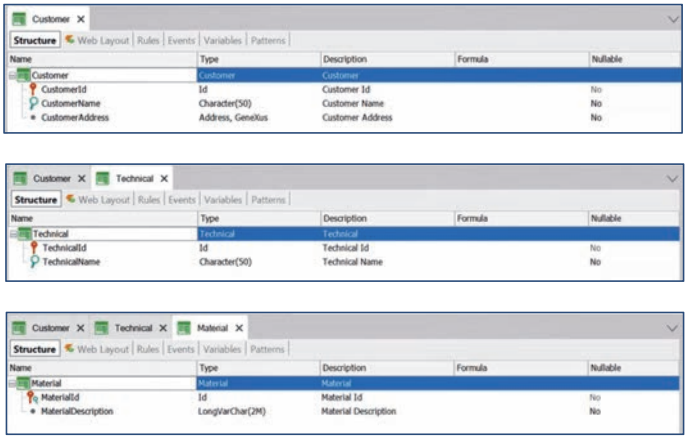
並行トランザクション

GeneXus™

GeneXus

並行トランザクション



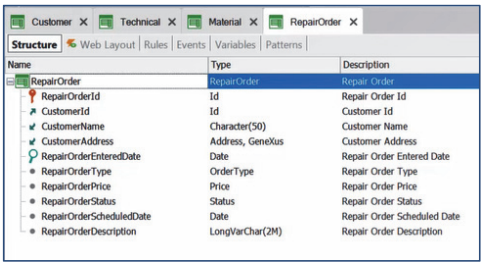
客先で技術サービスを提供する企業向けの小規模なアプリケーションを設計しているとします。このアプリケーションから、顧客から依頼された修理依頼の注文の作成などが可能です。

この例では、関心のあるトランザクションのみを見てみましょう。
顧客情報を記録するトランザクション、会社の技術者を記録するトランザクション、技術者が顧客の問題を解決するために使用する資料を記録するためのトランザクションがあります。

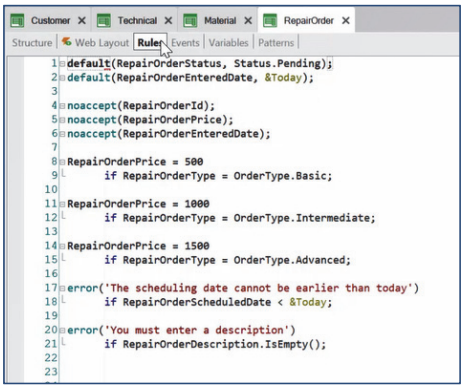
GeneXus

RepairOrder Trn

構造



ルール



また、顧客の依頼を受けたオペレーターが、サービスの注文を作成するために必要なすべてのデータを入力する RepairOrder トランザクションがあります。
このトランザクションでは、自動採番される RepairOrderId 項目属性が主キーとなっています。各注文には顧客が関連付けられているので、このトランザクションでは Customer テーブルから項目属性を推論しています。
トランザクションには、そのほかにも項目属性があります。入力日、注文の種類 (データタイプは複数の値を取る列挙型)、料金、ステータス (このデータタイプも複数の値を取る列挙型)、予定日、デスクリプションを記録するための項目属性です。

