Data Engineering Best Practices

Develop Intelligence

Kerry Nakayama

Best Practices

Develop Intelligence

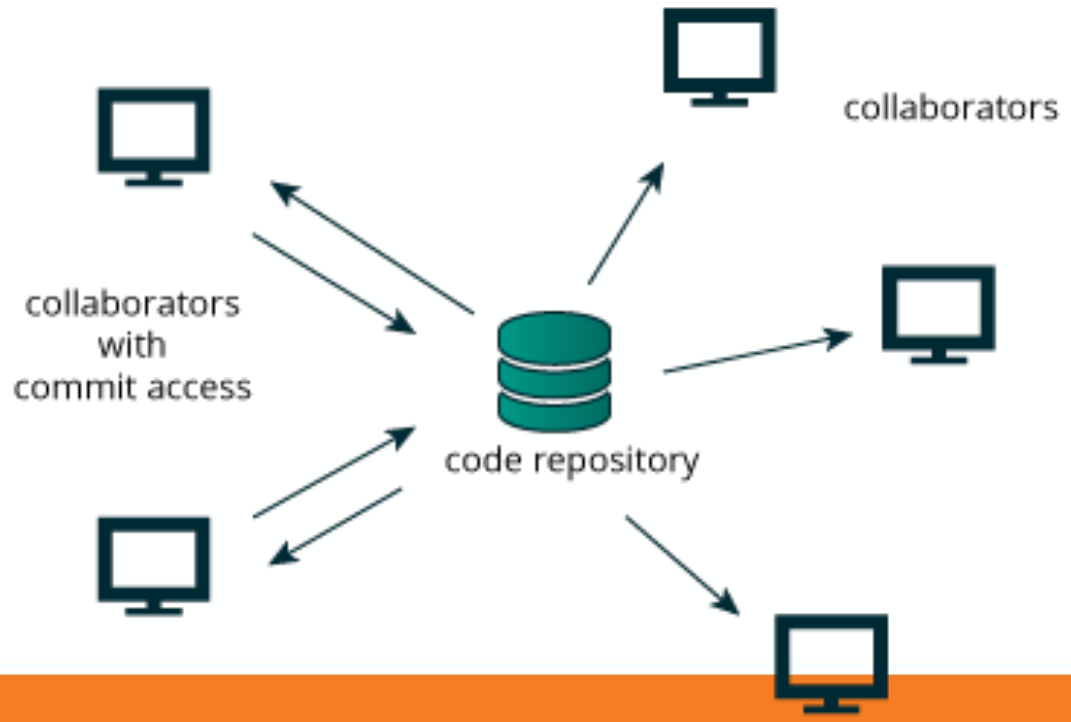
Kerry Nakayama

Objectives

- Creating a code repository
  - Hooks
  - Linting
  - Automatic builds on push to master
  - Parallelization
- dbt integration with code repo
  - YML files
  - Testing
    - Unit testing
    - Integration testing
  - Dealing with Failed data
  - Automatic reruns

Creating Code Repository

- What are they for?
  - Version control
  - Central location for code
  - Code review – peer review
  - Data docs
  - CI/CD
  - Access control



