Data Council Austin '24

tinybird

# Events Sourcing with Kafka at Scale

**Alejandro Martin**
@alejandromav_

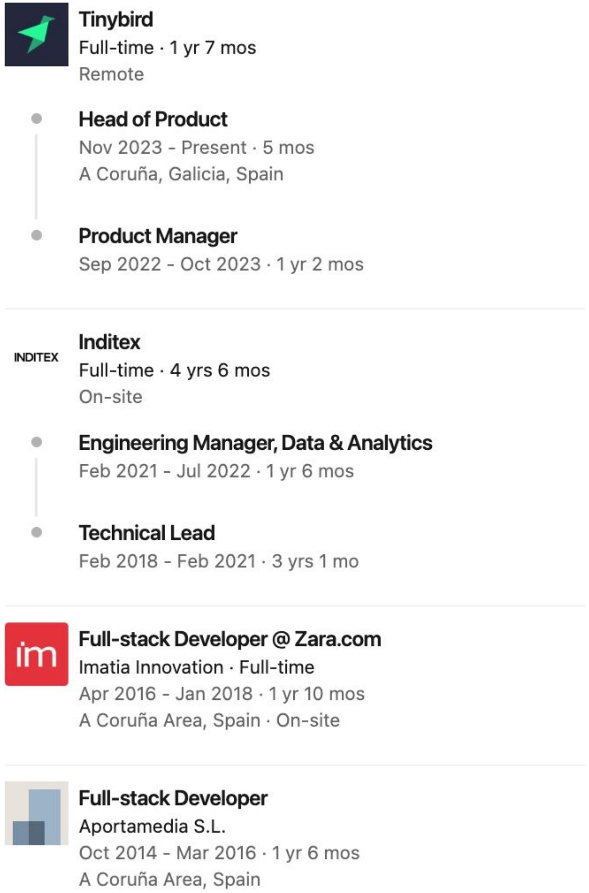
Austin Data Council '24 — Event Sourcing with Kafka at Scale

## Great to meet you!

**Alejandro Martin**

Head of Product @ Tinybird

@alejandromav_



**Tinybird**
Full-time · 1 yr 7 mos
Remote

**Head of Product**
Nov 2023 - Present · 5 mos
A Coruña, Galicia, Spain

**Product Manager**
Sep 2022 - Oct 2023 · 1 yr 2 mos

**Inditex**
Full-time · 4 yrs 6 mos
On-site

**Engineering Manager, Data & Analytics**
Feb 2021 - Jul 2022 · 1 yr 6 mos

**Technical Lead**
Feb 2018 - Feb 2021 · 3 yrs 1 mo

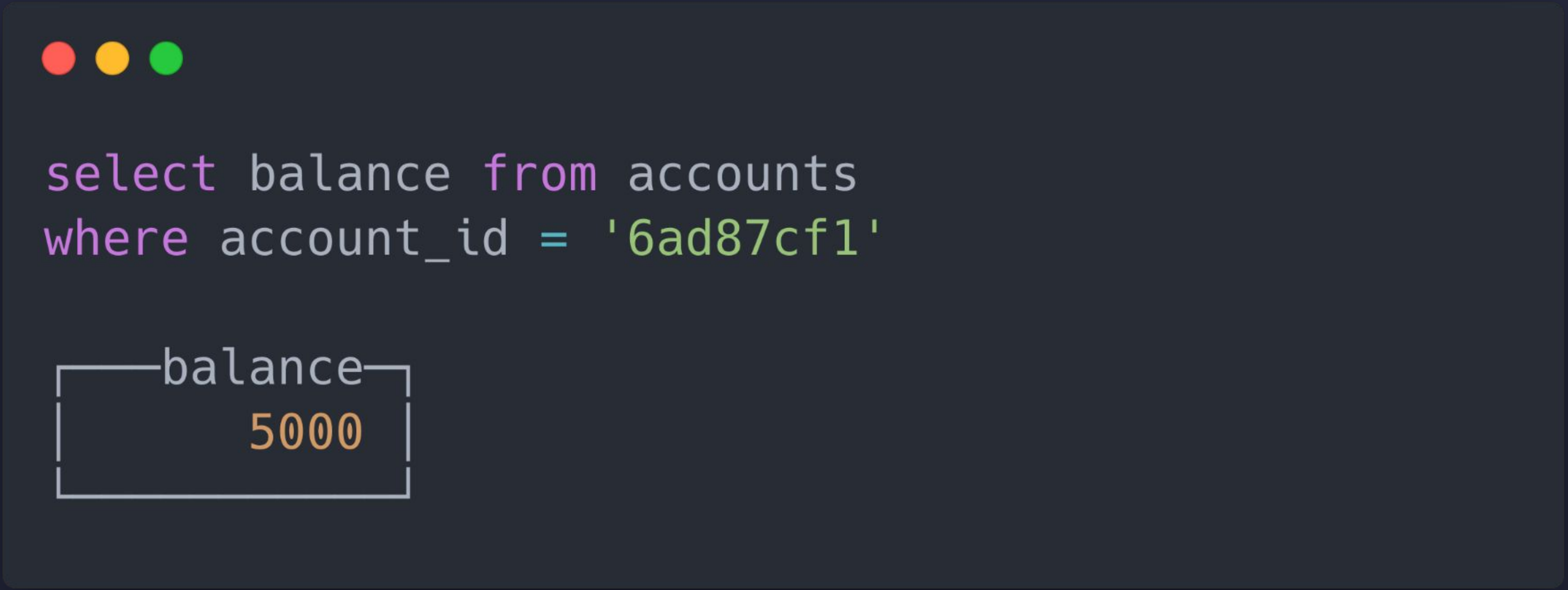
**Full-stack Developer @ Zara.com**
Imatia Innovation · Full-time
Apr 2016 - Jan 2018 · 1 yr 10 mos
A Coruña Area, Spain · On-site

