DATA WAREHOUSING 101

ABOUT ME

Ezequiel Orbe

- → PhD. in Computer Science (FaMAF - UNC).
  - ◆ Research: Modal Logics and Automated Reasoning.
- → Sr. Data Engineer @ Jampp.
  - ◆ Leading the Data Infrastructure team.
  - ◆ Previously, Sr. Data Engineer @ Olapic.
- → Associate Professor @ FaMAF - UNC
  - ◆ Programming Paradigms.
  - ◆ Databases.
- → (Suffered) Fan of San Lorenzo de Almagro.
- → Frustrated basketball player.

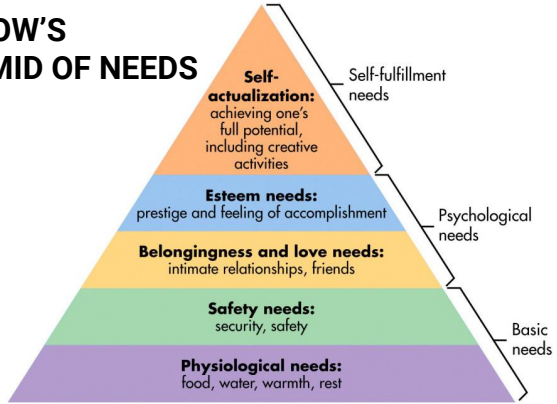
DATA WAREHOUSING 101

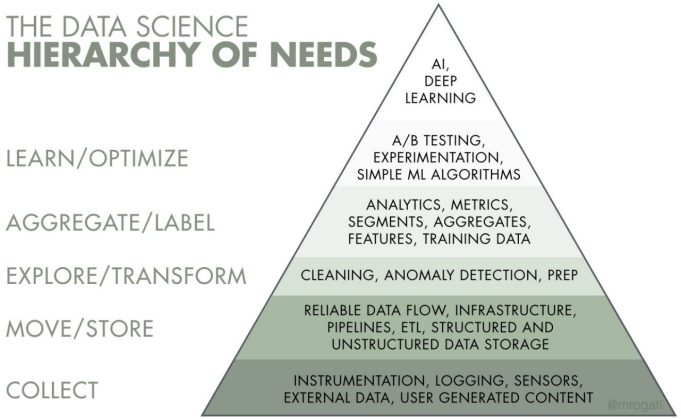
(A Survival Kit For Data Scientists)

I signed for a Data Warehousing course,
Why do I need this?

The AI Hierarchy of Needs

MASLOW'S PYRAMID OF NEEDS





"Think of Artificial Intelligence as the top of a pyramid of needs. ***Yes, self-actualization (AI) is great, but you first need food, water, and shelter (data literacy, collection, and infrastructure)"***

Monica Rogati Follow

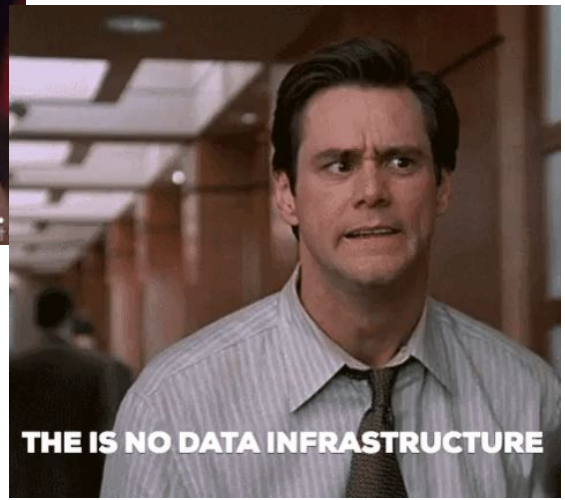
Data Science advisor. Turning data into products and stories. Former VP of Data @Jawbone & @LinkedIn data scientist. Equity partner @DCVC. CMU CS PhD.

Aug 1, 2017 · 6 min read

The AI Hierarchy of Needs

Being a Data Scientist: Expectations

Being a Data Scientist: Reality



The truth is that...

