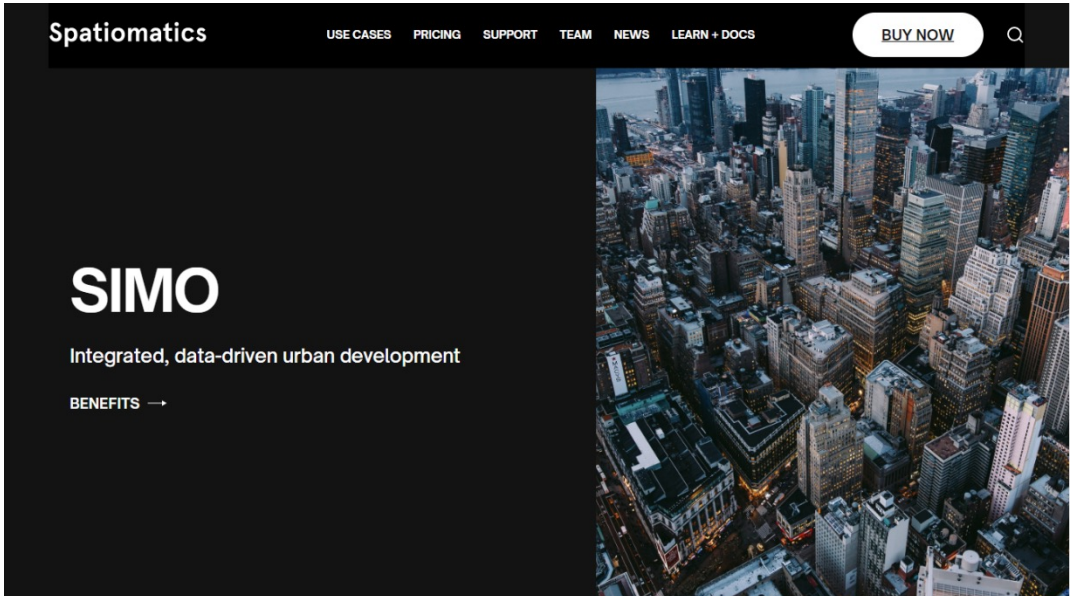
Darrel Ronald – Spatiomatics

AMS Institute – 2023.09.20
Data Engineering – QuickStart Introduction

https://www.linkedin.com/in/darrelronald/
www.Spatiomatics.com

Experience



Pre-2020

Architect, Urban Designer
Computational Designer
Teaching TU Delft (Architecture)

Co-founder: **Open Form Architecture**
Worked at: **KCAP, MVRDV, Maxwan, TSPA**

Post-2020

Software Developer (Python, .NET)
Executive MBA (IMD, Lausanne)
Teaching TU Delft (Urbanism)

Founder: **Spatiomatics**
Product Manager: **SIMO app**

Lecture Focus

This lecture aims to satisfy new and experienced programmers
We do not focus on Big Data Streaming, Storage and Processing

1. **What is Data**
2. **Project Lifecycle and Data Engineering**
3. **Python for Data Engineering**
4. **Code and Data Quality**
5. **Data Models and Schemas**
6. **Databases and Languages**
7. **Tools and Sources**
8. **Pipelines**
9. **Privacy and Governance**

Red = Key Takeaways
No Generative AI was used

Learning Resources

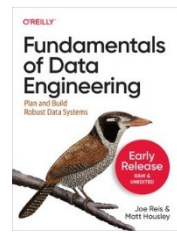
Resources

Books
- Fundamentals of Data Engineering
- Designing Data Intensive Applications
- Data-Oriented Programming
- Learning SQL
- Best Python Books – Real Python
- Effective Computation in Physics

Websites / Courses
- Real Python
- Learn Data Engineering
- Datacamp
- Full Stack Python
- Udemy
  - Jose Portilla
  - Kirill Eremenko
  - Frank Kane

Newsletters / YouTube / Podcasts
- Seattle Data Guy - Youtube
- Plumbers of Data Science
- Kahan Data Solutions
- Data Engineering Central

Urban Computing / Geospatial
- Spatial Thoughts
- Qiusheng Wu
- Anita Graeser
- Microsoft Urban Computing (Paper)



Intro to Data

The most important question

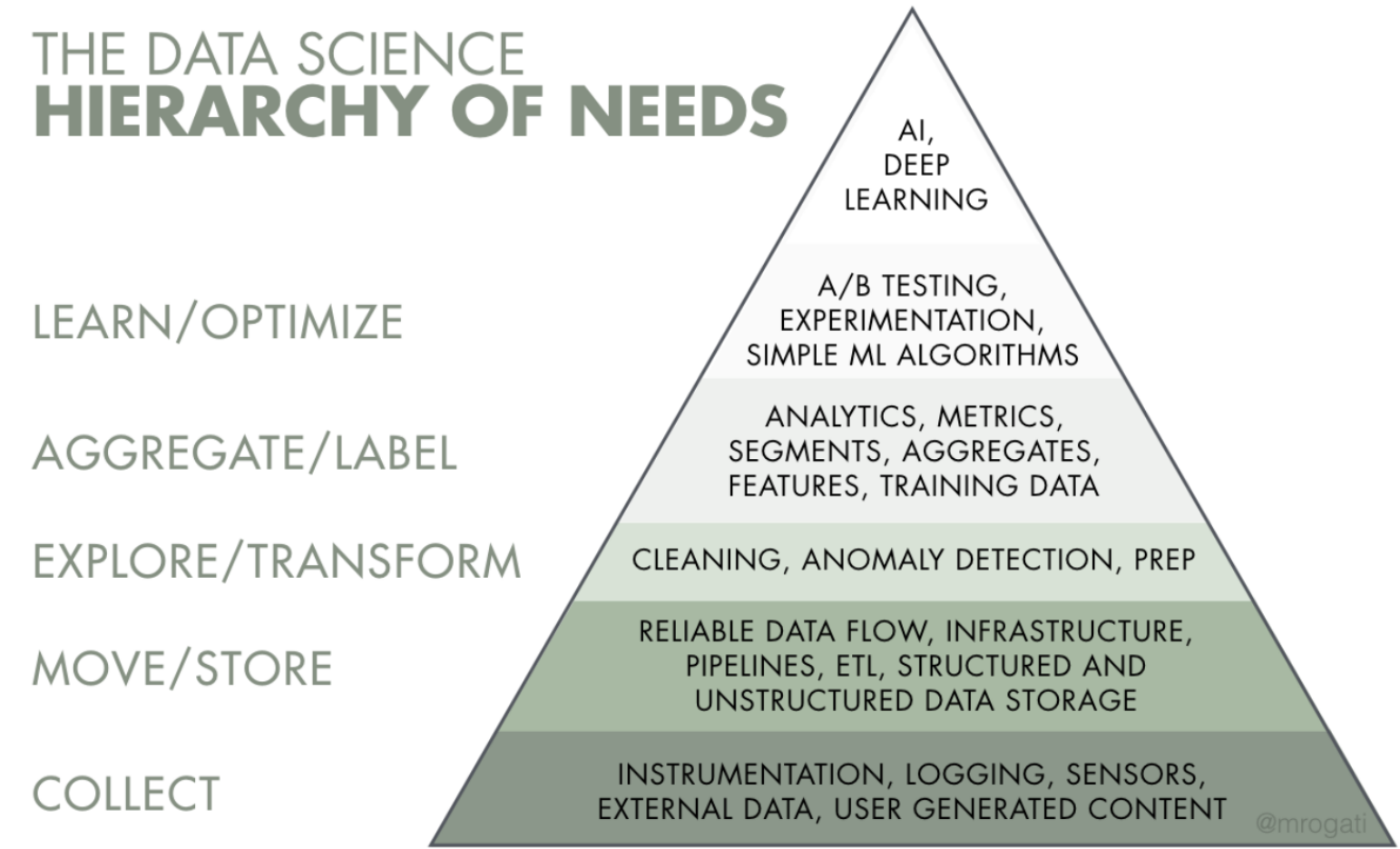
Why?

