# Synthetic Demographic Population Data Generator.

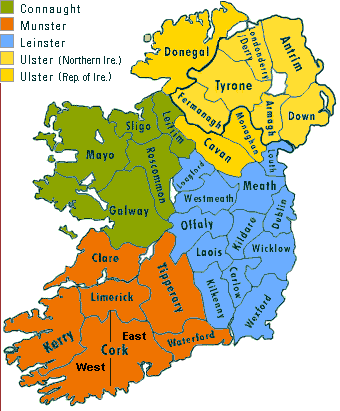
(17 Aug 2025), updated (18 Aug 2025)

# Overview

As part of a [previous blog](https://medium.com/@georgelza/fraud-analytics-using-a-different-approach-graphdb-data-platform-part-1-807c68d03bff) I found in addition to synthetic financial transactions, I also needed a population to represent. It all started very innocently, but the more I worked the deeper the little Rabbit hole became.

So, what we have here is Ireland (see data/ireland.json). Which is comprised out of 4 provinces and 28 counties and multiple cities and towns.

Provinces:



Counties:



For each we have a population count which is used as a weighting, i.e. when, well we at not so random picking a province the population % out of the total populations across the 4 provinces implies the weighting. Similar concept is used for counties in the province and cities/towns in the county. From here we simply then compile an address for the province, for the county for the town.

Next, we have data/ie\_banks.json. This paints the picture of the 3 national banks and then 3 additional prominent international banks that provide services in Ireland. Now we know there are many more, and you’re welcome to localize this to your liking. You will notice we also have an array of card networks to which the banks are subscribed, and a weighting for each.

The rest of the generation is driven by app/option\_lists.py, simply, the various name/value pairs are the options. When the weighted selection is called the scale is passed in (the scale is the sum of the values from the pick list).

From the app/option\_lists.py we retrieve a structure:

age\_distribution = [

{"name": 20, "value": 0.17, "count": 875353},

{"name": 30, "value": 0.20, "count": 1029827},

{"name": 40, "value": 0.17, "count": 875353},

{"name": 50, "value": 0.19, "count": 978336},

{"name": 60, "value": 0.16, "count": 823862},

{"name": 70, "value": 0.11, "count": 566405}

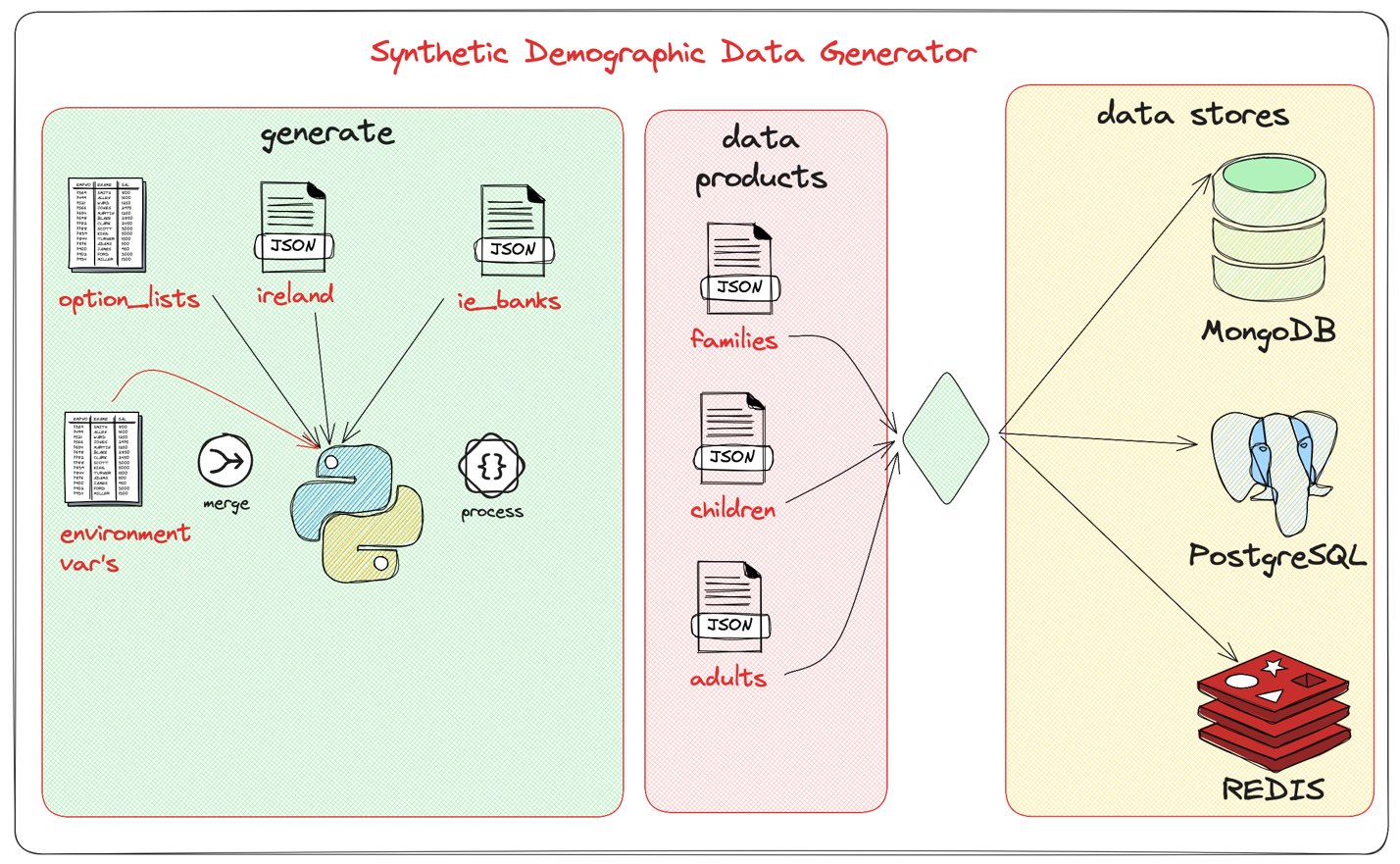
]