Business don't wait

Given

→ the need to differentiate/survive,
→ the pressure to hit the market first, and,
→ the lack of knowledge,

companies tend to start from the top forcing you as data scientist to be a

one man band

So, keep this in mind...

→ As a data scientist, your capability to extract value from data is tightly coupled with the maturity of the data platform of the company.

→ If the company doesn't have a mature data infrastructure in place, you'll find yourself in need of some degree of skills in data engineering.

*"The more experienced I become as a data scientist, the more convinced I am that **data engineering is one of the most critical and foundational skills in any data scientist's toolkit**. I find this to be true for both evaluating project or job opportunities and scaling one's work on the job."*

Robert Chang Follow

Data @Airbnb, previously @Twitter. Naturally opinionated, but opinions are my own

Jan 8 · 14 min read

Data Engineering Defined

What's Data Engineering all about?

*"Data engineering field could be thought of as a superset of **business intelligence** and **data warehousing** that brings more elements from **software engineering**. This discipline also integrates specialization around the operation of so called "big data" distributed systems, along with concepts around the extended Hadoop ecosystem, stream processing, and in computation at scale."*

Maxime Beauchemin
Data engineer extraordinaire at Lyft, creator of Apache Airflow and Apache Superset
Jan 20, 2017 · 12 min read

The Rise of the Data Engineer

*"Data engineering primarily focus on the following areas: **Build and maintain the organization's data pipeline systems** ...**Clean and wrangle data into a usable state**"*

James Furbush
James Furbush is a freelance technology writer who focuses on enterprise mobility and healthcare IT.
more

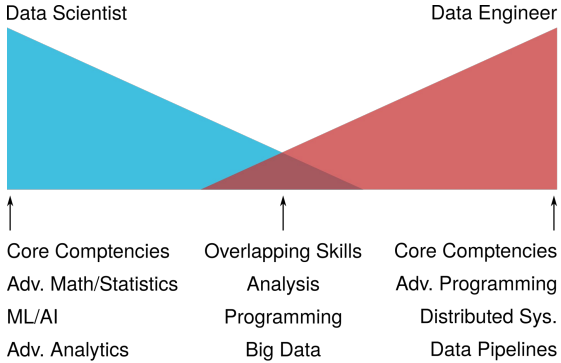
Data engineering: A quick and simple definition

Data Scientists vs Data Engineers

→ Well, Data Scientists are more glamorous..

→ A Data Engineer:

- Has a strong Software Engineering background.
- Builds tools, infrastructure, frameworks, and services.
- Operates an organization's (big) data infrastructure.



The Survival Kit

Supply #1: Some basic concepts

Different types of workloads

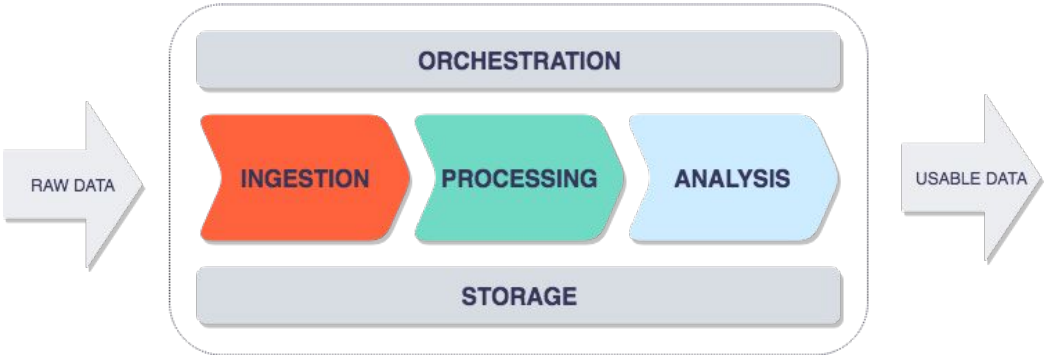
→ Generating and analyzing data are very different businesses.

