```
-- ============================================================
-- HOUSEHOLD LINKAGE USING POSTGIS ADDRESS STANDARDIZER
-- For CU Medical Dataset (ROW_ID VERSION)
-- Each row gets its own household_id based on its address
-- ============================================================

-- Step 1: Ensure extensions are installed
CREATE EXTENSION IF NOT EXISTS postgis;
CREATE EXTENSION IF NOT EXISTS address_standardizer;
CREATE EXTENSION IF NOT EXISTS address_standardizer_data_us;

-- ============================================================
-- Step 2: Create table and import CSV
-- ============================================================

DROP TABLE IF EXISTS cu_data;

CREATE TABLE cu_data (
   arb_person_id BIGINT,
   mrn TEXT,
   start_date TEXT,
   first_name TEXT,
   last_name TEXT,
   birth_date TEXT,
   race TEXT,
   ethnicity TEXT,
   sex TEXT,
   marital_status TEXT,
   address TEXT,
   city TEXT,
   state TEXT,
   zip_code TEXT,
   cell_phone_number TEXT,
   home_phone_number TEXT,
   email TEXT,
   ssn TEXT
);

-- Import CSV
COPY cu_data FROM '/path/to/your/cu_data.csv'
WITH (FORMAT CSV, HEADER TRUE, DELIMITER ',', NULL '');

-- Verify
SELECT COUNT(*) AS total_rows FROM cu_data;
```

```
-- ============================================================
-- Step 3: Add unique row_id to cu_data
-- ============================================================

ALTER TABLE cu_data ADD COLUMN row_id SERIAL;

-- Verify row_id was added
SELECT row_id, arb_person_id, address FROM cu_data LIMIT 10;

-- ============================================================
-- Step 4: Create normalized address table using row_id
-- ============================================================

DROP TABLE IF EXISTS cu_addresses_normalized;

CREATE TABLE cu_addresses_normalized AS
SELECT
   row_id,
   arb_person_id,
   address AS original_address,
   city AS original_city,
   state AS original_state,
   zip_code AS original_zip,
   CONCAT_WS(' ',
      NULLIF(TRIM(address), ''),
      NULLIF(TRIM(city), ''),
      NULLIF(TRIM(state), ''),
      NULLIF(TRIM(zip_code), '')
   ) AS full_address
FROM cu_data
WHERE address IS NOT NULL AND TRIM(address) != '';

-- ============================================================
-- Step 5: Add columns for standardized components
-- ============================================================

ALTER TABLE cu_addresses_normalized
ADD COLUMN std_house TEXT,
ADD COLUMN std_predir TEXT,
ADD COLUMN std_name TEXT,
ADD COLUMN std_suftype TEXT,
ADD COLUMN std_sufdir TEXT,
ADD COLUMN std_unit TEXT,
```

```
ADD COLUMN std_city TEXT,
ADD COLUMN std_state TEXT,
ADD COLUMN std_postcode TEXT,
ADD COLUMN normalized_key TEXT;

-- ============================================================
-- Step 6: Clean addresses (remove problematic characters)
-- ============================================================

UPDATE cu_addresses_normalized
SET full_address = REGEXP_REPLACE(
   REGEXP_REPLACE(
      REGEXP_REPLACE(
         REGEXP_REPLACE(full_address, '[&*#@!]', '', 'g'),
         '\s+', ' ', 'g'),
      '^\s+|\s+$', '', 'g'),
   ',', '', 'g')
WHERE full_address IS NOT NULL;

-- ============================================================
-- Step 7: Delete rows that still look like intersections
-- ============================================================

DELETE FROM cu_addresses_normalized
WHERE full_address ILIKE '% and %'
  OR full_address ILIKE '% at %'
  OR full_address ILIKE '%/%';

-- ============================================================
-- Step 8: Call standardize_address ONCE per row
-- ============================================================

ALTER TABLE cu_addresses_normalized ADD COLUMN std_result stdaddr;

UPDATE cu_addresses_normalized
SET std_result = standardize_address('us_lex', 'us_gaz', 'us_rules', full_address);

-- ============================================================
-- Step 9: Extract components from stored result
-- ============================================================

UPDATE cu_addresses_normalized
SET
   std_house = (std_result).house_num,
```

```
   std_predir = (std_result).predir,
   std_name = (std_result).name,
   std_suftype = (std_result).suftype,
   std_sufdir = (std_result).sufdir,
   std_unit = (std_result).unit,
   std_city = (std_result).city,
   std_state = (std_result).state,
   std_postcode = (std_result).postcode;

-- ============================================================
