# Fraud Analytics using a different approach, GraphDB data platform – (part 2)

(8 Aug 2025)

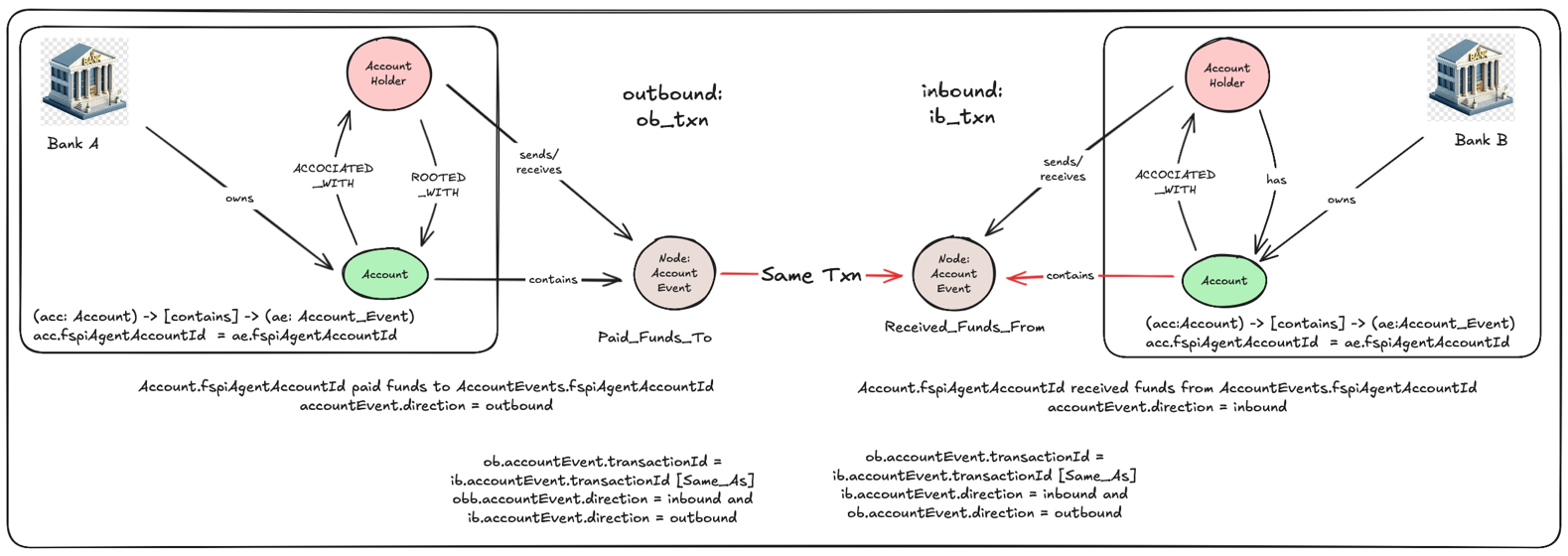
**Overview**

This is part 2 of a series… not yet sure how many parts there will be. This will be a short part though.

In [part 1](https://medium.com/@georgelza/fraud-analytics-using-a-different-approach-graphdb-data-platform-part-1-807c68d03bff) we explored how to push data into a GraphDB platform (build using Neo4J) as a graph based data model using the cypher language.

