National Address Register (NAR)

1. Dataset Overview

The National Address Register (NAR) is a standardized list of civic addresses for Canada.
Each record describes:

- a **location**: a physical building, identified by `LOC_GUID`, and
- an **address**: an addressable unit inside that building (for example, a house, an apartment, or an office), identified by `ADDR_GUID`.

For a detached house, there is usually **one location and one address**.
For an apartment building or office tower, there is **one location with many addresses**.

The NAR address CSV files (such as `Address_10.csv`, `Address_24_part_1.csv` to `Address_62.csv`) share a common structure. They include:

- Building and address identifiers: `LOC_GUID`, `ADDR_GUID`
- Civic and unit details: `APT_NO_LABEL`, `CIVIC_NO`, `CIVIC_NO_SUFFIX`
- Official street information: `OFFICIAL_STREET_NAME`, `OFFICIAL_STREET_TYPE`, `OFFICIAL_STREET_DIR`
- Province and census subdivision information: `PROV_CODE`, `CSD_ENG_NAME`, `CSD_FRE_NAME`, `CSD_TYPE_ENG_CODE`, `CSD_TYPE_FRE_CODE`
- Mailing address fields: `MAIL_STREET_NAME`, `MAIL_STREET_TYPE`, `MAIL_STREET_DIR`, `MAIL_MUN_NAME`, `MAIL_PROV_ABVN`, `MAIL_POSTAL_CODE`
- Land survey and coordinates: `BG_DLS_QTR`, `BG_DLS_SCTN`, `BG_DLS_TWNSHP`, `BG_DLS_RNG`, `BG_DLS_MRD`, `REPPOINT_LATITUDE`, `REPPOINT_LONGITUDE`, `BG_X`, `BG_Y`
- Building usage and delivery information: `BU_USE`, `BU_N_CIVIC_ADD`

The project also provides reference information for **province codes** and **building usage codes**.
For example, province code `35` corresponds to Ontario, and building usage code `1` means "Residential".

The goal was to:

1. Analyze this raw data
2. Design a **3NF database schema**
3. Create an ERD in Mermaid
4. Write a **Python-only ETL script** that builds and populates the database in SQL Server.

---

2. Entities and Relationships

From the NAR attributes, I identified the following main entities:

1. **Province**
   - Each province or territory in Canada.
   - Attributes: `ProvinceCode`, abbreviation, English and French names.
   - Example: `35`, `ON`, "Ontario".
2. **BuildingUsage**
   - Describes how a building is used.
   - Attributes: `BU_USE`, English and French description.
   - Example codes: 1 = Residential, 2 = Partial Residential, 3 = Non-Residential, 4 = Unknown.
3. **CensusSubdivision (CSD)**
   - Represents municipalities or equivalent areas within a province.
   - Attributes: English and French names, type codes, and a foreign key to `Province`.
   - Many addresses share the same CSD.
4. **PostalCode**
   - Represents Canadian mailing postal codes.
   - Each postal code is linked to a province.
5. **Location**
   - A physical building or site, identified by `LOC_GUID`.
   - Attributes: land survey components, coordinates, building usage, and a link to `CensusSubdivision` and `Province`.
   - One location can have many addresses.
6. **Address**
   - An addressable unit (e.g., one apartment or one house), identified by `ADDR_GUID`.
   - Attributes: civic number, apartment label, official and mailing street details, mailing municipality name, mailing province abbreviation, postal code, and additional delivery information.
   - Each address belongs to exactly one location.

Key relationships:

- One **Province** has many **CensusSubdivisions**.
- One **Province** has many **PostalCodes**.
- One **Province** has many **Locations**.
- One **CensusSubdivision** has many **Locations**.
- One **BuildingUsage** category is used by many **Locations**.
- One **Location** has many **Addresses**.
- One **PostalCode** is used by many **Addresses**.

This results in a clear, 3NF design centred around `Location` and `Address`, with descriptive lookup tables around them.

---

3. Normalization Process (1NF, 2NF, 3NF)

3.1 First Normal Form (1NF)

To reach 1NF:

- Created a single staging table `Stg_NAR_Address` that matches the CSV columns.
- Stored each column as a single, atomic value (for example, `CIVIC_NO` and `CIVIC_NO_SUFFIX` are separate fields).
- Used appropriate data types, such as `NVARCHAR` for text, `INT` for numeric codes, and `FLOAT` for coordinates.

At this point, all data was in one large table with many repeated values, especially for:

- `PROV_CODE` (only a small set of province codes, repeated across millions of rows), and
- `CSD_ENG_NAME` + `CSD_TYPE_ENG_CODE` (the same census subdivision names reused for many addresses).

3.2 Second Normal Form (2NF)

NAR already provides unique identifiers:

- `ADDR_GUID` uniquely identifies an address.
- `LOC_GUID` uniquely identifies a location.

Chosen attributes:

- `ADDR_GUID` as the **primary key** of the `Address` table.
- `LOC_GUID` as the **primary key** of the `Location` table.

