**Title: Common Issues and Solutions in Synthetic Data Generation Across Multiple Tables**

**Overview**

Creating fake data (synthetic data) is often used when we don't have access to real data or want to test a system safely. But generating this kind of data across many connected tables (like customers, orders, support tickets, etc.) can create a lot of problems. These problems are not always easy to see, especially when data looks fine on the surface.

This document explains all the common problems that can happen while creating synthetic data. It also provides simple solutions to each of these problems. The language used here is very easy to help anyone understand the concepts, even if they are not experts in data or programming.

**1. Keys Don't Match Between Tables**

* **Problem**: If you generate user IDs randomly for two tables (like customers and orders), they may not match. That means an order may belong to a customer that doesn't exist.
* **Solution**: First, generate the list of customers and store their IDs. Then use these IDs when creating orders.

**2. Random Data Changes in Different Places**

* **Problem**: A user's name or email might be different in two tables because they were generated separately.
* **Solution**: Save the fake user data in one place and reuse it in every table where it's needed.

**3. Time Errors (Dates Don't Make Sense)**

* **Problem**: An order might show it was placed before the customer signed up.
* **Solution**: Always use the customer's signup date as a starting point when generating other dates.

**4. Lifecycle Errors (Wrong Order of Events)**

* **Problem**: A customer who has already left (churned) is shown making a new purchase.
* **Solution**: Track when a customer joins, becomes active, and leaves. Only allow actions during the valid period.

**5. Too Much Repeated Information**

* **Problem**: Country names or agent names are repeated in many rows, increasing file size and errors.
* **Solution**: Create a separate list (called a lookup table) and use short codes like "IN" for India.

**6. Different Results Every Time**

* **Problem**: Running the code multiple times gives different data.
* **Solution**: Set a seed value using random.seed() and Faker.seed() so results are always the same.

**7. Duplicate Emails or Usernames**

* **Problem**: Two users might get the same email, which should be unique.
* **Solution**: Use unique values or add random numbers to ensure no two users have the same value.

**8. Wrong Data Types**

* **Problem**: A field that should be a number is saved as text.
* **Solution**: Always define what type of data each column should have (text, number, date, etc.).

**9. Flat Distributions (Everything Looks the Same)**

* **Problem**: All countries have the same number of users, which is not realistic.
* **Solution**: Use weighted random choices to make some values more common than others.

**10. No Missing Values**

* **Problem**: Real-world data often has empty or missing fields. Synthetic data might be too clean.
* **Solution**: Randomly leave some values blank (null) based on a percentage.

**11. Uneven Table Sizes**

* **Problem**: There are many customers but only a few orders, which looks odd.
* **Solution**: Decide how many orders each customer should have and generate accordingly.

**12. Hardcoded Values**

* **Problem**: Using fixed values like "90 days" in code makes it hard to change later.
* **Solution**: Keep important numbers as variables at the top of the script.

**13. No Error Handling**

* **Problem**: One bad data value can crash the whole data generation.
* **Solution**: Use try-except to catch errors and skip bad data.

**14. Missing Parent Data**

* **Problem**: Orders exist for customers who were removed later.
* **Solution**: Lock the customer list and don't delete it after orders are created.

**15. Too Much Data for One Value**

* **Problem**: Most data belongs to one country or city.
* **Solution**: Spread the data more evenly by adjusting how often each value is used.

**16. Mixed Language or Address Formats**

* **Problem**: Some names are Indian-style, others are American-style, and so on.
* **Solution**: Use locale settings in Faker (like Faker('en\_IN')) to match regions.

**17. Static vs Changing Values**

* **Problem**: Fields like "account type" should not change for a user, but they do.
* **Solution**: Mark such fields as static and keep them the same across all related data.

**18. Using IDs Before They're Created**

* **Problem**: Orders use product IDs that haven't been generated yet.
* **Solution**: Always create the base tables first (like products) before generating data that depends on them.

**19. Invalid Event Order**

* **Problem**: A user files a support ticket after they left the platform.
* **Solution**: Use valid time ranges based on user activity.

**20. Everything in One Script**

* **Problem**: Having all code in one file makes it hard to reuse or change.
* **Solution**: Split code into parts (one for customers, one for orders, etc.).

**21. Fake Patterns Becoming Too Obvious**

* **Problem**: Feedback score is always high when issue is "billing".
* **Solution**: Add randomness and make sure outcomes vary.

**22. Two Tables Depend on Each Other**

* **Problem**: Table A needs data from B and B needs data from A.
* **Solution**: Break the loop by using placeholder values first, then replace them later.

**23. Can't Reproduce a Specific Row**

* **Problem**: You can’t create the same customer or row again for testing.
* **Solution**: Use fixed logic to generate the same row every time, like using a unique ID based on input.

**24. No Errors or Typos in Data**

* **Problem**: Real data has mistakes; fake data looks too perfect.
* **Solution**: Add some fake typos or incorrect values to simulate real-world errors.

**25. Code Becomes Slow with Large Data**

* **Problem**: Creating too much data takes too long or crashes.
* **Solution**: Use tools that work with large data (like dbldatagen or Spark) and optimize memory usage.

**Conclusion**

Making fake data look real is not just about random values. It’s about making sure that everything fits together, like pieces of a puzzle. IDs must match, dates must make sense, and values must behave like they do in real life. Using the tips and fixes shared in this document, you can build clean, realistic, and useful synthetic datasets for any project.

These points are written to help even beginners avoid mistakes and understand what's going on. If you follow this guide, your synthetic data will be much more accurate and reliable.

Reference - <https://databrickslabs.github.io/dbldatagen/public_docs/APIDOCS.html>