The above is used as the primary driver. We create the respective number of records per age group, with a weighting/distribution as per the value column, capped as per the count column.

We follow this pattern per age bracket, other for 20 it implies aged 20-30 so we go 30 years back. Then determine how records need to be created (count column), we then look at run.sh for the batchsize value, which then determine the number of batches (days/dates) for which we can create records.

We then take this number of dates and uniformly distribute them over the 10-year period and create the batch size of records for that date… and so on.

[GIT Repo - Synthentic\_Demographic\_Generator](https://github.com/georgelza/Synthentic_Demographic_Generator.git)



As it stands, storing generating and storing the data I was able to accomplish the below numbers on my ARM based M1 based Macbook Pro. (The below includes all 3 primary data products as per below):

* [MongoDB](https://www.mongodb.com/), 5850 complete data products / second – (5.25GB disk used)
* [PostgreSQL](https://www.postgresql.org/) 2450 complete data products / second – (8.8GB disk used)
* [REDIS](https://redis.io/) 6500 complete data products / second, I have seen 8000/sec (151MB disk used)
* [KAFKA](https://kafka.apache.org/) 6550 complete data products / second – 7.34GB on disk

Now realize this is both the parents as individual adult records, all the children as individual records and the family record, so even if the number does not look amazing, it’s quite a bit higher than what’s seen at first glance. See the logs/ directory for an output of all the structures as per current configuration.

Now I simply selected a batchsize of 400, I might have been able to get better performance by increase or decreasing this number.

Next up is a high-level Overview of our various payload structures created and used:

We will be storing:

* Adults,
* Children and
* Families

Into our selected persistent datastore (currently we have [MongoDB](https://www.mongodb.com/), [PostgreSQL](https://www.postgresql.org/), [REDIS](https://redis.io/) and [KAFKA](https://kafka.apache.org/)) supported. The framework is easily extended by adding the additional stores into connections.py. Do take note you will need make some changes to utils.py and various other files also…

Adults:

{

"\_id": "fa0a58a1-3fb9-4dc4-950d-27c96caad683",

"name": "Lisa-Marie",

"surname": "Hearty",

"uniqueId": "1915492K",

"gender": "F",

"dob": "55/11/24",

"marital\_status": "Widowed",

"partner": "5687475E",

"status": "Deceased",

"account": [

{

"bank": "Allied Irish Banks, p.l.c.",

"bicfi\_code": "AIBKIE2D",

"swift\_code": "AIBKIE2D",

"accountNumber": "IE29AIBK93115289539755",

"accountType": "Savings/Deposit"

