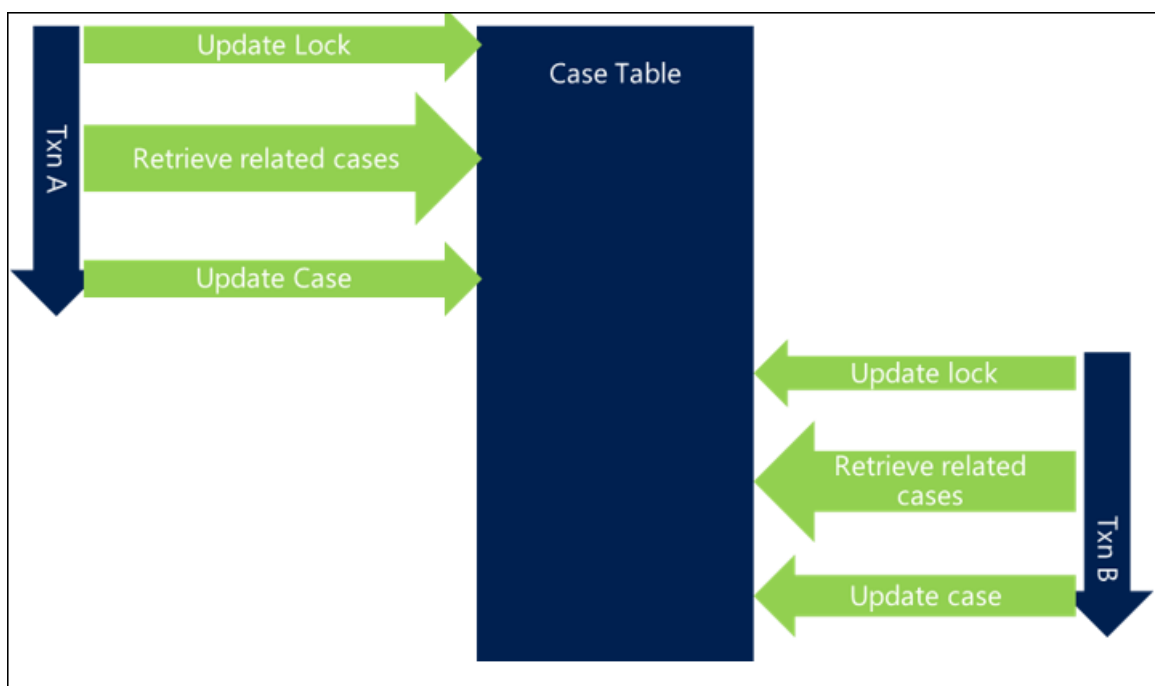
- クエリが、必要なもの、たとえば列、レコード、エンティティタイプなどを尋ねるだけです。
 - これにより、インデックスを使用してクエリを効率的に処理できる可能性が最大化されます
 - アクセスする必要があるテーブルとリソースの数を減らし、データベース サーバ内の他のリソースのオーバーヘッドを減らし、クエリ時間を短縮します
 - 特に、別のテーブルへの結合が要求があったが、その要求を回避される可能性がある場合や、その要求が不要な場合に、不要なリソースをブロックしないようにします
- クエリを補助するためのインデックスが用意されているか、効率的な方法でクエリを実行しているか、スキャンではなくインデックス シークが行われます

インデックスを導入しても、基礎となるテーブル内のレコードの作成または更新がロックされるのを避けることはできません。インデックス自体が変更される可能性があるため、関連レコードが更新されると、インデックス内の項目もロックされます。インデックスが存在しても、この問題を完全に回避することはできません。

次の例では、関連する案件の取得が最適化されておらず、トランザクション全体の長さが増し、スレッド間でブロックが発生しています。



クエリを最適化することで、クエリの実行にかかる時間が短縮され、衝突の可能性が低くなるため、ブロックが減少します。



データベース サーバーが可能な限り効率的にクエリを処理できることを確認することで、トランザクションの全体的な時間を大幅に短縮し、ブロックの可能性を減らすことができます。

