

1. Generations Study Questionnaire Data ETL

Repo for maintaining JSON schemas, scripts, and non-PII quality check (QC) outputs to produce derived datasets for Generations Study (GS) data.

This README was last updated 12/11/2025.

2. Overview & Background

The Generations Study questionnaire data began collection in 2004 on paper using Optical Character Recognition (OCR) to read each questionnaire into database storable data. This methodology for data collection continued throughout baseline data collection across the cohort. This ETL process aims to better document the current state of the data, update and simplify the data processing, and make the derivation methodology of variables from the raw data available to the public.

This ETL creates data and metadata in JSON and JSON schema, respectively. It is advised that, if the user is new to JSON formats, before attempting to work with the data, the user reads through and completes the exercises in the following JSON reference lesson: Introduction to JSON.

3. Data Scope

As of the last update this ETL applies to all Baseline (R0) questionnaire data. There are 19 raw sections in the ETL, plus an additional raw derivation section for variables that need to be derived within the secure server (Trusted Research Environment, TRE) to preserve participant identities. The raw data read in covers over 1,850 different questions from the SQL database they are stored in, while the output processed data covers over 950 variables in JSON format due to aggregation of date fields and the removal of variables that potentially contain Personally Identifying Information (PII). Following is the list of all raw sections of baseline data: - Alcohol, Smoking & Diet - Birth Details - Breast Cancer - Breast Disease - Cancer in Relatives - Contraceptive & Hormone Replacement Therapy - General Information - Jobs - Mammograms - Medical History Cancers & Benign Tumors - Medical History Drugs & Supplements - Medical History Illnesses - Menstrual & Menopause - Other Breast Surgery - Other Lifestyle Factors - Physical Activity - Physical Development - Pregnancies - X-Rays

The raw derivation step uses the Mailing database (with variables `ADOB`, `EventDate`, `Random` and `TCode`) to create a small set of entry variables that are safe to export: shifted date of birth (`DOB`), unshifted year of birth (`YOB`), shifted date of entry (`EntryDate`), unshifted year of entry (`EntryYear`), and age in whole years at entry (`AgeEntry`)

4. Repository Structure

```
Questionnaire/  
  R0/  
    json_schemas/  
      post_pii/  
        raw/  
      scripts/  
      validation/  
        <SectionName>_ValidationSummary/
```

- `json_schemas/`

Schemas that define the expected JSON structure for each questionnaire and document the provenance of the variable.

- `raw/`

Original “as-collected” schemas for each section (before pseudo-anonymisation and date aggregation/derivation).

- `post_pii/`

Post-processing schemas that describe the final pseudo-anonymised output of R0 non-derived variables:

- * `StudyID -> TCode.`
 - * Raw date components aggregated to pseudo-anonymised complete dates.
 - * PII fields dropped.

- `scripts/`

Code to run the ETL and the raw derivation step (both executed in the TRE, but independent of each other):

- Questionnaire ETL consists of `run_all_sections.py` plus helper modules (`cleaning_utils.py`, `nested_utils.py`, `pseudo_anon_utils.py`, `schema_utils.py`, `qc_utils.py`, etc.) to:

- * extract from SQL
 - * pivot/clean
 - * restructure to nested JSON
 - * pseudo-anonymise and validate against the post-PII schemas.

- Raw derivation – `RawDerivation.ipynb` uses the Mailing database to derive a small set of pseudo-anonymised sensitive entry variables before export:

- * `DOB` (shifted date of birth, YYYY-MM-DD)
 - * `YOB` (unshifted year of birth)

- * EntryDate (shifted entry date, YYYY-MM-DD)

- * EntryYear (unshifted entry year)

- * AgeEntry (age in whole years at entry)

These are written to a JSON keyed and linked by TCode and then used alongside the section-level questionnaire outputs.

- validation/

Per-section QC outputs:

- <SectionName>_ValidationSummary/ folders hold:

- * resolver index (raw -> schema field mapping)

- * JSON Schema validation output

- * QC JSONs such as `variable_check.json` and `value_reconciliation.json`.

