DATASTAX

ACID-транзакции в Apache Cassandra 5.0

Александр Волочнев

Aleksandr Volochnev

Developer Advocate Lead

dtsx.io/aleks

- IT Exorcist
- Apache Cassandra™ Trainer
- Cloud Architect certified

cassandra

Agenda

- Что есть ACID (КОРОТКО!)
- Проблемы с ACID в распределённых системах
- Protocol Accord
- Использование транзакций

ACID

Atomicity

Атомарность

Всё или ничего

В любой момент времени все мутации транзакции либо ещё не выполнены (не видны в базе), либо уже выполнены (видны все мутации).

Consistency

Согласованность

Транзакция не может перевести данные в логически несогласованное состояние, то есть все мутации должны сработать с учётом существующих внешних ключей, правил, каскадов, триггеров и т.д. и т.п.

Isolation

Изоляция

Параллельно выполняемые транзакции не могут влиять друг на друга. Если транзакции затрагивают одни и те же данные, они могут выполняться только последовательно.

Durability

Устойчивость

После “применения” (COMMIT) мутации данных должны быть применены персистентно, то есть сохранены в *non-volatile memory.*

Немного истории

- **1973** *“IBM Information Management System supported ACID transactions”* (Но термин ещё не введён)
- **1983** Andreas Reuter and Theo Härder вводят акроним ACID
- **2013** PAXOS Apache Cassandra 2.0 получает Light-Weight Transactions (single partition)
- **2023** Accord Apache Cassandra 5.0 получает GPT транзакции

Что не так с Кассандрой?

В 2008 году Facebook первым достиг числа в 100,000,000 пользователей.

Прочитайте это число ещё раз.

Ситуация

- Пользователи по всему миру
- Быстрый интернет
- Огромный объём данных
- Высокие требования SLA

“Bad news, people. There is no database like that.”

Требования

GEO

Пользователи
по всей планете

VOLUME

- Петабайты данных
- Миллионы QPS

SLA: Time

Немедленный ответ

SLA: HA

“Девять девяток”

Архитектурные решения

GEO

Несколько активных ЦОД

VOLUME

- Петабайты данных
- Миллионы QPS

SLA: Time

Немедленный ответ

SLA: HA

“Девять девяток”

Архитектурные решения

GEO

Несколько активных ЦОД

VOLUME

Партиционирование

SLA: Time

Немедленный ответ

SLA: HA

“Девять девяток”

Архитектурные решения

GEO

Несколько активных ЦОД

VOLUME

Партиционирование

SLA: Time

- Денормализация
- Роутинг по ключу

SLA: HA

“Девять девяток”