Data is slow → know precisely why you need it
Data is endless → filter as precisely as you can
Data is confusing → but it creates the illusion of accuracy
Data is hard → it requires investigation, validation, updating

How

- State your **Research Question**
- Design the **workflow** first
- Develop a **proof of concept**, then improve
- **Document** everything you do

Data Science



Your Data is Small Data

Big Data is typically defined by 6 characteristics:

1. **Volume** – massive datasets (ex. Petabytes)
2. **Variety** – diverse types (ex. Structured, unstructured, blob)
3. **Velocity** – continuous high volume (ex. Every second, minute)
4. **Veracity** – high quality
5. **Value** – more value is derived by massive datasets
6. **Variability** – dynamic sources, formats, processes

Data Maxims

Data either exists, or it doesn't.
If it doesn't exist, find a proxy. 🎧

Data is either  **or** 💩
Separate the good from the bad

More data isn't always better < = >
Focus on quality first, then quantity...
Especially for training ML models!

Data Project Lifecycle, Team & Roles

Data Science Project Lifecycle

Cross-Industry Standard Process for Data Mining (CRISP-DM)

6 Phases

1. Business Understanding
2. Data Understanding
3. Data Preparation
4. Modeling
5. Evaluation
6. Deployment

Data Science Project Lifecycle
Cross-industry standard process for data mining

Northeastern University

LIFE CYCLE OF A DATA ANALYSIS PROJECT

Based on CRISP-DM Methodology

Business Issue Understanding

- Define business objectives
- Gather required information
- Determine appropriate analysis method
- Clarify scope of work
- Identify deliverables

Data Understanding

- Collect initial data
- Identify data requirements
- Determine data availability
- Explore data and characteristics

Data Preparation

- Gather data from multiple sources
- Cleanse
- Format
- Blend
- Sample

Exploratory Analysis and Modeling

- Develop methodology
- Determine important variables
- Build model
- Assess model

Validation

- Evaluate results
- Review process
- Determine next steps
- Results are valid → proceed to step 6
- Results are invalid ← revisit steps 1-4

Visualization and Presentation

- Communicate results
- Determine best method to present insights based on analysis and audience
- Craft a compelling story
- Make recommendations

Reference Problem Solving with Advanced Analytics

Reference https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_mining

Lifecycle of a Data Analysis Project

Data Team & Roles

Chief Data Officer
Leadership level, defines strategy and business value

Data Engineer
Creates and manages the data plumbing

Data Consumers

Data consumers use data to make data-driven decisions, and actively have informed conversations with data practitioners.

Business Analysts

Business Analysts are responsible for tying data insights to actionable results that increase profitability or efficiency. They have deep knowledge of the business domain and often use SQL alongside non-coding tools to communicate insights derived from data.

Data Analysts

Similar to Business Analysts, Data Analysts are responsible for analyzing data and reporting insights from their analysis. They have a deep understanding of the data analysis workflow and report their insights through a combination of coding and non-coding tools.

Machine Learning Scientists

Machine Learning Scientists design and deploy machine learning systems that make predictions from the organization's data. They solve problems like predicting customer churn and lifetime value and are responsible for deploying models for the organization to use. They work exclusively with coding-based tools.

Statisticians

Similar to Data Scientists, Statisticians work on highly rigorous analysis, which involves designing and maintaining experiments such as A/B tests and hypothesis testing. They focus on quantifying uncertainty and presenting findings that require exceptional degrees of rigor, like in finance or healthcare

Data Scientists

Data Scientists investigate, extract, and report meaningful insights in the organization's data. They communicate these insights to nontechnical stakeholders and have a good understanding of machine learning workflows and how to tie them back to business applications. They work almost exclusively with coding tools, conduct analysis, and often work with big data tools

Data Engineers

Data Engineers are responsible for getting the right data in the hands of the right people. They create and maintain the infrastructure and data pipelines that take terabytes of raw data coming from different sources into one centralized location with clean, relevant data for the organization.

Programmers

Programmers are highly technical individuals that work on data teams and work on automating repetitive tasks when accessing and working with an organization's data. They bridge the gap between traditional software engineering and data science and have a thorough understanding of deploying and sharing code at scale.

Source: Data Team

Data Types, Structures & Collections in Python

Python Built-in Types (partial list)

```
# Boolean
True
False

# Null Object
None

# Comparison Operators
<
<=
>
>=
==
!=
is
is not
```

```
# Numeric Types
int(1)
float(1.0000)

# Sequence Types
list = [1, 2, 3]
tuple = (1, 2, 3)
range = range(0, 10, 1)
text = 'string text'

# Set Types
set = set([1, 2, 3, 3, 3, 4])
set = {'amsterdam', 'metropolitan', 'solutions'}

# Mapping Types
dictionary = dict(one=1, two=2, three=3)
dictionary = {'one':1, 'two':2, 'three':3}
```

Built-in Types — Python 3.11.5
5. Data Structures — Python 3.11.5

Special Python Data Types & Modules (partial list)

```
# Datetime Module & ISO 8601
import datetime
datetime.date(2023, 1, 31)
datetime.time(1, 20, 30)
datetime.date.today()
datetime.now()

# Collections - Alternatives to
dict, list, set, tuple
import collections
collections.namedtuple()
collections.deque()
collections.ChainMap()
collections.Counter()
collections.OrderedDict()
```

```
# Enumerations
from enum import Enum
class Road_types(Enum):
    HIGHWAY = 1
    REGIONAL = 2
    LOCAL = 3
    SHARED = 4
```

Data Types — Python 3.11.5
Using Python datetime – Real Python
collections — Python 3.11.5
enum — Python 3.11.5

@dataclass in Python 3.7+

```python
from dataclasses import dataclass

@dataclass
class InventoryItem:
    """Class for keeping track of an item in inventory."""
    name: str
    unit_price: float
    quantity_on_hand: int = 0

    def total_cost(self) -> float:
        return self.unit_price * self.quantity_on_hand
```

Since Python 3.7 you can create classes with the **@dataclass** Decorator. You can strongly **type** the individual properties within and also set default values. Internal **Methods** are optional.

dataclasses — Data Classes — Python 3.11.5
Data Classes in Python 3.7+
Data-Oriented Programming

Code and Data Quality

Write Pythonic Code

```
# Zen of Python – PEP 20
import this

'''
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
'''
```

Python should follow the the **PEP 8** style guide and the PEP 20 **Zen of Python**.