5. Pipeline Flow

Raw processing ETL step-by-step: 1. Extract from source (SQL). 2. Pivot/standardise variables. 3. Clean and type-cast according to schemas. 4. Restructure into nested JSON. 5. Pseudo-anonymise. 6. Validate against new PII JSON schemas. 7. Output and QC reports.

Runing `RawDerivation.ipynb` occurs separately as its own task and is self contained in one script.

6. Installation & Prerequisites

Before running the ETL, ensure you have:

Python

- Python 3.10+.
- Recommended: Python 3.11 or later.
- Dependencies are listed in `requirements.txt`, and are:

```
pandas==2.3.3
numpy==2.3.4
matplotlib==3.10.7
SQLAlchemy==2.0.44
jsonschema==4.25.1
pyodbc==5.3.0
```

Operating system and environment

- A machine with access to the Windows network paths used in `config.py`.

- Ability to create and activate a virtual environment (e.g. `venv`, `conda`, `uv`).

Database & drivers

- Appropriate ODBC drivers installed, matching the `config.py` settings:
 - ODBC Driver 17 for SQL Server (32-bit)
 - SQL Server (64-bit)
 - Microsoft Access Driver for SQL.

7. Configuration

The paths for schemas, output data, QC, etc. are in `config_utils.py` which can all be updated by adjusting the base `delivery_process` string variable. You must also make sure you have a SQL account that you can connect through a Microsoft Access Driver.

8. Running the ETL

Run the `run_all_sections.py` script, and that will run the full ETL. For example, in VS Code, press the play button when opening the Python script. Or, using a Command Line Interface (CLI), `python run_all_sections.py`.

On current infrastructure, a full run for all sections takes on the order of a few hours. Runtime will vary by environment.

9. Schemas & Validation

The ETL is schema-driven. Every transformation step is designed to produce JSON that conforms to an explicit JSON Schema.

9.1. Schema layout

- Raw input schemas describe the structure of the section as it comes from the questionnaire (`schemas/raw/<SectionName>_JSON.json`).
- Pseudo-anonymised output schemas describe the final JSON written out (`schemas/pseudo_anon/<SectionName>_PseudoAnon.json`), after:
 - Replacing `StudyID` with `TCode`.
 - Aggregating and deriving combined date fields.
 - Removing pii and date components.
- Each pseudo-anon schema keeps a reference back to the raw schema (via `$defs`), so you can trace where fields came from.

9.2. How schemas are used in the ETL

This section will be most helpful for data managers and developers.

9.2.a. Load schema

For each section, the script loads the raw schema, and `load_schema` also resolves any `$ref` references so the rest of the pipeline sees a fully expanded schema.

9.2.b. Schema driven cleaning and types

The ETL uses the schema to drive the processing of data in the following ways:

9.2.b.1 Built-in keywords

The schema provides the following JSON built-in schema keywords that are extracted and used in cleaning processing (`process_nested_data`):

- expected JSON types to cast values to the correct type (`string`, `integer`, `number`, etc.)
- numeric bounds where values outside are set to null (`minimum`, `maximum`)
Only use maximum if it is a finite numeric scale or categorical numeric value, i.e. not continuous (day of the month, numerical value of month in the year, etc.)
Only use minimum where values cannot be negative (height, weight, age, etc.)
- allowed values that guide recodes of special values and unknown codes (`enum`)

9.2.b.2 Custom annotations

The schema provides the following custom annotations that are not built in to JSON validation, but help the user with processing or context:

- question ID that corresponds to the SQL metadata where questions that ask the same contextual question have the same ID (`questionID`)
- variable name as it corresponds to the SQL database, differs depending on level of variable (`name`):

Flat (non-array) fields

- use the original raw variable names from SQL
- are 1:1 with the human-readable variable names that were created when the data were first collected.

Because there is only one value per participant and it always refers to the same question, the schema can safely use the SQL name (or a stable equivalent).

Example - flat variable, 1:1 association:

- Question: "Have you ever been pregnant?"
- SQL column: Q5_3_1

No pattern recognition needed.

