2024 年 9 月

MongoDB マルチドキュメント ACID トランザクション

MongoDB®

目次

はじめに

MongoDB は、2018 年の MongoDB 4.0 リリースで、マルチドキュメント ACID トランザクションのサポートを開始し、2019 年の 4.2 リリースでシャードクラスタに対応した分散型トランザクション機能を追加し、サポート範囲を拡張しました。

MongoDB のシングルドキュメントに対するアトミックな操作は、トランザクションセマンティクスを提供し、多くのアプリケーションが必要とするデータ整合性の要件をこれまでも満たしてきました。これに加えて、マルチドキュメント ACID トランザクションのサポートが追加されたことで、開発者は、MongoDB でより幅広い領域のユースケースに対応できるようになりました。スナップショットの分離と、「完全に実行されるか、全く実行されないか」を保証する処理により、広く分散したシャードクラスタの環境でも、アプリケーションはトランザクションデータの整合性を維持できます。

MongoDB のマルチドキュメントトランザクションの機能は、リレーショナルデータベースでのトランザクション開発経験があれば、必要なアプリケーションに容易に組み込み、すぐに利用できます。

このホワイトペーパーでは、MongoDB がマルチドキュメント ACID トランザクションをサポートするようになった理由、その設計目標と達成状況、開発者向けのベストプラクティス、この機能を実現するために 5 年以上にわたって行われたエンジニアリング投資について説明します。フルマネージドでオンデマンドの MongoDB Atlas クラスタを利用するか、MongoDB Atlas クラスタをダウンロードしてオンプレミスのインフラで実行することで、すぐに MongoDB を使い始めることができます。

「ACID 特性を持つトランザクションは、特に、コマース領域のデータ処理を中心とする分野で、ビジネスクリティカルなトランザクションシステムを実現するのに不可欠な要素の 1 つです。NoSQL の処理能力を有し、ACID トランザクションをコレクション横断でサポートできるデータベースは、MongoDB 以外にありません。この組み合わせにより、MongoDB の性能を活用したミッションクリティカルなアプリケーションの開発が容易になります」

シスコシステムズ eコマース部門ディレクター Dharmesh Panchmatia 氏

マルチドキュメント ACID トランザクションの重要性

MongoDB は、2009 年に初めてリリースされて以来、データベース設計の新たなアプローチを中心としたイノベーションを継続的に創出し、従来のリレーショナルデータベースが持つ開発上の制約を解消し続けてきました。

MongoDB は、リッチで柔軟な自然言語のドキュメントを慣用的なプログラミング言語の API からアクセスすることを前提に設計されているため、アプリ開発のスピードがこれまでの 2〜3 倍に向上します。また、分散型のシステムアーキテクチャを採用しており、従来よりも多くのデータを処理できるほか、ユーザーのニーズに応じたデータの配置や常時可用性が特長です。このようなアプローチにより、あらゆる業界で、強力で洗練されたアプリケーションの作成が可能になり、幅広いモダンユースケースに対応できるようになりました。

サブドキュメントや配列を利用することで、均一な行と列から構成された相互に関連性のある別々のテーブルにデータを分割するのではなく、ドキュメントは関連性のあるデータを単一のデータ構造で自然かつ詳細にモデル化できます。この結果、MongoDB の単一ドキュメントによる原子性の保証でも、ほとんどのアプリケーションで求められるデータの整合性のニーズを満たすことが可能です。ドキュメントモデルの充実した機能を活用することで、推定で 80% から 90% のアプリケーションには、マルチドキュメントトランザクションの必要はありません。

図 1： MongoDB を導入してイノベーションを起こしている企業の例

しかし、一部の開発者や DBA は、40 年にわたるリレーショナルデータモデリングの利用経験を基に、データモデルに関係なく、マルチレコードトランザクションがデータベースに必須であると考えています。また、現時点ではマルチドキュメントトランザクションが必要ないアプリケーションも、将来的には必要になることを懸念している開発者もいます。さらに、一部のワークロードでは、複数のレコードにまたがって ACID 特性を持つトランザクションをサポートする必要があります。

このような状況を踏まえ、MongoDB ではマルチドキュメントトランザクションのサポートを開始しました。これにより、MongoDB は、これまで以上に幅広いユースケースに対応可能になりました。また、アプリケーションの機能を必要に応じて強化できるため、開発者は、将来的なニーズにもデータベースが対応できるという安心感が得られます。

データモデルとトランザクション

MongoDB におけるマルチドキュメントトランザクションの処理について触れる前に、まずは、データベースで使用するデータモデルがトランザクションのスコープにどのように影響するかについて考察します。

リレーショナルデータモデル

リレーショナルデータベースでは、1 つのエンティティのデータを、複数のレコードと、親子関係を持つ複数のテーブルに分割してモデル化します。そのため、トランザクションでは、これらのレコードやテーブルがそのトランザクションで扱う対象となるよう、スコープを設定する必要があります。図 2 の例は、顧客データベースの連絡先情報をリレーショナルスキーマでモデル化したときの例を表しています。データは複数のテーブル全体にわたって正規化されており、顧客名や、住所、市区町村名、国名、電話番号、関連事項、顧客の関心事項などに関するテーブルがあります。

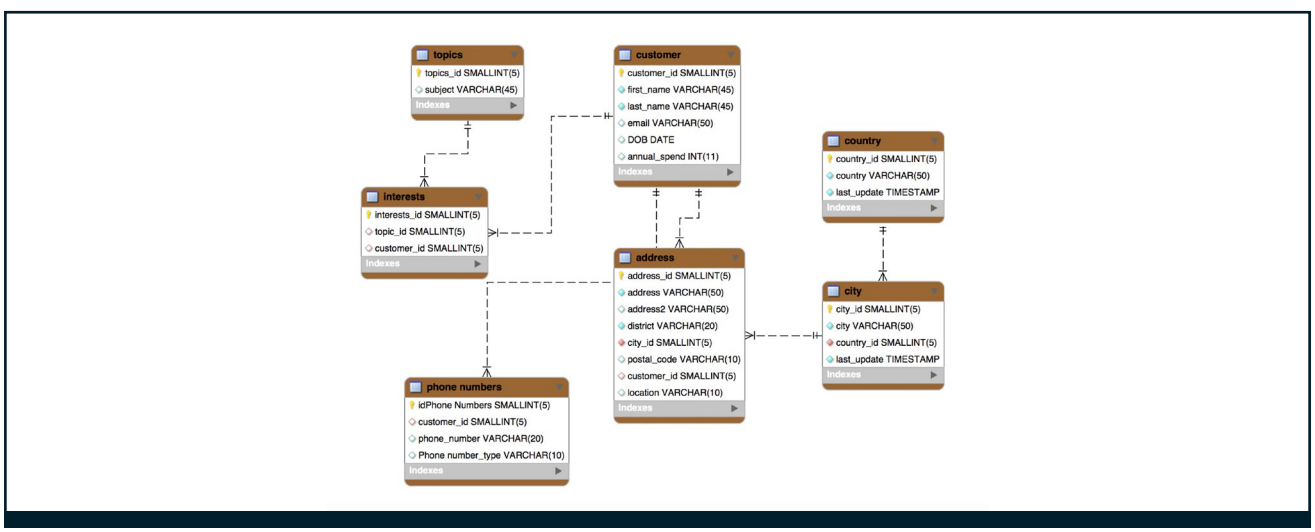

図 2： リレーショナルデータベースでは、複数の独立したテーブルに顧客データを分割してモデル化する

連絡先情報に記載された顧客の勤務先が変わるなど、何らかのかたちで顧客データに変更が生じた場合、複数のテーブルを更新する必要があります。あらゆるテーブルは、「完全に実行されるか、全く実行されない」トランザクションの形式で更新されなければなりません。このプロセスを図 3 で示しています。