イベントのチェーンを削減する

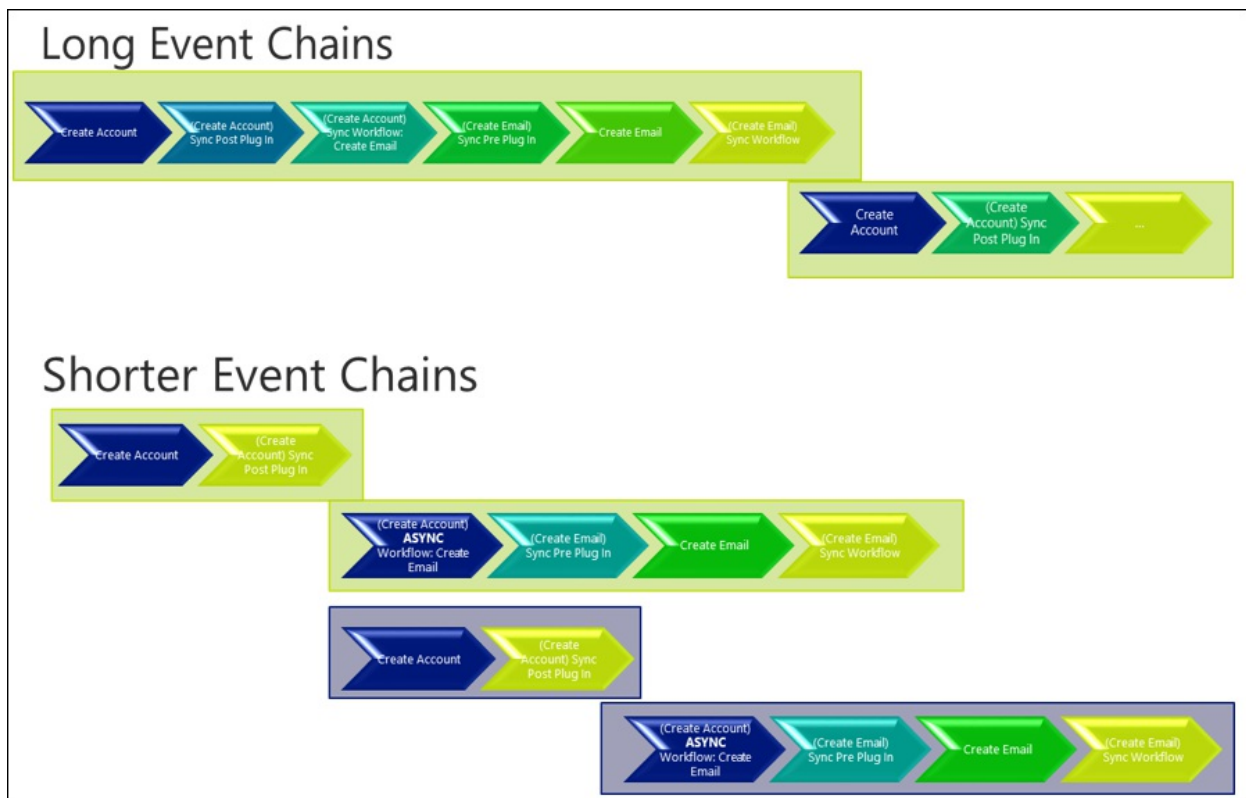
前の例で示したように、関連する一連のイベントの結果はトランザクション時間全体に重大な影響を与えるかもしれないので、ブロックの可能性が生じます。これは、同期プラグインとワークフローをトリガーし、それが他のアクションをトリガーし、さらに同期プラグインとワークフローをトリガーする場合に特に当てはまります。

同期して発生する長いイベントのチェーンを回避するために、実装を慎重に検討および設計することは、トランザクション全体の長さを短縮するのに役立ちます。これにより、実行されたロックをより迅速に解放し、ブロックする可能性を減らすことができます。

また、二次ロックが大きな問題になる可能性も低くなります。アカウント作成の自動付番の例では、最初の問題は自動付番テーブルへのアクセスです。しかし、多くの異なるアクションが1つのシーケンスで実行されると、関連するユーザー レコードへの更新など、二次ブロックも表面化し始める可能性があります。複数の競合するリソースが関与すると、ブロックを回避することがさらに難しくなります。

いくつかのアクティビティが同期または非同期である必要があるかどうかを検討することは、同じアクティビティが達成される一方で初期の影響が少ないという意味です。特に実行時間が長いアクションや、競合の激しいリソースに依存するアクションの場合は、非同期アクションで実行しメインランザクションからそれらを分離すると、大きなメリットがあります。次の自動付番の値で警察犯罪レポートを更新して連続番号スキームを確実に維持するなど、アクションをより広いプラットフォーム ステップで完了または失敗させる必要がある場合、このアプローチは機能しません。これらのシナリオでは、影響を最小限に抑えるために他のアプローチをとる必要があります。