Popular Python Linters

- Sonar
- Pylint
- Pyflakes
- Flake8

PEP 8 – Style Guide for Python Code
PEP 20 – The Zen of Python
Python Code Quality
Python Best Practices
Python Cheat Sheets - DataCamp

Testing you Python Code

```
import unittest

class TestStringMethods(unittest.TestCase):

        def test_upper(self):
                self.assertEqual('foo'.upper(), 'FOO')

        def test_isupper(self):
                self.assertTrue('FOO'.isupper())
                self.assertFalse('Foo'.isupper())

        def test_split(self):
                s = 'hello world'
                self.assertEqual(s.split(), ['hello', 'world'])

                # check that s.split fails when the separator
                is not a string
                with self.assertRaises(TypeError):
                    s.split(2)

if __name__ == '__main__':
    unittest.main()
```

Testing your code is essential for production work. But you can skip it for now as you learn

Popular Python Libraries

- unittest – standard library
- pytest

unittest — Python 3.11.5
Getting Started With Testing in Python

Current International Standards

Many Applicable and Interlinked Standards

~ 21580 ISO, International Standards Organization
~ 275 ECMA, European Computer Manufacturers Association
~ 275 W3C, Worldwide Web Consortium (*recommendations*)
~ ? IETF, Internet Engineering Task Force
~ 30 OGC, Open Geospatial Consortium

Use international standards for your data where possible

Examples

- UUID4 for Unique Identifiers – Python UUID module
- ISO 8601 for Time Stamps

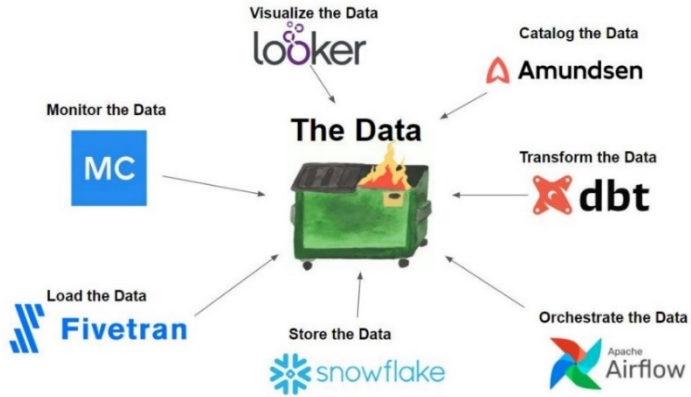
Sources of Data Problems

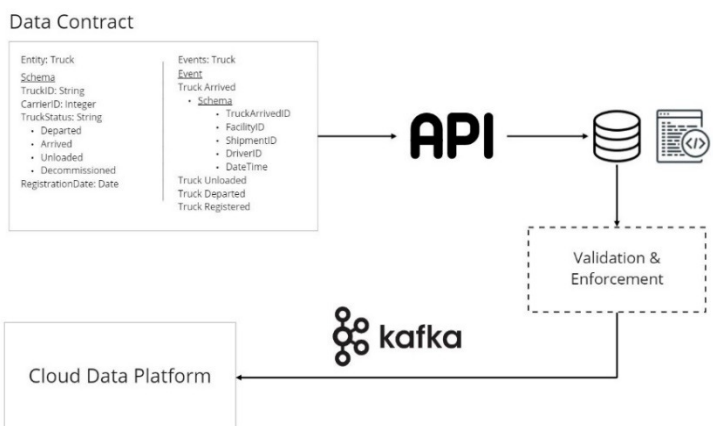
Common Errors
- Input Errors ❌
- Wrong Formatting ❌
- Transformation Errors ❌
- Duplicate Data ❌
- Lack of standards ❌
- Outdated Information ❌
- Proprietary data standards ❌

Missing Metadata
- Source: Device, Location ❓
- Owner: License, Copyright ❓
- Origin: Date, Time, Location ❓

Internationalization
- Language 🈲
- Date & Time Methods ♾️
- Localized Standards (ex. Street classification) 🗺️

The Rise of Data Contracts...





With the exponential increase in data across organizations, many teams have lost control of large systems and big data.

GIGO Cycle = Garbage In, Garbage out

1. Databases are treated as nonconsensual APIs
2. With no contract in place, databases can change at any time
3. Producers have no idea how their data is being used downstream
4. Cloud platforms (Snowflake) are not treated as production systems
5. Datasets break as changes are made upstream
6. Data Engineers inevitably must step in to fix the mess
7. Data Engineers begin getting treated as middle-men
8. Technical debt builds up rapidly - a refactor is the only way out
9. Teams argue for better ownership and a 'single throat to choke'
10. Critical Production systems in the cloud (ML/Finance) fail
11. Blatant Sev1's impact the bottom line, while invisible errors go undetected
12. The data becomes untrustworthy
13. Big corporations begin throwing people at the problem
14. Everyone else faces an endless up-hill battle

[The Rise of Data Contracts - Chad Sanderson]

Data Validation

```python
from datetime import datetime
from pydantic import BaseModel, PositiveInt


class User(BaseModel):
    id: int
    name: str = 'John Doe'
    signup_ts: datetime | None
    tastes: dict[str, PositiveInt]

external_data = {
    'id': 123,
    'signup_ts': '2019-06-01 12:22',
    'tastes': {
        'wine': 9,
        'cheese': 7,
        'cabbage': '1',
    },
}

user = User(**external_data)

print(user.id)
#> 123
```

```python
print(user.model_dump())

"""
{
    'id': 123,
    'name': 'John Doe',
    'signup_ts':
datetime.datetime(2019, 6, 1, 12,
22),
    'tastes': {'wine': 9, 'cheese':
7, 'cabbage': 1},
}
"""
```

Pydantic is a popular Python validation library.

Your class object inherits from the Pydantic **BaseModel** and applies strict type checking.

Pydantic
Pydantic Docs

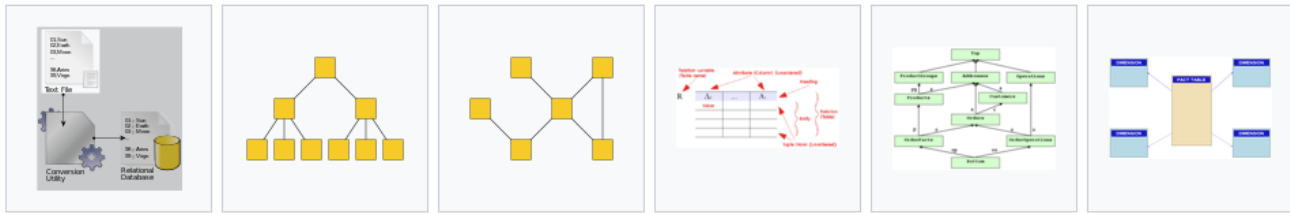
Data Models & Schemas
The “Blueprints”

Data Model & Database Schema

The **Data Model** is an abstract description of the relationship of data elements. It applies to both software design and database design and both organizes and standardizes the **relationships** and **properties** of each real-world **entities**.