Array fields (repeated structures)

- use new, human-readable schema field names
- arrays are repeated structures (e.g. multiple drug regimens, pregnancies, jobs)
- the raw SQL variable names follow inconsistent patterns
- a single raw name cannot reliably represent all repeats

Using stable schema field names keeps the schemas compact and easier to understand.

Example - array variable (unreliable pattern in SQL naming):

Contraceptive pill name is a repeated question: each instance refers to a different pill.

Raw SQL columns:

First pill name: Q6_1_1_1

Second pill name: Q6_1_2_1

Third pill name: Q6_1_3_1

Fourth pill name: ocname4o

The first three appear to follow a pattern (Q6_1_{instance}_1), but the fourth switches to a different naming convention (ocname4o). There is no single, reliable pattern that covers all repeats.

Using the human-readable variable name, all of these are represented by the same pattern:

ContracepPill{instance}_Name

The array index (0, 1, 2, 3, etc.) captures which pill instance it is.

The **name** annotation removes the instance within the human-readable name and the processing picks up on the patterning and is able to find the correct raw variable name using just this:

ContracepPill_Name

- question as it was written in the questionnaire (**question**)

- description of the variable and question to help the user decipher the context and use of the variable (**description**)
- allowed value descriptions defining what numeric values mean (**enumDescriptions**)
- answer IDs where repeated answers have the same ID that correlate to the metadata (**answerID**)

9.2.c. Drive restructuring

The schema also defines where fields live in the nested structure (arrays, objects, index fields). `restructure_utils` uses this metadata, together with `nested_utils`, to build the final nested JSON objects that match the schema structure.

9.2.d. Pseudo-anonymised schema generation

After restructuring, `pseudo_anon_utils` takes the raw schema and:

- inserts new derived date fields
- removes raw date components and PII fields
- replaces `StudyID` with `R0_TCode`

The result is the pseudo-anonymised schema, which is what we validate the final output against.

9.2.e. Raw derivation inside the TRE

A small set of entry variables is derived separate to the main ETL run, also within the TRE:

These variables are written to `RawDerivedVariables.json` and then treated as part of the “processed-raw” inputs to the questionnaire ETL. The underlying identifying dates (`ADOB`, `EventDate`) and the `Random` offset never leave the TRE. More details on how the variables are derived are in the schema, `RawDerivedVariables_Schema.json`.

9.3. Validation process

Validation is performed using the JSON schema through a helper in `common_utils`.

Any validation errors are:

- logged with the record index and field path
- summarised for review in the ETL logs or in the CLI

If no errors, the section’s JSON is considered structurally valid.

9.4. Adding or updating schemas

When extending or updating the ETL:

1. Add/update the raw schema under `schemas/raw/`.
2. Regenerate the pseudo-anon schema via `pseudo-anon utilities`.
3. Run the ETL + validation for that section and review any schema errors.

10. Logging & QC

This ETL is designed to surface problems early via structured logs and a set of QC artefacts generated per section.

10.1. Runtime logging

- The ETL uses a standard Python `logging` logger configured in the entry scripts / notebooks.
- Typical log messages include:
- Start/end of each major stage (extract, pivot, clean, restructure, pseudo-anon, validate).
- Row counts before/after key operations.
- Summaries of validation and QC checks.

10.2. Change-tracking output

During cleaning, the ETL records any value-level changes (e.g. "1" -> 1, out-of-range -> null) into a change-tracking structure. Due to sensitivity of some of the data in the files, the change-tracking JSONs have not been uploaded to the repo.

- Saved to: `validation/<CHANGE_TRACKING_DIR>/<SECTION_SLUG>_change_tracking.json`
- Structure (per section):

```
{
  "R0_000001": {
    "R0_FieldName": [
      {"old": "1", "new": 1},
      {"old": -10, "new": null}
    ]
  }
}
```

Change-tracking is used by `qc_utils` to explain differences in value distributions before and after ETL.

10.3. Validation and resolver summaries

For each section, schema validation and variable resolution generate artefacts under a `ValidationSummary` directory at the end of each section's ETL.

