# BASES DE DATOS ANALÍTICAS

## Práctica 2

Noviembre 2025


**Víctor Suesta Arribas**

# ÍNDICE

# Ejercicio 1 — Diseño conceptual

En 1.1 se corrigen solo inconsistencias del modelo base; la multipropiedad y las sanciones se tratan en 1.2.

## 1.1 Revisión del modelo conceptual

En esta sección se revisa el modelo conceptual propuesto, identificando errores y omisiones en entidades, atributos y relaciones. Para cada incidencia se justifica la corrección con base en los requisitos del enunciado y en principios del modelo relacional (identificación unívoca, integridad referencial, dominios y cardinalidades consistentes). A continuación se presenta una tabla “Problema–Justificación–Corrección”.

| Problema detectado | Justificación | Corrección propuesta |
|---|---|---|
| VEHICLE referencia PROVINCE en lugar de MUNICIPALITY | La residencia del vehículo se define por municipio; la relación con provincia no refleja el nivel real de localización. Teoría: la FK debe apuntar a la entidad que materializa el requisito. | Cambiar la relación a VEHICLE → MUNICIPALITY y eliminar la dependencia directa con PROVINCE. |
| OWNER sin identificador único ni documento oficial | Requisito: propietario unívocamente identificable; teoría: toda entidad debe tener PK y un atributo identificador con unicidad. | Añadir owner_id (PK) y document_id (UNIQUE, NOT NULL) que admita DNI/NIE/Pasaporte. |
| DGT_TYPE sin clave primaria definida (tipo/subtipo) | Toda entidad de referencia necesita PK. En los datos de carga, un mismo cod_subtype_dgt puede repetirse bajo distintos cod_type_dgt, por lo que no es único por sí solo. | Definir PK compuesta (cod_type_dgt, cod_subtype_dgt) y referenciar desde VEHICLE por ese subtipo (o por la PK compuesta). |
| VEHICLE no vincula explícitamente su categoría DGT por clave | Teoría: la clasificación DGT debe ser referencial, no un texto suelto, para garantizar consistencia y permitir consultas por tipo/subtipo. | Incluir FK VEHICLE → DGT_TYPE por el código de subtipo (o por la PK definida en DGT_TYPE). |
| Dominios y restricciones no documentados (p. ej., new_used) | Buenas prácticas: explicitar dominios para integridad semántica (evitar valores fuera de catálogo). | Documentar new_used ∈ {'N','U'} y demás dominios relevantes (fechas válidas, longitudes de códigos). |

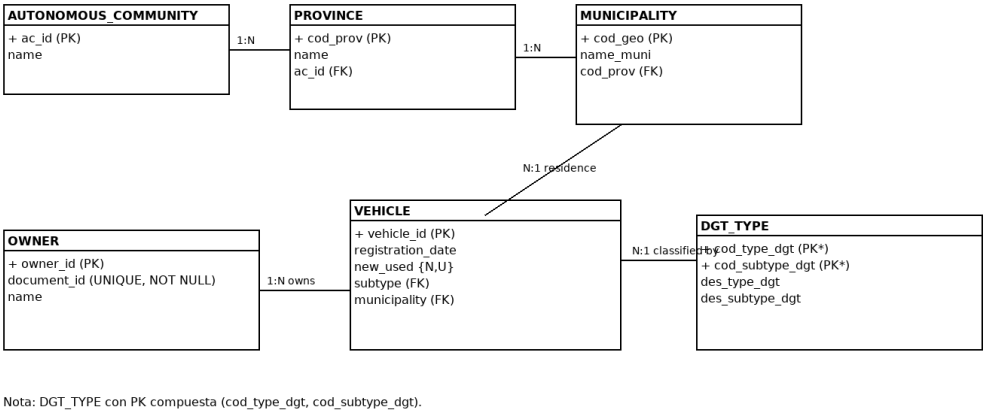



Figura 1. Diagrama E/R corregido. DGT_TYPE con PK compuesta (cod_type_dgt, cod_subtype_dgt).

# 1.2 Evolución del modelo conceptual

- Multipropiedad. Se solicita permitir varios propietarios por vehículo. Para ello se introduce la entidad intermedia OWNERSHIP que materializa una relación N:M entre OWNER y VEHICLE. Su PK compuesta es (owner_id, vehicle_id).

- Sanciones y conductor opcional. Se solicita registrar sanciones por vehículo y municipio del hecho, pudiendo identificar el conductor si se conoce. Se añaden las entidades DRIVER (con driver_id y document_id único) y FINE con fine_id, fine_date, vehicle_id (FK), cod_geo de MUNICIPALITY (FK) y driver_id (FK nullable).

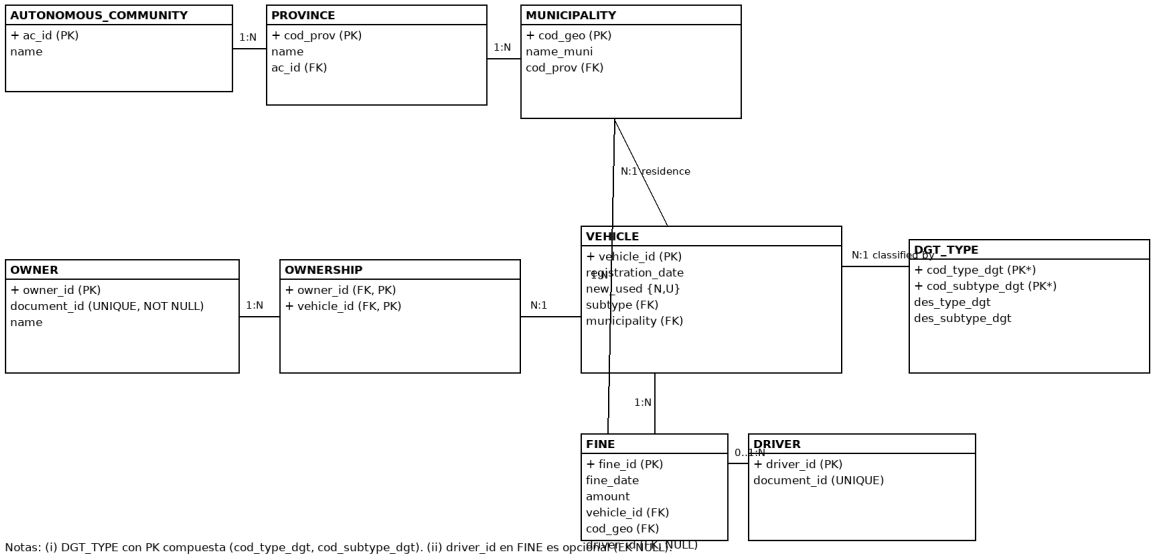



Figura 2. Diagrama E/R evolucionado. DGT_TYPE con PK compuesta.

# Ejercicio 2 — Diseño lógico

## 2.1 Esquema lógico final (PK/AK/FK)

DGT_TYPE (dbo.dgt_type)

- PK: (`cod_type_dgt`, `cod_subtype_dgt`)
- Atributos: `des_type_dgt`, `des_subtype_dgt`

MUNICIPALITY (dbo.municipalities)

- PK: (`cod_geo`)
- Atributos: `name_muni`, `province`, `population_muni`, `altitude`, …

OWNER (dbo.owner)

- PK: (`owner_id`)
- AK: (`document_id`) [UNIQUE, NOT NULL]
- Atributos: `name`, …

VEHICLE (dbo.vehicles)

- PK: (`id`)
- FK1: (`municipality`) → MUNICIPALITY(`cod_geo`)
- FK2: (`type`, `subtype`) → DGT_TYPE(`cod_type_dgt`, `cod_subtype_dgt`)
- Dominios: `new_used` ∈ {`'N'`, `'U'`}
- Atributos: type, subtype, brand, model, registration_date

OWNERSHIP (dbo.ownership)

- PK: (owner_id, vehicle_id)
- FK1: (owner_id) → OWNER(owner_id)
- FK2: (vehicle_id) → VEHICLE(id)
- Atributos opcionales: ownership_start_date, ownership_end_date

DRIVER (dbo.driver)

- PK: (driver_id)
- AK: (document_id) [UNIQUE]
- Atributos: license_issue, license_expiry, ...

FINE (dbo.fine)

- PK: (fine_id)
- FK1: (vehicle_id) → VEHICLE(id)
- FK2: (cod_geo) → MUNICIPALITY(cod_geo)
- FK3: (driver_id) → DRIVER(driver_id) [NULLABLE]
- Atributos: fine_date, amount

## 2.2 Justificación de cambios

- VEHICLE → MUNICIPALITY (FK a cod_geo): la residencia se define por municipio; garantiza integridad y consultas correctas por localidad.
- VEHICLE → DGT_TYPE (FK a (type, subtype)): la clasificación DGT es referencial; permite filtrar por tipo/subtipo sin incoherencias.
- Multipropiedad: se modela N:M mediante OWNERSHIP con PK compuesta; evita duplicidad y admite trazabilidad temporal si se usan fechas.
- Sanciones: FINE depende de VEHICLE y del MUNICIPALITY del hecho; DRIVER es opcional (FK nullable) para casos sin conductor identificado.
- Unicidades civiles: document_id en OWNER y DRIVER como AK para identificación unívoca.
- Dominios: new_used ∈ {'N','U'} para integridad semántica.

