# Fraud Analytics using a different approach, GraphDB data platform

(30 July 2025)

**Overview**

This all started with my manager nudging me to look at GraphDB platforms, in relation to a use case and my back ground.

Well anyone say Rabbit hole… ;)

Financial institutions on their own only have what’s commonly referred to as a “on US” view of activity on an account. As it stands Fraud actors are taking advantage of this, hiding their activity by building up accounts/profiles over time, spanning multiple financial service providers and moving funds around between these distributed accounts.

The below scenario is positioned from the viewpoint of a national clearing house (NCH), acting as a Clearing and Settlement interchange, which operates between Financial service providers.

Using this position the NCH in question has the unique position where by they can provide centralized inter-bank Fraud Analytics service, through this they are able to Fraud Score all inter-bank financial transfers, thus they will have a more complete view of the wider flow of funds.

For the Blog below I’ve localized everything as based in Ireland. As such we’re using Irish personal identification numbers, Banks and addresses, mobile device and land line numbers, etc.

Ireland uses a **Personal Public Service Number (PPS Number)** as their social security identifier, unique identification number. Below is the format:

## PPS Number Format

* **Pattern**: 7 digits + 1 or 2 letters
* **Example**:  1234567A or 1234567AB

## Structure

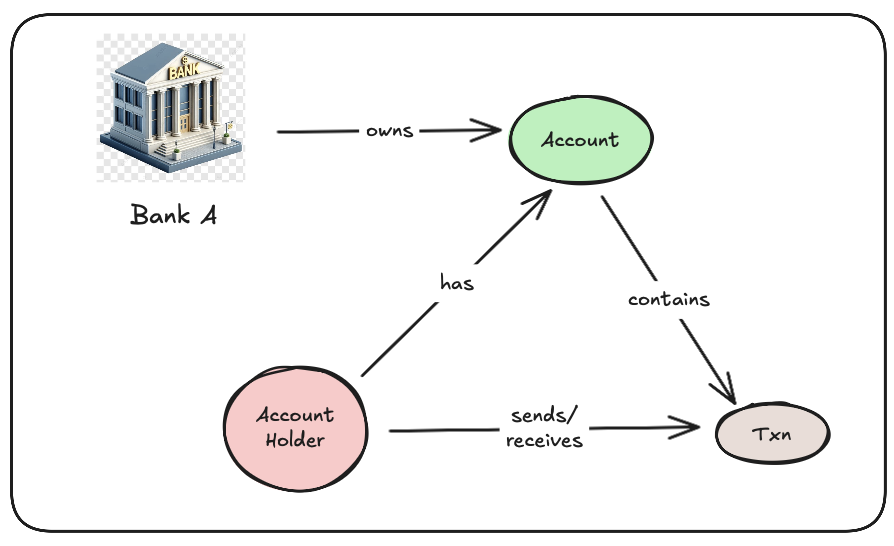
* **First 7 characters**: Sequential numbers (1234567)
* **8th character**: Always a letter (A-W, excluding I and O to avoid confusion)
* **9th character**  (optional): Second letter for people born after 2013, or in special cases

The below view considers we have access to 5 primarily data products, feeds.

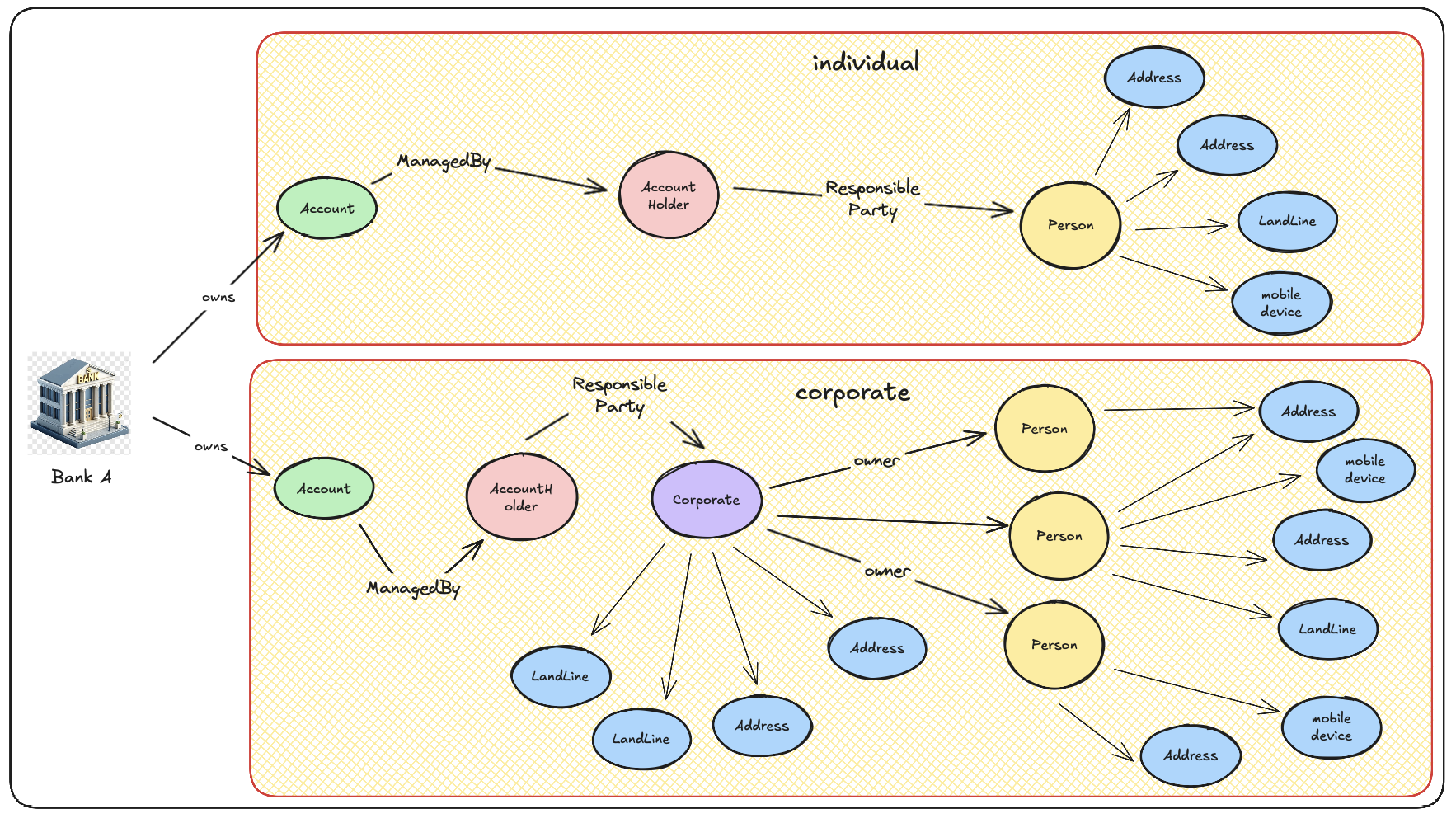
1. Fraud scoring for all transactions.
2. Banks to provides for individually held accounts an accoutNumber mapping to PPS/Social Security Number (SSN) list.
3. Banks to provide for corporate held accounts and accoutNumber mapping to Company Registration Numbers.
4. National Company register provide a company registration number together with the PPS/SSN of the registered owners, if this includes registered company address would be highly beneficial.
5. National Dept of Home Affairs (Home Office) provide profile for PPS/SSN.
6. A Stretch wish’s, Mobile Telco Operators provide Mobile Device Number (Cell Phone Number) mapped to PPS/SSN. Further add to this the Telco’s “Know your Customer/KYC” information referencing the registered residential address of the owner of the device.