-- Step 10: Drop the temporary column
-- ============================================================

ALTER TABLE cu_addresses_normalized DROP COLUMN std_result;

-- ============================================================
-- Step 11: Create normalized key for matching
-- ============================================================

UPDATE cu_addresses_normalized
SET normalized_key = UPPER(CONCAT_WS('|',
   COALESCE(TRIM(std_house), ''),
   COALESCE(TRIM(std_predir), ''),
   COALESCE(TRIM(std_name), ''),
   COALESCE(TRIM(std_suftype), ''),
   COALESCE(TRIM(std_sufdir), ''),
   COALESCE(TRIM(std_unit), ''),
   COALESCE(TRIM(std_city), ''),
   COALESCE(TRIM(std_state), ''),
   COALESCE(LEFT(TRIM(std_postcode), 5), '')
));

CREATE INDEX idx_normalized_key ON cu_addresses_normalized(normalized_key);
CREATE INDEX idx_row_id ON cu_addresses_normalized(row_id);

-- ============================================================
-- Step 12: Verify standardization results
-- ============================================================

SELECT
   row_id,
   arb_person_id,
   original_address,
   std_house,
```

```
    std_name,
    std_suftype,
    std_city,
    normalized_key
FROM cu_addresses_normalized
LIMIT 20;

-- ============================================================
-- Step 13: Assign Household IDs
-- ============================================================

DROP TABLE IF EXISTS cu_households;

CREATE TABLE cu_households AS
WITH ranked_keys AS (
    SELECT
        normalized_key,
        DENSE_RANK() OVER (ORDER BY normalized_key) AS household_id
    FROM cu_addresses_normalized
    WHERE normalized_key IS NOT NULL
      AND normalized_key != '||||||||'
    GROUP BY normalized_key
)
SELECT
    a.row_id,
    a.arb_person_id,
    a.original_address,
    a.original_city,
    a.original_state,
    a.original_zip,
    a.normalized_key,
    r.household_id
FROM cu_addresses_normalized a
JOIN ranked_keys r ON a.normalized_key = r.normalized_key;

CREATE INDEX idx_household_id ON cu_households(household_id);
CREATE INDEX idx_households_row_id ON cu_households(row_id);

-- ============================================================
-- Step 14: Final output with household assignments
-- Join using row_id (unique) instead of arb_person_id
-- ============================================================

DROP TABLE IF EXISTS cu_data_with_households;
```

```
CREATE TABLE cu_data_with_households AS
SELECT
   c.*,
   h.household_id,
   h.normalized_key AS normalized_address
FROM cu_data c
LEFT JOIN cu_households h ON c.row_id = h.row_id;

-- ============================================================
-- Step 15: Verify results
-- ============================================================

-- Should match original cu_data count
SELECT COUNT(*) AS final_count FROM cu_data_with_households;
SELECT COUNT(*) AS original_count FROM cu_data;

-- Total households
SELECT COUNT(DISTINCT household_id) AS total_households
FROM cu_data_with_households;

-- Check person with multiple addresses gets multiple household_ids
SELECT arb_person_id, COUNT(DISTINCT household_id) AS num_households
FROM cu_data_with_households
WHERE household_id IS NOT NULL
GROUP BY arb_person_id
HAVING COUNT(DISTINCT household_id) > 1
LIMIT 10;

-- See example of same person with different households
SELECT
   arb_person_id,
   first_name,
   last_name,
   address,
   city,
   household_id
FROM cu_data_with_households
WHERE arb_person_id IN (
   SELECT arb_person_id
   FROM cu_data_with_households
   WHERE household_id IS NOT NULL
   GROUP BY arb_person_id
   HAVING COUNT(DISTINCT household_id) > 1
```

```
   LIMIT 3
)
ORDER BY arb_person_id, household_id;

-- Household size distribution
SELECT
   household_size,
   COUNT(*) AS num_households
FROM (
   SELECT household_id, COUNT(*) AS household_size
   FROM cu_data_with_households
   WHERE household_id IS NOT NULL
   GROUP BY household_id
) sizes
GROUP BY household_size
ORDER BY household_size;

-- Sample multi-person households
SELECT
   household_id,
   arb_person_id,
   first_name,
   last_name,
   address,
   city
FROM cu_data_with_households
WHERE household_id IN (
   SELECT household_id
   FROM cu_data_with_households
   WHERE household_id IS NOT NULL
   GROUP BY household_id
   HAVING COUNT(*) > 1
   LIMIT 5
)
ORDER BY household_id, last_name;

-- Records without household assignment
SELECT COUNT(*) AS records_without_household
FROM cu_data_with_households
WHERE household_id IS NULL;
```