**Full-stack Developer**
Aportamedia S.L.
Oct 2014 - Mar 2016 · 1 yr 6 mos
A Coruña Area, Spain

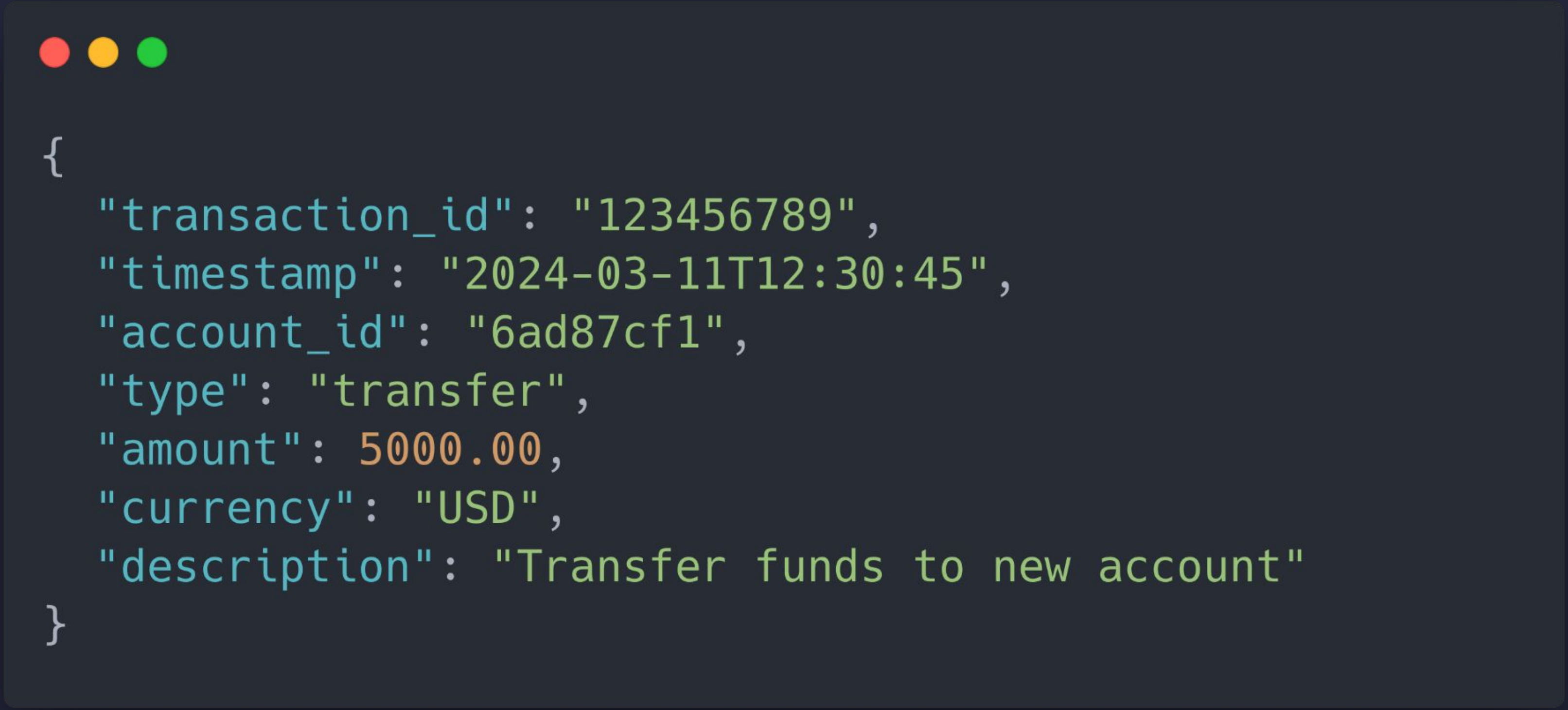
## Disclaimer

## Intro.
## Event Sourcing in a nutshell

## Storing changes as a sequence of immutable events



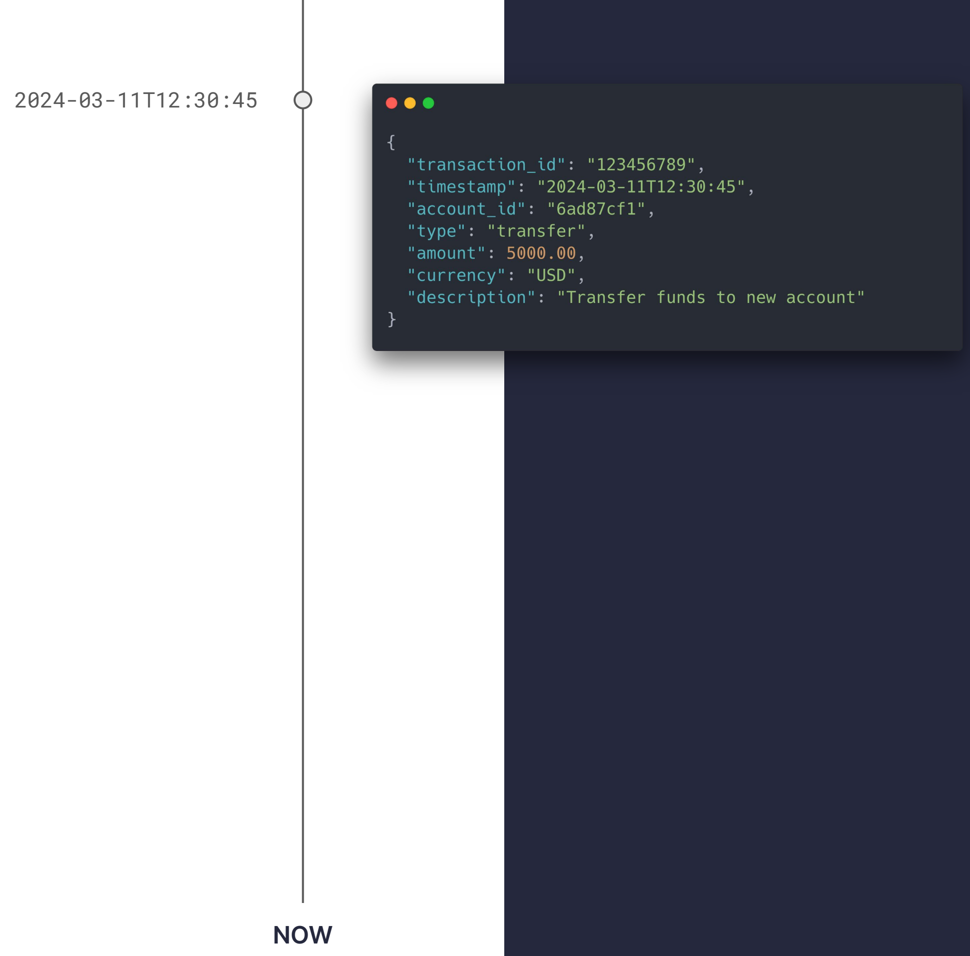
```
select balance from accounts
where account_id = '6ad87cf1'

┌───balance─┐
│      5000 │
└───────────┘
```



```json
{
  "transaction_id": "123456789",
  "timestamp": "2024-03-11T12:30:45",
  "account_id": "6ad87cf1",
  "type": "transfer",
  "amount": 5000.00,
  "currency": "USD",
  "description": "Transfer funds to new account"
}
```