With the above data products and the concepts below I will show how we can re-write Fraud Analytics.

First, a view of our Bank, account, account holder and transactions.

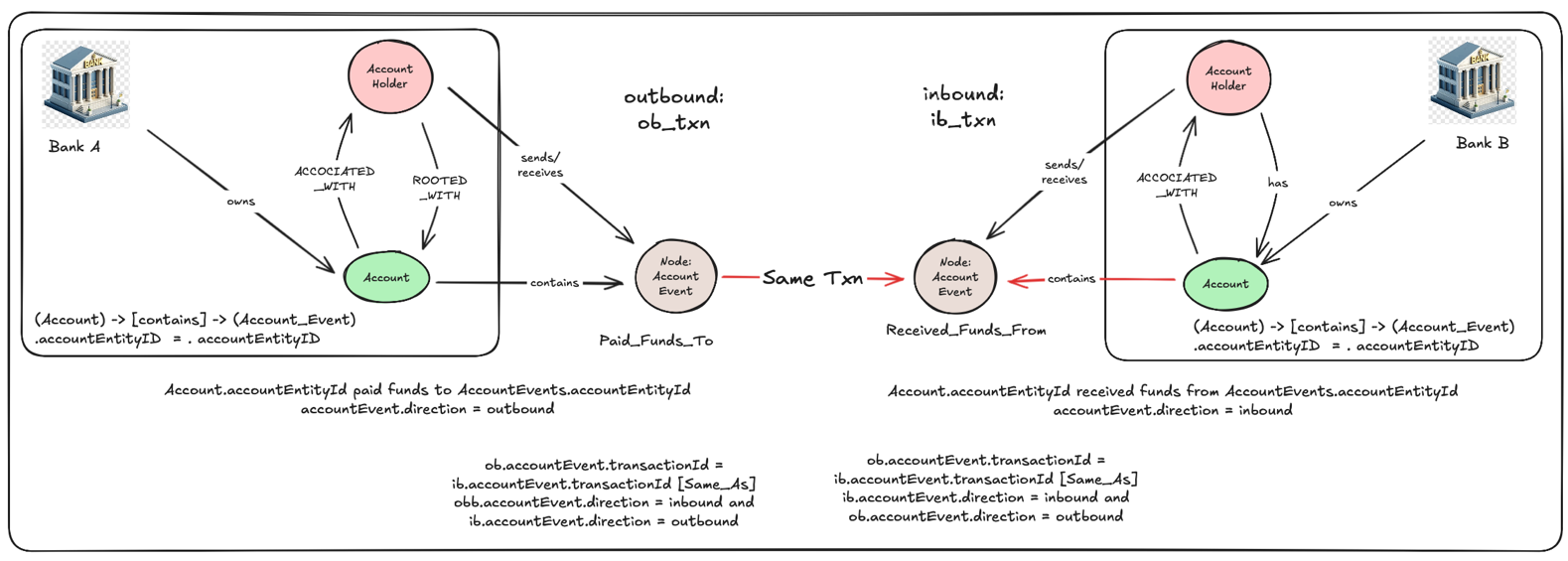


Now… If we expand this to incorporate company’s and their ownership as a dimension.



