365 DataScience

INTRODUCTION TO DATA ENGINEERING

WITH

SHASHANK KALANITHI

365

DATA ENGINEERING
THE CAREER

365

WHAT IS A DATA ENGINEER?

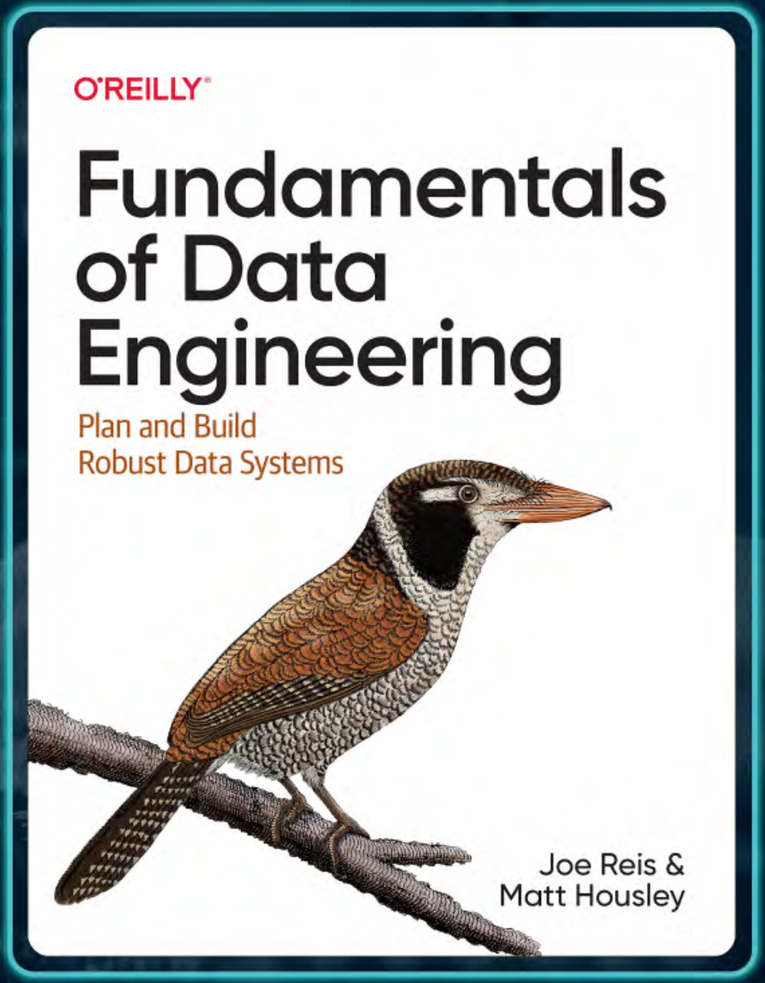
What is a Data Engineer?

Data engineering is the development, implementation, and maintenance of systems and processes that **take in raw data and produce high-quality, consistent information** that supports downstream use cases, such as analysis and machine learning.

Data engineering is the intersection of security, data management, DataOps, data architecture, orchestration, and software engineering.

A **data engineer** manages the data engineering lifecycle, beginning with getting data from source systems and ending with serving data for use cases, such as analysis or machine learning.

- ***Fundamentals of Data Engineering*** – *Joe Reis & Matt Housley*



365

What is a Data Engineer - Simplified

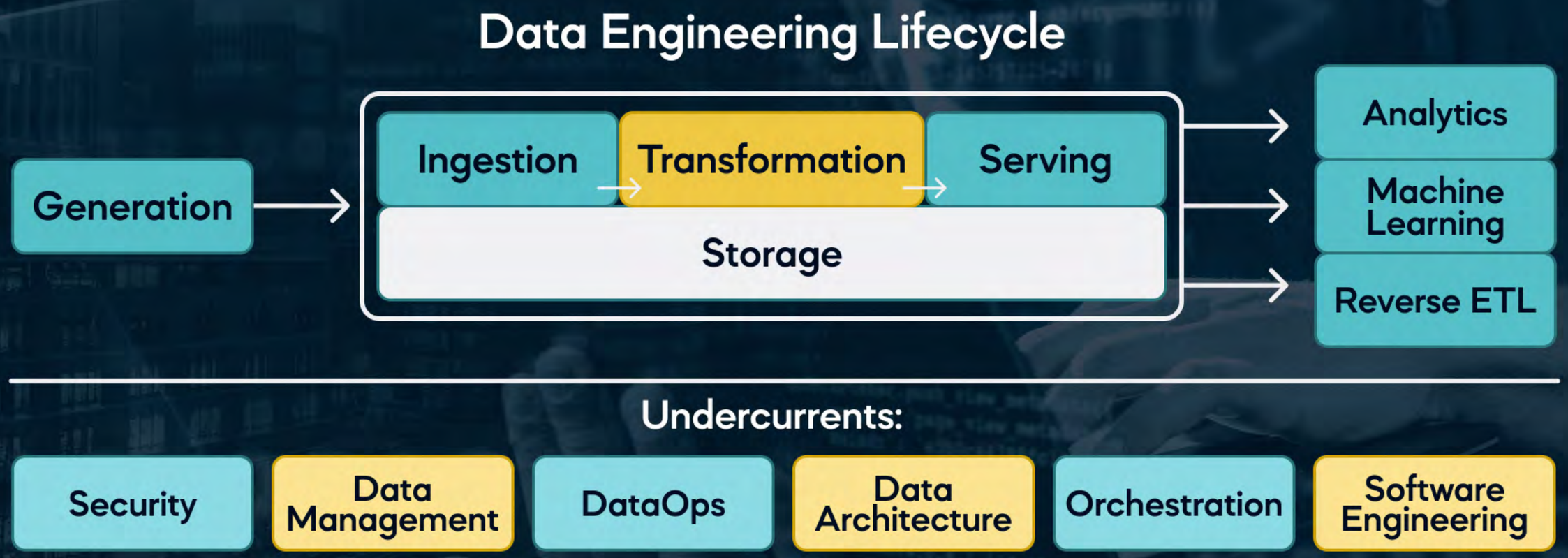
> **Data Engineers** take data from multiple sources and put it somewhere convenient and secure for other people to use.

- Shashank Kalanithi

365

THE DATA ENGINEERING LIFECYCLE

Data Engineering Lifecycle



SIMILAR CAREERS TO
DATA ENGINEERING

Similar Careers to DE – Software Engineers

Software Engineer

Focus:
Designs and develops software applications or systems.

Key activities:
Implementing software solutions, ensuring application performance, security, and scalability.

Differences from Data Engineer

Scope of work: Broader, focusing on application development rather than on managing and optimizing data pipelines.

Tools and Technologies: Uses a wide range of programming tools and environments, not necessarily specialized in data processing technologies like Hadoop or Spark.

Collaboration with Data Engineer

Integration: Work together to integrate data-driven applications with backend data systems.

System Design: Collaborate on designing systems that are both efficient in data processing and effective in meeting user needs.

Similar Careers to DE – Data Scientists

Data Scientist

Focus:
Uses statistical methods and machine learning to analyze data and generate insights.

Key activities:
Building predictive models, performing complex data analyses, and statistical testing.

Differences from Data Engineer

Objective: Primarily concerned with data analysis and model creation rather than infrastructure and pipeline construction.

Skills and Tools: Expertise in statistical analysis and machine learning models, using tools like Python, R, and their associated libraries, which contrasts with the data orchestration and storage optimization tools used by Data Engineers.

Collaboration with Data Engineer

Data Pipeline Design: Data Scientists rely on Data Engineers to design and maintain data pipelines that provide clean, usable data for analysis.

Model Deployment: Collaborate to deploy machine learning models into production, ensuring they are scalable and accessible.

Similar Careers to DE – Data Analysts

Data Analyst

Focus:
Collects, processes, and analyzes data to support business decisions.

Key activities:
Manipulating data, creating reports, and visualizing data insights.

Differences from Data Engineer

Scope of work: Data Analysts work with data primarily in its final form to extract insights, unlike Data Engineers who work on the backend aspects of data preparation and pipeline management.

Tools and Technologies: Uses data visualization and reporting tools like Tableau and Power BI, contrasting with the ETL (Extract, Transform, Load), database management, and big data technologies used by Data Engineers.

Collaboration with Data Engineer

Data Engineers provide the tools and systems that Data Analysts need to access and query data efficiently.

Feedback Loop: Data Analysts often identify and communicate data quality issues or additional needs that Data Engineers can address to improve the overall data infrastructure.

DATA ENGINEERING

SERVICE MODELS

Data Engineering Service Models

	FOCUS	STRUCTURE	WORKFLOW	KEY CHARACTERISTICS
SERVICE MODEL	The service model emphasizes serving clients or internal departments, with a focus on specific requests and projects.	The team is typically organized by skill sets or services offered, such as data warehousing, ETL (extract, transform, load) processes, data analytics, etc.	Work is project-oriented, prioritizing client or stakeholder needs and deadlines, often handling multiple projects with various objectives and timelines.	High client interaction, flexible to changing demands, diverse range of projects, emphasis on timely delivery and adherence to specifications.
STARTUP MODEL	The environment is dynamic and emphasizes agility and innovation, with frequent infrastructure development.	Team members must be adaptable, ready to learn on the job, and fill emerging gaps, with a flexible approach to unforeseen product planning needs.	Workflow involves extensive experimentation, rapid prototyping, and iterative development, requiring adaptability to shifting priorities.	Agile, innovative, risk-taking, adaptable to change, often working with limited resources and under tight timelines.
PRODUCT MODEL	Focused on building and maintaining data products – these could be internal tools, customer-facing applications, or any data-driven software product.	Engineers collaborate closely with Data Scientists, Product Managers, and cross-functional teams, primarily supporting product-focused efforts.	The workflow is structured around software development practices like Agile or Scrum, focusing on continuous improvement and user feedback.	Product-centric, long-term focus, cross-functional collaboration (with product managers, designers, etc.), emphasis on scalability, reliability, and user experience.

