ORACLE
DATABASE

Oracle Database 12c Release 2でのトランザクション・ガード

システム停止の不可視化

Oracle ホワイト・ペーパー | 2017 年 3 月

ORACLE

目次

概要

Oracle Database 12c のトランザクション・ガードは、実際の結果ではなく、エラーやタイムアウトがクライアントに返された場合に、現在実行されているトランザクションのコミット結果を返す、信頼性のあるプロトコルおよびインタフェースです。アプリケーションまたはインフラストラクチャは、エラー処理にトランザクション・ガード API を組み込んで、システム停止が発生したときに、トランザクション・ガードを使用して実際の結果を返します。

トランザクション・ガードによって、ユーザーのフラストレーション、カスタマー・サポートへの問合わせ、機会損失の原因となる曖昧なエラーをクライアントが受け取ることで発生するコストがかからなくなります。トランザクション・ガードがないと、エラーやタイムアウト後にアプリケーションやユーザーによって操作が再試行されるため、重複したトランザクションや順序が正しくないトランザクションがコミットされ、論理的破損が発生する可能性があります。トランザクション・ガードは、既知のコミット結果や最大 1 回の実行が可能な自社開発ソリューションや外部ソリューションと比べて、オーバーヘッドを低く抑えてパフォーマンスを大幅に向上しながら、正確性を保証し、クラウド・レベルやインターネット・レベルに規模を拡大します。

Oracle Database 12c の事例および Oracle Open World 2015 でのお客様の声

「トランザクション・ガードは、広く普及してインターネットの標準となるべきです。」「トランザクション・ガードを使用すれば、当社のアプリケーション・プログラマーはほとんどのエラー状況に適切に対処できます。」

「トランザクション・ガードを使用すると、POS リクエストを非同期で送信できるため、お客様をお待たせすることはありません。」「トランザクション・ガードにより、インフラストラクチャに追加のオーバーヘッドが生じることなく、売上の送金を把握できます。」「トランザクション・ガードの節約効果は、満足度の向上、サポートへの問い合わせの減少、オーバーヘッドの軽減などに波及しています。」「トランザクション・ガードを使用することで、不確実なリクエスト取消しではなくトランザクション・ガードによる確実なリクエスト取消しが可能になったため、再実行が重複する恐れがなくなりました。」

はじめに

トランザクション・ガードを使用すると、停止後に、最後の送信がコミットおよび完了されたかどうかがアプリケーションとユーザーに返されるため、エンドユーザー・エクスペリエンスが大幅に向上します。リクエストの送信後に、資金の送金、請求書の支払い、書類の提出などが実行されたかどうか分かること、または実行されておらず再送信しても問題ないと分かることは、大変重要です。トランザクション・ガードを使用しなければ、停止後にユーザーは曖昧で不確かなエラー・メッセージを受信し、最後に実行中だった操作に何が発生したのかを把握できないままになる可能性があります。アプリケーションには通常、次のようなフラストレーションの原因となりうるメッセージが表示される場合があります。

» カスタマー・サポートにお問い合わせください

» 再送信やリロードをクリックしないでください

» BackSpace キーを使用しないでください

» 現在システムで問題が発生しています。後で再度お試しください。

»

開発者は、トランザクション・ガード API をアプリケーションや中間層のエラー処理に組み込むことで、確実なリクエスト結果を強制できます。トランザクション・ガードのプロトコルにより、コミット結果がアプリケーションに返されるときに、アプリケーションに返される値がその結果に永続的に保持されます。トランザクション・ガードを使用して、コミット済みの、またはコミットされていない結果がアプリケーションに返されると、結果はそのままの状態になります。これは非常に重要です。コミット済みの結果はコミット済みのままです。コミットされていない結果はコミットされていないままのため、たとえばユーザーが再送信できるという合図となります。また、停止時にアプリケーション自身で再送信できるという合図にもなります。図 1 では、トランザクション・ガードを使用しない以前のエクスペリエンスを示しています。リクエストに対してエラーやタイムアウトが返されると、ユーザーは送信したリクエストがどうなったかを把握できません。

図1：リクエストがコミットされてもユーザーはエラーを受信する可能性がある

図2：トランザクション・ガードを採用すると、エンドユーザーは曖昧なエラーではなく実際のレスポンスを受信

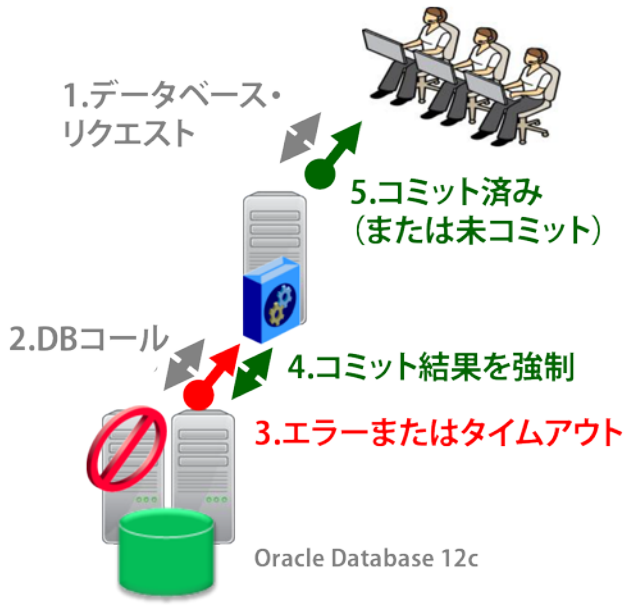


図 2 では、トランザクション・ガードを採用して実際のコミット結果が返されるようになると、ユーザー・エクスペリエンスが大幅に向上することを示しています。図 2 は、以下のような流れです。

» （手順 1）クライアントがアプリケーションにリクエストを送信する

» （手順 2）アプリケーションが Oracle Database 12c にデータベース・コールを送信する

» （手順3）リクエストの実行中に基盤となるシステムまたはネットワークで障害が発生したため、エラーまたはタイムアウトが発生する。標準のエラー処理がこの例外を捕捉する

» （手順4）エラー処理でトランザクション・ガードが起動され、そのセッションで最後に実行中であった処理のコミット結果が返される

» （手順5）曖昧なエラーではなく信頼できるコミット結果が返されるため、ユーザーは処理が成功したか、しなかったかを把握できる

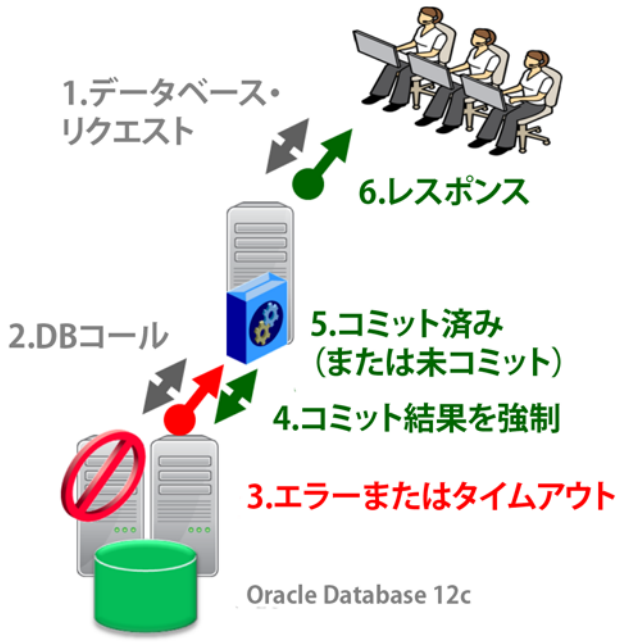
トランザクション・ガードを採用すると、アプリケーションと中間層は問題なく成功を返したり、未コミットの結果を受信した後に自身で再送信したりできます。図 3 は図 2 のワークフローを強化したものであり、手順 5 と手順 6 でアプリケーションは再送信を確実に実行します。Oracle Database 12c では、多くのアプリケーションは、不確実なリクエスト取消しではなくトランザクション・ガードによる確実なリクエスト取消しの後に、再実行を行います。また、アプリケーションの下層ですべての処理を代わりに実行してくれるアプリケーション・コンティニュイティを使用することもできます。