Архитектурные решения

GEO

Несколько активных ЦОД

VOLUME

Партиционирование

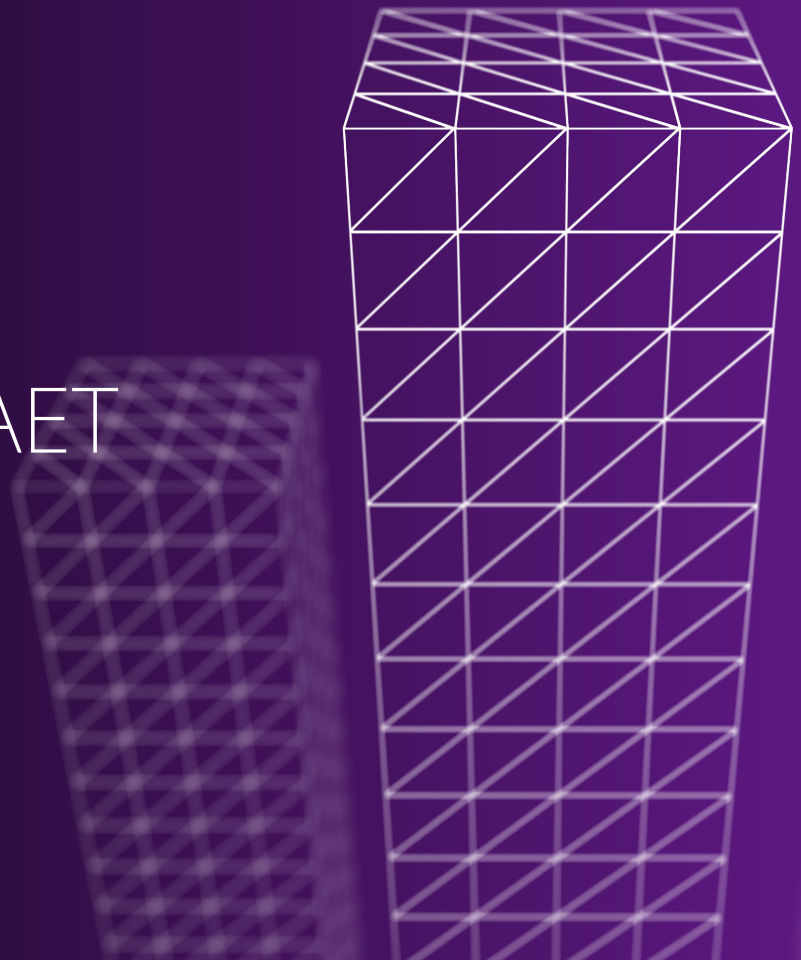
SLA: Time

- Денормализация
- Роутинг по ключу

SLA: HA

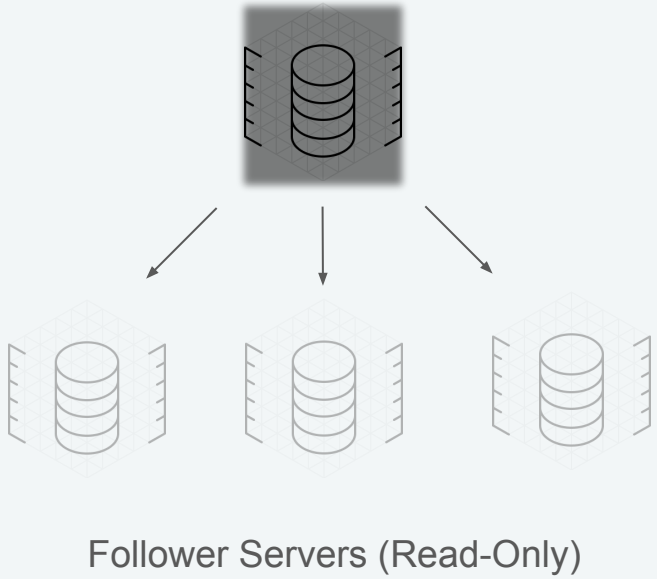
- Репликация
- Децентрализация

КАК ЭТО РАБОТАЕТ



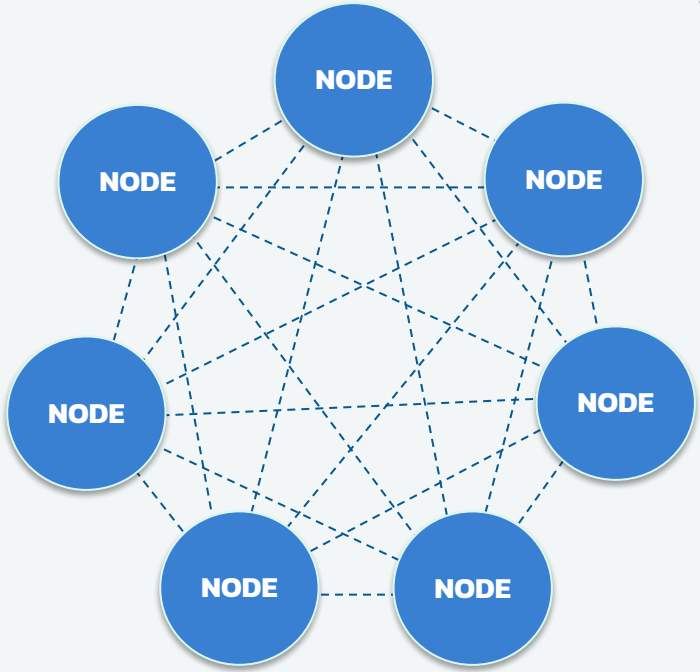
Централизация (Leader-Follower)

1. Single Point of Failure
2. Плохо масштабируется на запись
3. Пишем только на лидера (и надо знать, кто это)



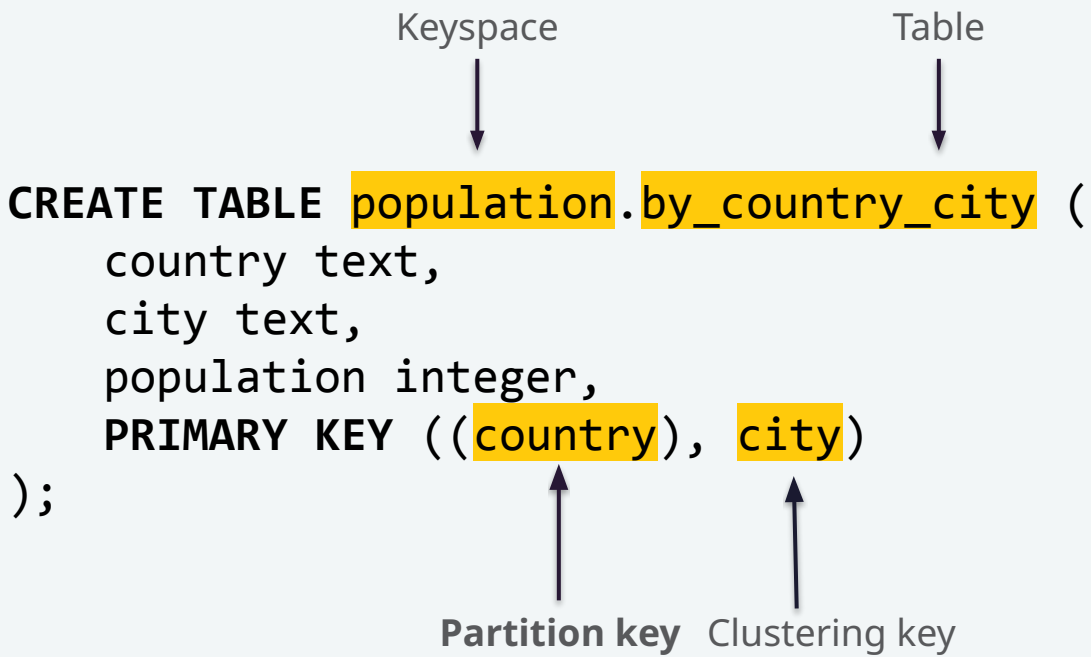
Децентрализация
Leaderless

1. HET Single Point of Failure
2. Масштабируется и на запись
3. Работаем с любым узлом (при аварии просто обращаемся к следующему)



Partitioning

Key-Based Partitioning



Partitioning

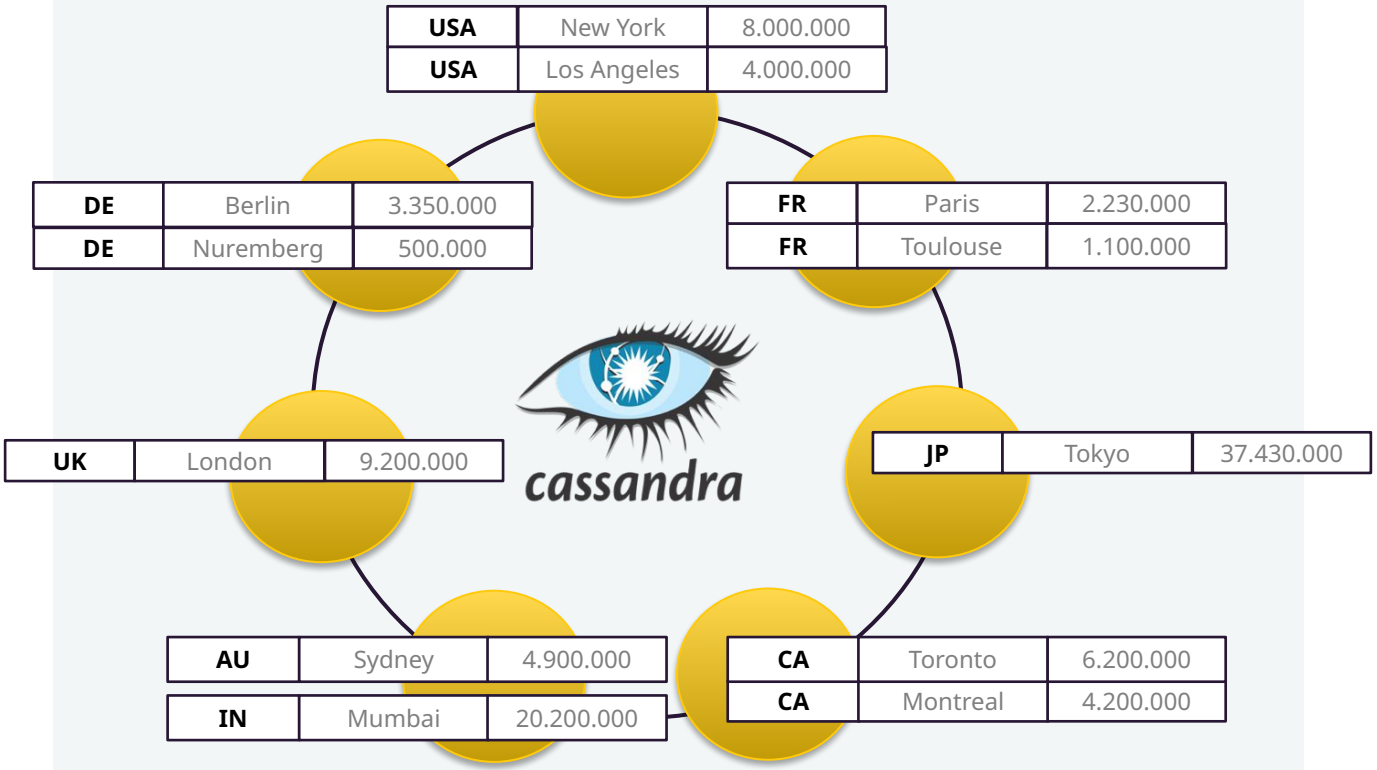
Sample data

Country	City	Population
USA	New York	8.000.000
USA	Los Angeles	4.000.000
FR	Paris	2.230.000
DE	Berlin	3.350.000
UK	London	9.200.000
AU	Sydney	4.900.000
DE	Nuremberg	500.000
CA	Toronto	6.200.000
CA	Montreal	4.200.000
FR	Toulouse	1.100.000
JP	Tokyo	37.430.000
IN	Mumbai	20.200.000