365

Further Reading:

Different Types Of "Data Engineering" Teams

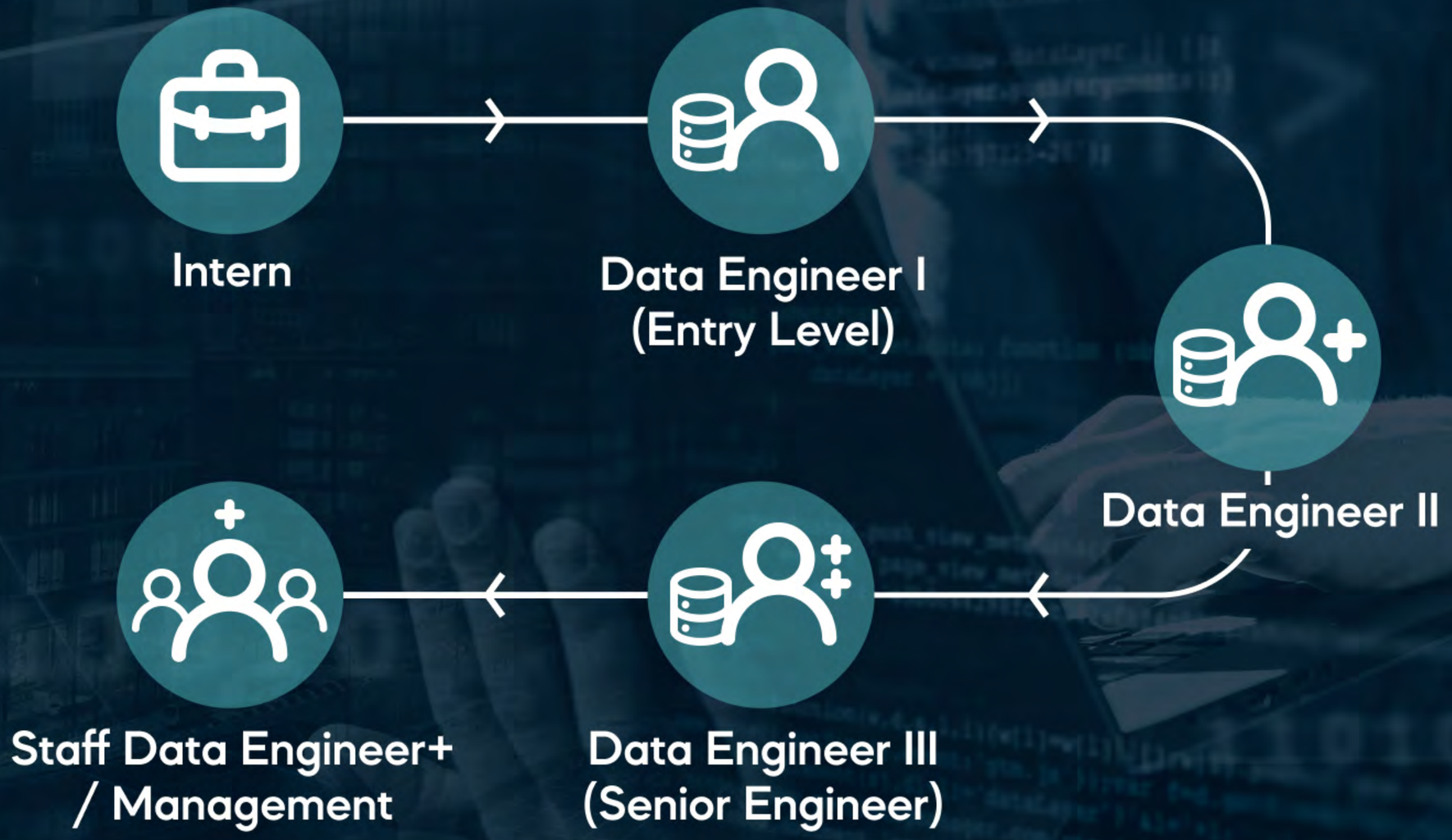
365

SAMPLE

DATA ENGINEER

LEVELING GUIDE

Sample Data Engineer Leveling Guide



Intern

- Generally must be enrolled or recently graduated from a degree program
- Good internships pay workers well, great internships pay close to entry-level wages
- Matched up a with a mentor
- Expected to finish 1-2 projects which might be high-importance but low urgency for a team
- Can be extended a return offer for good performance

Data Engineer I (Entry Level Engineer)

Generally expected to understand DE / SWE fundamentals

Some companies train new grads

Will be given clearly defined tasks to accomplish

Expected to be promoted in < 3 years

A willingness to learn is paramount to success

Understanding what you don't know is key

Data Engineer II

- Solid understanding of your team's tech stack
- People will start asking you questions
- Showing growing project management skills is important
- Will still be given defined projects but might be invited to project planning meetings
- Senior engineers from "lower-tier" companies might be downleveled

Data Engineer III (Senior Engineer)

- Career/Terminal-level position
- Workhorses of an organization (combination of # and competence)
- Potentially have apex level technical skills
- Oftentimes the most knowledgeable engineers about specific parts of the codebase
- Mentoring is important
- Delivers small projects end-to-end
- Budding leadership skills

Staff Engineer / Management

- Divergence in career options
- Large drop-off in career progression
- Can be flexibly transferred onto projects to help teams meet deadlines

TECHNICAL SKILLS of a DATA ENGINEER

Technical Knowledge Skills for Data Engineers

SQL

Data Modeling

Infrastructure as Code

Data Orchestration

Expert Level

- Query Optimization

2 DATA ARCHITECTURE

365

WHAT IS
DATA ARCHITECTURE?

What is Data Architecture

Overarching structure and design principles guiding the **collection, storage, management, and use of data** within an organization encompassing:

- **Data Integration** - Consolidating data from disparate sources
- **Data Storage** – Storing data in a compliant, accessible, and cost-effective manner
- **Data Security / Compliance** – New government requirements make this a tougher problem than past years
- **Data Flow and Pipeline Design** – Maintaining and deprecating pipelines, being wary of costs associated with ETL processes
- **Data Quality and Reliability** – Unreliable data will quickly lead to your org losing credibility

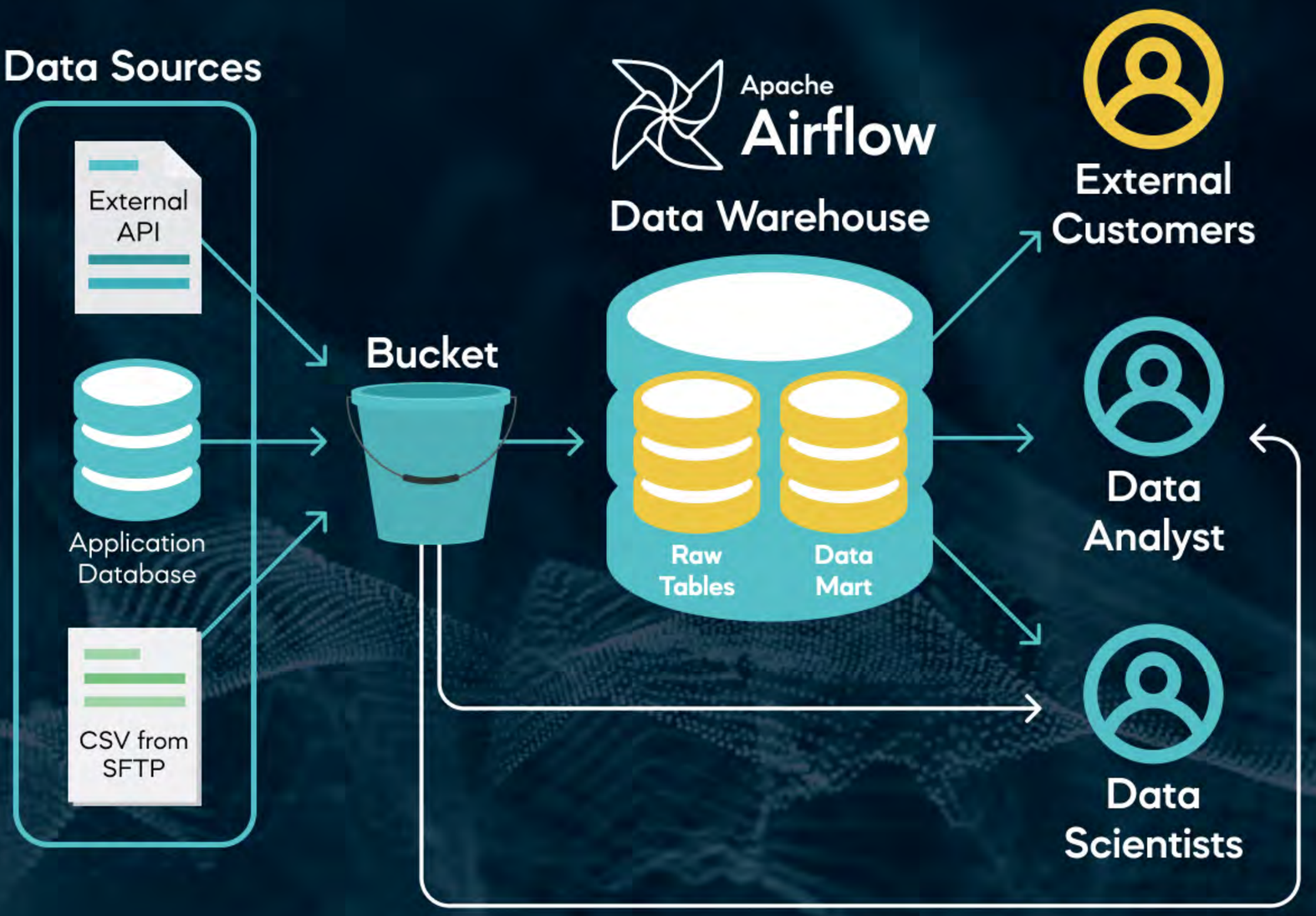
A SAMPLE
DATA ARCHITECTURE

A Sample Data Architecture

Data Sources
- External APIs
- Application Database
- CSV from SFTP

Intermediate Storage
- Bucket
  - Reverse ETL from here

Data Warehousing
- Raw Tables
  - Provides traceability
- Data Mart
  - For normal consumption



MODERN
DATA STACK

Modern Data Stack

Defined in opposition to legacy (older) data stacks

Characteristics

Cloud Based:
As opposed to on-prem

Consumption Based Pricing:
Like AWS

ELT Bias
Do your transformations in a Cloud Warehouse

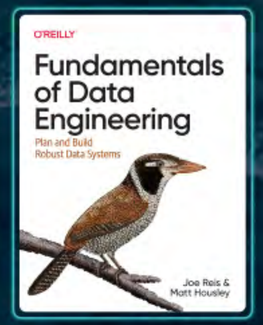
Open Source Bias
Tools like Airflow, DBT, Spark

Component Based:
No end-to-end turnkey solutions

GOOD DATA ARCHITECTURE

Good Data Architecture

1. Choose common components wisely
2. Plan for failure
3. Architect for scalability
4. Architect for leadership
5. Always be architecting
6. Build loosely coupled systems
7. Make reversible decisions
8. Prioritize security
9. Embrace FinOps



*Source - **Fundamentals of Data Engineering** – Joe Reis & Matt Housley*

3 DATA ORCHESTRATION

DATA ORCHESTRATION

Data Orchestration

Data orchestration helps us coordinate, schedule, and run data workflows. Things that make Data Orchestration useful include:

- Dependency Management
- Automation
- Compliance and Security
- Improved Data Quality and Security