- Root directory (per round): `validation/`

- Per-section subfolder:
 - `validation/<SectionName>_ValidationSummary/`

Inside each section's folder you will find:

- `<SectionSlug>_resolver_index.json`
- Built by `restructure_utils.build_resolver_cache_from_columns`
- Maps schema fields to the raw variables that populate them (and their index bands).

10.4. QC reports

Outputs include:

- Value frequency reconciliation:
 - Compares value counts in the raw pivot vs. the final JSON.
 - Uses change-tracking to explain where values changed.
 - Saved to: `validation/<SectionName>_ValidationSummary/value_reconciliation.json`
- Variable alignment check:
 - Maps SQL variable names to new, human readable variable names.
 - Utilizes the resolver.
 - Checks against removed date and PII fields to make sure all variables are accounted for.
 - Saved to: `validation/<SectionName>_ValidationSummary/variable_check.json`

These artefacts help answer “which raw variable ended up in which JSON field?”, “why did validation fail?”, “are all variables accounted for?”.

10.5. Interpreting failures

If a section fails QC or validation:

- Check the log file in for stack traces and high-level error messages.
- Inspect the JSON validation output to see which fields broke schema rules.
- Open the resolver cache (`<SECTION_SLUG>_resolver_index.json`) to verify that variables are mapped to the expected schema fields.
- Review change-tracking and QC outputs for unexpected occurrences:
- Look for unexpected large numbers of changes for a single field.

If necessary, re-run the section notebook with a higher log level (e.g. `DEBUG=True`) and a smaller sample size to iterate quickly on issues.

11. Data Privacy & Security

This ETL is designed to work with sensitive questionnaire data, so data protection is built into both the code and the workflow.

11.1 Pseudo-anonymisation

The pipeline never writes out raw identifiers directly from the source database and the data follows a similar pseudo-anonymisation process as the schema in 9.2.d:

- **StudyID** replaced pseudo-identifier stored in private server, **TCode**.
- Date components aggregated to derived dates.
 - Inside the TRE, **RawDerivation.ipynb** uses actual dates of birth and entry (**ADOB**, **EventDate**) together with the participant-specific **Random** offset to create:
 - * Shifted dates (**DOB**, **EntryDate**) that mask the true calendar dates but preserve within-person intervals.
 - * Unshifted year-only and age variables (**YOB**, **EntryYear**, **AgeEntry**) that are safe to export.
 - In the questionnaire ETL, individual day/month/year components from the raw R0 tables are aggregated to derived dates and then shifted using the same **Random** offset logic, with partial dates completed according to pre-defined rules as follows: aggregated dates with no day of the month are given the 15th day of the month, aggregated dates with no month values are given July, and if both day and month are missing the values given are the 1st of July. Then, the format that the date came in is preserved and used in the no-PII data, i.e. if the day was originally missing from the questionnaire and became the 15th for shifting, the day will not show in the no-PII data to protect the exact number of days that the participant has their dates shifted by. Only these derived, shifted or aggregated forms appear in the processed JSON; the original date components and **Random** remain on the secure server.
- Removal of direct PII data.
 - No variables that have identifying information, or potentially identifying information are included in the processed data
 - Includes names, addresses, names of towns, names of hospitals, and any variable that was asked as open ended text like medications, cancer types, and familial relationships.

11.2 Handling of real data

- This repo is intended to contain code and schemas only.
- Real questionnaire data, SID mappings, and any intermediate extracts must not be committed to Git or shared via this repository.
- Database connection details (servers, usernames, passwords) are removed in **config** and must not be committed to the repo.

11.3 Access control and usage

- This ETL is intended for use by authorised team members only, in compliance with:
 - The study’s data governance policies, and any applicable legal/regulatory requirements (e.g. GDPR for EU/UK datasets).
- Users running the ETL are responsible for:
 - Ensuring they have permission to access the underlying databases.
 - Not exporting or sharing outputs outside approved channels.

12. License / Usage

Internal ICR use only. Do not distribute or reuse without appropriate approvals.

13. Contributing

If you need to extend or modify the ETL, raise an issue or discuss changes with the data engineering / GS team.

14. Contacts

Please contact Tal Cohen, tal.cohen@icr.ac.uk, if you have any questions or concerns.