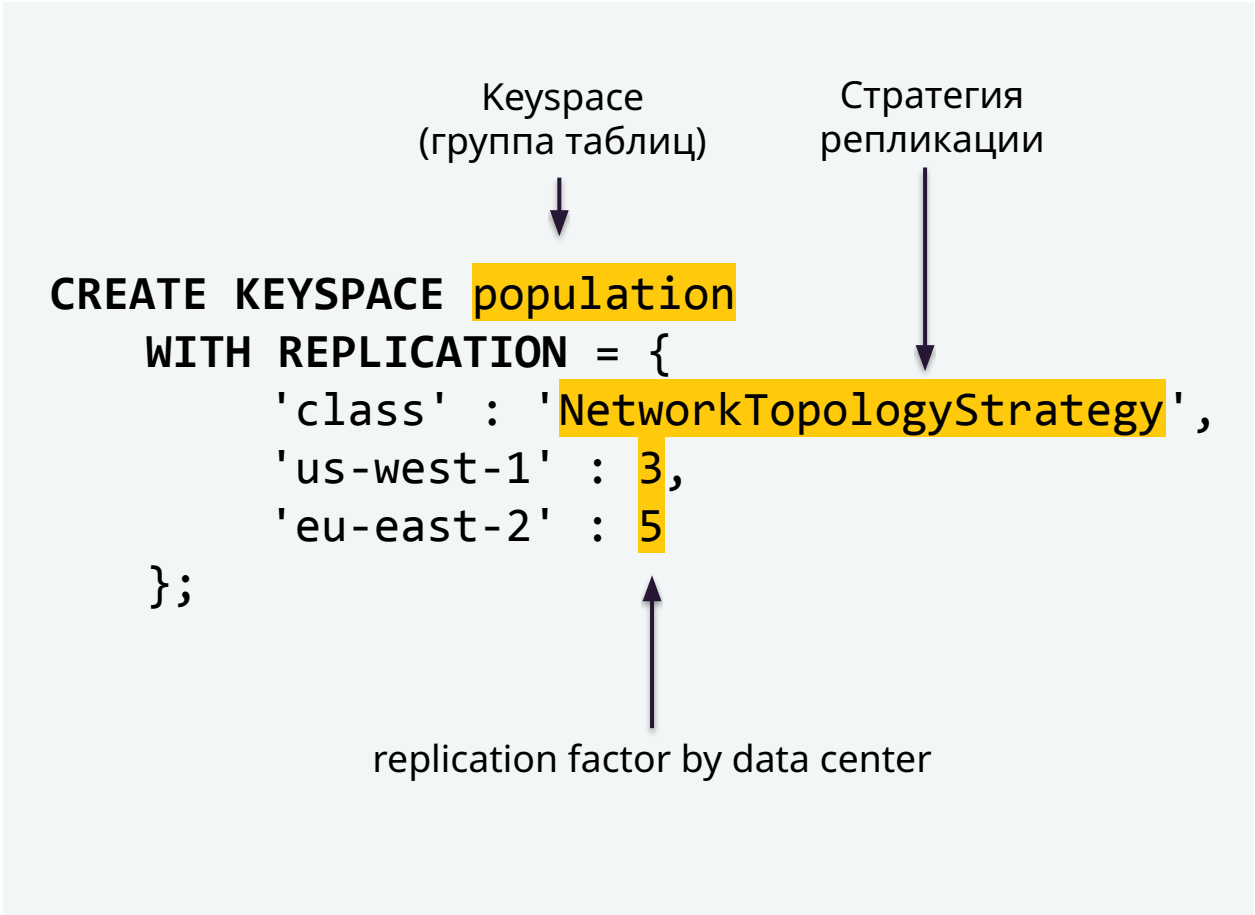
Partition Key

Partitioning

Sample data over 7-nodes DC

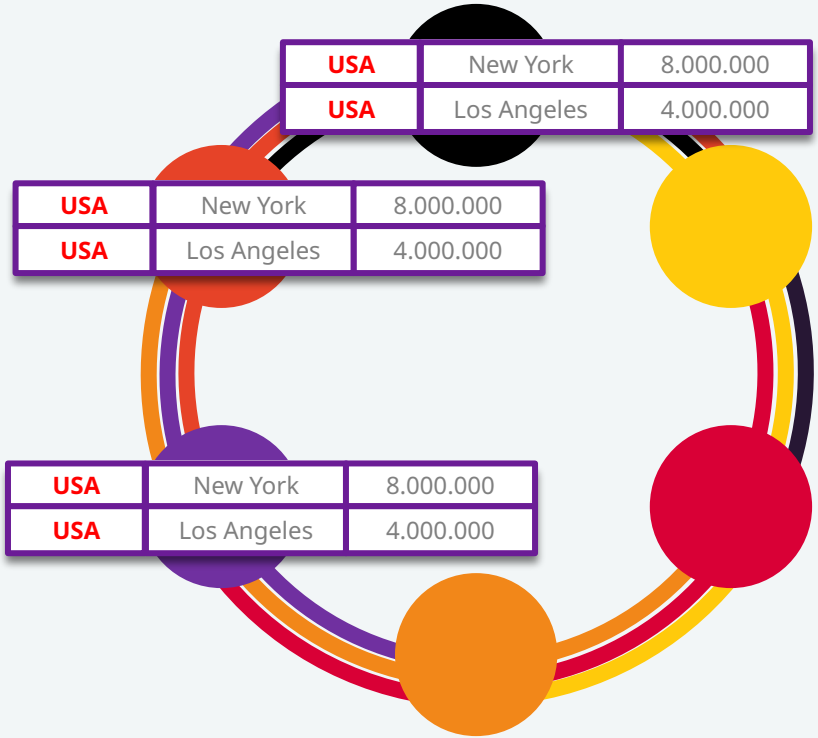


Replication



RF = 3

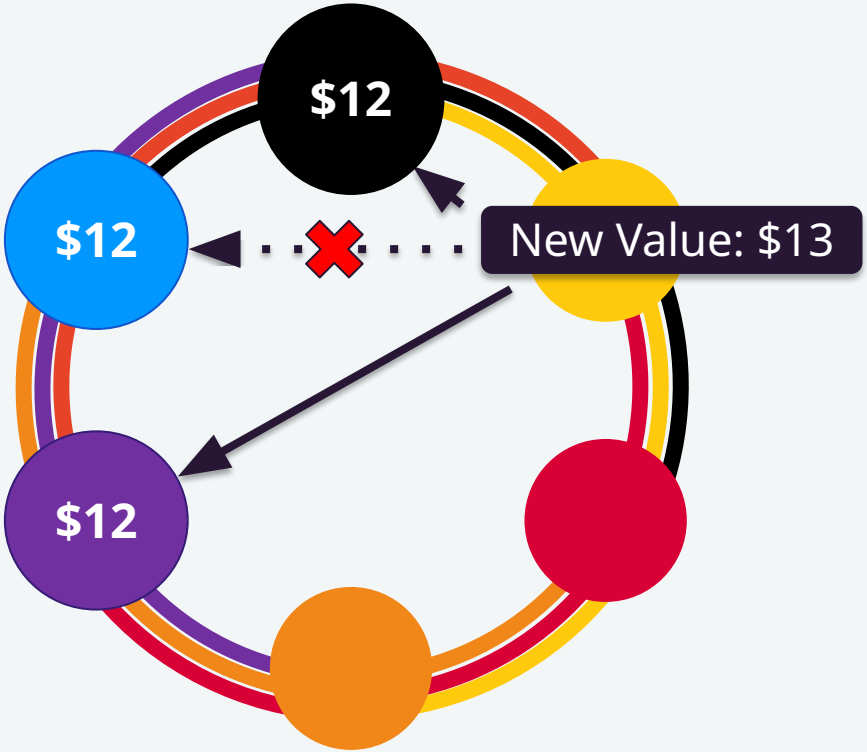
Replication Factor 3 означает, что каждый раздел размещён на трёх разных узлах



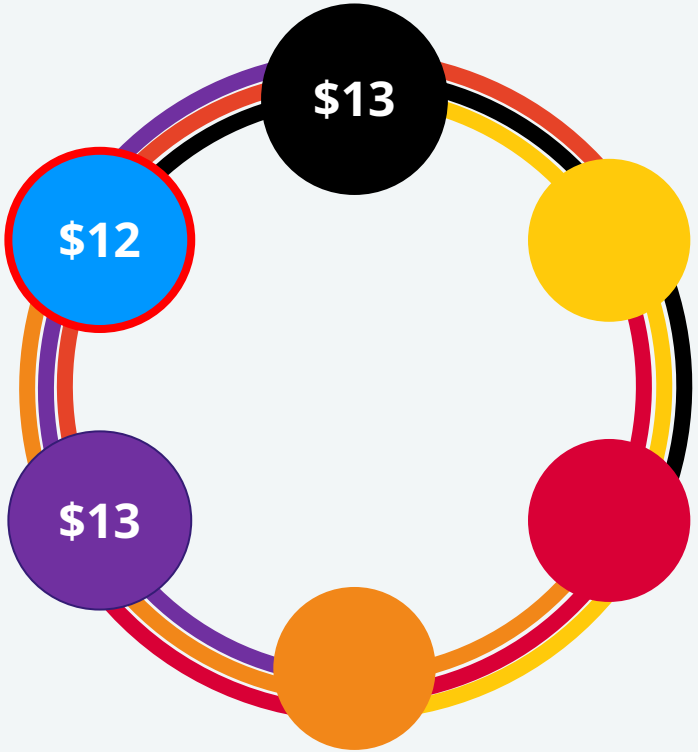
САМАЯ БОЛЬШАЯ ПРОБЛЕМА РЕПЛИКАЦИИ?

1. Дисковое пространство
2. Сетевые операции
3. Потребление вычислительных ресурсов
4. Правильного ответа нет на слайде

Потенциальная неконсистентность



Потенциальная неконсистентность



Почему в Кассандре не было GPT транзакций?

Потому что это, блин, сложно. Крайне сложно при таких требованиях. Кассандра - первая децентрализованная СУБД с leaderless ACID транзакциями.

Cassandra Biggest Users (and Developers)

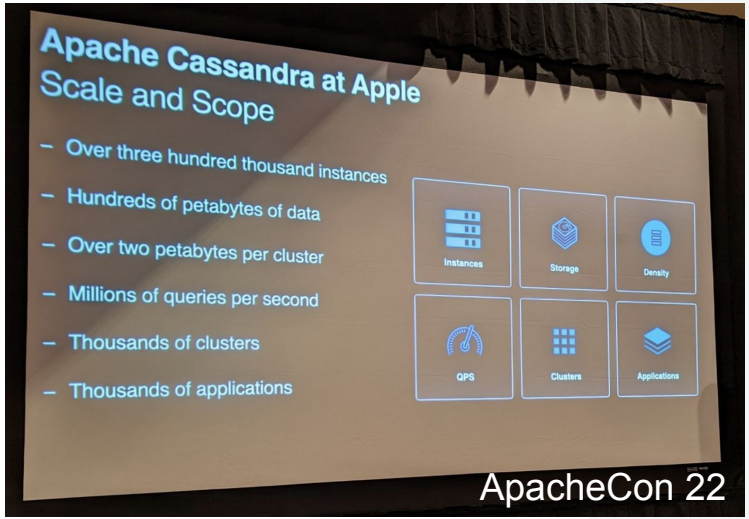
Apache Cassandra @ Netflix

- 98% of streaming data is stored in Apache Cassandra
- Data ranges from customer details to viewing history to billing and payments
- Foundational datastore for serving millions of operations per second

- 30 million ops/sec on most active single cluster
- 500 TB most dense single cluster
- 9216 CPUs in biggest cluster

O(100) Clusters
O(10000) Instances
O(10,000,000) Replications per second
O(100,000,000) Operations per second
O(1,000,000,000,000,000) Petabytes of data

dtsx.io/cassandra-at-netflix



And many others...

Cassandra 2.0 2013 Light-Weight Transactions

Pros

- Paxos - Проверенный протокол
- CAS functionality - Просто
- Гарантируют согласованность

Cons

- Только на один раздел
- Не сериализованные
- Медленно (исправлено в Paxos v2)

```
INSERT INTO cycling.cyclist_name (id, lastname, firstname)
VALUES (4647f6d3-7bd2-4085-8d6c-1229351b5498, 'KNETEMANN', 'Roxxane')
IF NOT EXISTS;
```

```
UPDATE cycling.cyclist_name
SET firstname = ‘Roxane’
WHERE id = 4647f6d3-7bd2-4085-8d6c-1229351b5498
IF firstname = ‘Roxxane’;
```

Cassandra 1.2 2013 Batches

Принцип: если с первого раза не дошло, просто повтори ещё сто раз :)