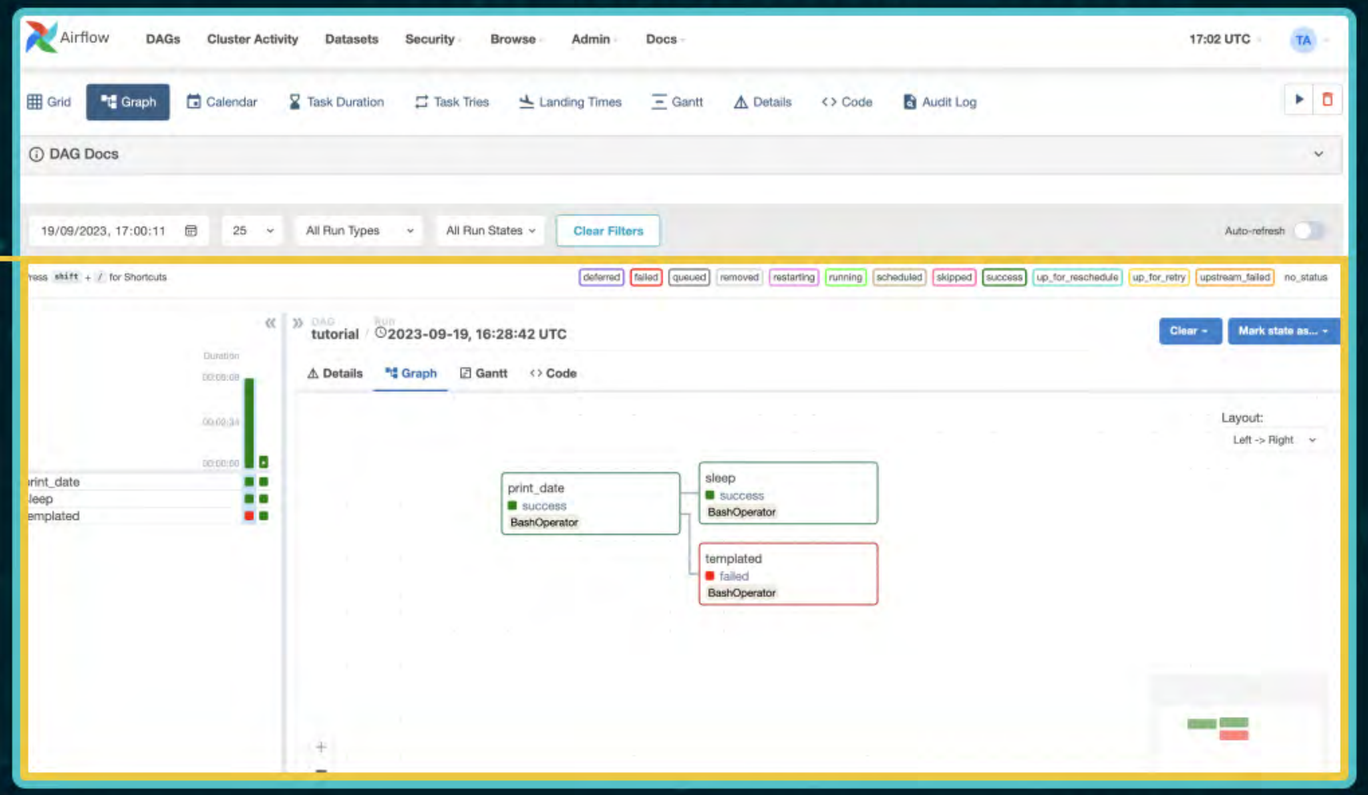
APACHE AIRFLOW

Apache Airflow Concepts

Apache Airflow is an open-source tool for orchestrating complex workflows and data pipelines, centered around these key concepts:

- **Directed Acyclic Graph (DAG):** The foundational concept in Airflow. A **DAG is a collection of all the tasks you want to run, organized in a way that reflects their relationships and dependencies**. A DAG is not a data streaming solution; it's a way to manage tasks. Each DAG defines a sequence of tasks that cannot loop back to a former task (hence acyclic).

- **Task:** A **task represents a unit of work within a DAG**. Each task is an instance of an 'Operator', and it's the Operators that determine what actually gets done.



Apache Airflow Concepts

- **Operator:** Operators are the building blocks of a task. Each operator in Airflow is designed to do a specific type of work. For example, the **PythonOperator** allows you to run a Python function, the **BashOperator** runs a Bash command, the **PostgresOperator** runs SQL commands for PostgreSQL, and so on.

- **Workflow:** **The sequence and organization of tasks in a DAG**, defining the logical flow of execution.

```python
from datetime import datetime

from airflow import DAG
from airflow.decorators import task
from airflow.operators.bash import BashOperator

# A DAG represents a workflow, a collection of tasks
with DAG(dag_id="demo", start_date=datetime(2022, 1, 1), schedule="0 0 * * *") as dag:
    # Tasks are represented as operators
    hello = BashOperator(task_id="hello", bash_command="echo hello")

    @task()
    def airflow():
        print("airflow")

    # Set dependencies between tasks
    hello >> airflow()
```

Apache Airflow Concepts

Scheduler: The component in Airflow responsible for **scheduling tasks**. It monitors all tasks and DAGs, then triggers the task instances once their dependencies are complete.

Executor: Responsible for **running the tasks**. Airflow supports several executors, with different characteristics in terms of scalability, isolation, and so on.

4 RELATIONAL DATABASES

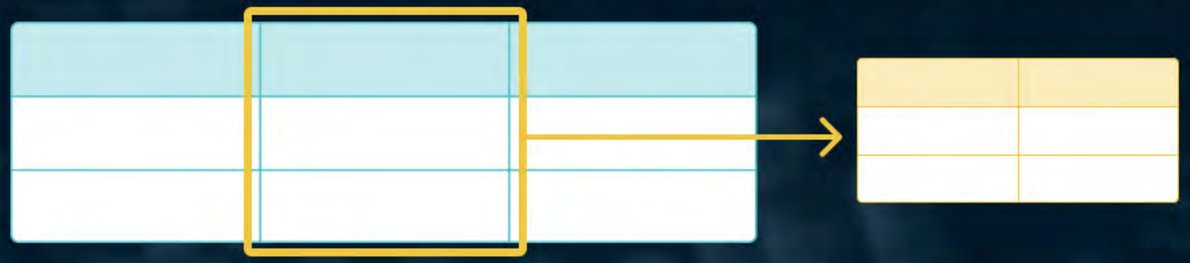
RELATIONAL DATABASES

OVERVIEW

Relational Databases

- Data is organized in **tables**

- Tables have **columns**, some of which **help Relate to other tables**



- Most analysis conducted by analysts will use a **Relational Database**

Relational Databases

- **Each row is a record, each column is an attribute**
  - A **record represents something important to the business** like a User or a Sale
  - An **attribute is a characteristic of something important to the business** like the age of a user or the price something was sold at
- **The primary key** is supposed to be a **column that's unique per row**

UserID (Primary Key)	UserName	Email	DateJoined	Age
1	john_doe	johndoe@example.com	2022-01-08	28
2	jane_smith	janesmith@example.com	2022-03-15	32
3	zelda_jones	zeldajones@example.com	2022-07-21	24
4	bob_brown	bobbrown@example.com	2022-09-10	29

RELATIONAL DATABASES

OPERATIONS

Relational Database Operations

- There are many data transformations that we can perform on a relational database:
  - **UNIONS** - Stacks two tables on top of on another
  - **JOINS** - Laterally combines tables based on some linking column (usually called a foreign key)

- **UNIONs and JOINs** allow us to keep data separated into logical components and only combine them when strictly necessary

Relational Database Operations (UNION)

UserID (Primary Key)	UserName	Email	DateJoined	Age
1	john_doe	johndoe@example.com	2022-01-08	28
2	jane_smith	janesmith@example.com	2022-03-15	32
3	zelda_jones	zeldajones@example.com	2022-07-21	24
4	bob_brown	bobbrown@example.com	2022-09-10	29

UserID (Primary Key)	UserName	Email	DateJoined	Age
5	charlie_green	charliegreen@example.com	2022-10-05	26
6	diana_white	dianawhite@example.com	2022-11-12	30
7	link_green	linkgreen@example.com	2022-12-20	35

UserID (Primary Key)	UserName	Email	DateJoined	Age
1	john_doe	johndoe@example.com	2022-01-08	28
2	jane_smith	janesmith@example.com	2022-03-15	32
3	zelda_jones	zeldajones@example.com	2022-07-21	24
4	bob_brown	bobbrown@example.com	2022-09-10	29
5	charlie_green	charliegreen@example.com	2022-10-05	26
6	diana_white	dianawhite@example.com	2022-11-12	30
7	link_green	linkgreen@example.com	2022-12-20	35

Relational Database Operations (JOIN)

UserID (Primary Key)	UserName	Email	DateJoined	Age
1	john_doe	johndoe@example.com	2022-01-08	28
2	jane_smith	janesmith@example.com	2022-03-15	32
3	zelda_jones	zeldajones@example.com	2022-07-21	24
4	bob_brown	bobbrown@example.com	2022-09-10	29

OrderID (Primary Key)	UserName (Foreign Key)	ProductName	OrderDate	Quantity
1001	1	Laptop	2024-01-10	1
1002	2	Smartphones	2024-01-15	2
1003	3	Headphones	2024-01-20	1
1004	4	Tablet	2024-01-25	1

UserID (Primary Key)	UserName	Email	OrderID	ProductName	OrderDate	Quantity
1	john_doe	johndoe@example.com	1001	Laptop	2024-01-10	1
2	jane_smith	janesmith@example.com	1002	Headphones	2024-01-15	2
3	zelda_jones	zeldajones@example.com	1003	Smartphone	2024-01-20	1
4	bob_brown	bobbrown@example.com	1004	Tablet	2024-01-25	1

RELATIONAL DATABASES

TYPES

Relational Database Types

Online Transaction Processing (OLTP)

- **Primary Purpose:** Designed for managing transaction-oriented applications.
- **Operations:** Focuses on insert, update, and delete operations (known as CRUD - Create, Read, Update, Delete).
- **Query Type:** Processes a large number of short, atomic transactions.
- **Performance:** Optimized for a large number of small transactions.
- **Concurrency:** High level of concurrency; supports multiple transactions by many users.
- **Examples:** Banking systems, retail sales, reservation systems.
- **Data Updates:** Frequent and in real-time.

Online Analytical Processing (OLAP)

- **Primary Purpose:** Designed for query-intensive operations and data analysis.
- **Operations:** Focuses on complex queries for analyzing and aggregating data.
- **Query Type:** Processes fewer but more complex queries.
- **Performance:** Optimized for querying and reporting, often involving large volumes of data.
- **Concurrency:** Lower level of concurrency compared to OLTP.
- **Examples:** Business reporting functions, data warehouses, decision support systems.
- **Data Updates:** Less frequent; often batch-updated, not real-time.

Exercises

Complex queries involving large volumes of data for reporting purposes are a characteristic of?

Batch-updated data is more common in?

A system which is optimized for handling a large number of small transactions?

ORGANIZING RELATIONAL DATABASES

Components of a Relational Database

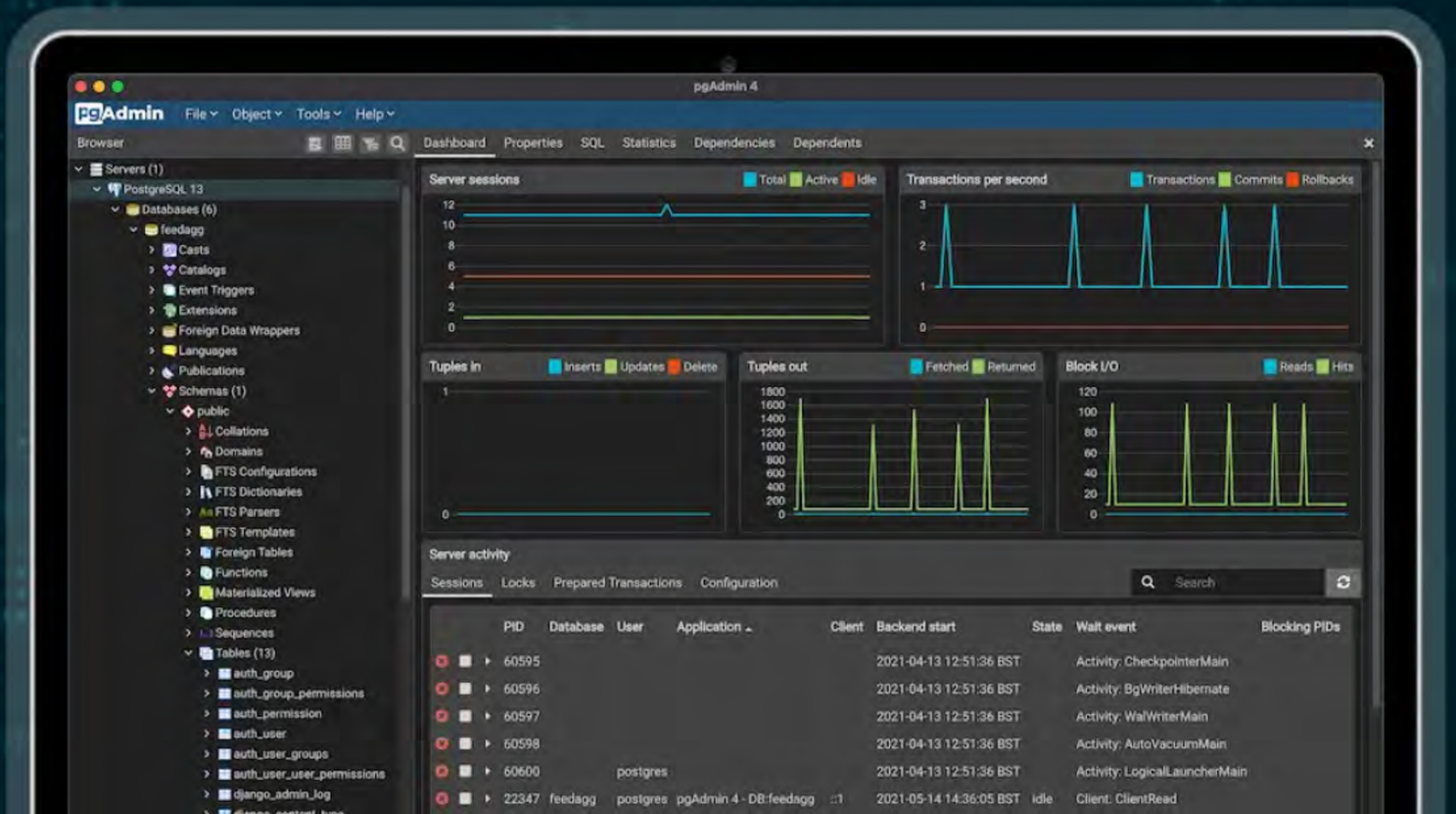
- **Servers:** A program (or the hardware it runs on) that **offers database services to other programs** or computers, handling database access, executing queries, retrieving data, and ensuring security and integrity.