At the AccountHolder we define the account as either being “individual” or “corporate”. Most bank by tradition operate in 2 sub divisions, corporate banking or personal banking. I also further show how a corporate or a individual is associated with various addition data points/nodes

Ok, so we depicted the basic natural relationships between Banks and the eventual “Responsible Party’s”. Let’s now model the cash flow as a transaction but in real 2 separate financial events.



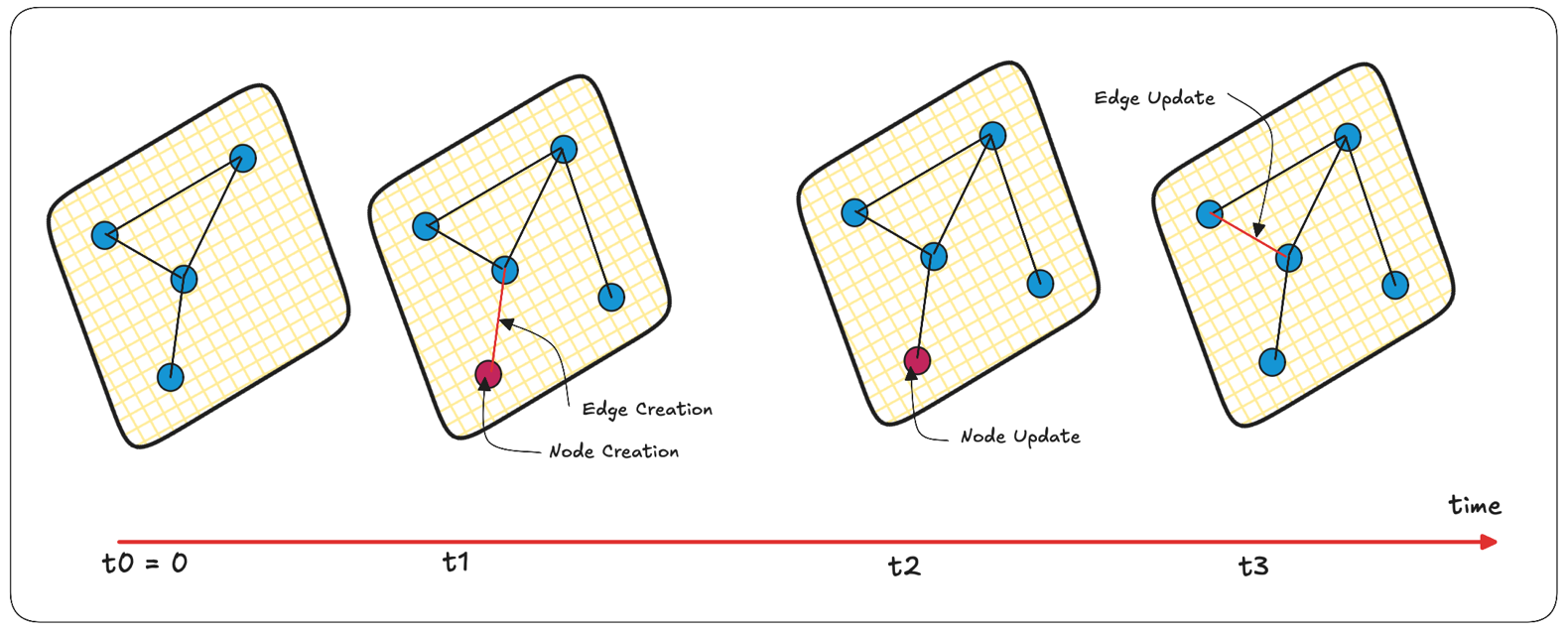
Above I show how one account pays funds to another. The transaction is model as 2 separate events, the first being an outbound debit cash flow from the payers account and then a 2nd even for the inbound payee as a credit event.

But, we’re note done, are we ever.

The last component, for which the importance should never be under estimated is time… The ability to identify and analyse patterns as they develop/emerge over time.

Bad actors don’t create a profile in one step and then use them. They build up a profile over time, slowly so as not to attract attention.

For this we need to introduce what’s referred to as a temporal dimension, basically a time based view by decorating all activity/events with a eventTime, be that a financial transaction or a data update for an account, corporate, person, an address etc.



An account as a node and its surrounding nodes and relationships change over time.

Why Graph Database?

For many years relationship databases have been used to model 1 to many relationships, and they are amazing doing that, but it have to be recognised that they do battle with Many to Many.

Many-to-many relationships are an important distinguishing feature between different data models. If you application has mostly one-to-many relationships (tree-structured-data) or no relationships between records, the document model is appropriate or a standard RDBMS.