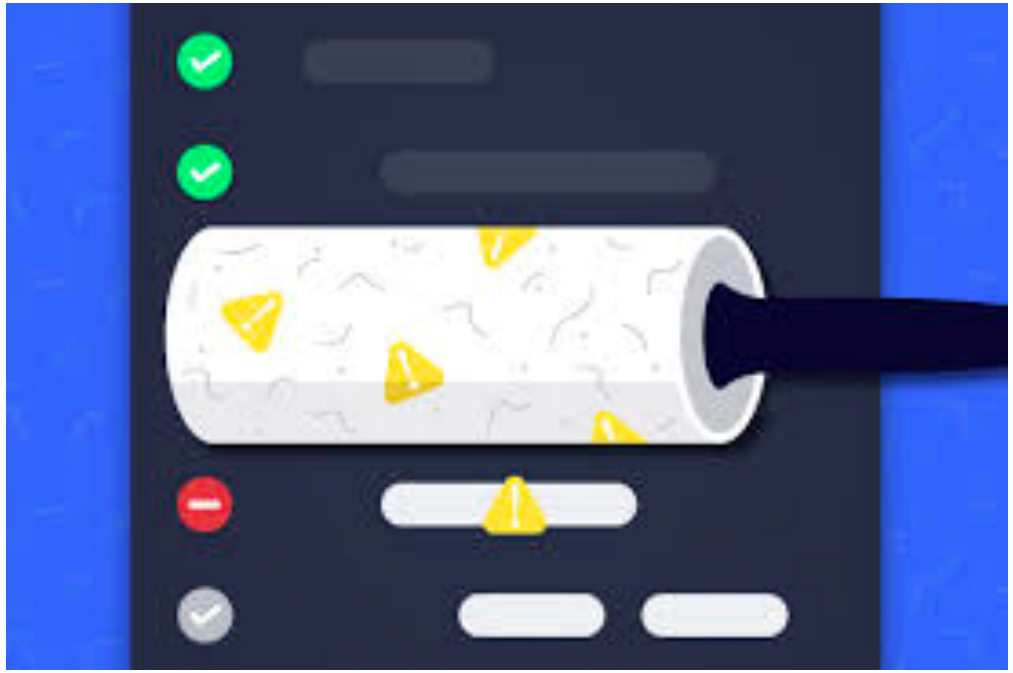
Hooks

- What are Git Hooks?
  - Git hooks are scripts that Git executes before or after events such as: **commit**, **push**, and **receive**
  - Every Git repository has a .git/hooks folder with a script for each hook you can bind to
  - being able to push to your staging or production environment without ever leaving Git is just plain awesome

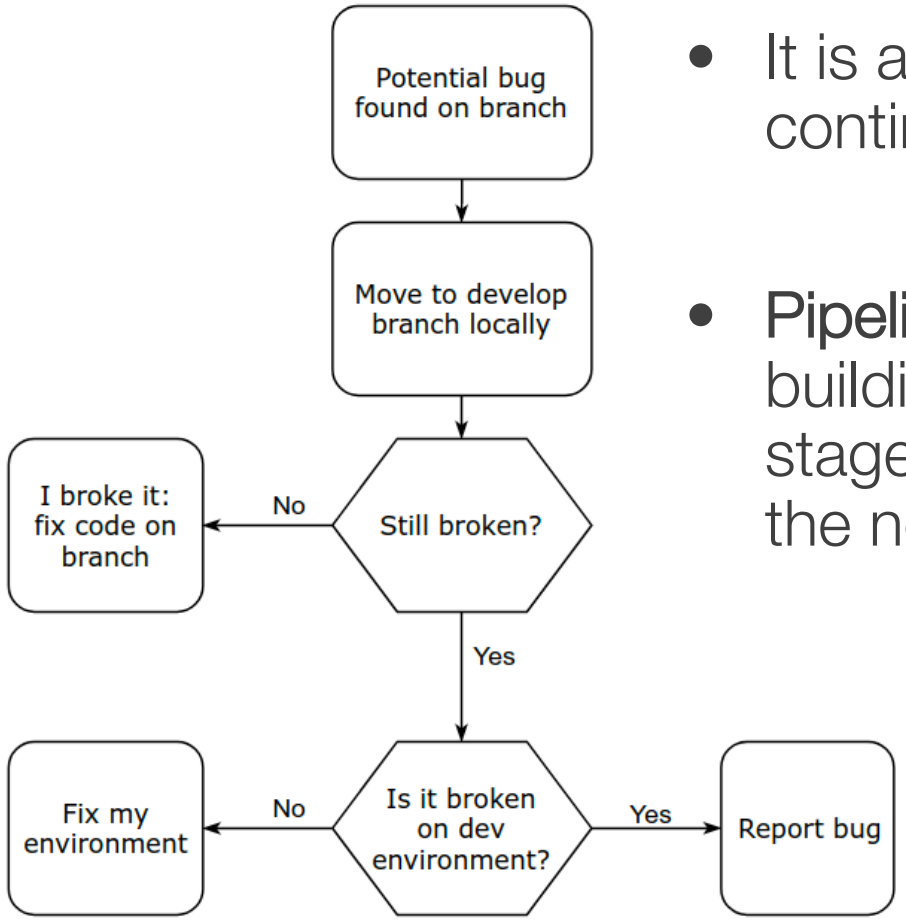
Linting

What is Code Linting?