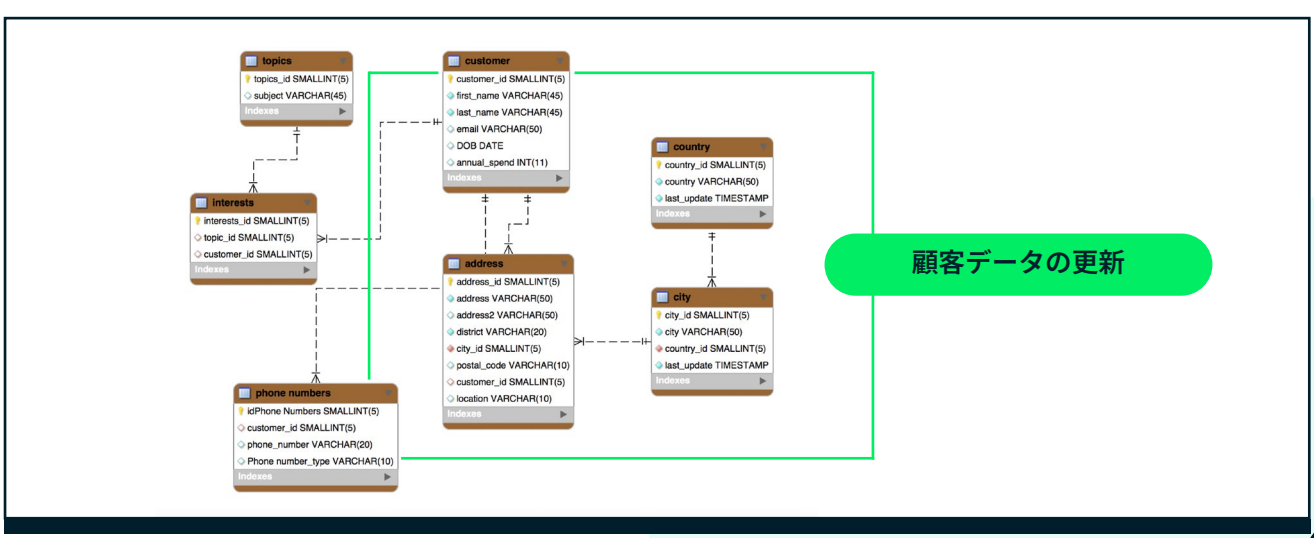

図 3： リレーショナルデータベースでの顧客データの更新

ドキュメントデータモデル

一方、ドキュメントデータベースの場合は、方法が全く異なります。関連性のあるデータを分割して親子関係のある複数のテーブルに分散させるのではなく、図 4 に示すように、ドキュメントを用いて、サブドキュメントや配列を含むリッチな型付き階層構造に、相互に関連性を持つデータをまとめて保存します。

図 4：単一のリッチドキュメント構造でモデル化された顧客データ

MongoDB の場合、既存のトランザクションプロパティは、ドキュメントのレベルに合わせて常にスコープが調整された状態になります。図 5 に示すように、1 つまたは複数のフィールドを 1 回の操作でアトミックに書き込み、複数のサブドキュメントやネストされた配列を含む任意の配列の要素を更新します。MongoDB では、ドキュメントの更新時に独立性が保証され、エラーが発生しても、処理はロールバックされるため、ドキュメントは一貫性のある状態に維持されます。このような設計により、アプリケーションのオーナーは、従来のリレーショナルデータベースと同じレベルで、データの整合性の保証が得られます。

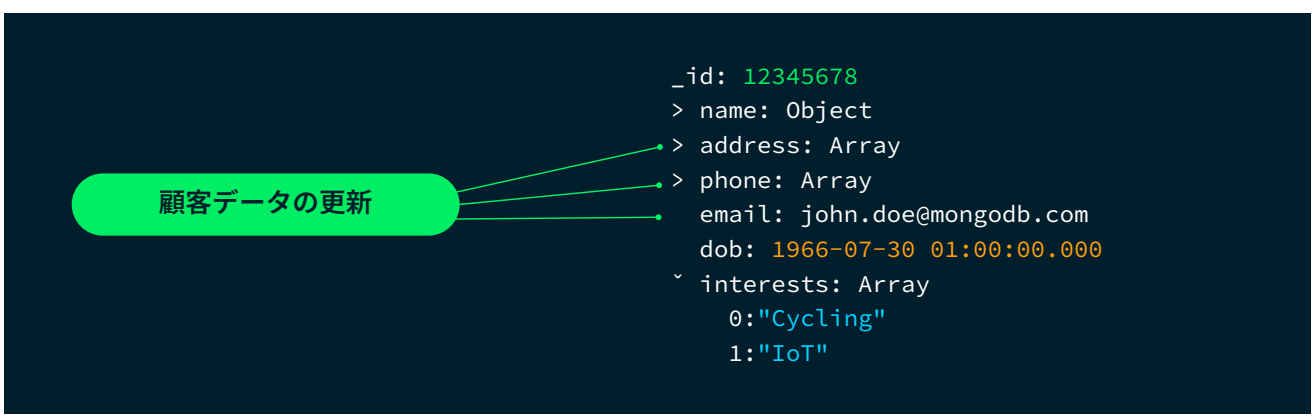

図 5：ドキュメントデータベースでの顧客データの更新

マルチドキュメントトランザクションが最適なユースケース

複数のドキュメントに一連の操作を行う場合、トランザクションにACID保証が必要なことがあります。最も一般的なものは、異なる要素間での値の交換を行い、その処理が「完全に実行されるか、全く実行されない」状態を保つ必要のあるアプリケーションです。バックオフィスに存在する「記録管理システム」や「基幹業務（LOB）」アプリケーションは、マルチドキュメントトランザクションの活用が最適なワークロードの典型的な例です。その他、次のようなユースケースがあります。

- 銀行の口座間における資金の移動、決済処理システム、取引プラットフォーム：株式や為替の新しい取引をティックデータストアに登録すると同時に、リスクシステムや市場データのダッシュボードの情報を更新する処理など。
- サプライチェーンシステムや契約システムを通じた商品やサービスのオーナーシップの移動：注文のコレクションに新しい注文を挿入し、在庫コレクションから在庫の数を減らす処理など。
- 料金システム：携帯電話の利用者が通話を終えた時点で新しい通話明細の記録を追加し、通話プランの月次料金の内容を更新する処理など。
- ドキュメントのシャードキーの値を修正することで、データをシャードクラスタに再配置。

MongoDB は、マルチドキュメントトランザクションのサポートを開始する前から、既に多くのユースケースに対応していました。現在では、データベースでマルチドキュメントトランザクションが自動的に処理されるようになり、開発者の作業はさらに容易になりました。以前は、トランザクションの制御をプログラムでアプリケーションに実装しなければなりませんでした。データの整合性を確保するためには、データベースの更新がコミットされるまでに、全ての操作が正常終了している必要があり、そうでない場合は全ての変更をロールバックする必要がありました。このようなプロセスは複雑さが増し、アプリケーションの開発スピードを減速させる要因となります。ある金融サービス業界の MongoDB ユーザー企業は、マルチドキュメントトランザクションを採用したことで、アプリケーションのコードを 1,000 行以上削除できたと報告しています。

