Big Data Vilnius

Design philosophy of
Apache Airflow ETL pipelines

I've.. SEEN things... you people wouldn't believe...

- 1200 line SQL views over 15 tables, calculating 3 types of similar measures, duplicated because of a single nullable column, pushed into the same table

I’ve.. SEEN things… you people wouldn’t believe…

- 1200 line SQL views over 15 tables, calculating 3 types of similar measures, duplicated because of a single nullable column, pushed into the same table
- 4 different implementations of *margin* in the same warehouse

I've.. SEEN things… you people wouldn't believe…

- 1200 line SQL views over 15 tables, calculating 3 types of similar measures, duplicated because of a single nullable column, pushed into the same table
- 4 different implementations of *margin* in the same warehouse
- Over a billion records replicated daily out of 250 tables

I've.. SEEN things... you people wouldn't believe...

- 1200 line SQL views over 15 tables, calculating 3 types of similar measures, duplicated because of a single nullable column, pushed into the same table
- 4 different implementations of *margin* in the same warehouse
- Over a billion records replicated daily out of 250 tables
- 3 different implementations of "Sold Product Quantity"

I’ve.. SEEN things… you people wouldn’t believe…

- 1200 line SQL views over 15 tables, calculating 3 types of similar measures, duplicated because of a single nullable column, pushed into the same table
- 4 different implementations of *margin* in the same warehouse
- Over a billion records replicated daily out of 250 tables
- 3 different implementations of “Sold Product Quantity”

Time to do things differently!

“ETL with airflow”

- Process data in “partitions”

“ETL with airflow”

- Process data in “partitions”
- Rest data between tasks (from “data at rest” to “data at rest”)

bigdata REPUBLIC

“ETL with airflow”

- Process data in “partitions”
- Rest data between tasks (from “data at rest” to “data at rest”)
- Deal with changing logic over time (conditional execution)

“ETL with airflow”

- Process data in “partitions”
- Rest data between tasks (from “data at rest” to “data at rest”)
- Deal with changing logic over time (conditional execution)
- Use Persistent Staging Area (PSA)

“ETL with airflow”

- Process data in “partitions”
- Rest data between tasks (from “data at rest” to “data at rest”)
- Deal with changing logic over time (conditional execution)
- Use Persistent Staging Area (PSA)
- “Functional” data pipelines:
  - Idempotent
  - Deterministic

“ETL with airflow”

- Process data in “partitions”
- Rest data between tasks (from “data at rest” to “data at rest”)
- Deal with changing logic over time (conditional execution)
- Use Persistent Staging Area (PSA)
- “Functional” data pipelines:
  - Idempotent
  - Deterministic
- Parameterized workflow

“ETL with airflow”

- Process data in “partitions”
- Rest data between tasks (from “data at rest” to “data at rest”)
- Deal with changing logic over time (conditional execution)
- Use Persistent Staging Area (PSA)
- “Functional” data pipelines:
  - Idempotent
  - Deterministic
- Parameterized workflow
- Data checks as part of your workflow

“ETL with airflow”

- Process data in “partitions”
- Rest data between tasks (from “data at rest” to “data at rest”)
- Deal with changing logic over time (conditional execution)
- Use Persistent Staging Area (PSA)
- “Functional” data pipelines:
  - Idempotent
  - Deterministic
- Parameterized workflow
- Data checks as part of your workflow
- Alerting and SLA’s