- **Databases:** Collection of tables, views, schemas, and other database objects, managed by a **Database Management System (DBMS)**. Databases are designed to store, retrieve, add, update, and delete data efficiently.

- **Schemas:** A logical grouping of database objects like tables, views, and functions, tied to a database user. Schemas organize objects, allow inter-object references without explicit names, and manage access permissions collectively.

- **Tables:** Collection of related data held in a structured format within a database

- **Views:** Virtual table in SQL databases that is based on the result-set of an SQL statement



365

INTERACTING WITH RELATIONAL DATABASES

SQL

SQL

- **Primary method** for interacting with **Relational Databases**
  - **Fundamental skill** for any Data professional, especially Data Engineers
- Used to create, manipulate, and query databases
- **Versatile:** SQL can interact with a wide range of database systems, from small-scale applications to large enterprise systems.
- **Functional:** It allows for complex querying, data aggregation, and manipulation, enabling users to extract meaningful insights from large datasets.
- **Standardized:** SQL syntax is fairly consistent across different database systems, such as MySQL, PostgreSQL, Oracle, and SQL Server, making skills transferable across platforms.

UserID (Primary Key)	UserName	Email	DateJoined	Age
1	john_doe	johndoe@example.com	2022-01-08	28
2	jane_smith	janesmith@example.com	2022-03-15	32
3	zelda_jones	zeldajones@example.com	2022-07-21	24
4	bob_brown	bobbrown@example.com	2022-09-10	29

```
SELECT
        FirstName,
        LastName,
        Department,
        Salary
FROM Employees
WHERE Department = 'Engineering'
ORDER BY LastName ASC;
```

FirstName	LastName	Department	Salary
Alice	Johnson	Engineering	70,000
Bob	Smith	Engineering	75,000
Carol	Taylor	Engineering	68,000
David	Brown	Engineering	72,000

365

NON-RELATIONAL DATABASES

ACID DATABASES

ACID Properties

Atomicity:

Ensures that all operations within a database transaction are completed successfully. If not, the transaction is aborted and the database state is left unchanged.

Isolation:

Ensures that transactions are securely and independently processed at the same time without interference, despite being executed concurrently.

Consistency:

Ensures that a transaction brings the database from one valid state to another, maintaining database constraints.

Durability:

Once a transaction has been committed, it will remain so, even in the event of a system failure.

ACID Properties

Database Paradigm	Atomicity	Consistency	Isolation	Durability
Relational Databases	Yes	Yes	Yes	Yes
NoSQL Databases	Varies	Varies	Varies	Varies
Document-Based Stores	No	No	No	No
Key-Value Stores	No	No	No	No
Graph Databases	Varies	Varies	Varies	Varies
Object-Oriented Databases	Yes	Yes	Yes	Yes

DOCUMENT DATABASES

Document Databases

- Store data as Documents formatted like a {j s o n} object
- Generally schema-less, each document doesn't have to follow a specified schema (depends on the database)
- Much more flexible and generally easier to horizontally scale than a Relational Database
- Sometimes preferred by Software Engineers because it's very flexible and requires less time to build a data model

Examples: mongoDB® **Amazon** DocumentDB **GCP Firestore**

Document Database Example

```
{
  "title": "The Great Gatsby",
  "author": "F. Scott Fitzgerald",
  "publishedYear": 1925,
  "genres": ["Fiction", "Classic", "Literature"],
  "availability": {
    "print": true,
    "ebook": true,
    "audiobook": false
  },
  "ratings": {
    "average": 4.2,
    "reviews": 1523
  }
}
```

Accessing Data:

- db.books.find(
  { title: “The Great Gatsby”
  }
)
- db.books.find(
  { ratings:
  { reviews:
  {$lt: 5
  }
  }
  }
)

Document Databases

Feature	Document Database (NoSQL) Benefits	Relational Database (SQL) Benefits
Schema Flexibility	Schema-less nature allows for flexibility in data structure, accommodating varied and evolving data models without the need for migrations.	Defined schema enforces data consistency and integrity, making it easier to maintain standards across large and complex datasets.
Data Model	Optimized for storing semi-structured or unstructured data in a format (e.g., JSON, BSON) that closely matches the data structures used in application code.	Uses a structured data model that efficiently organizes data into tables, rows, and columns, facilitating complex queries and data relationships.
Scalability	Designed to scale out horizontally, making it easier to distribute data across multiple servers in cloud environments.	Traditionally scales vertically (by adding more power to a single server), though modern solutions also offer horizontal scalability with more complex setup.
Query Language	Uses flexible and descriptive query languages or APIs tailored to document structures, which can be more intuitive for developers.	Utilizes SQL, a powerful and standardized query language, for complex queries, including sophisticated joins and aggregations.
Performance	High performance for read and write operations, especially with large volumes of data, due to the lack of joins and the ability to distribute data.	Efficient at handling complex transactions and queries across related data, benefiting from mature indexing and optimization techniques.
Relationships	Handles relationships by embedding documents or by using document references, which can simplify access patterns for certain use cases.	Excel at managing complex relationships between data entities through foreign keys and joins, offering robust support for data normalization.
Development Speed	Quick iteration and development cycles are possible due to schema-less design and close alignment with object-oriented programming models.	The need for upfront schema design and modifications can slow initial development, but ensures data consistency and integrity in the long term.
Use Cases	Well-suited for applications with rapidly changing data models, big data applications, content management systems, and mobile apps with varied and evolving data types.	Ideal for applications requiring complex transactions, strict data integrity, and established relationships between data entities, such as financial systems, inventory management, and other enterprise applications.

KEY-VALUE DATABASE

Document Database

- Stores data in **key-value pairs**
- Sounds similar to a **Document Database** except values don't have to be documents (JSON objects), they can basically be **anything**
  - **Document Databases** store values as semi-structured data, **key-value databases** can store unstructured data as well.
- Essentially a more **generalized version of a Document Database**

- **Examples:** **Amazon DynamoDB**

OBJECT STORAGE

Object Storage

- **Stores objects in a flat hierarchy** (no folder structure, just the files)
  - This leads to **less metadata and faster retrieval of data** than a typical file system which stores objects in a directory tree
- Oftentimes used as **staging between systems**
  - Data might be **extracted from one database, put into object storage, then put into another database**
  - This could be done to **ensure that there's always a copy of the "raw data"** and because Object Storage is dead simple to use (fewer chances of failure vs a Database Insertion)
- **Examples:** **Amazon S3**, **Azure Blob Storage**, **GCP Cloud Storage**

Further Reading:

Original AWS S3 Press Release

How Facebook Stores Billions of Photos

GRAPH DATABASE

Graph Database

- Stores data in **Nodes, Edges, Properties**
  - **Nodes:** Data objects (people, transactions)
  - **Edges:** Relationships between nodes (likes, type of transaction)
  - **Properties:** Attached to nodes or edges as attributes
- Great for **modeling relationships between objects** (think social networks, fraud detection)
- **Examples:** AWS Neptune, neo4j, Azure Cosmos DB

COLUMNAR DATABASE

Columnar Database

- As opposed to Relational Databases which are optimized for row storage, **columnar databases are optimized along columns**
  - This typically leads to **faster data retrieval for analytics**
    - If you wanted to get the average sales from a sales table, you'd need to quickly **access the 'REVENUE' column**, you don't necessarily care about the other values in the same row
- The goal is to **reduce the number of I/O operations performed on the disk**
  - Analytical queries typically **only ever need a subset of columns**

- **Examples:** cassandra DB

NON-RELATIONAL
DB EXERCISES

Non-Relational DB Exercises

Which **ACID property** ensures that transactions are processed reliably and survive system failures?

In a graph database, **what term** is used to describe the **relationships between nodes?**

What format do document databases typically use **to store data**, allowing for nested structures?

Why are **columnar databases** considered efficient for analytical queries and data warehousing?

365

SOFTWARE ENGINEERING BASICS

HORIZONTAL SCALING
VS
VERTICAL SCALING

Horizontal Scaling vs Vertical Scaling

Vertical Scaling: Using more powerful computers to scale

Simpler to implement from a software perspective

Cost effective at a small scale

Horizontal Scaling: Using a larger number of computers to scale

You can scale almost limitlessly

Can be very difficult to design applications to work across multiple servers

In many companies, **Data Warehousing** is one of the **most computationally intensive operations run.**

- As a **Data Engineer** you might encounter a lot of **decision points** where you need to make a cost effective decision between the ease yet expense of vertical scaling, and the difficulty yet cost savings of horizontally scaling

APIs

APIs

Application Programming Interfaces – Act as a **communication layer** that allows you to interact with another piece of software in a controlled manner

Kind of like a waiter at a restaurant

As a DE, you'll often interact with **REST APIs** to pull data from a service to ETL into a database

APIs

With a **REST API**, there are **5 basic operations** you can perform:

- **GET** – Obtain data/resources from the server
- **POST** – Obtain, write, or modify data/resources in the server
- **PUT** – Modify data/resource in the server
- **DELETE** – Delete data/resources on the server
- **PATCH** – Partially modify data/resources on the server

Example:

```
api_key = "YOUR_API_KEY_HERE"
address = "1600 Amphitheatre
Parkway, Mountain View, CA"

url = f"https://maps.googleapis.com/
maps/api/geocode/json?
address={address}&key={api_key}"

response = requests.get(url)

data = response.json()
```

You'll mostly be interacting with the **GET** and **POST** operations.

CRON

CRON

- Time-based job scheduler
- Enables users to schedule scripts or commands to run automatically at specified times, dates, or intervals
- Usually implemented in configuration files

* * * * * Command to execute

- day of the week (0 - 7) (Sunday = 0 or 7)
- month (1 - 12)
- day of the month (1 - 31)
- hour (0 - 23)
- min (0 - 59)

Example:

* * * * * - Every minute

0 0 * * * - Daily at Midnight

30 1 * * 0 – Weekly on Sunday at 1:30 AM

*/15 * * * * - Once every 15 minutes

CRON Exercises

* * * * 1 = ?

