TRANSFORM PROD DATA INTO DEV DATA

3 PRACTICAL STRATEGIES

2023/06/27

DR. REMI CURA, PRINCIPAL DATA SCIENTIST, CENTAUR LABS

REMI.CURA@GMAIL.COM

Today's topics

Benefits of

- Using a DEV database
- … based on PROD data
- … that has been obfuscated

A testimony from the medical tech space

TESTIMONY NEED CONTEXT

AKA, HOW CRAZY AM I?

CONTEXT: WHO AM I

Game: put everything in Postgres

PhD : Computer science / GIS

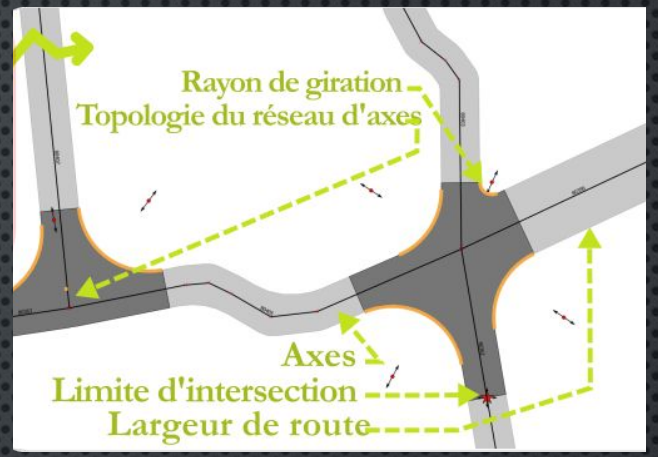
- Vector + topology
- Point clouds (Lidar, stereovision)
- Postgres as a backend? (ML, street modeling)

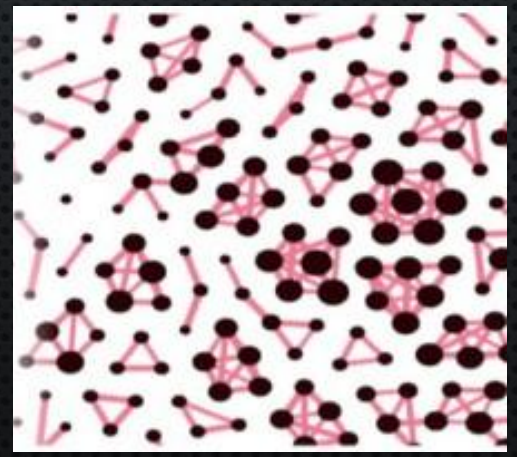
Postdoc: Paris School of Economics

- Historical maps + geocoding in postgres (fuzzy place + time)

MIT: Political Science Department (Lobbyview)

- Money in politics : record matching, graphs (lobbying+campaign+econ)





CONTEXT: CENTAUR LABS

CENTAURLABS.COM

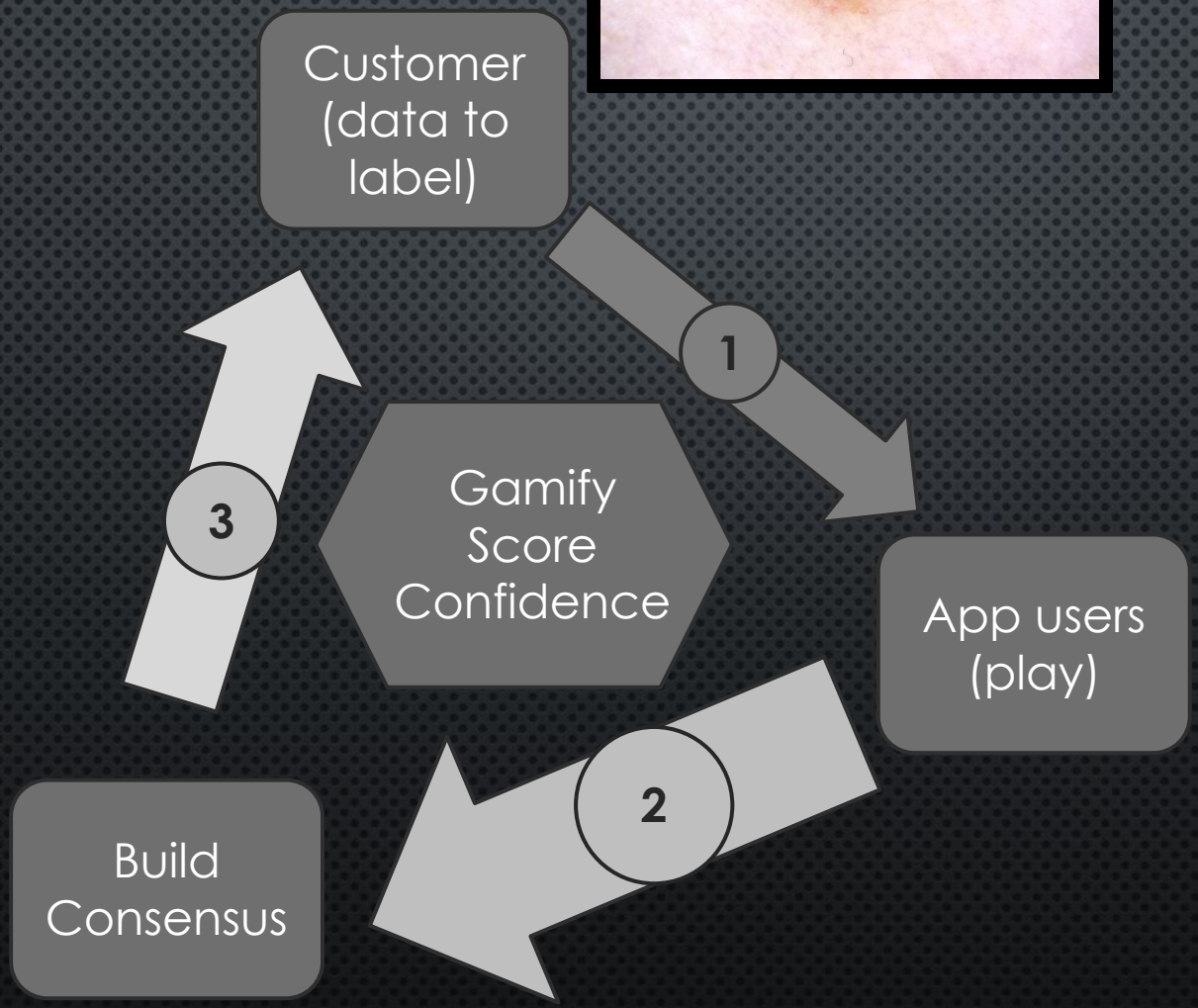
TECH STARTUP (20 EMPLOYEES, 10 DEVS, SERIE A)