Event Sourcing in a nutshell

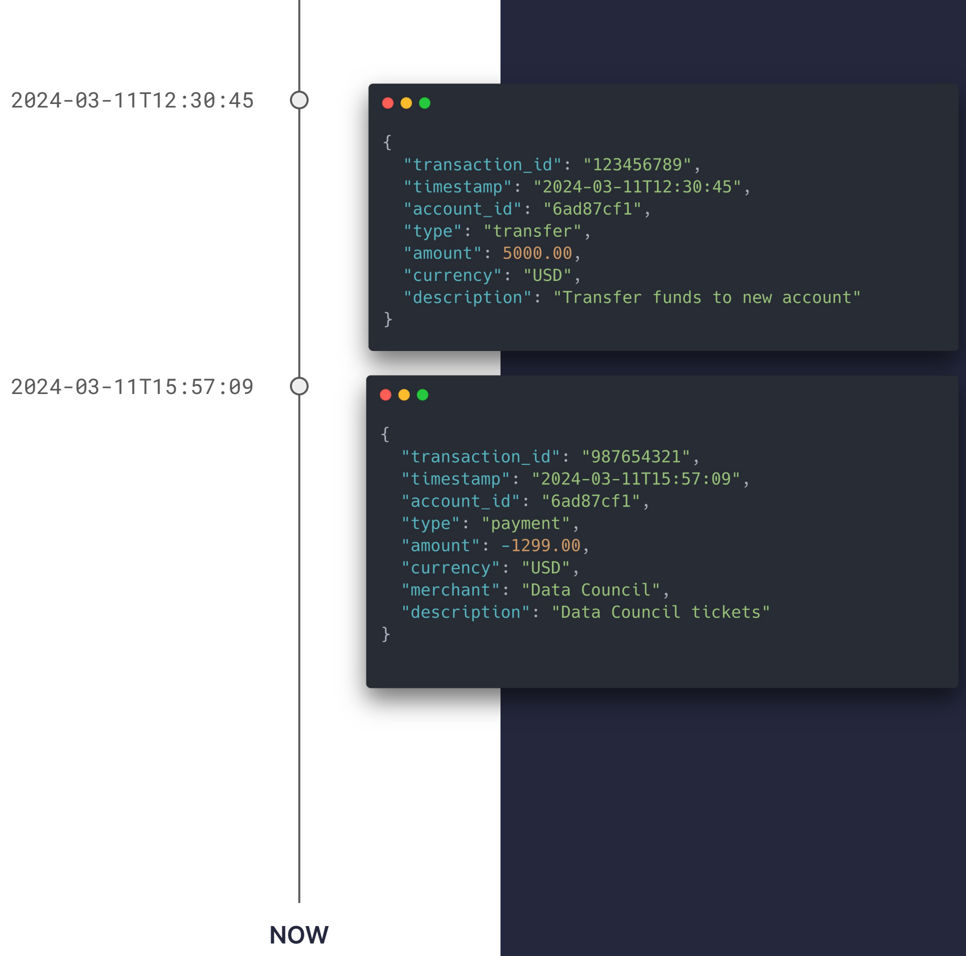
## Example



```
{
  "transaction_id": "123456789",
  "timestamp": "2024-03-11T12:30:45",
  "account_id": "6ad87cf1",
  "type": "transfer",
  "amount": 5000.00,
  "currency": "USD",
  "description": "Transfer funds to new account"
}
```