図 3 は図 2 の続きであり、手順 5 が次のように置き換えられています。

» （手順5）コミットされた場合、コミット済みの結果をユーザーに返す。コミットされなかった場合、アプリケーションは自身で再送信しても問題ない

» （手順6）成功すると、ユーザーにレスポンスが返されるため、ほとんどの場合、エラーが発生しなかったかのように処理が完了する

図3：トランザクション・ガードの採用により、アプリケーションによる確実な取消しと再実行が可能に



トランザクション・ガードの使用には、次のような利点があります。

» ビジネスの場合、ユーザー・エクスペリエンスが大幅に向上し、サポートへの問い合わせや機会損失が減少します。

» ユーザーの場合、停止後に送信した最後の処理について信頼できるコミット結果が得られます。

» 開発者の場合、停止への対処とエラーの処理を適切に行うことで、生産性が向上します。

» 全体では、リクエストの取消しと再送信において、自社開発ソリューションや外部ソリューションと比較して、クラウドのスケーラビリティが実現し、パフォーマンスと安全性が向上します。

Oracle Database 12cでのアプリケーション・フェイルオーバーの新しい概念

論理トランザクション識別子（LTXID）

アプリケーションでは、論理トランザクション識別子（LTXID）と呼ばれる概念を使用して、停止後にデータベース・セッションで開かれている最後のトランザクションのコミット結果が判断されます。LTXID はデータベースによって所有され、OCI セッション・ハンドル内、および JDBC Thin ドライバと、管理対象と管理対象外の両方の Oracle Data Provider for .NET（ODP.NET）ドライバの接続オブジェクト内にコピーが保持されます。論理トランザクション識別子は、コミット結果を得るために使用されるほか、データベース・リンク経由でローカル・トランザクション、1 フェーズ・トランザクション、および 2 フェーズ・トランザクションの最大 1 回のセマンティックを強制するために使用されます。

信頼できるコミット結果

クライアントは、トランザクション REDO の書込み後に生成された、コミット結果と呼ばれる Oracle メッセージを受け取ったときに、トランザクションがコミットされたと認識します。しかし、COMMIT メッセージは永続的ではありません。このメッセージが失われると、アプリケーションは、トランザクションがコミットされたかどうかとは関係のないエラーやタイムアウトを受信する場合があります。トランザクション・ガードは、リカバリ可能なエラーの後にコミット結果が失われた場合に、コミット結果を確実に取得します。確実性は重要な要素です。トランザクション・ガードは、エラーがいつどこで発生しようと、コミット済みか、未コミットかを返します。トランザクションがいったんトランザクション・ガードによって強制されると、返された結果は、問合せの回数にかかわらず変更されることはありません。

リカバリ可能なエラーの分類

リカバリ可能なエラーとは、実行中のアプリケーション・セッションのロジックとは関係なく、外部のシステム障害によって発生するエラーです。リカバリ可能なエラーは、計画メンテナンスの間、およびセッション、ネットワーク、ノード、ストレージ、データベースの計画外停止の後に発生します。また、システムが応答せずにリクエストがタイムアウトになった場合にも発生します。アプリケーションはエラー・コードを受け取りますが、この方法では、最後に送信されたユーザー・コールのステータスをアプリケーションで把握できない可能性があります。リカバリ可能なエラーの機能は Oracle Database 12c で強化されており、より多くのエラーに対応し、OCI 用のパブリック API が追加されています。新しい API を使用すれば、アプリケーションのコード内でエラー番号の一覧を記述する必要はなくなります。

» この API はトランザクション・ガードと関係しているため、コミット済みかどうかは、エラー・コードがリカバリ可能かどうかによって決まることはありません。たとえば、OSD エラーを受け取ったからといって、最初にコミットされなかったとは限りません。ただし、アプ

リケーションが自動的に再送信を行うことを予定している場合は、受信した元のエラー・コードがリカバリ可能であれば、再送信が成功する可能性は高いでしょう。リカバリ可能なエラーの API は、以下のようにドライバごとです。

» JDBC Thin ドライバ用の SQLRecoverableException

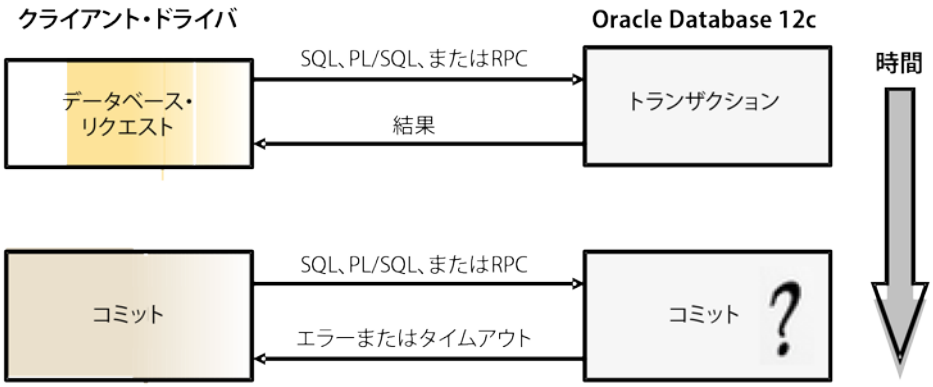
» ODP.NET 用の OracleException.IsRecoverable プロパティ

» OCI 用の OCI_ATTR_ERROR_IS_RECOVERABLE エラー・ハンドル属性

トランザクション・ガードが解決する問題

停止後にアプリケーションをリカバリする際の根本的な問題の 1 つは、クライアントに返送されるコミット・メッセージが永続的でないことです。クライアントとサーバー間に途絶されている箇所がある場合、通信が失敗したことを表すエラー・メッセージがクライアントに表示されます。このエラーでは、送信によってコミット操作が実行されたかどうかや、図 4 に示すように、プロシージャ・コールが予定されたすべてのコミットやセッション状態の変更を完了まで実行したのか、コールが途中で失敗したのか、またはクライアントから切断されたまま現在も実行中というさらに悪い状態になっているのかは、アプリケーションには通知されません。

図4：トランザクション・ガードが失われたコミット結果を判断して解決



通信エラーやタイムアウトの後で、最後のコミット操作の結果を信頼できるスケーラブルな方法で判断することは、いまだに解決されていない問題です。データベースへの送信がコミットされたかどうかをアプリケーションが把握する必要がある場合は、カスタムの例外コードを追加して、アプリケーション内のすべての可能なコミット・ポイントに対して結果を問い合わせる必要があります。システムのあらゆる場所で障害が発生する可能性があることを考えると、送信ごとに固有なチェックが必要となるこの方法は、一般的に非実用的です。アプリケーションを構築して本番環境に移行した後は、完全に非実用的です。さらに、チェックを実行した直後にトランザクションがコミットされる可能性もあるため、チェックによって正確な回答を得ることはできません。実際、クライアントがタイムアウトを受信した後、クライアントがリクエストを破棄したことに気付かずに、サーバーが送信を実行し続ける場合があります。コミットの問題に加えて、データベース内で PL/SQL または Java を使用する場合、プロシージャ型の送信が完了まで実行されたか、または途中で中断されたかに関する記録も存在しません。そのようなプロシージャがコミットされながらも、そのプロシージャに必要な後続の操作が実行されていない場合があります。

最後の送信がコミットされたか、近いうちにコミットされるか、または完了まで実行されなかったかを把握できないと、再送信を試行するユーザーとアプリケーションは、すでに永続化されている変更の再実行を試みて、トランザクションの重複と他の形態の"論理的破損"を引き起こす可能性があります。ユーザーが、プロセスの必要な手順を行わずに、次の操作に進む場合もあります。

Oracle Database 12cでのトランザクション・ガードの対応範囲

トランザクション・ガードは、単一インスタンスのデータベース、Real Application Clusters、Data Guard、Active Data Guard での再起動やフェイルオーバーを含め、システム内の各データベースで使用できます。トランザクション・ガードは、以下の Oracle Database 12c 構成を備えた Enterprise Edition、およびその上位エディションでサポートされます。

» 単一インスタンスの Oracle RDBMS

» Oracle Real Application Clusters

» Oracle RAC One

» Oracle Data Guard

» Oracle Active Data Guard

» アンプラグ/プラグおよびオンライン PDB 再配置を含むマルチテナント

» 上記のデータベース構成の Oracle Global Data Services

トランザクション・ガードでは、Oracle Database 12c に対して以下のトランザクション・タイプがサポートされます。

» ローカル・トランザクション

» DDL トランザクションと DCL トランザクション

» 分散トランザクションとリモート・トランザクション

» パラレル・トランザクション

» 成功時のコミット（自動コミット）

» 埋込みコミットを使用した PL/SQL

» Oracle Database 12c Release 2（12.2.0.1）より、XA コミット・フラグ TMONEPHASE と読取り最適化を含む 1 フェーズ最適化を使用した XA トランザクション