Pros

- Надёжные
- Гарантируют запись*

Cons

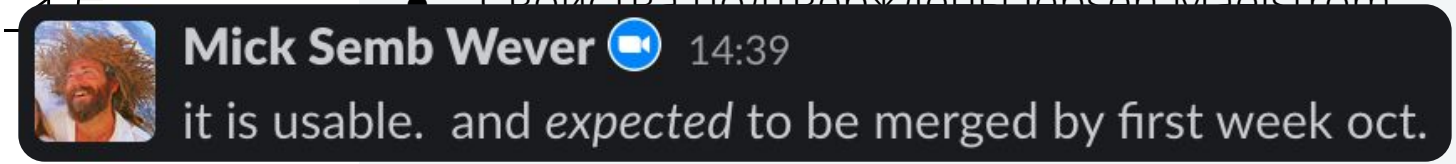
- Не атомарные*
- Не изолированные! (multi-partition)

```
BEGIN LOGGED BATCH
  INSERT INTO cycling.cyclist_names (cyclist_name, race_id) VALUES ('Vera ADRIAN', 100);
  INSERT INTO cycling.cyclist_by_id (race_id, cyclist_name) VALUES (100, 'Vera ADRIAN');
APPLY BATCH;
```

Leaderless SMR (Distributed State Machine)

- Совместный проект Apple и University of Michigan
- Официально принят в 5.0
- Но ещё не в trunk, ожидается к *alpha2*
- На данный момент в стадии финальной доработки
- Свойства подтверждены Jepsen Maelstrom

CEP-15



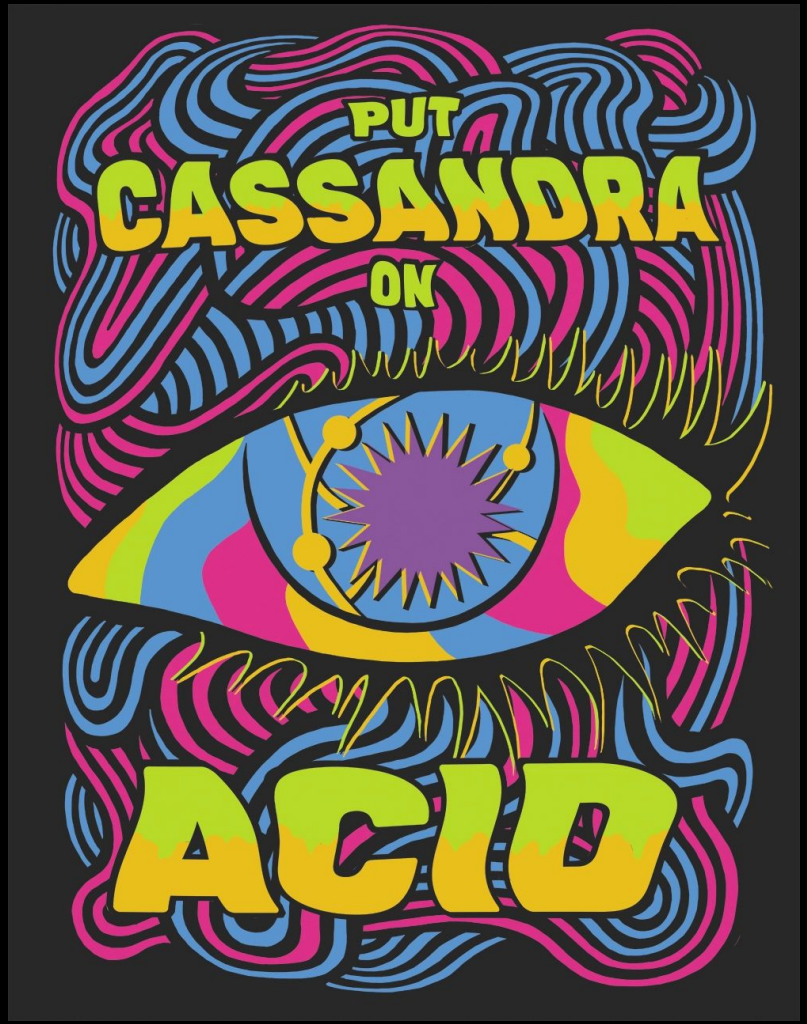
- Strict-Serializable Consistency *(Serializable AND Linearizable)*
- 1 round trip 🤞 (fast path)
- No bottlenecks
- Live Migration from Paxos

CEP-15: General Purpose Transactions

Goals

- General purpose transactions (may operate over any keys in the database at once)
- Strict-serializable isolation
- Optimal latency: one wide area round-trip for all transactions under normal conditions
- Optimal failure tolerance: latency and performance should be unaffected by any minority of replica failures
- Scalability: there should be no bottleneck introduced
- Should have no intrinsic limit to transaction complexity
- Must work on commodity hardware
- Must support live migration from Paxos

TL;DR ACID Transactions in Cassandra


PUT
CASSANDRA
ON
ACID

Protocol Accord

Accord

Accord (computer science)

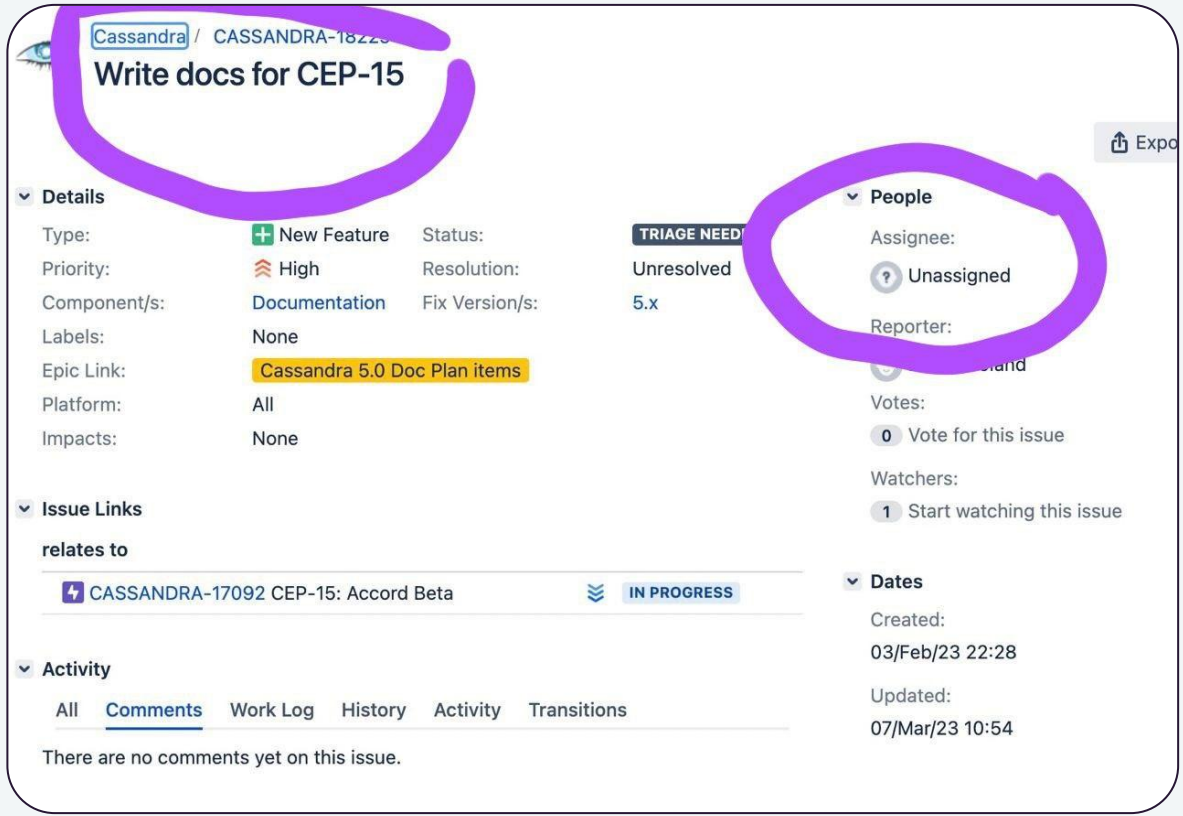
Article Talk Tools

From Wikipedia, the free encyclopedia

Wikipedia does not have an article with this exact name. Please search for *Accord (computer science)* in Wikipedia to check for alternative titles or spellings.