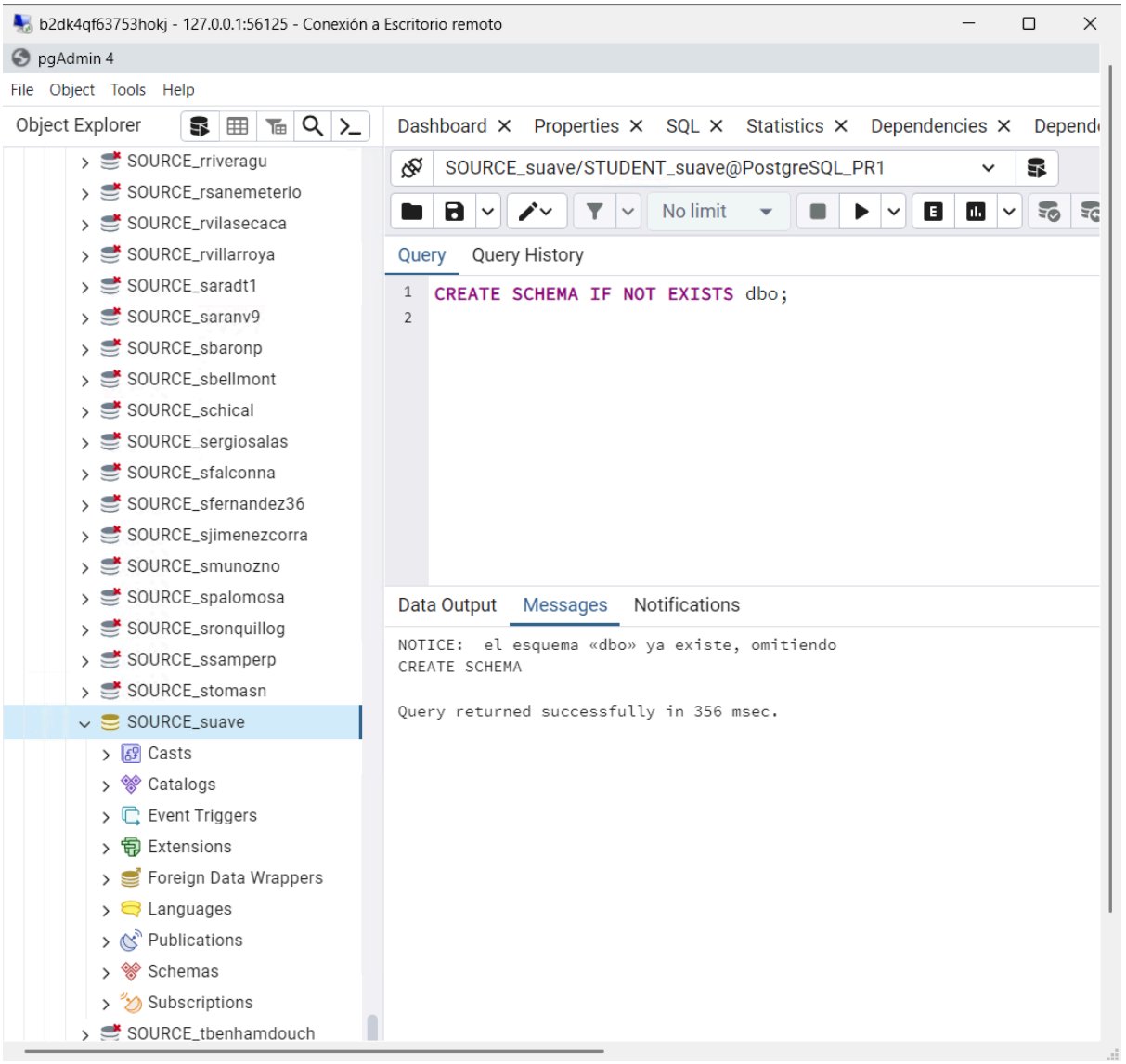
La PK de DGT_TYPE es compuesta (`cod_type_dgt`,`cod_subtype_dgt`); por ello, en el lógico la FK de VEHICLE se modela sobre (type, subtype). En el físico proporcionado por los scripts, el atributo `type` no existe y las uniones operativas se realizan por `subtype`, pero el diseño lógico correcto exige la clave compuesta.

# Ejercicio 3 — Modelo físico SQL

## 3.1 Carga de datos y verificación (con evidencias)

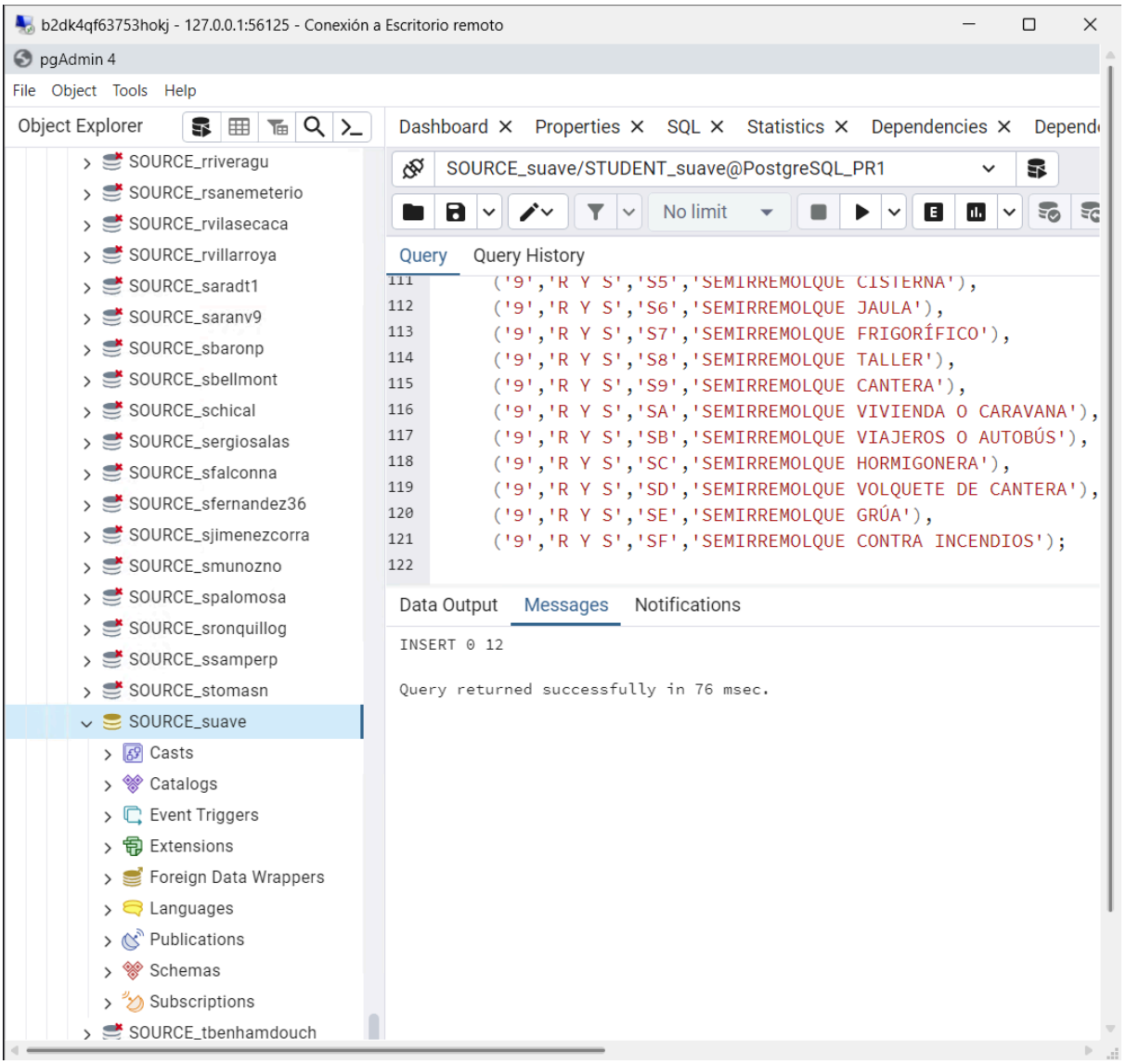
Entorno: PostgreSQL (pgAdmin 4). Se ha creado el esquema dbo y ejecutados los scripts dgt_type.sql, municipalities.sql y vehicles.sql en este orden. En esta subsección se documentan las evidencias de creación del esquema y de carga/verificación de datos. Para cada evidencia se incluye la captura de pantalla y un pie con el identificador (E1–E6).