All attributes in `Address` (for example, `APT_NO_LABEL`, `CIVIC_NO`, `MAIL_POSTAL_CODE`) describe that specific address, not part of a composite key.
All attributes in `Location` (for example, `REPPOINT_LATITUDE`, `BU_USE`) describe that specific location.

For `CensusSubdivision`, created a surrogate key `CSD_ID` instead of using a composite key of name + type + province. This simplifies foreign keys and still maintains uniqueness.

3.3 Third Normal Form (3NF)

To reach 3NF, removed transitive dependencies and repeated descriptive data:

- Province names and abbreviations are determined by `ProvinceCode`, not by individual addresses.
  → Created a `Province` table and stored only `ProvinceCode` in other tables.
- Building usage descriptions are determined by `BU_USE`.
  → Created a `BuildingUsage` table with descriptions in English and French, and stored only `BU_USE` in `Location`.

- Census subdivision names and types depend on the combination of `CSD_*` attributes and `PROV_CODE`.
  → Created a `CensusSubdivision` table keyed by `CSD_ID`, and `Location` references `CSD_ID`.
- Mailing postal code and province are logically linked.
  → Created a `PostalCode` table keyed by `PostalCode`, with a foreign key to `Province`. The `Address` table stores only the `PostalCode` key.

After these changes:

- Every non-key attribute in each table depends only on that table's primary key.
- No attributes depend on other non-key attributes.
- Common descriptive values (province names, building usage descriptions, CSD names) are stored once in their own tables instead of repeated in every address row.

---

4. ETL Approach

For GA5, I chose the **Python option** instead of separate T-SQL script files.

This ETL is implemented in a single Python script `ga5_nar_etl.py` which uses the `pyodbc` library to connect to the SQL Server database `25F_CST2112`. The script performs the following steps:

1. **Connect to SQL Server**
   - Uses `pyodbc.connect(...)` with the provided server address, port, database name, username, and password.
2. **Create staging table**
   - Executes a `CREATE TABLE` statement (inside Python) to build `dbo.Stg_NAR_Address`, matching the CSV columns.
3. **Bulk-load all CSV files**
   - Runs `BULK INSERT` commands for each of the 26 files (`Address_10.csv`, `Address_11.csv`, …, `Address_62.csv`) located in `c:\cst2112_data\ga5_NAR`.
   - Uses `FIRSTROW = 2`, UTF-8 code page, and `TABLOCK` to handle large data efficiently.
4. **Create normalized 3NF tables**
   - Drops existing tables if they exist.
   - Creates:
     - `Province`
     - `BuildingUsage`
     - `CensusSubdivision`
     - `PostalCode`
     - `Location`
     - `Address`
   - All `CREATE TABLE` commands are sent from Python using `cursor.execute()`.
5. **Populate lookup tables**

    - `Province`: inserted from constant values based on the provided province code list.
    - `BuildingUsage`: inserted as constants for codes 1–4 (Residential, Partial Residential, Non-Residential, Unknown).
    - `CensusSubdivision`: inserted using `SELECT DISTINCT` from `Stg_NAR_Address`, grouped by CSD names, types, and `PROV_CODE`.
    - `PostalCode`: inserted using `SELECT DISTINCT MAIL_POSTAL_CODE, PROV_CODE` from `Stg_NAR_Address`.
6. **Populate Location table**
    - Inserts one row per unique `LOC_GUID`.
    - Joins staging data to `CensusSubdivision` (by `CSD_ENG_NAME` and `PROV_CODE`) to resolve `CSD_ID`.
    - Carries over land survey fields, coordinates, and `BU_USE` into `Location`.
7. **Populate Address table**
    - Inserts one row per `ADDR_GUID` from `Stg_NAR_Address`.
    - Copies address attributes and links each address to its `LOC_GUID` and `PostalCode`.

All of these steps are committed once at the end of the script, so the professor can run the Python script on an empty database and fully rebuild the schema and data without making manual edits.

---

5. Challenges and learning

1. Understanding the difference between location and address
At first, it was tempting to treat each row as just "an address". By reading the NAR description and looking at the attributes (`LOC_GUID` vs `ADDR_GUID`), I learned that:

- `LOC_GUID` represents the **building** (location), and
- `ADDR_GUID` represents the **address unit** inside that building.

This led to design two related tables: `Location` and `Address`, with a one-to-many relationship.

2. Handling large data (16.5 million rows)
The assignment notes that GA5 data is very large. I quickly realized that inserting rows one by one in Python would be far too slow. So learned to use `BULK INSERT` from Python, pointing to the shared data folder on the SQL Server. This is much faster and is the right tool for this volume of data.

3. Identifying repeated data for normalization
When I inspected the staging table, I saw the same `PROV_CODE` and `CSD_ENG_NAME` repeated across many rows. This gave us confidence that these should be separate entities (`Province` and `CensusSubdivision`) instead of staying in the address table. Also, saw that building usage codes and postal codes repeat many times, making them good candidates for lookup tables.

4. Keeping the model simple but meaningful
It was possible to normalize even more (for example splitting street name and type into separate linked tables), but I decided to focus on the most meaningful entities:

- Province
- CensusSubdivision
- BuildingUsage
- PostalCode
- Location
- Address