Partition ingested data and rest data between tasks

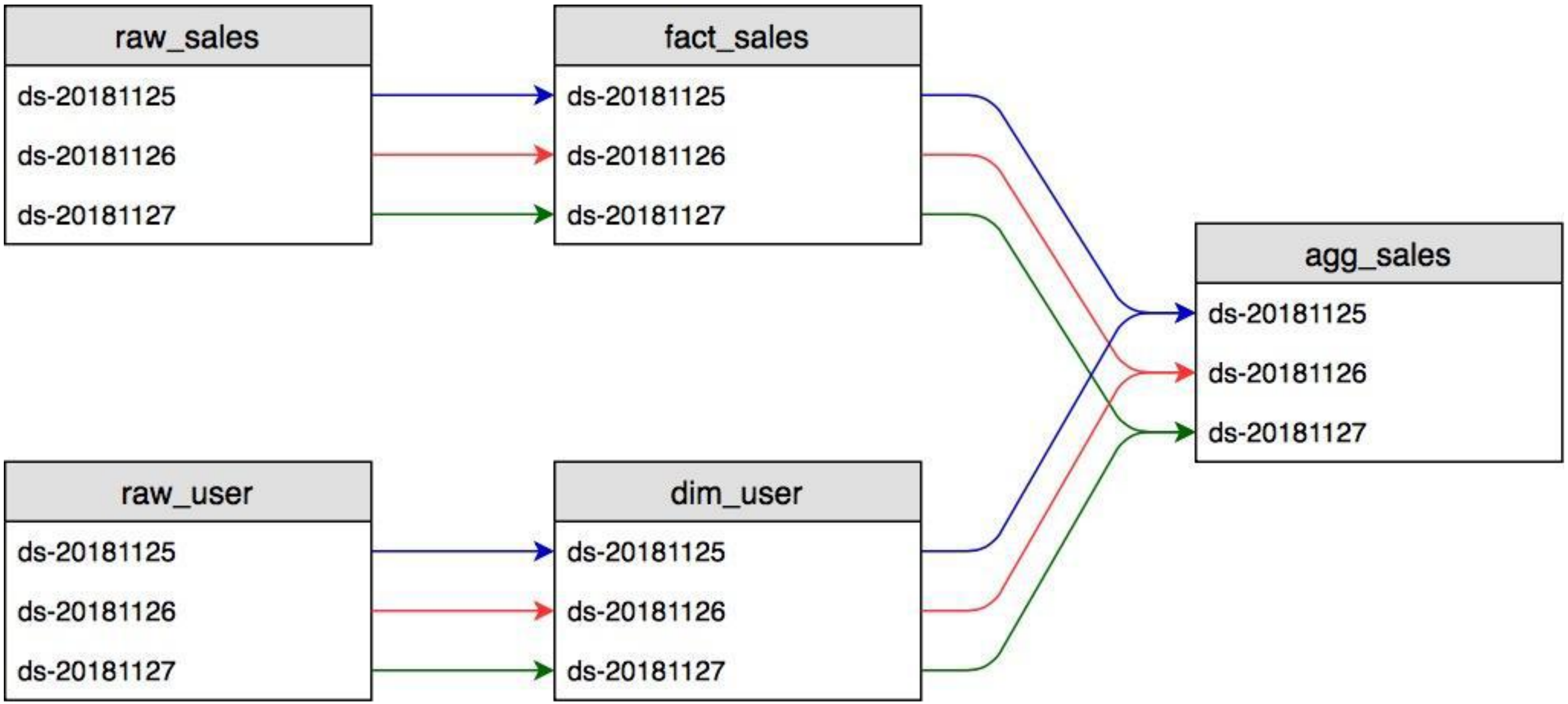
Partitioning data means you build immutable sets of partitions