- You need to log in or create an account to create this page.
- Search for "*Accord (computer science)*" in existing articles.
- Look for pages within Wikipedia that link to this title.

Документация



Accord

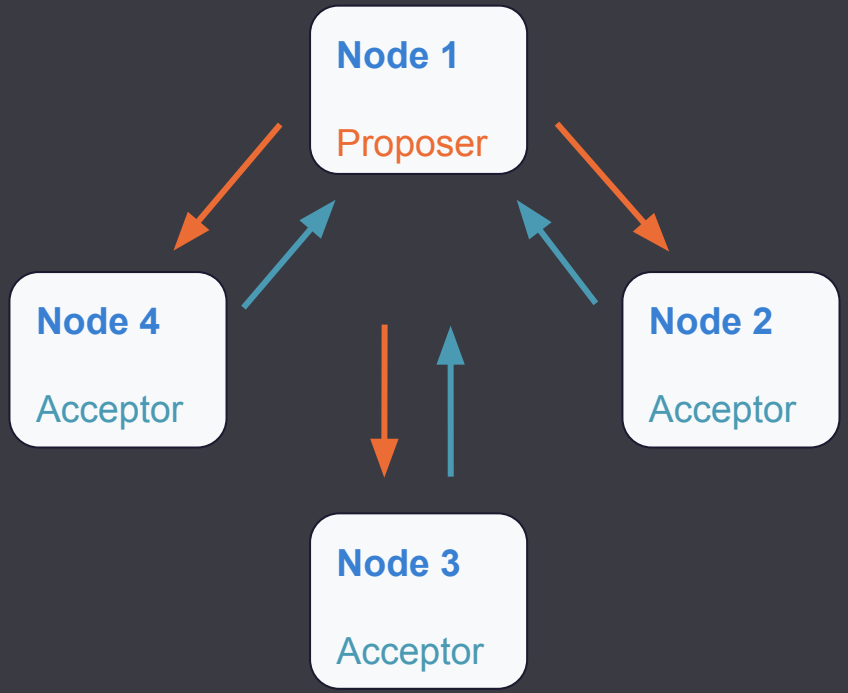
- Принципиально новый алгоритм
- Leaderless (petabytes of data over thousands of nodes)
- Основах на таймстампах (принцип Lamport Clock)
- Reorder Buffer (для быстрого консенсуса)
- Fast Path Electorates (для отказоустойчивости)

“To the best of our knowledge, no commercial or open-source database systems offer strict-serializable transactions across regions in a single wide area round-trip”

Accord Whitepaper

Существующие опции

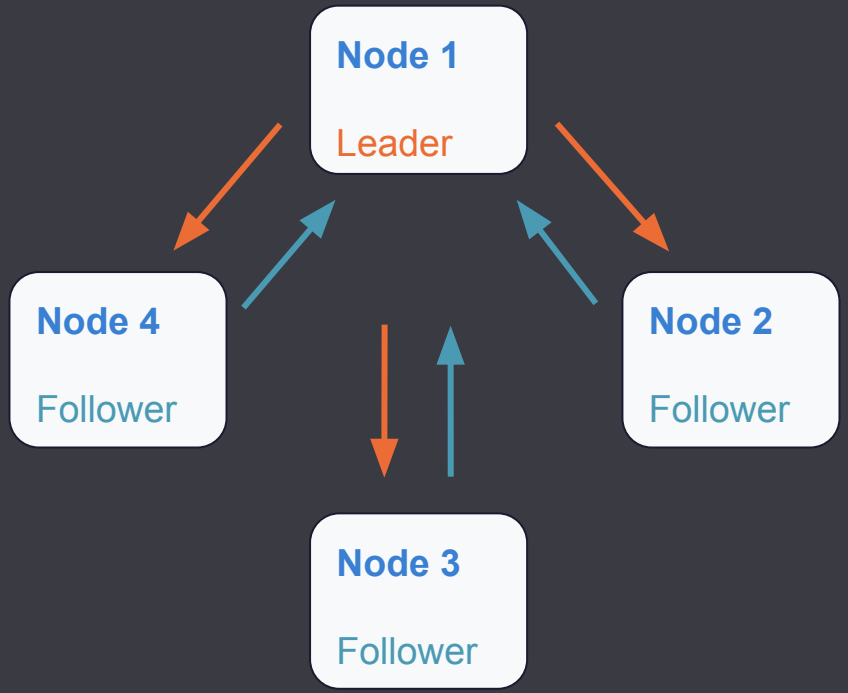
Paxos



- Origin of most consensus protocols
- **Proposer**: I want to do something
- **Acceptor**: Ok or Nope
- Do that until you have majority
- Network round trips add up
- Used in LWT(multi-paxos)

1989

Raft

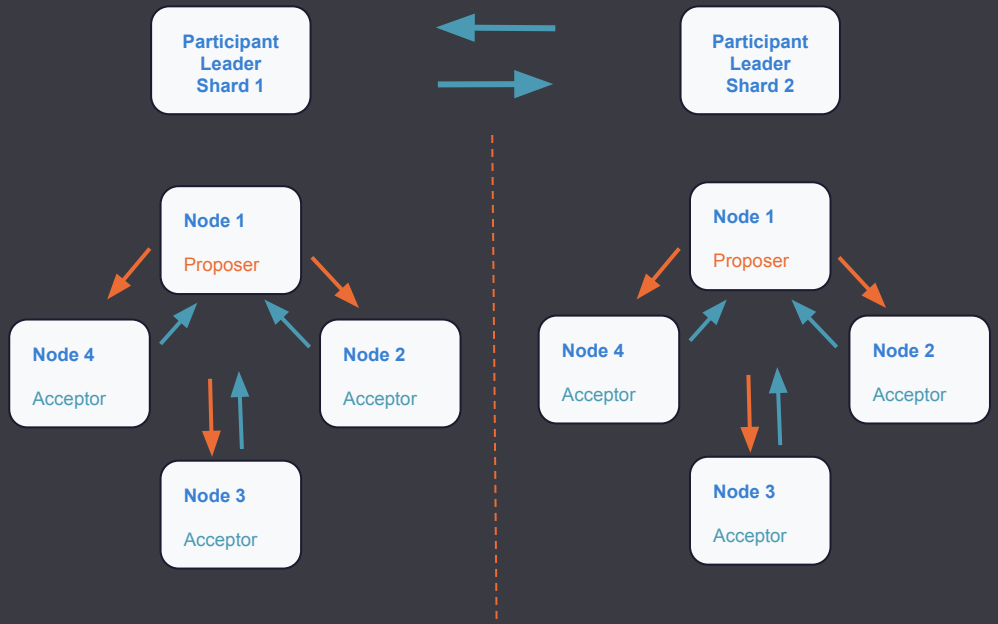


- Leader election to eliminate round trips
- **Leader**: All changes through me
- **Follower**: I trust dear leader

