AWS re:Invent

NOV. 29 – DEC. 3, 2021 | LAS VEGAS, NV

DAT321

Managing optimistic concurrency in Amazon QLDB

Dan Blaner
Sr. QLDB Specialist SA
AWS

Chris Brack
Sr. Developer Advocate, QLDB
AWS

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Why Amazon Quantum Ledger Database (Amazon QLDB) optimistic concurrency?

Serialized transactions with

- No locks to code
- No locks to degrade performance under load



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

How it works

```
START TRANSACTION
```

Amazon QLDB starts tracking documents in a transaction scope

```
SELECT * FROM Person WHERE GovId = '123-45-6789'
```

Documents with GovId '123-45-6789' added to transaction scope

```
INSERT INTO Person Value ?
```

Document added to transaction scope

```
COMMIT TRANSACTION
```

Amazon QLDB checks to see if any document in the transaction scope has been modified by another transaction

If not, `COMMIT`

If so, `Retry` or `OCC Exception`

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What happens?

```
// Transaction starts
driver.execute(txn -> {
    Result rs1 = txn.execute("select * from Person
where GovId = ?", govId);
    IonStruct person = (IonStruct)
rs1.iterator().next();

    // Do other business logic here
    doStuff(person);

    txn.execute("update Person set status = 'ok' where
GovId = ?", govId);
    sendPersonUpdatedNotification(person);
});
// Transaction ends
```

Re-executed on transaction retry

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

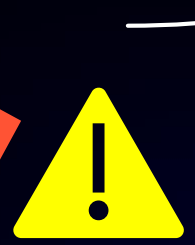
What happens?

```
// Transaction starts
driver.execute(txn -> {
    Result rs1 = txn.execute("select * from Person
where GovId = ?", govId);
    IonStruct person = (IonStruct)
rs1.iterator().next();

    // Do other business logic here
    doStuff(person);

    txn.execute("update Person set status = 'ok' where
GovId = ?", govId);
    sendPersonUpdatedNotification(person);
});
// Transaction ends
```

Re-executed on transaction retry



DANGER!

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Transaction scope

```
SELECT * FROM MyTable
```

- Every document in the table is in your transaction scope
- Any concurrent INSERT/UPDATE/DELETE in that table will give you an OCC

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Transaction scope

```
SELECT * FROM MyTable WHERE not_indexed='bar'
```

- Every document in the table is in your transaction scope
- Any concurrent INSERT/UPDATE/DELETE in that table will give you an OCC

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Transaction scope

```
SELECT * FROM MyTable WHERE index1='foo'
```

- Every document in the table with an index1 value of 'foo' is in your transaction scope
- Any concurrent INSERT/UPDATE/DELETE of a document in that table with that index1 value will give you an OCC
- Does not matter if a matching document already exists

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Transaction scope

```
SELECT * FROM MyTable WHERE index1='foo' AND
not_indexed = 'bar'
```

- Every document in the table with an index1 value of 'foo' is in your transaction scope
- Any concurrent INSERT/UPDATE/DELETE of a document in that table with that index1 value will give you an OCC
- Does not matter if a matching document already exists

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Transaction scope

Transaction 1

```
select * from MyTable where index1 = ‘one' and
index2 = ‘two'
```

Transaction 2

```
insert into MyTable value {'index1’: ‘ten',
'index2’:’two'}
```

This will NOT generate an OCC for Transaction 1 even though index2 is the same; only the left-most indexed field matters

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Transaction scope

Transaction 1

```
select * from MyTable where index1 = ‘one' and
index2 = ‘two'
```

Transaction 2

```
insert into MyTable value {'index1': ‘one',
'index2’:’three'}
```

This will generate an OCC for Transaction 1 because the value of index1 is the same

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Transaction scope

Transaction 1

```
select * from MyTable where foo = ‘bar' and index2
= ‘two'
```

Transaction 2

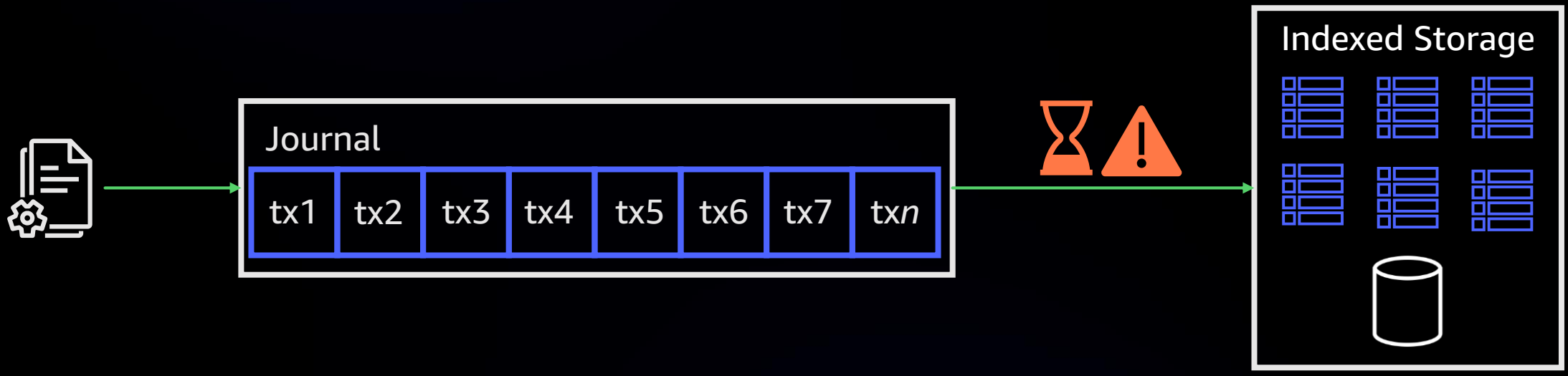
```
insert into MyTable value {'index1’: ‘ten',
'index2’:’two'}
```