},

{

"card\_holder": "L Hearty",

"card\_number": "4223843492070632",

"exp\_date": "11/25",

"card\_network": "Visa",

"issuing\_bank": "Permanent TSB"

}

],

"address": {

"street": "39 Fitzpatrick Street Street",

"town": "Limerick City",

"county": "Limerick",

"state": "Munster",

"post\_code": "X348394",

"country": "Ireland"

},

"family\_id": "344de5a9-fa58-4947-b887-73996047a228"

}

Children.

{

"\_id": "1b09d0fe-e57d-46a9-8be1-5f077abe33a2",

"name": "Lewis",

"surname": "O'Gara",

"gender": "Male",

"dob": "83/12/18",

"uniqueId": "9144598B",

"father\_idNumber": "1395127Q",

"mother\_idNumber": "6997438K",

"address": {

"street": "59 Jackson Street Street",

"town": "Fingal",

"county": "Dublin",

"state": "Leinster",

"post\_code": "E28 9K1C",

"country": "Ireland"

},

"family\_id": "c86bee1d-0b57-4dc4-abef-9215ecc0cb4c"

}

Families:

{

"\_id": "4004db21-05a7-4524-a990-1aa0203a0c24",

"husband": {

"name": "Oran",

"surname": "Hardiman",

"uniqueId": "6455409D",

"gender": "M",

"dob": "54/09/15",

"marital\_status": "Married",

"partner": "4486096P",

"status": "Living",

"account": [

{

"bank": "Bank of Ireland",

"bicfi\_code": "BOFIIE2D",

"swift\_code": "BOFIIE2D",

"accountNumber": "IE79BOFI90583835990934",

"accountType": "Current Accounts"

},

{

"bank": "Barclays Bank Ireland plc",

"bicfi\_code": "BARCIE2D",

"swift\_code": "BARCIE2D",

"accountNumber": "IE19BARC90100417221297",

"accountType": "Current Accounts"

},

{

"bank": "Citibank Europe plc",

"bicfi\_code": "CITIIE2X",

"swift\_code": "CITIIE2X",

"accountNumber": "IE53CITI99003336666053",

"accountType": "Current Accounts"

},

{

"bank": "Bank of Ireland",

"bicfi\_code": "BOFIIE2D",

"swift\_code": "BOFIIE2D",

"accountNumber": "IE79BOFI90583879228923",

"accountType": "Current Accounts"

},

{

"bank": "Allied Irish Banks, p.l.c.",

"bicfi\_code": "AIBKIE2D",

"swift\_code": "AIBKIE2D",

"accountNumber": "IE29AIBK93115217364160",

"accountType": "Current Accounts"

},

{

"bank": "Barclays Bank Ireland plc",

"bicfi\_code": "BARCIE2D",

"swift\_code": "BARCIE2D",

"accountNumber": "IE19BARC90100495214836",

"accountType": "Business Accounts"

},

{

"card\_holder": "O Hardiman",

"card\_number": "4295106831188124",

"exp\_date": "11/25",

"card\_network": "Visa",

"issuing\_bank": "Allied Irish Banks, p.l.c."

}

]

},

"wife": {

"name": "Paula",

"surname": "Hardiman",

"uniqueId": "4486096P",

"gender": "F",

"dob": "57/12/22",

"marital\_status": "Married",

"partner": "6455409D",

"status": "Living",

"account": [

{

"bank": "Bank of America Europe DAC",

"bicfi\_code": "BOFAIE3X",

"swift\_code": "BOFAIE3X",

"accountNumber": "IE58BOFA99006105822922",

"accountType": "Savings/Deposit"

},

{

"bank": "Bank of Ireland",

"bicfi\_code": "BOFIIE2D",

"swift\_code": "BOFIIE2D",

"accountNumber": "IE79BOFI90583844975923",

"accountType": "Current Accounts"

},

{

"card\_holder": "P Hardiman",

"card\_number": "4197997192488510",

"exp\_date": "11/25",

"card\_network": "Visa",

"issuing\_bank": "Permanent TSB"