WISDOM OF THE CROWD FOR MEDICAL DATA

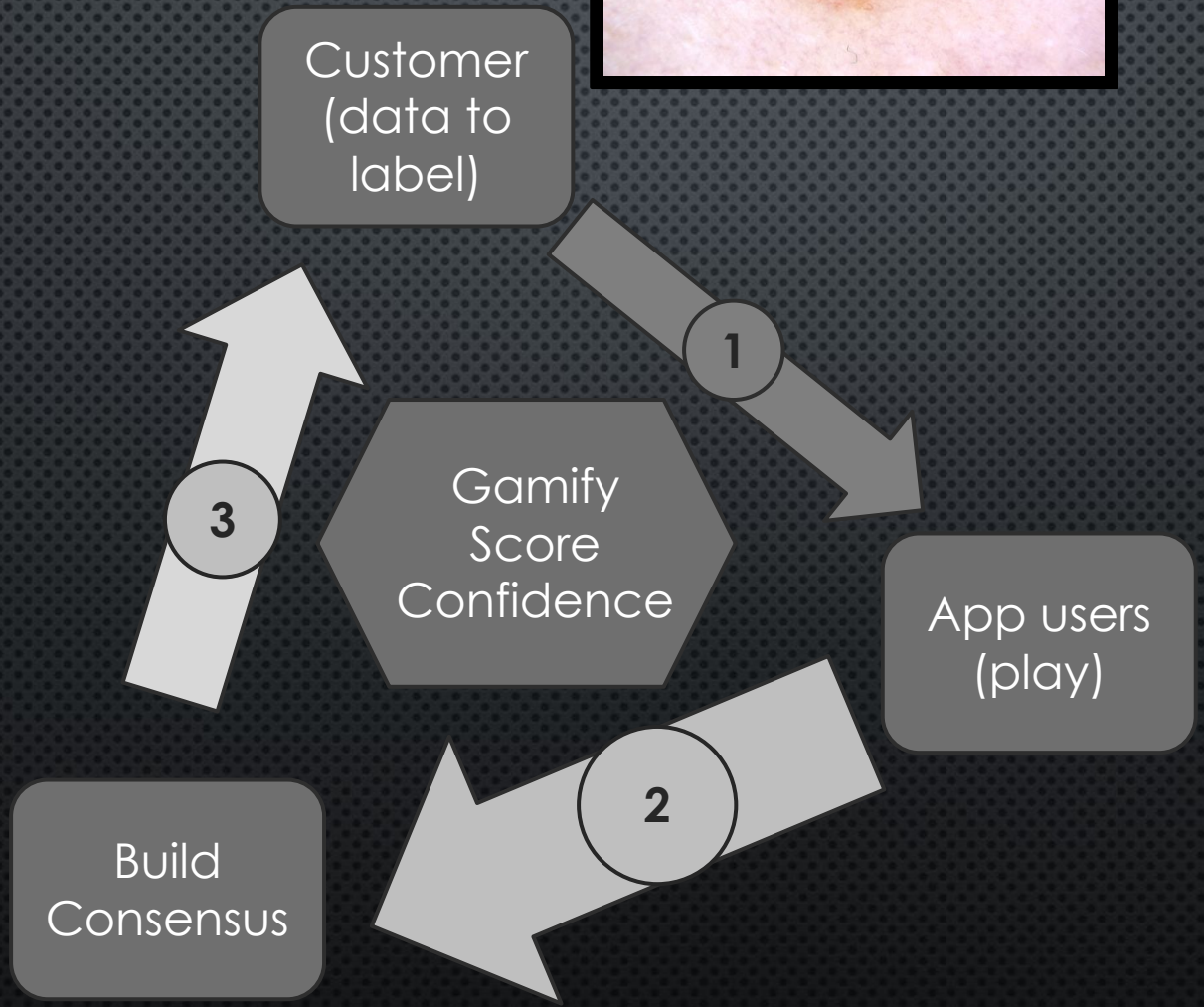
GAMIFIED THROUGH AN APP : DIAGNOSUS.COM

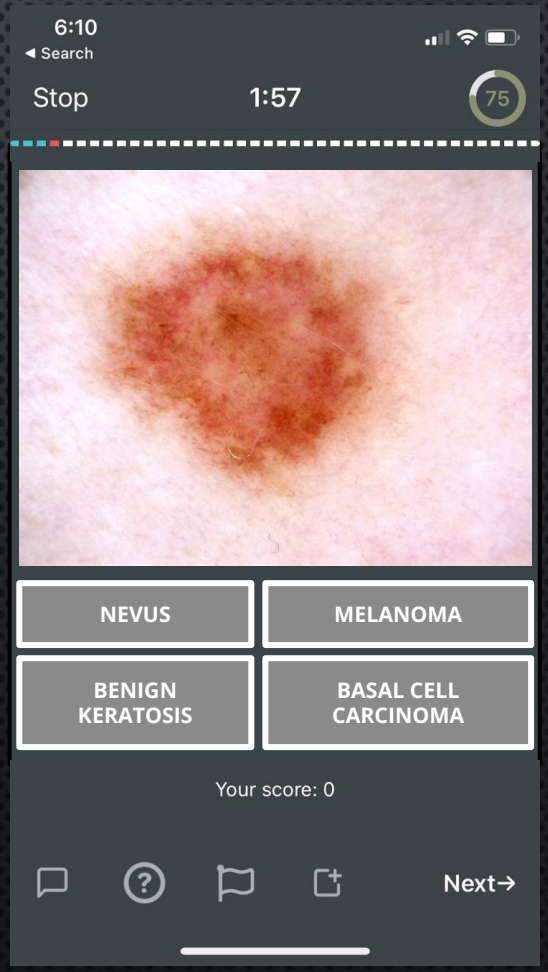
~100 TABLES, 1000 COLUMNS, 300GB, ~400M ROWS

CONTEXT: CENTAUR LABS



CONTEXT: CENTAUR LABS





CONTEXT: CENTAUR LABS

Customer (data to label)

1

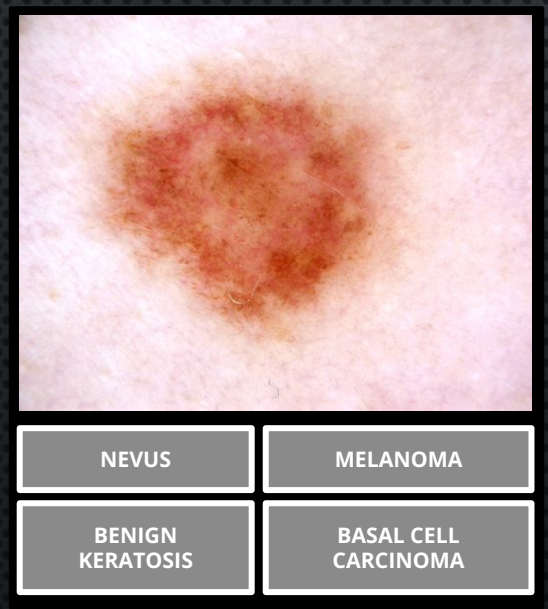
App users (play)