また、クライアント側でトランザクションを実行すると、アプリケーションにパフォーマンスのオーバーヘッドが生じるおそれがあります。例えば、企業データの管理と統合を手掛けるあるグローバル ISV では、既存のクライアント側トランザクションロジックをマルチドキュメントトランザクションに移行した結果、マスターデータ管理ソリューションにおけるMongoDBのパフォーマンスが向上しました。スループットは90%増加し、2 つのコレクションにわたって 6 つの更新を実行するトランザクションのレイテンシは 60% 以上削減されました。しかし、この企業の場合は、パフォーマンスの向上に大きく影響した要因は、クライアント側にあった処理を移行したことよりも、書き込み保証の処理が変更されたことにありました。従来の方法では、トランザクション内の各書き込み操作で、アプリケーションが過半数のレプリカで操作が完了したことを確認するため、majority 書き込み保証を待機する必要があり、この確認を受け取るまで次の書き込みに進むことができませんでした。しかし、マルチドキュメントトランザクションでは、アプリケーションはトランザクションをコミットするときのみ待機を求められます。

マルチドキュメント ACID トランザクションの MongoDB での対応

MongoDB のトランザクションは、リレーショナルデータベースと大きな違いはありません。マルチステートメントであり、構文も似ているため、トランザクションを取り扱った経験があれば誰でもすぐに使いこなせます。

以下の Python のコードスニペットは、トランザクション API の例です。

```
with client.start_session() as s:
s.start_transaction()
collection_one.insert_one
  (doc_one, session=s)
collection_two.insert_one
  (doc_two, session=s)
s.commit_transaction()
```

以下に示すスニペットは、Java のトランザクション API を示しています。

```
try (ClientSession clientSession
= client.startSession()) {
clientSession.startTransaction();
collection.insertOne(clientSession,
  docOne);
collection.insertOne(clientSession,
  docTwo);
clientSession.commitTransaction();
}
```

上記の例が示すように、トランザクションには MongoDB のクエリ言語と通常の構文が使用されています。トランザクションは、レプリカセット内でドキュメントやコレクションを横断的に実行する場合でも、MongoDB 4.2 の分散トランザクションを使用してシャードクラスタ全体で実行する場合でも、一貫した実装がされています。

以下のトランザクションのブロックコードのスニペットでは、MongoDB の構文と MySQL の構文を比較しています。このコード例からわかるように、従来のリレーショナルデータベースの使用経験があれば、ドキュメントトランザクションを容易に理解できます。

MySQL

```
db.start_transaction()
cursor.execute(orderInsert, orderData)
cursor.execute(stockUpdate, stockData)
db.commit()
```

MongoDB

```
s.start_transaction()
orders.insert_one(order, session=s)
stock.update_one(item, stockUpdate,
session=s)
s.commit_transaction()
```

スナップショットの分離により、トランザクションでは、データの一貫性が確保され、処理は「完全に実行されるか、全く実行されない」状態となり、データの整合性が維持されます。トランザクションは、1 つまたは複数のコレクションやデータベースに含まれる複数のドキュメントに対する処理に適用できます。また、マルチドキュメントトランザクションの対応に伴う MongoDB での変更が、マルチドキュメントトランザクションを必要としないワークロードのパフォーマンスに影響を与えることはありません。

トランザクションの実行中、トランザクションは自身のコミットされていない書き込みの読み取りができます。ただし、外部の操作からは、そのトランザクションのコミットされていない書き込みは見えません。また、コミットされていない書き込みは、トランザクションがデータベースにコミットされるまでセカンダリのノードには複製されません。トランザクションがコミットされると、その変更はすぐに複製され、全てのセカンダリレプリカにアトミックに適用されます。

「MongoDB のデータプラットフォームで ACID トランザクションの専用サポートが提供されたこと、私たちのコラボレーションの結果が Spring Data MongoDB の Lovelace リリースに反映されていることをうれしく思います。この製品には、Spring の代表的なアノテーション駆動型の同期トランザクションサポートが追加されており、応答性の高いトランザクションには MongoTransactionManager を利用しています。この MongoTransactionManager は、ReactiveMongoTemplate で公開されている MongoDB の ReactiveStreams ドライバーと Project Reactor データタイプを基盤としています。」

Pivotal Spring 製品・マーケティング部門リード Pieter Humphrey 氏

読み取り・書き込みの保証

複数のレプリカセットやシャードクラスタを横断してトランザクションを機能させる場合、MongoDB のトランザクションのドキュメントで推奨される読み取り・書き込み保証を各トランザクションのはじめに構成することが、開発において重要です。

プライマリのエレクションが行われるときに書き込みの耐久性を確保するには、書き込み保証は majority を使用し、読み取り保証は majority またはスナップショットを選択します。

MongoDB 4.4 以降では、クラスタ全体で読み取り・書き込み保証を設定できるようになり、アプリケーション管理者がサーバー側で最適なデフォルト設定を行うことが可能です。こうした強力でグローバルな既定の読み取り・書き込み保証を構成することで、トランザクションごとにユーザーが明示的な設定を行う必要がなくなります。また、Atlas で MongoDB を実行している場合でも、MongoDB を自社で管理している場合でも、全てのクライアントで動作を統一できます。さらに、必要に応じてクライアントごとやセッションごとにドライバーのデフォルト設定を再定義することも可能です。

ポイントインタイムでのグローバルな読み取り

MongoDB のトランザクションインフラを提供することで、スナップショットの読み取りの分離によって、読み取り専用のトランザクションで実行されるクエリや集計が、シャードクラスタのどのプライマリレプリカでも、グローバルに一貫性のあるデータベースのスナップショットを参照して実行します。

この仕組みにより、データが複数のシャードに分散している場合や、同時書き込みによって同時に変更された場合でも、クライアントには一貫性のあるデータが返されます。

トランザクションのベストプラクティス

前述のとおり、MongoDB における既存のドキュメントのアトミック性保証は、アプリケーションのトランザクションニーズの 80～90% を満たしており、アプリケーションのデータ整合性を確保するための有効な方法です。マルチドキュメントトランザクションが必要な場合は、開発者に推奨するベストプラクティスがいくつかあります。

長時間実行されるトランザクションの作成や、ACID 特性を持つ単一のトランザクションで膨大な数の処理を実行すると、WiredTiger ストレージエンジンのキャッシュに大きな負荷がかかるリスクがあります。これは、キャッシュが最も古いスナップショットが作成されてから以降の全ての書き込み状態を維持しなければならないためです。トランザクションは実行中、常に同じスナップショットを使用するため、新しい書き込みがキャッシュに蓄積します。しかし、これらの書き込みは、古いスナップショット上で現在実行中のトランザクションがコミットまたは中断するまで、フラッシュできません。中断された時点でトランザクションがロックを解放し、WiredTiger がスナップショットを削除できるようになります。データベースのパフォーマンスを予測可能なレベルに維持するうえで、開発者は以下の点を考慮する必要があります。