» サービスでトランザクション・ガードを使用する場合の、Service 句を使用した ALTER SESSION SET Container

トランザクション・ガードでは、以下のクライアント・ドライバがサポートされます。

» 12c JDBC Thin ドライバ

» 12c OCI および OCCI クライアント・ドライバ

» 12c ODP.NET 管理対象ドライバ

» 12c ODP.NET 管理対象外ドライバ

トランザクション・ガードでの除外事項

» トランザクション・ガードは、再帰的トランザクションおよび自律型トランザクションを意図的に除外し、これらを再実行できるようにします。Oracle Database 12c Release 2 でのトランザクション・ガードでは、以下が除外されます。

» 転送トランザクション用の読取り/書込み DB リンクのある Active Data Guard

» JDBC OCI はサポートされません。JDBC Thin ドライバを使用してください

» 外部で管理される 2 フェーズ XA トランザクション。XA を使用する場合、トランザクション・ガードは、1 フェーズ XA トランザクションのコミット結果を返し、外部で管理される 2 フェーズのコミット結果を無効化します。この結果が TP モニターによって所有されるためです（「トランザクション・ガードと XA トランザクション」を参照）

» トランザクション・ガードは、TAF、または OCI および ODP.NET のアプリケーション・コンティニュイティ向けの TAF コールバック、もしくは Java のアプリケーション・コンティニュイティ向けの JDBC 初期化コールバックでは使用できません。TAF とアプリケーション・コンティニュイティは、トランザクション・ガードを内部で処理します。

» トランザクション・ガードを有効化した状態で全データベースのインポートは実行できません。全データベースのインポートには、トランザクション・ガードを使用せずに管理サービスを使用してください。ユーザーとオブジェクトのインポートは除外されません。

トランザクション・ガードでは、以下のレプリケーション・テクノロジーによって保守されるデータベース間のフェイルオーバーが除外されます。

» Oracle Golden Gate へのレプリケーション

» ロジカル・スタンバイへのレプリケーション

» サード・パーティのレプリケーション・ソリューション

» PDB クローン句（PDB オンライン再配置 12c Release 2 以降を除く）

Golden Gate、ロジカル・スタンバイ、サード・パーティのレプリケーション・ソリューションなどのレプリケーション・テクノロジーを使用したデータベース・レプリカを利用している場合、この構成では、1 次データベースと 2 次データベース間でトランザクション・ガードを使用できません。トランザクション・ガードは、レプリケーションに参加している各データベース内では使用できます。このユースケースでは、データベースごとに異なるデータベース識別子（DBID）を使用する必要があります。各データベースの DBID を取得するには、V$DATABASE を使用します。

トランザクション・ガードと XA トランザクション

Oracle Database 12c Release 2 より、トランザクション・ガードでは、1 フェーズ最適化の結果を判断するために XA トランザクションがサポートされます。トランザクション・ガードでは、コミット操作中に TMONEPHASE を使用するローカル・トランザクションと XA トランザクションがサポートされます。アプリケーションが TMTWOPHASE を使用する XA トランザクションを発行する場合、トランザクション・ガードはそのトランザクションのために自身を無効化し、自動的に再有効化して次のトランザクションに備えます。この方法により、トランザクション・ガードは XA データソース上の以下のトランザクションをサポートできます。

» 自動コミットを使用するローカル・トランザクション

» 明示的なコミットを使用するローカル・トランザクション

» TMONEPHASE フラグを使用してコミットする XA トランザクション

TP モニターとアプリケーションは、トランザクション・ガードを使用して、これらのトランザクション・タイプのコミット操作結果を取得できます。トランザクション・ガードは、外部で管理される TMTWOPHASE コミット操作のために自身を無効化し、自動的に再有効化して次のトランザクションに備えます。トランザクション・ガード API が TMTWOPHASE トランザクションとともに使用される場合、トランザクション・ガードは無効化されているため、警告メッセージが返されます。TP モニターそのものが、TMTWOPHASE のコミット結果を所有します。この機能により、TP モニターは、TMONEPAHSE 操作の明白な結果を返すことができるようになります。

トランザクション・ガードを使用するためのOracle Database構成

必要な手順

Oracle Database Release 12.1 以降を使用します。

すべてのデータベース処理にアプリケーション・サービスを使用します。Oracle RAC を使用する場合は srvctl を、Oracle RAC を使用しない場合は DBMS_SERVICE を使用してサービスを作成します。GDSCTL を使用することもできます。

トランザクション・ガード用に、サービスにプロパティ COMMIT_OUTCOME = TRUE を設定します。アプリケーション・ユーザーに DBMS_APP_CONT パッケージの実行権限を付与します。

DDL 文とともにトランザクション・ガードを使用する場合、DDL_LOCK_TIMEOUT を増加させます（たとえば、10 秒などに）。

推奨される手順

サービス・パラメータ RETENTION_TIMEOUT を参照し、履歴が保持される期間（秒単位）を確認します。この値を、24 時間以上などの大きい値に保ちます。デフォルト値で十分です。

パフォーマンスを確認します。最適なパフォーマンスを得るために、必要に応じてトランザクション履歴表（LTXID_TRANS）を見つけます。マルチテナントを使用している場合は、PDB ごとにトランザクション履歴表が 1 つあります。

Oracle Real Application Clusters（RAC）または Oracle Data Guard を使用している場合、12c データベース・クライアントと通信できるよう、FAN が ONS を使用して構成されていることを確認します。

サービスで、AQ_HA_NOTIFICATIONS = TRUE を設定します（OCI および ODP.NET FAN 向け）。

接続プールを使用します。また、セッションの有効期限を短くしないでください。これは、基本的なパフォーマンスを得るために重要です。これにより、トランザクション・ガードのコストも極めて低くなります。

サービス・パラメータ

データベース・サービスは使用しないでください。このサービスは管理用であり、アプリケーション用ではないためです。つまり、db_name、db_unique_name、または pdb_name に設定されているデフォルトのサービスは使用しないでください。

COMMIT_OUTCOME

説明：COMMIT_OUTCOME は、コミットが実行された後で、トランザクションのコミット結果にアクセスできるかどうかを確認します。データベースではコミットが常に永続化されますが、トランザクション・ガードではコミットの結果が永続化され、停止前に実行された最後のトランザクションのステータスを強制するために、アプリケーションによって使用されます。

デフォルト：FALSE

値：TRUE および FALSE

制限事項：

PI/SQL コール GET_LTXID_OUTCOME を使用するには、COMMIT_OUTCOME 属性が設定されている必要があります。

COMMIT_OUTCOME は、読取りモードの Active Data Guard や読取り専用データベースに対しては効果がありません。Active Data Guard を DML 転送と組み合せてトランザクション・ガードで使用することは、サポートされていません。

COMMIT_OUTCOME は、ユーザー定義のデータベース・サービスに対して許可されています。デフォルトのデータベース・サービスやデフォルトのプラガブル・データベース・サービスへの設定はサポートされていません。

RETENTION_TIMEOUT

説明：RETENTION_TIMEOUT は、COMMIT_OUTCOME とともに使用されます。このパラメータは、COMMIT_OUTCOME が保持される時間を決定します。トランザクション履歴表は小さいため、かなり後に返すセッションで結果を特定できるように、RETENTION_TIMEOUT を大きい値にします。ほとんどの環境では、デフォルト値で十分です。

DBMS_SERVICE、srvctl、または gdsctl を使用して設定

単位：秒

デフォルト：24 時間（86400）

最大値：30 日間（2592000）

サービス設定例

Oracle RAC でサービスを作成および変更する例：

Oracle RAC または Oracle RAC One を使用する場合、srvctl を使用してサービスを作成および変更します。

ポリシーで管理されるサービスの例：

```
srvctl add service -d orcl -s GOLD -g ora.Srvpool -commit_outcome TRUE -retention 86400

srvctl modify service -d orcl -s GOLD -commit_outcome TRUE
```

管理者によって管理されるサービスの例：

```
srvctl add service -d codedb -s GOLD -r serv1 -a serv2 -commit_outcome TRUE 60 –stopoption immediate

srvctl modify service -d orcl -s GOLD -commit_outcome TRUE
```

単一インスタンスのデータベースでサービスを変更する例：

（Oracle RAC や Global Database Services ではなく）単一インスタンスのデータベースを使用する場合、DBMS_SERVICE を使用してサービスを変更します。DBMS_SERVICE を使用して PDB でサービスを作成または変更するには、その PDB に接続する必要があります。

```
declare

params dbms_service.svc_parameter_array;

begin params('COMMIT_OUTCOME'):='true';

params('RETENTION_TIMEOUT'):=86400;

dbms_service.modify_service('[your service]',params);

end;
```