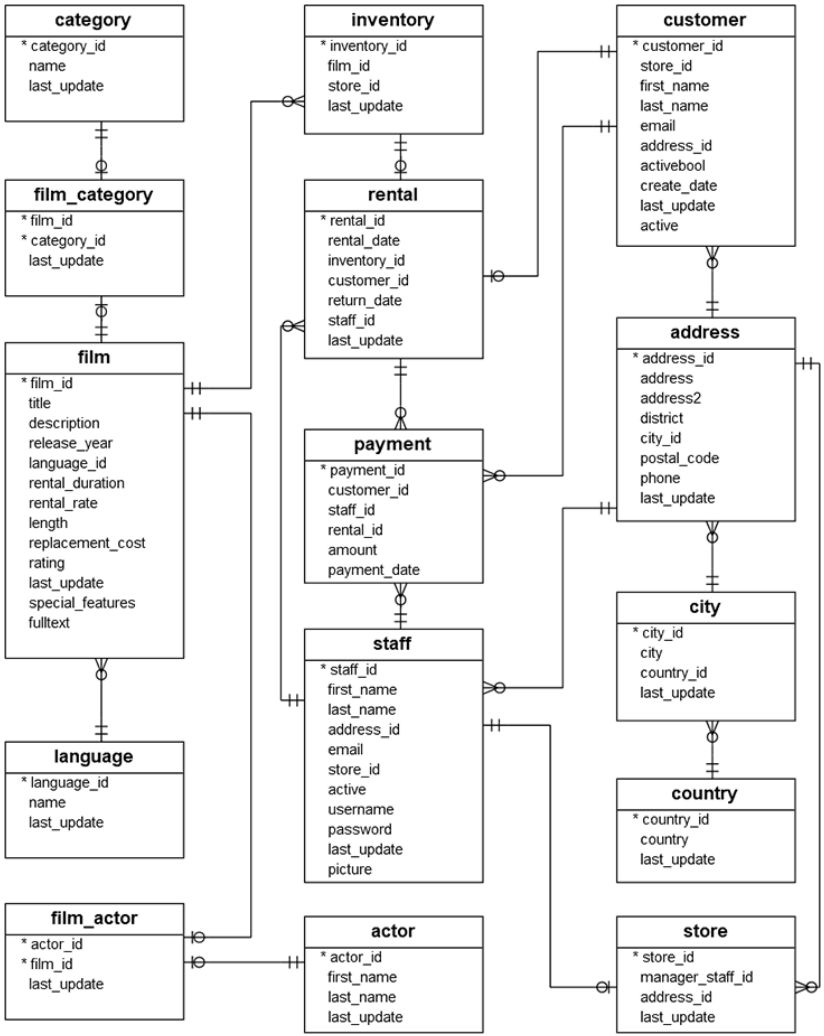
Database Models

- Flat Model – simple **arrays** of data (ex. CSV, TXT, TSV)
- Hierarchical Model – **tree**-like structure (ex. GIS data, XML database)
- Network Model – **graph** data relationship (ex. RDF, NEO4j database)
- Relational Model – most common for **relational** databases (ex. SQL databases)
- Object-Relational Model – less common and for **object-oriented** programming
- Object-Role Model – less common and for business logic
- Star Schema – for data warehouses



Data model
What is Data Modeling?

Example: DVD Rental Entity-Relationship Model



PostgreSQL Sample Database Tables
Has 15 tables in the DVD Rental database:

- **actor** – stores actors data including first name and last name.
- **film** – stores film data such as title, release year, length, rating, etc.
- **film_actor** – stores the relationships between films and actors.
- **category** – stores film's categories data.
- **film_category** - stores the relationships between films and categories.
- **store** – contains the store data including manager staff and address.
- **inventory** – stores inventory data.
- **rental** – stores rental data.
- **payment** – stores customer's payments.
- **staff** – stores staff data.
- **customer** – stores customer data.
- **address** – stores address data for staff and customers
- **city** – stores city names.
- **country** – stores country names.

PostgreSQL Sample Database

Data Schemas

Data Schemas are important but can be overwhelming at first.
Start with simple tutorials first. All software and web APIs (**Application Programming Interfaces**) uses a data schema to ensure safe machine-readable communication.

Common Schemas

- JSON – standard format for web APIs
- JSON-LD – linked web data
- GeoJSON – geospatial data
- XML – early web data
- XML/RDF – graph data relationships
- GML – geospatial data
- HTML – standard web page structure

Tutorials

- JSON Introduction and Understanding JSON Schema
- JSON-LD Introduction
- XML Introduction
- XML RDF Introduction
- HTML Introduction

JSON Schema

```
{
  "type": "object",
  "properties": {
    "first_name": { "type": "string" },
    "last_name": { "type": "string" },
    "birthday": { "type": "string", "format": "date" },
    "address": {
      "type": "object",
      "properties": {
        "street_address": { "type": "string" },
        "city": { "type": "string" },
        "state": { "type": "string" },
        "country": { "type" : "string" }
      }
    }
  }
}
```

By “validating” the first example against this schema, you can see that it fails:

```
{
  "name": "George Washington",
  "birthday": "February 22, 1732",
  "address": "Mount Vernon, Virginia, United States"
}
```

However, the second example passes:

```
{
  "first_name": "George",
  "last_name": "Washington",
  "birthday": "1732-02-22",
  "address": {
    "street_address": "3200 Mount Vernon Memorial Highway",
    "city": "Mount Vernon",
    "state": "Virginia",
    "country": "United States"
  }
}
```

[Understanding JSON Schema]

JSON-LD Schema

```
<script type="application/ld+json">{
  "@context": "http://schema.org",
  "@type": "Movie",
  "name": "Movie Name",
  "dateCreated": "17 May, 1980",
  "director": "Donald Trumph",
  "actors": ["ActorOne", "ActorTwo", "ActorThree"],
  "image": "http://example.com/image/movie.pgn",
  "countryOfOrigin": "America",
  "duration": "2:10:20",
  "musicBy": "Awesome Band",
  "productionCompany": "Awesome Movies inc.",
  "subtitleLanguage": "en, nb",
  "trailer": {
    "@type": "VideoObject",
    "name": "Official Trailer WAtW!",
    "description": "Trailer for the awesome new movieWigs Around the World!",
    "thumbnailUrl": "http://examples.com/thumbnail.png",
    "uploadDate": "10 May 1980"
  }
}</script>
```

Movie — 0 ERRORS 0 WARNINGS

@type	Movie
name	Wigs Around the World
dateCreated	1980-05-17
image	http://example.com/image/movie.pgn
duration	2:10:20
subtitleLanguage	en, nb
director	
@type	Person
name	Donald Trumph
actors	
@type	Person
name	ActorOne
actors	
@type	Person
name	ActorTwo
actors	
@type	Person
name	ActorThree
countryOfOrigin	
@type	Country
name	America
musicBy	
@type	Thing
name	Awesome Band
productionCompany	
@type	Organization
name	Awesome Movies inc.
trailer	
@type	VideoObject
name	Official Trailer WAtW!
description	Trailer for the awesome new movieWigs Around the World!
thumbnailUrl	http://examples.com/thumbnail.png
uploadDate	1980-05-10

[Movie - JSON-LD markup]
[Movie - Schema.org Type]

XML Schema

Example
The Purchase Order Schema, po.xsd

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:annotation>
    <xsd:documentation xml:lang="en">
     Purchase order schema for Example.com.
     Copyright 2000 Example.com. All rights reserved.
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>

  <xsd:element name="comment" type="xsd:string"/>

  <xsd:complexType name="PurchaseOrderType">
    <xsd:sequence>
      <xsd:element name="shipTo" type="USAddress"/>
      <xsd:element name="billTo" type="USAddress"/>
      <xsd:element ref="comment" minOccurs="0"/>
      <xsd:element name="items"  type="Items"/>
    </xsd:sequence>
    <xsd:attribute name="orderDate" type="xsd:date"/>
  </xsd:complexType>

  <xsd:complexType name="USAddress">
    <xsd:sequence>
      <xsd:element name="name"   type="xsd:string"/>
      <xsd:element name="street" type="xsd:string"/>
      <xsd:element name="city"   type="xsd:string"/>
      <xsd:element name="state"  type="xsd:string"/>
      <xsd:element name="zip"    type="xsd:decimal"/>
    </xsd:sequence>
    <xsd:attribute name="country" type="xsd:NMTOKEN"
                   fixed="US"/>
  </xsd:complexType>

  <xsd:complexType name="Items">
    <xsd:sequence>
      <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="productName" type="xsd:string"/>
            <xsd:element name="quantity">
              <xsd:simpleType>
                <xsd:restriction base="xsd:positiveInteger">
                  <xsd:maxExclusive value="100"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:element>
            <xsd:element name="USPrice"  type="xsd:decimal"/>
            <xsd:element ref="comment"   minOccurs="0"/>
            <xsd:element name="shipDate" type="xsd:date" minOccurs="0"/>
          </xsd:sequence>
          <xsd:attribute name="partNum" type="SKU" use="required"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <!-- Stock Keeping Unit, a code for identifying products -->
  <xsd:simpleType name="SKU">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{3}-[A-Z]{2}"/>
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