1. デフォルトでは、マルチドキュメントトランザクションは実行時間が 60 秒を超えると、MongoDB によって自動で停止されます。ただし、サーバーに書き込むデータの量が少なければ、トランザクションの実行時間を長く設定できます。タイムアウトを防ぐために、トランザクションを小さく分割し、設定した時間内に実行が完了できるようにすることが重要です。また、トランザクション内で高速なデータにアクセスできるように、インデックスの範囲を適切に設定して、クエリパターンを最適化する必要があります。
2. 1 つのトランザクション内で読み取ることのできるドキュメント数に制限はありません。ベストプラクティスとして、1 つのトランザクション内で変更するドキュメントの数は、1,000 以下が推奨されます。1,000 を超えるドキュメントの変更が必要な場合は、トランザクションを複数に分割し、各トランザクションでバッチ処理を行うようにします。
3. トランザクションには適切な読み取り／書き込みの設定を行うことが重要です。
4. トランザクションが中止された場合は、例外がドライバーに返され、トランザクションは完全にロールバックされます。開発者は、ネットワークの一時的な障害やプライマリレプリカのエレクションなどでトランザクションが中断された場合、それを補足して再実行できるロジックをアプリケーションに組み込む必要があります。再試行可能な書き込みを使用すると、MongoDB ドライバーが自動でトランザクションのコミットステートメントを再実行します。
5. 複数のシャードに影響を与えるトランザクションでは、ネットワーク経由で参加している複数のノードに処理を割り振ると、パフォーマンスの大幅な低下を引き起こす原因となります。
6. 複数のシャードにわたるトランザクションでは、参加するシャードのいずれかにレプリカセットのアービターが組み込まれていると、エラーが発生し、処理が中断します。
7. シャード間のデータバランスの再調整チャンクマイグレーションを使用している場合は、対象のコレクションに対してトランザクションを実行すると、チャンクマイグレーションのパフォーマンスに悪影響を与えるリスクがあります。

レプリカセットと分散型トランザクションに関する全ベストプラクティスは、マルチドキュメントトランザクションに関する MongoDB のドキュメントでご覧いただけます。

トランザクション対応に向けた機能強化の歩み

トランザクション対応に向けた MongoDB の歩みは、MongoDB に WiredTiger ストレージエンジンを統合した 2015 年に始まります。数年にわたるエンジニアリングの歴史の中で、ストレージレイヤー自体から、レプリケーションコンセンサスプロトコル、シャーディングアーキテクチャに至るまで、プラットフォームの全ての部分をカバーして基盤を構築してきました。MongoDB は、高粒度な一貫性と耐久性の保証を実現し、グローバル論理クロックや、リファクターされたクラスタメタデータ管理などの機能を提供しました。これらの機能強化はいずれも MongoDB のドライバーから利用可能な API を通じて公開されています。

図 6 は、MongoDB のマルチドキュメントトランザクションを可能にするための 2019 年 6 月時点での主要なエンジニアリングプロジェクトを時系列で示しています。これらのプロジェクトに共通する設計目標は、MongoDB のドキュメントモデルの強力な機能と分散システムの利点を活かし、マルチドキュメントトランザクションを必要としないワークロードのパフォーマンスを低下されることなく新機能を実装するというものでした。

MongoDB 3.0	MongoDB 3.2	MongoDB 3.4	MongoDB 3.6	MongoDB 4.0	MongoDB 4.2
新しいストレージエンジン（WiredTiger）	レプリケーションプロトコルの強化：厳格な一貫性と耐久性	シャードメンバーシップの認知	シャードクラスタにおけるセカンダリの一貫性のある読み取り	レプリカセットのトランザクション	分散型トランザクション
	WiredTiger デフォルトストレージエンジン		論理セッション	タイムスタンプ対応のカタログ	Oplog Applier Prepare Support
	Config サーバーの管理性の強化		再試行可能な書き込み	スナップショットの読み取り	分散型コミットプロトコル
	“majority” 読み取り保証		因果整合性	リカバリ可能なロールバックと ST チェックポイント	ポイントインタイムでのグローバルな読み取り
			クラスタ全体を対象とする論理クロック	タイムスタンプのリカバリ	WiredTiger による修復の拡張
			テンプレートの使用に関するストレージ API の変更	シャードカタログの強化	トランザクションマネージャー
			読み取り保証における majority 機能の常時有効化		
			コレクションカタログのバージョニング		
			シャーディングの UUID		
			WiredTiger の大規模ドキュメントを対象にした高速のインプレースアップデート		

図 6： トランザクション対応に向けた機能強化の歩み — 複数年にわたるエンジニアリング投資と複数の製品リリース

プロジェクトを進めるうえで特に難しかった点は、2 つの優先事項のバランスを保つことでした。まず、MongoDB 4.0 で開始するマルチドキュメントトランザクションに必要な基盤の整備が必要でした。一方で、有用な機能をすぐにユーザーに公開し、MongoDB をデータ処理に最適なソリューションとする必要もありました。MongoDB では、この 2 つの優先事項を可能な限り両立させるため、以下のような改良を行いました。

- WiredTiger Inc. を買収し、WiredTiger のストレージエンジンを MongoDB 3.0 に統合することで、ドキュメントレベルの同時実効性制御と圧縮によるスケーラビリティの大幅な向上を実現しました。また、MVCC をサポートし、マルチドキュメントトランザクションに対応可能なストレージレイヤーの基盤を整備しました。
- MongoDB 3.2 で実施されたコンセンサスプロトコルの強化により、プライマリレプリカセットメンバーやネットワークパーティショニングの障害から、より迅速、確実に復旧できるようになりました。また、書き込みの耐久性の保証も一層強化されました。これらの新機能は、当時の MongoDB ユーザーにとって即効性があったばかりでなく、現在でも、トランザクション対応には欠かせない機能です。
- MongoDB 3.2 から導入された readConcern クエリオプションにより、アプリケーションで読み取りの分離レベルを処理ごとに指定できるようになりました。これにより、強力で高粒度な同時実効性制御も可能となり、MongoDB 4.0 以降でのマルチドキュメントトランザクション対応に必要な分離レベルも実現されました。

MongoDB 3.6 で導入された多くの新機能は、グローバルな論理クロックとストレージレイヤータイムスタンプの導入を基にしています。その基盤は既存の学術的な概念として存在していた Lamport クロックとタイムスタンプの実装によるものです。この機能により、分散クラスタ内の全ての操作で一貫した時間が適用され、マルチドキュメントトランザクションでスナップショットの分離が保証されます。また、これらの機能により MongoDB 3.6 のリリースと同時に、開発者の生産性を高める次のようなさまざまな機能も提供可能となりました。

- **変更ストリーム**：データベースで発生するデータの変更をリアルタイムで表示、フィルタリング、処理できるため、反応性の高いアプリケーションの開発が可能になります。論理クロックとタイムスタンプを使用することで、変更ストリームの再開性が向上し、一時的なノードの障害が発生しても自動で復旧します。これにより、アプリケーションは、ノード障害後の変更を適用するだけで処理を再開できます。
- **論理セッション**：因果整合性と再試行可能な書き込みの基盤として機能します。マルチドキュメントトランザクションの観点から、この機能は、トランザクションにおける各ステートメントの実行コンテキストを管理し、分散型ノード全体でクライアントとサーバーの処理を調整できる点に価値があります。
- **因果整合性**：論理セッションとクラスタの時間で有効化される機能です。開発者は、インテリジェントな分散型データプラットフォームの高い拡張性と可用性を活用できると同時に、「自身の書き込みの読み取り」保証で強力なデータの整合性を維持できます。
- **再試行可能な書き込み**：クラスタのエレクションや一時的な障害が発生した場合に、一回限りの処理セマンティックスをサーバーに試行させるアプリケーションを容易に開発できます。