Commit_Outcome を使用するための権限の付与

DBMS_APP_CONT パッケージの権限が、GET_LTXID_OUTCOME をコールするデータベース・ユーザーに付与されていることを確認します。

```
GRANT EXECUTE ON DBMS_APP_CONT TO <user-name>;
```

パフォーマンス

リソース使用率

トランザクション・ガードは、スケーラブルなプロトコルとして設計されています。トランザクション・ガードは必然的に軽量であり、Real Application Clusters 間や単一インスタンスのデータベース上でパーティション化を使用することで、スケーラビリティを発揮します。命令カウントから高い OLTP ワークロードまでを使用したパフォーマンス測定では、測定したすべてのワークロードで、CPU 使用率の増加は 0.05%未満でした。トランザクション・ガードを追加してもそうとは分からず、自社開発ソリューションをトランザクション・ガードに置き換えることでコストが低減されます。

トランザクション・ガードを使用した場合はコード・パスが最適化されるため、FAST_START_MTTR_TARGET を設定した測定では、高い OLTP パフォーマンスのテストで、クライアントの平均経過時間と合計経過時間が同様の値になりました。高い OLTP ワークロードを実行したクライアント・トランザクション平均経過時間では、差は 0.04%未満でした。ベンチマークには AROLTP（eBusiness Suite）を使用し、同時セッション数 500 および 1,000 で、Commit_OUTCOME の False と True を比較しました。

トランザクション履歴表

トランザクション履歴表（LTXID_TRANS）は、データベースの作成時およびアップグレード時にSYSAUX 表領域にデフォルトで作成されます。インスタンスの追加時に、最後のパーティションの記憶域を使用して新規パーティションが追加されます。SYSAUX では通常、自動ストレージ管理（ASSM）が使用されます。

この表領域の場所がパフォーマンスにとって最適でない場合、DBA はパーティションを別の表領域に移動できます。履歴表を移動するには、各パーティションに対して alter table を実行します。

以下に例を示します。

```
alter TABLE LTXID_TRANS move partition LTXID_TRANS_4

tablespace FastPace

storage ( initial 100M next 100M minextents 20 maxextents 121 );
```

トランザクション・ガードを使用したアプリケーション開発

TAF を使用している場合は、「トランザクション・ガードと透過的アプリケーション・フェイルオーバー」に移動してください。トランザクション・ガードを使用するには、以下の手順を実行します。

» ユーザー・セッションを使用できなくしているエラーを捕捉します。ここでもっとも重要な手順は、FAN が DBA によって構成されていることを確認することです。FAN のためのアプリケーション・コードは必要ありません。ただし、アプリケーションが TCP/IP タイムアウトで停止せずにリアルタイムでエラーを受け取ることができるよう、FAN を有効にする必要があります。（12c Release 2 では、高可用性のための推奨 TNS、TRUE に設定された AQ_HA_NOTIFICATION、および OCI の oraccess.xml 内の EVENTS を使用して、JDBC および OCI ドライバ・レベルで FAN が自動的に有効化されています。）

» 未コミットの結果の後で再送信することを計画している場合、エラーがリカバリ可能であれば、再送信が成功する可能性はより高くなります。コミットは、クライアントに返されたエラー・クラスに関係なく完了できるため、コミット結果のエラー・クラスを確認する必要はありません。たとえば、osd エラーを受け取ったからといって、処理がコミットされなかったとは限りません。

» クライアント・ドライバの提供する API（JDBC には getLogicalTransactionId、ODP.NET には LogicalTransactionId、OCI と OCCI には OCI_ATTR_GET と LTXID）を使用して、以前に失敗したセッションの LTXID を取得します。

» 新しいセッションを取得します。この新しいセッションには、固有の論理トランザクション識別子が割り当てられます。

» API から取得した LTXID を使用して、PL/SQL プロシージャ GET_LTXID_OUTCOME を起動します。返される状態によって、最後のトランザクションの COMMITTED が TRUE か FALSE か、USER_CALL_COMPLETED が TRUE か FALSE かがドライバに通知されます。クライアントとデータベースが同期していない場合（たとえば、同じデータベースでない場合や、データベースがリストアされた場合）は、この PL/SQL ファンクションによってエラーが返されます。

» アプリケーションが結果を返し、ユーザーが判断します。一部のアプリケーションは、結果が未コミットの場合に、自身で再実行する場合があります。自身で再実行を行い停止が生じる場合は、再実行セッションの LTXID が、GET_LTXID_OUTCOME ファンクションで使用されます。

トランザクション・ガードの一般的な使用法

12c クライアント・ドライバが FAN の停止イベントまたはエラーを受け取ります（トランザクションは、TMTWOPHASE コミット・フラグを使用して TP 管理されません）。

停止したセッションが FAN によって強制終了されます

例外処理において以下を行います

古いセッションを閉じます

停止したセッションから最後の LTXID を取得します（下の API を参照）

LTXID が null の場合// トランザクション・ガードは無効化され元のエラーをスローします

そうでない場合

// トランザクション・ガードは有効化されます

新しいデータベース・セッションを取得します

最後の LTXID を使用して DBMS_APP_CONT.GET_LTXID_OUTCOME をコールし、COMMITTED および USER_CALL_COMPLETED のステータスを取得します

コミット結果の取得でエラーが返される場合、元のエラーをスローします

そうでない場合

COMMITTED かつ USER_CALL_COMPLETED の場合

アプリケーションはコミットされて安全に返されます

あるいは COMMITTED かつ NOT USER_CALL_COMPLETED の場合

アプリケーションはコミットされた結果を返すことができます。ただし、アプリケーションが OUT バインド、行数、またはコミットで返されなかった句を返す DML などの情報に依存する場合、アプリケーションは続行できないことがあります。ほとんどのアプリケーションはコミットでの結果セットに依存しません。結果セットに依存するそのようなアプリケーションに対しては、完了のステータスが提供されます。

あるいは NOT COMMITTED の場合

元のエラーがリカバリ可能な場合（OCI_ATTRIBUTE。JDBC クライアントの場合は isRecoverable）、アプリケーションは、再送信するか、未コミット・ステータスをユーザーに返すかを決定できます

開発者向けステップ・バイ・ステップのLTXID使用法

再実行や結果を返すために、アプリケーションまたはサード・パーティ・コンテナは、各セッションにおいてサーバーで次にコミットされる LTXID にアクセスする必要があります。リカバリ可能な停止の後で、失敗したセッションの LTXID を取得するには、API（JDBC には getLogicalTransactionId、ODP.NET には LogicalTransactionId、OCI には OCI_ATTR_GET と LTXID）を使用します。

JDBC Thin ドライバからも、サーバーから受け取った各 LTXID 変更に対して実行される、コミット結果コールバックが得られます。イベントから新しい LTXID が得られますが、以前にコミットされ

たかどうかは示されません。サード・パーティ・コンテナは、このコールバックを使用して、コミット結果が失われた場合に再送信などに使用する LTXID を取得できます。http://adc2180604.us.oracle.com/JDBC_Javadoc/MAIN/latest/oracle/jdbc/OracleConnection.html#addLogicalTransa ctionIdEventListener-oracle.jdbc.LogicalTransactionIdEventListener

開発者のための重要なルール

再送信のプロセスでさらに停止が発生する場合、表 1 に示すように、アプリケーションは以前の LTXID ではなく、失敗したときに適用されていた最後の LTXID を GET_LTXID_OUTCOME に使用する必要があります。この表では、LTXID-A、LTXID-B、LTXID-C は異なるセッションの別々の LTXID を示しています。逆に、特定の LTXID のコミット結果を取得するプロセスでさらに停止が発生する場合は、同じ LTXID を使用してその結果を引き続き問い合わせることができます。