	OLTP (Online Transaction Processing)	OLAP (Online Analytical Processing)
Purpose	Capture and create data.	Consolidate data and support data analysis.
Designed To	Deal efficiently with large volumes of transactions	Allow analysts to efficiently answer questions that require data from multiple source systems.
Storage Needs	Small, current snapshot of a process	Large, historical data of a process.
Examples	Operational systems: CRM, finance, etc.	Data warehouses, Data Lakes

Data Infrastructure

Data infrastructure *refers to the systems, processes and infrastructure required to collect, move, store and transform an organization's data.*

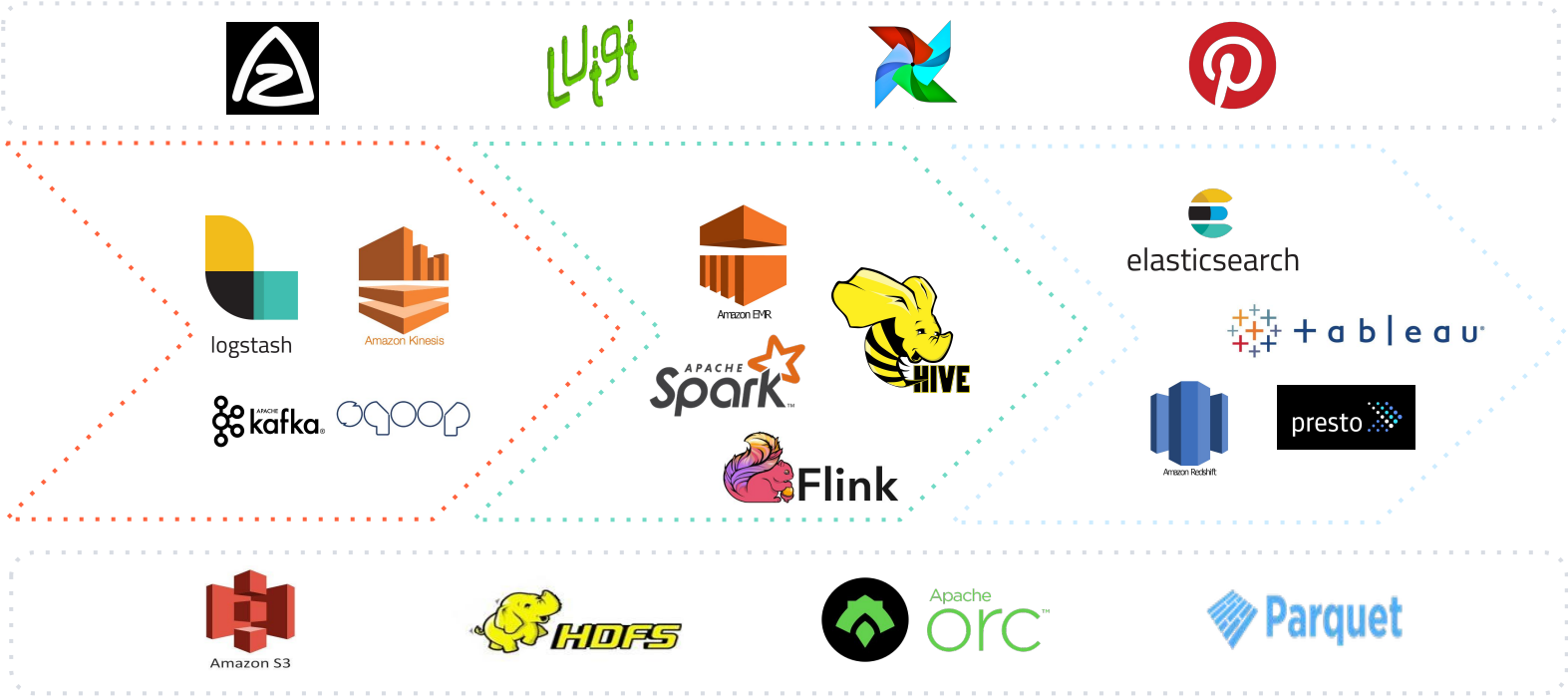
→ You could think of your DI as made of a set of layers and stages:



→ Some requirements:

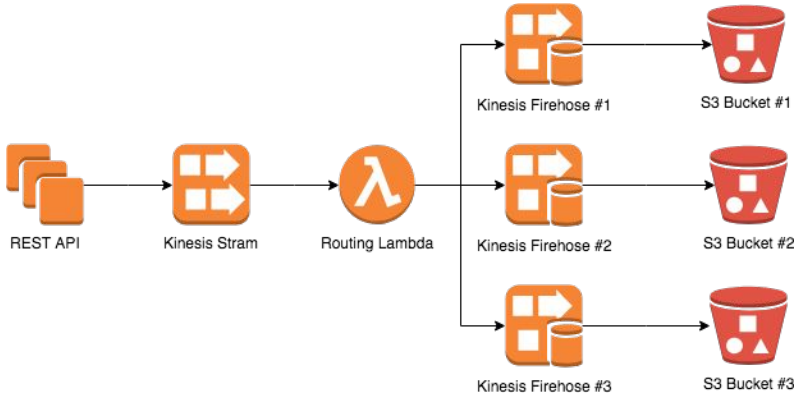
- It must make information easily **accessible**.
- It must present information **consistently**.
- It must adapt to **change**.
- It must present information in a **timely** way.
- It must **scale** with the business.
- It must be **secure**

The data ecosystem (a small part of it)

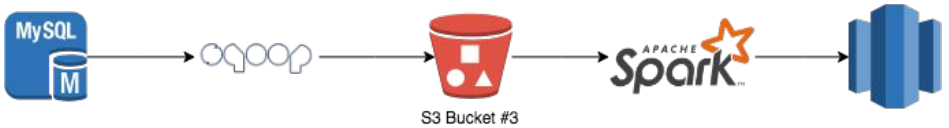


Data Pipelines

A **data pipeline** is any set of processing elements that move data from one system to another, possibly transforming the data along the way.

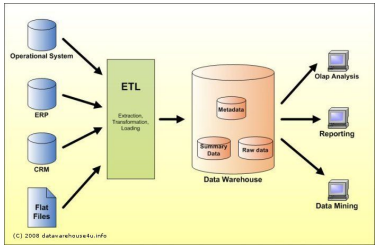


An **ETL pipeline** is a data pipeline that extracts data from one system, transforms it, and loads it into some database or data warehouse. It usually implies that the pipeline works in batches.

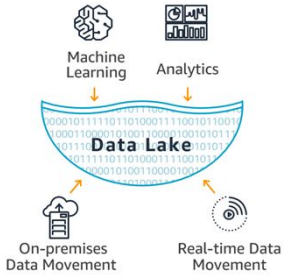


Data Warehouse vs Data Lake

A **data warehouse** is a centralized repository of integrated data from different sources specifically structured for query and analysis.



The **data lake** is a centralized repository of structured and unstructured data.



→ In a DW:

- Schema on Write.
- High quality data.
- Used mostly by business analysts.
- Used for reporting and BI.

→ In a DL:

- Schema on Read.
- Raw data.
- Used mostly by Data Scientists.
- Used for machine learning.

Supply #2: Storage Formats

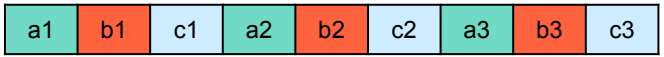
Row-Oriented vs Column-Oriented

Let's say you have the following table:

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3

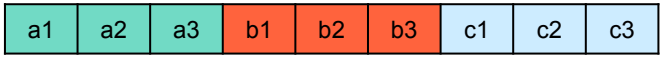
→ In a **row-oriented** storage:

- All the values from one row of a table are stored next to each other.
- Typical in most OLTP databases.



→ In a **column-oriented** storage:

- All the values from one column of a table are stored next to each other.
- Suitable for OLAP workloads.



→ Benefits of the column-oriented storage:

- Need to read less data, only files containing columns used in a query.
- Improved compression, data in a column is similar
- Faster querying processing.