* 5 * 2 * = ?

? ? ? ? ? = Every minute past every 3rd hour in January.

*****

- day of the week (0 - 7) (Sunday = 0 or 7)
- month (1 - 12)
- day of the month (1 - 31)
- hour (0 - 23)
- min (0 - 59)

Example:

* * * * * - Every minute

0 0 * * * - Daily at Midnight

30 1 * * 0 – Weekly on Sunday at 1:30 AM

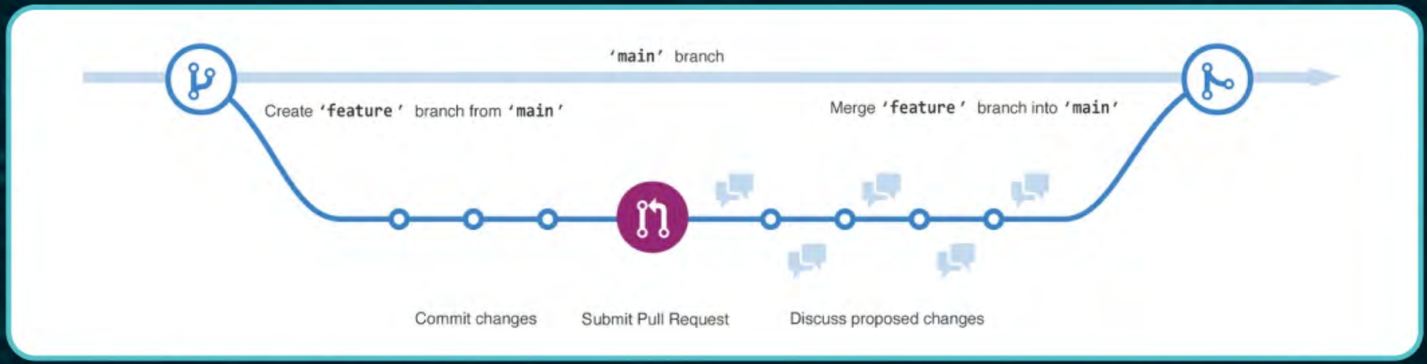
*/15 * * * * - Once every 15 minutes

365

VERSION CONTROL/ GIT/ MERCURIAL

Version Control / Git / Mercurial

- Helps track and manage changes in code
  - Allows multiple developers to work synchronously
  - Allows easy reversions to previous versions of code
  - Features can be easily tracked in code
- Git, SVN (Subversion), and Mercurial are the most common
  - git is by far the most common
- Not all DE orgs have a VC implementation, it's almost always a good idea to get one up and running



INFRASTRUCTURE MANAGEMENT

Infrastructure Management

As you scale up (especially in the cloud), you'll realize that using the GUI to manage your infrastructure is a recipe for some very expensive mistakes

Infrastructure as Code (IaC) is a solution to this complexity

- HashiCorp Terraform
- ANSIBLE
- CDKGlobal.

You can manage infrastructure as code (version control, audit, reuse complex components)

PYTHON

Python

Standard language along with **SQL** for most Data Engineers

- As long as extreme speed is not necessary, **Python or SQL** can solve almost any programming problem for today's **Data Engineers**

Often used to **pass instructions to another system**

- Apache Airflow
- Apache Spark

A **basic understanding of Python** will be enough for **90% of applications** as a Data Engineer

A deeper understanding of the language can really help you **stand out**

Python

Variable Assignment

```python
# Variable assignment and data types
name = "John Doe"  # String
age = 30  # Integer
height = 5.11  # Float
is_student = False  # Boolean
hobbies = ["reading", "cycling", "programming"]  # List
```

If Statement

For Loops

```python
# If statement
if age > 18:
    print(f"{name} is an adult.")
else:
    print(f"{name} is not an adult.")

# For loop
print(f"{name}'s hobbies:")
for hobby in hobbies:
    print(f"- {hobby}")
```

Python

Function

```python
# Functions
def greet(person):
    """Greets a person."""
    return f"Hello, {person}!"

# Calling a function
print(greet(name))
```

Classes

```python
# Classes
class Person:
    def __init__(self, name, age, hobbies):
        self.name = name
        self.age = age
        self.hobbies = hobbies

    def introduce(self):
        """Introduces the person and lists hobbies."""
        intro = f"My name is {self.name}, and I am {self.age} years old. My hobbies
        for hobby in self.hobbies:
            intro += f"\n- {hobby}"
        return intro

# Creating an instance of the class and using it
person_instance = Person(name, age, hobbies)
print(person_instance.introduce())
```

Python - Libraries

Libraries that DEs often use

NumPy - Numerical Python library

pandas - Data manipulation and cleaning

BeautifulSoup - HTML Parsing

Apache **Airflow** - Data Orchestration framework

APACHE **Spark** - Big Data Processing framework

SHELL SCRIPTING

BASH / Shell / Terminal

The terminal on a Mac or Linux ( ) system or PowerShell ( ) on Windows

Standard way to interact with cloud systems and servers

Basics in Shell scripting will be necessary to interact with the cloud and set up/work with SFTP servers

BASH / Shell / Terminal

pwd *(Print Working Directory):* Shows the current directory you're in.

ls *(List):* Lists files and directories in the current directory. Use ls -l for detailed information and ls -a to show hidden files.

cd *(Change Directory):* Changes the current directory. cd .. to go up one directory, cd to return to the home directory.

mkdir *(Make Directory):* Creates a new directory

rmdir *(Remove Directory):* Deletes an empty directory. Use rm -r to remove directories and their contents

BASH / Shell / Terminal

Deletes SSH Command: Secure Shell (SSH) is used to securely access remote machines. The basic syntax is **ssh username@hostname**

- **Example:** ssh user@example.com connects to example.com as user.

Key-based Authentication: For a more **secure and convenient way to use SSH** without entering a password every time, set up SSH keys using **ssh-keygen** and copy the public key to the remote server with **ssh-copy-id user@hostname.**

BASH / Shell / Terminal

scp *(Secure Copy):* Copies files or directories between hosts on a network. It uses SSH for data transfer, providing the same security.

Copy a file to a remote server:
scp file.txt user@hostname:/path/to/destination

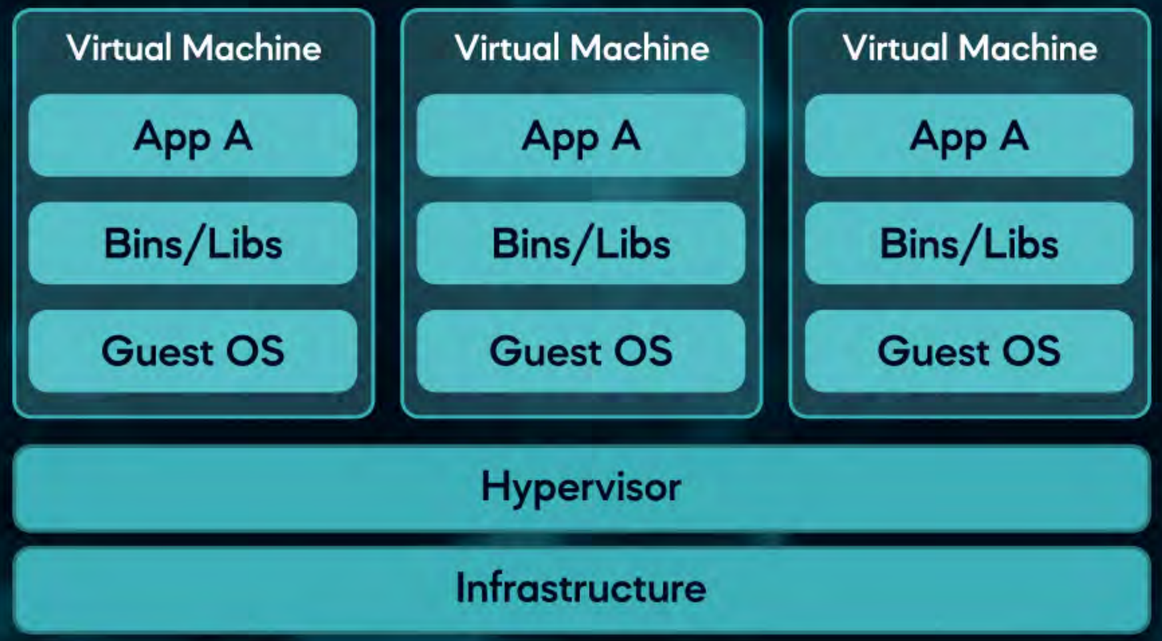
Copy a file from a remote server:
scp user@hostname:/path/to/file.txt/local/destination

DOCKER/ CONTAINERIZATION

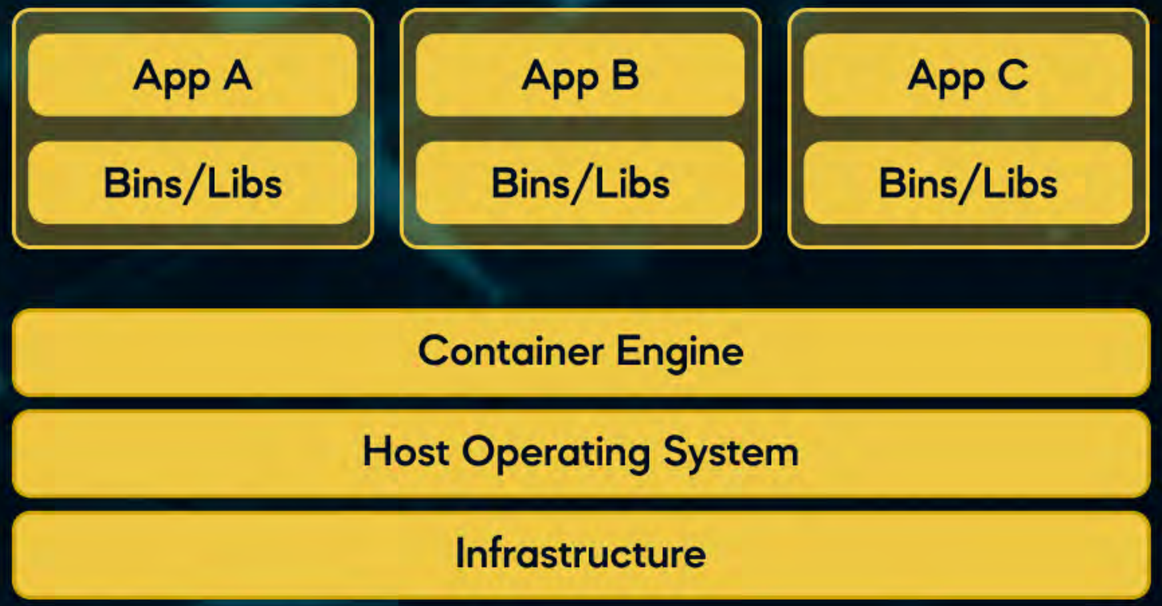
Docker / Containerization

A method for encapsulating an application and its dependencies into a container that can run consistently across any computing environment.

Virtual Machines



Containers



Docker / Containerization

Benefits:

- **Consistency Across Environments:** Ensures applications run the same in development, testing, and production.
- **Efficient Resource Use:** More lightweight and efficient than traditional virtual machines, leading to better utilization of system resources.
- **Simplified Deployment and Scaling:** Streamlines the process of deploying and scaling applications across different environments.
- **Enhanced Security:** Isolates applications from each other and the underlying infrastructure, minimizing security risks.
- **Facilitates DevOps Practices:** Supports Continuous Integration/Continuous Deployment (CI/CD) and DevOps workflows, enabling faster software delivery cycles.