Bad for Cassandra

- Failure Modes lead to latency
- Multi-datacenter? Nope

Google Spanner

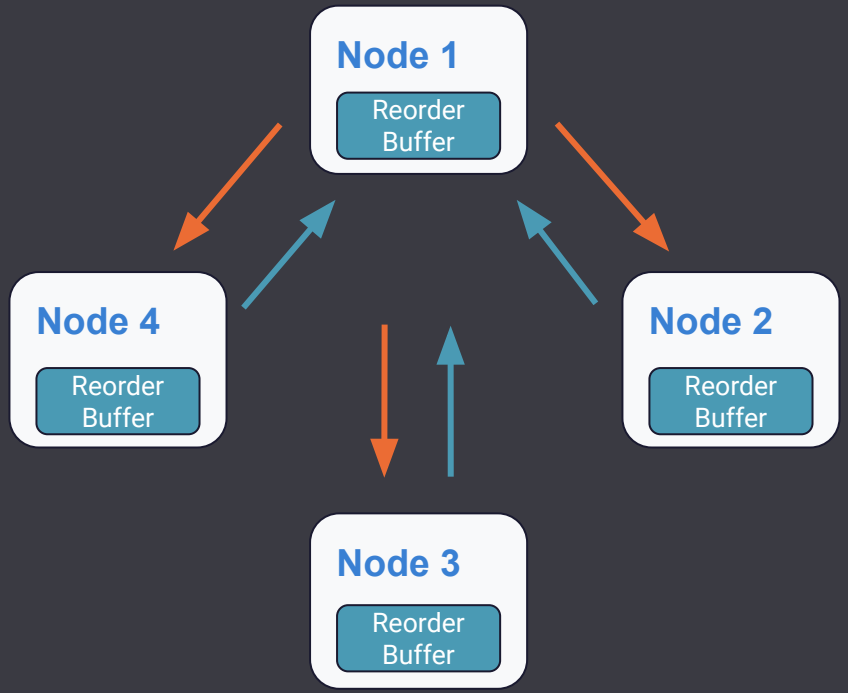


- One paxos group per shard
- Leaders coordinate for 2pc
- All depends on TrueTime™

Bad for Cassandra

- Depends on TrueTime™
- Many hops for one insert

Accord



- Every node has a Reorder Buffer
- Clock skew is cool
- Leaderless timestamp protocol
- Fast Path Electorates: Fault tolerance
- TL;DR One Round Trip - ish

Good for Cassandra

- Leaderless
- Scales like Cassandra
- Failure modes match

Значимые Особенности

Lamport Clock

- Алгоритм типа “Логические Часы”
- Разработан Лэсли Лэмпортом в 1978 году
- Решает проблему синхронизации узлов
  (полностью синхронизировать георасределённые узлы практически невозможно)
- Позволяет определить порядок событий в распределённой системе
  (Обеспечивая сериализуемость)
- Необходим, поскольку реальный порядок получения сообщений может отличаться от “идеального”

Принцип часов Лэмпорта

- Процесс А увеличивает счётчик для каждого события
- Процесс А, отправляя уведомление о событии, прикладывает к нему значение своего счётчика
  (В случае с Accord - ещё и свой уникальный идентификатор)
- Получая уведомление, процесс Б обновляет соответствующий счётчик.

Reorder Buffer

- Кластер определяет отставание часов между всеми репликами, учитывая и сетевые задержки.
- Транзакции буферизуются на реплике на период этого отставания и максимальную задержку (минус задержка между координатором транзакции и самой репликой).
- Транзакции обрабатываются только когда все “ожидаемые” потенциально конфликтующие транзакции уже “доехали” и попали в буфер.
- Транзакции обрабатываются по очерёдности часов Лэмпорта. Использование Reorder Buffer на каждой реплике позволит координатору транзакции максимизировать использование Fast Path.

Fast Path Electorates

- Большинство протоколов консенсуса использую простую модель подсчёта голосов. Этот принцип использован и в Accord.
- В дополнение к этому, представлена идея Electorate - те реплики, чьи голоса будут учитываться для целей fast-path голосования.
- Обычно все реплики являются участниками и простого, и fast-path electorates, но для целей fast-path некоторые реплики могут выводиться из electorate, уменьшая количество необходимых голосов.
- Например, replica set из 9 узлов может определить electorate из 9, 7 или 5 узлов для fast-path quorums из 7, 6 и 5 соответственно.
- Electorate из 9 может принять fast-path решения с менее чем 3 недоступными узлами,после чего electorate должен быть уменьшен до 7 или даже 5.

Алгоритм

Fast Path

```
Coordinator C:
    Send PreAccept(X, t0) to replicas of all shards

Replica R:
    if (have witnessed a newer conflicting timestamp) then
        t = some new higher timestamp issued by R
    else
        t = t0
    PreAccepted[X] = true
    Reply (t, deps = {conflicting transactions where t0 < t})

Coordinator C (with at least a simple quorum from each shard):
    If (a fast-path quorum from each shard had t = t0) then
        send Commit(X, t0, t0, union of all deps)
        go to Execution
    ...
```

Execution

```
Replica R receiving Commit(X, deps):
    Committed[X] = true

Coordinator C:
    send a read to one or more local replicas of each shard
        (containing those deps that apply on the shard)

Replica R receiving Read(X, t, deps):
    Wait for deps to be committed
    Wait for deps with a lower t to be applied locally
    Reply with result of read

Coordinator C (with a response from each shard):
    result = execute(read responses)
    send Apply(result) to all replicas of each shard
    send result to client

Replica R receiving Apply(X, t, deps, result):
    Wait for deps to be committed
    Wait for deps with a lower t to be applied locally
    Apply result locally
    Applied[X] = true
```

Slow Path

```
Coordinator C:
    ... else
        t = maximum t from responses
        send Accept(X, t0, t, deps) to replicas of all shards

Replica R receiving Accept(X, t0, t, deps):
    Accepted[X] = true
    Reply (deps = {conflicting transactions where t0 < t})

Coordinator C (with a simple quorum from each shard):
    send Commit(X, t0, t0, union of all deps)
    go to Execution
```