Not everything is a CSV

→ If you don't use a columnar database you could still use a columnar-base file format.

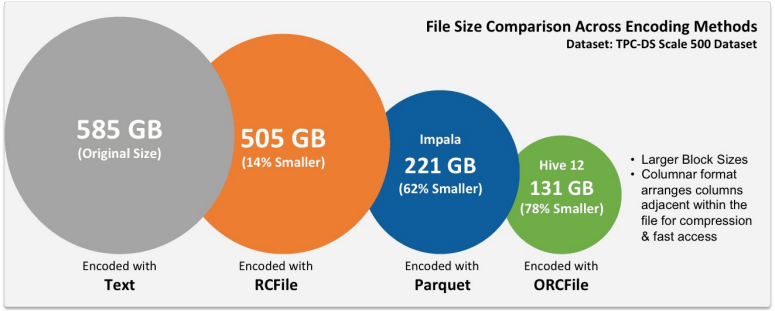
→ Apache Parquet
- Introduced by Cloudera and Twitter.
- Based on Google's Dremel format.

→ Apache ORC
- Introduced by Hortonworks.
- Supports ACID transactions in Hive.

→ Both:
- Support different types of compression: ZLIB, SNAPPY
- Complex Types (struct, maps, lists)
- Supports schema evolution.

Columnar Storage Formats: Some advices

- → Use columnar formats when you need to query your data using SQL.
- → ORC works better with Apache Hive and with more flattened structure.
- → Parquet does better with highly nested structures.
- → ORC + ZLIB tend to perform better than Parquet + Snappy compression.
- → You can create Parquet files using Apache Arrow
  - ◆ Pandas Dataframe <-> Parquet
- → You can create ORC files using ORC Java Tools
  - ◆ JSON -> ORC



Supply #3: ETLs

Toward programmatic ETLs

There are many drag-and-drop ETL Tools.

→ Pros:
- It's easier & faster to develop.
- Lots of connectors.

→ Cons:
- Generated code is not optimized.
- Difficult to debug the generated code.
- Abstractions are primitive.

Data engineers like to code their ETLs.

```
# Put upstream dependencies in a dictionary
wf_dependencies = {
    'wf_upstream_table_1': 'upstream_table_1/ds={{ ds }}',
    'wf_upstream_table_2': 'upstream_table_2/ds={{ ds }}',
    'wf_upstream_table_3': 'upstream_table_3/ds={{ ds }}',
}

# Define the sensors for upstream dependencies
for wf_task_id, partition_name in wf_dependencies.iteritems():
    NamedHivePartitionSensor(
        task_id=wf_task_id,
        partition_names=[partition_name],
        dag=dag
    )

# Put the tasks in a list
tasks = [
    ('hql', 'task_1'),
    ('hql', 'task_2'),
]

# Define the operators in the list above
for directory, task_name in tasks:
    HiveOperator(
        task_id=task_name,
```

→ Pros:
- Better abstractions.
- Ease of debugging.
- Flexibility and Performance.

→ Cons:
- ETLs are complex to code

SQL-based ETL v.s. Code-based ETL

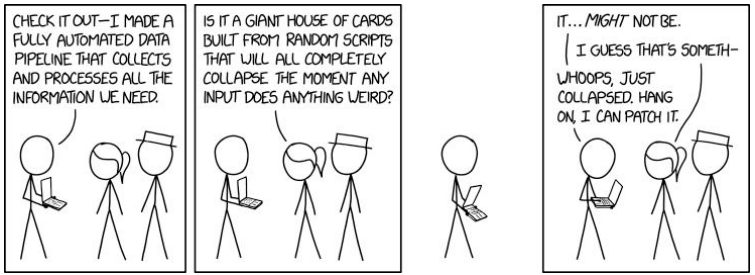
→ Code-based ETL
- Typically built in a JVM-based language (like Java or Scala) or Python.
- Jobs are defined in an imperative style.
- You have the full expressivity of the language to implement the ETL logic.
- Easier to unit test.