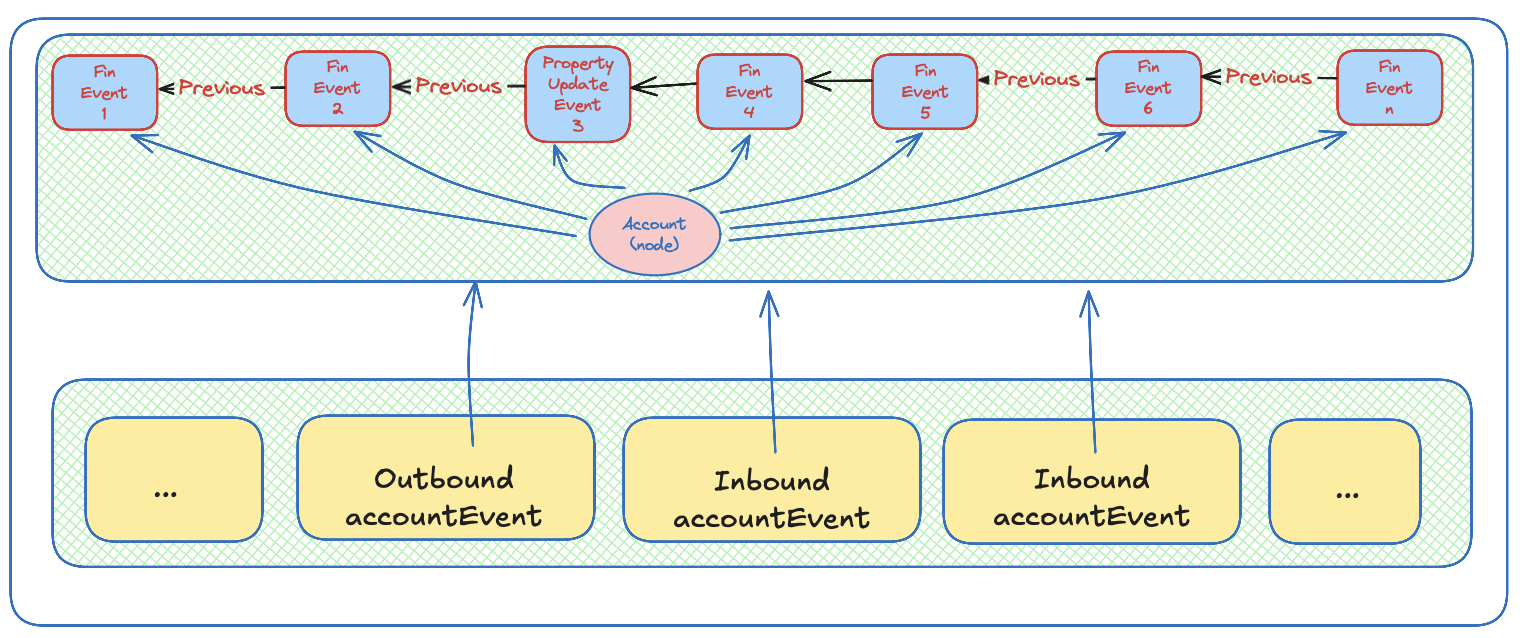
I also promised we will explore how to add a temporal view of our (accountEvents) nodes, aka transactions linked to an account, all visualized over time line.

Just a quick recap. Every financial transaction is in effect 2 events. An outbound financial event on the Payer side and an inbound financial event on the Payee side.



Above I show how one account pays funds to another. The transaction is model as 2 separate events, the first being an outbound debit cash flow from the payers account and then a 2nd even for the inbound payee as a credit event.

Now if we think a bit more, an account will have multiple events linked to it over time, some being debits and some credits, creating a history of events for the account, and as such a profile of behaviour for the account.



An account as a node and its surrounding nodes of accountEvents over time.

Using the following cypher, now heavily modified from our previous MERGE statement, we’re now adding an accountEventCounter to our new accountEvent node in addition to adding the eventId from the previous events into our new accountEvent.

You will notice the CALL block where we go looking for the previous newest accountEvent based on evenTime for the same fspiAgentAccountId, once found we then increment the accountEventCounter and add the previous eventId to the new node.

MERGE (n:AccountEvents {eventId: "9265405743564350480"})

ON CREATE SET

n.eventId= "9265405743564350480",

n.fspiAgentAccountId = "ULSBIE2D-427570486",

n.counterpartyAgentAccountId = "ULSBIE2D-427570487",

n.transactionID = "2299405743564352934",

n.eventTime = "20/07/2025 14:02:10",

n.baseValue = 2342.00,

n.currency = "EUR",

n.direction = "outbound",

n.paymentSystem = "RTPUSHPAY",

n.settlementMethod = "CLRG",

n.settlementClearingSystemCode = "RTPay",

n.paymentMethod = "TRF",

n.overallScore = null,

n.riskProfile = "no-risk"

WITH n

CALL (n) {

OPTIONAL MATCH (AC:AccountEvents {fspiAgentAccountId: "ULSBIE2D-427570486"})

WHERE AC.eventId <> n.eventId // Exclude the current node

WITH AC ORDER BY AC.eventTime DESC LIMIT 1

RETURN coalesce(AC.eventId, null) AS prevEventId,

coalesce(AC.accountEventCounter, 0) + 1 AS accountEventCounter

}

SET n.prevEventId = prevEventId,

n.accountEventCounter = accountEventCounter

RETURN n;

When combined the above with the below cypher, which creates a new edge, linking our n+1 to n, we now create a “thread” of accountEvents as associated with an account that can be transversed by anyone.