Event Sourcing in a nutshell

## Example

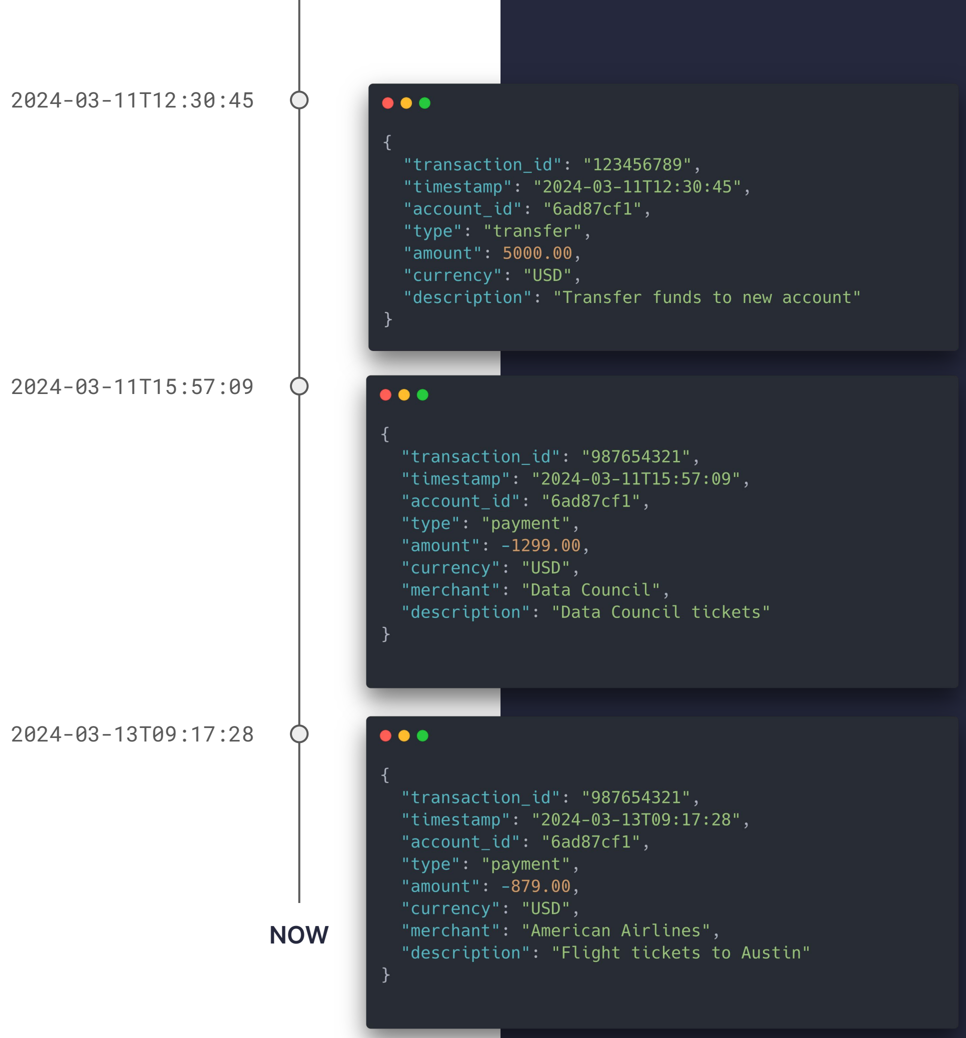


```
{
  "transaction_id": "123456789",
  "timestamp": "2024-03-11T12:30:45",
  "account_id": "6ad87cf1",
  "type": "transfer",
  "amount": 5000.00,
  "currency": "USD",
  "description": "Transfer funds to new account"
}
```

```
{
  "transaction_id": "987654321",
  "timestamp": "2024-03-11T15:57:09",
  "account_id": "6ad87cf1",
  "type": "payment",
  "amount": -1299.00,
  "currency": "USD",
  "merchant": "Data Council",
  "description": "Data Council tickets"
}
```

Event Sourcing in a nutshell

# Example



```
{
  "transaction_id": "123456789",
  "timestamp": "2024-03-11T12:30:45",
  "account_id": "6ad87cf1",
  "type": "transfer",
  "amount": 5000.00,
  "currency": "USD",
  "description": "Transfer funds to new account"
}
```

```
{
  "transaction_id": "987654321",
  "timestamp": "2024-03-11T15:57:09",
  "account_id": "6ad87cf1",
  "type": "payment",
  "amount": -1299.00,
  "currency": "USD",
  "merchant": "Data Council",
  "description": "Data Council tickets"
}
```

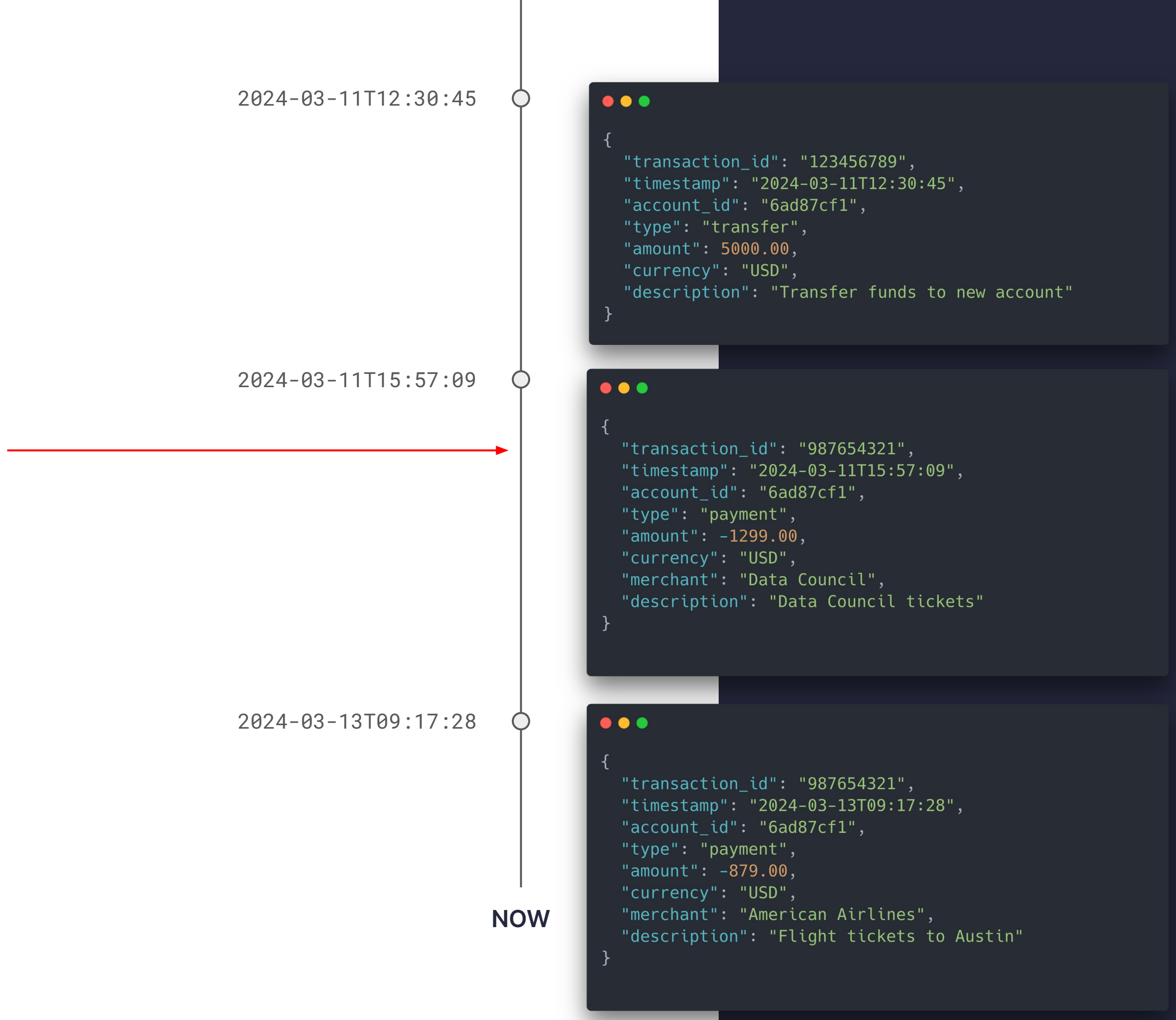
```
{
  "transaction_id": "987654321",
  "timestamp": "2024-03-13T09:17:28",
  "account_id": "6ad87cf1",
  "type": "payment",
  "amount": -879.00,
  "currency": "USD",
  "merchant": "American Airlines",
  "description": "Flight tickets to Austin"
}
```