→ SQL-based ETL
- Typically built using one of the SQL dialects (Presto, Hive, Redshift).
- Jobs are defined in a declarative way.
- SQL expressiveness might lead to unnecessary complexity.
- Testing is more challenging.

→ An hybrid approach combining code with SQL is very common.

ETL: Some advices

- → For simple ETLs use SQL.
  - They are easier to start with.
  - You already know SQL (or at least you should).
  - Invest time on thinking about test cases for your queries.
- → Parameterize your SQL queries.
  - Use a templating engine like Jinja.
- → Once the query complexity starts to increase, switch to code.
  - Ask your designated data engineer for help.
- → Remember, your ETLs will fail (eventually).
  - Implement quality checks on your data.
  - Use staging tables.
  - Monitor your ETL metrics.
- → Enforce idempotency at each stage of the pipeline.



Supply #4: Big Data Frameworks

Don’t fool yourself...

→ Companies at different stages produce data in different velocity, variety, and volume.

- A new start-up probably don’t need “big data” because there isn’t much data.
- As the start-up grows it will be more data intensive but might do just fine using PostgreSQL.
- At certain data scale, your only choice would be to resort to the Hadoop ecosystem.

→ Don't use Hadoop - your data isn't that big

If you need to...

→ The best advice I can give you: Understand your use case.
- The match use case / technology is critical.
- Choosing the wrong technology stack will lower your quality of life...

→ Some rule-of-thumb recommendations:
- To scale your SQL-based ETLs: Hive on Tez.
- To scale your code-based ETLs: Spark.
- To scale your SQL analysis capabilities: Presto.
- A simpler, but expensier alternative: Redshift / Vertica.

→ Just remember: There is no silver bullet.

→ This is Useless (Without Use Cases)

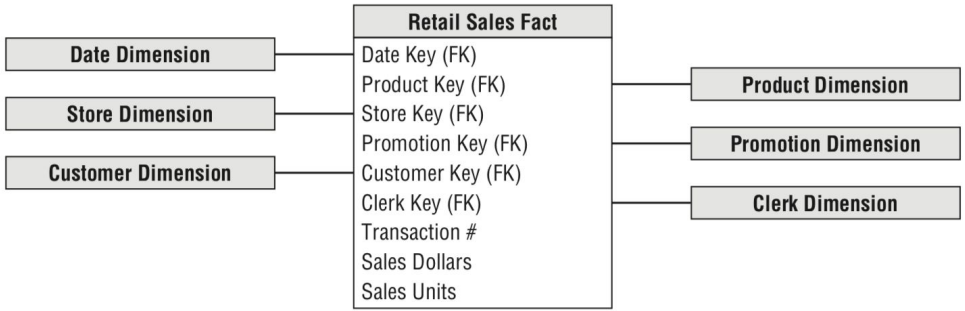
Supply #5: Data Modeling

DIMENSIONAL MODELING

→ Dimensional modeling is the preferred technique for presenting analytic data.
- Deliver data that's understandable to the business users.
- Deliver fast query performance.

→ A dimensional model is made of facts and dimension tables that model a business process.

→ Dimensional models implemented in RDBMS are referred to as star schemas.



Embracing redundancy, but not too much

- → The key difference between Third Normal Form (3NF) and dimensional models is the degree of normalization.
- → Normalized 3NF structures are useful in OLTP because an update or insert transaction touches the database in only one place.
- → Normalized 3NF models are not suitable for BI queries.
  - ◆ Most users can't understand, navigate, or remember normalized models.
  - ◆ RDBMS can't efficiently query a normalized model (not so true).
- → A dimensional model favors normalization for facts but embraces denormalization for dimensions.
- → A dimensional model packages the data in a format that delivers:
  - ◆ user understandability,
  - ◆ query performance, and
  - ◆ resilience to change.

FACT TABLE (I)

→ Stores the performance measurements (facts) resulting from a business process events.

- Each row in a fact table corresponds to a measurement event.
- *"The idea that a measurement event in the physical world has a one-to-one relationship to a single row in the corresponding fact table is a bedrock principle for dimensional modeling. Everything else builds from this foundation."* (Data Warehousing Toolkit 3rd Edition, page 11)

→ Data on each row is at a specific level of detail, referred to as the grain.