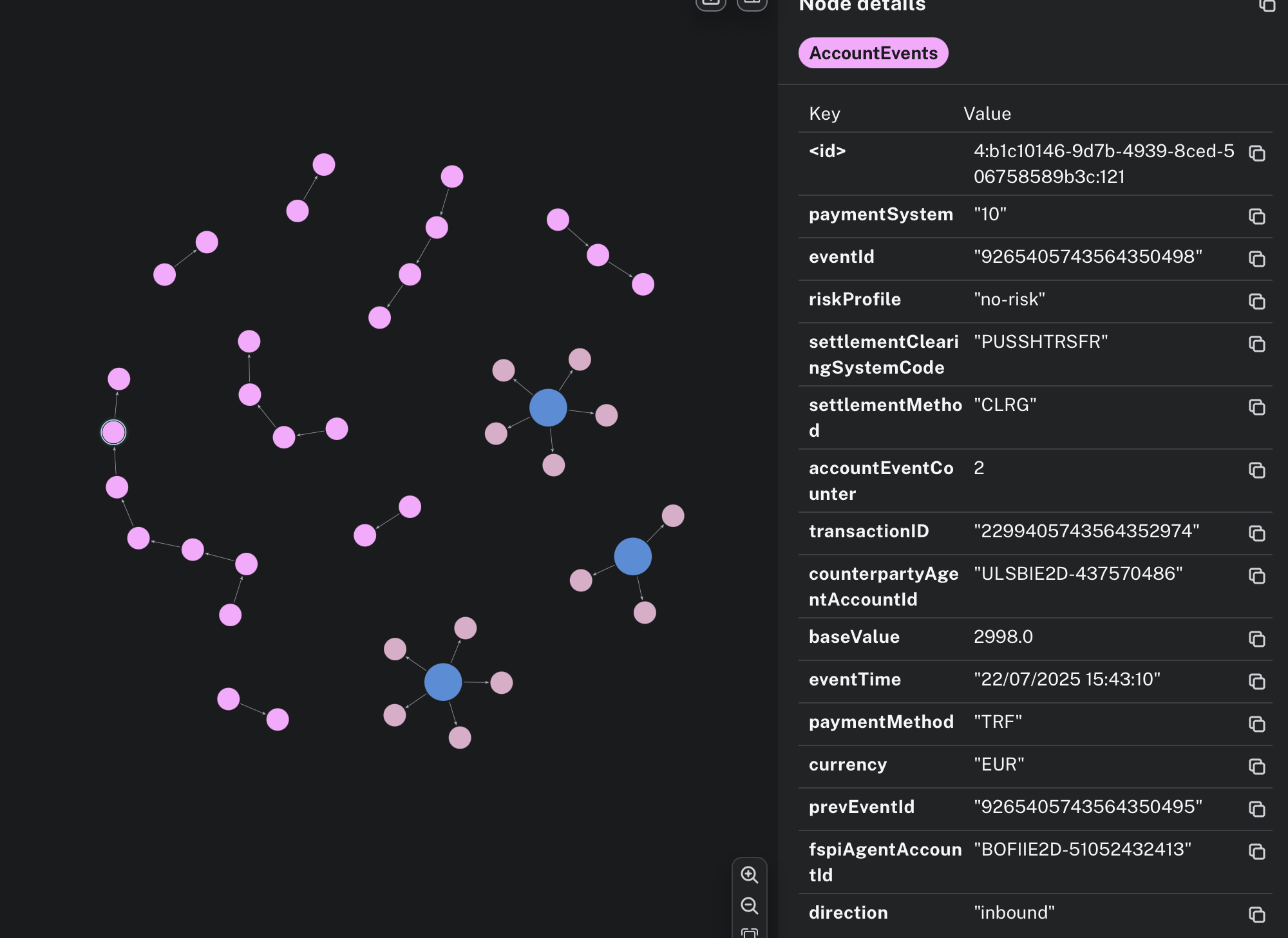
MATCH (aec:AccountEvents)

MATCH (aep:AccountEvents)