# Evidencia 1



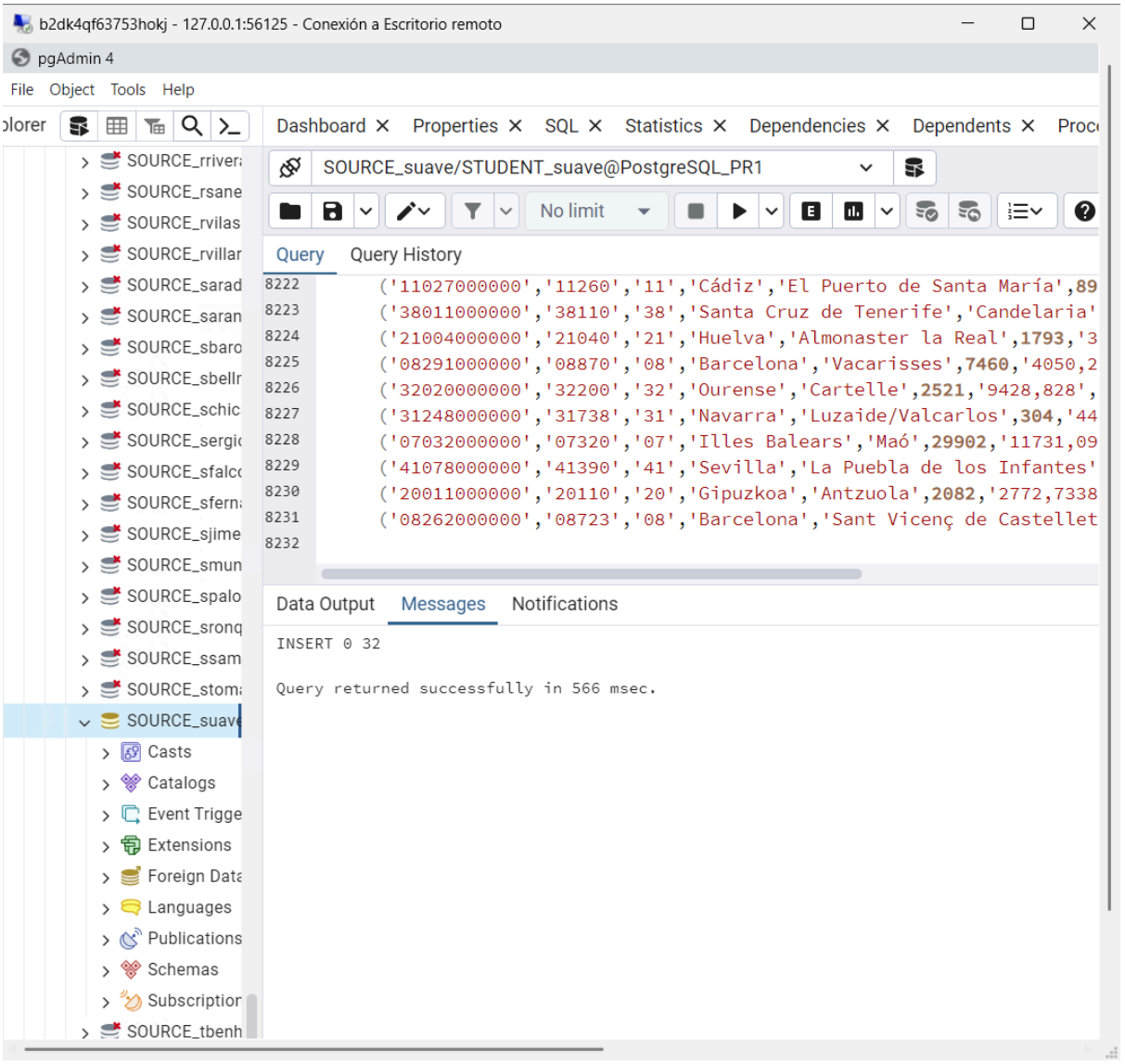

Creación de esquema dbo: se ejecutó CREATE SCHEMA IF NOT EXISTS dbo; y el entorno confirmó la ejecución correcta. (E1_dbo_schema_ok.png)

# Evidencia 2



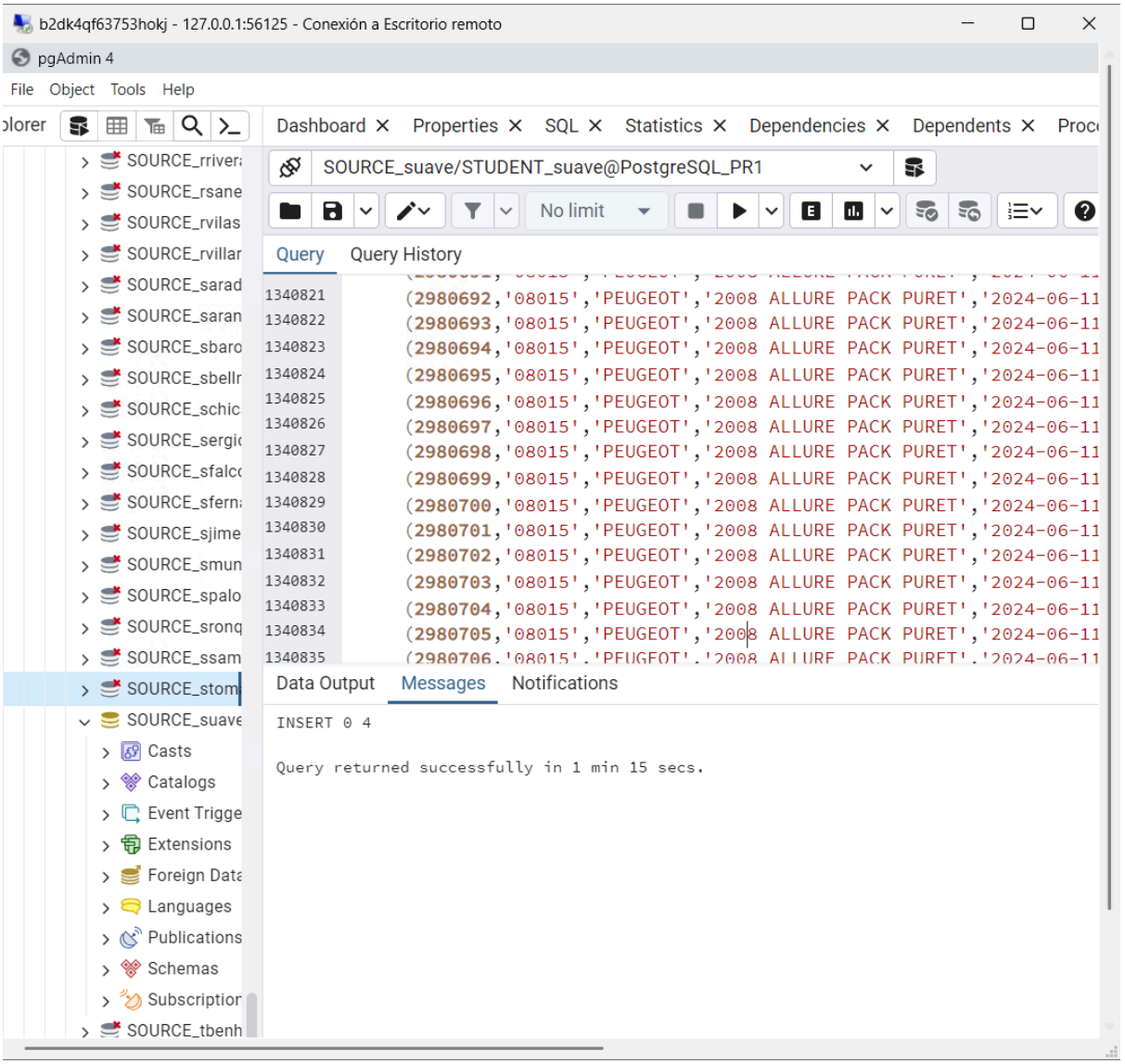

dgt_type.sql: creación y carga correctas de dbo.dgt_type. (E2_dgt_type_success.png)