- Best practices
  - ➢ Linting vs formatting
- Validate can be used to list all errors in the loaded data
- Use linting tool – most are open source



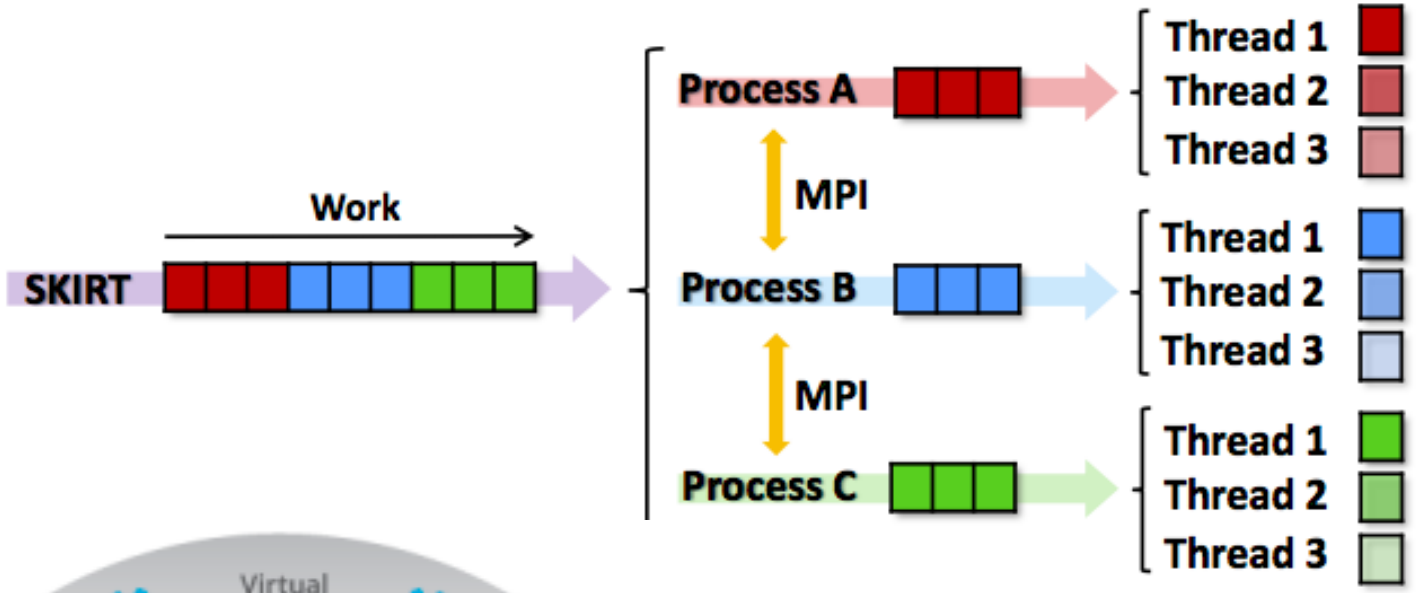
Automatic Builds on Push to Master



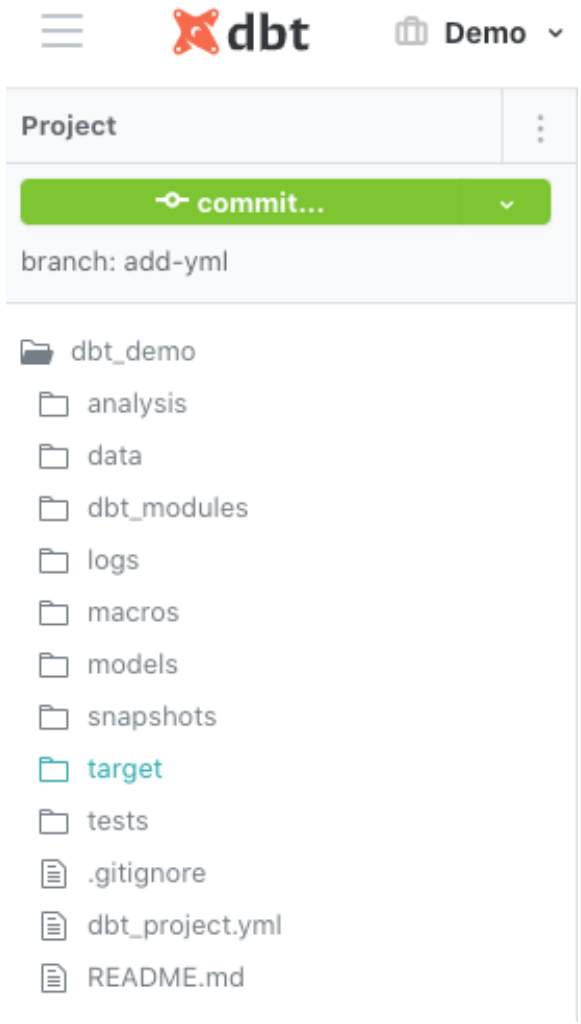
- It is arguably the most popular continuous integration tool in use today
- **Pipeline** - The process of automatically building code in stages – and at each stage, testing and promoting it on to the next stage