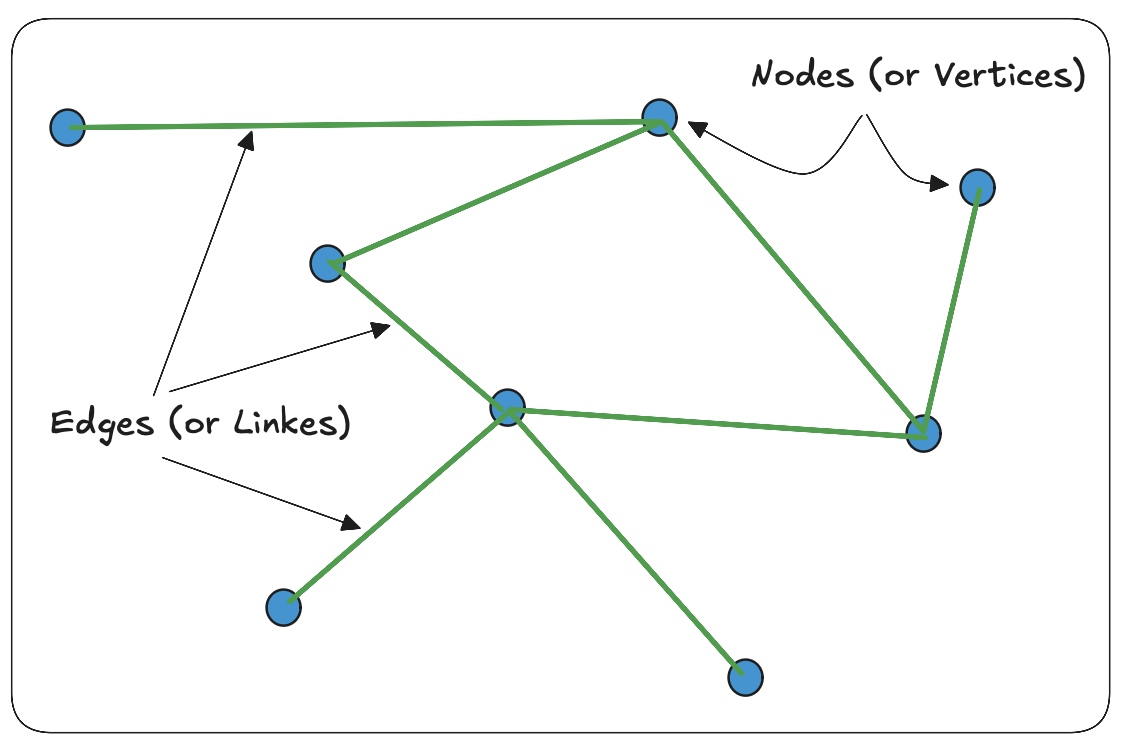
But what if many-to-many relationships are very common in your data? The relational model can handle simple cases of many-to-many relationships, but as the connections within your data become more complex, it becomes more natural to start modelling your data as a graph.

What is a Graph?

Graph databases are a type of NoSQL database that use graph structures (nodes, edges, and properties) to represent and store data, emphasizing relationships between data entities. Unlike relational databases that use tables, graph databases excel at handling complex, interconnected data and are well-suited for tasks like fraud detection, recommendation engines, and knowledge graphs.

Basic GraphDB Concepts

A graph consists of two kinds of objects: vertices (also known as nodes or entities) and edges (also known as relationships or arcs).



Data in a graph database is primarily stored as 3 different objects:

Based on Property Graphs model & the triple-store concept which defines 3 primary “actors”, namely:

Subject, Predicate, Object resulting in a Relationship Graph, modelled as:

* Nodes (Subject and Objects) - a node is the thing you are storing in the database. Thinking relationally, this is similar to a record in a table. If the table holds details on Transactions in this case, each record about a single Transaction will be their own node. A node can be an instance of any entity, a person, a place, an address, a thing, etc and the same graph database can hold instances of multiple types of these entities.
* Edges (Predicates) - an edge is the relationship between nodes. Again, thinking relationally, they are similar to a foreign key between nodes. Relationships are not mandatory but they can be many-to-many. The same nodes can be related to each other in multiple different ways.
* Properties (Properties associated) - extra non-mandatory attributes that can be added to either a node or an edge.

For our use case, at a simple level.

Subject (p): Who => Outgoing Account/the Debtor

Predicate (s): What => Payment/Transaction also referred to as the Verb

Object (o): Who => Incoming Account/the Creditor

Properties (p): Additional detail.

Graph databases primarily utilize specialized query languages to interact with and retrieve data, with some also supporting general-purpose programming languages for specific tasks. Popular graph query languages include Cypher (Neo4j), Gremlin (Apache TinkerPop), and SPARQL (for RDF-based graphs). Additionally, some databases offer their own proprietary languages like AQL (ArangoDB) and GSQL (TigerGraph), or utilize SQL extensions for graph querying.

All the languages are primarily structured around the user defining the desired outcome, result he/she wants from the query.

Creating Nodes:

Here we create an Bank node: The values in the ON CREATE SET block will be populated at the time of creation, if we find that tenantId: “ULSBIE2D” as specified in the Merge (n:Bank {xxxx}) already exist then the ON MATCH SET block is executed by following the += form that says, add what’s not defined and update if not as supplied. If a property is not specified then it is ignored, the value stays as is, if it is supplied and set to null then the property is removed from the node.

GraphDB’s follows a unstructured design, in other words the node and it’s properties are not predefined/rigit.

// Create/Update - If you wanted to update more than just a bit...

MERGE (n:Bank { tenantId: "ULSBIE2D"})

ON CREATE SET n = {

tenantId: "ULSBIE2D",

accountAgentId: "ULSBIE2D",

memberName: "Ulster Bank Ireland DAC",

displayName: "UBI",

bicfi: "ULSBIE2D",

memberNo: "210004",

sponsoredBy: "210004",

swift\_code: "ULSBIE2D"

}

ON MATCH SET n += {

memberNo: "210004",

sponsoredBy: "210004",

branchStart: 440000,

branchEnd: 449999,

mnemonic: "ACC",

acg: "universal",

swift\_code: "ULSBIE2D"

}

RETURN n;

Creating Edges:

What we do here is define relationships between nodes, links.

Below we associate the Account record/node with the accountHolder based on the accountEntityId and label the association as “Associated\_with”

MATCH (acc:Account)

MATCH (ah:AccountHolder)

WHERE acc.accountEntityId = ah.accountEntityId

MERGE (acc)-[:ACCOCIATED\_WITH]->(ah);

We will now associate the accountholder based on regId if present with a Corporate as the Responsible\_party and then a define reverse edge indicating the corporate as has\_account as defined by the AccountHolder.