# Evidencia 3



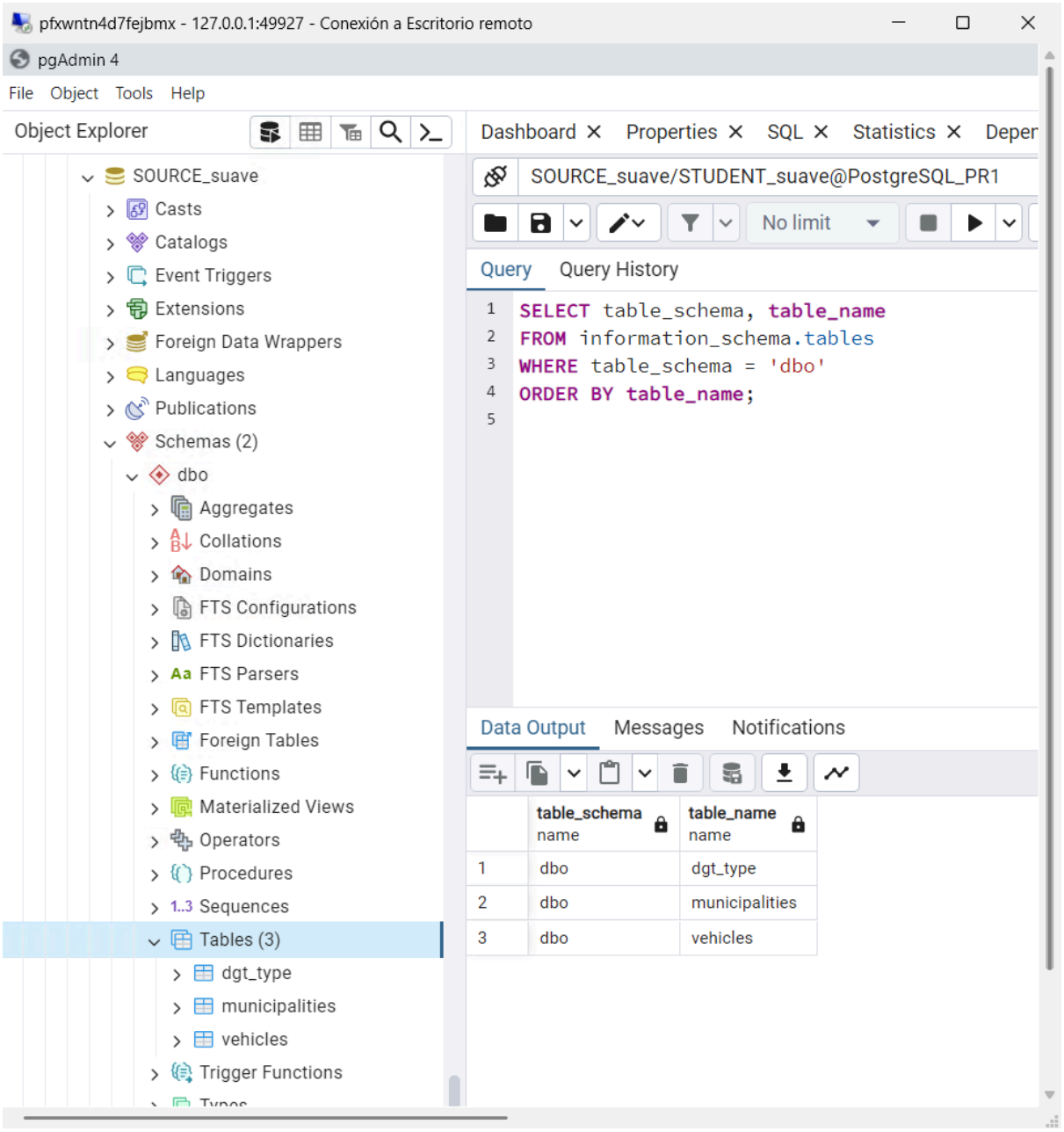

municipalities.sql: creación y carga correctas de dbo.municipalities.
(E3_municipalities_success.png)

# Evidencia 4



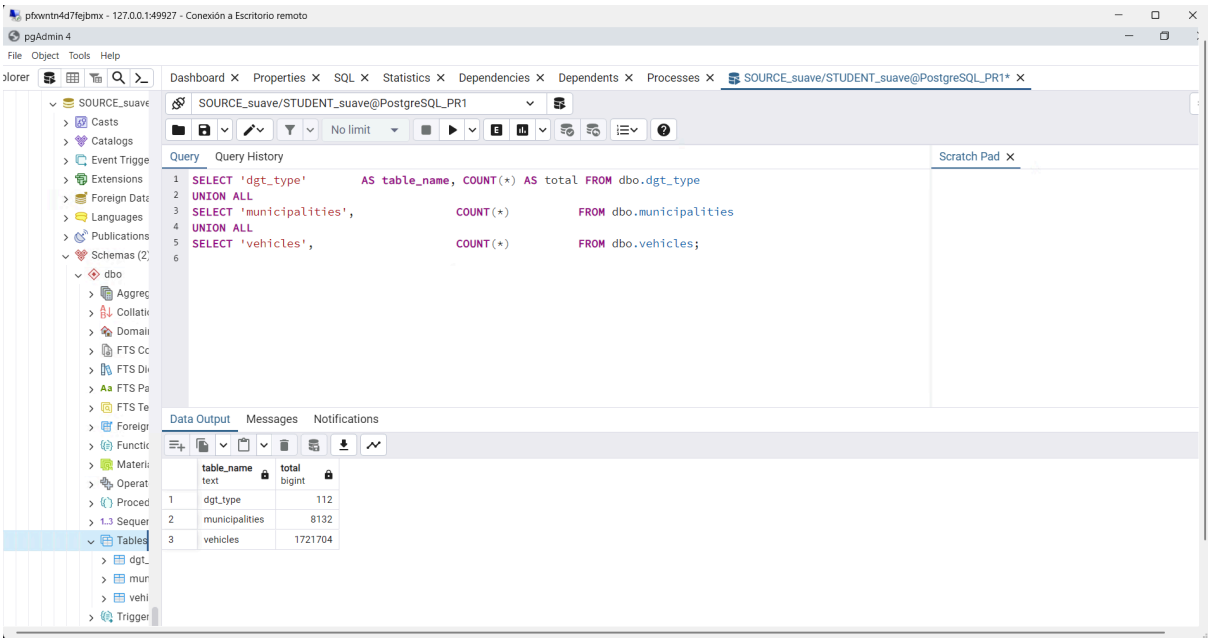

vehicles.sql: creación y carga correctas de dbo.vehicles. (E4_vehicles_success.png)

# Evidencia 5




Tablas en dbo: aparecen dgt_type, municipalities, vehicles. (E5_tables_in_dbo.png)

Evidencia 6




dgt_type = 112 subtipos; municipalities = 8 132 registros; vehicles = 1 721 704 registros. La carga coincide con los scripts y confirma que las tablas están pobladas correctamente. (E6_counts.png)

# 3.2 Consultas solicitadas: SQL + explicación

- Q1 — Subtipos DGT (ordenados):
  <<
  SELECT
  d.cod_subtype_dgt,
  d.des_subtype_dgt
  FROM dbo.dgt_type AS d
  ORDER BY d.cod_subtype_dgt, d.des_subtype_dgt;
  >>

  Listado del catálogo DGT (tabla dbo.dgt_type): se seleccionan los códigos de subtipo y sus descripciones y se ordena de forma estable por código y descripción. Sirve como verificación de referencia de que el diccionario está cargado y consistente antes de hacer consultas o modificaciones. Véase Captura Q1 en el apéndice.

- Q2 — Nº municipios “Sevilla”:
  <<
  SELECT COUNT(1) AS total_municipios_sevilla
  FROM dbo.municipalities AS m
  WHERE m.province = 'Sevilla';
  >>

Cómputo de municipios de la provincia de Sevilla (tabla dbo.municipalities). El filtro por province = 'Sevilla' y COUNT(1) devuelve 106, confirmando el volumen esperado de registros para esa provincia. Véase Captura Q2 en el apéndice.

- Q3 — Municipios de “Sevilla” que empiezan por A o U (orden DESC):
  <<
  SELECT
  m.name_muni,
  m.province
  FROM dbo.municipalities AS m
  WHERE m.province = 'Sevilla'
  AND (m.name_muni ILIKE 'A%' OR m.name_muni ILIKE 'U%')
  ORDER BY m.name_muni DESC;
  >>

  Filtro por provincia = 'Sevilla' y por nombre que comience por ‘A’ o ‘U’ (ILIKE para mayúsculas/minúsculas). Se muestran los municipios resultantes ordenados en orden descendente por nombre; aparecen 14 filas, lo que valida la condición de prefijo. Véase Captura Q3 en el apéndice.

- Q4 — INSERT controlado de un nuevo subtipo (Z9) + verificación:
  <<
  BEGIN;

  INSERT INTO dbo.dgt_type (cod_type_dgt, des_type_dgt, cod_subtype_dgt, des_subtype_dgt)
  VALUES ('4', 'TURISMOS', 'Z9', 'TURISMO PR2');

  -- Verificación
  SELECT cod_type_dgt, des_type_dgt, cod_subtype_dgt, des_subtype_dgt
  FROM dbo.dgt_type
  WHERE cod_subtype_dgt = 'Z9';
  >>

  Inserción controlada de un nuevo subtipo DGT con código ‘Z9’ y verificación inmediata: el SELECT posterior muestra exactamente 1 fila con cod_subtype_dgt = 'Z9'. Esta evidencia confirma que la operación INSERT fue correcta. Véase Captura Q4 en el apéndice.