Docker is a popular platform that offers tools, services, and workflows to build, ship, and run containers, making it a powerful solution for implementing containerization strategies.

Testing

Unit Testing and Integration Testing are both crucial methodologies in the software development lifecycle, aimed at ensuring the reliability, functionality, and quality of software applications.

Purpose: Focuses on testing the smallest parts of an application, such as functions or methods, in isolation from the rest of the system. The goal is to ensure that each unit performs as designed.

Early Bug Detection: Bugs can be identified and fixed early in the development process, saving time and resources.

Code Quality: Encourages developers to write more modular and reusable code.

Refactoring Confidence: Provides a safety net that helps developers to refactor code with confidence, knowing that any breakage will be caught by tests.

Testing

Integration Testing

Purpose: Tests the integration or interfaces between components, to expose faults in the interaction between integrated units. It aims to verify the combined functionality after integration.

Detect Interface Defects: Ensures that components work together as expected.

Verify Functional, Performance, and Reliability Requirements: Validates that the system meets specified requirements when units are combined.

System Integration Readiness: Confirms that the system's components can be integrated and will work together without issues.

Testing – Unit Testing

```python
# simple_math.py

def add(a, b):
    return a + b
```

```python
# test_simple_math.py

import unittest
from simple_math import add

class TestSimpleMath(unittest.TestCase):

    def test_add(self):
        self.assertEqual(add(3, 4), 7)
        self.assertEqual(add(-1, 1), 0)
        self.assertEqual(add(-1, -1), -2)

if _name_ == '_main_':
    unittest.main()
```

Testing – Integration Testing

```python
# complex_math.py

def add(a, b):
    return a + b

def add_then_multiply(a, b, multiplier):
    sum_result = add(a, b)
    return sum_result * multiplier
```

```python
# test_complex_math.py

import unittest
from complex_math import add, add_then_multiply

class TestComplexMath(unittest.TestCase):

    def test_add_then_multiply(self):
        self.assertEqual(add_then_multiply(3, 4, 2), 14)
        self.assertEqual(add_then_multiply(-1, 1, 3), 0)
        self.assertEqual(add_then_multiply(-1, -1, 2), -4)

if _name_ == '_main_':
    unittest.main()
```

DATA LAKES, SWAMPS, WAREHOUSES, AND MARTS

Data Lakes

Definition

A Data Lake is a **centralized repository** that allows you to **store all your structured and unstructured data at any scale.** It can store data in its native format, providing high flexibility.

Use Cases

- **Big Data processing and analytics.**
- **Machine learning model training.**
- **Real-time analytics on diverse datasets.**

Key Features

- **Scalability:** Can easily scale to store petabytes of data.
- **Flexibility:** Supports various data types, including logs, XML, images, and social media data.
- **Cost-Effectiveness:** Typically uses low-cost storage solutions to manage vast amounts of data economically.

Benefits

- Enables data democratization by making data accessible for various analytical tools and users.
- Facilitates advanced analytics through comprehensive data collection.

Oftentimes called Data Swamps because they become so hard to organize

Data Warehouse

Definition

A Data Warehouse is a **system used for reporting and data analysis**, and is considered a **core component of business intelligence.** It is optimized to analyze relational data from transactional systems and other sources.

Use Cases

- **Business intelligence and reporting.**
- **Historical data analysis for strategic decisions.**
- **Performance management.**

Key Features

- **Structured Data:** Primarily designed to store structured data.
- **Performance:** Optimized for fast query performance and complex queries.
- **Historical Data Analysis:** Ideal for analyzing historical data over time.

Benefits

- Improves decision-making by consolidating data from multiple sources into a single, comprehensive view.
- Enhances data quality and consistency through data cleansing and integration processes.

Compute costs can quickly spiral if controls are not put in place

Data Mart

Definition

A Data Mart is a **subset of a data warehouse designed to serve a specific function or department within an organization**. It is focused on a particular subject or area.

Benefits

- Provides more relevant and accessible data for specific teams or departments.
- Can provide a "single-source of truth" for a department or team

Key Features

Subject-Oriented: Tailored for specific user groups or business functions.

Speed: Smaller size allows for faster data retrieval.

Use Cases

Department-specific reports and analysis.

Quick access to relevant, processed data for end-users.

Supporting specific business objectives with tailored data.

Can create "multiple sources of truth" if metrics are not coordinated between data marts

Data Lakes, Swamps, Warehouses, Marts

Feature	Data Lake	Data Warehouse
Data Type	Structured and unstructured	Structured
Purpose	Storage and analysis of big data	Data analysis and reporting
Scalability	Highly scalable	Scalable but limited compared to data lakes
Cost	Cost-effective for large datasets	More expensive due to processing capabilities
Performance	Varied, depends on use case	High, optimized for fast queries
Use Case	Big data analytics, machine learning	Business intelligence, historical analysis
Data Accessibility	Flexible, raw data accessible	Structured, cleaned data, SQL

DATA PIPELINES AND ORCHESTRATION

Data Pipelines and Orchestration

Data Orchestration helps us coordinate, schedule, and run data workflows. Things that make Data Orchestration useful include:

- Dependency Management
- Automation
- Compliance and Security
- Improved Data Quality and Security

Data Pipelines and Orchestration

What are Data Pipelines?

Data pipelines are systems that **automate the flow of data from one point to another.**

They are generally not the systems used to process the data (not Apache Spark)

Imagine a pipe transferring water; similarly, a **data pipeline moves data from source to destination.**

Essential for processing **large varieties of data** for insights and data-driven decisions.

Data Pipelines and Orchestration

Why Data Pipelines are Critical

- Automated
- Auditable
- Scheduled
- Cost-effective

Data Pipelines and Orchestration

Anatomy of a Data Pipeline

- **Origin:** Starting point for data collection from various sources.
- **Destination:** Final point where data is stored or utilized for analysis or other applications.
- **Data Flow:** The journey of data, including extraction, transformation, and loading (ETL).
- **Additional Elements:** Storage, processing, monitoring, and alerting mechanisms to ensure smooth operation.

Data Pipelines and Orchestration

Varieties of Data Pipelines

Batch Processing:
Scheduled transfer of large data volumes.

Real-time Processing:
Instant data processing as it's generated.

Cloud-native:
Scalable solutions in cloud environments.

Open-source:
Customizable pipelines with community support.

Data Pipelines and Orchestration

Varieties of Data Pipelines

Batch Processing: Scheduled transfer of large data volumes.

Apache Airflow

Real-time Processing: Instant data processing as it's generated.

debezium

Cloud-native: Scalable solutions in cloud environments.

AWS MWAA | GCP Datawflow

Open-source: Customizable pipelines with community support.

- Open-source: Apache Airflow | Airbyte
- Closed-source: alteryx

Data Pipelines and Orchestration

Varieties of Data Pipelines

Batch Processing:
Scheduled transfer of large data volumes.

Apache Airflow

Real-time Processing:
Instant data processing as it's generated.

debezium

Cloud-native:
Scalable solutions in cloud environments.

AWS MWAA | GCP Datawflow

Open-source:
Customizable pipelines with community support.

Apache Airflow | Airbyte

Data Pipelines and Orchestration

Further Reading

https://www.astronomer.io/blog/data-pipeline/

DATA NORMALIZATION

Data Normalization

- Process of **simplifying database tables and reducing data redundancy** by separating columns into tables which can then be joined in many different ways
- The below table can **only be used for its specific purpose**, if we were to normalize it, we could use the data across many different tables
- There are three basic forms of normalization

StudentID	Name	Courses
1	John Doe	Math, Science
2	Jane Smith	Literature, Math, Art

Data Normalization – 1NF

Read: **First Normal Form**

Requirements:

- Eliminate repeating groups in individual tables
- Create a separate table for each set of related data
- Identify each set of related data with a primary key

Student #	Advisor	Advisor Room	Class 1	Class 2	Class 3
1022	Jones	412	101-07	143-01	159-02
4123	Smith	216	101-07	143-01	179-04

→

Student #	Advisor	Advisor Room	Class #
1022	Jones	412	101-07
1022	Jones	412	143-01
1022	Jones	412	159-02
4123	Smith	216	101-07
4123	Smith	216	143-01
4123	Smith	216	179-04

Source

Data Normalization – 2NF

Read: **Second Normal Form**

Requirements:

- Is at least 1NF
- Create separate tables for sets of values that apply to multiple records.
- Relate these tables with a foreign key.

Student #	Advisor	Advisor Room	Class #
1022	Jones	412	101-07
1022	Jones	412	143-01
1022	Jones	412	159-02
4123	Smith	216	101-07
4123	Smith	216	143-01
4123	Smith	216	179-04

→

Student #	Class #
1022	101-07
1022	143-01
1022	159-02
4123	101-07
4123	143-01
4123	179-04

Student #	Advisor	Advisor Room
1022	Jones	412
4123	Smith	216

Source

Data Normalization – 2NF

Read: **Second Normal Form**

Requirements:

- Is at least 2NF
- Eliminate data not dependent on key

Student #	Advisor	Advisor Room
1022	Jones	412
4123	Smith	216

→

Student #	Advisor
1022	Jones
4123	Smith

Name	Room	Department
Jones	412	42
Smith	216	42

Source

CONCEPTUAL, LOGICAL, AND PHYSICAL DATA MODELS

Conceptual, Logical, and Physical Data Models

Conceptual Data Model

- Designed for business, it is supposed to **map out the flow of the business and how that will eventually tie into the different entities** we'll create in the database.