// Create (AccountHolder) -> [RESPONSIBLE\_PARTY]-> (Corporate) edge

MATCH (ah:AccountHolder)

MATCH (cp:Corporate)

WHERE ah.regId = cp.regId

MERGE (ah)-[:RESPONSIBLE\_PARTY]->(cp);

// Create (Corporate) -> [HAS\_ACCOUNT]-> (AccountHolder) edge

MATCH (cp:Corporate)

MATCH (ah:AccountHolder)

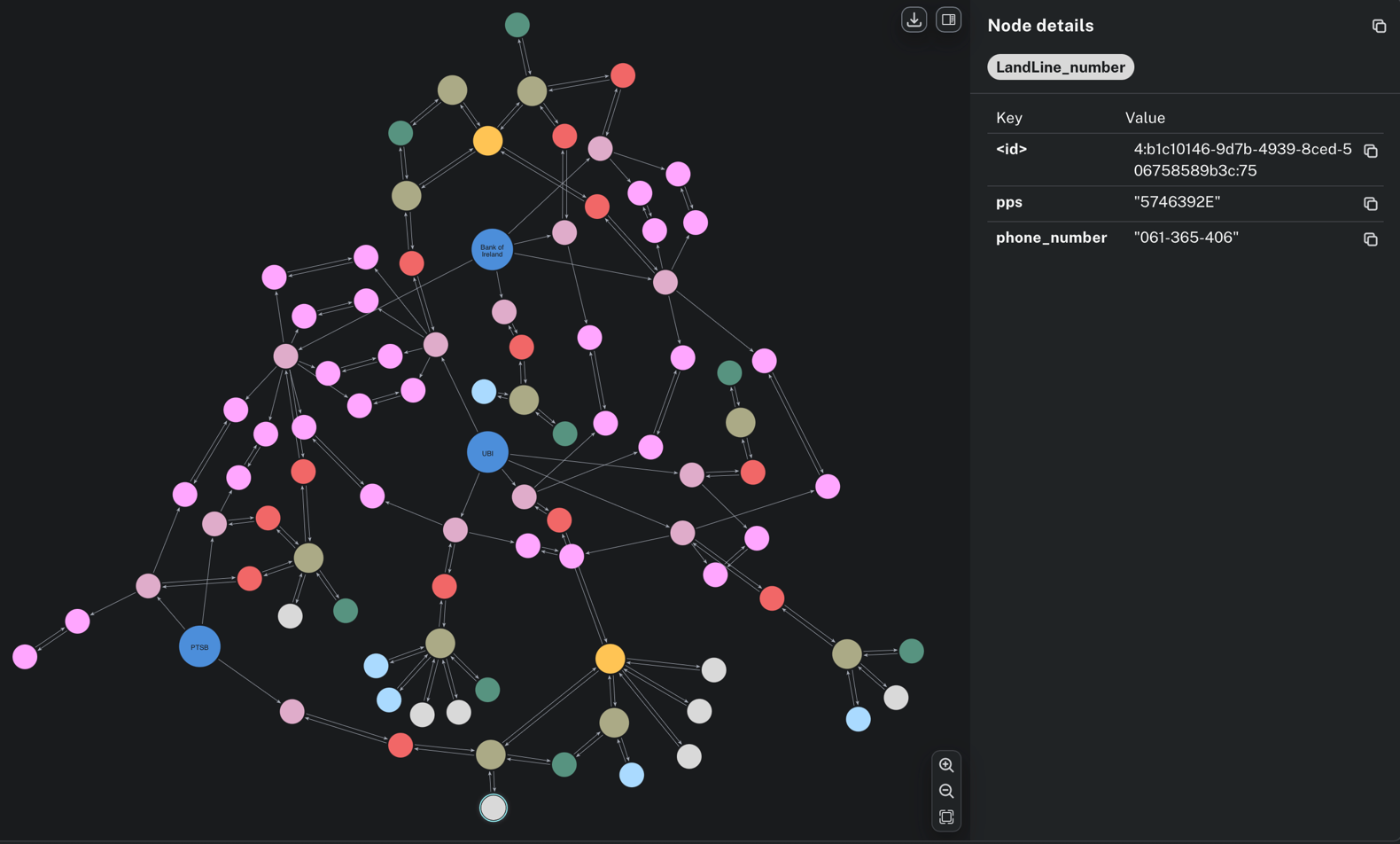
WHERE ah.regId = cp.regId

MERGE (cp)-[:HAS\_ACCOUNT]->(ah);

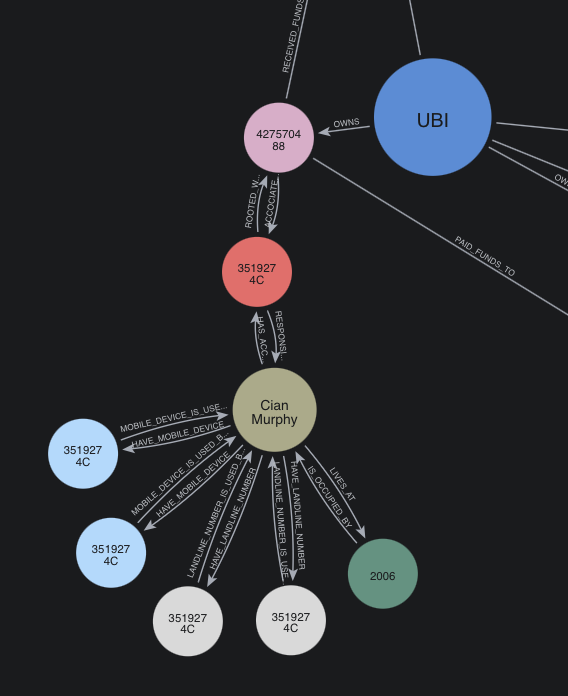
Our scenario is modelled around the following.