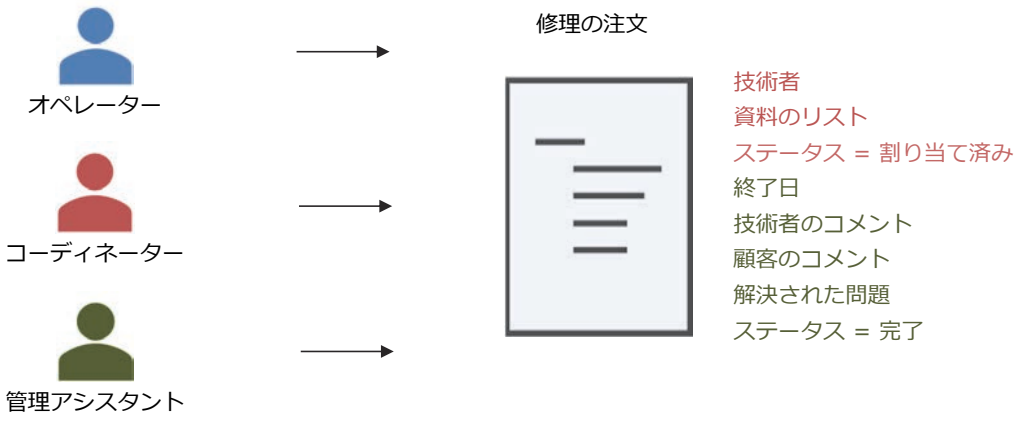
入力したルールを見てみましょう。

「default」ルールおよび「noaccept」ルールがいくつかあります。
1 つ目の default ルールは、注文のステータスを保持する項目属性に Pending (保留中) の「P」を割り当てます。2 つ目では作業指示書の入力日を保存する項目属性に今日の日付を割り当てます。

また、このトランザクションの RepairOrderPrice という項目属性は、注文の種類を記録する項目属性 RepairOrderType に入力された値から計算されることが分かります。
最後に、いくつかの「error」ルールを宣言します。

修理の注文を入力して、このトランザクションの実行を見てみましょう。

注文の入力および修正

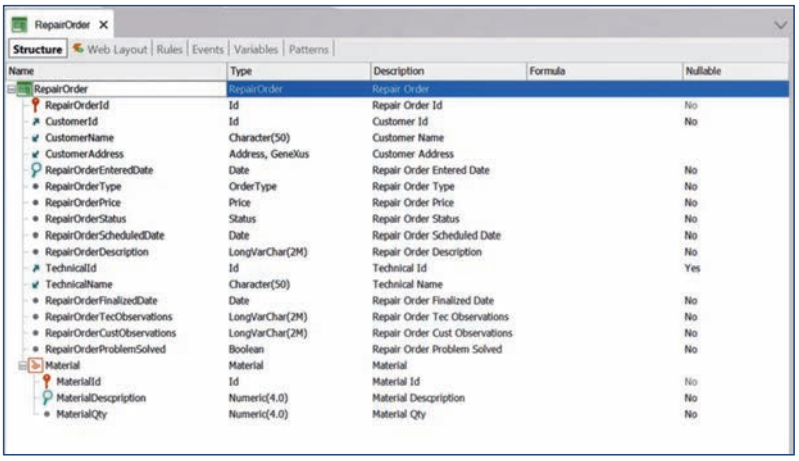


現実では、注文が入力された後、モニターするためにコーディネーターが技術者を割り当て、その技術者に提供する資料を割り当てる必要があるとします。また、ステータスを割り当て済みに変更する必要があります。

技術者がタスクを完了すると、管理アシスタントは終了日、技術者および顧客からのコメント、問題が解決されたかどうかを注文に記録し、ステータスを完了に変更できる必要があります。

これはどうすれば実現できるでしょうか。

RepairOrder Trn とすべての項目属性



これらすべての項目属性をトランザクションに追加し、必要なものをその都度補完するのも 1 つの方法です。
このようなトランザクションになります。

実行環境で確認してみましょう。

これはユーザーにとって使いやすいインターフェースとは言えません。関心のないフィールドや入力する必要のないフィールドが常に表示されるためです。

たとえば、オペレーターが顧客から注文を受け取り、それを初めて入力するとき (技術者に割り当てる前) に、入力されないフィールドがいくつか表示されます。これは比較的少ない方ですが、項目属性の数がもっと多くなると、ユーザーエクスペリエンスやシステムの操作性に影響が生じます。
コーディネーターがトランザクションにアクセスし、変更する注文を選択して更新モードに変更して、技術者を割り当て、資料をロードする場合も同様のことが起こります。

注文を検索して画面にロードすると、確認する必要のない項目属性も表示されます。資料をロードするためには、後で入力するフィールドを空にしておく必要があります。
これはデータ入力の点で使いやすいとは言えません。先ほど言ったとおり、この例は項目属性が少ないため、大きな影響はありません。ただし、項目属性の数が増えると、より多くのデータを記録する必要が生じるため、入力プロセスがさらに複雑になります。