- Ex: One row per product per transaction.
- Do not mix different grains in a table.
- Atomic data that has not been aggregated is the most expressive data.

→ Fact tables express many-to-many relationships.

- The primary key is a composite key.
- Foreign keys references the dimension tables.

Translates into

Retail Sales Facts
Date Key (FK)
Product Key (FK)
Store Key (FK)
Promotion Key (FK)
Customer Key (FK)
Clerk Key (FK)
Transaction #
Sales Dollars
Sales Units

FACT TABLE (II)

- → A fact represents a business performance measure.
- → Additive Facts.
  - ◆ Can be aggregated by any dimension.
  - ◆ Ex: sales amount.
- → Semi-additive facts.
  - ◆ Cannot be summed across the time dimension.
  - ◆ Ex. account balances.
- → Non-additive facts.
  - ◆ Can never be added.
  - ◆ Ex. Unit price.
- → Do not store redundant textual information in fact tables.
- → Fact tables are the largest datasets.
  - ◆ Deep in terms of the number of rows, but narrow in terms of the number of columns.

DIMENSIONS (I)

- → Dimensions contain the textual context associated with a fact.
  - They describe the "who, what, where, when, how, and why".
- → Dimension attributes are critical to making the DW usable and understandable.
- → Dimension attributes are the primary source of
  - Query constraints.
  - Groupings.
  - Report labels.
- → Each dimension table is defined by a single surrogate key.
  - Tend to have fewer rows than fact tables.
  - But can be wide with many large text columns
- → Dimension tables are denormalized.

Product Dimension
Product Key (PK)
SKU Number (Natural Key)
Product Description
Brand Name
Category Name
Department Name
Package Type
Package Size
Abrasive Indicator
Weight
Weight Unit of Measure
Storage Type
Shelf Life Type
Shelf Width
Shelf Height
Shelf Depth
...

Key Design Decisions

→ Select the business process.
- ◆ It defines a specific design target and allows the grain, dimensions, and facts to be declared
- ◆ Most fact tables focus on the results of a single business process.

→ Declare the grain.
- ◆ The grain establishes exactly what a single fact table row represents.
- ◆ Must be declared before choosing dimensions or facts.
- ◆ Always prefer atomic-grained data.
- ◆ Do not mix different grains in the same fact table.

→ Identify the dimensions.
- You need to have conformed dimensions.

→ Identify the facts.

Modern Data Warehousing

→ Further denormalization is preferred.

- Maintaining surrogate keys in dimensions is tricky, and it makes fact tables less readable.
- Usage of natural, human readable keys and dimension attributes in fact tables.
- Reducing the need for costly joins that can be heavy on distributed databases.
- Encoding and Compression in serialization formats like Parquet or ORC, or in database engines like Vertica, address most of the performance loss that would normally be associated with denormalization.

→ Grain Mixing is more tolerated.

- Improved support for JSON enables to store store multiple grains at once.

→ Shorter development iterations are possible.

- With the improved support for JSON, you can evolve your schema without executing DDL..
- You don’t need buy-in prior to development.

Modern Data Warehousing (cont.)

→ Handling slowly changing dimensions (SCD) is simplified.

- Systematically snapshotting dimensions is a simple generic approach.
- Fact tables denormalization it's easier for tracking a dimension attribute value at the moment of the transaction.
- SCD modeling techniques are not intuitive and complex to implement.

→ Conformance started happening as-we-go.

- Conformed dimensions are still extremely important, but it's less imperative and more of a tradeoff to have them upfront.
- Consensus and convergence can happen as a background process in the areas where the pain point of divergence become out-of-hand.

Supply #6: Airflow

Introducing Airflow

→ Apache Airflow is a platform to programmatically author, schedule and monitor workflows.

→ It's a great option to implement ETL pipelines or more generic data pipelines.

→ It has a great community and it's one of the more popular platforms.