Below is an imaginary flow of funds depicted, mapped out between:

* 4 x Banks,
* 13 x Accounts,
* 13 x Account Holders (ah), ah being either a Corporate or individual depicted as a Person’s.
* 3 x Corporate’s, owned by 1 or more individuals.
* 10 x individuals model as “person’s”.
* 25 x Address/s, associated with
* 5 x mobile devices and further
* 8 x landline numbers.
* … we can add more data points here, i.e. Internet Service Providers, IP’s, if available location based tracking of transactions. Physical lat/long values associated with the addresses etc.

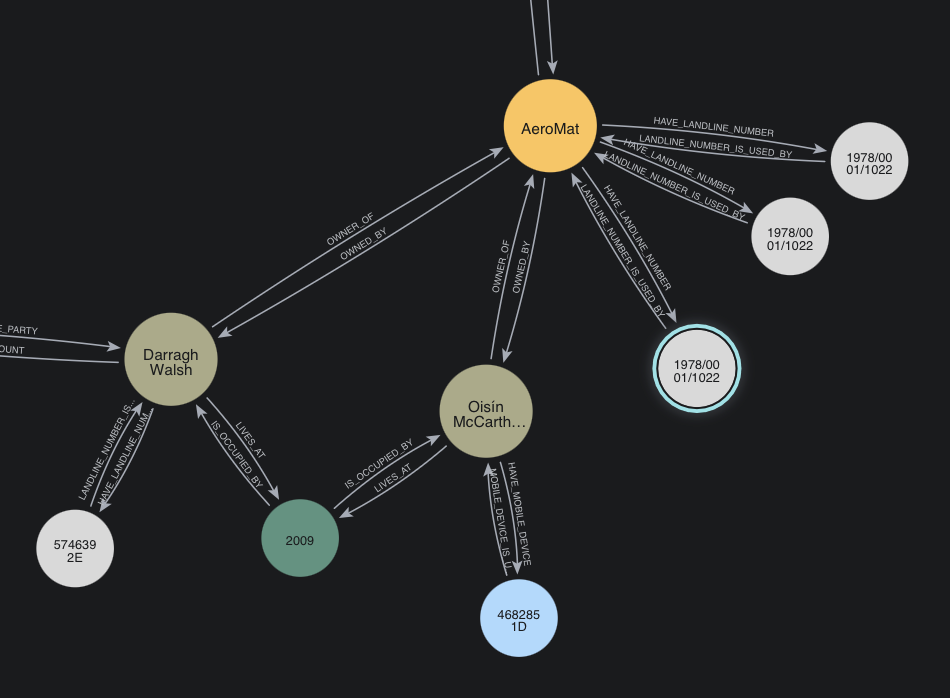


The above can be replicated by following the following GIT REPO: [financial\_txn\_flow\_using\_graphDB](https://github.com/georgelza/financial_txn_flow_using_graphDB.git)