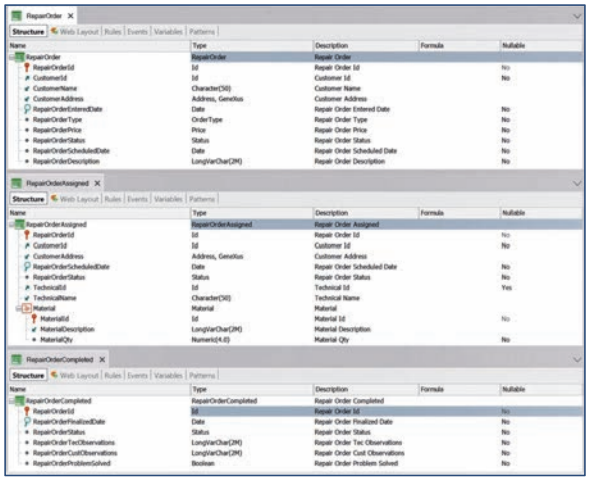
管理アシスタントがコメントを付けて注文を完了するときには、対象となる注文を選択してトランザクションを更新モードにする必要があります。このとき、いくつかの詳細情報を確認するだけなのに、すべての項目属性が表示されます。

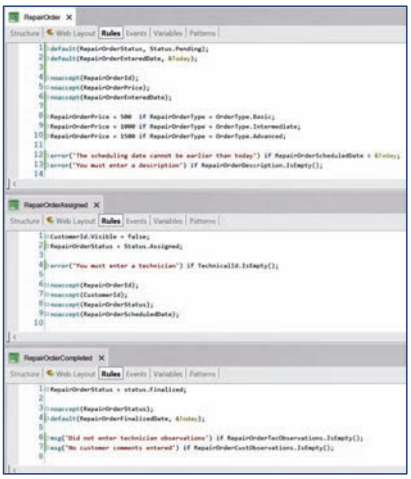
もう 1 つのデメリットは、このトランザクションの動作のプログラムが非常に複雑になることです。ここまでに見てきたように、このアプリケーションでは、後でステータスを変えたり、変更を加えたりします。また、項目属性自体も、毎回異なるコントロールが必要になることがあります。しかし、この設計では、すべてが同じオブジェクトでプログラムされます。先ほど確認したようなさまざまな時点に応じてルールやイベントをカスタマイズすることはできません。