"INSERT OVERWRITE" partitions

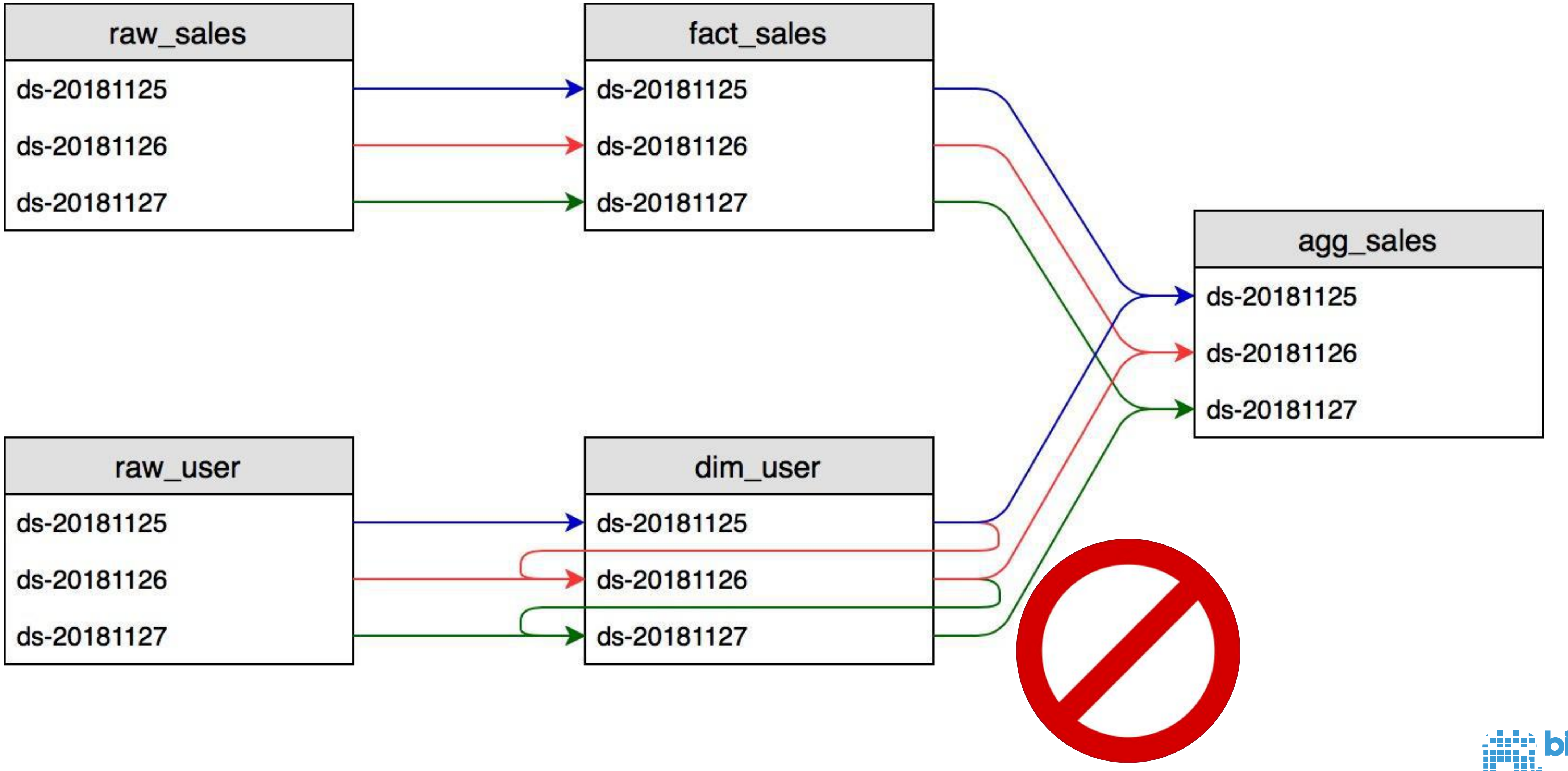
Partitions align with ETL schedule and intervals

- every "1" day?
- every 4 hours?
- every 15 minutes?

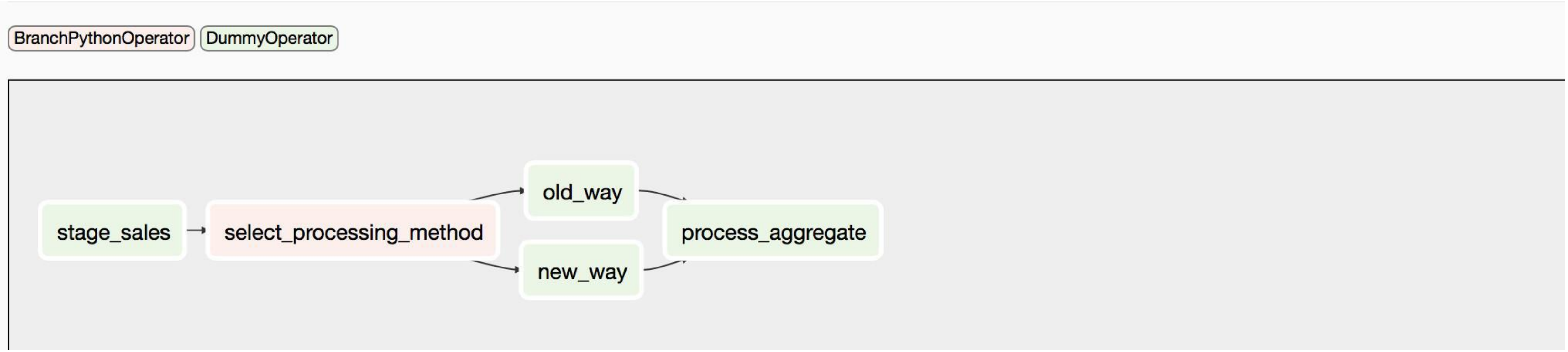
Partition ingested data and rest data between tasks