Event Sourcing in a nutshell

## Example

**Balance**

$5,000 - $1,299 = $3,701



```
{
  "transaction_id": "123456789",
  "timestamp": "2024-03-11T12:30:45",
  "account_id": "6ad87cf1",
  "type": "transfer",
  "amount": 5000.00,
  "currency": "USD",
  "description": "Transfer funds to new account"
}
```

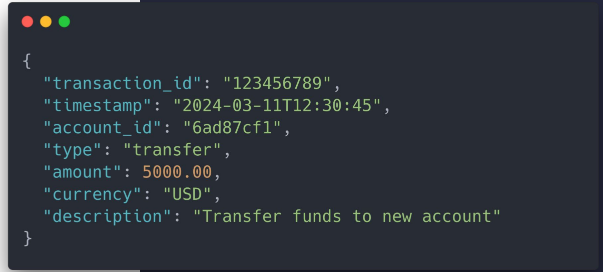
```
{
  "transaction_id": "987654321",
  "timestamp": "2024-03-11T15:57:09",
  "account_id": "6ad87cf1",
  "type": "payment",
  "amount": -1299.00,
  "currency": "USD",
  "merchant": "Data Council",
  "description": "Data Council tickets"
}
```

```
{
  "transaction_id": "987654321",
  "timestamp": "2024-03-13T09:17:28",
  "account_id": "6ad87cf1",
  "type": "payment",
  "amount": -879.00,
  "currency": "USD",
  "merchant": "American Airlines",
  "description": "Flight tickets to Austin"
}
```

Event Sourcing in a nutshell

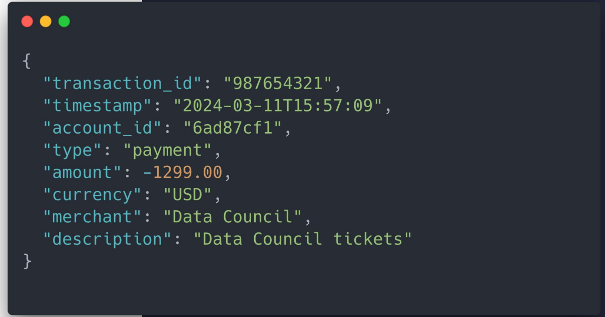
## Example

2024-03-11T12:30:45



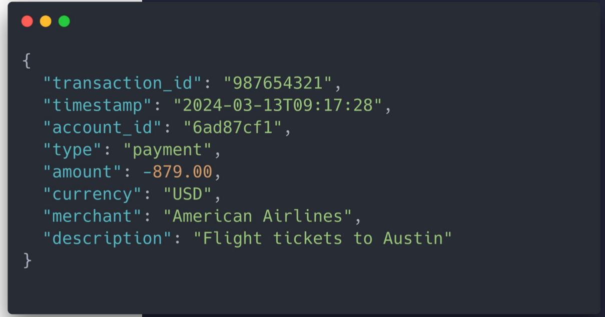
```
{
  "transaction_id": "123456789",
  "timestamp": "2024-03-11T12:30:45",
  "account_id": "6ad87cf1",
  "type": "transfer",
  "amount": 5000.00,
  "currency": "USD",
  "description": "Transfer funds to new account"
}
```

2024-03-11T15:57:09



```
{
  "transaction_id": "987654321",
  "timestamp": "2024-03-11T15:57:09",
  "account_id": "6ad87cf1",
  "type": "payment",
  "amount": -1299.00,
  "currency": "USD",
  "merchant": "Data Council",
  "description": "Data Council tickets"
}
```

2024-03-13T09:17:28



```
{
  "transaction_id": "987654321",
  "timestamp": "2024-03-13T09:17:28",
  "account_id": "6ad87cf1",
  "type": "payment",
  "amount": -879.00,
  "currency": "USD",
  "merchant": "American Airlines",
  "description": "Flight tickets to Austin"
}
```

**Balance**

$5,000 - $1,299 - $879 = $2,822 $\longrightarrow$ **NOW**

## When.
## Good use cases and scenarios

# Strong use cases

### Financial and Accounting systems

Immutable audit trail of transactions, ensuring regulatory compliance and enabling detailed analysis of financial activities, including fraud prevention.

### Billing, Security and Observability

Allowing real-time analysis, anomaly detection, and historical trend analysis while ensuring data integrity.

### eCommerce and Inventory management

Track order history, inventory changes, and customer interactions, allowing for accurate reporting, personalized recommendations, and order processing optimization.

tinybird

## Lessons learned.
## The good, the bad & the ugly

## Some actual metrics

| 500 MB/s | 15,000 | <100ms |
|---|---|---|
| Ingested data | Requests per second | Average read latency |

