SQL Event Store a MVCC

co to jest

Event Store

global_position	event_type	event_data	stream_name	stream_position	created_at
1	orders.item_added_to_basket	...	order$42	0	...
2	orders.submitted	...	order$42	1	...
3	payments.method_chosen	...	payment$123	0	...
4	payments.method_succeeded	...	payment$123	1	...

co to jest

Stream

globalny

global_position	event_type	event_data	stream_name	stream_position	created_at
1	orders.item_added_to_basket	...	order$42	0	...
2	orders.submitted	...	order$42	1	...
3	payments.method_chosen	...	payment$123	0	...
4	payments.method_succeeded	...	payment$123	1	...

nazwany

global_position	event_type	event_data	stream_name	stream_position	created_at
1	orders.item_added_to_basket	...	order$42	0	...
2	orders.submitted	...	order$42	1	...
3	payments.method_chosen	...	payment$123	0	...
4	payments.method_succeeded	...	payment$123	1	...

co to jest

Projection

```
new_state =
  [all_stream_events]
    .reduce(initial_state) do |acc, event|
      ...
    end
```

co to jest

Subscription

```
last_position_marker
previous_state

new_state =
  [stream_events_since_last_position_marker]
    .reduce(previous_state) do |acc, event|
      ...
    end
```

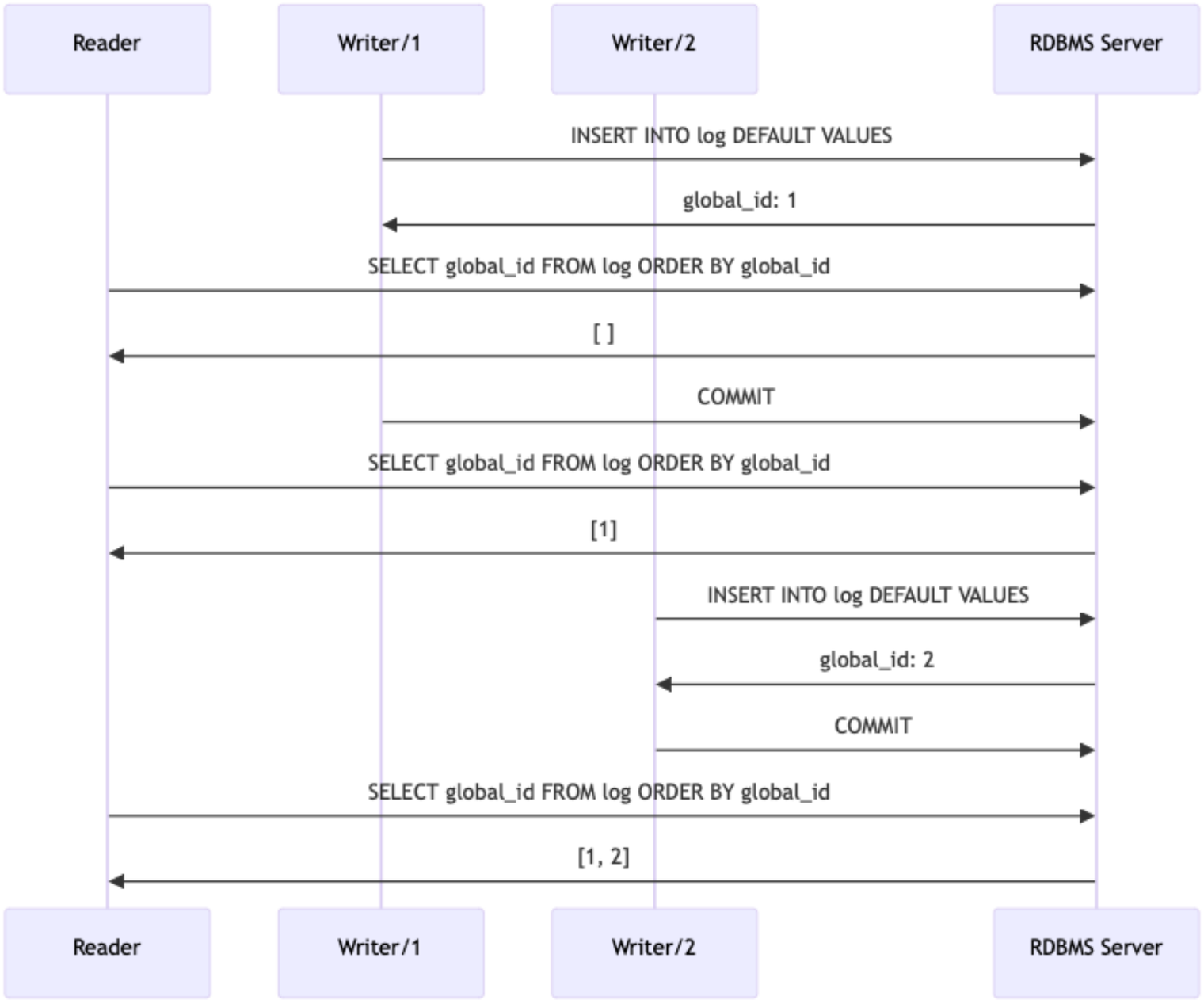
dlaczego

SQL RDBMS

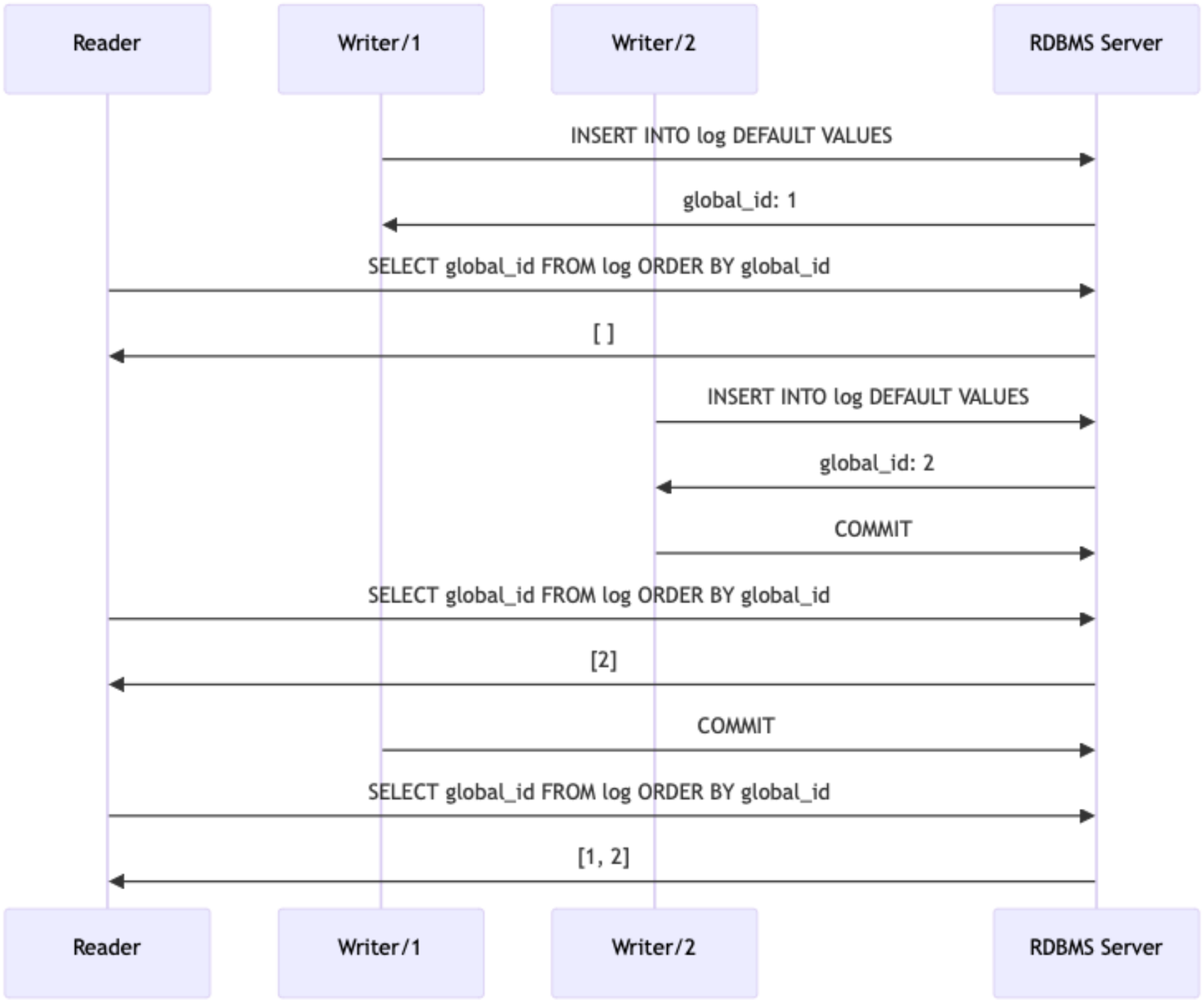
co to jest

Multi–Version Concurrency Control

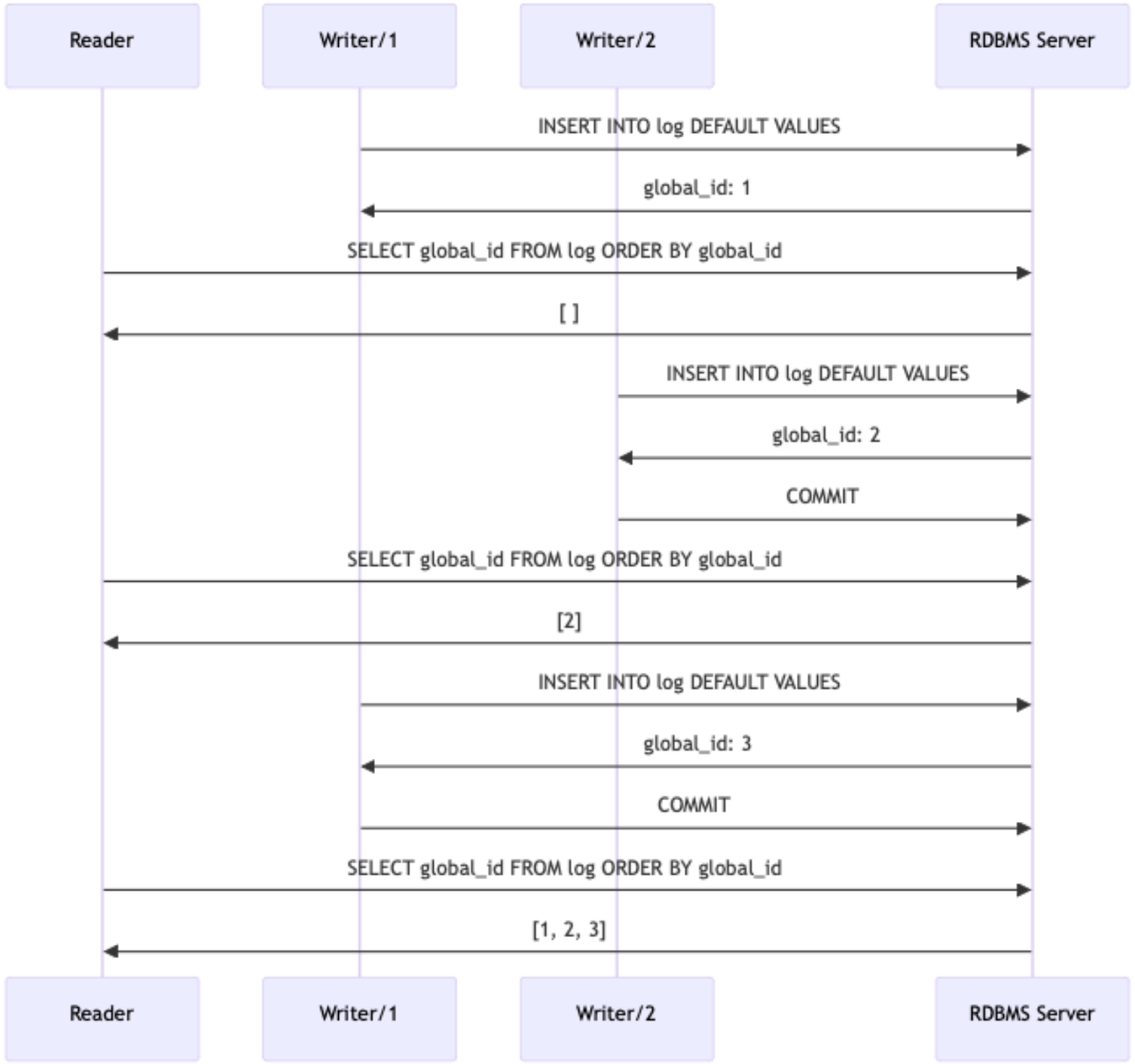
przykład transakcji po sobie (optymista)