Example
The Purchase Order, po.xml

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
   <shipTo country="US">
      <name>Alice Smith</name>
      <street>123 Maple Street</street>
      <city>Mill Valley</city>
      <state>CA</state>
      <zip>90952</zip>
   </shipTo>
   <billTo country="US">
      <name>Robert Smith</name>
      <street>8 Oak Avenue</street>
      <city>Old Town</city>
      <state>PA</state>
      <zip>95819</zip>
   </billTo>
   <comment>Hurry, my lawn is going wild<!/comment>
   <items>
      <item partNum="872-AA">
         <productName>Lawnmower</productName>
         <quantity>1</quantity>
         <USPrice>148.95</USPrice>
         <comment>Confirm this is electric</comment>
      </item>
      <item partNum="926-AA">
         <productName>Baby Monitor</productName>
         <quantity>1</quantity>
         <USPrice>39.98</USPrice>
         <shipDate>1999-05-21</shipDate>
      </item>
   </items>
</purchaseOrder>
```

[XML Schema Part 0: Primer Second Edition]

XML-RDF Schema

Triples of the Data Model

Subject	Predicate	Object
https://www.w3schools.com	https://www.w3schools.com/rdf/title	"W3Schools.com"
https://www.w3schools.com	https://www.w3schools.com/rdf/author	"Jan Egil Refsnes"

The original RDF/XML document

```
1: <?xml version="1.0"?>
2: <rdf:RDF
3: xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4: xmlns:si="https://www.w3schools.com/rdf/">
5: <rdf:Description rdf:about="https://www.w3schools.com">
6: <si:title>W3Schools.com</si:title>
7: <si:author>Jan Egil Refsnes</si:author>
8: </rdf:Description>
9: </rdf:RDF>
```

Graph of the data model


```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:si="https://www.w3schools.com/rdf/">
<rdf:Description rdf:about="https://www.w3schools.com">
  <si:title>W3Schools.com</si:title>
  <si:author>Jan Egil Refsnes</si:author>
</rdf:Description>
</rdf:RDF>
```

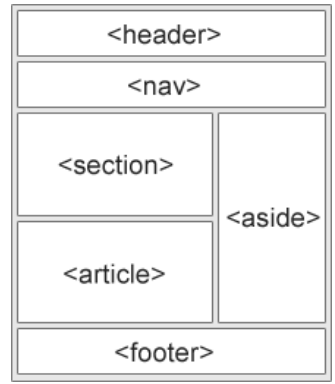
XML RDF
Triples of the Data Model

HTML Schema

Basic Web Page Structure

```
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>
  </body>
</html>
```

Semantic Elements



<article>
<aside>
<details>
<figcaption>
<figure>
<footer>
<header>
<main>
<mark>
<nav>
<section>
<summary>
<time>

Understanding HTML is important when scraping web data. Popular python libraries for this include Beautiful Soup and Scrapy.

HTML 4 Document Type Definition

Databases

Overview of Databases

Key Concept

- Roughly **340** types of databases
- Databases organize and store data
- Storage can be **Persistent** or **Ephemeral** (in-memory, volatile)
- Each database type has its own type of Database Management System and **Language** (ex. SQL, NoSQL)

Key Functionality

- Administration
- Schema, Data Definition
- Update with **CRUD Operations** = Create, Read, Update, Delete
- Retrieval (ex. API access)

What Are Databases?
Database of Databases

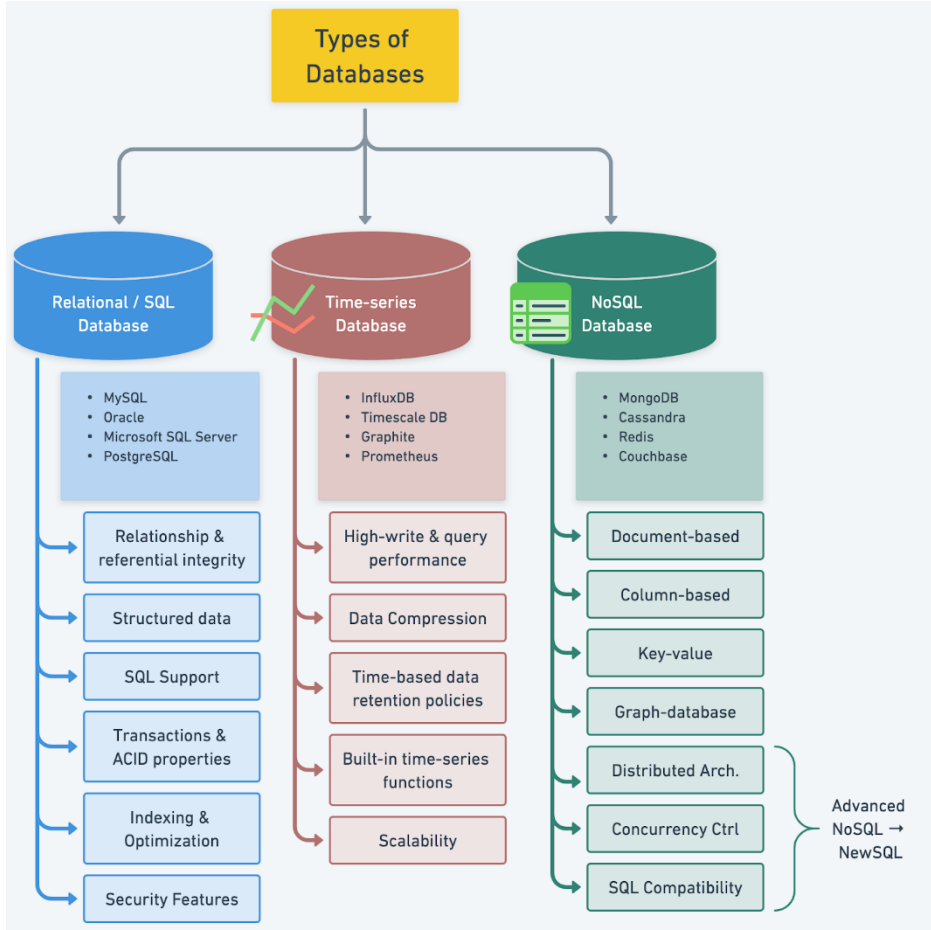
Types of Databases

Main Types of Databases

- Flat File (ex. CSV)
- SQL (Relational)
- NoSQL
- NewSQL
- Timeseries
- Vector (for ML)
- Serverless

Most Common

- Oracle
- MySQL
- Microsoft SQL Server
- PostgreSQL
- MongoDB
- SQLite
- CSV, JSON

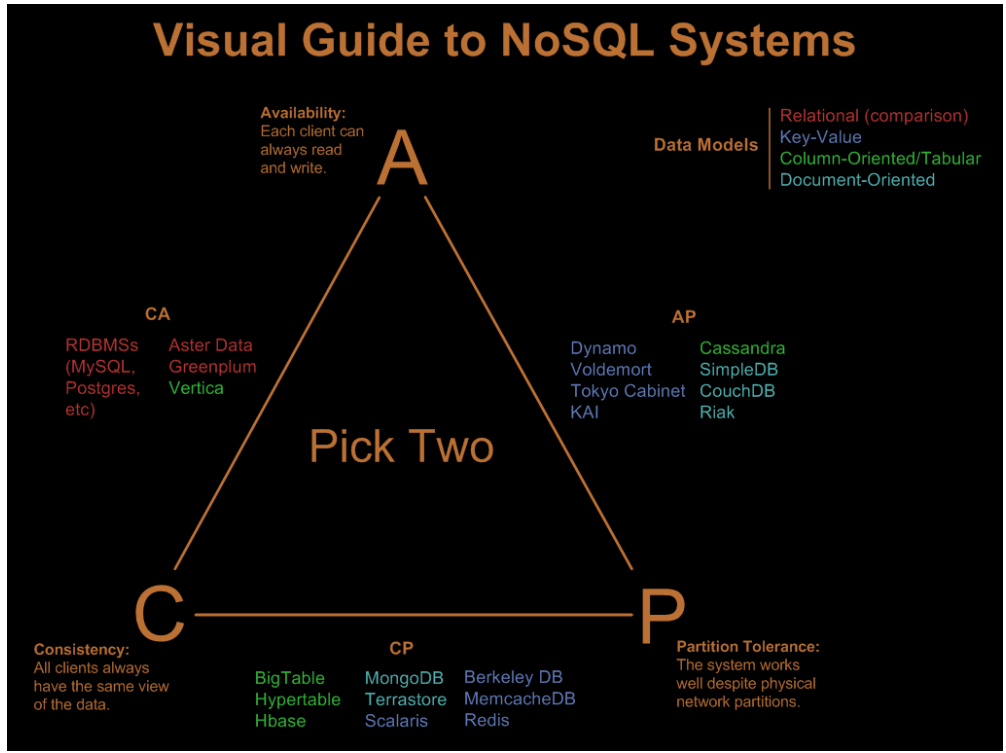


[Prisma Data Guide](#)
[Understanding Database Types](#)

Brewer's CAP Theorem

CAP Theorem states that you can only have 2 of the 3 following guarantees:

1. **Consistency** – every read receives the most recent write or an error
2. **Availability** – Every request receives a (non-error) response, without the guarantee that it contains the most recent write
3. **Partition Tolerance** – the system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes



Brewer's CAP Theorem
Visual Guide to NoSQL Systems
CAP Theorem

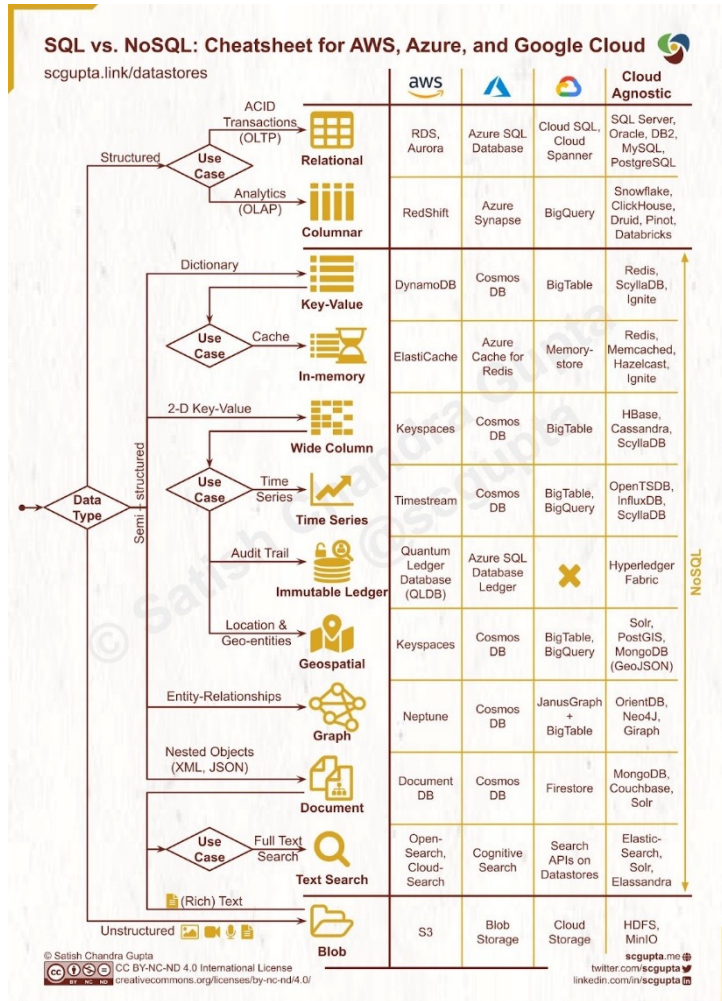
The Art of Database Selection

Choose based on Needs

- Language Support (ex. Python)
- Business logic (Schema)
- Data needs (ex. Geospatial)
- Hosted locally or cloud
- Compute Speed
- OLAP vs OLTP

OLAP - Online Analytical Processing
For efficient data reading and analysis

OLTP - Online Transaction Processing
For efficient data writing and storage



[OLTP vs OLAP](#)
[Understanding Database Types](#)

Database Languages

Database Languages: SQL