→ It provides configuration as code, monitoring, alerts, scheduling and backfilling.

→ Similar tools:

- Azkaban (Linkedin)
- Luigi (Spotify)
- Pinball (Pinterest)

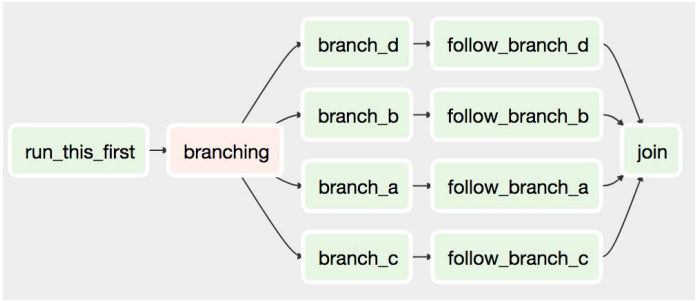
Workflows in Airflow

→ Workflows are defined as directed acyclic graphs (DAGs) defined in code.

→ Each node in the DAG is a operator and the relations between nodes represents the dependencies between operators.

→ DAGs describe how to run a data pipeline, operators describe what to do in a data pipeline.

```python
import airflow
from airflow.operators.python_operator import BranchPythonOperator
from airflow.operators.dummy_operator import DummyOperator
from airflow.models import DAG
import random


args = {
    'owner': 'airflow',
    'start_date': airflow.utils.dates.days_ago(2)
}

dag = DAG(
    dag_id='example_branch_operator',
    default_args=args,
    schedule_interval="@daily")

cmd = 'ls -l'
run_this_first = DummyOperator(task_id='run_this_first', dag=dag)

options = ['branch_a', 'branch_b', 'branch_c', 'branch_d']

branching = BranchPythonOperator(
    task_id='branching',
    python_callable=lambda: random.choice(options),
    dag=dag)
branching.set_upstream(run_this_first)

join = DummyOperator(
    task_id='join',
    trigger_rule='one_success',
    dag=dag
)

for option in options:
    t = DummyOperator(task_id=option, dag=dag)
    t.set_upstream(branching)
    dummy_follow = DummyOperator(task_id='follow_' + option, dag=dag)
    t.set_downstream(dummy_follow)
    dummy_follow.set_downstream(join)
```



Some available operators

Airflow provides a number of operators, including:

→ BashOperator (executes a bash command)
→ PythonOperator (calls an arbitrary Python function)
→ EmailOperator (sends an email)
→ SimpleHttpOperator (sends an HTTP request)
→ MySqlOperator/PostgresOperator/HiveOperator (executes a SQL command)
→ DockerOperator, SlackOperator… you get the idea!

It also provides sensors that waits for a condition to happen:

→ FileSensor, HttpSensor, HivePartitionSensor, etc.

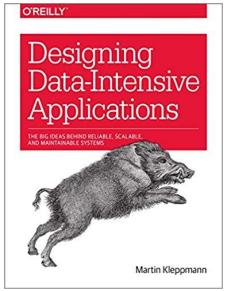
Demo Time..

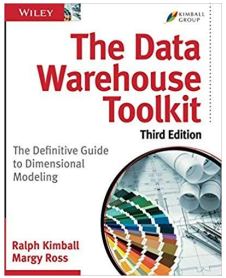
Reading Material

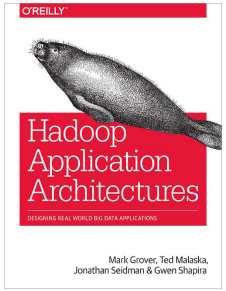
→ Interesting blog posts:

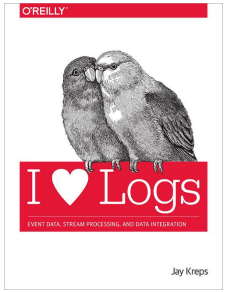
- A beginners guide to data engineering - Part I
- A beginners guide to data engineering - Part II
- A beginners guide to data engineering - Series Finale
- The downfall of the data engineer

→ Recommended Books:









Related books


I'M DONE