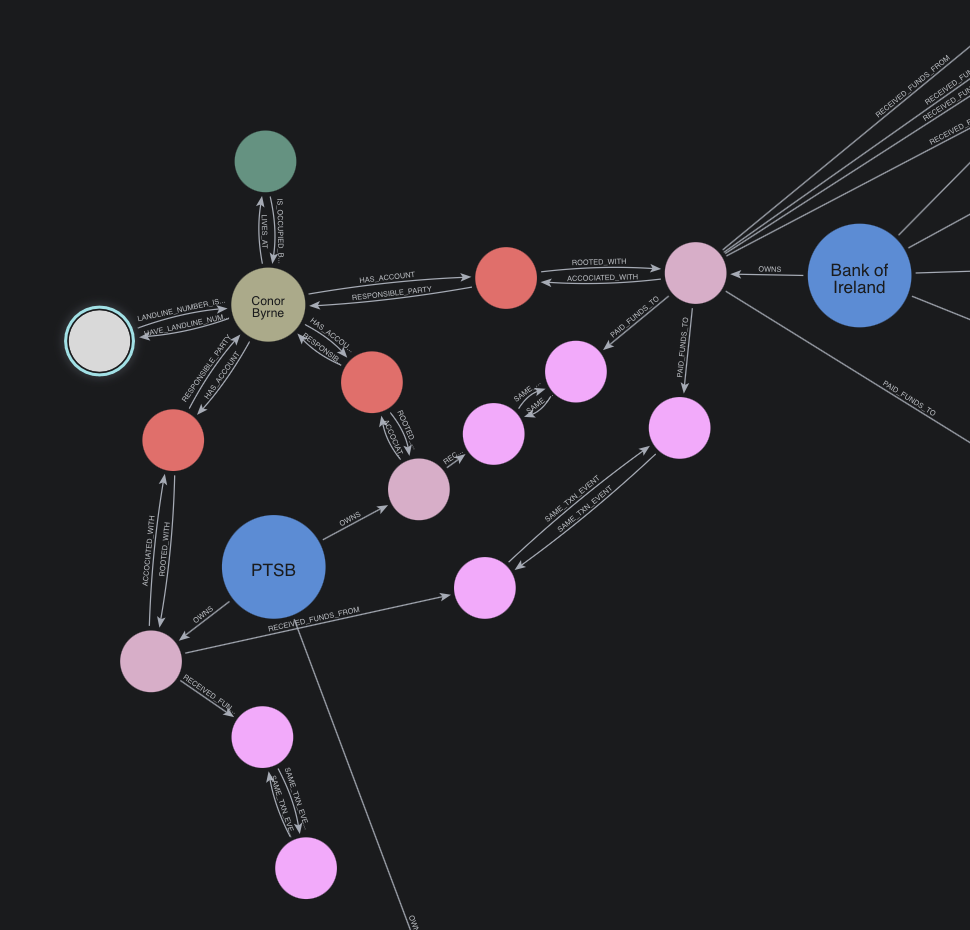


At the simplest here we’ve showing an individual as “has\_account” at UBI and reverse edge being “responsible\_party” for an account, we can further see the mobile devices, address and landlines associated with Cian.

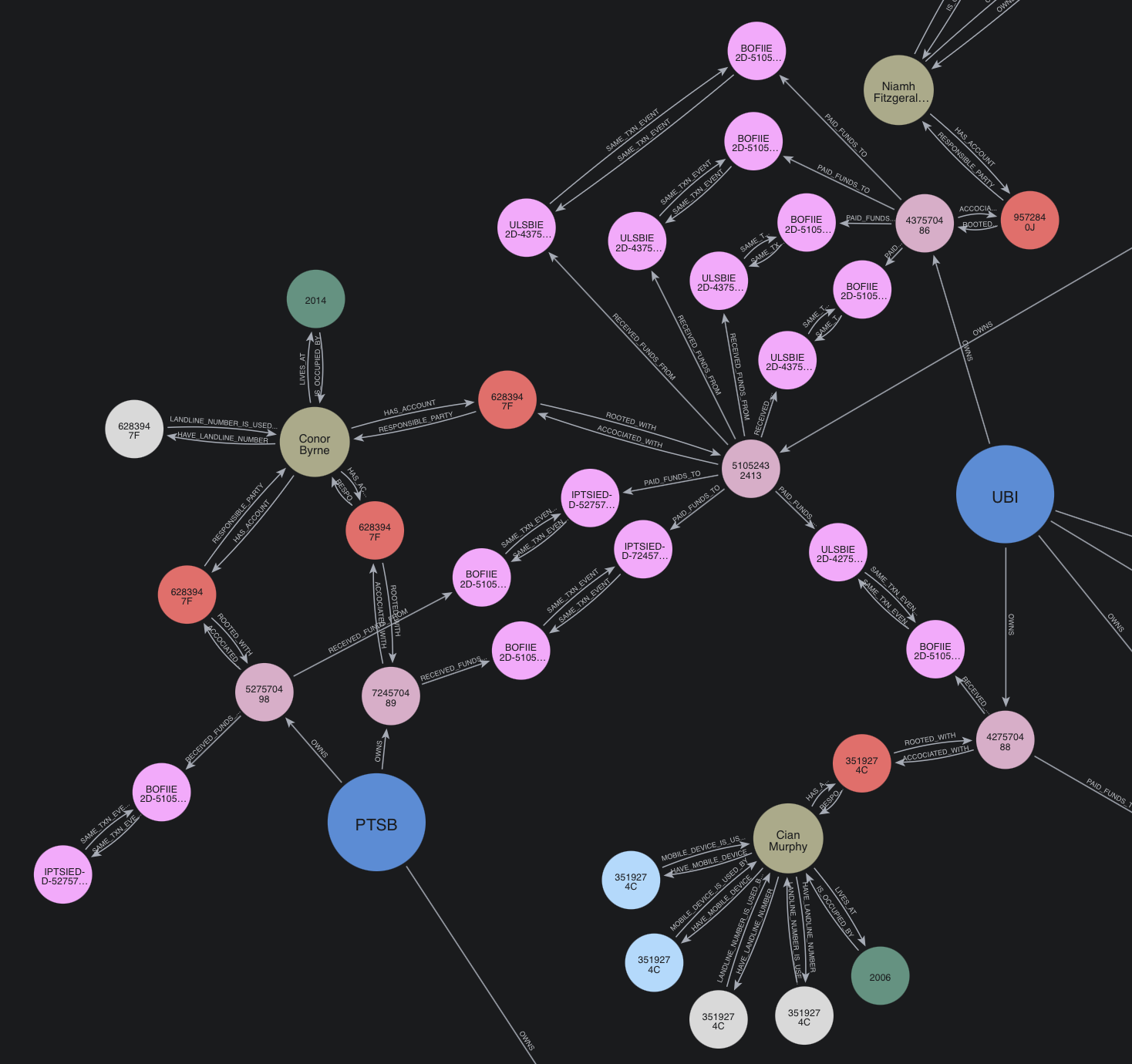
If we now take a closer Look:



Here we can see that Darragh and Oisin stay at the same address and both work at AeroMat.



With the above diagram things is starting to become interesting, we will now observe Connor have 3 bank accounts, 2 at PTSB and one at Bank of Ireland. We can also see landline number, mobile devices and physical address associated with Conor.



This is for me the most interesting diagram, notice the flow of funds via various account between Conor and Niamph.

And that, at the simplest level is the power of GraphDB’s, exposing relationships, intersect points.