表1：状況と、LTXIDを使用した開発者のアクション

状況	アプリケーションのアクション	次に使用するLTXID LTXID変更に対するコールバック：JDBC Thinドライバのみ
アプリケーションがリカバリ可能なエラーを受け取り、GET_LTXID_OUTCOMEをコールしてトランザクション・ステータスを確認します	アプリケーションは、新しい接続を確立し（固有のLTXID-B 0を使用）、最後に失敗したセッションのLTXID（LTXID-A）を使用してGET_LTXID_OUTCOMEをコールします	新規LTXID-B 0 登録時にもJDBCコールバックを使用して設定します
アプリケーションが、最後のセッション・トランザクションのステータスがCOMMITTEDおよびUSER_CALL_COMPLETEDであることを確認します	コミット済みステータスをクライアントに返します。アプリケーションは続行できる場合があります	
アプリケーションが、最後のセッション・トランザクションのステータスがCOMMITTEDおよびNOT USER_CALL_COMPLETEDであることを確認します	コミット済みステータスをクライアントに返して終了します。コール内の処理が完了していないため（OUTバインドや行数が返されなかったなど）、一部のアプリケーションは続行できません。アプリケーションが続行可能かどうかは、アプリケーションによって異なります	
アプリケーションが、最後のセッション・トランザクションのステータスがNOT COMMITTEDであることを確認します	アプリケーションが結果をユーザーに返すか、必要に応じてクリーンアップし、有効な新規セッションのLTXID（LTXID-B 0）を使用して再送信します 新規リクエストによってコミットが実行されると、サーバーはLTXID-B 2（およびそれ以降の番号のLTXID）を使用してコミット・メッセージを返します	新規LTXID-B 2～N 登録時にもJDBCコールバックを使用して設定します
アプリケーションが、別のリカバリ可能なエラーを受け取ります	アプリケーションは、新しい接続を確立し（LTXID-C 0を使用）、最後のセッションのLTXID（LTXID-B N）を使用してGET_LTXID_OUTCOMEをコールします	新規セッションのLTXID-C 0 登録時にもJDBCコールバックを使用して設定します
アプリケーションが、再実行時に別のリカバリ可能なエラーを受け取ります	アプリケーションは、新しい接続を確立し（LTXID-D 0を使用）、最後のセッションのLTXID（LTXID-C N）を使用してGET_LTXID_OUTCOMEを再度コールします	新規セッションのLTXID-D 0 登録時にもJDBCコールバックを使用して設定します

トランザクション・ガードとODP.NET

1) ODP.NET を使用する場合、両方のプロバイダで XA に昇格した後は、LTXID を使用できません。

2) 12.2 の ODP.NET は、可能な場合は常にアプリケーション向けのトランザクション・ガードを処理します。ODP.NET を使用する場合、ODP.NET がアプリケーションの代わりにコミット結果を取得できない場合に限り、LTXID がアプリケーションに公開されます。たとえば、Data Guard への拡張フェイルオーバー時などです。

3) TAF とアプリケーション・コンティニュイティは、アプリケーション向けのトランザクション・ガードを処理します。トランザクション・ガード向けにコーディングする必要はありません。

接続プールでのLTXID使用法

接続プールでは、接続とセッションが事前に確立されて共有されるため、LTXID を管理するために別のユースケースが作成されます。接続プールおよび中間層のもっとも簡単なモデルでは、LTXID はプール・セッションごとに存在します。LTXID は、接続プールからチェックアウトされる際にアプリケーション・リクエストに関連付けられ、プールに再びチェックインされる際にアプリケーション・リクエストとの関連付けが解除されます。チェックアウトとチェックインの間、セッションの LTXID はそのアプリケーション・リクエストによって排他的に保持されます。チェックイン後は、LTXID はアイドル状態のプール・セッションに属し、その接続をチェックアウトする次のアプリケーション・リクエストに関連付けられます。

このモデルでは、次のことがサポートされます。

» 現在の HTTP リクエストに関する重複の検出とフェイルオーバー

» サード・パーティ・コンテナによる基本的な再実行（リカバリ不可能な停止後に、最後のリクエストを再実行する）

透過的アプリケーション・フェイルオーバーとの統合

TAF が有効になっている場合は、トランザクション・ガードをそのまま使用しないでください。TAF が開発者の代わりにトランザクション・ガードを処理します。この動作は、TAF BASIC モードと TAF SELECT モードの両方に適用されます。透過的アプリケーション・フェイルオーバー（TAF）を有効にする場合、または有効にする可能性がある場合、ODP.NET または OCI との併用でトランザクション・ガードを開発する際は、“TAF を使用した ODP.NET”で提供される例と同様のコードを使用する必要があります。

フェイルオーバー時に、12c の透過的アプリケーション・フェイルオーバーは新しい接続を取得し、トランザクション・ガードが有効になっている場合は、トランザクション・ガードを起動してコミット結果を強制します。トランザクション・ガードがコミット済みおよび完了を返した場合、TAF は処理を続行し、アプリケーションにエラーは返されません。トランザクション・ガードが未コミットまたはコミット済みを返したものの、完了を返さなかった場合、TAF はアプリケーションに TAF エラーを返します。TAF は新しい接続を維持します。

TAF とトランザクション・ガードの両方を使用する場合、開発者は TAF エラー・コード（ORA-25402、ORA-25408、ORA-25405）を使用して、トランザクションを確実に再送信するか、あるいはユーザーにコミットされていないことを示すメッセージを返すかを決定できます。TAF とトランザクション・ガードの両方が有効になっている場合にのみ、これらのエラー・コードに基づき安全

に再送信する、または未コミットを返すことができます。TAF のみが有効になっている場合は、再送信する、または未コミットを返すのは、安全ではありません。ODP.NET のコード・サンプルで例を示します。

再送信するには、適切な環境が TAF コールバックに確立されており、トランザクション全体がロールバックおよび再送信される必要があります。以降の例に示すように、ブール変数を使用して、トランザクション・ガードが有効になっているかどうかを追跡します。この変数は TAF コールバックで評価する必要があります。アプリケーション・コンティニュイティを使用すると、これらすべてに加えて追加の操作がアプリケーションに対して実行され、追加のコードも必要ありません。アプリケーション・コンティニュイティには TAF が不要であり、トランザクション・ガードの明示的なコーディングや失われた処理の再送信も不要です。

注：TAF は、セッション・エラーでは起動されません（これには、オペレーティング・システム・レベルの'kill -9'、つまり alter system kill session と、タイムアウトが含まれます）。TAF は INSTANCE エラーと FAN NODE DOWN イベント、およびシャットダウン・トランザクションと切断の POST_TRANSACTION で起動されます。

XA 1フェーズ最適化のコミット結果の向上

トランザクション処理モニター（TPM）は、リソース・マネージャへの接続が失われた場合や曖昧なエラーが返された場合に、1 フェーズ最適化（TMONEPHASE フラグ）を使用してコミット操作の結果を判断することはできません。12c Release 2 以降のトランザクション・ガードと TPM を併用することで、より良い判断が可能です。

1 フェーズ・コミット操作の結果が曖昧である場合、またはコミットの発行後に接続が失われた場合、TPM はオラクル・リソースに対して新たな接続を確立し、GET_LTXID_OUTCOME を使用して確実な結果を提供します。

コミットが発行されていないと、トランザクションはロールバックされるため、TPM はロールバックを返します。

コミットが発行され、曖昧な結果が返されている場合、エラーがリカバリ可能であれば、TPM はトランザクション・ガードを使用してコミット結果を判断できます。

» COMMITTED の場合は、COMMITTED を返します。

» UNCOMMITTED の場合は、新しい接続を借りて COMMIT を再発行します。元の LTXID は、GET_LTXID_OUTCOME をコールすることでブロックされます。

トランザクション・ガードを使用したトランザクション・マネージャの例

あるアプリケーションが、XA データソースとともにトランザクション・マネージャを使用しています。XA データソースでは、XA はローカル・コミット、DDL または DCL、PL/SQL の埋込みコミット、リモート・プロシージャ・コールのリモート・コミットをサポートしません。

このアプリケーションは、サーブレットまたは EJB をトランザクション・マネージャに送信し、1 つの XA トランザクションと一緒に実行します。トランザクション・マネージャは、このサーブレットまたは EJB に加えて、再実行が必要な場合に備えて、中間層での初期状態を取得します。トランザクション・マネージャは、このアプリケーション・コードの実行を開始します。状態は正しくなければならないため、部分的な再実行は許可されません。コミットが成功すると、再実行は無効になります。リカバリ可能なエラーが発生すると、トランザクション・マネージャは、エラーがコミット処理の内部または外部で発生したか、コミットが行われたかどうかを次のように把握します。

1) トランザクション・マネージャによってコミット処理は行われず、XA データソースである。

トランザクション・マネージャは初期状態をリストアし、コールバックをコールしてアプリケーション状態をリセットし、リクエストを再送信します。中間層またはデータベースのいずれかで、副作用が繰り返される場合があります。