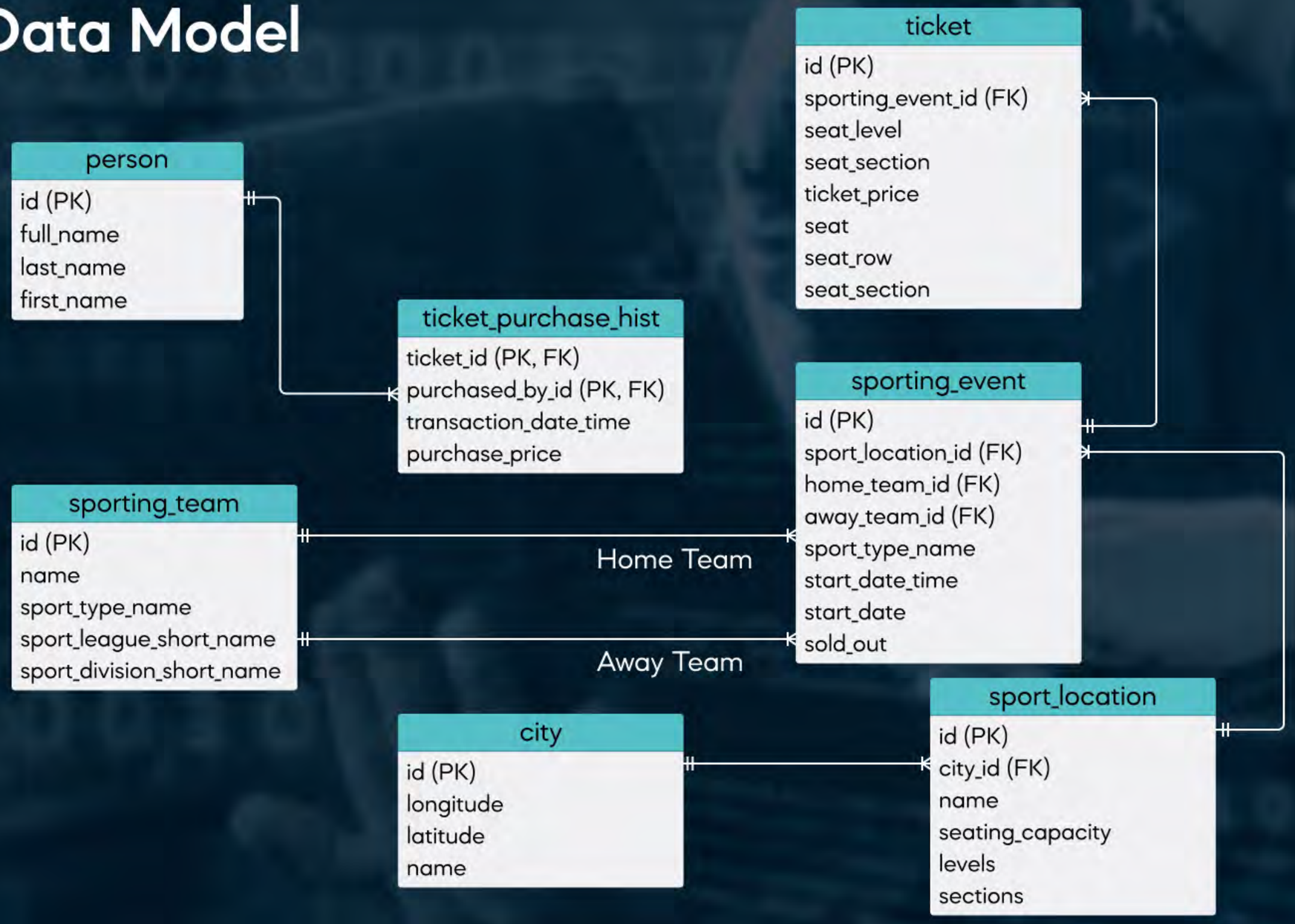
Logical Data Model

- **Maps out** the entities, what they contain, and how they relate to one another
- Designed for **Data Architects and Analysts**, it can use business friendly terms to define entities

Physical Data Model

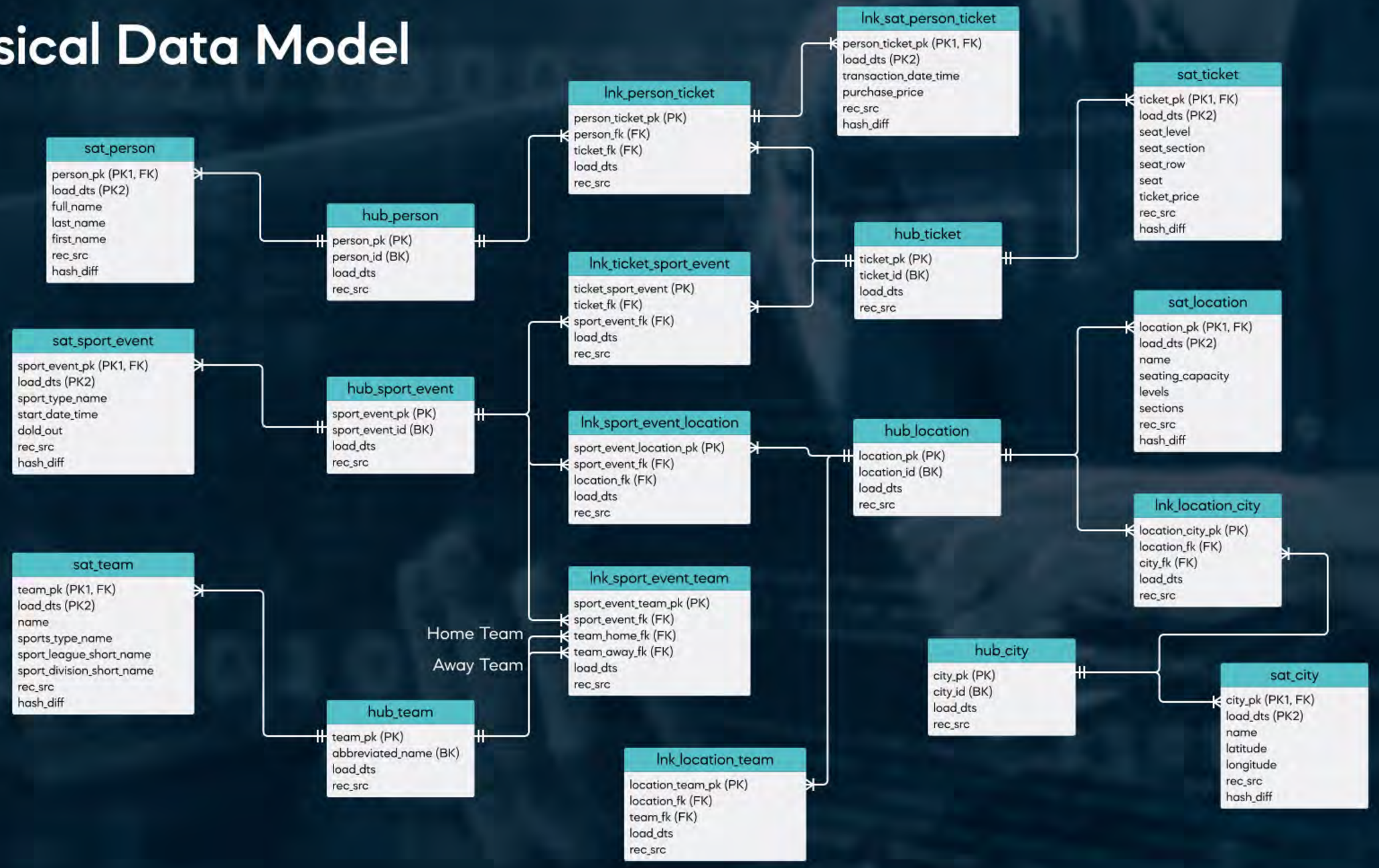
- **Contains data types and system-compliant names** that allow Database Administrators and Engineers to actually implement a Data Model

Logical Data Model



Source

Physical Data Model



Source

ENTITY RELATIONSHIP DIAGRAMS

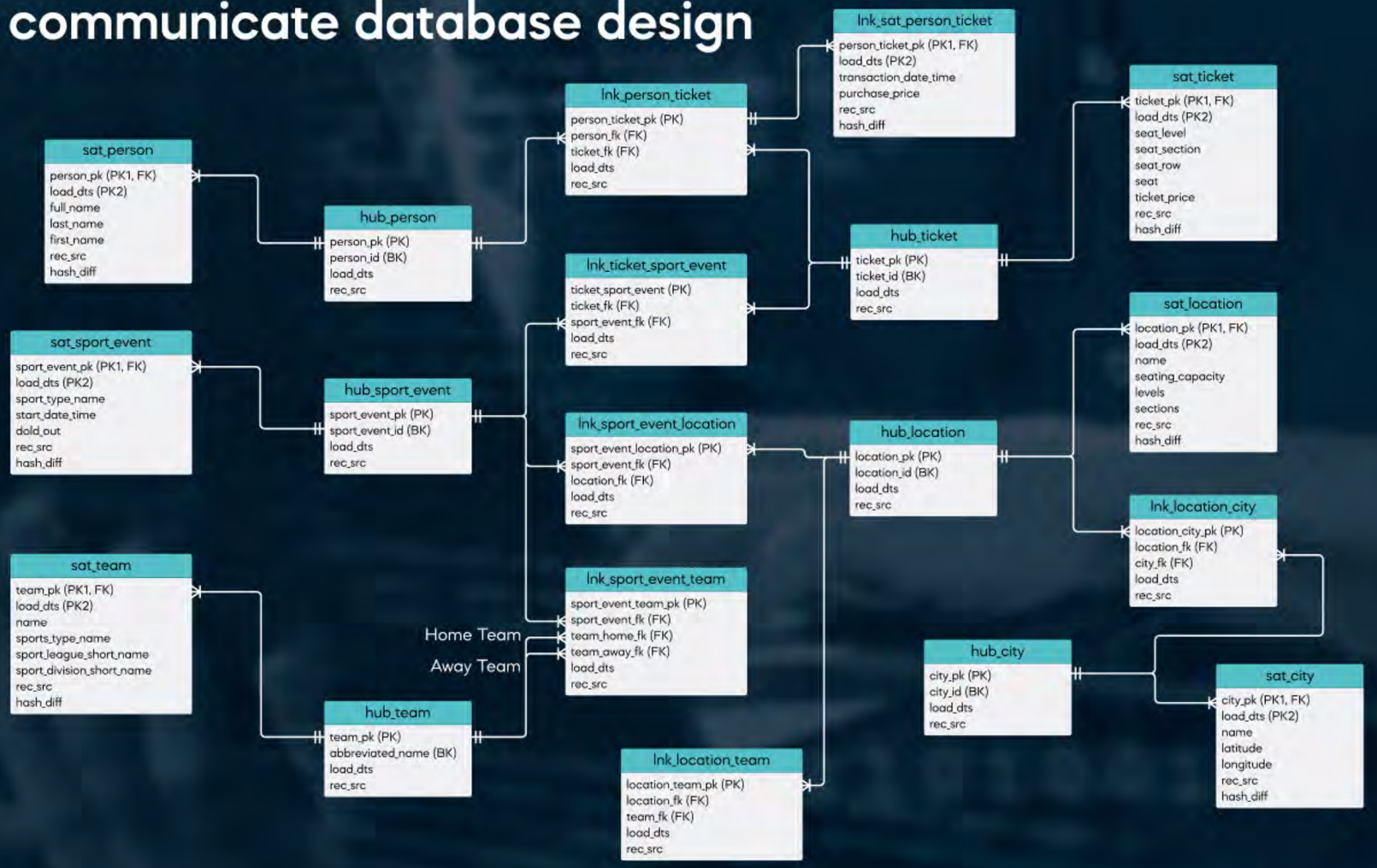
Physical Data Model

Universal language to communicate database design

one-to-many

one-to-one

many-to-many



SECURITY AND PIRACY

365

THE NON-OPTIONALITY OF SECURITY AND PRIVACY

The Non-Optionality of Security and Privacy

The Aadhaar Card: Cybersecurity Issues with India's Biometric Experiment

- https://jsis.washington.edu/news/the-aadhaar-card-cybersecurity-issues-with-indias-biometric-experiment/

Tesla Employee information leak

- https://www.theverge.com/2023/8/21/23839940/tesla-data-leak-inside-job-handelsblatt

Things to consider

Multiple legal frameworks

- California Consumer Privacy Act **(CCPA)**
- General Data Protection Regulation **(GDPR)**

Right to be Forgotten

- Customers can request that their data be dissociated from identification

PII (Personal Identifiable Information) **needs to be protected**

PERSONALLY IDENTIFIABLE INFORMATION

What is PII?

Direct PII

- Name
- National Identity Number
- Driver's License Number

Indirect PII

- Date of birth
- Place of birth
- Race
- Employment Information

What is PII?

- **Data minimization:** Collect only what is necessary.
- **Data security:** Implement strong security measures (encryption, access controls).
- **Transparency:** Clearly communicate with individuals about how their PII is being used.
- **Training:** Educate employees on the importance of protecting PII and how to do so.

PRINCIPLE OF LEAST PRIVILEGE

What is the Principle of Least Privilege?

Definition:

The Principle of Least Privilege requires that individuals and systems are granted the minimum levels of access – or permissions – needed to perform their duties.

Objective:

To minimize the attack surface by limiting access rights for users to the bare minimum necessary to perform their functions.