Parallelization

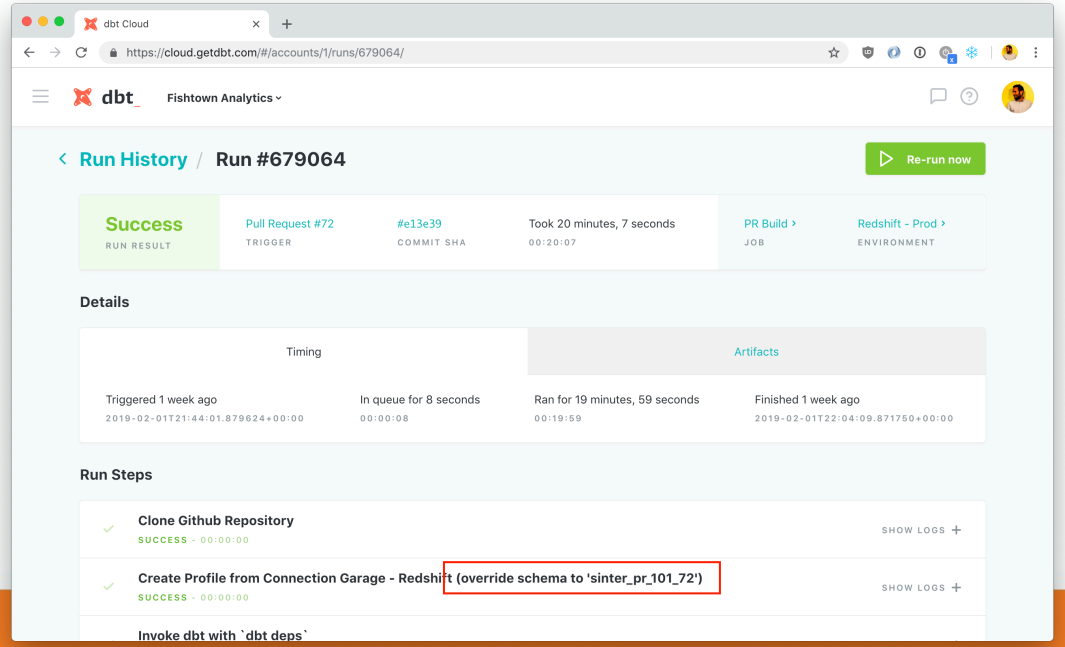
- What is Parallelization?
- How is it being used for your team?
- How Snowflake handles it



dbt integration with code repo



- dbt has created a function that sets up your code repository once connected
  - ➢ dbt init – initializes folders for you to start
- Enabling Continuous Integration (CI)
  - ➢ Use Webhooks to enable runs on Pull requests