Partition ingested data and rest data between tasks



Dealing with changing logic over time




```
def decide_flow(**context):
   if (context['execution_date'] < datetime.datetime(2018,1,1)):
      return "old_way"
   return "new_way"
```

Persistent staging area

- Decouples your analytics datasets from the sources

Persistent staging area

- Decouples your analytics datasets from the sources
- Storage is cheaper and distributed, unlike before

Persistent staging area

- Decouples your analytics datasets from the sources
- Storage is cheaper and distributed, unlike before
- Maintains full partitioned history with schema

Persistent staging area

- Decouples your analytics datasets from the sources
- Storage is cheaper and distributed, unlike before
- Maintains full partitioned history with schema
- Better than a db backup (99.999999999% durability) (availability is 99.99%)

Reproducibility

Reproducibility is foundational to scientific method

- Legal perspective
- Bug solving
- Fixing design issues
- Your sanity

Approaching ETL “functionally” yields reproducibility

Functional ETL/ELT

- Pure functions
- Immutable
- Idempotent
- Deterministic

Pure functions

- Limited to their own scope
- Output depends only on input
- No side effects
- Easy to unit test
- Never UPDATE, DELETE, APPEND (no mutations)
- Limited # of source partitions (input)

Pure functions

pure function:

```
def f(x):
    return x+1
```

not a pure function:

```
a = 5

def f(x):
    global a
    a = a + x
    return a
```

not a pure function:

```
def f(x):
    f = open('file', 'r')
    f.write(x)
    return x+1
```

Immutability

- “Once a variable is assigned, it is fixed”
- “Once a partition is processed, its data is not mutated”

Idempotency, Determinism

Idempotent:

“No changes in output state when called multiple times.”

Deterministic:

“A function’s output only depends on its input, not on hidden or global state.”

Parameterized workflow I

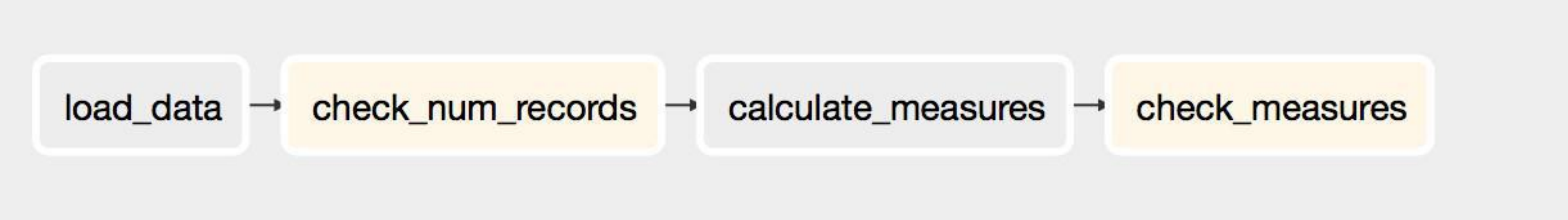
```
copy_task = BigQueryOperator(
    sql='my_data_pipeline/query.sql',
    destination_project_dataset_table='project.dataset.table',
    write_disposition='WRITE_TRUNCATE',
    create_disposition='CREATE_IF_NEEDED',
    bigquery_conn_id='gcp_svc_account',
    pool='my_pool')

copy_task >> some_other_task
```

Parameterized workflow II

```
SELECT   "{{ ds_nodash }}" as date, repo,
    SUM(stars) as stars_last_28_days,
    SUM(IF(_PARTITIONTIME BETWEEN TIMESTAMP("{{ macros.ds_add(ds, -6) }}")
            AND TIMESTAMP("{{ ds }}"), stars, null)) as stars_last_7_days,
    SUM(IF(_PARTITIONTIME BETWEEN TIMESTAMP("{{ yesterday_ds }}")
            AND TIMESTAMP("{{ ds }}") , stars, null)) as stars_last_1_day
FROM
     `airflow-cloud-public-datasets.github_trends.github_daily_metrics`
WHERE _PARTITIONTIME BETWEEN TIMESTAMP("{{ macros.ds_add(ds, -27) }}") AND
             TIMESTAMP("{{ ds }}")
GROUP BY
  date,
  repo
```