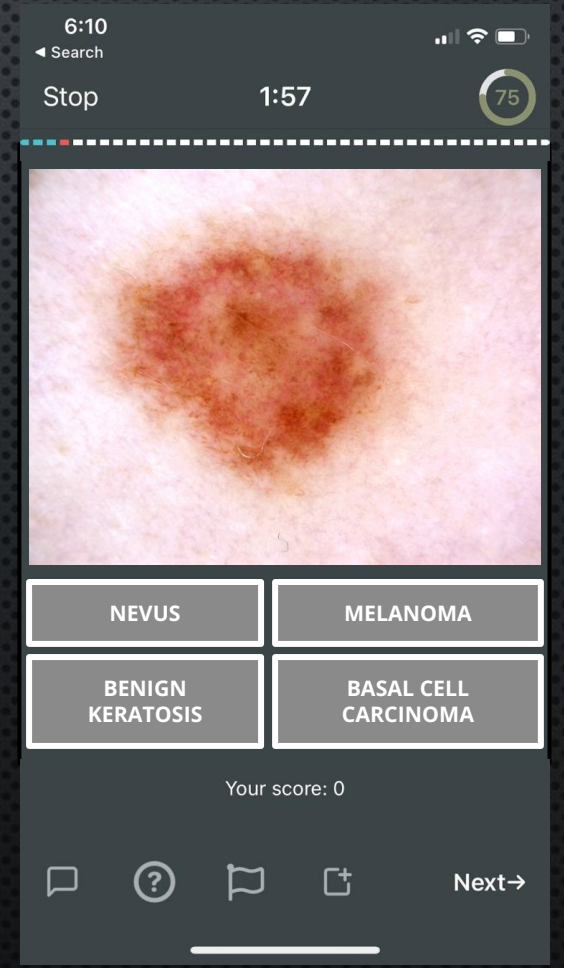
2

Build Consensus

3

Gamify Score Confidence



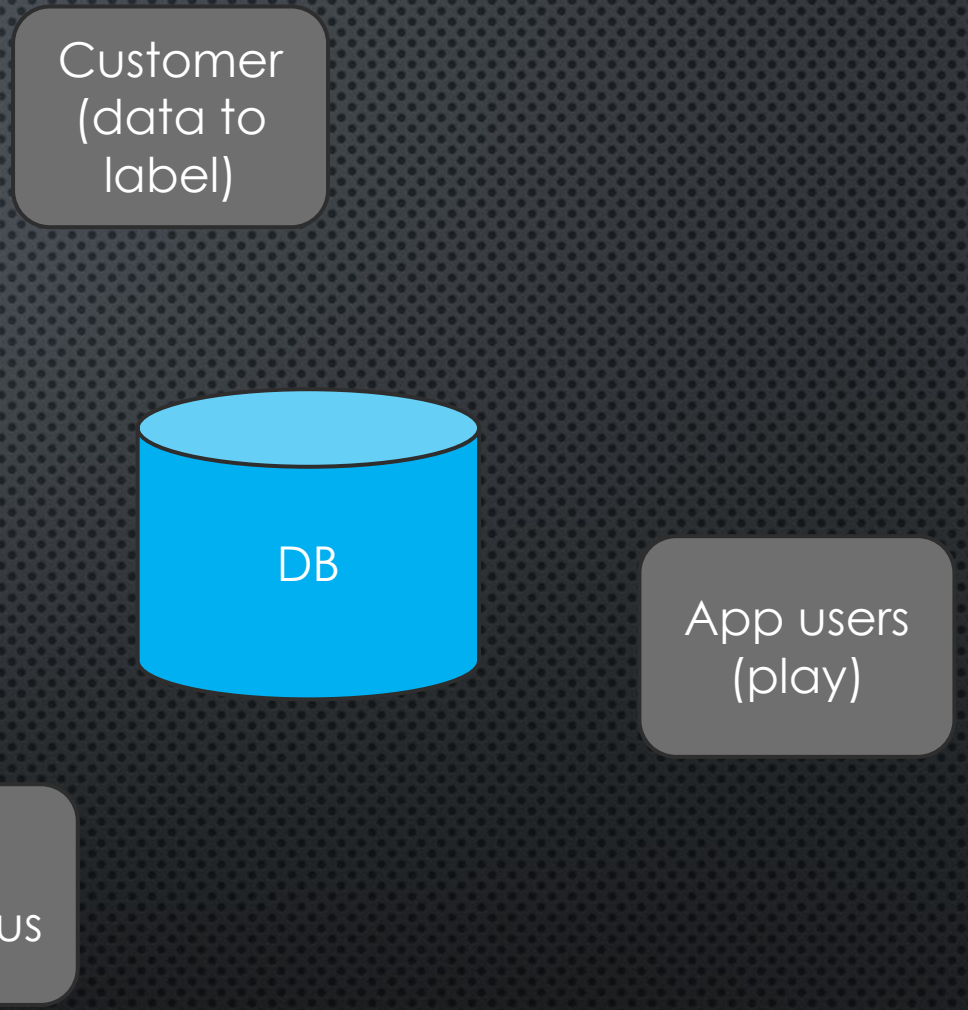


CONTEXT: SUMMARY

DB-CENTRIC: 8/10 DEVS WRITE SQL

DB IS NOT ONLY DATA STORAGE

- DATA TRANSFORM
- BUSINESS LOGIC
- CONVENIENCY/HELPER FUNCTIONS



WHY A DEV DB?

WHY A DEV DB? SAFER 1/2

- Cascaded safety
  - Parachute: 1/5k chance of using reserve
    - ~ odds of death biking
  - Parachute: 1/220k chance of dying
    - ~ odds of death by lightning
  - My own DBA stats at Centaur Labs:
    - DEV DB: ~1 error /week
    - PROD DB: ~1 incident/year
- Easier undo
  - Meh, why bother undoing
  - Reset whole dev db
    - Every 2 weeks or manually
  - Ok I'll undo … without time pressure/stress