This WILL generate an OCC for Transaction 1 even though index2 is not listed first; foo is not indexed, so index2 is the left-most indexed field

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Rare case – Journal to indexed storage lag



© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

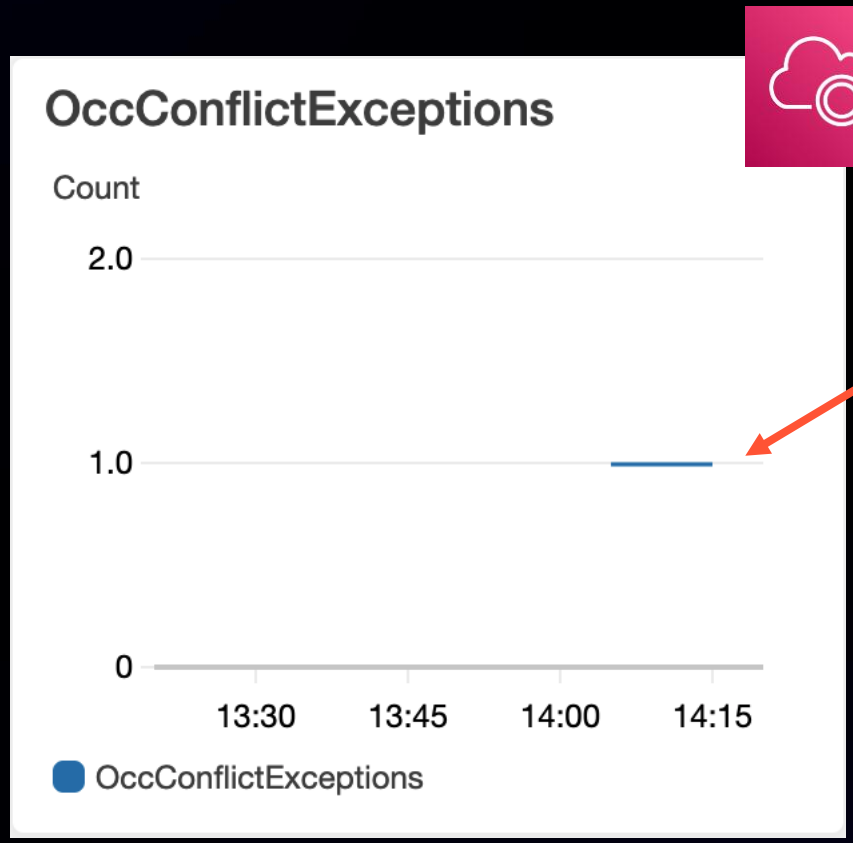
Minimizing OCC conflicts

- OCC conflicts are not inherently bad
- Keep transaction scope small
  - Make queries as targeted as possible
  - Do not scan tables — base all queries on a good index
  - Use unique indexes wherever possible
  - Use high-selectivity indices as next-best
  - Avoid low-selectivity indices that return a lot of documents to scan through

aws

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Managing OCC conflicts – Amazon CloudWatch



Problem? Not necessarily

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Managing OCC conflicts – Query stats

```
driver.execute(txn -> {
    Result rs = txn.execute("select * from Person where GovId = ?", govId);

    if (rs.getConsumedIOs() != null) {
        logger.info("Read IOs:  " + rs.getConsumedIOs().getReadIOs());
    }

    if (rs.getTimingInformation() != null) {
        TimingInformation timing = rs.getTimingInformation();
        logger.info("Query time (ms): " + timing.getProcessingTimeMilliseconds());
    }
});
```



Did you consume more ReadIOs than you think you should have?

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Managing OCC conflicts – App logs

```
2021-10-07 11:47:05,339 [main] DEBUG software.amazon.awssdk.request - Received error response: software.amazon.awssdk.services.qldbsession.model.OccConflictException: Optimistic concurrency control (OCC) failure encountered while committing the transaction (Service: QldbSession, Status Code: 400, Request ID: LeOJ5pMzOTKC6z6JMeiLTz, Extended Request ID: null)
2021-10-07 11:47:05,346 [main] INFO  software.amazon.qldb.QldbDriver - A recoverable error has occurred. Attempting retry #1.
2021-10-07 11:47:05,346 [main] DEBUG software.amazon.qldb.QldbDriver - Errored Transaction ID: O5K97ZHAmG22zbNo6JFyGR. Error cause:
software.amazon.awssdk.services.qldbsession.model.OccConflictException: Optimistic concurrency control (OCC) failure encountered while committing the transaction (Service: QldbSession, Status Code: 400, Request ID: LeOJ5pMzOTKC6z6JMeiLTz, Extended Request ID: null)
        at software.amazon.awssdk.core.internal.http.CombinedResponseHandler.handleErrorResponse(CombinedResponseHandler.java:123)
        at software.amazon.awssdk.core.internal.http.CombinedResponseHandler.handleResponse(CombinedResponseHandler.java:79)
...
```

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Managing OCC conflicts – Do it yourself

```
RetryPolicy policy = RetryPolicy.builder().maxRetries(0).build();

// Transaction starts
try {
    driver.execute(txn -> {
        txn.execute("select * from Person where GovId = ?", govId);
    }, policy);
} catch (OccConflictException e) {
    System.err.println("Ah HAH!");
}
// Transaction ends
```

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

Dan Blaner

dbblaner@amazon.com
linkedin.com/in/danblaner/

Chris Brack

cbrack@amazon.com

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.