- Q5 — UPDATE del subtipo Z9 + verificación:
  <<
  UPDATE dbo.dgt_type
  SET des_subtype_dgt = 'TURISMO PR2 ACTUALIZADO'
  WHERE cod_subtype_dgt = 'Z9';

  -- Verificación
  SELECT cod_type_dgt, des_type_dgt, cod_subtype_dgt, des_subtype_dgt
  FROM dbo.dgt_type

```
WHERE cod_subtype_dgt = 'Z9';
>>
```

Actualización del texto del subtipo recién insertado (‘Z9’) y verificación: tras el UPDATE, el SELECT refleja la descripción modificada (“TURISMO PR2 ACTUALIZADO”), evidenciando que el cambio persiste en la tabla. Véase Captura Q5 en el apéndice.

- Q6 — DELETE del subtipo Z9 + comprobación + ROLLBACK:

```
<<
DELETE FROM dbo.dgt_type
WHERE cod_subtype_dgt = 'Z9';

-- Verificación: no debe devolver filas
SELECT 1
FROM dbo.dgt_type
WHERE cod_subtype_dgt = 'Z9';
>>
```

Eliminación del subtipo de prueba ‘Z9’ y verificación de ausencia de filas (el SELECT ya no devuelve resultados). La transacción iniciada en Q4 (BEGIN) se mantiene abierta hasta Q6 y se cierra con ROLLBACK, dejando la BD como antes de Q4–Q6. Véase Captura Q6 en el apéndice.

- Q7 — Turismos con altitud del municipio > 950 (join por marca/modelo):

```
<<
SELECT
   v.brand,
   v.model,
   m.name_muni      AS municipality,
   m.altitude,
   d.des_type_dgt   AS dgt_type
FROM dbo.vehicles       AS v
JOIN dbo.municipalities AS m ON v.municipality = m.cod_geo
JOIN dbo.dgt_type       AS d ON v.subtype      = d.cod_subtype_dgt
WHERE d.des_type_dgt = 'TURISMOS'
  AND m.altitude > 950
ORDER BY v.brand ASC, v.model ASC;
>>
```

Join entre vehicles, municipalities y dgt_type para listar turismos (des_type_dgt = 'TURISMOS') con altitud del municipio > 950 m. Se muestran marca y modelo, municipio y tipo DGT; el orden por marca y modelo facilita detectar duplicidades/consistencias. Véase Captura Q7 en el apéndice.

- Q8 — Nº de vehículos por municipio (conteo simple):

```
<<
SELECT
```

```
    m.name_muni AS municipality,
    COUNT(*)    AS total_vehicles
FROM dbo.vehicles       AS v
JOIN dbo.municipalities AS m ON v.municipality = m.cod_geo
GROUP BY m.name_muni
ORDER BY m.name_muni ASC;
>>
```

Agregación por municipio: COUNT(*) de vehículos tras unir vehicles con municipalities por el código geográfico. El resultado devuelve un recuento por municipio y se ordena alfabéticamente, permitiendo comparar tamaños relativos. Véase Captura Q8 en el apéndice.

- Q9 — Municipios con < 15 000 vehículos (HAVING):

```
<<
SELECT
    m.name_muni AS municipality,
    COUNT(*)    AS total_vehicles
FROM dbo.vehicles       AS v
JOIN dbo.municipalities AS m ON v.municipality = m.cod_geo
GROUP BY m.name_muni
HAVING COUNT(*) < 15000
ORDER BY m.name_muni ASC;
>>
```

Misma agregación que Q8 pero con cláusula HAVING COUNT(*) < 15000 para filtrar municipios con menos de 15 000 vehículos. El resultado devuelve 2 municipios, mostrando el uso correcto de HAVING sobre la agregación. Véase Captura Q9 en el apéndice.

- Q10 — Nº de vehículos por tipo DGT × nuevo/usado (tabla cruzada básica):

```
<<
SELECT
    d.des_type_dgt AS dgt_type,
    v.new_used     AS new_or_used,  -- 'N' nuevo, 'U' usado
    COUNT(*)       AS total_vehicles
FROM dbo.vehicles AS v
JOIN dbo.dgt_type AS d ON v.subtype = d.cod_subtype_dgt
GROUP BY d.des_type_dgt, v.new_used
ORDER BY d.des_type_dgt ASC, v.new_used ASC;
>>
```

Cruce de vehicles con el catálogo DGT por subtipo para obtener el tipo DGT (des_type_dgt) y distribución por estado del vehículo (‘N’ nuevo / ‘U’ usado). El GROUP BY (tipo, new_or_used) y el ORDER BY muestran un resumen claro de totales por categoría y condición. Véase Captura Q10 en el apéndice.

## 3.3 Corrección de sentencias SQL

A continuación se muestran tres ejemplos típicos de errores y su versión corregida, indicando por qué la segunda cumple los principios de SQL y del modelo.

### Caso A — JOIN implícito con coma

Errónea:

```
<<
SELECT m.name_muni, v.brand
FROM dbo.municipalities m, dbo.vehicles v
WHERE v.municipality = m.cod_geo
  AND m.province = 'Sevilla';
>>
```

Corregida (JOIN explícito):

```
<<
SELECT m.name_muni, v.brand
FROM dbo.vehicles AS v
JOIN dbo.municipalities AS m
  ON v.municipality = m.cod_geo
WHERE m.province = 'Sevilla';
>>
```

Por qué ahora cumple:
Se emplea `JOIN ... ON` con la condición de unión separada del filtro. Mejora legibilidad, evita cruces accidentales y respeta la semántica de la relación FK `vehicles.municipality → municipalities.cod_geo`.

### Caso B — Precedencia AND/OR sin paréntesis

Errónea:

```
<<
SELECT d.cod_subtype_dgt, d.des_subtype_dgt
FROM dbo.dgt_type AS d
WHERE d.des_type_dgt = 'TURISMOS' OR d.des_subtype_dgt ILIKE 'A%'
  AND d.cod_subtype_dgt <> 'Z9';
>>
```

Corregida (paréntesis claros):

```
<<
SELECT d.cod_subtype_dgt, d.des_subtype_dgt
FROM dbo.dgt_type AS d
WHERE (d.des_type_dgt = 'TURISMOS' OR d.des_subtype_dgt ILIKE 'A%')
```

```
  AND d.cod_subtype_dgt <> 'Z9';
>>
```

Por qué ahora cumple:
La precedencia en SQL evalúa `AND` antes que `OR`. Los paréntesis fuerzan el criterio pretendido: primero “TURISMOS o subtipo empieza por A”, y después excluir `Z9`.

### Caso C — LEFT JOIN “anulado” por condición en WHERE

Errónea:
```
<<
SELECT m.name_muni, COUNT(v.id) AS total
FROM dbo.municipalities AS m
LEFT JOIN dbo.vehicles AS v
  ON v.municipality = m.cod_geo
WHERE v.new_used = 'N'
GROUP BY m.name_muni;
>>
```

Corregida (condición en la cláusula ON):
```
<<
SELECT m.name_muni, COUNT(v.id) AS total
FROM dbo.municipalities AS m
LEFT JOIN dbo.vehicles AS v
  ON v.municipality = m.cod_geo
 AND v.new_used = 'N'
GROUP BY m.name_muni;
>>
```

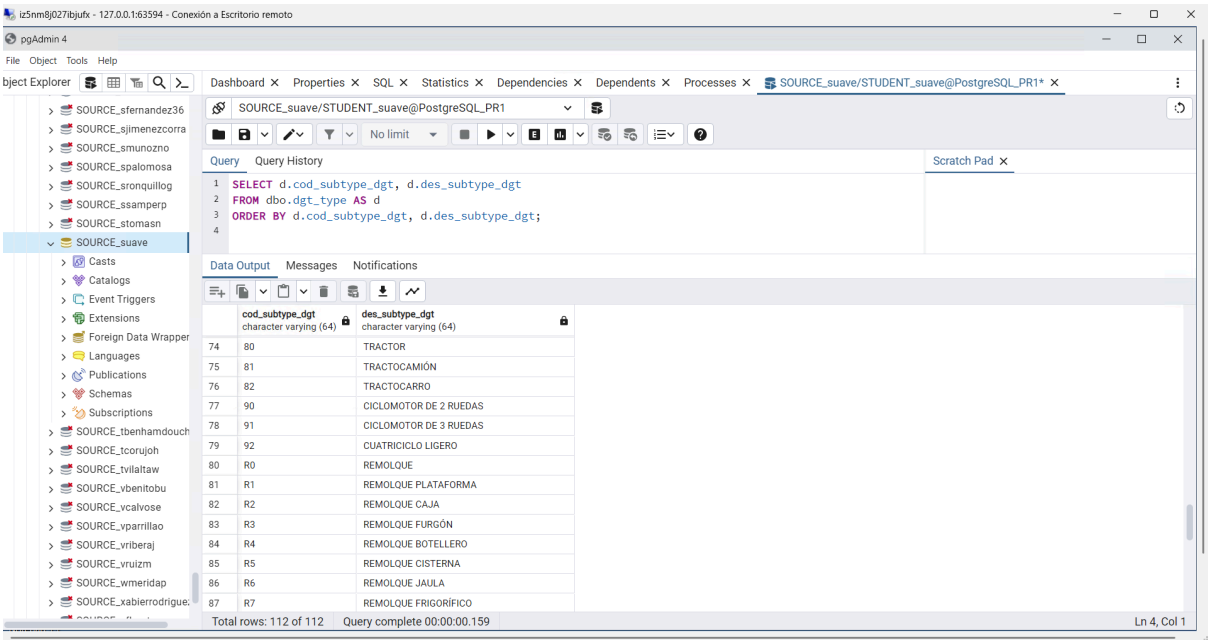
Por qué ahora cumple:
La condición sobre `v.new_used` se aplica en el `ON`, manteniendo los municipios sin vehículos nuevos (aparecen con `COUNT=0`). En la versión errónea, el `WHERE v.new_used='N'` elimina las filas nulas del `LEFT`, comportándose como `INNER`.