Basic Introduction to GraphDB

* [Learning Graph Databases - Part 1](https://medium.com/@wrisovkarmakar/cypher-on-aws-neptune-graph-db-ef20484a3e29)
* [Learning Graph Databases - Part 2](https://medium.com/@wrisovkarmakar/learning-graph-databases-c5aa54bb9807)
* [Learning Graph Databases - Part 3](https://medium.com/@wrisovkarmakar/learning-graph-databases-a694f43ea0db)

Popular GraphDB Technologies

* [Neo4J](https://neo4j.com/product/graph-data-science/))
* [Amazon Neptune](https://aws.amazon.com/neptune/)
* [ArangoDB](https://arangodb.com/)
* [Azure Cosmos DB](https://azure.microsoft.com/en-us/products/cosmos-db)
* [JanusGraph](https://janusgraph.org/)
* [TigerGraph](https://www.tigergraph.com/) and
* [OrientD](https://orientdb.dev/)
* …

As per a cursory Google search ;)

Here's a more detailed look at some of the leading graph database platforms:

1. Amazon Neptune: A fully managed graph database service, Amazon Neptune supports both Property Graph and RDF models, offering flexibility for different graph modelling needs. It's a good choice for cloud environments and regulated industries.
2. Neo4j: Often considered the most popular graph database, Neo4j is a native graph database known for its performance, scalability, and support for ACID transactions. It uses the Cypher query language, which is optimized for graph traversal and querying. It can be run separately as a server deployment or embedded into application.
3. ArangoDB: A multi-model database, ArangoDB supports graph, document, and key-value data models. It uses the AQL query language, allowing for complex graph traversals and joins.
4. Azure Cosmos DB: Another multi-model database, Azure Cosmos DB offers Gremlin API support (based on TinkerPop) for graph needs within Microsoft stacks.
5. JanusGraph: A distributed, open-source graph database designed for large-scale graphs, JanusGraph supports various storage backends like Apache Cassandra and HBase.
6. TigerGraph: A native graph database, TigerGraph focuses on deep link analytics and real-time recommendations.
7. OrientDB: A multi-model database that combines graph and document models.
8. [Memgraph](https://memgraph.com/): An in-memory graph database that excels in real-time analytics and stream processing.
9. [Stardog](https://www.stardog.com/): A knowledge graph platform that combines graph database capabilities with semantic reasoning.
10. [Dgraph](https://docs.hypermode.com/dgraph/overview): A distributed graph database designed for scalability and performance.
11. Apache Iceberg with a GraphDB plugin module. This is a very interesting option as Apache Iceberg is #1 a Lakehouse option in itself, tightly integrates with Apache Flink and is further being included in Confluent’s Kafka cluster as whats sold as their TableFlow option (and oh, they also include Apache Flink as part of this stack).

When choosing a graph database, consider factors like the size and complexity of your data, the types of queries you need to perform, your existing infrastructure, and the level of support and scalability required.

Summary

At this stage, we’re just scratching the surface of what’s possible. For those that follow my blogs will know I love real time data, streaming data. There as valuable as the insight that can be derived from fresh accurate data.

If we were to bring that dimension also into scope, using technologies like i.e.:

* [Apache Kafka](https://kafka.apache.org/) (with either direct publish onto a topic or using Apache Kafka Connect framework to source data),
* [Apache Flink](https://flink.apache.org/) for streaming processing (think [PyFlink](https://nightlies.apache.org/flink/flink-docs-master/docs/dev/python/overview/) calling external API’s to compute Embeddings), ye we can store embeddings values also into GraphDB’d.
* Add to that additional technologies like Document DB stores for our semi and unstructured data (we’re talking the JSON payloads and bank account statements etc) into e.i.:
  + [DocumentDB](https://aws.amazon.com/documentdb/) by Amazon or
  + [Mongo Atlas](https://www.mongodb.com/) by MongoDB for storing our unstructured data,
* Ingesting additional reference data from distributed data sources using Database CDC (Change Data Capture) technologies by [Debezium](https://debezium.io/).

#### All coming together to create a highly enriched real time data stream feeding into our GraphDB datastore, now further allowing us apply AI/ML, GNN’s ([Graph Neural Networks](https://neo4j.com/blog/developer/demystifying-graph-neural-networks/)), SNA (Social Network Analysis ), etc to expose patterns, relationships, the world quickly becomes very interesting.

The Lab itself is not complete, I still need to expand it to cater for the temporal view, meaning the ability to see the flow of events over time, at a point in time, current or in the pass and model how the flow changed over a time line.

Further Reading

Now that I got your attention, some further reading.

[Neo4J: industry-use-cases/data-models/transactions/transactions-base-model](https://neo4j.com/developer/industry-use-cases/data-models/transactions/transactions-base-model/?_gl=1*10za4cz*_gcl_au*MTc2MjA3MzA3NS4xNzUzMjY3Mzc5*_ga*NzU1MTc3ODQwLjE3NTMyNjczNzk.*_ga_DL38Q8KGQC*czE3NTMzNzE3MDUkbzUkZzEkdDE3NTMzNzE4OTUkajUwJGwwJGgw*_ga_DZ)

[A Comprehensive Guide to Temporal Graphs in Data Science](https://www.analyticsvidhya.com/blog/2023/12/a-comprehensive-guide-to-temporal-graphs-in-data-science/)

[Mastering Fraud Detection With Temporal Graph Modeling](https://neo4j.com/blog/developer/mastering-fraud-detection-temporal-graph/)

[Build real-time fraud detection solutions using Amazon Neptune ML](https://aws.amazon.com/blogs/database/build-a-real-time-fraud-detection-solution-using-amazon-neptune-ml/)



**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>