Data checks as part of your workflow



class **airflow.operators.check_operator.IntervalCheckOperator**(*table, metrics_thresholds, date_filter_column='ds', days_back=-7, conn_id=None, *args, **kwargs*) [source]

Bases: `airflow.models.BaseOperator`

Checks that the values of metrics given as SQL expressions are within a certain tolerance of the ones from days_back before.

Note that this is an abstract class and get_db_hook needs to be defined. Whereas a get_db_hook is hook that gets a single record from an external source.

Parameters:

- **table** (*str*) – the table name
- **days_back** (*int*) – number of days between ds and the ds we want to check against. Defaults to 7 days
- **metrics_threshold** (*dict*) – a dictionary of ratios indexed by metrics

Alerts and SLA’s

```
EMAIL = 'data-engineering-team@acme.com’

default_args = {
   'owner': 'airflow',
   'start_date': datetime.datetime(2018, 1, 1),
   'email': [EMAIL],
   'email_on_failure': True,
   'email_on_retry': False,
   'retries': 2,
   'retry_delay': timedelta(minutes=5),
   'sla': timedelta(hours=5),
   'execution_timeout': timedelta(hours=7)
}

dag = DAG(
   'sla_example_dag',
   default_args=default_args,
   description='A simple SLA demonstration DAG',
   schedule_interval=’0 0 * * *’
)
```

“ETL with airflow”

- Process data in “partitions”
- Rest data between tasks (from “data at rest” to “data at rest”)
- Deal with changing logic over time (conditional execution)
- Use Persistent Staging Area (PSA)
- “Functional” data pipelines:
  - Idempotent
  - Deterministic
- Parameterized workflow
- Data checks as part of your workflow
- Alerting and SLA’s

What's complicating with Kimball

- Many up-front design choices
- The DWH is subject to mutations (MERGE/UPDATE)
- Often too many concerns covered in one SQL statement
- History is lost with type 1 SCD

A bigdata way to solve Slowly Changing Dimensions

Snapshot the dimensions…

Current attributes:

```
SELECT * FROM
fact a INNER JOIN dim b
ON a.dim_id = b.dim_id AND
b.date_partition = '{current_date}'
```

Historical attributes:

```
SELECT * FROM
fact a INNER JOIN
dim b ON a.dim_id = b.dim_id AND
b.date_partition = a.date_partition
```

“Time series over your dimensions”

The future of airflow: “ETL code generation”

- **owner**: team-data-engineering@acme.com
- **time_frames**:
  - 1 day
  - 7 days
  - 1 month
- **dimensions**:
  - device_type
  - customer_type
- **source_data_set**: ab_experiment_B250
- **demographics**:
  - age
  - gender

The future of airflow: “metrics definition”

- **metric_name**: sold_product_quantity
- **subject**: user
- **sql**: SELECT … FROM … WHERE … GROUP BY …
- **dependencies**:
    - sold_product_history
- **dimensions**:
    - product

The future of airflow: “data lineage”

M | Airbnb Engineering & Data Science Following

HOME AI BACKEND DATA INFRASTRUCTURE NATIVE WEB | OPEN SOURCE

Applause from you, Chris C Williams, and 646 others

Chris C Williams
May 12, 2017 · 7 min read

Democratizing Data at Airbnb

By Chris Williams, Eli Brumbaugh, Jeff Feng, John Bodley, and Michelle Thomas

Search all Airbnb **metrics**

Apache Atlas

bigdata REPUBLIC

Meta-data engineering: “pipeline machinery”

Thank you!

Join at slido.com: #bigdata2018

bigdata REPUBLIC

bigdata REPUBLIC

DATA SCIENCE | BIG DATA ANALYTICS | BIG DATA ARCHITECTURES

+31 (0)1 - 68479294

Coltbaan 4C, Nieuwegein

@bigdatarep

www.bigdatarepublic.nl

/company/bigdata-republic

info@bigdatarepublic.nl

bigdata REPUBLIC