# Conclusión breve

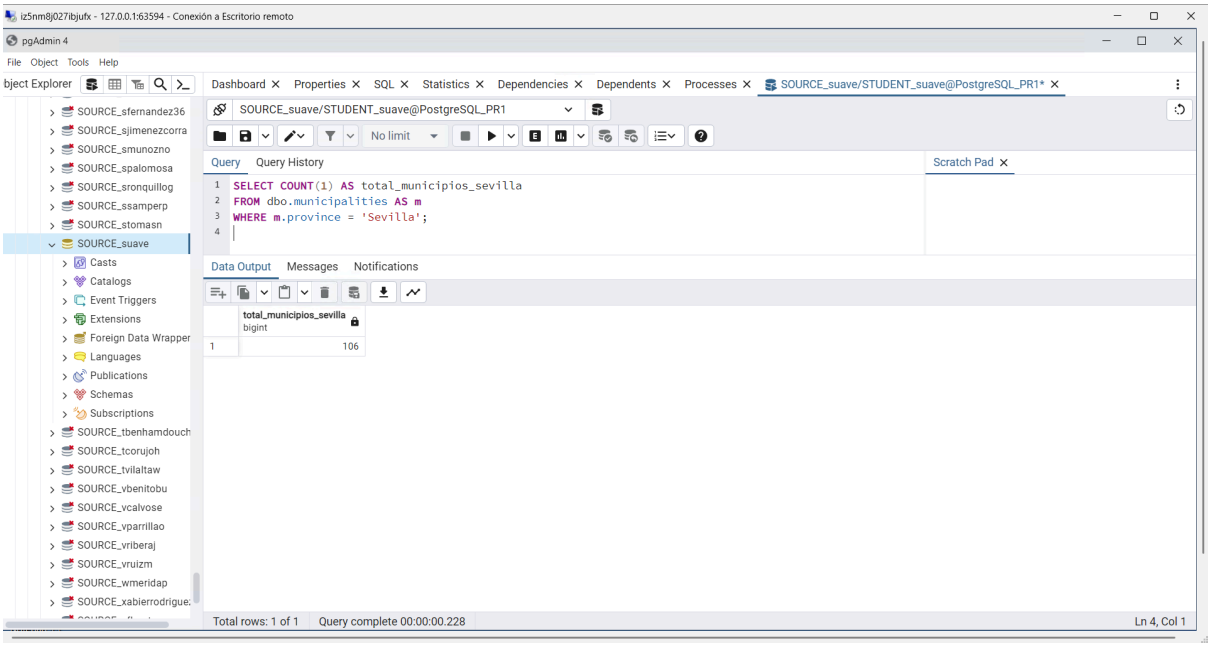
Con la práctica se recorre el ciclo completo de diseño relacional. En el Ejercicio 1, he identificado y corregido errores del modelo conceptual (identificación unívoca, cardinalidades, dominios) y he actualizado el E/R para cubrir multipropiedad (OWNER–OWNERSHIP–VEHICLE) y sanciones con conductor opcional (FINE y DRIVER). En el Ejercicio 2, se ha traducido al modelo lógico con PK/AK/FK coherentes —incluida la PK compuesta de DGT_TYPE (cod_type_dgt, cod_subtype_dgt) y la FK (type, subtype) en VEHICLE. En el Ejercicio 3, he verificado el modelo físico en PostgreSQL: creación del esquema, ejecución de scripts y validaciones por conteo (dgt_type = 112, municipalities = 8

132, vehicles = 1 721 704). Con las consultas 1–10 se demuestra selección, filtrado, ordenación, JOIN, agregación y HAVING, y corrigiendo las sentencias típicas (JOIN implícito, precedencia AND/OR, LEFT JOIN anulado) se justifica la semántica correcta. El resultado es un diseño coherente y reproducible, con integridad y correcta interpretación de los datos.

