**Step- 1 : Creating Tables**

DROP TABLE IF EXISTS outreach;

DROP TABLE IF EXISTS campaigns;

DROP TABLE IF EXISTS applicants;

CREATE TABLE applicants (

app\_id TEXT PRIMARY KEY,

country TEXT,

university TEXT,

phone\_number TEXT

);

CREATE TABLE campaigns (

campaign\_id TEXT PRIMARY KEY,

name TEXT,

category TEXT,

intake TEXT,

university TEXT,

status TEXT,

start\_date TEXT

);

CREATE TABLE outreach (

reference\_id TEXT,

recieved\_at TEXT,

university TEXT,

caller\_name TEXT,

outcome\_1 TEXT,

remark TEXT,

campaign\_id TEXT

);

**Step-2 : Import Cleans CSVs**

Do this for each table **in this order**: applicants → campaigns → outreach.

1. In pgAdmin → left sidebar → expand **Schemas → public → Tables**.
2. Right-click on **applicants** → **Import/Export Data…**
   * Format: **CSV**
   * File: choose applicants\_cleaned\_final.csv (downloaded from Colab)
   * Encoding: **UTF8**
   * Delimiter: ,
   * Header: ✅ check **Yes**
   * Click **OK**
3. Repeat for campaigns (choose campaigns\_cleaned\_final.csv)
4. Repeat for outreach (choose outreach\_cleaned\_final.csv)

**Step 4 — Verify the Import**

Run this SQL in pgAdmin Query Tool:

SELECT 'applicants' AS table\_name, COUNT(\*) FROM applicants

UNION ALL

SELECT 'campaigns', COUNT(\*) FROM campaigns

UNION ALL

SELECT 'outreach', COUNT(\*) FROM outreach;

=> The counts should match the number of rows in your final CSVs.

**Step-5: Checking if IDs match  
  
First, check if outreach IDs all match applicant IDs:**

SELECT COUNT(\*) AS orphan\_outreach

FROM outreach o

LEFT JOIN applicants a ON o.reference\_id = a.app\_id

WHERE a.app\_id IS NULL;

* **If orphan\_outreach = 0 → all reference IDs are valid.**
* **If >0 → some outreach rows don’t match an applicant.**

After running the above query, if everything was done correctly then it will display orphan\_outreach = 104 in the query data output section.

Now, We need to separate these orphan ids means ids which are not matching in the applicant and outreach dataset.

**Step 6: Move Orphans into a Separate Table**

**This keeps your main outreach table clean but saves the “bad” rows.**

-- 1. Create a new table to hold the orphans

CREATE TABLE outreach\_orphans AS

SELECT \*

FROM outreach

WHERE reference\_id NOT IN (SELECT app\_id FROM applicants);

**✅ Now you have a copy of those 104 rows in outreach\_orphans.**

**Step 7: Clean Outreach Table**

**Remove those orphan rows from the main outreach table:**

-- 2. Delete the orphan rows from outreach

DELETE FROM outreach

WHERE reference\_id NOT IN (SELECT app\_id FROM applicants);

**Now outreach only has rows that can successfully join with applicants**.

**Step 8: Add Foreign Key Constraint**

**This enforces strict relational integrity (so no new orphans sneak in later).**

-- 3. Add foreign key from outreach → applicants

ALTER TABLE outreach

ADD CONSTRAINT fk\_outreach\_applicant

FOREIGN KEY (reference\_id) REFERENCES applicants(app\_id);

**✅ From now on, every reference\_id in outreach must exist in applicants.**

**Step 9: Verify Clean State**

**(Run each one from below separately)Quick checks:**

-- Should be 0 now

SELECT COUNT(\*) AS orphan\_outreach

FROM outreach

WHERE reference\_id NOT IN (SELECT app\_id FROM applicants);

-- See how many rows are in outreach after cleanup

SELECT COUNT(\*) FROM outreach;

-- See the 104 saved orphan rows

SELECT COUNT(\*) FROM outreach\_orphans;

**Quick recap — what you *did* and *why***

* You created a database data\_visualization\_project.
* You created three tables: applicants, campaigns, outreach.
* You imported the **cleaned CSVs** (the \*\_final.csv files) into those tables.
* You ran a check and found **104** outreach.reference\_id values that did not match any applicants.app\_id (these are the *orphans*).
* Instead of forcing them into the main table or dropping them, you **moved those 104 rows into a separate table** called outreach\_orphans (so nothing was lost).
* You **deleted** those 104 rows from the main outreach table so outreach now only contains rows that have matching applicants.
* You then **added a foreign key constraint** on outreach.reference\_id → applicants.app\_id, so future data has to be consistent.

**Why this is good:** your main dataset is now relationally clean — joins will be reliable, queries will be simpler, and visualizations won’t be polluted by invalid references. The problematic rows were kept in outreach\_orphans for audit/review, so you can always investigate later.

**Where you can *see* everything (pgAdmin UI and SQL)**

**In pgAdmin (GUI)**

* **Database**: left-side tree → Servers → <Your Server> → Databases → data\_visualization\_project.
* **Tables**: expand Schemas → public → Tables. You should see:
  + applicants
  + campaigns
  + outreach
  + outreach\_orphans
* **View table rows**:
  + Right-click a table → **View/Edit Data → All Rows** (or **First 100 Rows**).
  + This opens a grid showing actual table data.
* **See table structure / constraints**:
  + Expand one table → **Constraints → Foreign Keys** to see the FK(s).
  + Expand **Columns** to see column names and types.

**What each table now contains (conceptual)**

* applicants → rows with valid app\_id and applicant info (country, phone). This is your **master applicant table**.
* campaigns → campaign metadata (campaign\_id, name, start\_date, etc.).
* outreach → only outreach rows that have a matching app\_id in applicants. This is the **clean child table**.
* outreach\_orphans → the 104 outreach rows that did **not** match an applicant originally. They are preserved for audit/fixing.

**Step-10: Create a master view for analysis**

**Once outreach and applicants (and campaigns) are clean, create a view that analysts / visualization tools can use directly:**

CREATE OR REPLACE VIEW master\_dataset AS

SELECT

a.app\_id,

a.country,

a.university AS applicant\_university,

a.phone\_number,

o.recieved\_at,

o.caller\_name,

o.outcome\_1,

o.remark,

c.campaign\_id,

c.name AS campaign\_name,

c.category AS campaign\_category,

c.start\_date

FROM outreach o

JOIN applicants a ON o.reference\_id = a.app\_id

LEFT JOIN campaigns c ON o.campaign\_id = c.campaign\_id;

**View the first 50 rows of master\_dataset:**

SELECT \* FROM master\_dataset LIMIT 50;

**NOTE:** The master\_dataset is not a physical table but a view. In pgAdmin, you'll find it under the Views section in the Object Explorer on the left.

Finding the View in pgAdmin

1. In the pgAdmin Object Explorer (the left sidebar), expand the database you're working in.
2. Expand the Schemas folder.
3. Expand the public schema (or the schema where you created the view).
4. Expand the Views folder.
5. You should see master\_dataset listed there.
6. Right click on it then view/edit data.