```sql
-- Select "all" from table "customers"
SELECT * from customers

SELECT * FROM artist AS art
INNER JOIN album AS alb
ON art.artist_id = alb.artist_id
```

SQL

- "Structured Query Language"
- Create, Read, Update, Delete databases, tables, entries
- Most common language

SQL Tutorial - w3schools
Python SQL Libraries – Real Python
earning SQL 3ed
SQL Cheat Sheets

Database Languages: MQL

```
# Find a limited number of results
db.users.find().limit(10)


# Find users by family name
db.users.find({"name.family": "Smith"}).count()


# Query Documents by Numeric Ranges
# All posts having “likes” field with numeric value greater
than one:
db.post.find({likes: {$gt: 1}})
# All posts having 0 likes
db.post.find({likes: 0})
#All posts that do NOT have exactly 1 like
db.post.find({likes: {$ne: 1}})
```

MongoDB Query Language
Only for MongoDB. Same language as the Mongo Database and CLI.

```
# Sort Results by a Field
# order by age, in ascending order (smallest values first)
db.user.find().sort({age: 1})

# Returns
'''
{
    "_id": ObjectId("5ce45d7606444f199acfba1e"),
    "name": {given: "Alex", family: "Smith"},
    "email": "email@example.com",
    "age": 27
}
{
    _id: ObjectId("5effaa5662679b5af2c58829"),
    email: “email@example.com”,
    name: {given: “Jesse”, family: “Xiao”},
    age: 31
}
'''
```

Using MongoDB: Syntax, Shell And Querying Data

Database Languages: GraphQL

Query

```
{
  hero {
    name
    appearsIn
  }
}
```

Result

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "appearsIn": [
        "NEWHOPE",
        "EMPIRE",
        "JEDI"
      ]
    }
  }
}
```

Graph Query Language

"GraphQL is a query language for your API, and a server-side runtime for executing queries using a type system you define for your data.

GraphQL isn't tied to any specific database or storage engine and is instead backed by your existing code and data."