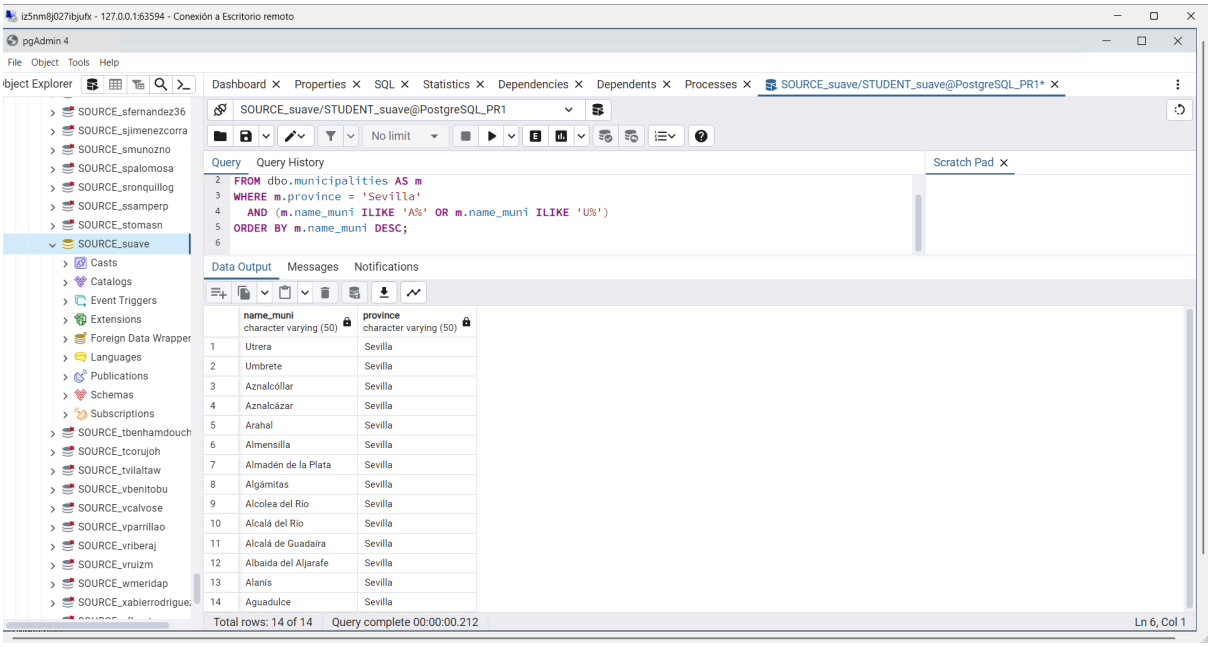
# Apéndice



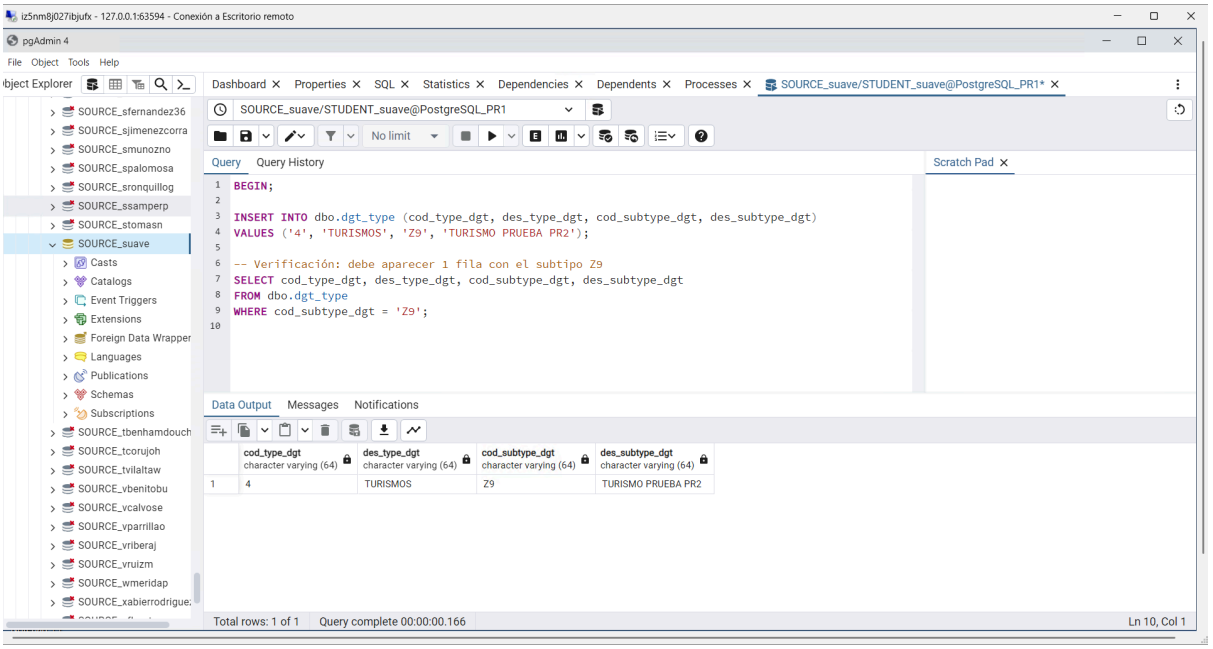

Captura Q1 — E7_Q1_dgt_subtypes.png



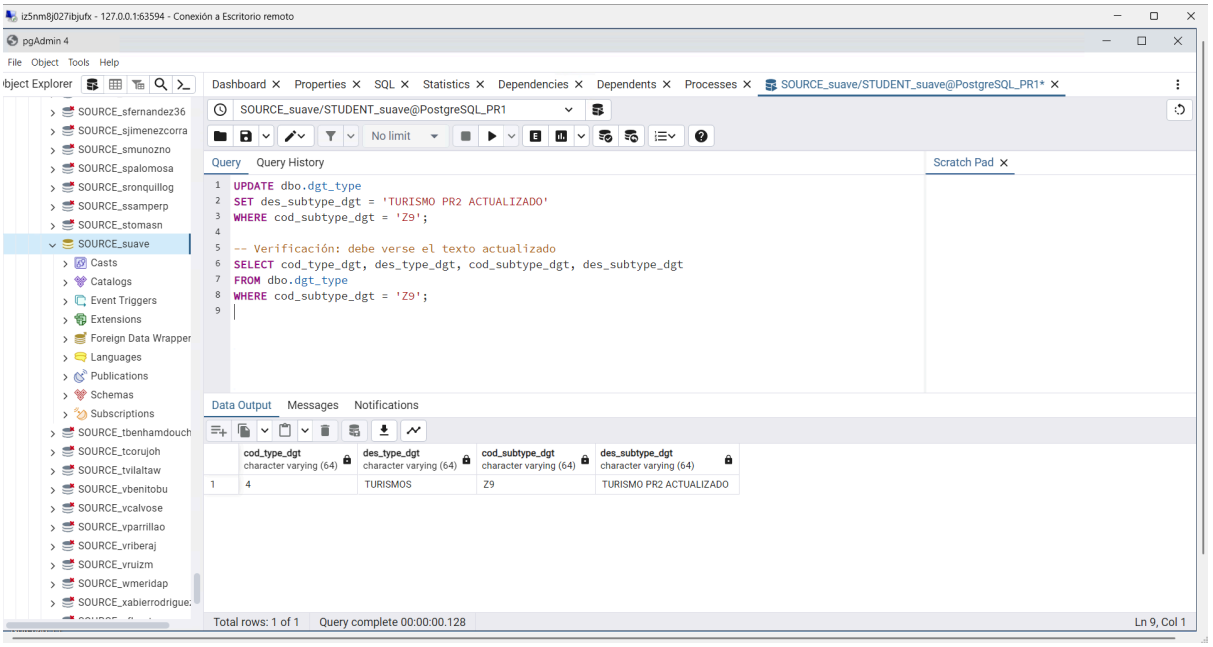

Captura Q2 — E8_Q2_count_sevilla.png



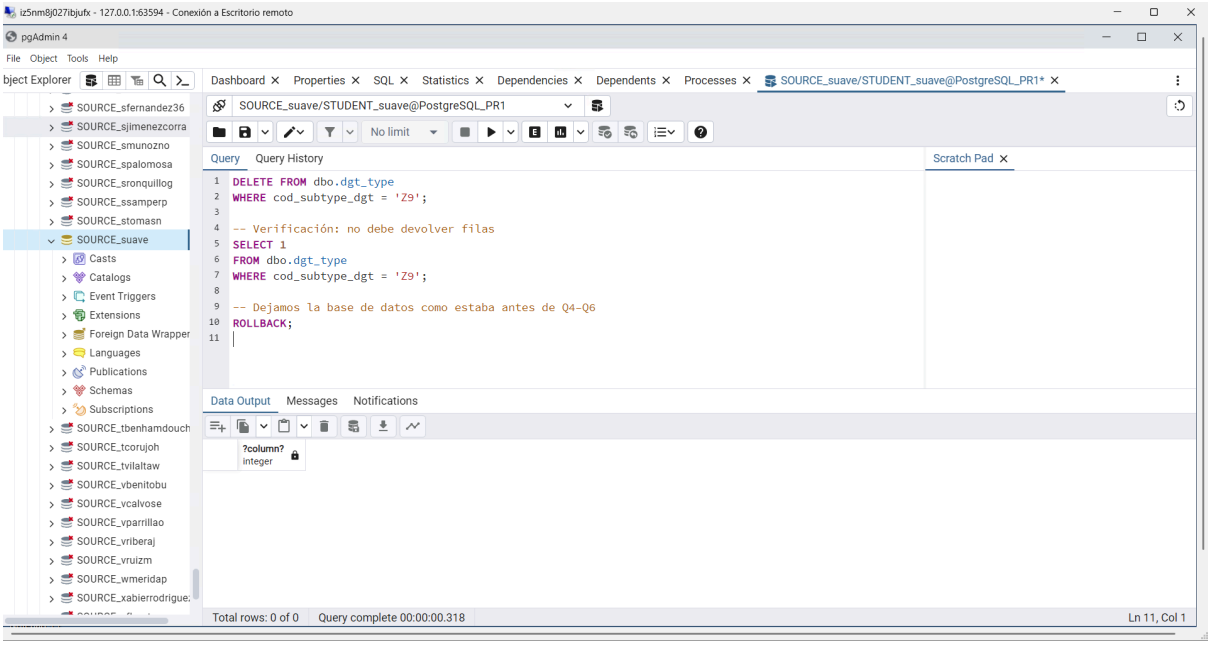

Captura Q3 — E9_Q3_sevilla_prefijos.png



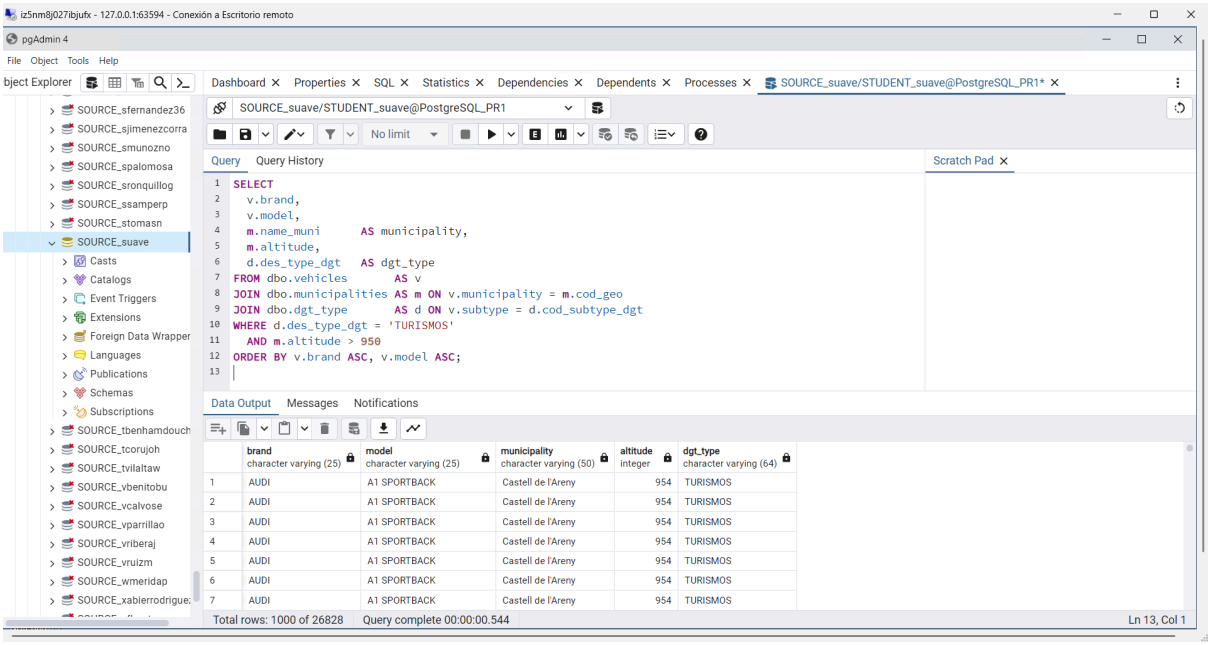

Captura Q4 — E10_Q4_insert_verify.png