次の例が示すように、単純にアクションをプラットフォーム トランザクションの外側で実行することを意味する、いくつかのアクションを非同期プロセスに移動することによって、トランザクションの長さが短くなり、並行処理の可能性が高まることを意味します。



同じレコードに対する複数の更新を避ける

複数レイヤーの機能アクティビティを設計するときは、必要なアクションを論理的で簡単に従うことができるアクティビティ フローに分割することをお勧めしますが、多くの場合、これによって同じレコードに対する、複数の別々の更新が発生します。

ケースを処理するシナリオでは、最初に提起された顧客に基づいて既存の所有者でケースを更新し、その後、その顧客に自動的に通信を送りケースに対する最終連絡日を更新する別のプロセスを待機することは、機能的には完全に論理的です。

ただし、これは、次のように、Dataverse が同じレコードを更新するための複数の要求があることを意味しています。

- 各要求は個別のプラットフォームの更新であり、Dataverse サーバーに全体的な負荷がかかり、トランザクション全体の長さを増大させるため、ブロックされる可能性が高くなります。
- また、ケース レコードがそのケースで最初に行われたアクションからロックされることを意味します。つまり、ロックは残りのトランザクションを通じて保持されます。ケースが複数の並列プロセスによってアクセスされる場合、それは他のアクティビティのブロックを引き起こす可能性があります。

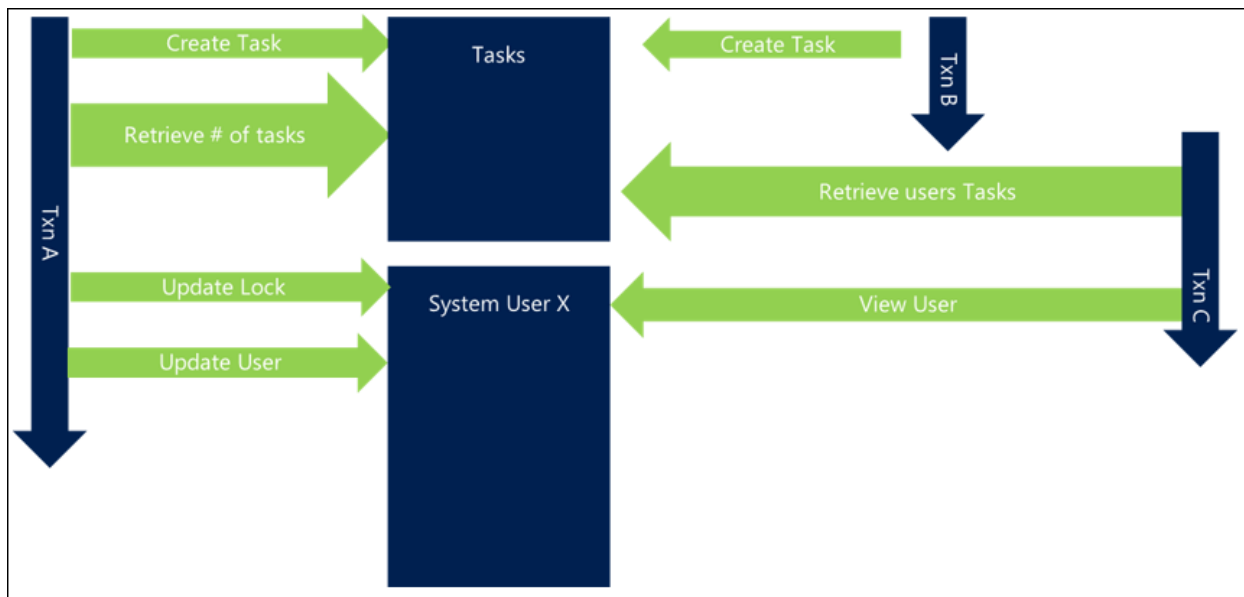
同じレコードへの更新を単一の更新ステップに統合し、その後のトランザクションで、特にレコードが作成後すぐに複数の人によって激しい競合やアクセスがある場合、全体的なスケーラビリティに大きな利点があります。

同じレコードに対する更新を単一のプロセスに統合するかどうかの決定は、実装の複雑さと、個別の更新によってもたらされる競合の可能性とのバランスをとることに基づいて行われます。しかし、ボリュームの大きいシステムでは、これは競合の激しいリソースにとって非常にメリットがあります。