GraphQL | A query language for your API
GraphQL Code Libraries, Tools and Services
GraphQL Specification Versions

Database Languages: Python ORM

Python

```python
from sqlalchemy import Column, Integer, String, ForeignKey, Table
from sqlalchemy.orm import relationship, backref
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()

author_publisher = Table(
    "author_publisher",
    Base.metadata,
    Column("author_id", Integer, ForeignKey("author.author_id")),
    Column("publisher_id", Integer, ForeignKey("publisher.publisher_id")),
)

book_publisher = Table(
    "book_publisher",
    Base.metadata,
    Column("book_id", Integer, ForeignKey("book.book_id")),
    Column("publisher_id", Integer, ForeignKey("publisher.publisher_id")),
)

class Author(Base):
    __tablename__ = "author"
    author_id = Column(Integer, primary_key=True)
    first_name = Column(String)
    last_name = Column(String)
    books = relationship("Book", backref=backref("author"))
    publishers = relationship(
        "Publisher", secondary=author_publisher, back_populates="authors"
    )

class Book(Base):
    __tablename__ = "book"
    book_id = Column(Integer, primary_key=True)
    author_id = Column(Integer, ForeignKey("author.author_id"))
    title = Column(String)
    publishers = relationship(
        "Publisher", secondary=book_publisher, back_populates="books"
    )

class Publisher(Base):
    __tablename__ = "publisher"
    publisher_id = Column(Integer, primary_key=True)
    name = Column(String)
    authors = relationship(
        "Author", secondary=author_publisher, back_populates="publishers"
    )
    books = relationship(
        "Book", secondary=book_publisher, back_populates="publishers"
    )
```

ORM – Object Relational Mapping

A type of Python library that wraps database queries with code to embed queries into Python. There are pros and cons to these.

Most popular is SQLAlchemy. But there are many. A great introduction can be found on Real Python

Object-relational Mappers (ORMs) - Full Stack Python
SQLAlchemy - The Database Toolkit for Python
Data Management With Python, SQLite, and SQLAlchemy

Data Tools

Resources

Python Coding Environments
- JetBrains PyCharm
- Visual Studio Code
- Miniconda – light Anaconda
- Anaconda – full data science
- Jupyter Notebook

Data Analytics Tools
- Microsoft Excel
- Microsoft PowerBI

Interface Builders
- Streamlit
- Plotly Dash
- Retool

Data Tools
- JetBrains Datagrip
- JetBrains Dataspell
- DB Browser for SQLite
- MITO for Exploratory Data Analysis (EDA) - Medium

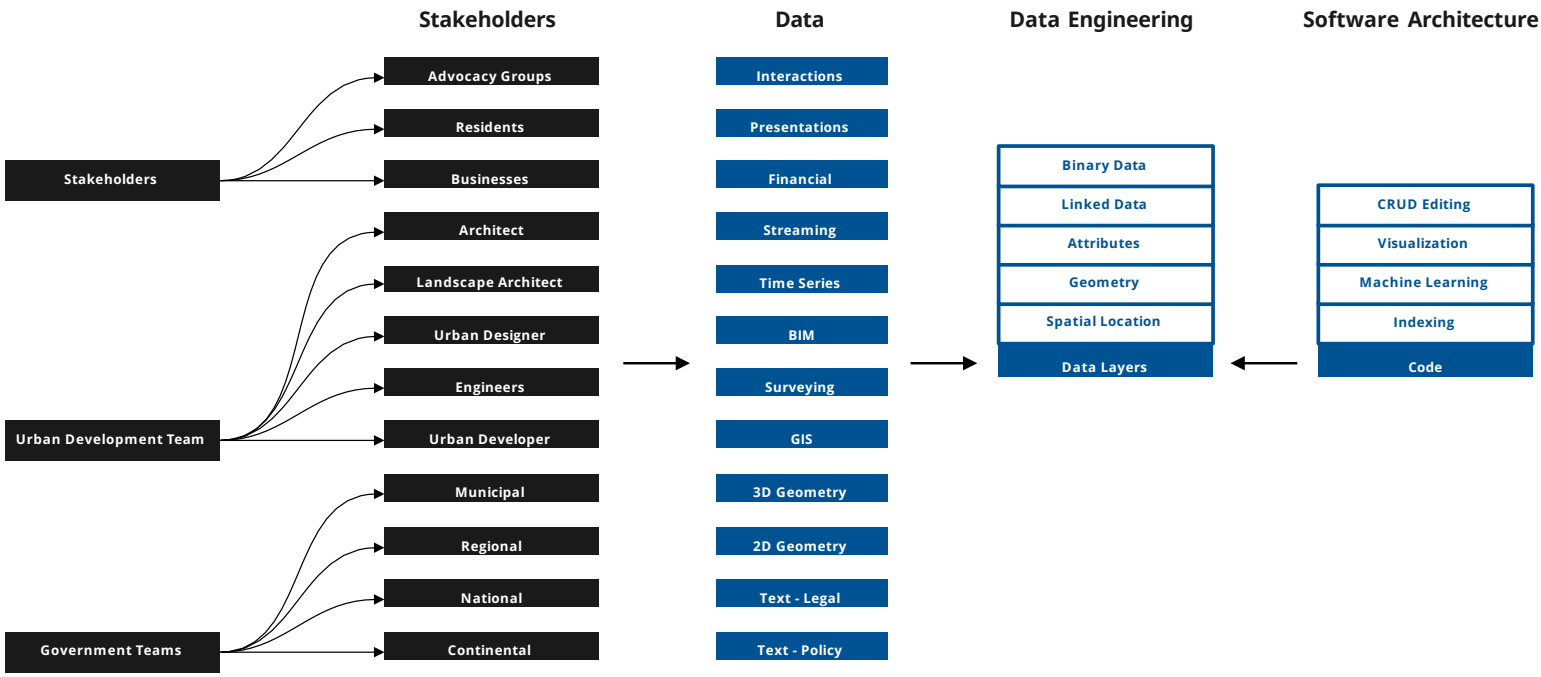
Advanced Tooling
- Data Version Control DVC Introduction
- Pipenv

Get Free Student Versions!
- JetBrains Education
- GitHub Education Pack

Jetbrains Education
GitHub Student Developer Pack
Downloading conda / Miniconda or Anaconda

Data Sources

Urban Data Complexity



Unlimited Stakeholders and Data Types

Data Sources

Files
- CSV, JSON, etc.

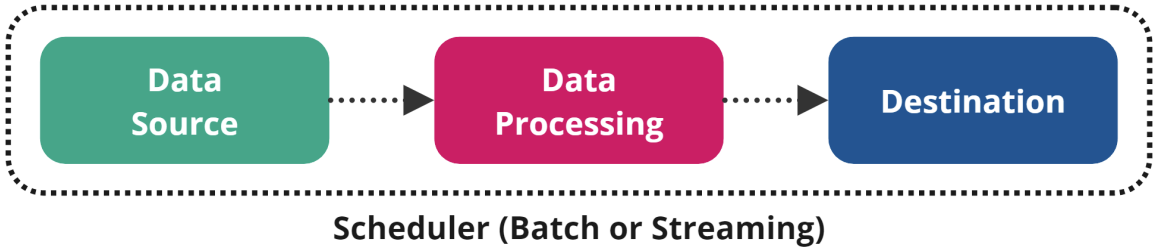
Web APIs
- Application Programming Interface
- Data or Code is exposed by the creator for use by developers.
- API can be Free or Paid
- Ex: Google Maps Place Data API

Web Scraping
- Extract data from web pages
- Check the legal limitations
- Some websites block web scraping (ex. Funda.nl)
- Python Packages
  - Beautiful Soup
  - Scrapy
- Some software exist for this

A Practical Introduction to Web Scraping in Python
Beautiful Soup: Build a Web Scraper With Python
The Top 10 Python Libraries for Web Scraping
10 Best Web Scraping Tools in 2023Earth Engine Data Catalog | Google for Developers
10 Free GIS Data Sources

Data Pipelines

Data Pipeline and Patterns



- Data Pipelines include the above 4 ingredients
- There are many different **patterns** of how they are setup
- They can be run anywhere: local computer, server, cloud
- They can be scheduled for **Batch** or **Streaming**

- **Batch** processing runs one time and is common data analytics (ex. Customer updates once per week)
- **Streaming** processing is continuous, used for big data or realtime analytics (ex. Realtime weather updates)

Writing Your First Pipeline - Seattle Data Guy
Data Pipeline - Data Engineering Wiki
Batch Data Processing
Stream Data Processing
What Data Pipeline Architecture should I use?
Data pipeline architecture : A complete guide

Data Processing / Business Logic

The Data Processing piece of a data pipeline. Depending on the business logic and Data Pipeline Pattern, you carry out specific tasks on the data. The most common are: **ETL** and **ELT**

ETL : Extract, Transform, Load

- **Extract** is the process of retrieving data from one or more sources—online, on-premises, legacy, SaaS, or others.
- **Transformation** involves taking that data, cleaning it, and putting it into a common format, so it can be stored in a targeted database, data store, data warehouse, or data lake.
- **Loading** is the process of inserting that formatted data into the target database, data store, data warehouse, or data lake

ELT : Extract, Load, Transform

ELT is the same process as ETL but in a different order. It is the more common technique with new big data workflows. In the case of ELT, all raw data is centralized (ex. Into a **Data Lake**) and only processed before the data analysis happens.