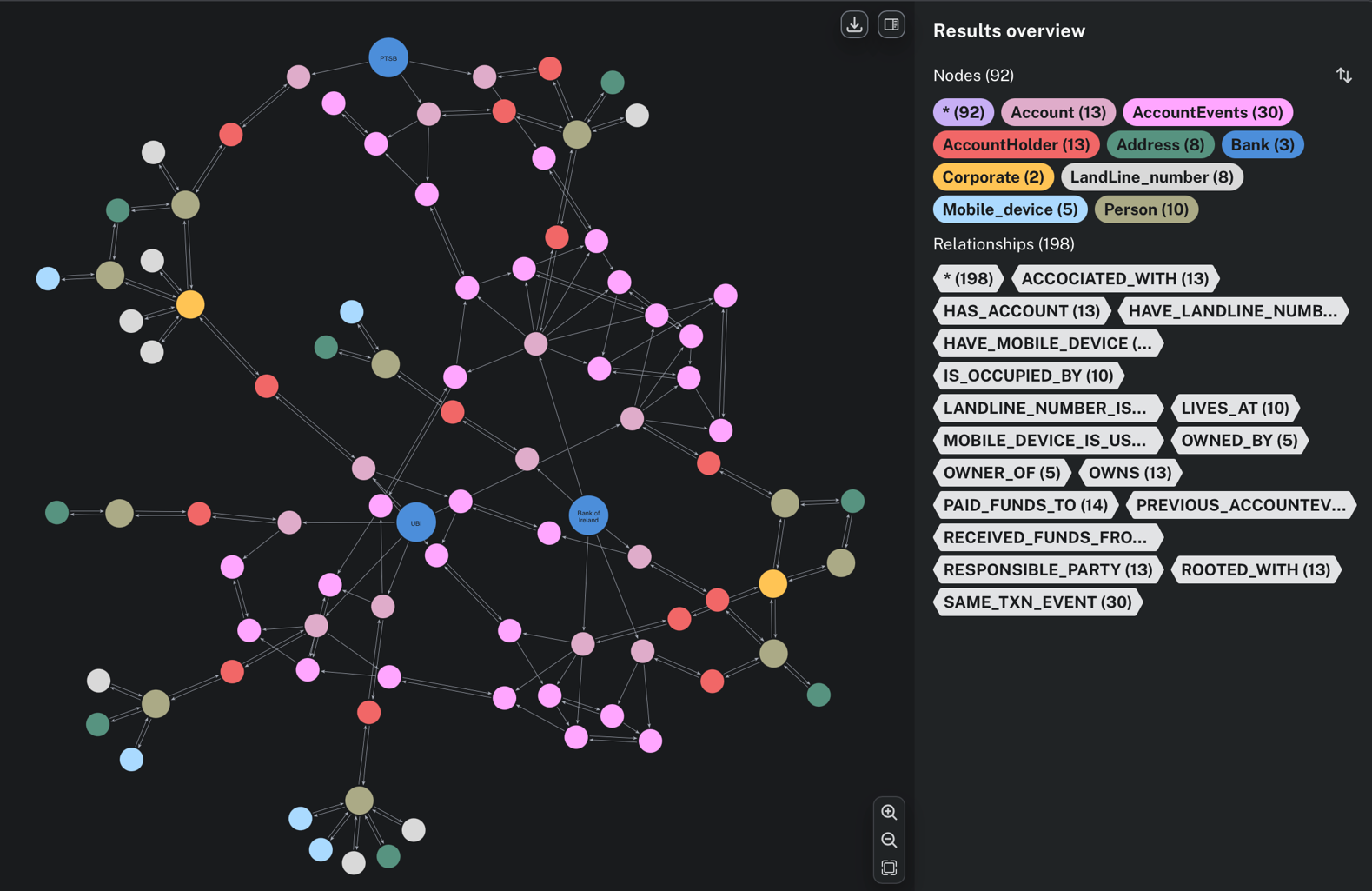
WHERE aec.eventId = aep.prevEventId

MERGE (aep)-[:PREVIOUS\_ACCOUNTEVENT]->(aec);

When we visualize this in our Web console, we can see the threads of accountEvents, one linked to the previous.