これらのデメリットは、**並行トランザクション**と呼ばれるものを使用することで簡単に解決できます。

GeneXus

並行トランザクション



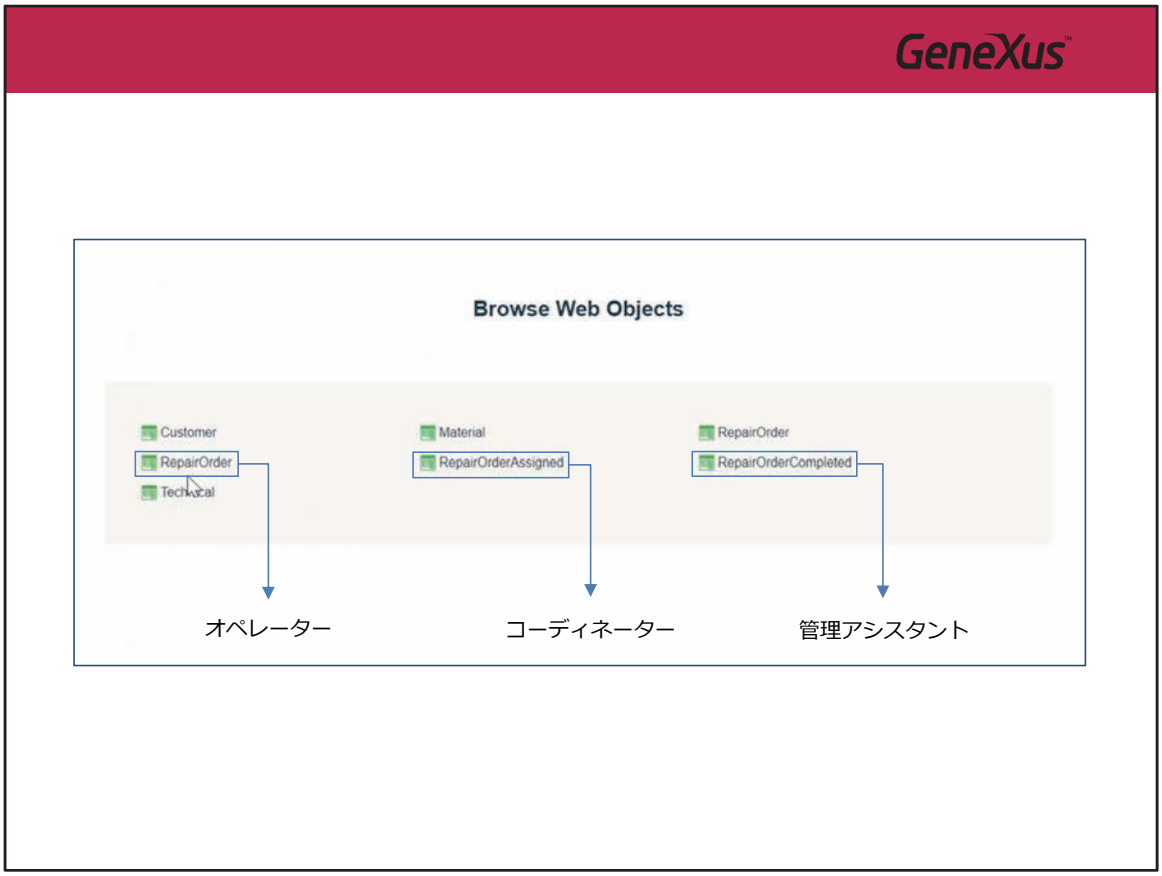


まず、オペレーターが注文を入力するトランザクションがあります。次に、コーディネーターがその注文を技術者に割り当て、提供する資料をロードするトランザクションがあります。最後に、管理アシスタントが技術者および顧客のコメント (ある場合) と、問題が解決されたかどうかをアップロードするトランザクションがあります。これらのトランザクションは、それぞれ実際に関係のある項目属性が含まれています。

そのためには、3 つのトランザクションすべてが同じ項目属性を主キーとして持っている必要があります。これにより、複数のトランザクションが並行トランザクションになります。
重要なのは、トランザクションの並行化はレベル単位で行われるということです。この例では、これらのトランザクションのメインレベルは並行になっています。

トランザクションの [Rules] エレメントを見ると、それぞれ完全に独立した独自のルールを持っていることが分かります。これは、並行トランザクションを使用する大きなメリットの 1 つです。同じことが、プログラムするすべてのイベントに当てはまります。