Event Sourcing — The good

# Full traceability

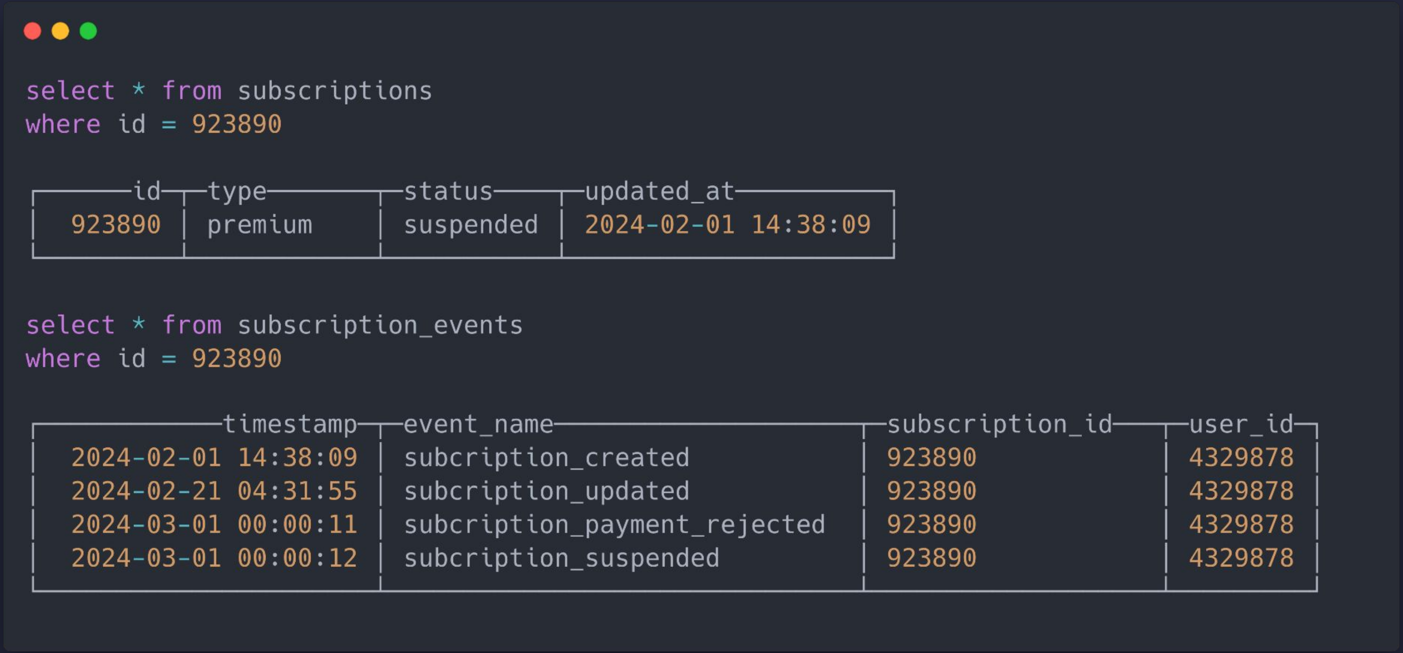
## Complete, inmutable trail of events

Now you have a complete history of changes to your application's state. This is great for traceability, debugging, and also building customer facing features such as an audit log.

You can investigate what happened in detail when something goes wrong. This is great for business operations, or even customer support.

Event Sourcing — The good

## Full traceability



```
select * from subscriptions
where id = 923890

┌───────id─┬─type─────────┬─status──────┬─updated_at──────────┐
│   923890 │ premium      │ suspended   │ 2024-02-01 14:38:09 │
└──────────┴──────────────┴─────────────┴─────────────────────┘

select * from subscription_events
where id = 923890

┌───────────timestamp─┬─event_name───────────────────────────┬─subscription_id─┬─user_id─┐
│ 2024-02-01 14:38:09 │ subcription_created                  │ 923890          │ 4329878 │
│ 2024-02-21 04:31:55 │ subcription_updated                  │ 923890          │ 4329878 │
│ 2024-03-01 00:00:11 │ subcription_payment_rejected         │ 923890          │ 4329878 │
│ 2024-03-01 00:00:12 │ subcription_suspended                │ 923890          │ 4329878 │
└─────────────────────┴──────────────────────────────────────┴─────────────────┴─────────┘
```

## Change business logic retroactively

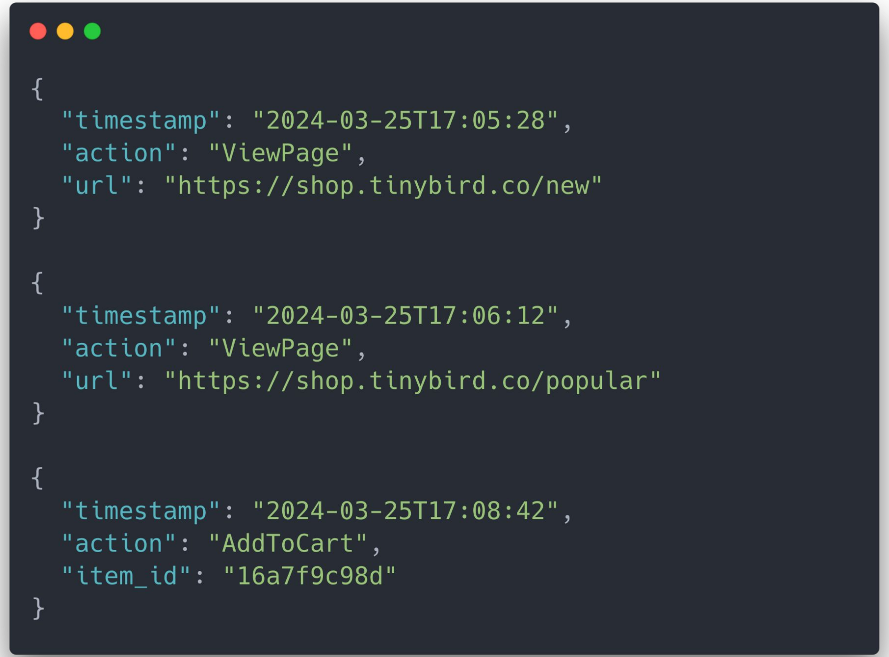
### Business rules evolve over time

Since you have the full logs of events, you can always rebuild the current state applying different business logic.

Even back-to-back testing and validating new business ideas with real data.