},

{

"card\_holder": "P Hardiman",

"card\_number": "4341141958700744",

"exp\_date": "08/27",

"card\_network": "Visa",

"issuing\_bank": "Permanent TSB"

}

]

},

"address": {

"street": "78 Moy Street Street",

"town": "Limerick City",

"county": "Limerick",

"state": "Munster",

"post\_code": "C98FCAC",

"country": "Ireland"

}

}

**Supporting Structures:**

Address:

{

"street": "78 Moy Street Street",

"town": "Limerick City",

"county": "Limerick",

"state": "Munster",

"post\_code": "C98FCAC",

"country": "Ireland"

}

Account:

[

{

"bank": "Bank of America Europe DAC",

"bicfi\_code": "BOFAIE3X",

"swift\_code": "BOFAIE3X",

"accountNumber": "IE58BOFA99006105822922",

"accountType": "Savings/Deposit"

},

{

"bank": "Bank of Ireland",

"bicfi\_code": "BOFIIE2D",

"swift\_code": "BOFIIE2D",

"accountNumber": "IE79BOFI90583844975923",

"accountType": "Current Accounts"

},

{

"card\_holder": "P Hardiman",

"card\_number": "4197997192488510",

"exp\_date": "11/25",

"card\_network": "Visa",

"issuing\_bank": "Permanent TSB"

},

{

"card\_holder": "P Hardiman",

"card\_number": "4341141958700744",

"exp\_date": "08/27",

"card\_network": "Visa",

"issuing\_bank": "Permanent TSB"

}

]

The idea is to re-purpose all of this into our previous [Graph DB](https://medium.com/@georgelza/fraud-analytics-using-a-different-approach-graphdb-data-platform-part-1-807c68d03bff) blog series as a next “part #”, as our demographic data set onto which we then create synthetic financial transactions using the created accounts.

Additionally… [Apache Fluss](https://fluss.apache.org/) has a Python client library, implying we will be able to post these payloads directly on an Apache Fluss cluster (Support for complex datasets is part of the priority list).

As it stands, I am currently inserting the records directly into the data store, it won’t be difficult to use [Apache Flink](https://flink.apache.org/) and the build in [Debezium CDC](https://nightlies.apache.org/flink/flink-cdc-docs-master/) engine to ingest the data into [Apache Flink](https://flink.apache.org/) for analytic processing, exposing to [Apache Fluss](https://fluss.apache.org/) or [Apache Kafka](https://kafka.apache.org/) (which has the entire Apache Kafka Connect framework) or sinking the data products directly into Lakehouse based analytical platforms, like [Apache Paimon](https://paimon.apache.org/), [Apache Iceberg](https://iceberg.apache.org/) etc.

Re the Application, I always start simple… code procedural primarily. This time round, once I was done I dipped my fingers into a little bit of [Claude.ai](https://claude.ai/new) and [Google Gemini](https://gemini.google.com/app) to refactor some of the code into Python OO based classes, as per the README.md, where it made sense for me, where it made it more elegant, for no other reason…

And before anyone screams GDPR… you will be able to see everything is done using faker and prompting it to create data random, with some weightings… the Account information and the Credit cards are all fake.

I’ve structure everything in such a way so that the user can localise it onto their own locale easily.

To run all of this start by reading the root folder located README.md and take it from there.



# And In Summary.

Now… that was fun… Got to say thank you to ChatGP (not) and definitely [Gemini](https://gemini.google.com/app), maybe, [Claude](https://claude.ai/new), definitely…

But in the end, they gave general direction, with still allow of work required by the biological… ;)

And all of this led to an idea for another Blog ;) Keep watching this space.

As always, I’m predictable, but I really do think this is pretty neat… Hope you enjoyed the exploration.

Good luck, this is all fraught with rabbit holes, as always, so many and you can disappear so easily… but then that’s ½ the fun.



*Note: to execute this blog start with README.md located in the root folder and work from there, it will tell you exactly what to execute in which order to download all the dependencies and build everything. If you have any problems, welcome to reach out to me via one of the below profiles.*

**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://www.linkedin.com/in/george-leonard-945b502/>

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