必要なものだけを更新する

有益なアクティビティを除外して Dataverse システムのメリットを減らさないことが重要ですが、ビジネス上の価値はほとんどないが技術的な複雑さを増すカスタマイズを含めるように要求されることがよくあります。

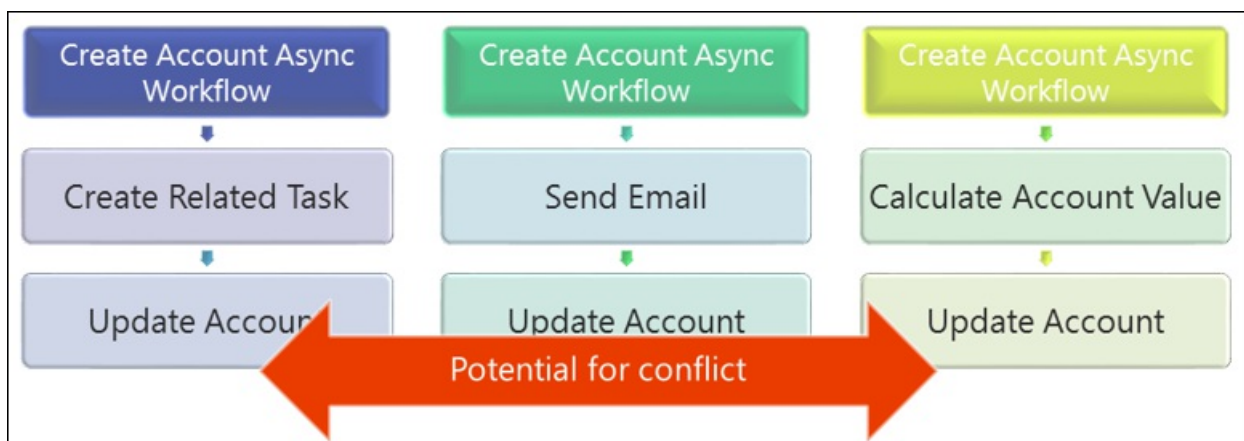
タスクを作成するたびに、現在割り当てられているタスクの数でユーザー レコードも更新すると、ユーザー レコードも競合する可能性があるため、二次レベルのブロックが発生する可能性があります。必ずしもアクションにとって重要ではないにもかかわらず、それが各要求がブロックして待つ必要があるかもしれない別のリソースを追加するでしょう。その例では、ユーザーに対するタスクの数を格納することが重要であるのか、それともオンデマンドで計算できるのか、Dataverse の階層およびロールアップ フィールド機能をネイティブで使用するといったような別の場所に格納できるのかどうかを慎重に検討してください。



後で示すように、システム ユーザー レコードを更新すると、スケーラビリティの観点から悪影響が出る可能性があります。

同じイベントでトリガーされた複数のカスタマイズ

同じイベントで複数のアクションをトリガーすると、リクエストの性質上、これらのアクションが同じ関連オブジェクトまたは親オブジェクトと対話する可能性が高いため、衝突の可能性が高くなります。



これは、慎重に検討または回避する必要があるパターンです。特に異なる人が異なるプロセスを実装するときには、競合を見落としがちなためです。

異なる種類のカスタマイズを使用する場合

カスタマイズの種類ごとに、使用方法が異なります。次の表は、それぞれを考慮して使用する必要がある場合、および使用に適していない場合の一般的なパターンをまとめたものです。

さまざまな動作間の妥協点を検討する必要がある場合が多いので、これにより、考慮すべきいくつかの共通の特性とシナリオについてのガイダンスが得られますが、各シナリオを評価し、すべての関連要因に基づいて適切なアプローチを選択する必要があります。