Event Sourcing — The good

## Attribution model example


```json
{
  "timestamp": "2024-03-25T17:05:28",
  "action": "ViewPage",
  "url": "https://shop.tinybird.co/new"
}

{
  "timestamp": "2024-03-25T17:06:12",
  "action": "ViewPage",
  "url": "https://shop.tinybird.co/popular"
}

{
  "timestamp": "2024-03-25T17:08:42",
  "action": "AddToCart",
  "item_id": "16a7f9c98d"
}
```

Event Sourcing — The good

## More flexible schema evolution

### Decoupled producers and consumers

Data models and business rules will evolve and change over time. Events capture domain-specific actions and intentions, and they can be versioned when needed.

Producers and Consumers be deployed in a more flexible way, and process the new event version asynchronously.

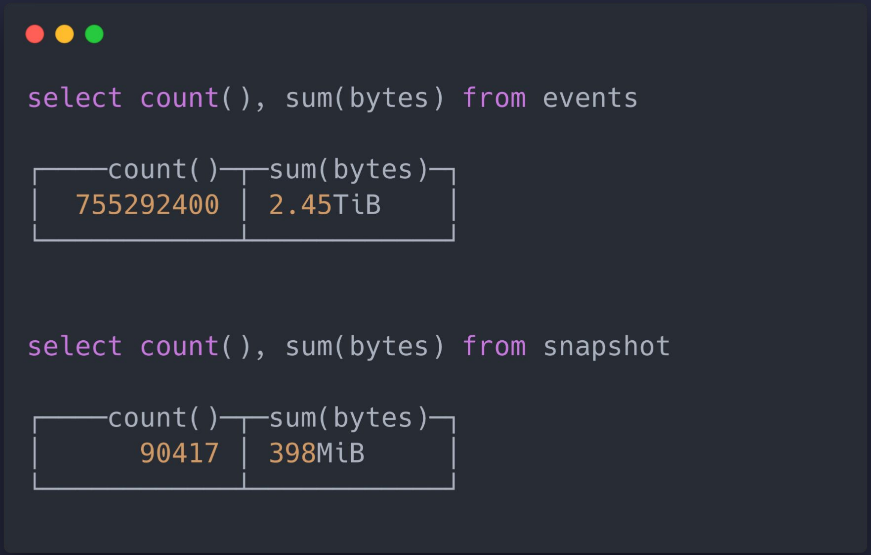
## The not-so-good

# Storage and Compute costs

### Up to 1,000x more disk

Storing a complete history of events usually leads to increased storage requirements compared to traditional state-based persistence methods like CRUDs.

This typically results in higher storage costs, especially for systems with a high volume of events.



```
select count(), sum(bytes) from events

┌──count()─┬─sum(bytes)─┐
│ 755292400 │ 2.45TiB    │
└──────────┴────────────┘


select count(), sum(bytes) from snapshot

┌──count()─┬─sum(bytes)─┐
│     90417 │ 398MiB     │
└──────────┴────────────┘
```

Event Sourcing — The not-so-good

## Eventual Consistency

### Data may not be ready for reads right away

Systems converge on a value over time but can lead to periods of inconsistent data, known as the inconsistency window.

**Availability vs Consistency**: sometimes is better not to make a decision, rather than doing it with partial or stale data.

## Complex day to day operations

### No more database UPDATEs

Businesses need to deal with lots of day to day nuances. No process is perfect, and there's always an exception.

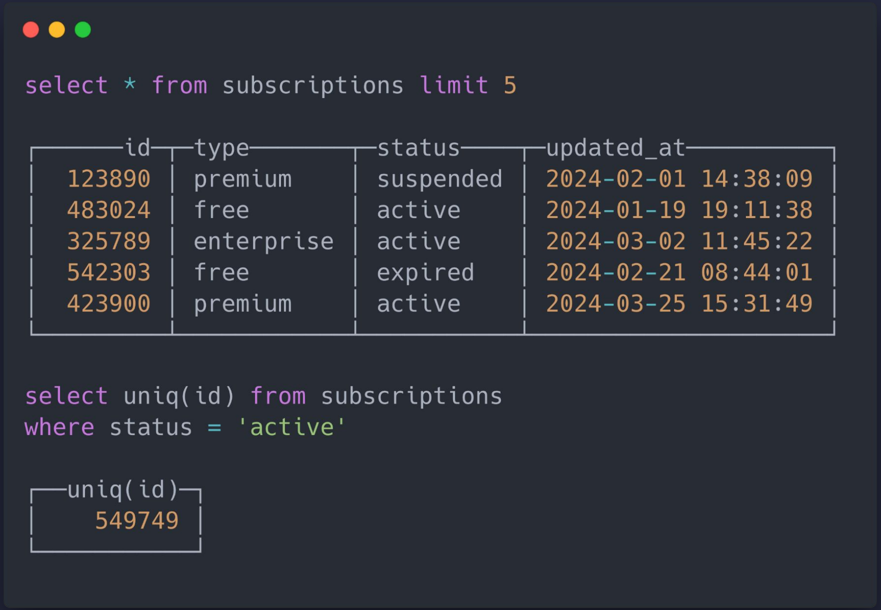
Say goodbye to direct database UPDATEs and DELETEs, **embrace compensation events and custom scripts**.