マルチドキュメントの ACID トランザクションはサポートが開始された MongoDB 4.0 では、レプリカセットに限定されていました。MongoDB 4.2 では、分散型トランザクションのサポートが追加され、シャードクラスタ全体で ACID 特性が保証されるようになりました。MongoDB 4.2 での分散型トランザクションの機能実現において、2 つの重要なプロジェクトは、Oplog Applier Prepare Support と分散型コミットプロトコルの提供です。

- **Oplog Applier Prepare Support：**クロスシャードトランザクションにより、ドキュメントをアトミックにコミットしている間に、スレッドにドキュメントの読み取りをさせない新しい状態が導入されています。これにより、MongoDB の一貫性と分離の保証が確実に達成されます。
- **分散型コミットプロトコル：**シャード間でトランザクションの実行を担うコーディネーターノードを選択します。この新しいプロトコルは、トランザクションをコミットするか中断するかを決定し、「処理が完全に実行されるか、全く実行されないか」の ACID 保証を実施します。

分散型トランザクションの機能を MongoDB 4.2 と組み合わせることで、アプリケーションのデプロイメントを簡素化する以下のような機能強化も実現しています。

- **大規模なトランザクション：**複数の oplog エントリー全体のトランザクションを表現することにより、既存の 60 秒のデフォルトランタイム制限内で、16 MB 以上のデータを単一の ACID トランザクションとして書き込むことが可能になりました。これにより、複数の大きなドキュメントをデータベースにアトミックに挿入するようなユースケースで、16 MB の制限に達するかを事前にテストする必要がなくなります。
- **トランザクションの詳細な診断：**mongod ログにトランザクションのメトリックが書き込まれるようになり、実行時間、ロック、処理されたドキュメントの数、ストレージエンジンでの処理にかかった時間などのテレメトリを把握できるようになり、アプリケーションのパフォーマンス調整が容易になりました。また、currentOp および serverStatus コマンドに、トランザクション固有のセクションが追加され、開発者はデータベースで実行中の全てのトランザクションを即座に把握できるようになりました。
- **トランザクションエラーへの対処：**ドライバー側のヘルパーにコールバック API が新たに搭載され、トランザクションの中断に対処するアプリケーションのリトライロジックの開発が容易になりました。

MongoDB 4.4 では、トランザクション内に新規のコレクションとインデックスを作成する機能が導入されています。

トランザクション処理の内部

マルチドキュメントの ACID トランザクションを追加するには、基盤となる MongoDB サーバーとストレージエンジンの強化が必要でした。以下のセクションでは、主な機能強化について説明します。また、トランザクションの Web ページもご覧ください。

MongoDB と WiredTiger における低レベルのタイムスタンプ

MongoDB の書き込み操作のタイムスタンプ表示を WiredTiger ストレージレイヤーに追加しました。この変更により、MongoDB の時間と順序の概念を照会して、特定の時間以前または以後のデータのみを抽出できるようになります。また、MongoDB のスナップショットを作成し、複数のデータベースの処理や複数のトランザクションを共通の時点から実行できます。

背景：MongoDB では、oplog という運用ログを使用してレプリケーションを実行しています。oplog は、データベースに適用された最新の処理をリスト化する専用のコレクションで、サーバーに置かれています。これらの処理をセカンダリのサーバーで再実行することで、レプリカを最新の状態に維持し、プライマリサーバーとの一貫性を維持します。プライマリサーバーの内容をレプリカに正しく反映させるには、oplog に正しい操作の順序が記録されていることが重要です。

MongoDB では、oplog の操作順序を管理するとともに、レプリカがその順序に従って oplog にアクセスできるように管理しています。また、WiredTiger の強力なストレージエンジンに切り替えてその能力を最大限に引き出すには、WiredTiger のストレージエンジンの順序に関する独自の考え方と、サーバーレイヤーの順序の考え方を調整する必要がありました。

WiredTiger Storage：WiredTiger は、キーと値のツリー構造で全てのデータを保存します。MongoDB のストレージレイヤーとして使用された場合には、データはドキュメントまたはインデックスの一部である可能性があり、どちらも WiredTiger のツリーに保存されます。キーの値が更新されると、WiredTiger は更新構造体を生成します。この構造体には、トランザクションに関する情報や、変更されたデータ、後続の変更を参照するためのポインタが含まれています。この構造体は WiredTiger によって元の値に付加されます。後から更新された構造は、それ以前の更新構造に追加され、時間の経過とともに値の異なるバージョンが連結されます。

これが、WiredTiger によって実装された同時実行性制御のマルチバージョンコンポーネントです。値の「現在」の状態を取得する際に、WiredTiger は独自のルールに従い、更新構造体の内容を読み取ります。更新を適用順番は、MongoDB の Oplog での順番とは異なります。WiredTiger は、セカンダリに対する複数の書き込みを可能な限り並列処理するためです。プライマリは多くの並列書き込みを処理できます。しかし、セカンダリはレプリケーションのためにプライマリのスループットと同等の並列書き込みを行う必要があります。

タイムスタンプ：WiredTiger ストレージエンジン内で MongoDB での順番を維持するために、更新構造体に「timestamp」フィールドが追加されました。このフィールドの値は、MongoDB から WiredTiger に渡され、重要なメタ情報アイテムとして扱われます。WiredTiger のクエリを構成するときはタイムスタンプを指定して、特定時点のデータの状態を正確に記録できます。こうすることで、MongoDB の順番と WiredTiger ストレージ内の順番をマッピングできるようになります。

セカンダリでの読み取り：セカンダリのレプリカとプライマリのレプリカの同期は、oplog の更新バッチの読み取りで行います。その後、セカンダリのレプリカは自身のストレージに変更を適用します。タイムスタンプがない場合は、更新バッチが完了するまで読み取りのクエリがブロックされます。これは、誤った順番で書き込みが行われないようにするためです。しかし、現在ではタイムスタンプが導入されているため、最新バッチの開始時点から、読み取りのクエリを継続して実行できます。タイムスタンプを使用することで、クエリに対する一貫性のある応答が確保され、セカンダリでの読み取りがレプリケーションの更新で妨げられることがなくなります。

レプリケーションのロールバック：MongoDB のクラスタにある複数のセカンダリサーバーがプライマリのレプリケーションによって更新される場合は、ある時点における同期の進行の程度はセカンダリサーバーによって異なります。このため、同期処理が完了したセカンダリサーバーの数が過半数に達する時点が存在し、これを「マジョリティコミットポイント」と呼びます。プライマリサーバーで障害が発生した場合、全てのサーバーで利用が保証されるデータは、マジョリティコミットポイントまでのデータに限られます。セカンダリサーバーのいずれか1つが、RAFT ベースのコンセンサスプロトコルによって新しいプライマリサーバーに選出されたときにも、マジョリティコミットポイントまでのデータが使用されます。

障害が発生したプライマリサーバーがクラスタに復帰する際に、そのサーバーをクラスタ内の残りのサーバーと同期させるプロセスは極めて複雑でした。復帰したプライマリサーバーには、マジョリティコミットポイント以降のデータが含まれている可能性があり、クラスタが把握していない変更を特定し、変更されたレコードを古いバージョンに復元する必要がありました。