🔍 / 🔍 🔍 🔍	🔍 / 🔍	🔍 🔍 🔍 🔍	🔍 🔍 🔍	WHEN_NOT_TO_USE_DMM
事前検証	Sync	プラグイン	入力値の短期検証	<p>長期実行アクション。</p> <p>後のステップが失敗した場合にロールバックする必要がある関連アイテムを作成するとき。</p>
事前操作	Sync	ワークフロー / プラグイン	<p>入力値の短期検証</p> <p>プラットフォームのステップの失敗の一部としてロールバックする必要がある関連アイテムを作成するとき。</p>	<p>長期実行アクション。</p> <p>アイテムとその GUID を作成するには、項目に対して保存する必要があるときに、プラットフォームを作成 / 更新します。</p>
事後操作	Sync	ワークフロー / プラグイン	<p>プラットフォームのステップに通常通り従い、後のステップが失敗した場合にロールバックする必要がある短期間のアクション (たとえば、新しく作成されたアカウントの所有者に対するタスクの作成)。</p> <p>作成したアイテムの GUID を必要とし、失敗した場合にプラットフォームのステップをロールバックする必要がある関連アイテムの作成</p>	<p>長期実行アクション。</p> <p>失敗がプラットフォームパイプラインのステップの完了に影響を及ぼさないようにします。</p>
イベント パイプラインには無し	同期	ワークフロー / プラグイン	<p>ユーザー エクスペリエンスに影響を与えと思われる中程度の長さのアクション。</p> <p>失敗した場合にロールバックできないアクション。</p> <p>失敗時にプラットフォームのステップのロールバックを強制するべきではないアクション。</p>	<p>非常に長い時間実行されているアクション。</p> <p>これらは、Dataverse で管理するべきではありません。</p> <p>非常にコストの低いアクション。非常にコストの低いアクションに対して非同期な振る舞いを生成することによるオーバーヘッドは、非常に大きくなる可能性があります。可能であれば、これらを同期的に行い、非同期処理のオーバーヘッドを避けてください。</p>

1 / 1 1111	11 / 111	11111111	111111	WHEN_NOT_TO_USE_DMM
該当なし 呼び出された場所のコンテキストを取ります。		ユーザー定義アクション	Web リソースなどの外部ソースから起動されたアクションの組み合わせ	常にプラットフォーム イベントにตอบสนองしてトリガーする場合は、プラグイン / ワークフローを使用してください。

プラグイン / ワークフローはバッチ処理メカニズムではありません

長時間実行されるアクションや大量のアクションは、プラグインやワークフローからの実行を想定していません。Dataverse は、計算プラットフォームとしての使用を想定したものではありません。特に、関連性のない大規模なアップデートを押し進めるコントローラーとしての使用は想定していません。

それを行う必要がある場合は、Azure ワーカー ロールなどの別のサービスからオフロードして実行します。

セキュリティ設定

非常に一般的なエスカレーション領域は、セキュリティ設定のスケラビリティです。これはコストのかかる操作なので、理解なく慎重に検討せず大量に行うと、問題が発生する可能性が常にあります。

チーム セットアップ

- 常に同じ順序でユーザーを追加します。次の方法でデッドロックを回避します。
- 更新が必要な場合のみユーザーを更新します。不必要にユーザーのキャッシュを無効にしないようにします。

所有者 v. アクセス チーム

- ユーザーのチームが定期的に変わる場合は、所有者チームの頻繁な使用は注意してください。変更するたびに、Web サーバーのユーザー キャッシュが無効になります
- 理想としては、夜間といったようなユーザーが作業をしていないときに変更を加え、影響を軽減します。