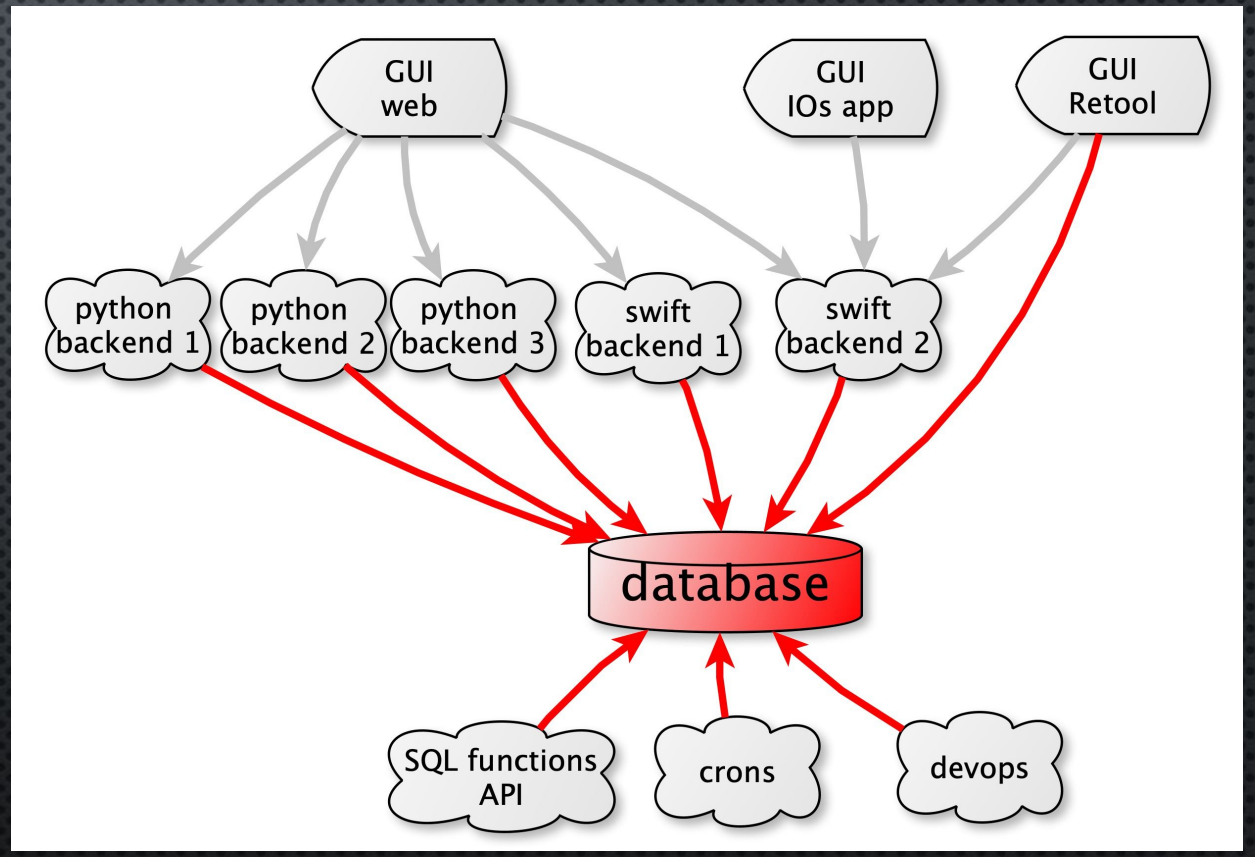
Image

WHY A DEV DB? SAFER 2/2

- **ANY** CHANGE : HOW DO YOU KNOW IT WORKS?
- TESTING !
  - "THIS FUNCTION SHOULD DETECT BAD DATA"
  - → I NEED BAD DATA TO TEST IT
  - → PUT BAD DATA IN PROD??
  - PROD: ALERTING, LOGGING, … → DON'T WANT TO TRIGGER THAT
- DEV DB IS THE PERFECT PLACE FOR THAT

WHY A DEV DB? EMPOWERS DEVS

- 3 GUI's , 5 BACKENDS …
- NEED EMPOWERED DEVS
  - WE LEARN BY MAKING MISTAKES
  - QA/ CODE REVIEW
    - MORE POWERFUL TO REVIEW SCHEMA+DATA THAN CODE IMO
  - PERMISSIONS
    - 3 DEVS HAVE DDL PERMISSIONS ON PROD
    - 9 HAVE DDL SKILLS AND PERMISSIONS ON DEV
- FIRST IDEA != BEST IDEA
  - FREEDOM TO EXPERIMENT



WHY A DEV DB? CONFIDENCE = SPEED

Image

Image

ON WHICH PATH CAN YOU GO FAST?

WHY?

- CONSEQUENCES

WHY A DEV DB? CONFIDENCE = SPEED

Image

Image

PROD □ IN USE!

ANY QUERY CAN SLOW IT /BRING IT DOWN

- DDL, DELETE, UPDATES
- SELECT (RESSOURCES, LOCKS)

REDUCE CONSEQUENCES □ FASTER DBA WORK

Why a dev DB? Take away

- DEV DB:
  - Safer
    - Cascaded safety
    - Tests
  - Empowers devs
  - More confidence, more speed

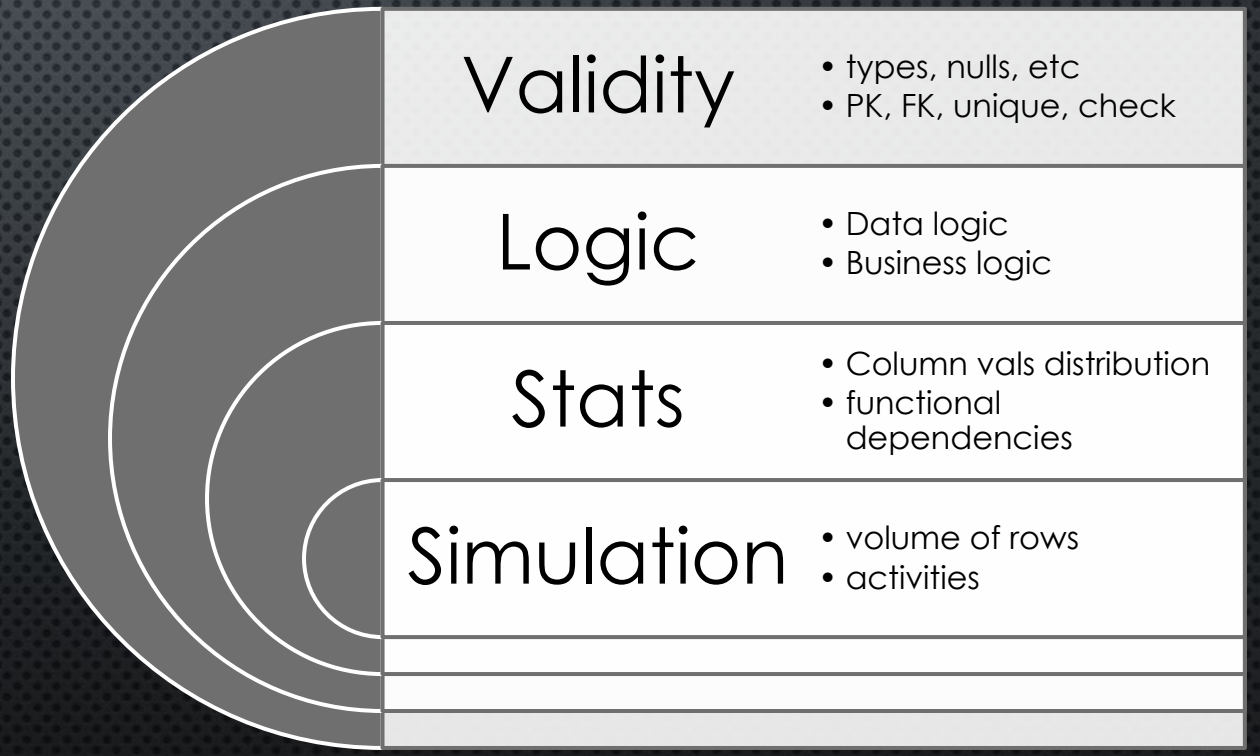
What dev data?