2) トランザクション・マネージャによるコミット処理が進行中であり、コミットは 2 フェーズである。

トランザクション・マネージャは 2 フェーズ・コミット処理を管理します。曖昧なリプライを受信した場合、トランザクション・マネージャは xa_prepare リクエストを繰り返すことができます。

トランザクション・マネージャは、すべての参加者がコミットの準備ができていると判断した場合、リプライに失敗した、または曖昧なエラーでリプライした参加者に対して、xa_commit を繰り返すことができます。同様に、ロールバックが確定された結果であり、任意の参加者が xa_rollback からリカバリ可能なエラーを戻した場合、xa_rollback を繰り返すことができます。

3) トランザクション・マネージャによるコミット処理が進行中であり、コミット操作は 1 フェーズである。

1 フェーズ・コミット処理は、データベースによって判断されます。単一ブランチ最適化または読取り専用最適化が適用されるため、TM は 1 フェーズのコミットをリクエストします。TM はフラグ TMONEPHASE を渡します。曖昧なエラーが返された場合、トランザクション・マネージャは新しいデータベース・セッションを取得し、コミットに使用されたセッションと全く同じセッションから取得した LTXID を使用して、その LTXID に基づく信頼性のあるコミット結果を取得します。

トランザクション・ガードが未コミットを返した場合、トランザクション・マネージャは再送信できます。

トランザクション・ガードがコミット済みと完了の両方を返し、コミットがサーブレットに対する最後のコールであった場合、トランザクション・マネージャはこの結果を返すことができます。

トランザクション・ガードがコミット済みを返したものの、完了を返さず、コミットがサーブレットに対する最後のコールであった場合、これもアプリケーションに示すことができます。

4) トランザクション・マネージャによるコミット処理は正常に終了し、サーブレット・コードがさらに存在する。

コミットが実行されると、このコミット後の再実行はサポートされません。トランザクション・マネージャは、コミットされたトランザクションを再実行しません。サーブレットまたは EJB は実行の最初の部分で状態を構築しており、これを最初のコミット後に復活させて継続させることはできません。通常のコミットが最後の操作です。

トランザクション・ガード開発における追加要件

トランザクション・ガードは、開発者が使用するツールであり、エラーやタイムアウト後に信頼性のあるコミット結果を提供します。最後のセッションが停止していることを示すエラーやタイムアウトが返された場合にのみサポートされます。

以下の場合はトランザクション・ガード API を使用しないでください。使用した場合はエラーがスローされます。

- LTXID を取得して例外処理の外部で保持しないでください。つまり、LTXID を取得して後で使用するということはしないでください。
- 現在のセッションで、現在のセッションの LTXID とともに GET_LTXID_OUTCOME を使用しないでください。エラーが返されます。LTXID の目的は、自身のセッションではなく、停止したセッションの結果を見つけることです。
- リカバリ可能なエラーを受信していないセッションに対して、GET_LTXID_OUTCOME を使用しないでください。そのセッションがコミットを実行できなくなります。
- 異なるユーザーまたは異なるデータベースの LTXID とともに GET_LTXID_OUTCOME を使用しないでください。エラーが返されます。
- 例外処理の LTXID を保存しないでください。GET_LTXID_OUTCOME は、最後に開かれていた、または完了したサブセッションに対してのみ有効です。同じセッションの以前のトランザクションとともに使用した場合、エラーが返されます。
- アプリケーションが TAF を使用している場合は、トランザクション・ガードをコーディングしないでください。代わりに新しい TAF エラー・コードを使用して結果を返します。

結論

トランザクション・ガードを使用しなければ、トランザクションが開始され、コミットが発行されている場合に、クライアントに返されるコミット・メッセージは永続的ではありません。クライアントは、トランザクションがコミットされたかどうかがわからない状態になります。非トランザクション状態が正確でない場合や、トランザクションがすでにコミットされている場合に、トランザクションを正しく再送信できません。コミットや完了に関する信頼できる情報がなければ、再送信によって、トランザクションが複数回適用される、あるいは正しくない順序や不適切な状態で適用される可能性があります。

トランザクション・ガードによって、ユーザーのフラストレーション、カスタマー・サポートへの問い合わせ、機会損失の原因となる曖昧なエラーのコストがかからなくなります。トランザクション・ガードは、自社開発ソリューションと比べて少ないオーバーヘッドで優れた安全性とパフォーマンスを実現し、既知の結果を導き出します。

JDBCでのトランザクション・ガードのコード・サンプル

このユースケースは、Jean de Lavarene が作成したものです。
https://blogs.oracle.com/dev2dev/entry/write_recovery_code_with_transaction

この簡単なリクエストでは、リカバリ可能なエラーが発生した場合に、曖昧な結果が返される可能性があります。返されたメッセージが失われる一方で、このリクエストに埋め込まれたコミットそのものは成功する可能性があります。

```
void giveRaiseToAllEmployees(Connection conn, int percentage) throws SQLException {
  Statement stmt = null;
try {
stmt = conn.createStatement();
stmt.executeUpdate("UPDATE emp SET sal=sal+(sal*"+percentage+"/100)");
  } catch (SQLException sqle ) {
    throw sqle;
  }
  finally {
    if(stmt != null)
  stmt.close();
  }
  // リクエストの最後で変更をコミット
  conn.commit(); // コミットは成功する可能性があるものの、コミット結果は失われる
}
```

次に、エラーがリカバリ不可能であり、コミットが成功しなかったことがトランザクション・ガードによって報告された場合に再試行する例外ブロックを追加します。この例では、以前の試行に対して getTransactionOutcome が TRUE を返す場合は、以前のトランザクションが正常にコミットされたことが Oracle Database によって保証されるため、アプリケーションによる再試行は不要です。これに対して、getTransactionOutcome が FALSE を返す場合は、以前の試行はコミットされておらず、今後もコミットされないことが Oracle Database によって保証されます。そのため、再試行しても問題ありません。

```
Connection jdbcConnection = getConnection();
boolean isJobDone = false;
while(!isJobDone) {
  try {
  // 昇給を適用（DML + コミット）
  giveRaiseToAllEmployees(jdbcConnection,5);
  // 例外はなく、プロシージャは完了
  isJobDone = true;
  } catch (SQLRecoverableException recoverableException) {
  // エラーがリカバリ可能であった場合のみ再試行
  try {
    jdbcConnection.close(); // 古い接続を閉じる
  } catch (Exception ex) {} // 他の例外を経験
```

```
Connection newJDBCConnection = getConnection(); // 再接続して再試行を許可
      // トランザクション・ガードを使用して最後のリクエスト（コミット済みまたは未コミット）
      を強制 LogicalTransactionId ltxid
      = ((OracleConnection)jdbcConnection).getLogicalTransactionId();
      isJobDone = getTransactionOutcome(newJDBCConnection, ltxid);
      jdbcConnection = newJDBCConnection;
    }
  }

  /**
   * GET_LTXID_OUTCOME_WRAPPER は DBMS_APP_CONT.GET_LTXID_OUTCOME のラッパー
   */
  private static final String GET_LTXID_OUTCOME_WRAPPER = "DECLARE PROCEDURE
    GET_LTXID_OUTCOME_WRAPPER("+
   "  ltxid IN RAW,"+
   "  is_committed OUT NUMBER ) "+ "IS " +
   "  call_completed BOOLEAN; "+ " committed BOOLEAN; "+ "BEGIN "+
   "  DBMS_APP_CONT.GET_LTXID_OUTCOME(ltxid, committed, call_completed); "+ " if
   committed then is_committed := 1; else is_committed := 0; end if; "+
   "END; "+
   "BEGIN GET_LTXID_OUTCOME_WRAPPER(?,?); END;";



  /**
   * getTransactionOutcomeはLTXIDがコミット済みの場合はTRUEを返し、そうでない場合は
     FALSEを返す。
   * この特定のバージョンは、ユーザー・コールの完了を考慮しないことに注意。
   */
  boolean getTransactionOutcome(Connection conn, LogicalTransactionId ltxid) throws
     SQLException {
    boolean committed = false; CallableStatement cstmt = null; try {
     cstmt = conn.prepareCall(GET_LTXID_OUTCOME_WRAPPER); cstmt.setObject(1, ltxid); //
     12.1.0.2 以降はこれを使用 cstmt.registerOutParameter(2, OracleTypes.BIT); cstmt.execute();
     committed = cstmt.getBoolean(2);
    }
    catch (SQLException sqlexc) { throw sqlexc;
    }
    finally {
     if(cstmt != null) cstmt.close();
    }
    return committed;
  }
```