たくさんのチーム メンバーシップ / BU

- 多数のチーム / BU が複雑な計算を追加するシナリオを慎重に検討してください。

カスケード動作

- 割り当てなど、カスケード共有を検討する

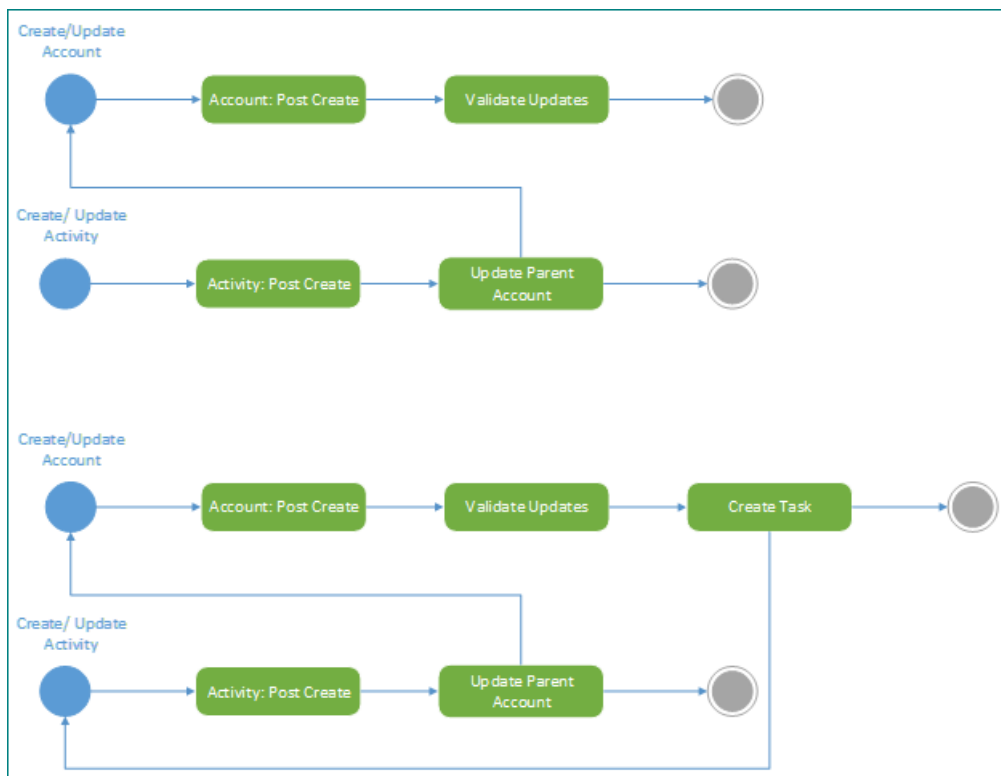
ユーザー レコードの慎重な更新

- 基本的な部分の変更がない限り、システム ユーザー レコードを定期的に更新しないでください。ユーザー キャッシュが強制的にリロードされ、セキュリティ特権が再計算されるため、負荷のかかる作業となります
- 例えば、システム ユーザーを使用して、そのユーザーのオープン アクティビティの数を記録しないようにしてください

ダイアグラム関連のアクション

予防策として、またブロックの問題を診断するためのツールとして非常に有益なアクティビティは、Dataverse プラットフォームでトリガーされる関連アクションを図にすることです。これを行うことは、システム内の意図的および意図的でない依存関係とトリガーの両方を強調するのに役立ちます。解決するために、これができない場合、実装が実際のところ何をするかについての明確な像が、自分の中で描けていない可能性があります。意図しない結果を明らかにすることができるので、実装時にそのような図を作成することをお勧めします。

次の例では、最初は 2 つのプロセスがどのように完璧に連携して動作するのかを強調していますが、継続的なメンテナンスでタスクを作成するための新しい手順を追加すると意図しないループが発生する可能性があります。このドキュメントにある方法を使用すると、設計段階でこれが強調され、システムへ影響を与えないようにすることができます。



システムが取得した統計情報を確認する

問題がデータベースレイヤーの外側で発生した場合、何が起きているのかを判断する方法がいくつかあります。1つ目はプラグイン パフォーマンスの分析です。[PluginTypeStatistic エンティティ](#) でクエリを実行して、プラグインが実行されている頻度と、実行にかかる時間の統計を表示できます。

特定のエラーが発生している場合は、サーバーのトレース ファイルを使用して、関連する問題がプラットフォーム内のどこで発生している可能性があるのか知る際にも便利です。詳細: [トレースの使用](#)

サマリー

[Dataverse でのスケーラブル カスタマイズ設計](#) の内容とそれに続くトピック [データベースのトランザクション](#)、[同時実行の問題](#)、このトピックでは、次の概念と、Dataverse のスケーラブルなカスタマイズを設計および実装する方法を理解するのに役立つ例と方法を紹介しました。

覚えておくべきいくつかの重要なことは次のとおりです。

ロック / トランザクション

- ロックとトランザクションは、健全なシステムにとって不可欠です
- しかし、誤って使用すると、問題が発生する可能性があります

プラットフォームの制約

- プラットフォームの制約はしばしばエラーの形で現れます
- しかし、制約が問題の原因であることはめったにありません
- プラットフォームや他のアクティビティが影響を受けないよう、保護するために存在します

トランザクション用の設計

- 実装がトランザクションの動作を念頭に置いて設計されている場合、これにより、はるかに高いスケーラビリティとユーザー パフォーマンスの向上がもたらされます

NOTE

ドキュメントの言語設定についてお聞かせください。 [簡単な調査を行います](#)。(この調査は英語です)

この調査には約 7 分かかります。個人データは収集されません ([プライバシー ステートメント](#))。