WHAT DATA FOR THE DEV DB?

SYNTHETIC VS PROD DATA

SYNTHETIC DATA

- SYNTHETIC: INSERT MADE-UP STUFF
  - WHAT IS "MADE-UP"?
  - DEPENDS ON USE CASE
    - **ALL** : VALIDITY
    - **DEV**: LOGIC
    - **DATA SCIENCE**: STATS
    - **DEVOPS**: SIMULATION



Synthetic data

- Pro:
  - Perfect for tests !
  - For each component: Making up rare data
    - testing SQL injection robustness
    - bad geometries collection
  - For all component: Integration test
    - Cover "basic"/ "most common" scenario

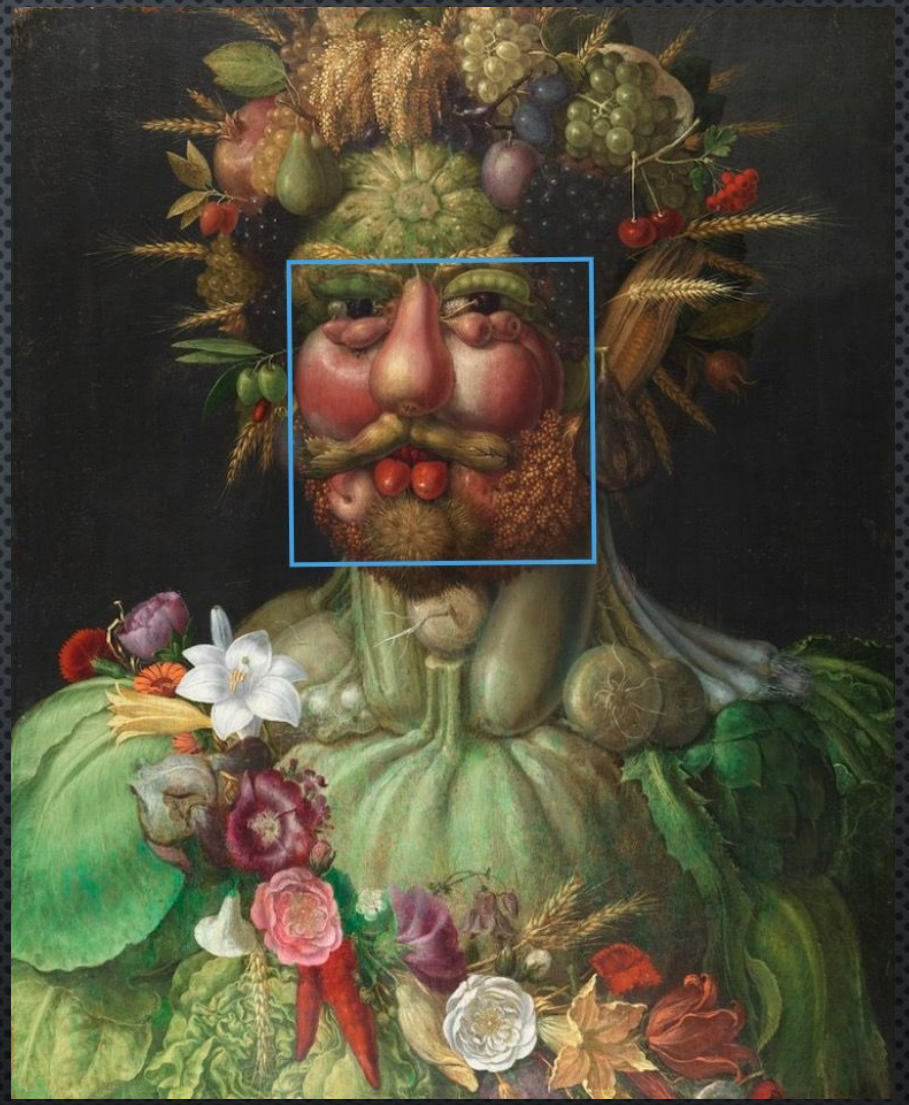


Image with face detection service

Synthetic data

- Cons:
  - Tradeoff "realistic" vs "pain to generate"
    - "realistic" is very very hard:
      chatgpt: realistic text generation .. for 1 column …
    - Companies sell this saas
  - Another piece of code to maintain
  - You will need some prod data (nomenclature tables)