Recovery

```
Coordinator C:
    send Recover(X) to replicas of each shard

Replica R receiving Recover(X):
    Ensure X is PreAccepted; if only PreAccepted compute deps
    Wait = {Accepted transactions with lower t0 but higher t than X}
    Superceding = {Accepted transactions with higher t0 that did not witness X}
              ∪ {Committed transactions with higher t than t0 of X, that did not witness
X}
    Reply (local state for X, Wait, Superceding)

Coordinator C (with a quorum of responses from each shard):
    If (any R in responses had X as Applied, Committed, or Accepted)
       then continue the state machine from there
    Otherwise
        If (any shard has insufficient R where t = t0 to have taken the fast path)
            then propose the highest t of any response on the Slow Path
        If (any R.Superceding is non-empty)
            then propose the highest t in any response on the Slow Path
        If (any R.Wait is non-empty)
            then wait for them to commit and retry
        Otherwise propose t0 on the Slow Path
```

Использование транзакций

Transaction Syntax - Boundaries

```
BEGIN TRANSACTION
   LET <tuple> = ( SELECT <column1>,<column2>..  FROM <table> WHERE <condition);

   SELECT <return_column>  FROM <table> WHERE <condition>;

   IF <tuple_condition> THEN
       UPDATE|INSERT|DELETE..
   END IF
COMMIT TRANSACTION;
```

Transaction Syntax - State collection

```
BEGIN TRANSACTION
   LET <tuple> = (SELECT <column1>,<column2>.. FROM <table> WHERE <condition);

   SELECT <return_column>  FROM <table> WHERE <condition>;

   IF <tuple_condition> THEN
       UPDATE|INSERT|DELETE..
   END IF
COMMIT TRANSACTION;
```

Transaction Syntax - Return value

```
BEGIN TRANSACTION
   LET <tuple> = ( SELECT <column1>,<column2>..  FROM <table> WHERE <condition);

   SELECT <return_column> FROM <table> WHERE <condition>;

   IF <tuple_condition> THEN
       UPDATE|INSERT|DELETE..
   END IF
COMMIT TRANSACTION;
```

Transaction Syntax - Conditional mutation

```
BEGIN TRANSACTION
   LET <tuple> = ( SELECT <column1>,<column2>..  FROM <table> WHERE <condition);

   SELECT <return_column>  FROM <table> WHERE <condition>;

   IF <tuple_condition> THEN
       UPDATE|INSERT|DELETE..
   END IF
COMMIT TRANSACTION;
```

ПРИМЕР
Inventory Management

Multi-Table/Multi-Partition Exclusive Update

Inventory Management - Setup

```
CREATE TABLE ks.products (
   item text,
   inventory_count int,
   PRIMARY KEY (item)
);
```

```
CREATE TABLE ks.shopping_cart (
   user_name text,
   item text,
   item_count int,
   PRIMARY KEY (user_name, item)
);
```

```
INSERT INTO ks.products(item, inventory_count) VALUES ('PlayStation 5', 100);
```

Inventory Management - Pre-Condition

```
BEGIN TRANSACTION
   // Find out how many PlayStations are left
   LET inventory = (SELECT inventory_count FROM ks.products WHERE item='PlayStation 5');

   // Return the inventory count before deducting
   SELECT item, inventory_count FROM ks.products WHERE item='PlayStation 5';

   // Take a PlayStation out of inventoryand put in users shopping cart
   IF inventory.inventory_count >0 THEN
       UPDATE ks.products SET inventory_count -= 1 WHERE item='PlayStation 5';
       INSERT INTO ks.shopping_cart(user_name, item, item_count)VALUES ('patrick', 'PlayStation 5', 1);
   END IF
COMMIT TRANSACTION;
```

Inventory Management - Output Previous State

```
BEGIN TRANSACTION
   // Find out how many PlayStations are left
   LET inventory = (SELECT inventory_count FROM ks.products WHERE item='PlayStation 5');

   // Return the inventory count before deducting
   SELECT item, inventory_count FROM ks.products WHERE item='PlayStation 5';

   // Take a PlayStation out of inventoryand put in users shopping cart
   IF inventory.inventory_count >0 THEN
       UPDATE ks.products SET inventory_count -= 1 WHERE item='PlayStation 5';
       INSERT INTO ks.shopping_cart(user_name, item, item_count)VALUES ('patrick', 'PlayStation 5', 1);
   END IF
COMMIT TRANSACTION;
```

Inventory Management - Conditional Statement

```
BEGIN TRANSACTION
   // Find out how many PlayStations are left
   LET inventory = (SELECT inventory_count FROM ks.products WHERE item='PlayStation 5');

   // Return the inventory count before deducting
   SELECT item, inventory_count FROM ks.products WHERE item='PlayStation 5';

   // Take a PlayStation out of inventory and put in users shopping cart
   IF inventory.inventory_count > 0 THEN
       UPDATE ks.products SET inventory_count -= 1 WHERE item='PlayStation 5';
       INSERT INTO ks.shopping_cart(user_name, item, item_count) VALUES ('patrick', 'PlayStation 5', 1);
   END IF
COMMIT TRANSACTION;
```

Real Atomic Batch

Foreign Key Management

Real Atomic Batch - Setup

```
CREATE TABLE ks.user (
   user_id UUID,
   email text,
   country text,
   city text,
   PRIMARY KEY (user_id)
);
```

```
CREATE TABLE ks.user_by_email (
   email text,
   user_id UUID,
   PRIMARY KEY (email)
);
```

```
CREATE TABLE ks.user_by_location (
   country text,
   city text,
   user_id UUID,
   PRIMARY KEY ((country, city), user_id)
);
```

Real Atomic Batch - Existence Check

```
BEGIN TRANSACTION
   // Find any existing users with same email
   LET existCheck = (SELECT user_id FROM ks.user_by_email WHERE email='patrick@datastax.com');

   // If email isn't in use, then add the new user
   IF existCheck IS NULL THEN
       INSERT INTO ks.user(user_id, email, country, city)
       VALUES (94813846-4366-11ed-b878-0242ac120002, 'patrick@datastax.com', 'US', 'Windsor');

       INSERT INTO ks.user_by_email(email, user_id)
       VALUES ('patrick@datastax.com', 94813846-4366-11ed-b878-0242ac120002);

       INSERT INTO ks.user_by_location(country, city, user_id)
       VALUES ('US', 'Windsor', 94813846-4366-11ed-b878-0242ac120002);
   END IF
COMMIT TRANSACTION ;
```

Real Atomic Batch - Execution

```
BEGIN TRANSACTION
   // Find any existing users with same email
   LET existCheck = (SELECT user_id FROM ks.user_by_email WHERE email='patrick@datastax.com');

   // If email isn't in use, then add the new user
   IF existCheck IS NULL THEN
       INSERT INTO ks.user(user_id, email, country, city)
       VALUES (94813846-4366-11ed-b878-0242ac120002, 'patrick@datastax.com', 'US', 'Windsor');

       INSERT INTO ks.user_by_email(email, user_id)
       VALUES ('patrick@datastax.com', 94813846-4366-11ed-b878-0242ac120002);

       INSERT INTO ks.user_by_location(country, city, user_id)
       VALUES ('US', 'Windsor', 94813846-4366-11ed-b878-0242ac120002);
   END IF
COMMIT TRANSACTION ;
```

DEMO

DATASTAX

СПАСИБО!

Давайте найдёмся после доклада!

Aleks Volochnev

IT Exorcist | Developer Advocate Lead

dtsx.io/aleks