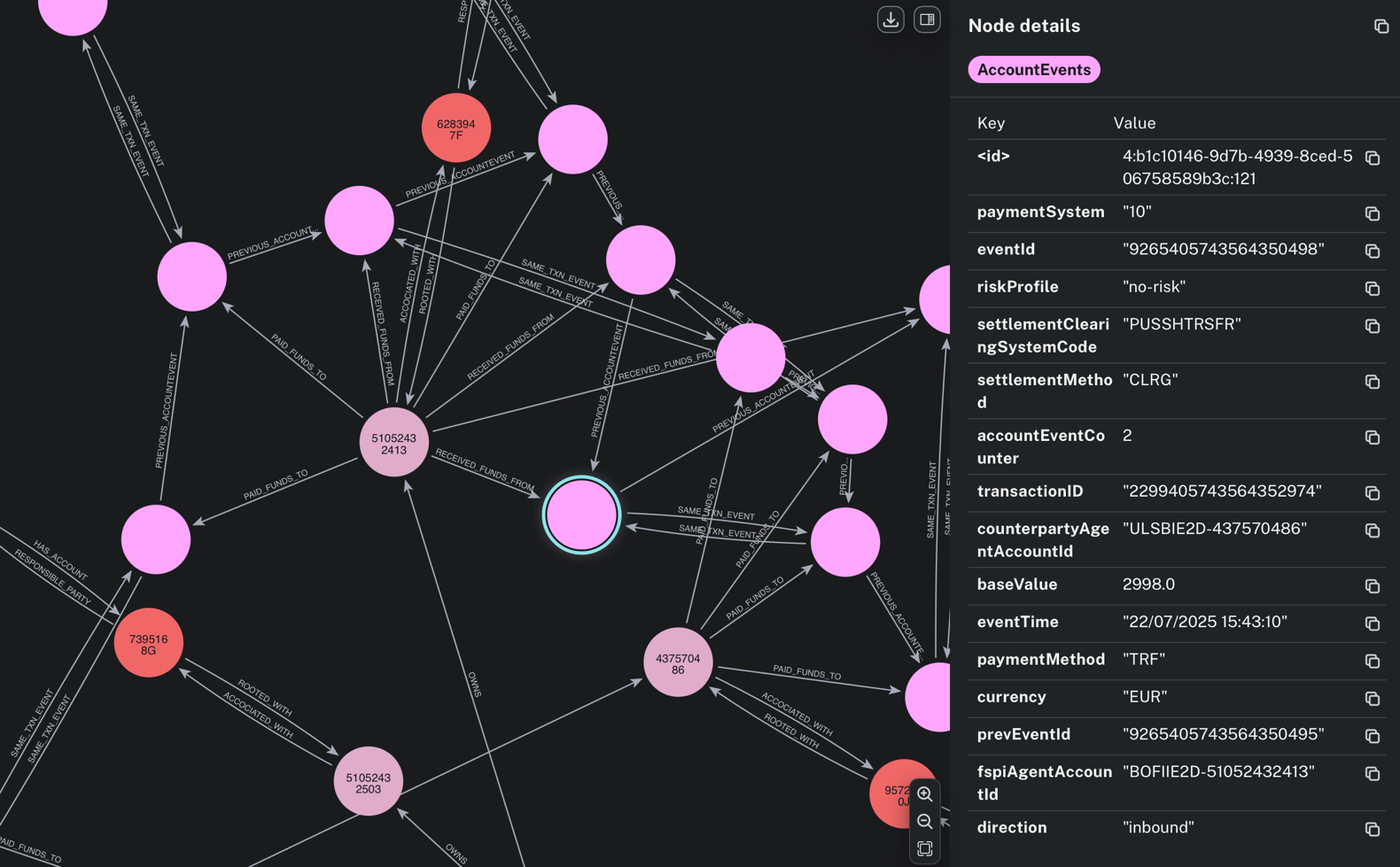
With the above modification and our previous code all combined will result in the below.



All of the above can be replicated using the code from GIT REPO: [financial\_txn\_flow\_using\_graphDB – Part 2](https://github.com/georgelza/financial_txn_flow_using_graphDB-part2),

NB; I renamed/refactored the nodes/properties… so don’t mix this version with the previous.

And if we zoom in on the above, we can see the various [PAID\_FUNDS\_TO] and [RECEIVED\_FUNDS\_FROM] links to the central (account) node, but we can also further see the edge/link linking one (accountEvent) to the previous as shown by the label [PREVIOUS\_ACCOUNTEVENT], see below.



Summary

I only depicted financial transactions and the associated events above. What is important here is the same concepts can be applied to our other node types;

* corporate,
* person,
* address,
* mobile number,
* landline number
* etc

For each creating a lineage (think version control) depicting the node as it changes over time.

As we mentioned before. Bad actors don’t create a large profile day 1. They create a small, nicely polished profile. All isolated, and then grow that over time. The trick is to pick this up and notice how the profiles are changed and then go look for more of the same behaviour. Eventually allowing us to stop fraud before it happens and not simply identify it afterwards.

The series is not complete. You will notice in the README.md I have some more rabbit hole to explore.

Thanks for following. Till next time.



**About Me**

I’m a techie, a technologist, always curious, love data, have for as long as I can remember always worked with data in one form or the other, Database admin, Database product lead, data platforms architect, infrastructure architect hosting databases, backing it up, optimizing performance, accessing it. Data data data… it makes the world go round.

In recent years, pivoted into a more generic Technology Architect role, capable of full stack architecture.

[George Leonard](https://www.linkedin.com/in/george-leonard-945b502/)

[georgelza@gmail.com](mailto:georgelza@gmail.com)

<https://medium.com/@georgelza>