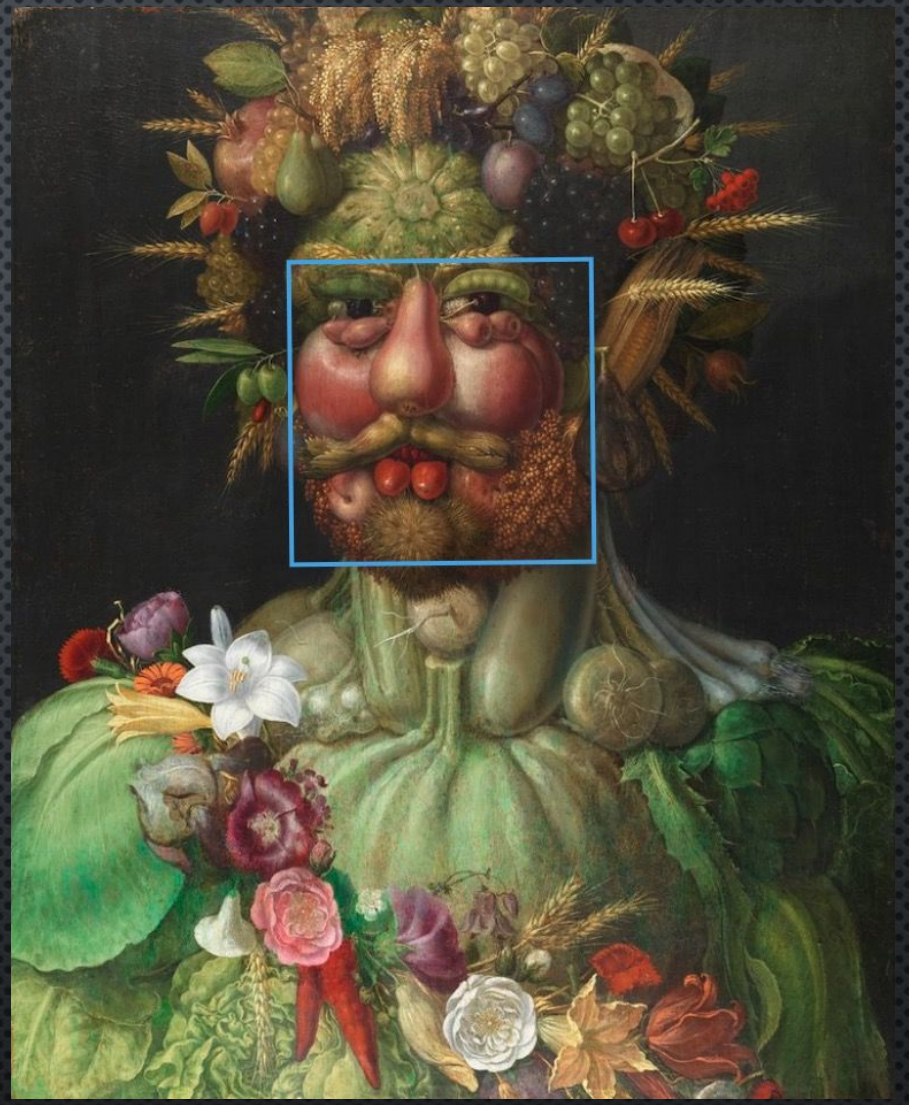
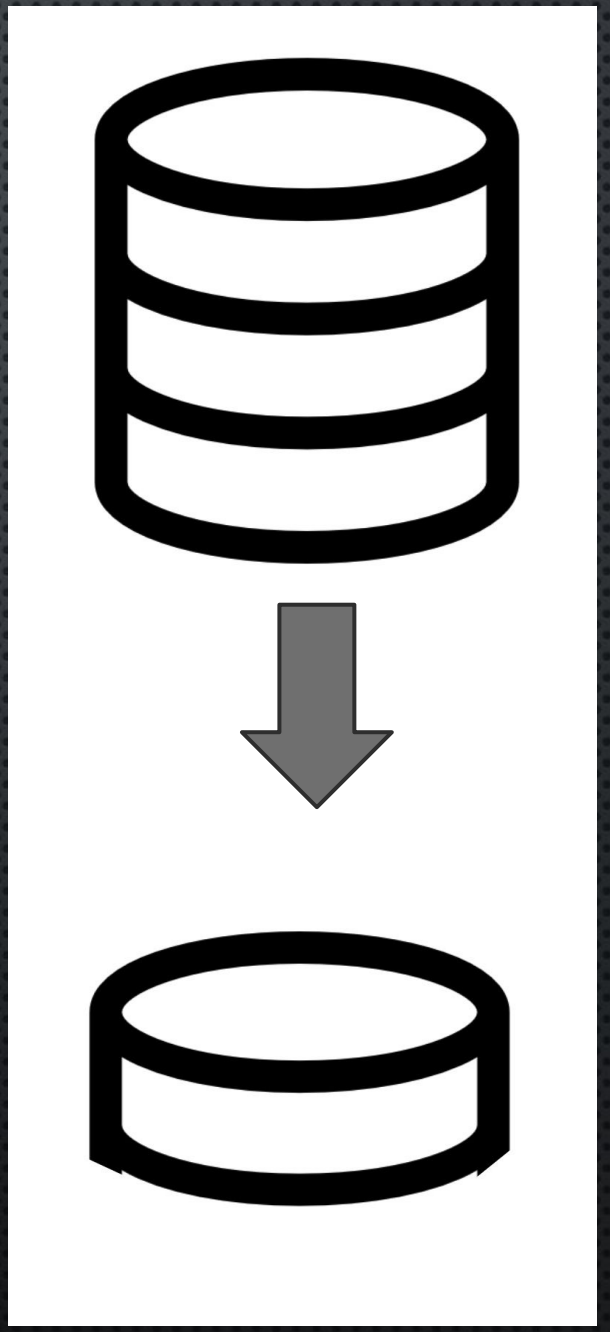


Image with face detection service

PROD DATA

- Prod data: copy data from prod
  - Still need to fulfill PK/ FK
    - All data dependencies fulfilled
    - Right order
  - PRO
    - Ultimate "realistic" data
    - Debug using actual data trail
  - CONS
    - What to copy?
  - Big security liability
    - Must be obfuscated / masked / scrambled



Synthetic data / prod data takeaway

- Data for DEV takeaway
- Synthetic
  - Great to manufacture tests
  - "realistic" is hard
- Prod
  - As "real" as it gets
  - Must be obfuscated

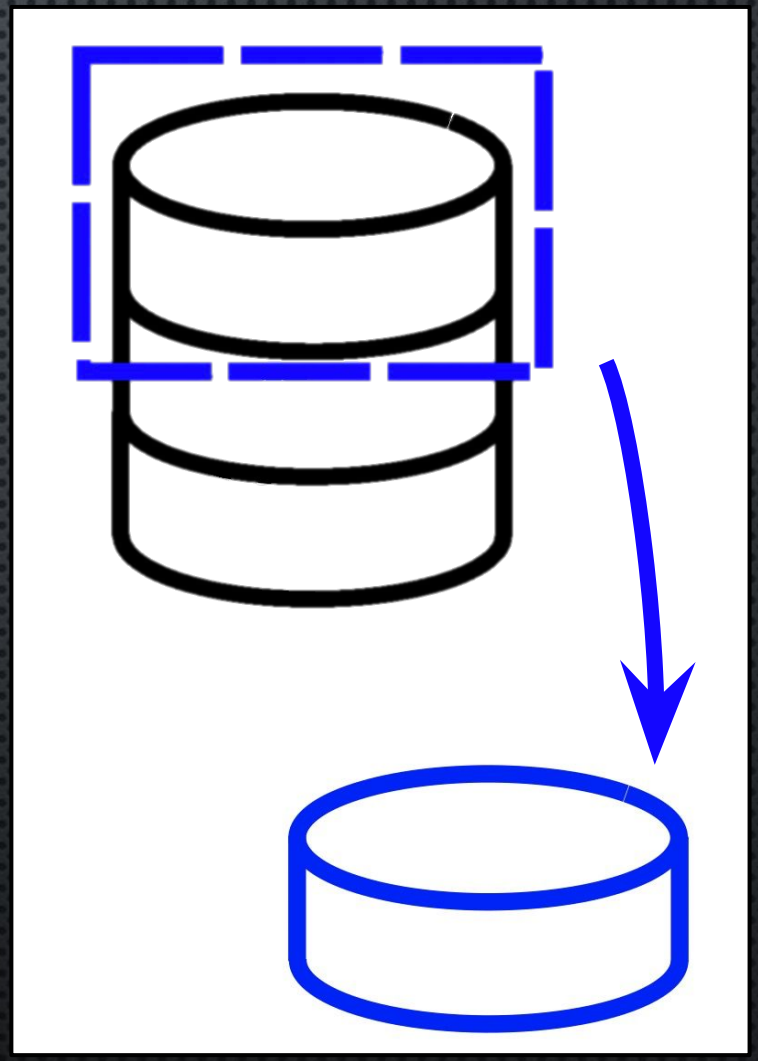
- *How to get PROD data into DEV DB? → 3 strategies*