Implementing Principle of Least Privilege

- Conduct a **Privilege Audit:** Review current access levels across your systems to identify unnecessary permissions.
- Adopt **Role-Based Access Control (RBAC):** Assign permissions based on roles rather than individuals to streamline access management.
- Implement **Just-In-Time (JIT) Access:** Grant temporary privileges as needed for specific tasks, reducing the window of opportunity for abuse.
- Use **Privileged Access Management (PAM) Tools:** Automate the enforcement of PoLP to ensure consistent application across the organization.

BIG DATA & ENGINEERING

WHAT IS BIG DATA?

What is Big Data?

There is no exact definition and it changes with time

Definition of Big Data: Vast volumes of data that cannot be processed effectively with traditional applications. Characterized by the 3 Vs: Volume, Velocity, and Variety.

Role of a Data Engineer: Develop, construct, test, and maintain architectures, such as databases and large-scale processing systems.

Importance of Big Data Processing: Enables organizations to analyze and extract valuable insights from large datasets to inform decision-making and strategy.

Dealing with Big Data

Leveraging Big Data Frameworks: Examples include Apache Hadoop for distributed storage and processing, Apache Spark for in-memory processing, and Apache Kafka for real-time data streaming.

Data Lakes for Storage: Implement data lakes to store raw data in its native format, allowing for flexible processing and analysis.

Scalable and Elastic Solutions: Use cloud platforms (AWS, Google Cloud, Azure) for scalable, cost-effective storage and computing resources.

Continuous Learning and Adaptation: Stay updated with the latest technologies and methodologies in big data processing to address evolving challenges.

HADOOP

What is Hadoop?

Distributed Data Storage: Hadoop's HDFS allows for storing vast amounts of data across many machines, making it ideal for companies with large datasets that need long-term storage.

Batch Processing: Perfect for processing large data batches, such as historical data analysis and processing logs or social media data to extract trends, patterns, and user behavior insights.

Data Lakes: Companies use Hadoop to build data lakes where raw data is stored in its native format until needed, supporting flexible data exploration, discovery, and analysis across various business units.

Scalable Analytics: Hadoop scales from a single server to thousands, allowing businesses to grow their data analysis capabilities as they accumulate more data.

SPARK

What is Apache Spark?

Stream Processing: Spark Streaming enables real-time analytics, making it ideal for monitoring and reacting to live data streams, such as social media feeds or sensor data in IoT applications.

Interactive Analysis: Spark's ability to process in-memory makes it excellent for interactive data exploration and iterative algorithm development, commonly used in scientific research and complex analytics.

Machine Learning: With MLlib, Spark supports scalable machine learning algorithms, allowing for advanced analytics on customer data, predictive modeling, and real-time recommendations in e-commerce.

Graph Processing: GraphX enables graph processing for social network analysis, identifying influential users, community detection, and relationship mapping, beneficial in marketing and social research.

KAFKA

What is Kafka? kafka

Event-Driven Architecture: Kafka is pivotal for applications with an event-driven architecture, enabling real-time messaging and stream processing, critical in finance for transaction monitoring or in retail for stock level alerts.

Log Aggregation: Collects logs from multiple services, making it easier for companies to process and analyze operational data for monitoring, security analysis, and compliance reporting.

Data Pipelines: Kafka acts as the backbone for data pipelines, efficiently moving data between systems and applications, enabling real-time data synchronization across databases, search indexes, and analytic systems.

Microservices Communication: Facilitates decoupled, asynchronous communication between microservices, improving system reliability and scalability, especially in complex, distributed applications like online marketplaces or streaming services.

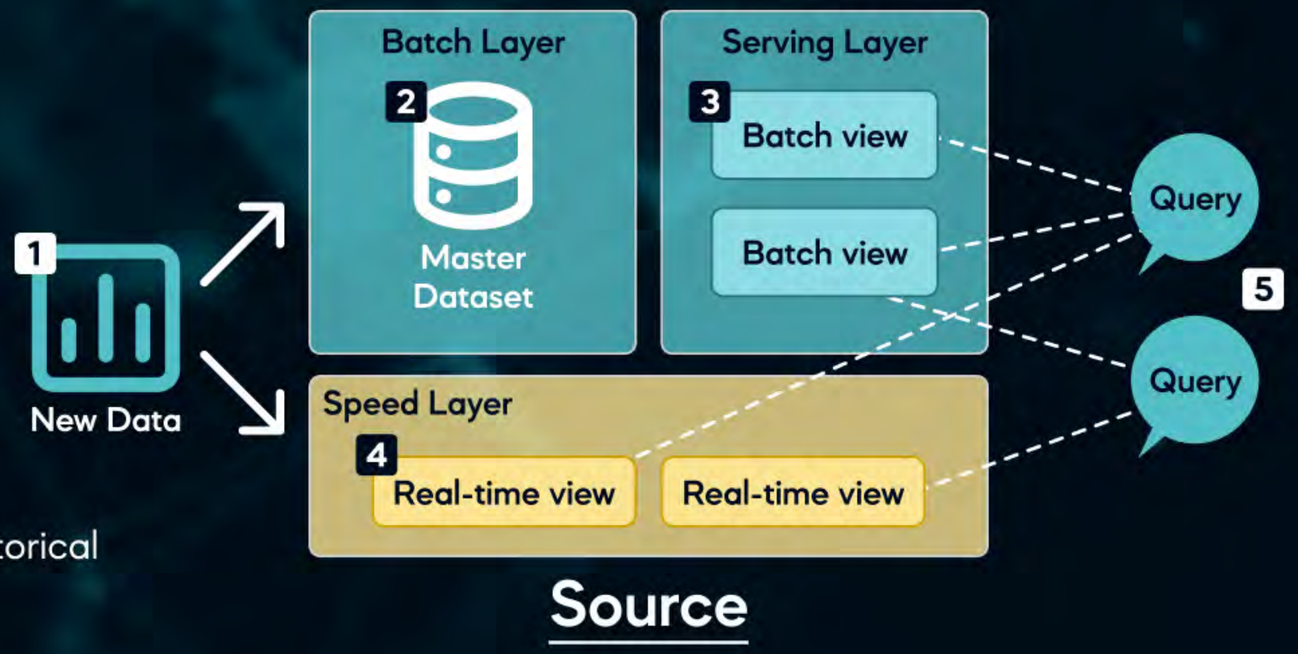
LAMBDA ARCHITECTURE

Lambda Architecture

Concept: A data-processing architecture designed to handle massive quantities of data by taking advantage of both batch and stream-processing methods.

Structure

- **Batch Layer:** Manages the master dataset and pre-computes the batch views.
- **Speed Layer:** Processes data in real-time to compensate for the latency of the batch layer.
- **Serving Layer:** Indexes and exposes the data views for query purposes.



Advantages

- Provides a comprehensive view of data, combining historical (batch) and real-time analysis.
- Ensures fault tolerance and scalability, leveraging the strengths of both processing methods.

Use Cases:

- Ideal for complex analytical systems where both historical and real-time data analysis are crucial, such as in financial risk analysis or targeted advertising.

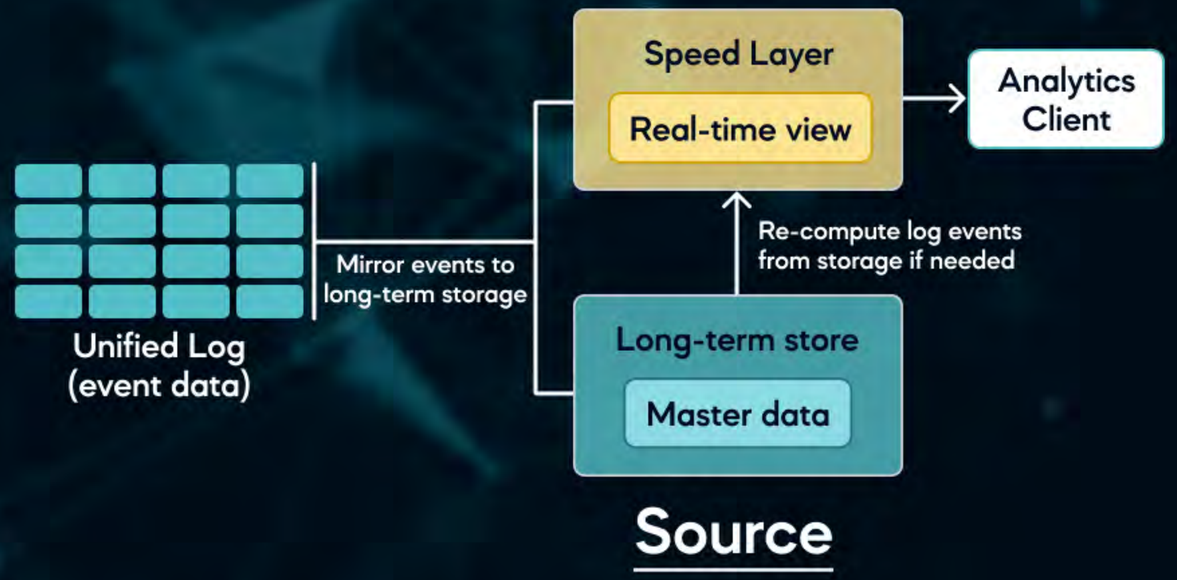
KAPPA ARCHITECTURE

Lambda Architecture

Concept: A streamlined architecture that uses a single stream processing layer to handle both real-time and historical data processing, simplifying the Lambda architecture.

Structure

- **Stream Processing Layer:** The core of Kappa Architecture, where incoming data is processed in real-time.
- **Serving Layer:** Similar to Lambda, it makes processed data available for querying and analysis



Advantages

- Simplifies the data architecture by eliminating the need for separate batch and speed layers.
- Easier to maintain and manage, with potentially lower operational complexity and cost.

Use Cases:

- Suitable for applications where data freshness is paramount and the distinction between historical and real-time data is minimal, such as in real-time monitoring systems or live dashboards.

CONNECTING TO DATA

Where does data live?

Emails

Databases

People's Desktops

External Services

How should we obtain that data?

Standardized

Consistent

Fail loudly

Stop other processes if failure occurs

Places we can receive data into

Databases

Cloud Storage Buckets

SFTP Servers

Some Considerations

- Be wary of **one-size-fits all** solutions
- Need to create a balance between **standardization and customization**
- Code generally allows for the **flexibility of customization** without an insurmountable barrier to understanding
  - **Open-source common frameworks**

Some Recommendations

- Most data → Cloud bucket
- Use **external APIs** when you can
- SFTPs as a **last resort** (some cloud services allow a bucket to act as an SFTP server)
- Graduate to more advanced forms of data ingestion like kafka
  - **Can be hard to manage and troubleshoot vs batch data**

INMON
DATA WAREHOUSE

Inmon Data Warehouse

Father of the Data Warehouse

Subject-oriented

- Finance, Supply Chain

Integrated

- Consolidated and normalized (3NF)

Nonvolatile

- Data remains unchanged

Time-variant

- Varying time ranges can be queried

KIMBALL
DATA WAREHOUSE

Kimball Data Warehouse

More denormalized

Uses Fact and Dimension tables

Fact Table

- Quantitative or event related

Dimension Table

- Provides reference data

Kimball Data Warehouse

Fact Table

FactSalesID	DateKey	ProductKey	CustomerKey	StoreKey	SalesAmount	QuantitySold
1	20200101	101	501	401	120.00	2
2	20200101	102	502	401	200.00	4
3	20200102	103	503	402	150.00	3
4	20200102	104	504	402	60.00	1

Dimension Table

ProductKey	ProductName	Category	Price	Supplier
101	Widget A	Electronics	60.00	Acme Electronics
102	Gadget B	Home Goods	50.00	Global Home Goods
103	Tool C	Hardware	50.00	Hardware Co.