ODP.NET でのトランザクション・ガードのコード・サンプル（TAFを使用しない場合）

```
========================= start ========================
using System;
using Oracle.DataAccess.Client; // ODP.NET 管理対象外ドライバ向け
// または、ODP.NET 管理対象ドライバ向けの“Oracle.ManagedDataAccess.Client;”を使用

class TransactionGuardSample
{
   static void Main()
   {
      bool bReadyToCommit = false;
      string constr = "user id=hr;password=hr;data source=oracle"; OracleConnection con = new
      OracleConnection(constr); OracleTransaction txn = null;
      OracleCommand cmd = null;

      try
      {
         string sql = " update employees set salary=10000 where employee_id=103"; con.Open();
         txn = con.BeginTransaction();
         cmd = new OracleCommand(con, sql); cmd.ExecuteNonQuery(); bReadyToCommit = true;
      }
      catch (Exception ex)
      {
         // SQL 実行が成功していないため、ここでロールバック txn.Rollback();
         Console.WriteLine(ex.ToString());
      }

      try // TX が成功しているため、ここに進む
      {
         if (bReadyToCommit) txn.Commit();
      }
      catch (Exception ex)
      {
         if (ex is OracleException)
         {
   // エラーがリカバリ可能であり、トランザクションがコミットされておらず、トランザクシ
   ョン・ガードが有効な場合、処理を再送信しても安全
            if (ex.IsRecoverable && ex.OracleLogicalTransaction != null &&
            !ex.OracleLogicalTransaction.Committed)
            {
               // 処理を再送信しても安全
            }
```

```
            else
            {
               // 処理を再送信してはならない
            }
        }     finally
        {
          // すべてのオブジェクトを破棄
          txn.Dispose(); cmd.Dispose();
          con.Dispose(); // 接続プールに接続を戻す
        }
    }
}
```

ODP.NET でのトランザクション・ガードのコード・サンプル（TAFを使用する場合）

```
// 注：ユーザーには、SYS.DBMS_APP_CONT の実行アクセスが必要
// 注：データベース・サービスには、commit_outcome = true および FAILOVER_TYPE = BASIC または SELECT が設定されていなければならない
========================= start =========================
// コード・サンプルでは、ODP.NET 12.1.0.2 以降を使用する必要がある using System;
using Oracle.DataAccess.Client;
// もしくは、ODP.NET 管理対象ドライバ向けの“Oracle.ManagedDataAccess.Client;”を使用（Oracle Data Access Components 12c Release 4 以降で使用可能）

class TransactionGuardSample
{
   static void Main()
   {
      bool bReadyToCommit = false;

      // このコードは、トランザクション・ガードが有効かどうかを検査するために必要

      bool bTGEnabled = false;

      string constr = "user id=hr;password=hr;data source=oracle"; OracleConnection con = new OracleConnection(constr); OracleTransaction txn = null;
      OracleCommand cmd = null;

      try
      {
         string sql = " update employees set salary=10000 where employee_id=103"; con.Open();

           // この bTGEnabled ブール変数は、TG が有効かどうかを確認するために必要
           // 接続後直ちに、および再び TAF コールバックでこの変数を設定
           bTGEnabled = con.OracleLogicalTransaction != null;
         txn = con.BeginTransaction();
         cmd = new OracleCommand(con, sql); cmd.ExecuteNonQuery(); bReadyToCommit = true;
      }
      catch (Exception ex)
      {
         // SQL 実行が成功していないため、ここでロールバック txn.Rollback();
         Console.WriteLine(ex.ToString());
      }

      try // TX が成功しているため、ここに進む
      {
```

```
        if (bReadyToCommit) txn.Commit();
      }
      catch (Exception ex)
      {
        // 一覧の TAF エラーのみを、トランザクション・ガードが有効な場合に限り処理

          if (bTGEnabled && (ex.Number == 25402 || ex.Number == 25408 || ex.Number == 25405
          ))
        {
            // アプリケーションはクリーンアップし、その後ロールバックして現在のトランザ
            クションを再送信する場合がある
        }
        else
        {
            // エラーを以前のように処理。再送信してはならない
        }
      }
      finally
      {
        // すべてのオブジェクトを破棄 txn.Dispose(); cmd.Dispose();
        con.Dispose(); // 接続プールに接続を戻す
      }
    }
}
```

OCCI/OCIでのトランザクション・ガードのコード・サンプル（TAFが有効でない場合）

OCI の完全なデモは OTN に掲載されています。tgdemo.c を参照してください。次のコード・サンプルは、エラー処理でコールされます。

```
static void checkTransOutcome(
                        OCIEnv *envhp,
                        OCISvcCtx *svchp,
                        OCIError *tmpErrhp,
                        OraText *poolName,
                        ub4      poolNameLen,
                        boolean *committed,
                        boolean *callComplete)
  {
    OCISession *embUsrhp;     /* サービス・コンテキストに埋め込まれたセッション・ハンドル
    */ub1    *ltxidPtr;             /* 埋込みセッションの LTXID */
    ub4       ltxidLen;                    /* LTXID の長さ */

    OCISvcCtx *newSvchp = (OCISvcCtx *)0; /* get_ltxid_outcome をコールするために使用
    */OCIStmt *getLtxidStm = (OCIStmt *)0; /* get_ltxid_outcome をコールするために使用
    */OCIBind *bnd1p, *bnd2p, *bnd3p;
    boolean    cmtd = FALSE, compl = FALSE;

    /* get_ltxid_outcome をコールしてトランザクション・ステータスを判断できるよう、
     * セッションのLTXIDを取得。get_ltxid_outcomeに渡されるLTXID
     * パラメータは、エラーに遭遇したセッションと関連していなければ
     * ならない。
     */

    /* 最初に、コール元のサービス・コンテキストに埋め込まれたセッション・ハンドルを取得
    */(void) OCIAttrGet(svchp, OCI_HTYPE_SVCCTX,
                (dvoid *)&embUsrhp, (ub4 *)0, (ub4)OCI_ATTR_SESSION, tmpErrhp);

    /* 次に、そのセッションの LTXID に対するポインタを取得 */
    (void) OCIAttrGet(embUsrhp, OCI_HTYPE_SESSION, (dvoid *)&ltxidPtr,(ub4 *)&ltxidLen,
                (ub4)OCI_ATTR_LTXID, tmpErrhp);

    /* 元のセッションはリカバリ可能なエラーを受信したため、get_ltxid_out を
     * コールするために使用できない。代わりに、プールから新しいセッションを取得。
    if (checkerr(tmpErrhp, OCISessionGet(envhp, tmpErrhp, &newSvchp,
                            (OCIAuthInfo *)0,
                            (OraText *)poolName, (ub4)poolNameLen, NULL, 0, NULL, NULL, NULL,
                            OCI_SESSGET_SPOOL)))
    {
```

```
  printf("OCISessionGet failed to get a session from the pool.\n"); goto
  done;
}

/* get_ltxid_id コールを解析 */
if (checkerr(tmpErrhp,OCIStmtPrepare2(newSvchp,&getLtxidStm, tmpErrhp, (CONST OraText
                              *)getLtxid,
                              (ub4)sizeof(getLtxid), (const oratext *)0, (ub4)0,
                              OCI_NTV_SYNTAX, OCI_DEFAULT)))
{
  printf("OCIStmtPrepare2 failed.\n");
  goto done;
}

/* get_ltxid_outcome は次の 3 つのバインドを受け取る
 *  1.エラーに遭遇したセッションのLTXID
 *  2. コミット済み：最後のコールはコミットされたか
 *  3. 完了：最後のコールは完了したか
 */
if (checkerr(tmpErrhp, OCIBindByPos(getLtxidStm, &bnd1p, tmpErrhp, 1,
                              (dvoid *)ltxidPtr, (sword)ltxidLen, SQLT_BIN, (dvoid *)0,
                              (ub2 *) 0, (ub2 *) 0, (ub4) 0, (ub4 *) 0,
                              OCI_DEFAULT)) ||

    checkerr(tmpErrhp, OCIBindByPos(getLtxidStm, &bnd2p, tmpErrhp, 2, (dvoid *) &cmtd,
                              sizeof(cmtd), SQLT_BOL, (dvoid *)0,
                              (ub2 *) 0, (ub2 *) 0, (ub4) 0, (ub4 *) 0,

                              OCI_DEFAULT)) ||

    checkerr(tmpErrhp, OCIBindByPos(getLtxidStm, &bnd3p, tmpErrhp, 3, (dvoid *) &compl,
                              (sword)sizeof(compl), SQLT_BOL, (dvoid *)0,
                              (ub2 *) 0, (ub2 *) 0, (ub4) 0, (ub4 *) 0, OCI_DEFAULT)))

{
  printf("Failed to bind variables for get_ltxid_outcome\n");
  goto done;
}
if (checkerr(tmpErrhp,OCIStmtExecute(newSvchp, getLtxidStm, tmpErrhp,
                              (ub4)1, (ub4)0,
                              (OCISnapshot *)0, (OCISnapshot *)0, OCI_DEFAULT )))
{
  printf("Failed to execute get_ltxid_outcome\n"); goto done;
}
```

