Transactions in MongoDB 4.0

Thursday, May 30, 2019

Pythian
A global IT company that helps businesses leverage disruptive technologies to better compete.

Our services and software solutions unleash the power of cloud, data and analytics to drive better business outcomes for our clients.

Our 20 years in data, commitment to hiring the best talent, and our deep technical and business expertise allow us to meet our promise of using technology to deliver the best outcomes faster.
# Diversified, Blue Chip Client Base

<table>
<thead>
<tr>
<th>Media/Information Services</th>
<th>Retail</th>
<th>E-commerce</th>
<th>SaaS</th>
<th>Financial Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fox Sports</td>
<td>Sonos</td>
<td>Tech Data</td>
<td>Grasshopper</td>
<td>Pireaus Bank</td>
</tr>
<tr>
<td>National Geographic</td>
<td>Urban Outfitters</td>
<td>Arc’teryx</td>
<td>Mapp</td>
<td>Commerzbank</td>
</tr>
<tr>
<td>AEG</td>
<td>AllSaints</td>
<td>Freshdirect</td>
<td>Entech Datacard</td>
<td>Ford Direct.com</td>
</tr>
<tr>
<td>BBM</td>
<td>Medianet</td>
<td>Fast Retailing</td>
<td>Rockyou</td>
<td>Paymentus</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© The Pythian Group Inc., 2019
About me

45 years of working with data

Started with flat files on Tape drives

Over the years have worked with:

- ISAM, Codasyl Network databases, IBM IMS, Tandem SQL/MP
- IBM DB2, Sybase ASE, Informix, MySQL, Cassandra, MongoDB
- PostgreSQL and many others

Helped to migrate thousands of databases at AOL from Sybase to MySQL

Today I travel around the world helping Pythian’s customers get the best return on their database investments.
About me Part II

The fun stuff:

I like to travel

I love animals, Nature, National parks, Museums, Zoos and Gardens

I take a lot of pictures of flowers and landscapes. They are patient subjects

I take very few pictures of people. They are almost never patient subjects
Intro

- With the release last year of MongoDB 4.0 support for ACID transactions was announced
- We will talk about;
  - what ACID actually means and how it now applies to the MongoDB platform
  - Why it took so long for MongoDB to release transaction support
  - When you should use transactions
  - When you should not
  - And some general rules about how you should use them
ACID

- **ACID**
  - Atomic – All or none, an operation completes or it doesn’t
  - Consistent – The database remains in a consistent state at all times for all users
  - Isolated – What happens inside a transaction is invisible to other users until the transaction commits
  - Durable – If a transaction successfully commits the work it performed will remain and not be lost
- ACID transactions mean you can update multiple documents within a transaction and be sure that either all updates are committed or none are.
CAP Distributed Database Theorem

- Mentioned here because of that word consistent
- CAP
  - Consistent – all updates are seen on all replicas at the same time
  - Available – The database is always available
  - Partitioned – The database can be accessed even if some nodes are missing
- You can have any two but not all three.
- Mongo Replica Sets follow the CAP theorem
- The Consistent in CAP is usually not considered to be the same Consistent as the Consistent in ACID
MongoDB 4.0 Transaction Feature

- Works only on single replica sets
  - Sharded version scheduled for 4.2
- Is more expensive than operations not using transactions when using Majority consistency
- Based on built in Wild Tiger engine transaction support
- Appears to work similarly to the XA standard for a replica set if write and read concern are set to majority
- Works with eventual consistency if write concern is set to 1.
- Uses MVCC style Snapshot views for reads allowing reads to avoid blocking
Isolation

- MySQL and most other databases offer the concept of Isolation levels
- Example: Read-uncommitted, Read-committed, Repeatable and Serializable
- MongoDB offers two options Read-Uncommitted (which is the default) and Snapshot which is read-committed.
Example Transaction using Python

```python
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()

Or if the transaction needs to fail

    s.abort_transaction();
```
Why did it take so long?

- MongoDB has always supported atomic operations on a single document
- When using MongoDB you should model you should not:
  - Normalize your Tables/Collections
  - Split related data apart unless necessary
- So the need was not as strong as it would normally be for a relational database
- Also transactions can be misused to cause severe concurrency problems in a database
- But….
  - Updates to multiple documents in the same collection aren't protected without transactions
  - Updates to related documents in other collections aren't protected without transactions
Now they are here, How should you use Transactions

- Keep your transactions brief
  - Wild Tiger has to keep a snapshot view of the database throughout the transaction. As the transaction goes longer that can result in significant space consumption
  - The longer a transaction lasts the greater the chance another client request will be made to update documents in the current transaction.
- Never do a transaction that extends across user interactions
- Try not to update more than a few hundred documents in a single transaction
- When updating documents in different collections always use the same sequence of updates in your code
  - Limits the risk of deadlocks
Don’t use transactions to undo what you could easily do in your code.

- Do your field edits first before you start a transaction
- Verify the funds are in the bank before you start your withdrawal
- Make sure you can probably complete the transaction before you start it
- Transaction rollbacks should be uncommon
Don’t do this

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

User Interaction here

    collection_two.insert_one(doc_two, session=s)
    s.commit_transaction()
```
Do this

User Interaction

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()

User Interaction
Don’t Do this

APP 1
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()

APP 2
with client.start_session() as s:
    s.start_transaction()
    collection_two.insert_one(doc_one, session=s)
    collection_one.insert_one(doc_two, session=s)
    s.commit_transaction()
Do this

APP 1
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()

APP 2
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()
Summary

- Works only on single replica sets
- Is more expensive than operations not using transactions when using Majority consistency
- Based on built-in Wild Tiger engine transaction support
- Appears to work similarly to the XA standard for a replica set if write and read concern are set to majority
- Works with eventual consistency if write concern is set to 1.
- Uses MVCC style Snapshot views for reads allowing reads to avoid blocking
Thank You

John Schulz
Schulz@pythian.com
Pythian

LOVE YOUR DATA