przykład transakcji obok siebie (realista)



przykład transakcji obok siebie (pesymista)



kalm

expected_version

panik

expected_version: nil

jakie uszeregowanie zdarzeń

tylko złe odpowiedzi

Write Lock na zapisie

transakcyjna sekwencja (lock for update)

pg_advisory_xact_lock

Re: Sequences, txids, and serial order of transactions

If you are satisfied with the commit order, there is a way to do that with minimal loss of concurrency. As the very last thing before commit, take out an exclusive transactional advisory lock `pg_advisory_xact_lock`

Under cover of that lock, assign the number. You may need to write some custom code for assigning that across multiple backends with the right characteristics (e.g., the database may need to make a request of some external service for the number). There is some actual serialization of this small bit at the end of the transaction, but if you're careful it can be a very small window of time.

https://www.postgresql.org/message-id/CACjxUsMKA6k-mDOdkos3k0i-KE4HFRwkd=PXPArYy4UabTd-LA@mail.gmail.com

Write Lock na odczycie

```
/*
Play a little trick: We very briefly lock the table for  writes in order to
wait for all pending writes to finish. That way, we are sure that there are
no more uncommitted writes with a identifier lower or equal to window_end.
By throwing an exception, we release the lock immediately after obtaining it
such that writes can resume.
*/
    BEGIN
        EXECUTE format('LOCK %s IN EXCLUSIVE MODE', table_to_lock);
        RAISE 'release table lock';
    EXCEPTION WHEN OTHERS THEN
    END;
```

https://www.citusdata.com/blog/2018/06/14/scalable-incremental-data-aggregation/

created_at

transaction_timestamp() / now() / CURRENT_TIMESTAMP

statement_timestamp()

clock_timestamp() / timeofday()

track_commit_timestamp

pg_current_snapshot + pg_current_xact_id

```
CREATE TABLE log (
  id   serial      primary key,
  evid uuid        not null default gen_random_uuid(),
  txid xid8        not null default pg_current_xact_id(),
  time timestamptz not null default now()
)

SELECT id, txid::text
FROM log
WHERE txid > last_txid::xid8
AND txid < pg_snapshot_xmin(pg_current_snapshot())
ORDER BY txid, id
```

gap detection

listen + notify

Write-Ahead Log + Logical Decoding

```
CREATE PUBLICATION log_pub FOR TABLE log;

SELECT * FROM pg_create_logical_replication_slot("log_slot", "test-
decoding");

SELECT * FROM pg_logical_slot_get_changes('log_slot', NULL, NULL, 'include-
xids', '0');

SELECT * FROM pg_logical_slot_peek_changes('log_slot', NULL, NULL, 'include-
xids', '0');
```

unikanie problemu

repozytorium z reprodukcjami

https://github.com/mostlyobvious/sql-log-implementations

suba daj prelegentowi

https://mostlyobviou.us