YML Files

- dbt_project.yml
  - Every dbt project needs a dbt_project.yml file
  - this is how dbt knows a directory is a dbt project
  - It tells dbt how to operate on your project
  - When dbt runs, it will search through the source-paths directories of your dbt project looking for any files that end with .yml

dbt_project.yml

```yaml
name: string

config-version: 2
version: version

profile: profilename

source-paths: [directorypath]
data-paths: [directorypath]
test-paths: [directorypath]
analysis-paths: [directorypath]
macro-paths: [directorypath]
snapshot-paths: [directorypath]
docs-paths: [directorypath]

target-path: directorypath
log-path: directorypath
modules-path: directorypath

clean-targets: [directorypath]

query-comment: string

require-dbt-version: version-range | [version-range]

quoting:
  database: true | false
  schema: true | false
  identifier: true | false

models:
  <model-configs>

seeds:
  <seed-configs>

snapshots:
  <snapshot-configs>

sources:
  <source-configs>

on-run-start: sql-statement | [sql-statement]
on-run-end: sql-statement | [sql-statement]
```

Testing

- dbt makes it easy to define automated tests on your models

> **Tests are gifts you send to your future self**
>
> Testing is an essential part of a mature analytics workflow. It only takes a minute to set up a test -- your future self will thank you!

- Schema Tests
  - Schema tests are assertions that a model's schema adheres to basic rules: referential integrity, uniqueness, and nullity, for instance
- Custom data tests
  - Data tests are sql SELECT statements that return 0 rows on success, or more than 0 rows on failure

schema.yml

```yaml
version: 2

models:
  - name: people
    columns:
      - name: id
        tests:
          - not_null

      - name: account_id
        tests:
          - not_null

      - name: status
        tests:
          - not_null
```

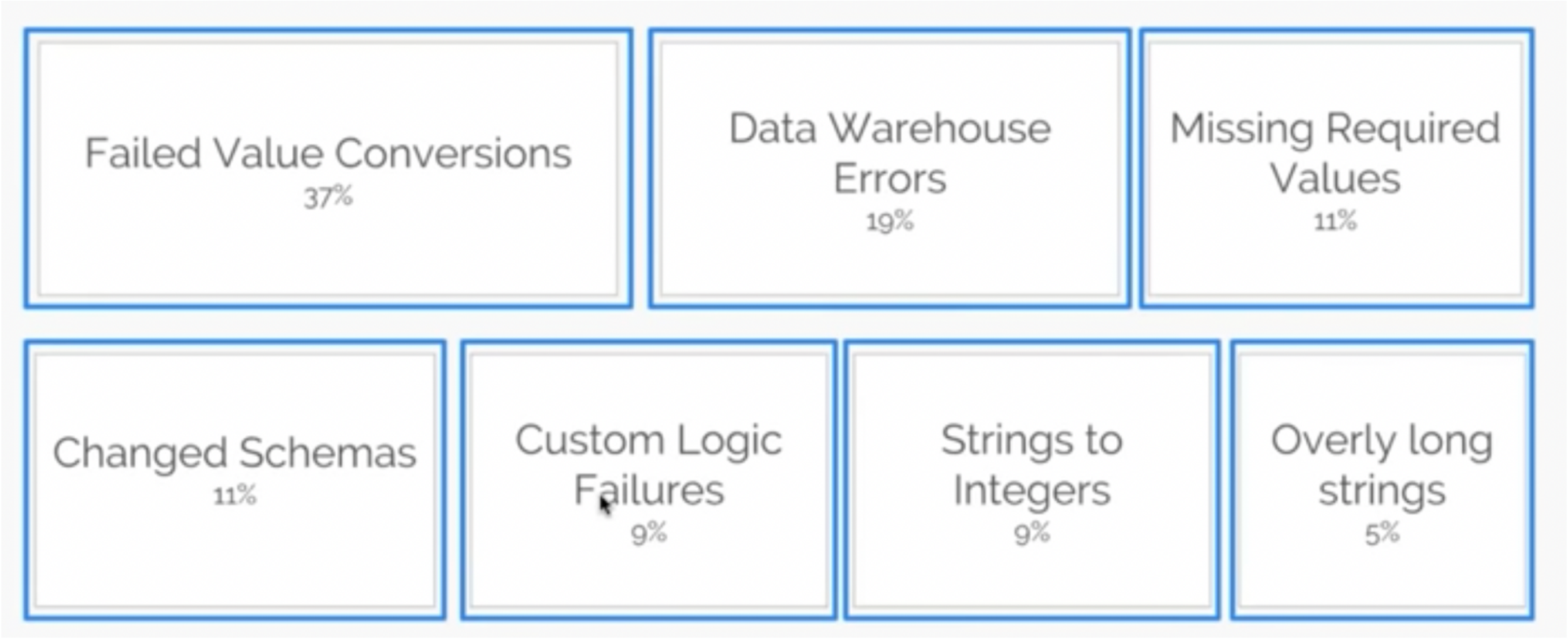
schema.yml

```yaml
version: 2

models:
  - name: people
    columns:
      - name: id
        tests:
          - unique
```