しかし、タイムスタンプが導入されたことで、このプロセスは大きく簡素化されました。マジョリティコミットポイントのタイムスタンプを取得し、復帰したプライマリサーバーのストレージに適用することで、そのタイムスタンプの時刻以降に発生した変更を排除できます。この処理が完了すると、ノードは再びクラスタに参加できるようになり、プライマリのレプリケーションを開始できます。

タイムスタンプとトランザクション：タイムスタンプ情報を WiredTiger のツリーの中核部分にプッシュすることで、WiredTiger のマルチバージョン同時実行制御により、ロックの抑制や再同期プロセスの効率化が可能になります。また、特定時点のスナップショットを作成することで、サーバーがその時点にロールバックできるようになります。この機能は、マルチドキュメント ACID トランザクションで正確な保証を実現するために不可欠な機能です。

論理セッション

論理セッションの作成により、単一処理のトランザクションでも複数処理のトランザクションでも、システム全体で使用されているリソースの状態を追跡できるようになりました。これにより、シンプルで正確な処理のキャンセルが可能となり、分散型のガベージコレクションが実現しました。

背景：これまで MongoDB には、クライアントから始まり、mongos クエリルーターを経て、クラスタを構成するシャードとレプリカセットに至るフローの中で、トラッキングすることでメリットを得られる操作が数多くありました。しかし、このような処理を追跡する識別子は存在せず、システムはさまざまなヒューリスティックに頼るほかありませんでした。

MongoDB は、論理セッションと論理セッション識別子を開発し、この課題を解決しました。論理セッション識別子は、lsid と呼ばれる小さな一意の識別子で、クライアントが MongoDB クラスタとの通信に付加し、MongoDB クラスタは、クライアントが使用する全てのリソースに lsid を付加します。

lsid は、MongoDB ドライバーによってクライアントに自動的に生成されるため、セントラル ID サービスを呼び出す必要がなくなります。lsid は、クライアントが生成するグローバルな一意 ID（GUID）である 1 つの識別子と、ユーザー名の SHA256 ダイジェストであるユーザー識別子で構成されています。

MongoDB 3.6 以降では、全てのクライアントの処理が論理セッションと関係付けられています。また、クラスタを横断するコマンドの処理には、lsid が関係付けられます。

論理セッションとキャンセル処理：操作では複数のリソースを使用します。例えば、find() 操作は、クラスタ内で関連する全シャードにカーソルが作成し、それぞれが最初のバッチが返す結果の取得を開始します。論理セッションが登場する以前は、こうした操作のキャンセルには、管理者権限で全てのシャードを対象に、該当するアクティビティを特定して停止する必要がありました。

さらに、コマンドの発行に関係する MongoDB のプロセスが停止したときは、キャンセル処理はより複雑になります。この場合は、カーソルが最初のバッチを構成して結果の取得がタイムアウトするまで待機する必要がありました。

しかし、論理セッションを利用することで、プロセス全体が簡素化されました。特定の論理セッションを対象にした停止コマンドをクラスタに発行することが可能になり、カーソルを含む全てのリソースに論理セッション識別子をタグ付けすることで、特定の lsid を持つリソースの停止や解放の操作が容易になります。また、lsid には、ユーザーID が含まれるため、特定のユーザーの全セッションリソースを削除するコマンドも発行できます。

論理セッションと分散型ガベージコレクション：従来、MongoDB ではリソースのタイムアウトはローカルで管理されており、リソースを格納しているノードが、タイムアウトやガベージコレクションの必要性を判断していました。将来的な MongoDB の機能強化に伴い、クラスタ全体でタイムアウトとガベージコレクションを認識する必要性がでてきたため、lsid を活用してそれらを有効化するように変更が加えられました。

MongoDB 3.6 では、mongod と mongos のプロセスで次の 2 つの機能を追加しました。セッションを管理するコントローラプロセスの実行と、コントローラのプロセスに紐付く lsid のリストの管理です。コントローラは 5 分おきにリストをセッションコレクションと同期し、セッションが最後に使用された時刻を更新します。この最終使用時刻を基準にして 30 分後にトリガーされる TTL インデックスを設定すると、30 分間どのコントローラによっても使用されていないセッションが判別され、該当セッションと関連するリソースが、クリーンアップの対象になります。

論理セッションとトランザクション：全ての操作とリソースに論理的なセッション ID をタグ付けすることで、MongoDB における長時間実行される広範囲に分散した操作の管理が容易になりました。lsid は、キャンセルやガベージコレクションのシナリオで即座に効力を発揮し、さらに MongoDB 4.0 以降における他の機能の基盤にもなっています。トランザクションをセッション内で確実に実行することで、トランザクションが正常にコミットされたか中断されたかを問わず、そのトランザクションの保存とクリーンアップが可能になります。

ローカルスナップショットの読み取り

トランザクションを効果的に利用するためには、特定の時点におけるサーバーの状態を把握できる機能も必要です。データをスナップショットの一部として識別する低レベルのタイムスタンプを使用することで、特定の時点での一貫性の確保が可能になります。リソースのトラッキングには、論理セッションを使用します。

背景：従来、長時間実行されるクエリは、カーソルに出力命令が出されるまでデータベースの単一のスナップショットからデータを取得し続けていました。カーソルの出力処理において、カーソルの状態が保存され、データベースのロックとスナップショットが解放されます。

ロックが再度獲得された状態に復帰すると、新しいスナップショットが取得され、カーソルがリストアされ、長時間機能するクエリは処理を続行します。これは、出力が行われるたびに、スナップショットの時間が前に進む可能性があることを意味します。

スナップショットの維持：ローカルスナップショットの読み取りでは、出力処理中にスナップショットやロックを解放しません。クライアントの論理セッションを利用し、サーバーのロックやスナップショットが記録され、処理やトランザクションの完了時にそれらのリソースの解放状況が追跡されます。また、このセッションを利用することで、サーバー全体でリソースの使用状況を関連付けることができます。

ローカルスナップショットの読み取りの使用：ローカルスナップショットの読み取りを機能させるには、「スナップショット」の readConcern を使い、マルチドキュメントトランザクションをクライアントのセッション内に作成する必要があります。「スナップショット」の readConcern を使用すると、明確なステートメントの数やネットワークトラフィックの量や出力ポイントの数に関係なく、操作は全て一貫性のあるスナップショットに基づいて実行されます。トランザクションが格納されているセッションに因果整合性がある場合は、セッション内の前の処理トランザクションと整合性のあるスナップショットも提供されます。

ローカルスナップショットの読み取りとトランザクション：ローカルスナップショットの読み取りにより、トランザクションは、クラスタ全体で任意の時点から操作を行うことが可能です。これにより、スナップショットの時刻が前に移動することなく一貫性を保ったままで、マルチドキュメント操作を行うことができます。

グローバル論理クロック

MongoDB では、グローバル論理クロックへの変更により、時刻同期の機能が強化されています。このクロックは、ハイブリッド論理クロックとして実装され、暗号化によって保護されたグローバルな論理時刻がシャードクラスタ全体に適用されます。特に、マルチドキュメント ACID トランザクションではグローバルなスナップショットの取得が必要なため、一貫したグローバルな時刻の把握が不可欠です。