Event Sourcing — The not-so-good

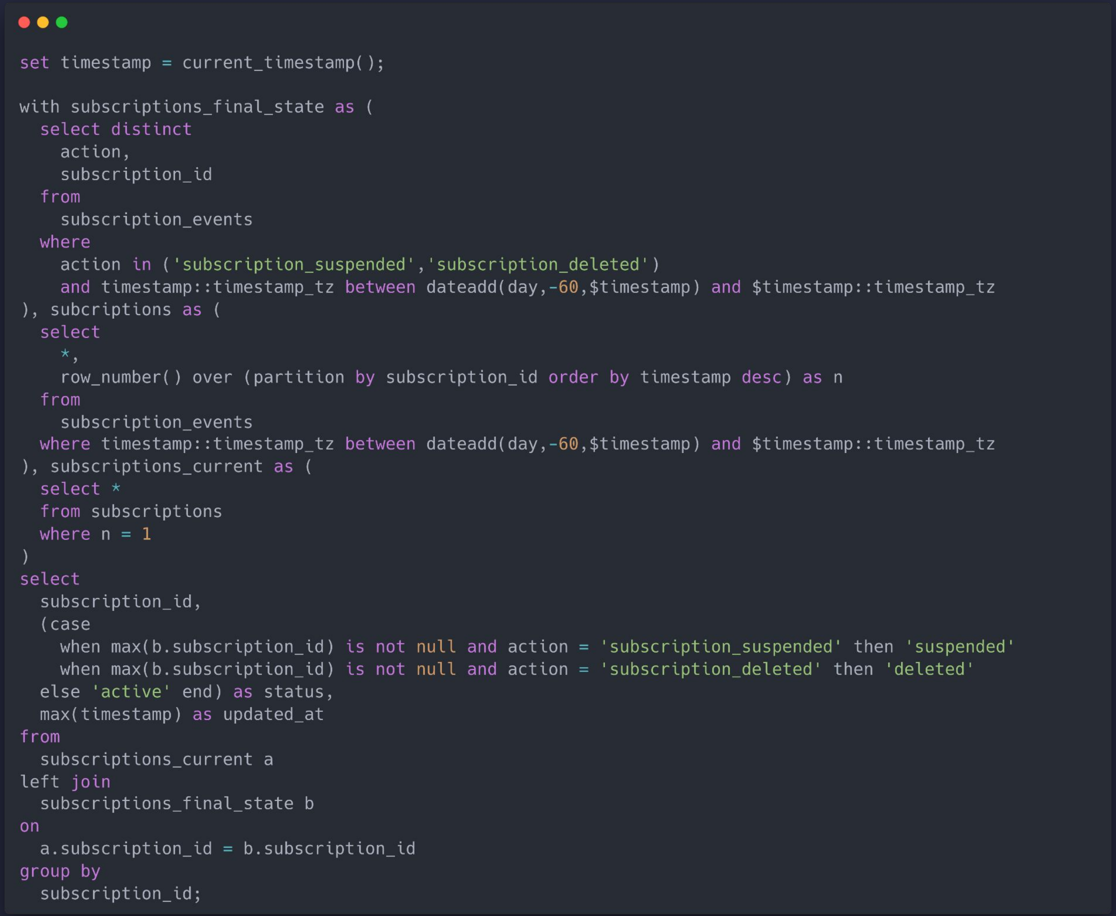
## Analytical complexity

### Way more complicated SQL queries

You'll need to account for sorting the events in time, handle duplicates, final states, and specific business logic.



```
select * from subscriptions limit 5

┌──────id─┬─type───────┬─status────┬─updated_at──────────┐
│  123890 │ premium    │ suspended │ 2024-02-01 14:38:09 │
│  483024 │ free       │ active    │ 2024-01-19 19:11:38 │
│  325789 │ enterprise │ active    │ 2024-03-02 11:45:22 │
│  542303 │ free       │ expired   │ 2024-02-21 08:44:01 │
│  423900 │ premium    │ active    │ 2024-03-25 15:31:49 │
└─────────┴────────────┴───────────┴─────────────────────┘

select uniq(id) from subscriptions
where status = 'active'

┌─uniq(id)─┐
│   549749 │
└──────────┘
```



```sql
set timestamp = current_timestamp();

with subscriptions_final_state as (
  select distinct
    action,
    subscription_id
  from
    subscription_events
  where
    action in ('subscription_suspended','subscription_deleted')
    and timestamp::timestamp_tz between dateadd(day,-60,$timestamp) and $timestamp::timestamp_tz
), subcriptions as (
  select
    *,
    row_number() over (partition by subscription_id order by timestamp desc) as n
  from
    subscription_events
  where timestamp::timestamp_tz between dateadd(day,-60,$timestamp) and $timestamp::timestamp_tz
), subscriptions_current as (
  select *
  from subscriptions
  where n = 1
)
select
  subscription_id,
  (case
    when max(b.subscription_id) is not null and action = 'subscription_suspended' then 'suspended'
    when max(b.subscription_id) is not null and action = 'subscription_deleted' then 'deleted'
  else 'active' end) as status,
  max(timestamp) as updated_at
from
  subscriptions_current a
left join
  subscriptions_final_state b
on
  a.subscription_id = b.subscription_id
group by
  subscription_id;
```

Event Sourcing — The not-so-good

# Handling duplicates

## Because yes, it will happen

Exactly-once semantics is really, really difficult to implement. Chances are you'll need to handle duplicate events at some time using some kind of transaction id.

## Old events will remain there

### Deprecating things is hard

As the system evolves, new versions of each events are created. However, old events with obsolete schemas may remain for a long time, and you will have to keep supporting them.

You'll make poor design decisions at the beginning, and you'll have to live with those for a while.

## Suggestions.
## Some heavily opinionated recommendations

# ~~Suggestions.~~
# Hot takes 🔥

Just don't do it

tinybird

Events are Data, and **What you really need is Information**

tinybird

## You need CQRS for reads, ideally an OLAP database

tinybird

## Snapshots are needed for performance

tinybird

## Materialized Views are awesome

tinybird

Time Series

Tokens

DATA PROJECT

Pipes (423)

Data Sources (139)

fedor_test
No API endpoint published

vercel_blob_usage_synthetic_test...
No API endpoint published

pipe_token_requests
No API endpoint published

mv_usage_rollup_v1_by_project_id...
No API endpoint published

playground_timer
No API endpoint published

nabeel_usage_facts_anomaly_detec...
No API endpoint published

pipe_datacache_business_intellig...
No API endpoint published

vsc_get_usage_facts_cardinality
No API endpoint published

vsc_set_usage_facts_cardinality
No API endpoint published

mv_proxy_response_usage_facts_pi...
No API endpoint published

api_neon_postgres_storage_last_u...
No API endpoint published

neon_postgres_storage_usage_fact...
No API endpoint published

mv_neon_postgres_storage__1hr_ne...
No API endpoint published

mv_neon_postgres_storage__1hr_ne...
No API endpoint published

mv_neon_postgres_storage__1hr_ne...
No API endpoint published

neon_postgres_storage_usage_fact...
No API endpoint published

neon_postgres_storage_usage_fact...
No API endpoint published

neon_postgres_storage_usage_fact...
No API endpoint published

# Q&A

# Thank you.