GETTING PROD DATA INTO DEV DB

3 STRATEGIES

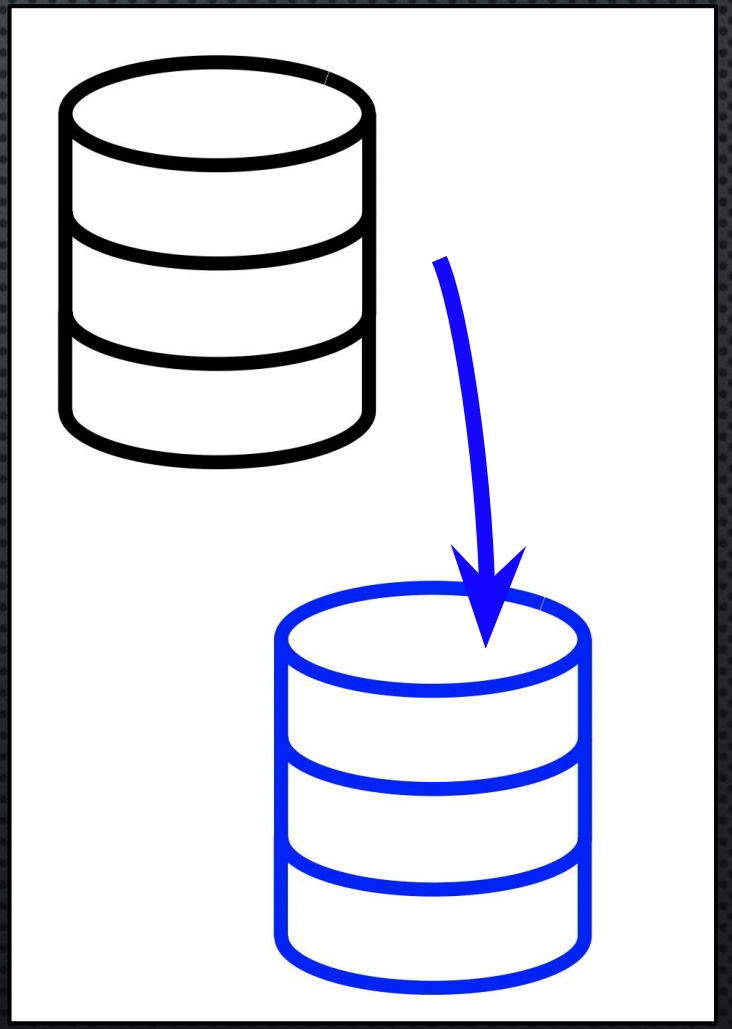
PROD□DEV : PARTIAL INSERT

- Partial insert
  - PROD: Dump schema
  - DEV : Create schema
  - DEV: create FDW
  - DEV: SQL: select from FDW + insert
- PROs:
  - get only what you need
    - Fast, clean
    - Unlimited undo (delete all then re-insert)
  - Can copy very fresh data for debug
- CONs:
  - Hard to write:
    - satisfy FK (insert right thing in right order)
    - Also need static data
  - Hard to maintain:
    - change to the PROD schema □ change partial copy script



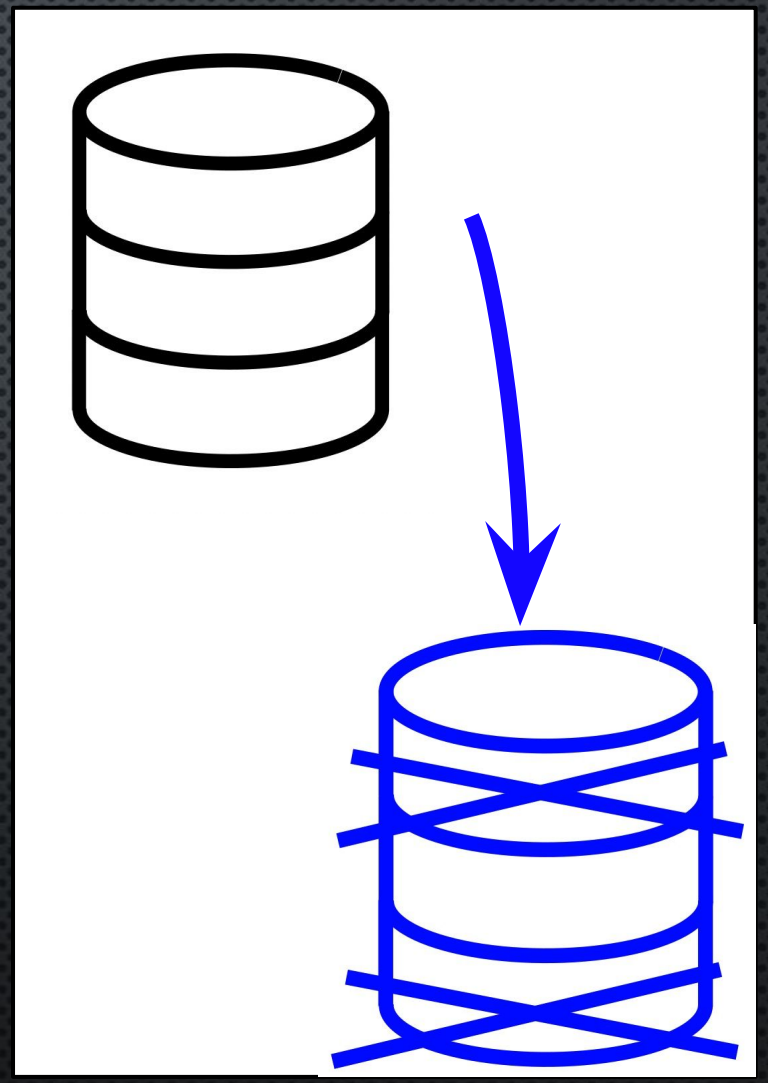
PROD□DEV : CLONE

- CLONE
  - PROD: CLONE
  - DEV : INSTANTIATE
  - DEV : PARALLEL VACUUM
- PROS:
  - SIMPLEST TO MAINTAIN
  - ALL THE DATA
    - HIGHER CHANCES TO CATCH BUGS/MORE REALISTIC
- CONS:
  - ALL THE DATA
    - NEED BEEFIER INSTANCE, SLOW OPS
  - MORE DATA TO OBFUSCATE
  - LESS CONTROL (SECURITY)