done:

```
/* 最後のコールのステータスに基づき、何を行うかを判断
 * 1.未コミット：
 *     処理中のトランザクションがコミットされておらず、今後もコミット
 *     されない。最後のトランザクションを再送信しても問題ない。
 * 2.コミット済みおよび未完了：
 *     たとえば、コミットを含むPL/SQLプロシージャの実行中に
 *     障害が発生した場合などに起きる可能性がある。
 *     コール元は、継続しても問題ないかどうかを判断する
 *     必要がある。
 * 3.コミット済みおよび完了：
 *     エラーを報告しない。アプリケーションは続行可能。
 */
if (!cmtd)
{                                       /* ケース 1 */
  printf("Recoverable error occurred; transaction re-execution is safe\n");
}
else if (cmtd && !compl)
{                                       /* ケース 2 */
  printf("Warning:Transaction committed but call not complete\n");
}
/* ケース 3（コミット済みおよび完了）：エラーや警告を報告しない */
/* リソースを解放 */if (newSvchp)
  (void) OCISessionRelease(newSvchp, tmpErrhp, NULL, 0, OCI_DEFAULT); if (getLtxidStm)
  (void) OCIStmtRelease((dvoid *) getLtxidStm, tmpErrhp, (void *)0, 0, OCI_DEFAULT);
/* 最後に OUT 値を更新 */
*committed = cmtd;
*callComplete = compl;
```

付録

トランザクション・ガードのおもな機能

トランザクション・ガードは、アプリケーションが最小限の操作でスケーラブルに冪等性（べきとうせい）を自動的かつ透過的に実現するために使用する、統合型ツールです。

トランザクション・ガードのおもな機能は以下のとおりです。

» Oracle Database 12c でサポートされているすべてのトランザクション・タイプに対して、コミット結果の永続性が確保されます。これには、自動コミットを使用して実行したトランザクション、PL/SQL 内からコミットされたトランザクション、分散トランザクションまたはリモート・トランザクション、1 フェーズ XA トランザクション、および汎用的な方法では識別できない、リモート・コールアウトで実行されたトランザクションが含まれます。

» 以前に実行中であった処理のコミットをブロックすることによって、コミット結果を保証します。これにより、その LTXID によって保護されている同じトランザクションの別の送信がコミットできなくなります。このアプローチはスケーラビリティを持ち、鍵に依存する外部のアプローチを使用した場合に発生する停止を回避します。

» セッションで最後に実行中であったトランザクションのコミット結果を取得するために、繰り返しの試行が受け入れられます。すべての試行で同じ結果が得られます。

» 同じ LTXID で識別される実行中トランザクションのコピーが複数存在する場合に、データベース・トランザクションが重複することがないように、最大 1 回の実行セマンティックがサポートされます。

» コミット済みの処理が、トップレベルのコール（クライアントからサーバー）の一部としてコミットされたのか、サーバーの PL/SQL などのプロシージャに埋め込まれていたのかを特定します。埋込みのコミット状態は、コミットは完了しているものの、コミットが実行されたプロシージャ全体がまだ完了まで実行されていないことを示します。そのプロシージャ自体がデータベース・エンジンに戻るまで、コミット以降の処理が完了したことを保証できません。たとえば、1 つのラウンド・トリップに複数のコミット操作が含まれており（OLTP アプリケーションには適していない）、1 つのコミットが実行されたものの、すべてのコミットが実行されたわけではないとします。この例では、COMMITTED は true になり、USER_CALL_COMPLETED は false になります。

» コミット解決の対象となるデータベースが元のユーザー送信より進んでいる、同期が取れている、または遅れているかどうかを識別し、クライアントからのトランザクションの送信順序にギャップがある場合は拒否します。従属のトランザクションが失われている可能性があるため、サーバーまたはクライアントがサーバーと同期されていない場合は、結果を取得しようとすると、エラーとみなされます。

» クライアントからサーバーへのコールによって LTXID が増加されると、JDBC Thin クライアント・ドライバでコールバックが起動されます。このコールバックは、WebLogic Server やサード・パーティ Java クライアントなどの上位レイヤーのアプリケーションが、すぐに使用可能な現在の LTXID を必要に応じて保持するために使用できます。

» Oracle Multitenant 12c インフラストラクチャに統合されるデータベース全体にわたって、サービス名が一意になります。Multitenant では、トランザクション・ガードはプラガブル・データベース・レベルで機能します。テナントごとに一連の LTXID 構造が存在するため、アンプラグや再配置を透過的に実行できます。

トランザクション・ガードのプロトコル

論理トランザクション識別子（LTXID）の説明

論理トランザクション識別子（LTXID）は、セッション確立時に自動的に割り当てられます。スケーラビリティを確保するために、データベースのトランザクションがそのデータベース・セッションでコミットされても、LTXID そのものは変わりません。トランザクション・ガードのプロトコルにより、次のことが保証されます。

各論理トランザクション識別子の実行が一意になります。

保存期間中に、サポートされているすべてのコミット・ポイントについて重複を検出します。

結果を取得する際に LTXID をブロックすることで、その LTXID を使用している、以前に実行中であったバージョンのトランザクションがコミットできなくなります。

データベースが 12c でない場合や、トランザクション・ガードが無効になっている場合、LTXID は null になります。

最大 1 回の実行

トランザクション・ガードを使用する場合、トランザクションの重複を回避するために LTXID が使用されます。LTXID はコミットで変更され、ロールバック後に再利用されます。通常の実行時に、論理トランザクション識別子（LTXID）は、データベース・トランザクションごとに、クライアントとサーバーの両方でセッション内に自動的に保持されます。コミット時に、論理トランザクション識別子はトランザクションのコミットの一環として変更されます。

最大 1 回のプロトコルでは、再試行のために同意した保存期間中は、RDBMS が LTXID を保持することが必要になります。デフォルトの保存期間は 24 時間ですが、必要に応じて、数日もしくはそれ以上に延長するよう指定することも可能です。保存期間が長くなるほど、最大 1 回のチェックが継続する期間、つまり同じ LTXID を使用した古いトランザクションのコミットがブロックされる期間が長くなります。この設定はサービスごとに可能であり、変更することもできます。Data Guard、Active Data Guard、または PDB のアンプラグ/プラグ（クローンなし）を使用する場合や、PDB オンライン再配置を使用する場合など、複数のデータベース物理コピーが関連する場合は、論理トランザクション識別子が各データベースにレプリケートされます。トランザクション・ガードを Golden Gate やロジカル・スタンバイ・サイトで使用することや、他のサード・パーティ・レプリケーション・テクノロジーと併用することは、現在サポートされていません。各サイト内でのトランザクション・ガードの使用がサポートされています。

12c Oracle JDBC Thin ドライバ向けに提供されている、OCI、OCCI、および ODP.Net クライアント向けと同様の API getLogicalTransactionId を使用すると、そのセッションの次のデータベース・トランザクションに使用される次の論理トランザクション識別子をアプリケーションが取得できます。

PL/SQL プロシージャ DBMS_APP_CONT.GET_LTXID_OUTCOME を使用すると、アプリケーションは

現在の論理トランザクション識別子を使用して、最後に実行中であったトランザクションの結果を特定できます。GET_LTXID_OUTCOMEをコールすると、データベースによってLTXIDのコミットがブロックされるため、結果が判明する場合があります。これは、結果の強制とも呼ばれます。トランザクション・ガードを使用するアプリケーションは、エラーまたはタイムアウトの後で LTXID を取得します。その後、GET_LTXID_OUTCOME をコールした後に、再実行を試みるか、結果をユーザーに返すよう試みます。

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

海外からのお問い合わせ窓口
電話：**+1.650.506.7000**
ファクシミリ：**+1.650.506.7200**

ORACLE

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Integrated Cloud Applications & Platform Services

Oracle Database 12c R2 でのトランザクション・ガード 2016 年 8 月
著者：Carol Colrain
共著者：Stefan Roesch、Jean de Lavarene、Kiminari Akiyama、Nancy Ikeda