では、これらの変更を加えたアプリケーションを実行してみましょう。

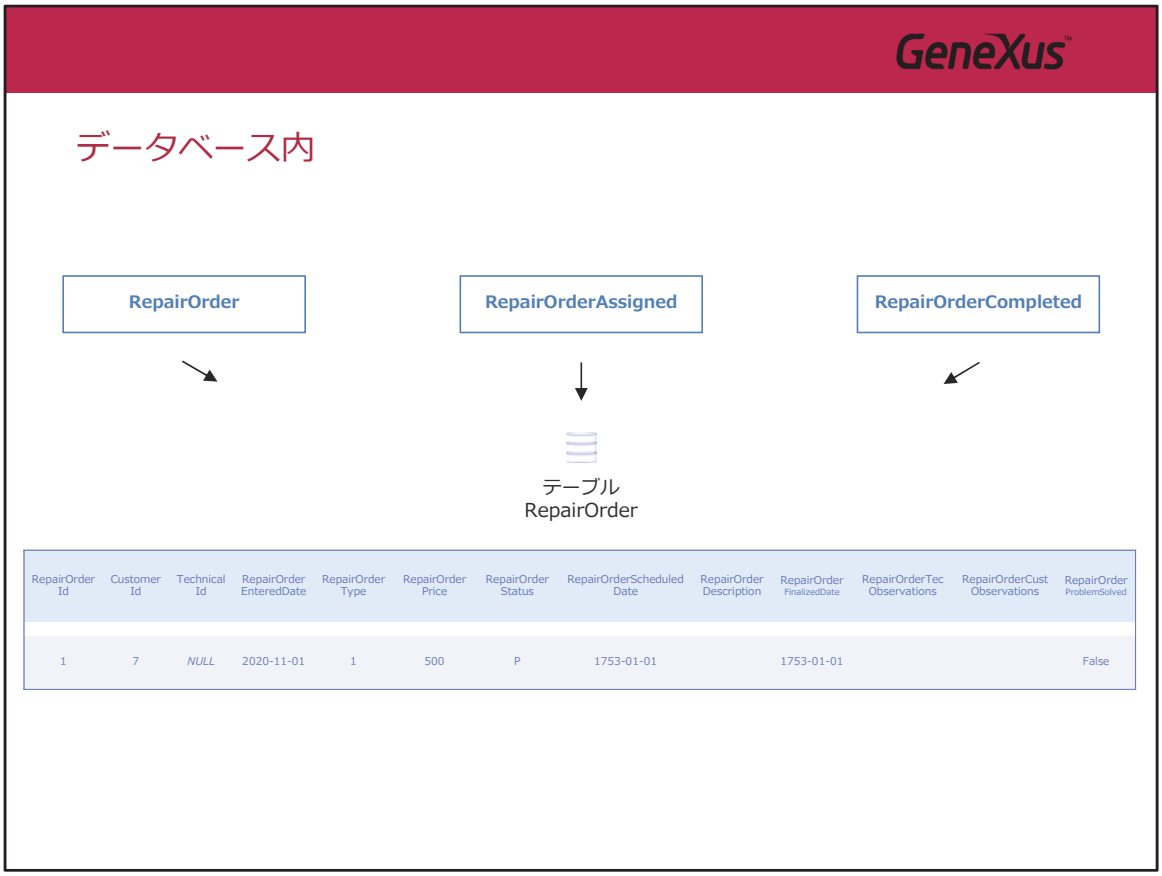


オペレーターは、RepairOrder トランザクションから顧客の依頼を受け取り、すべてのフィールドに入力します。入力すると、default ルールの定義により、保留中を示す値「P」が自動的に割り当てられます。

その後、コーディネーターが RepairOrderAssigned トランザクションにアクセスして注文を検索し、必要なデータを入力します。注文のステータスを表す値が、自動的に「A」(割り当て済み) に変更されます。

最後に、タスクが完了し、技術レポートを受け取ると、管理アシスタントは 3 つ目の RepairOrderCompleted トランザクションで、対応するフィールドに必要事項を入力して管理します。注文のステータスを表す値が自動的に「F」(完了) になります。

このように、ユーザーにとって非常にシンプルで分かりやすく、より直観的に操作できるようになります。また、将来、注文入力のプロシージャーの変更に対応できるよう、アプリケーションの拡張性を確保するという点でも役立ちます。



データベースレベルでは、これら 3 つのトランザクションのメインレベルから 1 つの物理テーブルが生成されます。これは、GeneXus が正規化の基準に従ってデータベースを設計するためです。

生成されたテーブルには、3 つのトランザクションを関連付けた結果としての項目属性が含まれます。
1 つ目のトランザクションでレコードを入力すると、データベースのテーブルで、値を割り当てていない項目属性はどうなるでしょうか。それらはどのような値を持つでしょうか。

既定では、空の値は、次のように割り当てられます: Numeric フィールドは「0」、Character タイプのフィールドは空の文字列、Date タイプのフィールドは空の値を表す既定の日付、ブール値の項目属性は False。

並行トランザクションは、同じテーブル内にある、別のトランザクションで処理される情報に関係なく、そのプログラム内で扱いたい情報のみを各トランザクションの構造に保持する明確な方法を提供します。

このタイプのトランザクションを使用することになる現実にはいろいろなものがありますが、ここでは詳しい説明は行いません。このトピックの詳細については、GeneXus の Wiki を参照してください。