背景：MongoDB のクロックは、MongoDB シャードのプライマリにある oplog でトラッキングされています。全ての操作は、データに永続的な変更を加えるため、oplog に記録されます。各処理が記録されるときには、プライマリの論理クロックの時刻が「照会」され、そのクロックの「時刻」が操作と一緒に oplog に記録されます。この結果、oplog に順序設定が記録され、シャード内の全ノードで同期が可能になります。

ハイブリッド論理クロック：シャードには、シャード固有のクロックがあります。しかし、クラスタ全体で機能させることはできません。このクロックには、個々の時刻に関する共通の規準や単位がありません。MongoDB 3.6 からは、ハイブリッド論理クロックが実装されるようになりました。

このハイブリッドクロックは、システムの物理時刻と同時刻に発生した操作のカウンタを組み合わせ、ハイブリッドタイムスタンプを生成します。この新しいタイムスタンプは、クラスタ全体に確立された全コネクションを通じて拡散されます。サーバーがタイムスタンプ付きのメッセージを受け取ったときに、そのタイムスタンプがノードの現在のタイムスタンプより後になっている場合、サーバーは、最後に確認した自身のタイムスタンプを新しい値で更新します。新しい変更が発生すると、最新の時刻は即座にインクリメントされます。

改ざんの防止：攻撃者が不正な時刻やカウントをシステムに注入し、新たな操作のインクリメントが行えないようにするリスクと、シャードが新しい変更を受け入れられなくなります。このような攻撃のリスクは常にあります。しかし、プライマリ自身のプライベートキーでタイムスタンプのハッシュを作成することで、この問題の発生を抑制できます。このハッシュで、タイムスタンプのソースがクラスタによって形成された信頼できるゾーン内にあることを検証します。攻撃者がプライベートキーを入手した場合は、より広範な侵害がシステム全体に発生するおそれがあります。

グローバル論理クロックとトランザクション：ハイブリッド論理クロックの導入により、レプリカセット内またはクラスタの複数のシャード間におけるノードの調整や同期が容易になりました。MongoDB 4.2 では、トランザクションのスコープが複数のシェードにまで拡張されたため、このスコープ内で機能する信頼性の高い同期型のクロックが重要な基盤なりました。クロックの改ざんを防ぐことで、その基盤の信頼性がさらに向上します。

安全なセカンダリの読み取り

ドキュメントのチャンクや、シャードキーのレンジ内にあるドキュメントをシャード間で移行すると、問題が発生することがあります。この問題は、安全なセカンダリの読み取りを使用することにより解決できます。この機能では、移行に参加しているシャードのプライマリだけでなくセカンダリも、移行中や移行済みのチャンクを把握できます。このプロジェクトにより、MongoDB の動的自動バランシングの機能がさらに使いやすくなりました。

背景：安全なセカンダリの読み取りが導入される以前は、どのシャードがどのデータのチャンクを保有しているのかを詳細に示すルーティングテーブルは、シャードのプライマリのみが保有していました。プライマリにクエリから照会がかかると、このルーティングテーブルを使用して、移行中のドキュメントをフィルタリングし、複数のシャードからのクエリ結果に重複が生じないようにしていました。

クラスタ全体でデータを均等に分散する再分散プロセスなどで、データのチャンクをシャード間で移行する際は、コピー元のシャードからコピー先のシャードにデータチャンクをコピーし、次に両方のルーティングテーブルを更新してデータのチャンクの新しい場所を表示し、その後コピー元のシャードからデータのチャンクを削除します。

この手法は、シャード内のプライマリにルーティングされるクエリでは有効です。しかし、シャード内のセカンダリに対するクエリは、移行が進行中であることを認識せず、自身が保持している結果を返します。移行の進行中は、両方のシャードのセカンダリにデータのコピーが存在するため、セカンダリからの読み取りでは両方のシャードからコピーが返される可能性があります。

ルーティングのレプリケーション：この問題に対する最もシンプルな解決策は、シャード内のプライマリノードからセカンダリノードにルーティングテーブルをレプリケートする方法です。移行中にルーティングテーブルが変更されると、その内容もセカンダリのノードにレプリケートされます。これにより、セカンダリノードは移行を効果的に認識し、指定の宛先にクエリを直接、正確にルーティングできるようになります。このルーティング情報をセカンダリの読み取りで利用するには、クエリで ReadConcern “local” または “majority” を送ります。

安全なセカンダリの読み取りとトランザクション：安全なセカンダリの読み取りを、統合タイムスタンプやグローバル論理クロックなどの機能と組み合わせることで、トランザクションをより予測可能な状態で動作させることができます。

再試行可能な書き込み

ドライバーが同じ書き込みを正確かつ安全に再実行できるようにし、サーバーで書き込みが重複するリスクを回避することで、エラー処理のコード量を削減し、クライアントの信頼性をさらに高めることができます。

背景：ドライバーがサーバーに書き込みを送信した際にエラーが発生すると、そのエラーが書き込み時に発生したものなのか、または確認応答が返される途中で発生したものなのかは、必ずしも特定できません。これは、サーバーが書き込みのコマンドを実行したかどうかを判断できないことを意味します。

フィールドをインクリメントする書き込みを例にして考えてみましょう。クライアントのドライバーがそのリクエストをサーバーに送信し、サーバーがそれに基づいて動作し、その後、クライアントに結果を返す前に接続が失われたとします。サーバー側では、フィールドがインクリメントされています。だたし、クライアントのドライバーにはこの情報は反映されていません。クライアント側で、リクエストがサーバーに到達する前に接続が失敗したと判断して書き込みを再試行すると、フィールドは再びインクリメントされてしまいます。これは明らかに誤った判断です。クライアント自身は、データベースの状態から書き込みが適用されたかどうかを判断しようとしても、多くのケースで判断を誤るリスクがあります。

再試行可能な書き込み（クライアント側）：再試行可能な書き込みは、複数回要求できます。ただし、適用できるのは1回のみです。retryWrites の接続設定を使用してドライバーレベルで実装し、少なくとも書き込み操作の承認（writeConcern > 0）が必要です。利用できるのは特定の操作に限られています。MongoDB 4.0 以降では、トランザクションのコミットと中断の処理も再試行可能になりました。ただし、これらはドライバーで明示的に有効化するのではなく自動的に再試行されます。いずれの場合でも、書き込みの再試行は1回限りです。

ドライバーはリトライを繰り返さないため、再試行可能な書き込みは、ネットワーク障害が続くようなケースよりも、一時的なネットワークエラーやプライマリのエレクションに適しています。書き込みの要求がサーバーに送信されるときには、論理セッション ID とトランザクション ID が付加されます。

再試行可能な書き込み（プライマリ側）：書き込みの要求がサーバーに到達すると、まずトランザクションテーブルと突き合わせたチェックが行われます。このリストには、論理セッションID、最後に参照されたトランザクション ID、操作が記録された oplog をポイントするポインタが記載されています。トランザクションテーブルに、一致する論理セッション ID やトランザクション ID があると、再試行可能な書き込みのロジックは、要求されている書き込みが再試行であると判断します。以前の書き込み操作が成功したと仮定した場合の生成結果がまとめられ、クライアントに確認応答として返されます。

トランザクションテーブルに一致する ID が存在しない場合は、書き込みは新規の操作であるとみなされ、そのまま適用されます。その後、oplog に新たに情報が登録されます。そして、トランザクションテーブルは、セッション ID に対してトランザクション ID と、oplog の新しいエントリーへのポインタを記録します。その後、受付確認がクライアントに返されます。