schema.yml

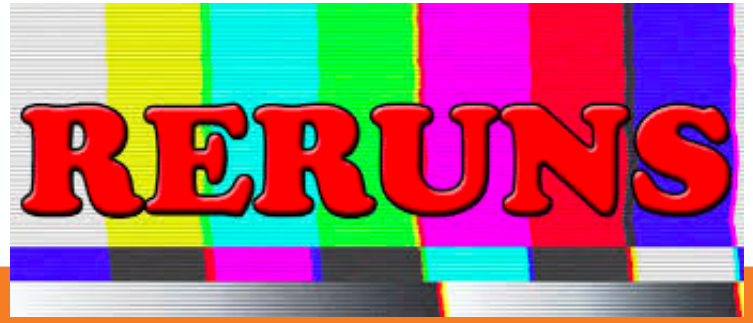
```yaml
version: 2

models:
  - name: people
    columns:
      - name: account_id
        tests:
          - relationships:
              to: ref('accounts')
              field: id
```

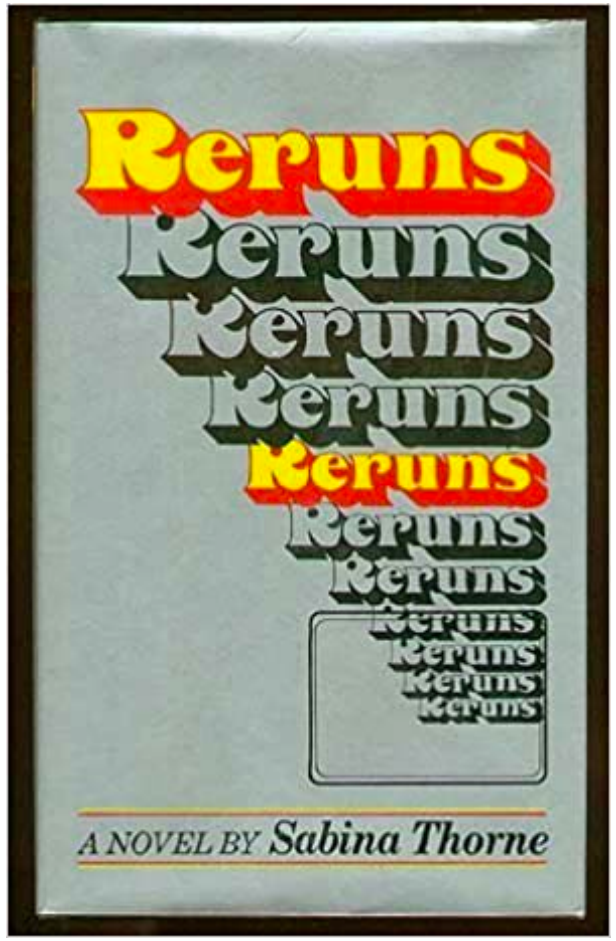
Examples of Errors



Automatic Reruns

- 3rd Party failures for pulling the data happen all the time
- Tests in dbt to check for a data failure
  - ➢ Failures include
    - No data
    - Duplicate data – test for uniqueness
    - Incomplete data – expecting x rows – warning
- Set up a trigger to kick off a scripting loop
  - ➢ Python or jinja code can be used to create loops





Confused? ASK QUESTIONS

Break & Lab 3