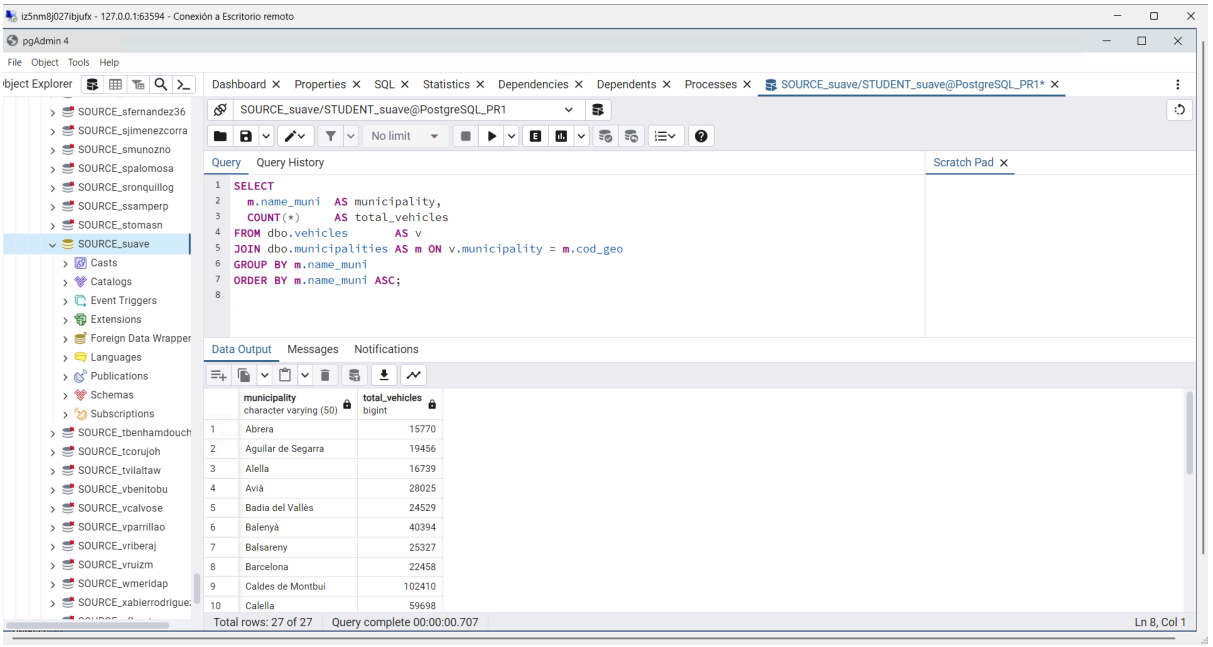

Captura Q5 — E11_Q5_update_verify.png



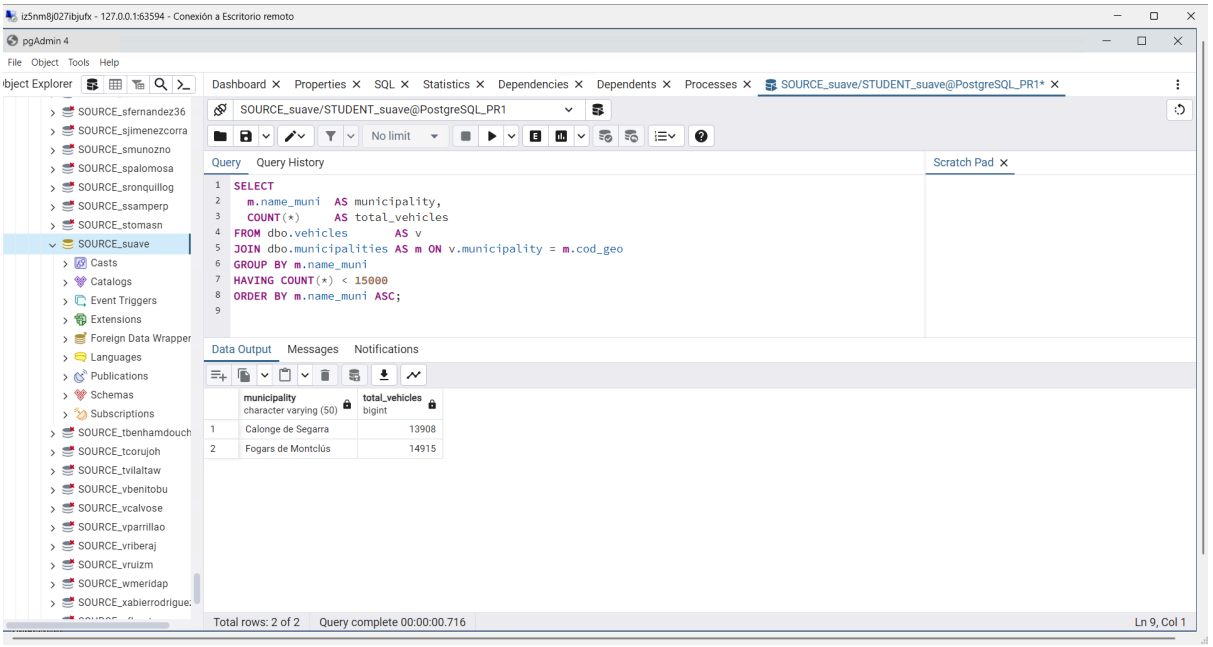

Captura Q6 — E12_Q6_delete_verify.png



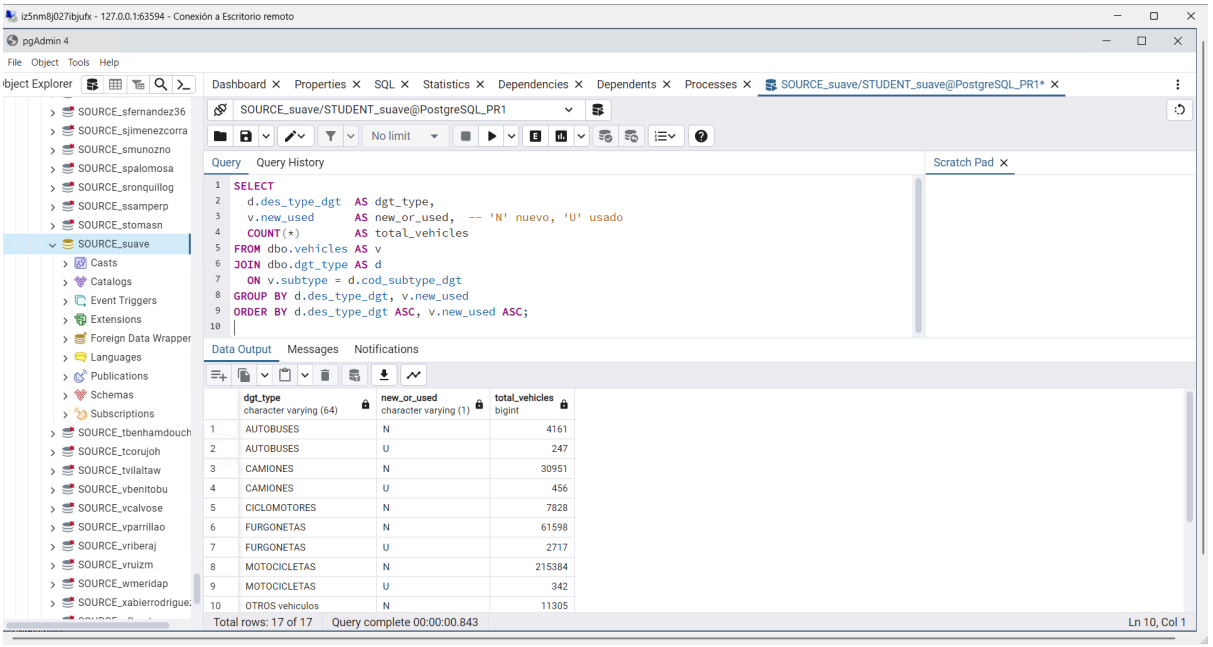

Captura Q7 — E13_Q7_turismos_alt950.png




Captura Q8 — E14_Q8_count_por_municipio.png




Captura Q9 — E15_Q9_municipios_lt_15000.png




Captura Q10 — E16_Q10_tipo_y_nuevo_usado.png