再試行可能な書き込み（セカンダリ側）：フェイルオーバーが発生し、セカンダリノードのいずれかがプライマリを引き継ぐまで、セカンダリはアプリケーションからの書き込みを直接処理しません。何も変更がないままプリマリの引き継ぎが生じると、セカンダリのトランザクションテーブルは空になり、以前にどの操作が行われたかどうかを確認できなくなってしまいます。この問題を解決するには、トランザクションテーブルのレプリケーションを実行します。他のレプリケーション操作とは異なり、トランザクションテーブルのレプリケーションは oplog に記録されている個々の書き込み処理の情報と統合されます。その結果、セカンダリノードは、論理的なセッション ID とトランザクション ID を取得して独自のトランザクションテーブルを構築し、特定の oplog エントリーまで修正できるようになります。

その後、プライマリがフェイルオーバーしてセカンダリがプライマリの処理に参加すると、完全なトランザクションテーブルも用意されてデータのレプリカが準備され、書き込みができるようになります。

再試行可能な書き込みとトランザクション：再試行可能な書き込みは、トランザクションと同じリトライの仕組みを実現するための基盤です。前述のとおり、コミットと中断の操作はどちらも再試行が可能です。しかし、再試行は、再試行可能な書き込みとは異なり、トランザクション全体として 1 回の操作として行われます。このため、MongoDB では、一時期なネットワークの障害やプライマリのエレクションの影響を受けずに、トランザクションが確実にコミットされる仕組みを提供します。

MongoDB におけるトランザクションとデータモデリングへの影響

MongoDB にトランザクション機能が追加されても、それで MongoDB がリレーショナルデータベースに変わるわけではありません。開発者の多くは、データを表形式で管理するリレーショナルデータベースよりも、MongoDB のような柔軟なドキュメントモデルが優れていると考えています。

MongoDB のデータモデリングに関するベストプラクティスは、マルチドキュメントトランザクションや、$lookup の集計パイプラインステージから提供される JOIN などの高度な機能を利用する場合にもそのまま当てはまります。エンティティに関連するデータは、可能な限り単一で豊富なドキュメント構造で格納することを推奨します。リレーショナルテーブル向けに構造化されたデータを MongoDB に移行するだけでは、MongoDB が持つ自然で高速かつ柔軟なドキュメントモデルや、分散型システムアーキテクチャの利点を最大限に活用できません。

「RDBMS を MongoDB に移行するためのガイド」では、アプリケーションをリレーショナルデータベースから MongoDB に移行する際のベストプラクティスを紹介しています。

まとめ

MongoDB は、モダンアプリケーションに最適な業界トップクラスのデータベースとしての地位を既に確立しています。ドキュメントデータモデルは、内容が豊富で、自然かつ柔軟にデータを扱うことができるモデルです。MongoDB では、従来のドライバーを使用して複数のドキュメントにアクセスできるため、アプリケーション開発の効率が 2～3 倍向上します。また、分散型のシステムアーキテクチャを採用しており、より多くのデータを処理し、ユーザーのニーズに応じた配置が可能で、常に高い可用性を維持します。

MongoDB は、これまでも、シングルドキュメントに対するアトミックな操作に対してトランザクションセマンティクスを既に提供し、多くのアプリケーションに対してデータ整合性のニーズを満たしてきました。新たにマルチドキュメント ACID トランザクションの機能が追加されたことで、多様なユースケースのサポートが可能になり、開発者に選択肢と安心感を提供しています。

マルチドキュメントトランザクションに関する Web ページでは、トランザクションを構築した MongoDB エンジニアによるドキュメントやその他の主要なリソースを紹介しています。MongoDB Atlas のフルマネージド型オンデマンドクラスタをご利用いただくか、MongoDB Atlas クラスタをダウンロードしてオンプレミス環境にインストールすることで、MongoDB のトランザクション機能をすぐにお試しいただけます。

「過去数十年にわたってトランザクションの保証機能はリレーショナルデータベースにとって不可欠な機能でした。しかし、非リレーショナルデータベースには通常実装されていませんでした。非リレーショナルデータベースの柔軟性や多用途性を選ぶか、トランザクション機能を優先するかを選択しなければなりませんでした。MongoDB が ACID トランザクションがサポートするようになったことで、トランザクションの保証と非リレーショナルデータベースの両方のメリットを望むユーザーにとって最適な選択肢となりました。」

Redmonk プリンシパルアナリスト Stephen O’Grady 氏

MongoDB の製品とサービス

MongoDB, Inc は、MongoDB の開発と運用を行うデータベースの専門企業です。17,000 社以上の企業が MongoDB の製品を導入し、活用しています。MongoDB では、データベースの利用を簡素化するためのさまざまなクラウドサービやソフトウェアを提供しています。

MongoDB Atlas：さまざまな最新の用途に対応できるフルマネージド型のグローバルクラウドデータベースサービスで、AWS、Azure、Google Cloud 上でホストされています。クラス最高の自動化を実現しており、実証済みの手法で、可用性、拡張性、セキュリティ基準の準拠を保証します。

MongoDB Enterprise Advanced：自社のインフラで MongoDB を運用するのに最適なエンタープライズ向けデータベースソリューションで、高度なソフトウェア、サポート、認証、その他のサービスなど、ビジネスニーズに合わせたパッケージ製品です。

MongoDB Charts：MongoDB のビジュアルデータの作成、共有、組み込みに最適なツールです。ネスト化された複雑なデータの分析を迅速、容易に可視化します。個別のチャートはどの Web アプリケーションにも組み込みが可能で、複数のチャートを組み合わせてライブダッシュボードを作成し、共有することも可能です。

Realm Mobile Database：iOS デバイスや Android デバイスでローカルにデータを保存するための直感的に利用できるリッチなデータモデルを提供します。MongoDB Realm sync-to Atlas と組み合わせることで、応答性と信頼性に優れ、オフラインでも利用できるアプリケーションの開発が容易になります。

MongoDB Realm：開発者が重要な機能を迅速に評価し、開発できるツールです。Realm Sync for mobile や Realm の GraphQL サービスなどのアプリケーション開発サービスは、Realm Functions、Triggers、Data Access Rules と組み合わせて使用できるため、セキュリティと性能に優れたアプリケーションを簡素化されたコードで開発できます。

MongoDB Cloud Manager：オンプレミスインフラにある MongoDB の管理を支援するクラウドベースのツールです。プロビジョニングの自動化、高粒度な管理、継続的バックアップが可能なフル機能の管理スイートで、少ない運用負荷で、データベースを完全に制御できます。

MongoDB Consulting：開発スピードの向上を可能にし、本番環境におけるパフォーマンス調整やスケーリングのサポートを提供します。また、次期リリースに向けたリソースの集約を支援します。

MongoDB Training：MongoDB のスキルを身につけ、設計からミッションクリティカルな大規模システムの運用まで対応できるようになります。開発者、データベース管理者、アーキテクトが MongoDB の知識を深めることができます。

関連リソース

MongoDB について詳しくは Web ページをご覧ください。

導入事例

プレゼンテーション資料

オンライントレーニング（無料）

イベント・Web セミナー

MongoDB のドキュメント

MongoDB Atlas：
MongoDB 向けデータベース・
アズ・ア・サービス

MongoDB Enterprise の
ダウンロード

MongoDB Stitch
サーバーレスプラットフォーム

MongoDB Realm