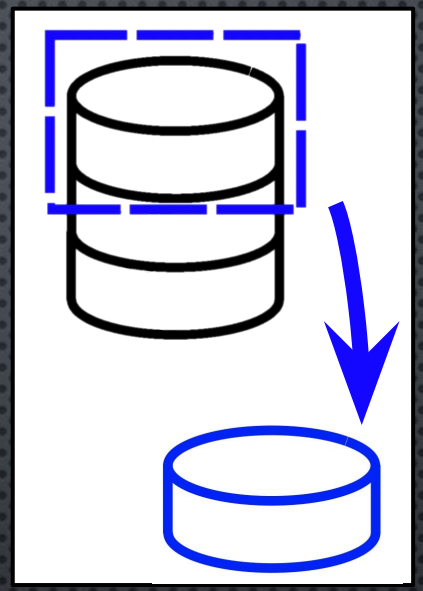
PROD□DEV : CLONE + DELETE

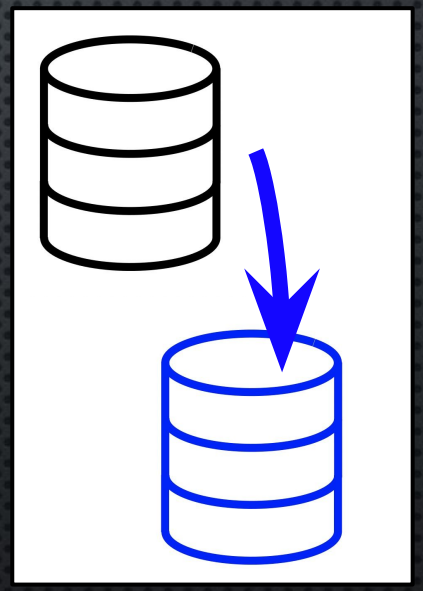
- CLONE THEN PARTIAL DELETE
  - PROD: CLONE
  - DEV : INSTANTIATE
  - DEV VACUUM
  - DEV: CHANGE ALL FK TO "ON DELETE CASCADE"
  - DEV: ADD INDEXES FOR FAST DELETION
  - DEV: PARALLEL DELETE
- PROS:
  - KEEP ONLY WHAT YOU NEED
  - SCRIPT TO DELETE VERY EASY TO MAINTAIN
- CONS:
  - INDEXING FOR FAST DELETION: LONG + A PAIN
  - DELETE CAN BE VERY SLOW IF LOTS OF DATA

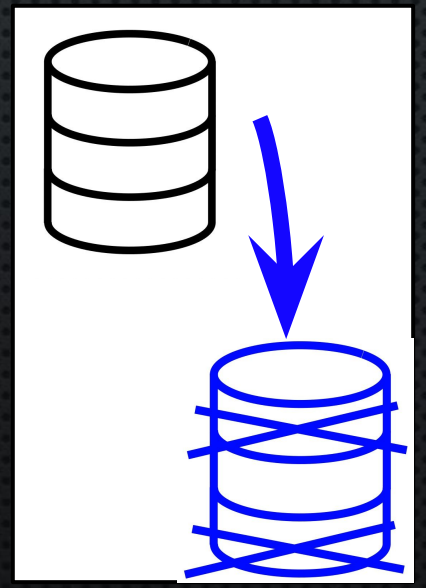


PROD□DEV : TAKEAWAY

- 3 strategies
  - Partial insert
  - Full clone
  - Clone + delete







In all cases: prod data in dev ?

- Regulation (data breach in waiting)
- Confidentiality (customer names, …)
- Privacy (Password, emails)

□ we need to obfuscate it!

OBFUSCATING PROD DATA

What is obfuscation

Obfuscation:

- Keep data useful but disguise it
- Maintain (constraints, structure, format)
- 3 main methods
  - Keep original content, but hide it
    - Encryption: (can be undone)
  - Replace original content, but can trace back
    - Tokenization, (deterministic) uuids
  - Replace original content, random
    - Scrambling

risky

Random is annoying for test

OBFUSCATION WITH UUIDS 1/3

USE DETERMINISTIC UUIDS

- UUID_GENERATE_V5() : TEXT TO UUID
- 'I LOVE PGDAY.DE' → '44F37065-35FF-5C70-8650-38A5BC931556'

- '44F37065-35FF-5C70-8650-38A5BC931556' → 'I LOVE PGDAY.DE'
  - CAN'T BE UN-HASHED
  - USING PROD: CAN BE MAPPED (ANNOYING)

- IF INPUT IS UNIQUE, OUTPUT IS ~ GUARANTEED TO BE UNIQUE
  - OK FOR PK AND FK AND UNIQUESS
  - INCLUDING COMPOSITE PK!
- BUT … CHANGES FORMAT

OBFUSCATION WITH UUIDS 2/3

How to preserve (some) of the original format / pass constraints?

 Use several uuids !

- email:
  - IlovePGday@conf.de uuid1@uuid2.de
- File path
  - /ilove/pgday/conf.de /uuid1/uuid2.de
- AWS S3 custom type
  - (bucket, file_path, region) (uuid1, uuid2, uuid3)

You get the idea

Obfuscation with UUIDs 3/3

Some details:

- May hide more or less
  - Customer emails: hide everything
  - Players email: keep domain in clear
- Obfuscate a PK need to update all tables in the relation at once
- Heavy DB work (need vacuum etc.)

Obfuscation for real

- Using uuid is enough for us
- You can do much better and much more complex
  - See the Postgres Anonymization extension (Dalibo)
  - (vendors options as well)

What to obfuscate? 1/2

Wait … what should be obfuscated in the first place??

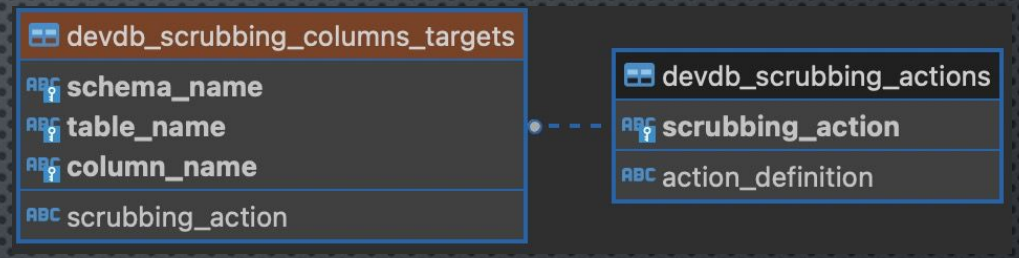
- Regulation
  - Data belonging to customer
  - Patient Health Information
- Privacy
  - Password, api keys, …
  - Emails
  - Display name, etc.
- Confidentiality
  - Customer names !

That's a lot! How to find/keep track?

WHAT TO OBFUSCATE? 2/2

FINDING COLUMNS WITH CUSTOMER NAME

1. LIST OF ALL TEXT COLUMNS
2. LIST OF ALL CUSTOMER NAMES
3. REGEXP MATCHES
4. MANUAL VALIDATION
5. STORE WHAT SHOULD BE DONE WHERE
6. (GENERATE CODE BLUEPRINT)
7. CORRECT CODE
8. PARALLELIZE, WRITE TESTS



schema_name	table_name	column_name	scrubbing_action
providers	customers	name	scramble_string
providers	customers	email	scramble_email

Some DevOps

All credits to Jon Cortez our Senior Dev Ops

Our devops stack:

- Jenkins pipelines to create/destroy the dev db
    - Terraform to book instance + manage parameters
        - Bash script to orchestrate
            - Bash scripts to obfuscate
                - SQL scripts / commands

TAKEAWAY MESSAGE

TAKEAWAY

Testimony:

- A dev db helps
- Using prod data covers many usages
- Safer with obfuscating

- Especially relevant when Postgres is more than stashed data
  - Postgis on medical images
  - SQL function API (>100 fn)
  - Analytics, caching, …

THANK YOU

Thank you very much to the community

Questions