Extract, transform, load
What is ETL? - AWS
What is ETL? - Google Cloud
What Data Pipeline Architecture should I use?

Privacy and Governance

Data Privacy and Security

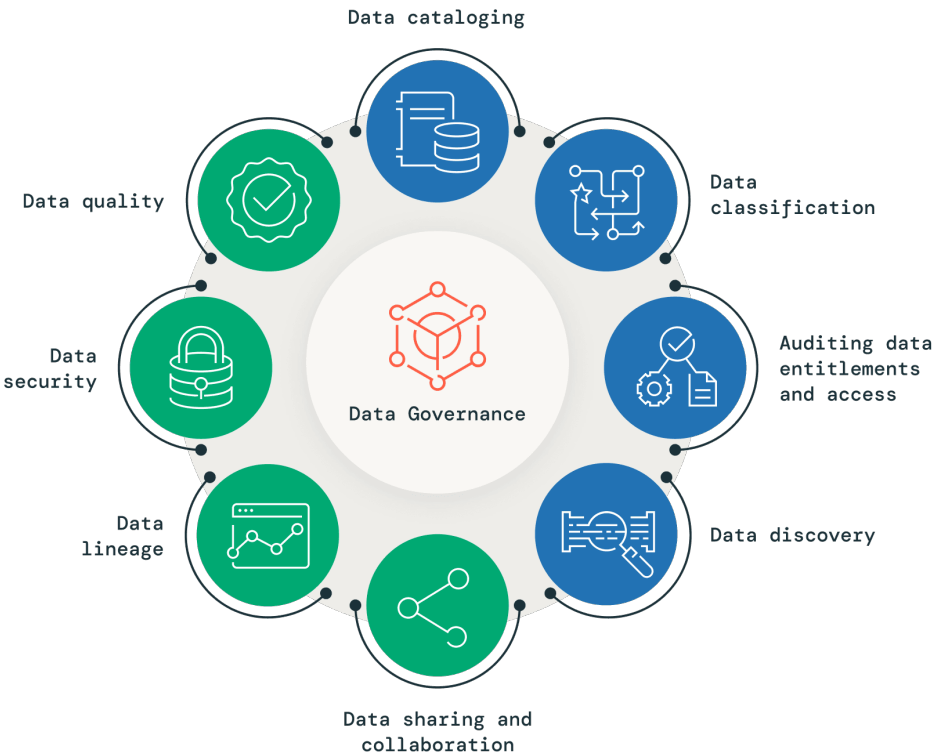
Data is mission critical for all organizations today. With new laws arriving every year, security breaches and mistrust of companies, data engineers must take **Privacy** and **Security** very seriously.

In the EU we have the GDPR (**General Data Protection Regulation**) and in the US we have CCPA (**California Consumer Privacy Act**). Many other countries have other local regulations.

Of most important, all data should be secure, encrypted and not available publicly without reason. Only collect the data you need and only keep it for the minimum time needed. Store data sensitive in the geographic region where it is legally allowed.

The New Rules of Data Privacy – HBR
Data protection in the EU – Europa
General Data Protection Regulation - Wikipedia
Data Privacy: Runbook for Engineers
Nishant Bhajaria Training Courses - LinkedIn Learning

Data Governance

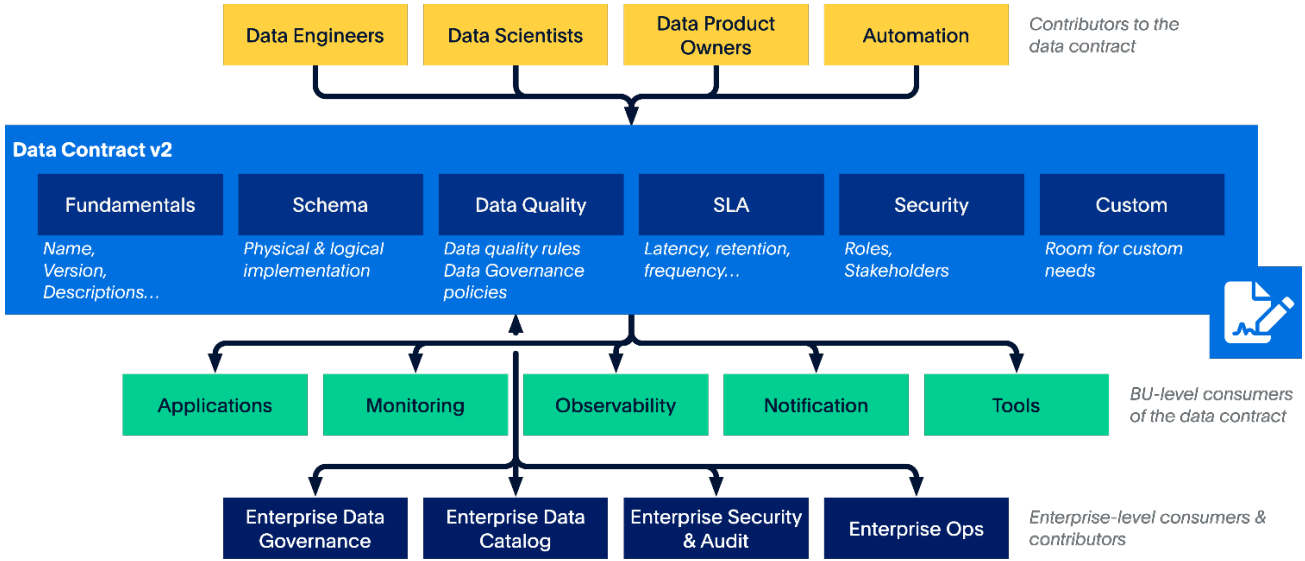


Data Governance is especially a business and management process for all organizational data. It extends to personal rights and privacy.

"The key focus areas of data governance include availability, usability, consistency, data integrity and data security, standard compliance and includes establishing processes to ensure effective data management throughout the enterprise such as accountability for the adverse effects of poor data quality and ensuring that the data which an enterprise has can be used by the entire organization."

Data governance – Wikipedia
What Is Data Governance? Databricks
ISO/IEC 38500:2015 Governance of IT for the organization
COBIT | Control Objectives for Information Technologies | ISACA

Data Contracts



A data contract defines the agreement between a data **producer** and **consumer**. It contains:

- Fundamentals.
- Schema.
- Data quality.
- Service-level agreement (SLA).
- Security & stakeholders.
- Custom properties.

These are a recent evolution in data management, especially to support big data across organizations. It was popularized by Andrew Jones (Book).

Keep them simple and let them serve to improve collaboration.

PayPal Data Contract Template - github
Data Contracts 101 – Monte Carlo Data
The Rise of Data Contracts - Chad Sanderson